MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA
IP CENTRUL DE EXCELENȚĂ ÎN INFORMATICĂ ȘI TEHNOLOGII INFORMAȚIONALE

CATEDRA INFORMATICA II

ANDRIAN DASCAL **VASILII DERMENJI**

UTILIZAREA TEHNICILOR CLASICE DE PROGRAMARE ÎN C++

Suport de curs
destinat cadrelor didactice și elevilor din IP CEITI

CHIȘINĂU, 2023

Elaborat conform Curriculumului modular „Utilizarea tehnicilor clasice de programare”, ediția 2020. Lucrarea a fost discutată și examinată la ședința catedrei „Informatica II” din 26.04.2023, Proces Verbal Nr. 9, șef catedră, Luminița Gribineț

Autori: Vasilii Dermenji, prof. la discipline de informatică, grad didactic I, IP CEITI
Andrian Dascal, prof. la discipline de informatică, grad didactic II, IP CEITI

Recenzie: Ioan Jeleascov, magistru în Informatică și Tehnologii Informaționale, prof. la discipline de informatică la IP CEITI, mun. Chișinău

Redactare științifică: Ioan Jeleascov, „Architect-Advisor” în cadrul "Stone-Hard Unity", Londra, Marea Britanie

„Programatorul este precum un copil ce încearcă necunoscutul doar pentru a atinge absolutul . ” - ***V. S. Sărmășanu***

DRAGI PRIETENI,

Algoritmii de sortare sunt esențiali în lumea calculatoarelor și programării, deoarece sunt utilizați pentru a organiza datele într-un mod specific și eficient. Sortarea este o sarcină comună în multe domenii, cum ar fi bazele de date, analiza datelor, grafica computerizată și multe altele.

Algoritmii de căutare sunt la fel de importanți ca și algoritmii de sortare în programare. Căutarea este o problemă frecventă în multe domenii, inclusiv bazele de date, algoritmii de procesare a imaginilor, aplicațiile de căutare web și multe altele.

Algoritmii de aproximare și de optimizare în grafuri sunt esențiali în programare și teorie, deoarece permit găsirea soluțiilor apropiate sau optime pentru problemele complexe de grafuri. Algoritmii de aproximare și de optimizare sunt utilizați pentru a rezolva probleme dificile din domenii precum: rețele de comunicații, analiză socială, optimizare a resurselor și multe altele.

Algoritmii SIMPLEX și DUAL SIMPLEX sunt extrem de importanți în programarea liniară și sunt folosiți într-o gamă largă de aplicații, inclusiv în probleme de planificare, logistica, producție și multe altele. Algoritmul SIMPLEX este utilizat pentru a găsi soluții optime, ceea ce înseamnă că este utilizat pentru a maximiza sau a minimiza o funcție liniară sub un set de restricții liniare.

În general, scopul acestui suport de curs este de a oferi o bază solidă de cunoștințe și abilități de programare studenților sau programatorilor începători, care să poată fi utilizate într-o varietate de domenii și aplicații.

În plus, acest material poate fi util pentru programatorii care doresc să își consolideze cunoștințele existente și să învețe noi abordări de programare, dar și pentru cadrele didactice debutante în domeniu. Învățarea tehnicilor clasice de programare este importantă pentru a dezvolta abilități solide de programare și pentru a fi pregătit pentru a aborda problemele complexe de programare într-un mod eficient și structurat.

Autorii

CUPRINSUL

INTRODUCERE

Unitatea de curs „Utilizarea tehnicilor clasice de programare" conține noțiunile fundamentale privind structurile dinamice de date, algoritmi de sortare și de căutare, tehnici de programare necesare pentru elaborarea unui software performant şi portabil.

Această unitatea de curs face parte din categoria disciplinelor opţionale de specialitate pentru formarea specialiştilor în domeniul tehnologiilor informaţionale şi comunicaţiilor. Unitatea de curs este recomandată a fi studiată în anul III de studii şi reprezintă un îndrumar eficient de formare profesională. Modulul în cauză poate fi studiat după însuşirea în mod obligatoriu a următoarelor unităţi de curs: Programarea structurată, Programarea procedurală și Programarea calculatorului.

Studierea acestui modul va contribui la formarea şi dezvoltarea de competențe profesionale ce corespund nivelului patru de calificare:

- *cunoștințe faptice, principii, procese și concepte generale din domeniul elaborării produselor program;*
- *abilități cognitive și practice necesare pentru elaborarea aplicațiilor de consolă conform tematicilor incluse;*
- *asumarea responsabilității pentru mentenanţa de aplicaţii.*

În cadrul untății de curs vor fi formate şi dezvoltate următoarele competențe profesionale specifice:

CS1. *Prelucrarea tipurilor dinamice de date în cadrul aplicațiilor de consolă.*
CS2. *Utilizarea structurilor dinamice de date pentru problemele întâlnite în activitatea profesională.*
CS3. *Gestionarea eficientă a memoriei interne a calculatorului.*
CS4. *Utilizarea tehnicilor clasice de programare pentru problemele întâlnite în activitatea profesională.*
CS5. *Utilizarea grafurilor pentru problemele întâlnite în activitatea profesională.*
CS6. *Alegerea tehnicii clasice de programare adecvate problemei.*

Competenţele formate şi dezvoltate în cadrul acestui modul vor fi necesare pentru studierea unităţilor de curs orientate spre elaborarea/dezvoltarea produselor program.

Capitolul I

ALGORITMI DE SORTARE

De ce avem nevoie de algoritmi de sortare?

Sortarea este procesul de plasare a elementelor dintr-o colecție într-un fel de ordine, spre exemplu: ordinea alfabetică sau ordinea numerică, dar să nu uităm că fiecare ordine are unul din tipurile: ascendent sau descendent. De exemplu, o listă de nume de clienți ar putea fi sortată și în funcție de vârstă sau după data nașterii sau după un alt criteriu propus de utilizator. Sortarea unei liste de articole poate dura mult timp, mai ales dacă este o listă mare. Un program de calculator poate fi creat pentru a face acest lucru, facilitând sortarea unei liste de date. Există multe tipuri de algoritmi de sortare pe care dorim să-i studiem în acest compartiment. La fel ca și căutarea, eficiența unui algoritm de sortare este legată de numărul de elemente procesate.

Care sunt caracteristicile fundamentale ale algoritmilor de sortare?

Metodele de sortare se caracterizează prin:

- ***Stabilitate.*** *O metodă de sortare este considerată stabilă dacă ordinea relativă a elementelor ce au aceeași valoare a cheii nu se modifică în procesul de sortare.*
- ***Naturalețe.*** *O metodă de sortare este considerată naturală dacă numărul de operații scade odată cu distanța dintre array-ul inițial și cel sortat. O măsură a acestei distanțe poate fi numărul de inversiuni al permutării corespunzătoare array-uului inițial.*
- ***Eficiență.*** *O metodă este considerată eficientă dacă nu necesită un volum mare de resurse. Din punctul de vedere al spțiului de memorie, o metodă de sortare pe loc este mai eficientă decât una bazată pe o zonă de manevră de dimensiunea tabloului. Din punct de vedere al timpului de execuție este important să fie efectuate cât mai puține operații. În general, în analiză se iau în considerare doar operțiile efectuate asupra elementelor tabloului (comparații și mutări). O metodă este considerată optimală dacă ordinul său de complexitate este cel mai mic din clasa de metode din care face parte.*
- ***Simplitate.*** *O metodă este considerată simplă dacă este intuitivă și ușor de înțeles.*

1.1 Algoritmi de sortare eficienţi - Partea I

Obiective

- *Descrierea algoritmilor de sortare eficientă (ShellSort şi QuickSort).*
- *Prezentarea situațiilor de aplicare a algoritmilor de sortare eficientă.*
- *Elaborarea algoritmilor de sortare eficientă conform specificațiilor propuse.*
- *Implementarea algoritmilor de sortare eficientă în limbajul de programare.*

***ShellSort**, cunoscut și sub denumirea de metoda Shell, este un tip de comparație la fața locului. Poate fi văzut fie ca o generalizare a sortării prin schimb (sortare cu bule), fie sortare prin inserție. Metoda începe prin sortarea perechilor de elemente aflate la o distanță una de alta, apoi reducând progresiv decalajul dintre elementele care trebuie comparate. Începând cu elemente îndepărtate, poate muta unele elemente deplasate în poziție mai repede decât un simplu schimb de vecini.*

Donald Shell a publicat prima versiune de acest fel în 1959. Durata de funcționare a ShellSort depinde în mare măsură de secvența de goluri pe care o folosește. Pentru multe variante practice, determinarea complexității timpului lor rămâne o problemă deschisă. Shellsort este o optimizare a tipului de inserție care permite schimbul de elemente care sunt departe. Ideea este de a aranja lista de elemente astfel încât, începând de oriunde, luând fiecare element hth să producă o listă sortată. Se spune că o astfel de listă este sortată în h. Poate fi considerat, de asemenea, ca h liste intercalate, fiecare sortată individual. Începând cu valori mari ale h, elementele se pot deplasa pe distanțe mari în lista originală, reducând rapid cantități mari de tulburări și lăsând mai puțin de lucru pentru pașii mai mici de sortare h. Dacă lista este apoi sortată k pentru un număr întreg mai mic k, atunci lista rămâne sortată h. Urmarea acestei idei pentru o secvență descrescătoare a valorilor h care se termină cu 1 este garantată pentru a lăsa o listă sortată la final.

*Acest algoritm folosește sortarea inserției pe elemente răspândite pe scară largă, mai întâi pentru a le sorta și apoi sortează elementele mai puțin spațiate. Această distanță este denumită interval. Acest interval este calculat pe baza formulei lui Knuth: h=(h*3)+1, unde h este interval cu valoarea inițială 1.*

Exemplu pas cu pas:

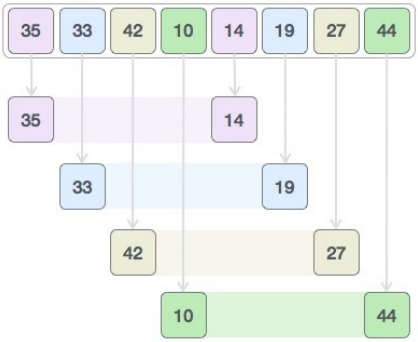
i	0	1	2	3	4	5	6	7
X[i]	14	19	27	10	35	33	42	44

Urmează algoritmul pentru ShellSort:

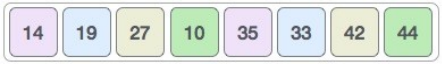
1. *Inițializați valoarea h;*
2. *Împărțiți lista într-o sub-listă mai mică cu interval egal h;*
3. *Sortați aceste sub-liste folosind sortarea prin inserare;*
4. *Repetați până când lista completă este sortată.*

Soluția ar fi următoarea:

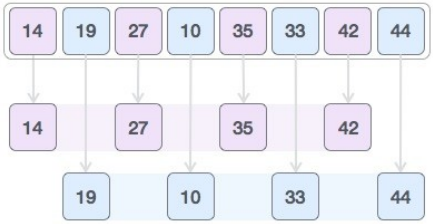
Luăm același array pe care l-am folosit anterior. Pentru exemplul nostru și ușurința de înțelegere, luăm intervalul de 4. Facem o sub-listă virtuală a tuturor valorilor situate la interval de 4 poziții. Aici aceste valori sunt {35, 14}, {33, 19}, {42, 27} și {10, 44}.



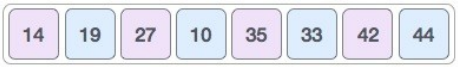
Comparăm valorile din fiecare sub-listă și le schimbăm (dacă este necesar) în array-ul original. După acest pas, noul array ar trebui să arate astfel:



Apoi, luăm intervalul 1 și acest decalaj generează două sub-liste - {14, 27, 35, 42}, {19, 10, 33, 44}:



Comparăm și schimbăm valorile, dacă este necesar, în array-ul original. După acest pas, array-ul ar trebui să arate astfel:



În cele din urmă, sortăm restul array-ului folosind intervalul de valoare 1. Shell sort folosește sortarea prin inserție pentru a sorta array-ul.

Urmează descrierea pas cu pas:

Partea I	Partea II
14 19 27 10 35 33 42 44	14 10 19 27 35 33 42 44
14 19 27 10 35 33 42 44	10 14 19 27 35 33 42 44
14 19 27 10 35 33 42 44	10 14 19 27 35 33 42 44
14 19 27 10 35 33 42 44	10 14 19 27 33 35 42 44
14 19 10 27 35 33 42 44	10 14 19 27 33 35 42 44

Observăm că a necesitat doar patru swap-uri (interschimbări) pentru a sorta restul array-ului.

Quicksort *este un celebru algoritm de sortare, în medie, efectuează O (n log n) comparații pentru a sorta n elemente. În cazul cel mai defavorabil, efectuează $O(n^2)$ comparații. De obicei, în practică, QuickSort este mai rapid decât ceilalți algoritmi de sortare de complexitate O (n log n), deoarece bucla sa interioară are implementări eficiente pe majoritatea arhitecturilor, în plus, în majoritatea implementărilor practice se pot lua, la proiectare, decizii ce ajută la evitarea cazului când complexitatea algoritmului este de $O (n^2)$.*

Algoritmul a fost dezvoltat de C. A. R. Hoare în 1960, pe când lucra la mica firmă britanică producătoare de calculatoare Elliot Brothers. Quicksort este un algoritm eficient de sortare. Dezvoltat de informaticianul britanic Tony Hoare în 1959 și publicat în 1961, este încă un algoritm utilizat în mod obișnuit pentru sortare. Atunci când este implementat bine, poate fi de aproximativ două sau trei ori mai rapid decât principalii săi concurenți, MergeSort și HeapSort.

Quicksort este un algoritm de divizare și cucerire. Funcționează selectând un element „pivot" din array-ul unidimensional și partiționând celelalte elemente în două sub-array-uri unidimensionale, în funcție de faptul dacă acestea sunt mai mici sau mai mari decât pivotul. Sub-array-uri unidimensionale sunt apoi sortate recursiv. Acest lucru se poate face în loc, necesitând mici cantități suplimentare de memorie pentru a efectua sortarea.

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7	8	9
X[i]	7	4	2	8	5	9	3	10	1	6

Urmează algoritmul pentru QuickSort:

1. *Alegeți un element, numit pivot, din masiv.*
2. *Partiționare: reordonați masivul astfel încât toate elementele cu valori mai mici decât pivotul să vină înainte de pivot, în timp ce toate elementele cu valori mai mari decât pivotul vin după acesta (valori egale pot merge în ambele sensuri). După această partiționare, pivotul se află în poziția sa finală. Aceasta se numește operație de partiție.*
3. *Aplicați recursiv pașii de mai sus sub-masivului de elemente cu valori mai mici și separat sub-masivului de elemente cu valori mai mari.*

Soluția ar fi următoarea:

Să presupunem că elementul pivot la noi va fi primul element din array. Inițial se compară elementul pivot cu X[i], se interschimbă în cazul nostru:

7	4	2	8	5	9	3	10	1	6

Avansez cu primul: (4<7. Nu se fac interschimbări)

6	4	2	8	5	9	3	10	1	7

Se avansează la 2. : (2<7. Nu se fac interschimbări)

6	4	2	8	5	9	3	10	1	7

Se avanseaza cu primul: (8>7. Se interschimbă)

6	4	2	8	5	9	3	10	1	7

și se devansează în partea dreaptă, 7>1. Se interschimbă.

6	4	2	7	5	9	3	10	1	8

Se avansează în stânga, 5<7. Nu se interchimbă.

6	4	2	1	5	9	3	10	7	8

Se avansează la 9, 9>7. Se interschimbă.

6	4	2	1	5	9	3	10	7	8

Se devansează din dreapta, 7 și 10 sunt în ordine.

6	4	2	1	5	7	3	10	9	8

Se trece la 3, care se schimbă cu 7.

6	4	2	1	5	7	3	10	9	8

u și p ajung la aceeasi valoare, caz în care se încheie secvanța.

6	4	2	1	5	3	7	10	9	8

p=u, STOP! k=p

În continuare se aplică aceeaşi secvență pe porțiunile din stânga, respectiv dreapta pivotului:

6	4	2	1	5	3

şi

10	9	8

Pentru a stabili complexitatea algoritmului QuickSort aplicată unui array cu n componente, notăm cu C(n) numărul de comparaţii efectuate şi remarcăm că, la fiecare pas al algoritmului au loc n comparaţii însoţite de interschimbări ale elementelor şi două invocari recursive ale metodei de sortare, deci: C(n)= n+C(k)+C(n-k), unde k este numărul de componente din zona din stânga a array-ului, iar C(k) şi C(n-k) sunt complexităţile pentru cele două subzone care urmează a fi sortate recursiv.

Cazul cel mai favorabil ar fi când, la fiecare recursie, cele două subzone (cu elemente mai mici şi respectiv mai mari decat elementul pivot) ar fi egale. În acest caz, calculul complexităţii se face la fel ca la MergeSort şi deci complexitatea este O(n.log(n)).

*Cazul cel mai defavorabil ar fi cel în care, la fiecare recursie, elementul pivot ar fi ales în mod atat de nefericit, încat una din cele două subzone obţinute după interschimbări ar fi vida. In acest caz C(n) = n + C(n-1) = n + (n-1) + C(n-2) = ... sau, continuand pana la C(1), C(n) = n + (n-1) + (n-2) + ... + 1 = (n+1)*n/2 şi deci complexitatea algoritmului este O(n2). Acest caz "nefericit" este însă foarte puţin probabil. Cel mai probabil este că, în realitate, complexitatea sa fie situată în jurul valorii O(n log(n)).*

Cazul de bază al recursiunii este masivul de dimensiunea zero sau unu, care sunt în ordine prin definiție, astfel încât nu trebuie niciodată sortate. Etapele de selecție pivot și partiționare pot fi realizate în mai multe moduri diferite; alegerea schemelor de implementare specifice afectează foarte mult performanța algoritmului.

Implementarea recursivă a algoritmilor în limbajul C++

```
#include <iostream>
using namespace std;
// Sortare ascendentă
```

```
int ShellSortAscendent(int a[], int n){
    for (int dec=n/2; dec>0; dec/=2){ //dec =  decajal
        for (int i=dec;i<n;i+=1){
            int j,temp=a[i]; // temporar
            for (j=i;j>=dec && a[j-dec]>temp; j-=dec)
                a[j] = a[j - dec];
                a[j] = temp;
        }
    }
}
// Funcția care schimbă două elemente
void schimb(int* a,int* b){
    int t=*a; *a=*b; *b=t;
}
// Partiția tabloului folosind ultimul element ca pivot
int partitionare (int X[], char start, char finish){
    int pivot = X[finish];    // pivot
    int i=(start-1);
    for (int j=start; j<=finish-1; j++) {
//Dacă elementul curent este mai mare decât pivotul, creștem
elementul start. Schimbăm elementul de la poziția i la j.
        if (X[j]>=pivot) {
            i++; schimb(&X[i],&X[j]);
        }
    }
    schimb(&X[i+1],&X[finish]);
    return (i+1);
}
//Algoritmul quickSort descendent
void QuickSortDescendent(int X[], int start, int finish){
    if (start<finish) {
        //Partișionarea masivului
        int pivot=partitionare(X,start,finish);
        //Sortăm masivul secundar independent
        QuickSortDescendent(X,start,pivot-1);
        QuickSortDescendent(X,pivot+1,finish);
    }
}
// Afișarea la ecran a array-ului unidimensional
void AfisareArray(int a[], int n){
    for (int i=0;i<n;i++)
        cout<<a[i]<<" ";
```

```
}
//Programul principal
int main(){
    // Vectorul de numere întregi declarat implicit
    int a[]={45, 23, 53, 43, 18, 24, 8, 95, 101}, i;
    int n=sizeof(a)/sizeof(a[0]);
    cout<<"Array-ul inainte de sortare: \n"; AfisareArray(a,n);
    // Aplicăm functia pentru sortarea ascendentă
    ShellSortAscendent(a,n);
    cout<<"\n\nShellSort ascendent:\nArray-ul sortat: ";
    AfisareArray(a, n);cout<<endl;
    // Aplicăm functia pentru sortarea deascendentă
    QuickSortDescendent(a,0,n-1);
    cout<<"\nQuickSort descendent:\nArray-ul sortat: ";
    AfisareArray(a,n);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Array-ul inainte de sortare:
45 23 53 43 18 24 8 95 101

ShellSort ascendent:
Array-ul sortat: 8 18 23 24 43 45 53 95 101

QuickSort descendent:
Array-ul sortat: 101 95 53 45 43 24 23 18 8
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi litere majuscule. Aplicați ambii algoritmi de sortare în același program.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi cuvinte citite dintr-un fișier extern separate printr-un spațiu. Aplicați ambii algoritmi de sortare în același program.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei cozi. Aplicați ambii algoritmi de sortare.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei stive. Aplicați ambii algoritmi de sortare.

1.2 Algoritmi de sortare eficienţi - Partea II

Obiective

- *Descrierea algoritmilor de sortare eficientă (MergeSort și HeapSort).*
- *Prezentarea situațiilor de aplicare a algoritmilor de sortare eficientă.*
- *Elaborarea algoritmilor de sortare eficientă conform specificațiilor propuse.*
- *Implementarea algoritmilor de sortare eficientă în limbajul de programare.*

Algoritmii de sortare prin inserție și prin selecție necesită timp pătratic, atât în cazul mediu, cât și în cazul nefavorabil. Cu toate ca acești algoritmi sunt excelenți pentru cazuri mici (elemente puține), pentru cazuri mari (foarte multe elemente) avem algoritmi mai eficienti. Câțiva algoritmi de sortare sunt: merge sort și QuickSort.

În informatică, sortarea prin interclasare este un algoritm de sortare eficient, cu scop general, bazat pe comparație. Majoritatea implementărilor produc un sortare stabilă, ceea ce înseamnă că ordinea elementelor egale este aceeași în intrare și ieșire. ***MergeSort*** *este inventată de John von Neumann în 1945. În cazul sortării prin interclasare vectorii care se interclasează sunt două secvențe ordonate din același vector. Sortarea prin interclasare utilizeaza metoda Divide et Impera:*

- *se împarte vectorul în secvențe din ce în ce mai mici, astfel încât fiecare secvență să fie ordonată la un moment dat și interclasată cu o altă secvență din vector corespunzătoare.*
- *practic interclasarea va începe când se ajunge la o secvență formată din două elemente. Aceasta odată ordonată se va interclasa cu o alta corespunzătoare. Cele doua secvențe vor alcătui în subșir ordonat din vector mai mare, care la rândul lui se va interclasa cu subșirul corespunzător ș.a.m.d.*

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7
X[i]	8	7	9	3	6	4	17	16

Urmează algoritmul pentru MergeSort:

1. *Dacă lista este de lungime 0 sau 1, atunci este deja sortată. Altfel:*
2. *Împarte lista nesortată în două subliste aproximativ egale.*
3. *Sortează fiecare sublistă recursiv prin reaplicarea algoritmului merge sort.*
4. *Se interclasează cele două liste și se obține lista inițială sortată.*

Soluția ar fi următoarea:

1. *Se împarte în două secvenţe: (8 7, 9, 3) şi (6, 4, 17, 16). În continuare pentru prima secvenţă se procedează la fel: (8, 7) şi (9, 3). După o nouă împărţire se obţine: (8), (7).*
2. *Se incepe interclasarea. Se consideră că avem doi vectori de lungime 1 care se interclasează: (8),(7). Rezultă: (7,8). La fel şi pentru secvenţa: (9,3). Se consideră că avem iar doi vectori de lungime 1 care se interclasează: (9),(3). Rezultă: (3, 9). Ceea ce înseamnă că cele doua secvenţe determină obţinerea următorului subşir din vector: (7, 8, 3, 9). Pentru care se interclasează cele doua zone. Rezultă:(3, 7, 8, 9)*
3. *La fel se procedează şi cu secvenţa: (6, 4, 17, 16). Se obţine: (6),(4). Care se interclasează şi se obţine:(4, 6). La fel pentru: (17),(16). Se obţine: (16, 17). Se obţine o nouă secvenţă: (4, 6, 16, 17). Care prin interclasarea celor două zone conduce la: (4, 6, 16, 17).*
4. *Cele două secvenţe iniţiale din vector au devenit: (3, 7, 8, 9, 4, 6, 16, 17). Care se interclasează obţinând: (3, 4, 6, 7, 8, 9, 16, 17).*

Pentru a stabili complexitatea algoritmului de sortare prin interclasare, remărcam urmatoarele:

- *pentru un vector cu n elemente, numărul de înjumătăţiri succesive până se ajunge la subvectori de lungime 1 sau 2 este de aproximativ log2n şi nu depinde de valorile din vector;*
- *la fiecare din aceste divizări succesive, se mută din x într-un vector auxiliar pentru interclasare şi invers în total n elemente.*

În consecinţă, numarul de operaţii elementare executate este de ordinul $O(n \cdot \log_2 n)$. Constatăm deci că, pentru tablouri cu număr mare de componente, timpul de calcul este mult mai mic în cazul sortării prin interclasare, decât în cazul folosirii algoritmilor simpli, cum ar fi cel al selecţiei, inserţiei sau al bulelor, a căror complexitate este $O(n^2)$. Algoritmul de sortare prin interclasare consumă însă de două ori mai multă memorie decât cei simpli mentionaţi, deoarece necesită spaţiu suplimentar pentru vectorul auxiliar.

*În informatică, **HeapSort** este un algoritm de sortare bazat pe comparație. HeapSort poate fi considerat un algoritm de sortare prin selecție îmbunătățit: la fel ca sortarea de selecție, heapsort își împarte intrarea într-o regiune sortată și o regiune nesortată și micșorează iterativ regiunea nesortată extragând cel mai mare element din acesta și inserându-l în regiunea sortată.*

Spre deosebire de algoritmul de sortare prin selecție, heapsort nu pierde timpul cu o scanare liniară a regiunii nesortate; mai degrabă, sortarea heap menține regiunea nesortată într-o structură de date heap pentru a găsi mai rapid cel mai mare element din fiecare pas. Deși oarecum mai lent în practică pe majoritatea sistemelor de calcul decât un algoritm quicksort bine implementat, are avantajul unui timp de execuție O (n log n) mai favorabil. Heapsort este un algoritm în loc, dar nu este un tip stabil.

Heapsort a fost inventat de J. W. J. Williams în 1964. Aceasta a fost și nașterea grămezii, prezentată deja de Williams ca o structură de date utilă în sine. În același an, R. W. Floyd a publicat o versiune îmbunătățită care ar putea sorta o matrice în loc, continuând cercetările sale anterioare asupra algoritmului arbore. Metoda de sortare prin selecţie directă se bazează pe selecţia repetată a ultimei chei dintre n elemente, apoi dintre n-1 elemente rămase, etc. Pentru a găsi cea mai mică cheie dintre n elemente sunt necesare n-1 comparaţii, apoi găsirea următoarei dintre n-1 elemente are nevoie de n-2 comparaţii, etc. ⟹ n(n-1)/2 comparaţii.

Această sortare se poate îmbunătăţi prin reţinerea, de la fiecare scanare, de mai multă informaţie decât identificarea unui singur element, cel mai mic. De exemplu, cu n/2 comparaţii se poate determina cheia mai mică pentru fiecare pereche de elemente dintre cele n elemente, apoi cu alte n/4 comparaţii se poate determina cheia cea mai mică pentru fiecare pereche ale cheilor determinate anterior, şi aşa mai departe. Astfel, cu n-1 comparaţii se poate construi arborele de selecţie.

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7	8	9
X[i]	21	54	37	18	88	12	57	73	14	65

Urmează algoritmul pentru HeapSort:

1. *presupunem că vectorul formează un arbore binar, fiecare poziție din vector reprezentând un nod, cu rădăcina pe poziţia 0(zero) şi cu fiecare nod k*

având copiii 2k+1 şi 2k+2(dacă nu există poziţia din vector cu indicele respectiv, atunci nu există nod copil ⟹ NULL)

2. *formăm un max-heap cu aceeaşi reprezentare(pe vector, fără a construi altă structură pentru noduri)*
3. *extragem maximul din rădăcina heap-ului (poziţia 0 din vector) şi facem o intersschimbare între poziţia maximului şi ultima poziţie din vector. Acum maximul se află pe poziţia dorită şi putem să-l excludem din heap.*
4. *repetăm paşii(refacem forma de heap, extragem noul maxim, reducem cu 1 numărul de elemente nesortate), cât timp mai sunt elemente în heap.*

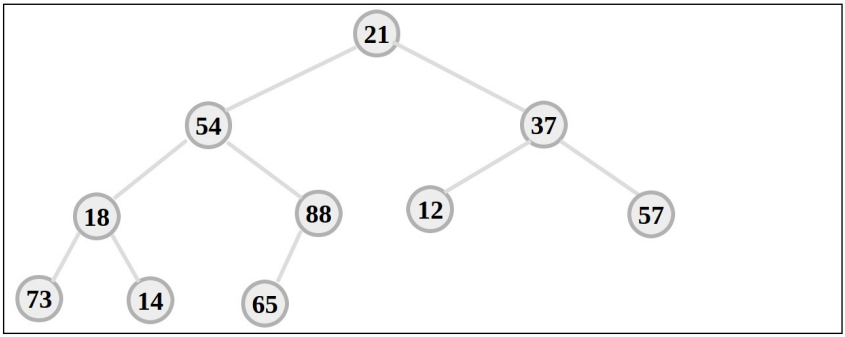
Definiţie:

Un MinHeap este un arbore binar complet cu proprietăţile:

1. *informaţiile din noduri sunt valori dintr-o mulţime total ordonată, numite chei;*
2. *pentru orice nod x[i], cheia memorată în x[i] este mai mică sau egală cu cheia oricăruia dintre fii;*
3. *oricare ar fi i din mulţimea de indici{1, ... , [n/2]}, atunci x[i]<=x[2*i] şi x[i]<=x[2*i+1], dacă 2*i+1<=n.*

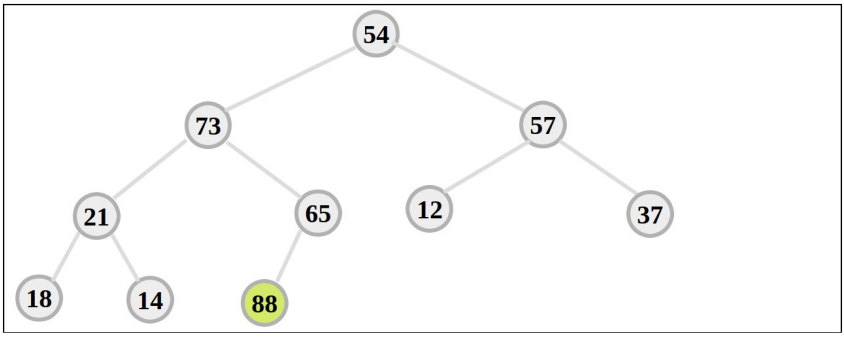
În cazul arborelui reprezentat pentru un vector de tip MinHeap, orice nod subordonează noduri cu etichete mai mari, iar în cazul vectorului de tip MaxHeap, orice nod subordonează noduri cu etichete mai mici.

Soluția ar fi următoarea:

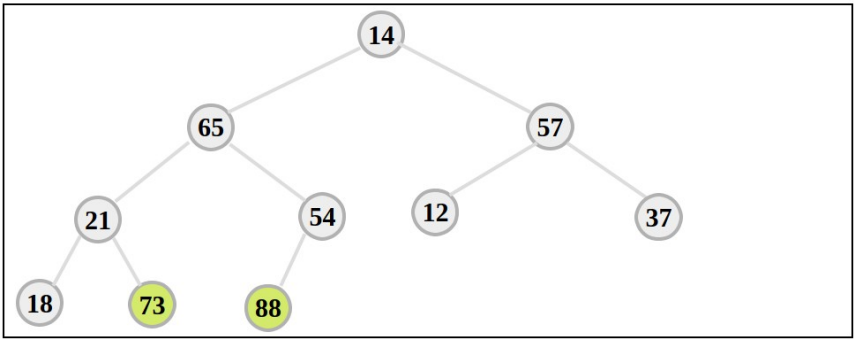


Reprezentarea arborescentă a datelor inițiale este prezentată în imaginea de mai sus. Analizați atent cum se construiește această structură arborescentă binară.

Scopul este de a obține un Heap maxim pentru a sorta ascendent elementele acestui array unidimensional. Un Heap maxim se obține atunci când pe fiecare ramură a arborelui nodurile sunt sortate ascendent de la frunze spre rădăcină. Un exemplu de Heap maxim este prezentat în imaginea de mai jos:

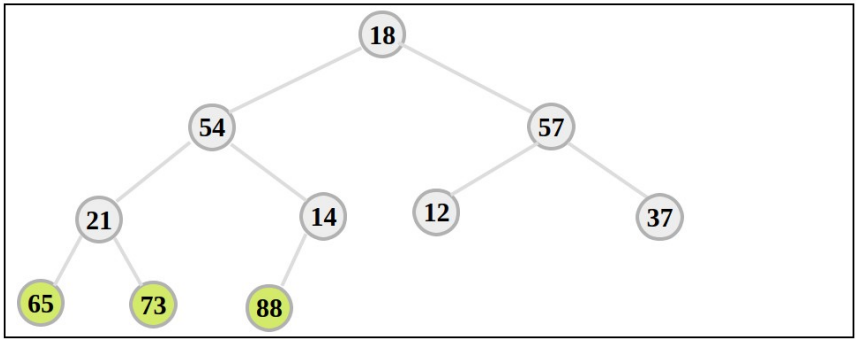


Elementul 88 a fost în vârful arborelui binar, astfel am obținut elementul maxim ce se va afla pe ultima poziție în array-ul sortat ascendent. Acum am interschimbat 88 cu 54 (ultimul element de pe ultimul nivela al arborelui binar). După aceasta vom sorta iarăși ramurile arborelui de la frunze spre rădăcină, acum deja elementul 88 nu se va mai include în sortare. Următorul Heap maxim va avea penultimul element din array-ul sortat, adică numărul 73, vezi imaginea următoare cu al doilea Heap maxim.

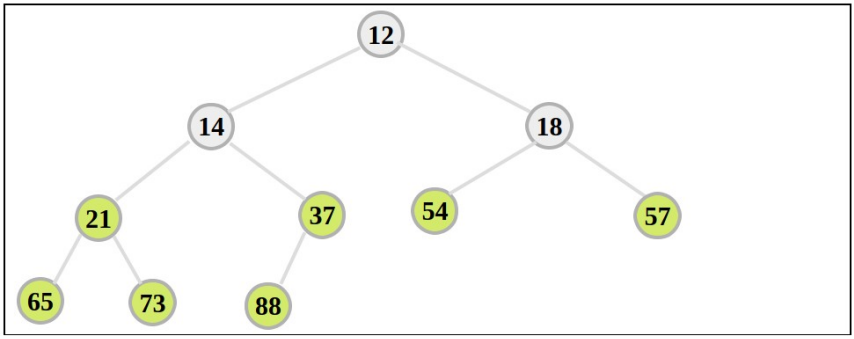


Elementul 73 a fost în vârful arborelui binar, astfel am obținut elementul maxim ce se va afla pe penultima poziție în array-ul sortat ascendent. Acum am inter-

schimbat 73 cu 14 (penultimul element de pe ultimul nivela al arborelui binar). După aceasta vom sorta iarăși ramurile arborelui de la frunze spre rădăcină, acum deja elementul 73 nu se va mai include în sortare.



Elementul 65 a fost în vârful arborelui binar, astfel am obținut elementul maxim de pe poziția a 3-a de la sfârșit în array-ul sortat ascendent. Acum am interschimbat 65 cu 18 (a 3-a valoare de pe ultimul nivela al arborelui binar). După aceasta vom sorta iarăși ramurile arborelui de la frunze spre rădăcină, acum deja elementul 65 nu se va mai include în sortare.



În așa mod procedăm pe toate nivelurile arborelui binar, până obținem elementul minim în arbore situat în nodul rădăcină. După ce vom realiză câteva interschimbări de pe nivelurile rămase, vom obține array-ul sortat ascendent.

i	0	1	2	3	4	5	6	7	8	9
X[i]	12	14	18	21	37	54	57	65	73	88

Implementarea recursivă a algoritmilor în limbajul C++

```
// Exemplu cu implementarea C++ STL 17
#include <iostream>
using namespace std;
void Combina(int a[],int const S,int const M,int const D){
    auto const subArray1=M-S+1;
    auto const subArray2=D-M;
    // Cream array-ul temp
    auto *ArrayS=new int[subArray1],*ArrayD=new int[subArray2];
    // Copiem datele din array-ul temp
    for (auto i=0;i<subArray1;i++)
        ArrayS[i]=a[S+i]; // array-ul stang
    for (auto j=0;j<subArray2;j++)
        ArrayD[j]=a[M+1+j]; // array-ul drept
    auto i1=0,i2=0; // Indicele sub array-urilor 1 si 2
    int imerge=S; // indicele array-ului combinat
    // Combinăm cele două sub-array-uri (stânga și dreapta)
    while (i1<subArray1 && i2<subArray2) {
        if (ArrayS[i1]<=ArrayD[i2]) {
            a[imerge]=ArrayS[i1]; i1++;
        }
        else {
            a[imerge]=ArrayD[i2]; i2++;
        }
        imerge++;
    }
    // Copiem elementele ramase din stânga, dacă există
    while (i1<subArray1) {
        a[imerge]=ArrayS[i1]; i1++; imerge++;
    }
    // Copiem elementele ramase din dreapta, dacă există
    while (i2<subArray2) {
        a[imerge]=ArrayD[i2]; i2++; imerge++;
    }
}
// Funcția de sortare ascendenta
void MergeSortAscendent(int a[], int inceput, int sfarsit){
```

```
    if (inceput>=sfarsit)return;
    auto mijloc=inceput+(sfarsit-inceput)/2;
    MergeSortAscendent(a,inceput,mijloc);
    MergeSortAscendent(a,mijloc+1,sfarsit);
    Combina(a,inceput,mijloc,sfarsit);
}
// Functia de sortare descendenta
void sorteaza(int st, int dr, int x[20]){
    int aux;
    if (x[st]<x[dr]) { // descendent
        aux=x[st]; x[st]=x[dr]; x[dr]=aux;
    }
}
// Functia de combinare a solutiei
void combina(int st, int dr, int m, int x[20]){
    int b[20], i, j, k;
    i=st; j=m+1; k=1;
    while(i<=m && j<=dr)
        if(x[i]>=x[j]){ // descendent
            b[k]=x[i]; i++; k++;
        }
        else{
            b[k]=x[j]; j++; k++;
        }
        if(i<=m)
        for(j=i;j<=m;j++){
            b[k]=x[j]; k++;
        }
        else
            for(i=j;i<=dr;i++){
                b[k]=x[i]; k++;
            }
        k=1;
        for(i=st;i<=dr;i++){
            x[i]=b[k]; k++;
        }
}
// Functia HeapSort de sortare descendenta
void HeapSortDescendent(int st,int dr,int x[]){
    if((dr-st)<=1) sorteaza(st, dr, x);
    else{
        int m=(st+dr)/2; HeapSortDescendent(st,m,x);
```

```
        HeapSortDescendent(m+1,dr,x); combina(st,dr,m,x);
    }
}
// Funcția de afișare a datelor
void AfisareArray(int a[],int n){
    for (auto i=0;i<n;i++)
        cout<<a[i]<<" ";
}
// Programul principal
int main(){
    int a[]={45, 23, 53, 43, 18, 24, 8, 95, 101};
    auto n=sizeof(a)/sizeof(a[0]);
    cout<<"Array-ul inainte de sortare: \n"; AfisareArray(a,n);
    // Aplicăm functia pentru sortarea ascendentă
    MergeSortAscendent(a,0,n-1);
    cout<<"\n\nMergeSort ascendent:\nArray-ul sortat: ";
    AfisareArray(a, n);cout<<endl;
    // Aplicăm functia pentru sortarea deascendentă
    HeapSortDescendent(0,n-1,a);
    cout<<"\nHeapSort descendent:\nArray-ul sortat: ";
    AfisareArray(a,n);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Array-ul inainte de sortare:
45 23 53 43 18 24 8 95 101

MergeSort ascendent:
Array-ul sortat: 8 18 23 24 43 45 53 95 101

HeapSort descendent:
Array-ul sortat: 101 95 53 45 43 24 23 18 8
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi litere majuscule. Aplicați ambii algoritmi de sortare în același program.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi cuvinte citite dintr-un fișier extern separate printr-un spațiu. Aplicați ambii algoritmi de sortare în același program.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei cozi. Aplicați ambii algoritmi de sortare.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei stive. Aplicați ambii algoritmi de sortare.

1.3 Algoritmi de sortare distributivă

Obiective

- *Descrierea algoritmilor de sortare distributivă. (BucketSort & RadixSort).*
- *Prezentarea situațiilor de aplicare a algoritmilor de sortare distributivă.*
- *Elaborarea algoritmilor de sortare distributivă conform specificațiilor propuse.*
- *Implementarea algoritmilor de sortare distributivă în limbajul de programare.*

*Sortarea cupei (**BucketSort**) sau sortarea coșurilor este un algoritm de sortare care funcționează prin distribuirea elementelor unui tablou într-un număr de cupe. Fiecare cupă este apoi sortată individual, fie utilizând un algoritm de sortare diferit, fie aplicând recursiv algoritmul de sortare a cupei. BucketSort poate fi implementată cu comparații și, prin urmare, poate fi considerată și un algoritm de sortare a comparației. Complexitatea de calcul depinde de algoritmul folosit pentru sortarea fiecărei cupe, de numărul de cupe de utilizat și dacă intrarea este distribuită uniform.*

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7
X[i]	43	21	37	49	9	25	3	29

Urmează algoritmul pentru BucketSort:

- *se divide intervalul [0,1) în n subintervale de mărimi egale, numerotate de la 0 la n–1, apoi se distribuie elementele a[i] în intervalul corespunzător: n·a[i];*
- *se sortează fiecare cupă folosind o altă metodă, apoi se combină cele n cupă într-o listă sortată.*

Soluția ar fi următoarea:
Dacă am avea numere din intervalul [0, 1), atunci vom avea 10 cupe de date. Deoarece avem numere din intervalul [0, 100), vom avea 10 cupe de date conform următorii repartizări:

0-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99

Acum vom selecta toate elementele ce corespund acestor intervale de mai jos:

0-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99
9 3		21 25 29	37	43 49					

Observăm că am obținut 4 cupe, dintre care una are doar un element, prin urmare ne rămâne să sortăm ascsndent sau descendent aceste 3 cupe rămase după un anumit algoritm dorit. După sortarea ascendentă a fiecărei cupe în mod individual vom avea următoarele rezultate:

0-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99
3 9		21 25 29	37	43 49					

Combinăm rezultatele sortate din fiecare cupă și obținem datele sortate:

[0, 1)		[20, 29)			[30, 39)	[40, 49)	
3	9	21	25	29	37	43	49

Optimizări

O optimizare obișnuită este să puneți elementele nesortate ale cupelor înapoi în vectorul inițial, apoi să rulați sortarea inserției peste vectorul complet; deoarece durata de execuție a sortării prin inserție se bazează pe cât de departe este fiecare element de poziția sa finală, numărul de comparații rămâne relativ mic, iar ierarhia memoriei este mai bine exploatată prin stocarea listei contigue în memorie. Dacă distribuția de intrare este cunoscută sau poate fi estimată, pot fi adesea alese cupe care conțin densitate constantă (mai degrabă decât având doar dimensiuni constante). Aceasta permite o omplexitate medie a timpului O(n), chiar și fără intrări distribuite uniform.

```
#include <iomanip>
#include <iostream>
using namespace std;
#define INTERVAL 10  // Capacitatea fiecarei cupe
int a[] = {42,14,32,33,17,37,19,47,51,85};
int n=sizeof(a)/sizeof(a[0]),nCupe=n-1;
// Structura nodului unei liste liniare simple
struct Node {
    int data;
    struct Node *next;
};
// Crearea indecsilor pentru cupe
int BucketIndex(int valoare) {
    return valoare/INTERVAL;
}
void AfisareCupe(struct Node *list) {
    struct Node *curent = list;
    while (curent){
        cout<<setw(3)<<curent->data; curent=curent->next;
    }
}
// Functia de a sorta elementele din fiecare cupa
struct Node *InsertionSort(struct Node *list) {
    struct Node *k, *nodeList;
    if (list==0 || list->next==0) {
        return list;
    }
    nodeList=list; k=list->next; nodeList->next=0;
    while (k!=0) {
        struct Node *ptr;
        if (nodeList->data > k->data) {
            struct Node *tmp; tmp = k; k = k->next;
            tmp->next = nodeList; nodeList = tmp; continue;
        }
        for (ptr=nodeList; ptr->next!=0; ptr=ptr->next){
            if (ptr->next->data > k->data) break;
        }
        if (ptr->next != 0) {
            struct Node *tmp; tmp=k; k=k->next;
            tmp->next = ptr->next; ptr->next = tmp; continue;
```

```
        } else {
            ptr->next=k; k=k->next;ptr->next->next=0;continue;
        }
    }
    return nodeList;
}
// Functia de sortare ascendenta
void BucketSort(int a[]) {
    int i,j; struct Node **cupe;
    // Cream cupe si alocam dimensiunea memoriei
    cupe=(struct Node **)malloc(sizeof(struct Node *) * nCupe);
    // Initializam cupele cu 0, goale
    for (i=0; i<n; ++i) {
        cupe[i]=NULL;
    }
    // Umplem cupele cu elementele respective
    for (i=0; i<n; ++i) {
        struct Node *current; int pos=BucketIndex(a[i]);
        current=(struct Node *)malloc(sizeof(struct Node));
        current->data=a[i]; current->next=cupe[pos];
        cupe[pos]=current;
    }
    // Afisam cupele impreuna cu elementele lor
    cout<<"\nSortarea datelor pe cupe: "<<endl;
    for (i=0; i<nCupe; i++) {
        cout <<"\tCupa["<<i<<"]: "; AfisareCupe(cupe[i]);
        cout<<endl;
    }
    // Sortam elementele din fiecare cupa
    for (i=0; i<nCupe; ++i) {
        cupe[i]=InsertionSort(cupe[i]);
    }
    cout<<"\n\nCupele dupa sortare: "<<endl;
    for (i=0; i<nCupe; i++) {
        cout<<"\tCupa["<<i<<"]: "; AfisareCupe(cupe[i]);
        cout<<endl;
    }
    // Fixam elementele sortate in array
    for (j=0, i=0; i<nCupe; ++i) {
        struct Node *node; node=cupe[i];
        while (node){
            a[j++]=node->data; node=node->next;
```

```
        }
    }
    for (i=0; i<nCupe; ++i){
        struct Node *node; node=cupe[i];
        while (node){
            struct Node *tmp; tmp = node;
            node = node->next;
        }
    }
}
// Afisarea datelor array-ului
void AfisareArray(int a[]) {
    for (int i=0; i<n; i++) {
        cout<<setw(3)<<a[i];
    }
    cout<<endl;
}
// Programul principal
int main() {
    cout<<"Array-ul inainte de sortare: \n";
    AfisareArray(a); BucketSort(a);
    cout<<"\nBucketSort ascendent:\nArray-ul sortat: ";
    AfisareArray(a);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Array-ul inainte de sortare:
 42 14 32 33 17 37 19 47 51 85

Sortarea datelor pe cupe:
        Cupa[0]:
        Cupa[1]:  19 17 14
        Cupa[2]:
        Cupa[3]:  37 33 32
        Cupa[4]:  47 42
        Cupa[5]:  51
        Cupa[6]:
        Cupa[7]:
        Cupa[8]:  85

Cupele dupa sortare:
        Cupa[0]:
        Cupa[1]:  14 17 19
        Cupa[2]:
        Cupa[3]:  32 33 37
        Cupa[4]:  42 47
        Cupa[5]:  51
        Cupa[6]:
        Cupa[7]:
        Cupa[8]:  85

BucketSort ascendent:
Array-ul sortat:  14 17 19 32 33 37 42 47 51 85
```

*În informatică, **CountingSort** este un algoritm pentru sortarea unei colecții de obiecte în funcție de chei care sunt mici întregi pozitivi; adică este un algoritm de sortare întregi. Funcționează numărând numărul de obiecte care posedă valori cheie distincte și aplicând suma prefixului pe aceste numere pentru a determina pozițiile fiecărei valori cheie în secvența de ieșire. Durata sa de funcționare este liniară în ceea ce privește numărul de articole și diferența dintre valoarea maximă a cheii și valoarea minimă a cheii, deci este potrivită doar pentru utilizarea directă în situații în care variația tastelor nu este semnificativ mai mare decât numărul de elemente. Este adesea folosit ca subrutină în sortarea radix, un alt algoritm de sortare, care poate gestiona tastele mai mari mai eficient.*

CountingSort nu este o sortare de comparație; folosește valori cheie ca indici într-o matrice și limita inferioară Ω (n log n) pentru sortarea comparației nu se va aplica. BucketSort poate fi utilizată în locul numărării sortării și implică o analiză de timp similară. Cu toate acestea, comparativ cu CountingSort, BucketSort necesită liste legate, array-uri dinamice sau o cantitate mare de memorie prealocată pentru a deține seturile de articole din fiecare cupă, în timp ce CountingSort stochează un singur număr (numărul de articole) pe cupă.

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7
X[i]	1	2	2	3	8	3	2	6

Soluția ar fi următoarea:

Găsim elementul maxim din array, max=8. Inițializăm un array de lungime max+1 cu toate elementele 0. Acest matrice este utilizat pentru stocarea numărului de elemente din array.

i	0	1	2	3	4	5	6	7	8
X[i]	0	0	0	0	0	0	0	0	0

Stocăm numărul fiecărui element la indexul respectiv în array-ul de numărare. De exemplu: dacă numărul elementului 2 este 3, atunci 3 este stocat în poziția a 2-a a array-ului de numărare. Dacă elementul "9" nu este prezent în array, atunci 0 este stocat în poziția a 9-a. Astfel obținem următorul tabel de date:

i	0	1	2	3	4	5	6	7	8
X[i]	0	1	3	2	0	0	1	0	1

Stocăm suma cumulativă a elementelor array-ului de numărare. Ajută la plasarea elementelor în indexul corect al array-ului sortat.

i	0	1	2	3	4	5	6	7	8
X[i]	0	1	4	6	6	6	7	7	8

Găsim indexul fiecărui element al array-ului original din array-ului de numărare. Acest lucru oferă numărul cumulativ. Plasăm elementul la indexul calculat. După plasarea fiecărui element în poziția corectă, micșorăm numărul acestuia cu 1 (cifra unu):

i	0	1	2	3	4	5	6	7
X[i]	1	2	2	3	8	3	2	6

i	0	1	2	3	4	5	6	7	8
X[i]	0	1	4	6	6	6	7	7	8

1-1=0 7-1=6 8-1=7

i	0	1	2	3	4	5	6	7
X[i]	1	2	2	2	3	3	6	8

*În informatică, **RadixSort** este un algoritm de sortare necomparativ. Evită comparația prin crearea și distribuirea elementelor în găleți în funcție de baza lor. Pentru elementele cu mai mult de o cifră semnificativă, acest proces de colectare se repetă pentru fiecare cifră, păstrând în același timp ordonarea etapei anterioare, până când toate cifrele au fost luate în considerare. Din acest motiv, RadixSort poate fi numită și BucketSort și sortare digitală. RadixSort poate fi aplicată datelor care pot fi sortate lexicografic, fie ele întregi, cuvinte, etc .*

Ideea RadixSort este de a sorta cifră cu cifră începând de la cifra cea mai puțin semnificativă până la cea mai semnificativă. RadixSort folosește sortarea de numărare (CountingSort) ca subrutină pentru sortare.

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7	8	9
X[i]	15	120	53	36	167	81	75	32	9	60

Soluția ar fi următoarea:

- *După cum am menționat anterior, algoritmul aranjează numerele folosind cifra cea mai puțin semnificativă până la cea mai semnificativă, prin urmare trebuie să cunoaștem gama acestor iterații. În acest scop, trebuie să găsim elementul maxim, deci* ***Maxim=167****.*
- *Trebuie să știm de câte ori trebuie să aranjăm tabloul, de aceea este necesar să numărăm numărul de cifre ale lui Maxim, adică* ***NMax=3****.*
- *Aranjamentul inițial de sortare ne cere să sortăm elementele pe baza cifrei lor mai puțin semnificative. Punctul cheie de remarcat aici este că numerele cu unități diferite sunt plasate, dar cifrele identice în alte poziții sunt aranjate reciproc în modul corect. De exemplu, numărul 36 de la indexul 3 și numărul 32 de la indexul 7 au o cifră diferită în locul unităților, dar alte cifre sunt identice. Inițial, 32 este după 36 în array, dar după aranjarea numerelor în funcție de locul unităților, 32 vine înainte de 36. Astfel obținem următorul array de date:*

i	0	1	2	3	4	5	6	7	8	9
X[i]	120	60	81	32	53	15	75	36	167	9

- *Păstrând aranjamentul actual intact, vom încerca să aranjăm numerele pe baza locului zecilor. În acest scop, vom avea nevoie de un tip stabil în timpul implementării. Algoritmii de sortare stabili sunt aceia, care păstrează dispunerea inițială a numerelor egale. Astfel obținem următorul array de date:*

i	0	1	2	3	4	5	6	7	8	9
X[i]	09	15	120	32	36	53	60	167	75	81

- *După finalizarea sortării bazate pe cifra zecilor, putem observa că toate numerele sub 100 sunt aranjate corect între ele. Practic, dacă un array conține numere sub 100, doar două iterații de aranjare vor sorta array-ul complet.*
- *În acest exemplu, cea mai semnificativă cifră este cifra de sute de locuri. Prin urmare, acesta este ultimul pas al algoritmului de sortare Radix.*

i	0	1	2	3	4	5	6	7	8	9
X[i]	9	15	32	36	53	60	75	81	120	167

- *După cum putem vedea, array-ul este acum complet sortat. Acest lucru se întâmplă de îndată ce array-ul este aranjat în funcție de cifra cea mai semnificativă. Înainte de a trece la implementarea sortării Radix, vă recomandăm să studiați conceptele algoritmului de numărare a sortării, care este folosit ca subrutină pentru sortarea numerelor pe baza cifrelor individuale.*

Implementarea recursivă a algoritmilor în limbajul C++

```
#include <iomanip>
#include <iostream>
using namespace std;
int a[] = {10,21,21,32,48,32,12,46};
int n=sizeof(a)/sizeof(a[0]);
// Functie pentru a obtine valoarea maxima din a[]
int ValoareMax(int a[],int n){
    int mx=a[0];
    for (int i=1;i<n;i++) if (a[i]>mx) mx=a[i];
    return mx;
}
int maxim=ValoareMax(a,n); // Gasim numarul maxim
// Functia care face numararea sortarilor din a[] in functie de
cifra reprezentata de exp.
void CountingSort(int a[], int n, int exp){
    int b[n],i,num1[10]={0};
    cout<<"\nAfisam suma cumulativa: "<<endl;
    for (i=0; i<n; i++) num1[(a[i]/exp)%10]++;
    // Schimbam num1[i] astfel incat num1[i] sa contina acum
pozitia reala a acestei cifre in b[]
    for (i=1; i<10; i++){
        num1[i]+=num1[i-1]; cout<<setw(5)<<num1[i];
    }
    // Counstruim array-ul b
    for (i=n-1; i>=0; i--) {
        b[num1[(a[i]/exp)%10]-1]=a[i]; num1[(a[i]/exp)%10]--;
    }
    cout<<endl;
    // Copiaem array-ul b[] în a[], astfel incat a[] sa contina
```

```
acum numere sortate in functie de cifra curenta
    for (i=0; i<n; i++) a[i]=b[i];
}
// Funcția Radix Sort
void RadixSortAscendent(int a[], int n){
    // Efectuam CountingSort pentru fiecare cifra și retinem ca
in loc sa trecem numarul cifrei, exp este expediat, exp este
10^i unde i este numarul cifrei curente.
    for (int exp=1; maxim/exp>0; exp*=10)
        CountingSort(a,n,exp);
}
// Afisarea datelor array-ului
void AfisareArray(int a[]) {
    for (int i=0; i<n; i++) cout<<setw(5)<<a[i];
    cout<<endl;
}
int main() {
    cout<<"Array-ul inainte de sortare: \n"; AfisareArray(a);
    RadixSortAscendent(a,n);
    cout<<"\nElementele sortate: \n"; AfisareArray(a);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Array-ul inainte de sortare:
   10   21   21   32   48   32   12   46
Afisam suma cumulativa:
    3    6    6    6    6    7    7    8    8
Afisam suma cumulativa:
    2    4    6    8    8    8    8    8    8
Elementele sortate:
   10   12   21   21   32   32   46   48
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă (vor realiza elevii cu numărul de ordine par în registrul grupei) și descendentă (vor realiza elevii cu numărul de ordine impar în registrul grupei) a datelor în cazul când aceste date vor fi litere majuscule. Aplicați ambii algoritmi de sortare în același program.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă (vor realiza elevii cu numărul de ordine impar în registrul grupei) și descendentă (vor realiza elevii cu numărul de ordine par în registrul grupei) a datelor în cazul când aceste date vor fi cuvinte citite dintr-un fișier extern separate printr-un spațiu. Aplicați ambii algoritmi de sortare în același program.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei liste liniare simplu înlănțuite. Aplicați ambii algoritmi de sortare studiați.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei unei liste liniare dublu înlănțuite. Aplicați ambii algoritmi de sortare studiați.

1.4 Algoritmi de sortare cu bule modificaţi

Obiective

- *Descrierea algoritmilor de sortare cu bule modificați. (CocktailSort & CombSort).*
- *Prezentarea situațiilor de aplicare a algoritmilor de sortare cu bule modificați.*
- *Elaborarea algoritmilor de sortare cu bule modificați conform specificațiilor propuse.*
- *Implementarea algoritmilor de sortare cu bule modificați în limbajul de programare.*

*Algoritmul **CocktailSort**, cunoscută și sub numele de sortare bidirecțională cu bule (care se poate referi și la o variantă de sortare de selecție), sortare ondulată, sortare amestecată sau sortare navetă, este o extindere a sortării cu bule. Algoritmul extinde sortarea cu bule operând în două direcții. Deși îmbunătățește sortarea cu bule, mutând mai repede articolele la începutul listei, oferă doar îmbunătățiri marginale ale performanței. La fel ca majoritatea variantelor de sortare cu bule, CocktailSort este folosit în primul rând ca instrument educațional. Algoritmi mai performanți, cum ar fi TimSort sau sortare fuzionată (FusionSort), sunt folosiți de bibliotecile de sortare încorporate în limbaje de programare populare, cum ar fi Python și Java.*

CocktailSort este o ușoară variație a sortării cu bule. Diferă prin faptul că, în loc să treacă repetat prin listă de jos în sus, trece alternativ de jos în sus și apoi de sus în jos. Poate obține performanțe ușor mai bune decât un tip de bule standard. Motivul pentru aceasta este că sortarea cu bule trece doar prin listă într-o singură direcție și, prin urmare, poate muta elementele înapoi cu un singur pas, fiecare iterație.

Un exemplu de listă care dovedește acest punct este lista (2,3,4,5,1), care ar trebui să treacă printr-o singură trecere de CocktailSort pentru a fi sortată, dar dacă se folosește o sortare cu bule ascendent ar dura patru verificări. Cu toate acestea, un permis de sortare pentru CocktailSort ar trebui să fie numărat ca două permise de sortare cu bule. De obicei, CocktailSort este de mai puțin de două ori mai rapid decât sortarea cu bule.

O altă optimizare poate fi că algoritmul amintește unde a fost făcut ultimul swap real (interschimb). În următoarea iterație, nu vor exista swap-uri peste ace-

astă limită, iar algoritmul are treceri mai scurte. Pe măsură ce CocktailSort merge bidirecțional, gama de swap-uri posibile, care este intervalul de testat, se va reduce pe trecere, reducând astfel ușor timpul total de funcționare. Algoritmul de sortare Bubble traversează întotdeauna elementele din stânga și mută cel mai mare element în poziția corectă în prima iterație și al doilea ca mărime în a doua iterație și așa mai departe. CocktailSort traversează un array dat în ambele direcții, alternativ.

Exemplu pas cu pas:

i	0	1	2	3	4	5	6
X[i]	5	1	4	2	8	0	2

Urmează algoritmul pentru CocktailSort:
Fiecare iterație a algoritmului este împărțită în 2 etape:

- *Prima etapă parcurge array-ul de la stânga la dreapta, la fel ca Bubble Sort. În timpul buclei, elementele adiacente sunt comparate și dacă valoarea din stânga este mai mare decât valoarea din dreapta, atunci valorile sunt schimbate. La sfârșitul primei iterații, cel mai mare număr va locui la sfârșitul array-ului.*
- *A doua etapă parcurge array-ul în direcție opusă - începând de la element chiar înainte de cel mai recent element sortat și revenind la începutul array-ului. Aici, elementele adiacente sunt comparate și sunt schimbate dacă este necesar.*

Soluția ar fi următoarea:

Primul pas înainte:

1. *(5 1 4 2 8 0 2) ? (1 5 4 2 8 0 2), Schimb de la 5> 1*
2. *(1 5 4 2 8 0 2) ? (1 4 5 2 8 0 2), Schimb de la 5> 4*
3. *(1 4 5 2 8 0 2) ? (1 4 2 5 8 0 2), Schimb de la 5> 2*
4. *(1 4 2 5 8 0 2) ? (1 4 2 5 8 0 2)*
5. *(1 4 2 5 8 0 2) ? (1 4 2 5 0 8 2), Schimb de la 8> 0*
6. *(1 4 2 5 0 8 2) ? (1 4 2 5 0 2 8), Schimb de la 8> 2*

După prima trecere înainte, cel mai mare element al array-ului va fi prezent la ultimul index al array-ului.

Prima trecere înapoi:

1. *(1 4 2 5 0 2 8)? (1 4 2 5 0 2 8)*
2. *(1 4 2 5 0 2 8)? (1 4 2 0 5 2 8), Schimb de la 5> 0*
3. *(1 4 2 0 5 2 8)? (1 4 0 2 5 2 8), Schimb de la 2> 0*
4. *(1 4 0 2 5 2 8)? (1 0 4 2 5 2 8), Schimb de la 4> 0*
5. *(1 0 4 2 5 2 8)? (0 1 4 2 5 2 8), Schimb de la 1> 0*

După prima trecere înapoi, cel mai mic element al array-ului va fi prezent la primul index al array-ului.

A doua trecere înainte:

1. *(0 1 4 2 5 2 8)? (0 1 4 2 5 2 8)*
2. *(0 1 4 2 5 2 8)? (0 1 2 4 5 2 8), Schimb de la 4> 2*
3. *(0 1 2 4 5 2 8)? (0 1 2 4 5 2 8)*
4. *(0 1 2 4 5 2 8)? (0 1 2 4 2 5 8), Schimb de la 5> 2*

A doua trecere înapoi:

1. *(0 1 2 4 2 5 8)? (0 1 2 2 4 5 8), Schimb de la 4> 2*

Acum, array-ul este deja sortat, dar algoritmul nostru nu știe dacă este finalizat. Algoritmul trebuie să finalizeze această întreagă trecere fără niciun schimb pentru a ști că este sortat.

2. *(0 1 2 2 4 5 8)? (0 1 2 2 4 5 8)*
3. *(0 1 2 2 4 5 8)? (0 1 2 2 4 5 8)*

***CombSort** este un algoritm de sortare relativ simplu proiectat inițial de Włodzimierz Dobosiewicz și Artur Borowy în 1980, redescoperit ulterior (și dat numele „Combsort") de Stephen Lacey și Richard Box în 1991. Sortarea pe pieptene se îmbunătățește la sortarea cu bule în același mod în care Shellsort îmbunătățește la sortarea prin inserare. Definiția „sortare în creștere în creștere" menționează termenul „sortare pe pieptene" ca vizualizare a unor treceri iterative ale datelor, „acolo unde ating dinții unui pieptene;" fostul termen este legat de Don Knuth.*

Comb Sort este forma avansată a sortării cu bule. Bubble Sort compară toate valorile adiacente în timp ce sortarea pe pieptene elimină toate valorile broaștei țestoase sau valorile mici de la sfârșitul listei. Factorii care afectează sortarea pieptenei sunt:

- *Îmbunătățește sortarea cu bule folosind un spațiu de dimensiune mai mare de 1.*
- *Decalajul începe cu o valoare mare și se micșorează cu factorul 1,3.*
- *Decalajul se micșorează până când valoarea atinge 1.*

Exemplu pas cu pas:

i	0	1	2	3	4	5	6	7	8	9
X[i]	7	5	1	50	3	–20	25	–4	30	0

Urmează algoritmul pentru CombSort:

1. *Creați și inițializați variabile decalaj și schimbare și factorul de reducere F: decalaj = dimensiunea array-ului, schimbare = false și F = 1.3.*
2. *Setăm decalaj = decalaj / F;*
3. *Iterați peste array de la i = 0 la i <n - decalaj:*
   dacă a[i]> a [i + decalaj]:
   a) schimbați elementele a [i] și a [i + decalaj]
   b) setăm schimbare = true
4. *Repetați pașii 2-4 în timp ce decalaj ! = 1 și schimbare = true.*

Soluția ar fi următoarea:

Creăm și inițializăm datele: decalaj = 10, F=1.3. Elementele din array care sunt evidențiate cu culoarea neagră, stil BOLD, se vor interschimba între ele. Pentru a înțelege mai bine, cum se interschimbă elementele, analizați interațiile 1-5 de mai jos:

Iterația 1	**Iterația 2**
După actualizare, decalaj = 10/1.3 = 7 **7** 5 1 50 3 -20 25 **-4** 30 0 -4 5 **1** 50 3 -20 25 7 30 **0** **-4 5 0 50 3 -20 25 7 30 1**	*După actualizare, decalaj = 7/1.3 = 5* **-4** 5 0 50 3 **-20** 25 7 30 1 -20 5 0 **50** 3 -4 25 7 **30** 1 -20 5 0 30 **3** -4 25 7 50 **1** **-20 5 0 30 1 -4 35 7 50 3**

Iterația 3	**Iterația 4**
După actualizare, decalaj = 5/1.3 = 3 -20 **5** 0 50 **3** -4 25 7 30 1 -20 3 **0** 50 5 **-4** 25 7 30 1 -20 3 -4 **30** 5 0 **25** 7 50 1 -20 3 -4 25 5 0 **30** 7 50 **1** **-20 3 -4 25 5 0 1 7 50 30**	*După actualizare, decalaj = 3/1.3 = 2* -20 3 -4 **25** 5 **0** 30 7 50 1 -20 3 -4 0 **5** 25 **1** 7 50 30 -20 3 -4 0 1 **25** 5 **7** 50 30 **-20 3 -4 0 1 7 5 25 50 30**

Penultima iterație va fi următoarea:

Iterația 5	Iterația 6
După actualizare, decalaj = 2/1.3 = 1 -20 **3** **-4** 0 1 7 5 25 50 30 -20 -4 **3** **0** 1 7 5 25 50 30 -20 -4 0 **3** **1** 7 5 25 50 30 -20 -4 0 1 3 **7** **5** 25 50 30 -20 -4 0 1 3 5 7 25 **50** **30** **-20 -4 0 1 3 5 7 25 30 50**	*Array-ul sortat ascendent:* **-20 -4 0 1 3 5 7 25 30 50**

Implementarea recursivă a algoritmilor în limbajul C++

```
#include <iostream>
using namespace std;
// Functia de sortare ascendenta
void CocktailSort(int a[], int n){
    bool Schimbat=true;
    int s=0,d=n-1;
    while (Schimbat){
        // resetați semnalizatorul schimbat la intrarea în bu-
clă, deoarece ar putea fi adevărat dintr-o iterație anterioară.
        Schimbat=false;
        // bucla de la stânga la dreapta la fel ca BubbleSort
        for (int i=s; i<d; ++i) {
            if (a[i]>a[i+1]) { // ascendent
                // swap inseamna interschimbare
                swap(a[i],a[i+1]); Schimbat=true;
            }
        }
        // dacă nu s-a mutat nimic, atunci este sortat.
        if (!Schimbat) break;
        // în caz contrar, resetam semnalizatorul schimbat ast-
fel încât să poată fi utilizat în etapa următoare
        Schimbat=false;
        // mutați punctul final înapoi cu unul, deoarece ele-
mentul de la sfârșit se află în locul potrivit
        --d;
        // de la dreapta la stânga, făcând aceeași comparație
```

```
ca în etapa anterioară
        for (int i=d-1; i>=s; --i){
            if (a[i]>a[i+1]) { // ascendent
                swap(a[i],a[i+1]); Schimbat=true;
            }
        }
        ++s;
    }
}
int DecalajNou(int d){
    d=(d*10)/13; // d inseamna decalajul
    if (d==9 || d==10) d=11;
    if (d<1) d=1; return d;
}
// Functia de sortare descendenta
void CombSortDescendent(int a[], int n){
    int d=n,temp,i;
    for (;;){
        d=DecalajNou(d); int Schimbat=0;
        for (i=0; i<n-d; i++){
            if (a[i]<a[i+d]){ // descendent
                temp=a[i];a[i]=a[i+d]; a[i+d]=temp;Schimbat=1;
            }
        }
        if (d==1 && !Schimbat) break;
    }
}
//Afisarea array-ului
void AfisareArray(int a[], int n){
    for (int i=0; i<n; i++) cout<<a[i]<<"  ";
    cout<<endl;
}
// Programul principal
int main(){
    int a[]={5,1,4,2,8,0,2,7,3,4,6}, n=sizeof(a)/sizeof(a[0]);
    cout<<"Array-ul initial pentru sortare:\n";
    AfisareArray(a, n); CocktailSort(a, n);
    cout<<"\nCocktailSort:\nArray-ul sortat ascendent:\n\t";
    AfisareArray(a, n); CombSortDescendent(a,n);
    cout<<"\nCombSort:\nArray-ul sortat descendent:\n\t";
    AfisareArray(a, n);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Array-ul initial pentru sortare:
5  1  4  2  8  0  2  7  3  4  6

CocktailSort:
Array-ul sortat ascendent:
       0  1  2  2  3  4  4  5  6  7  8

CombSort:
Array-ul sortat descendent:
       8  7  6  5  4  4  3  2  2  1  0
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă (vor realiza elevii cu numărul de ordine par în registrul grupei) și descendentă (vor realiza elevii cu numărul de ordine impar în registrul grupei) a datelor în cazul când aceste date vor fi litere minuscule.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă (vor realiza elevii cu numărul de ordine impar în registrul grupei) și descendentă (vor realiza elevii cu numărul de ordine par în registrul grupei) a datelor în cazul când aceste date vor fi cuvinte citite dintr-un fișier extern separate printr-o virgulă.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei liste circulare simplu înlănțuite.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă și descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei unei liste circulare dublu înlănțuite.

Algoritmi de sortare
Itemi recomandaţi pentru lecţiile practice 1-4

1. *Implementați algoritmii de sortare pas cu pas, ascendent vor efectua elevii cu numărul de ordine par și descendent vor efectua elevii cu numărul de ordine impar. Următoarele șiruri de date se vor citi dintr-un fișier extern. Pentru agoritmii de sortare aplicați se va utiliza un array alocat dinamic.*

Nr.	Partea 1	Nr.	Partea 2
1	82, 59, 34, 55, 16, 85, 53, 86, 57, 62	**9**	70, 37, 76, 43, 24, 11, 34, 22, 19, 48
2	39, 10, 85, 28, 66, 70, 79, 32, 19, 15	**10**	76, 11, 67, 49, 14, 43, 70, 45, 20, 73
3	86, 29, 78, 53, 85, 16, 70, 64, 21, 82	**11**	14, 48, 64, 80, 38, 61, 90, 82, 52, 41
4	34, 16, 81, 73, 24, 87, 33, 30, 18, 40	**12**	99, 35, 39, 34, 54, 33, 17, 62, 13, 24
5	73, 90, 36, 60, 21, 75, 76, 33, 80, 51	**13**	66, 56, 24, 34, 43, 57, 51, 65, 44, 69
6	80, 74, 13, 16, 82, 39, 89, 20, 96, 85	**14**	88, 23, 66, 55, 85, 15, 35, 52, 17, 90
7	49, 63, 64, 54, 98, 14, 67, 55, 60, 58	**15**	12, 51, 42, 92, 85, 82, 49, 56, 33, 47
8	20, 33, 45, 82, 66, 15, 57, 47, 69, 24	**16**	44, 97, 19, 18, 47, 99, 38, 65, 28, 77

2. *Implementați algoritmii de sortare pas cu pas, ascendent vor efectua elevii cu numărul de ordine par și descendent vor efectua elevii cu numărul de ordine impar. Următoarele șiruri de date se vor citi dintr-un fișier extern. Pentru agoritmii de sortare aplicați se va utiliza un array alocat dinamic.*

Nr.	Partea 1	Nr.	Partea 2
1	h z t e q m k n e m s o t l e u i a h c	**9**	n e m s o t l e u i a h c h z t e q m k
2	c u u i t u c h x i k b r h n q z m j x	**10**	k b r h n q z m j x c u u i t u c h x i
3	m b k c s m d m x w y x i y r r x z a y	**11**	w y x i y r r x z a y m b k c s m d m x
4	c e d l z d y t z u o k n q e o t c k f	**12**	z u o k n q e o t c k f c e d l z d y t

Nr.	Partea 1	Nr.	Partea 2
5	e m s o t l e u i a h c h z t e q m k n	**13**	e a h c h z t e q m k n m s o t l e u i
6	h x i k b r h n q z m j x c u u i t u c	**14**	n q z m j x c u u i t u c h x i k b r h
7	y x i y r r x z a y m b k c s m d m x w	**15**	a y m b k c s m d m x w y x i y r r x z
8	n q e o t c k f c e d l z d y t z u o k	**16**	k f c e d l z d y t z u o k n q e o t c

3. *Analizați cele 3 tehnici de parcurgere a elementelor unui array 2D:*

a. Parcurgerea sub formă de șarpe (șerpuită):

Date de intrare			Date de ieșire		
4	6	5	1	2	3
8	2	1	6	5	4
7	3	9	7	8	9

Date de intrare			Date de ieșire		
4	6	5	9	8	7
8	2	1	4	5	6
7	3	9	3	2	1

b. Parcurgerea sub formă de spirală:

Date de intrare			Date de ieșire		
4	6	5	1	2	3
8	2	1	8	9	4
7	3	9	7	6	5

Date de intrare			Date de ieșire		
4	6	5	9	8	7
8	2	1	2	1	6
7	3	9	3	4	5

c. Parcurgerea sub formă de diagonală:

Date de intrare			Date de ieșire		
4	6	5	1	2	6
8	2	1	3	5	7
7	3	9	4	8	9

Date de intrare			Date de ieșire		
4	6	5	9	8	4
8	2	1	7	5	3
7	3	9	6	2	1

4. *Implementați algoritmii de sortare pas cu pas, ascendent vor efectua elevii cu numărul de ordine impar și descendent vor efectua elevii cu numărul de ordine par. Următoarele array-uri de date se vor citi dintr-un fișier extern. Pentru agoritmii de sortare aplicați se va utiliza un array bidimensional alocat dinamic. Fiecare elev va realiza cele 3 parcurgeri conform celor 3 metode de parcurgere prezentate anterior: șerpuită, spirală (circulară) și diagonală.*

Nr.	Datele de intrare	Nr.	Datele de intrare
1	12 39 59 72 43 60 59 86 74	**9**	A C E B D F I H G
2	12 45 67 31 34 59 29 86 78	**10**	B D F A C E I H G
3	95 80 78 14 31 56 23 42 65	**11**	I H G A C E B D F
4	11 31 15 12 14 16 91 18 71	**12**	B H E C D L I J G
5	12 14 16 15 35 15 95 18 73	**13**	A C E B Z F P H G
6	91 18 71 17 31 15 22 44 16	**14**	V D L B H E I J G
7	11 63 75 22 54 86 *39 48 97*	**15**	I J G B D L P H E
8	18 33 51 27 44 60 96 85 72	**16**	A S E B D F G H M

5. *Fie un array unidimensional de date întregi (numere de 2 cifre), care sunt generate într-un fișier text. Afișați array-ul de date inițiale citit din fișier, apoi realizați subprograme care sortează:*

a) *ascendent elementele pare ale acestuia, iar elementele impare le sortează descendent;*
b) *ascendent elementele pozitive ale acestuia, iar elementele negative le sortează descendent;*
c) *ascendent elementele prime ale acestuia, iar elementele care nu sunt prime le sortează descendent;*
d) *ascendent elementele pătrate perfecte ale acestuia, iar elementele care nu sunt pătrate perfecte le sortează descendent;*
e) *ascendent elementele cuburi perfecte ale acestuia, iar elementele care nu sunt cuburi perfecte le sortează descendent.*

Date de intrare	Date de ieșire
9 -11 -20 16 30 25 27 -40 64 49	a) -40 -20 16 30 64 49 27 25 -11 b) 16 25 27 30 49 64 -11 -20 -40 c) -11 64 49 30 27 25 16 -20 -40 d) 16 25 49 64 30 27 -11 -20 -40 e) 27 64 49 30 25 16 -11 -20 -40

6. *Fie un array unidimensional de date întregi (numere de 2 cifre), care sunt generate într-un fișier text. Afișați array-ul de date inițiale citit din fișier, apoi realizați subprograme care sortează:*
   a) *ascendent elementele mai mici decât partea întreagă a elementului M=(minim+maxim)/2 și descendent – celelalte elemente din array;*
   b) *ascendent elementele mai mici decât partea întreagă a elementului M=(minim pătrat perfect +maxim cub perfect)/2 și descendent – celelalte elemente din array;*

Date de intrare	Date de ieșire
9 -11 -20 16 30 25 27 -40 64 49	M=(-40+64)/2=12 -40 -20 -11 64 49 30 27 25 16 M=(16+64)/2=40 -40 -20 -11 16 25 27 30 64 49

7. ****Un arhitector din Chișinău a reușit să rezolve o problemă de la primărie. El a stabilit că în oraș nu există clădiri cu înălțimea mai mare decât H. Primarul cere de la arhitector afișarea clădirilor în ordine ascendentă (sau descendentă), precum și o verificare: pentru fiecare clădire din lista ordonată, dacă înălțimea ei este egală cu media aritmetică a înălțimilor celor două*

clădiri vecine. Arhitectorul cere ajutorul ca să-și termine proiectul care a devenit puțin complicat de realizat, deoarece pentru 40% din teste n ≤ 50 000, pentru 80% din teste n ≤ 500 000 și se consideră că înaintea primei clădiri și după ultima clădire se află câte o pseudoclădire de înălțime 0 – care va interveni în verificarea cerută.

Date de intrare	Date de ieșire
10 5 10 10 9 5 2 5 8 5 2	10 10 9 8 5 5 5 5 2 2 0 0 1 0 0 1 1 0 0 0

Explicație:

- *Șirul devine 10 10 9 8 5 5 5 5 2 2*
- *Doar 9 si cei doi de 5 din mijloc respectă condiția.*

8. ****Mircea, profesor de algebră a găsit un șir cu N numere naturale și un număr natural p. Profesorul vă cere să-l ajutați efectuând următoarea sarcină: deteminați câți divizori are numărul din șir aflat pe poziția p. Afișați în ordine descendentă numerele din șir care au același număr de divizori ca cel aflat pe poziția p.*

Date de intrare	Date de ieșire
10 3 1 5 95 23 16 39 77 74 97 57	95 are 4 divizori 95 77 74 57 39

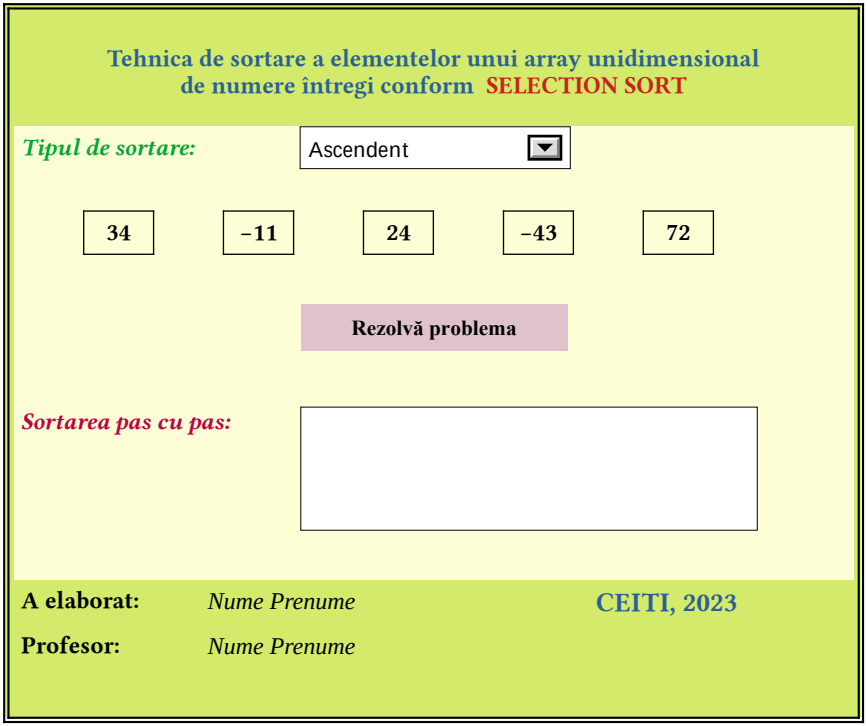
9. ****Eugenia, profesoară de geometrie a găsit un șir cu N numere naturale și un număr natural k. Profesoara vă cere s-o ajutați efectuând următoarea sarcină: deteminați câte diagonale are poligonul regulat cu lungimea laturii din șir aflată pe poziția k. Dacă numărul de diagonale ale poligonului dat este par, afișați în ordine ascendentă numerele din șir care au un număr impar de diagonale.*

Date de intrare	Date de ieșire
9 6 19 14 6 16 15 7 17 18 9	Cu 7 laturi are 14 diagonale 6 9 14 17 18

MODELE DE APLICAȚII IMPLEMENTÂND CONCEPTELE PROGRAMĂRII VIZUALE

1. Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: o componentă de tip Buton, o componentă de tip ComboBox, 5 componente de tip Panel, o componentă de tip MemoText și 8 componente Label.

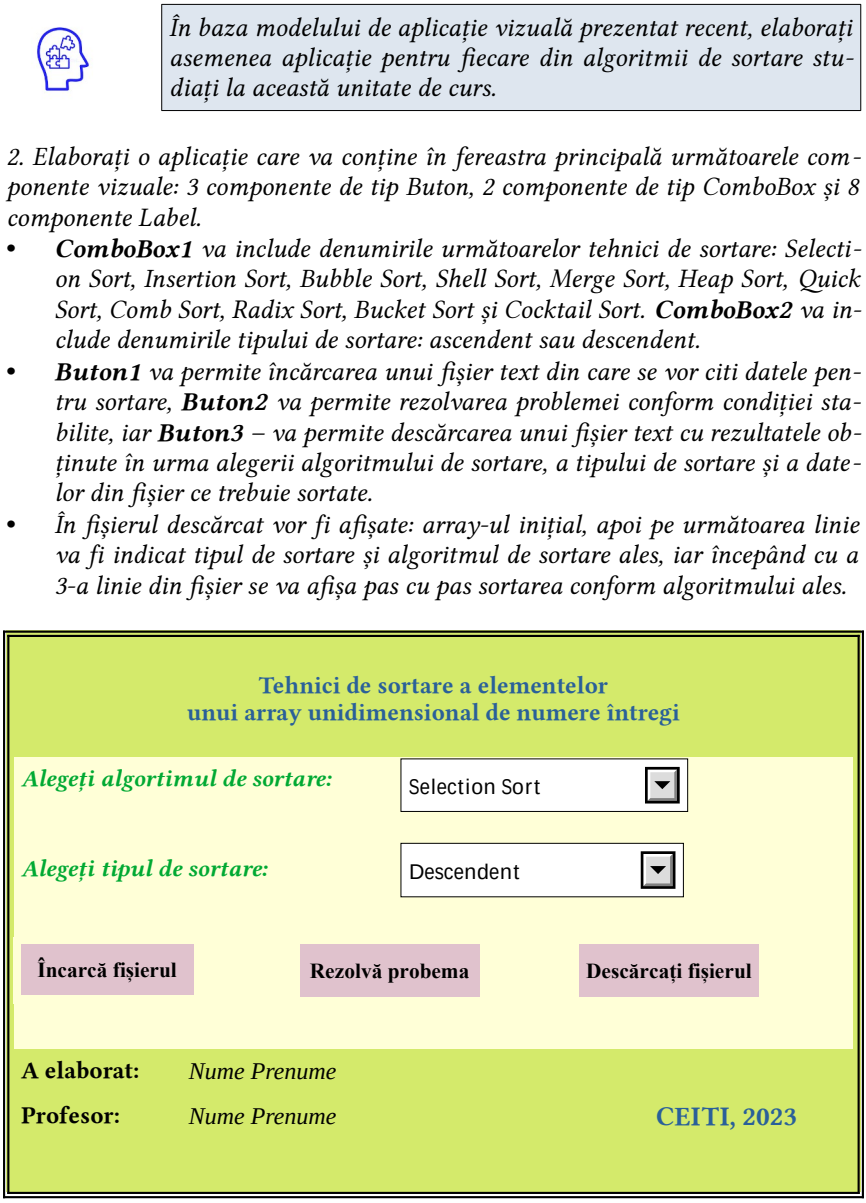
- ***ComboBox** va include denumirile tipului de sortare: ascendent sau descendent.*
- ***Buton** va permite rezolvarea problemei conform algoritmului de sortare și va afișa în componeta **MemoText** sortarea pas cu pas a datelor ce au fost introduse de utilizator pentru fiecare componentă de tip **Panel**, toate cele 5 componente **Panel** vor forma un array unidimensional de numere întregi.*



În baza modelului de aplicație vizuală prezentat recent, elaborați asemenea aplicație pentru fiecare din algoritmii de sortare studiați la această unitate de curs.

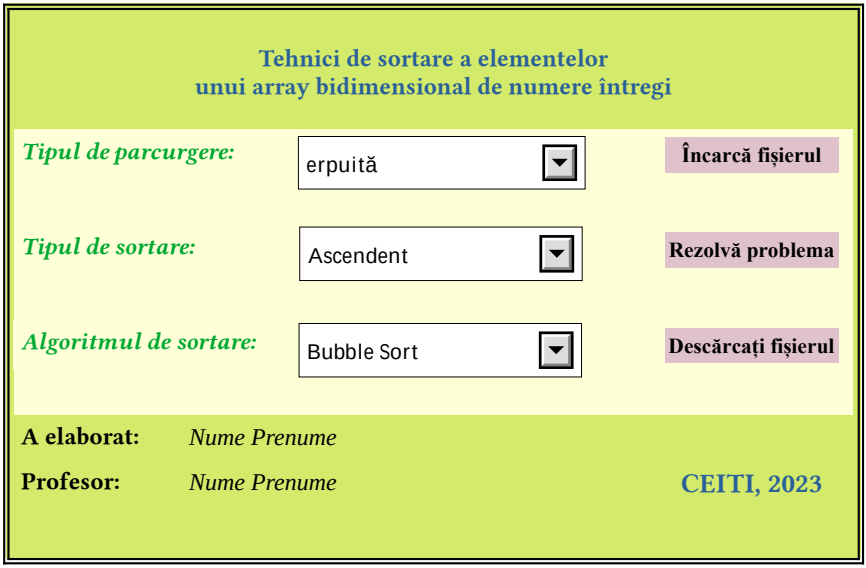
2. Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: 3 componente de tip Buton, 2 componente de tip ComboBox și 8 componente Label.

- ***ComboBox1** va include denumirile următoarelor tehnici de sortare: Selection Sort, Insertion Sort, Bubble Sort, Shell Sort, Merge Sort, Heap Sort, Quick Sort, Comb Sort, Radix Sort, Bucket Sort și Cocktail Sort. **ComboBox2** va include denumirile tipului de sortare: ascendent sau descendent.*
- ***Buton1** va permite încărcarea unui fișier text din care se vor citi datele pentru sortare, **Buton2** va permite rezolvarea problemei conform condiției stabilite, iar **Buton3** – va permite descărcarea unui fișier text cu rezultatele obținute în urma alegerii algoritmului de sortare, a tipului de sortare și a datelor din fișier ce trebuie sortate.*
- *În fișierul descărcat vor fi afișate: array-ul inițial, apoi pe următoarea linie va fi indicat tipul de sortare și algoritmul de sortare ales, iar începând cu a 3-a linie din fișier se va afișa pas cu pas sortarea conform algoritmului ales.*



3. *Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: 3 componente de tip Buton, 3 componente de tip ComboBox și 9 componente Label.*

- ***ComboBox1** va include denumirea unui tip de parcurgere a elementelor: Șerpuită, Spirală sau Diagonală. **ComboBox2** va include denumiriea unui tip de sortare: ascendent sau descendent. **ComboBox3** va include denumirea unei tehnici de sortare studiată: Selection Sort, Insertion Sort, Bubble Sort, Shell Sort, Merge Sort, Heap Sort, Quick Sort, Comb Sort, Radix Sort, Bucket Sort și Cocktail Sort.*
- ***Buton1** va permite încărcarea unui fișier text din care se vor citi datele pentru sortare, adică un array bidimensional pătratic de numere întregi, **Buton2** va permite rezolvarea problemei conform condițiilor stabilite, iar **Buton3** va permite descărcarea unui fișier text cu rezultatele obținute în urma alegerii tipului de parcurgere, tipului de sortare, algoritmului de sortare și încărcării datelor din fișierul exter.*
- *În fișierul descărcat vor fi afișate: array-ul bidimesnsional inițial, apoi pe următoarele 8 linii vor fi afișate denumirile celor 8 modalități de parcurgere conform tipului de parcurgere și elementele corespunzătoare, iar pe următoarele 8 rânduri, aceste 8 modalități vor fi afișate cu elementele sortate.*



Capitolul II

ALGORITMI DE CĂUTARE

De ce avem nevoie de algoritmi de căutare?

Adesea trebuie să găsim un anumit element de date între multe sute, mii, milioane sau mai multe. De exemplu, este posibil să trebuiască să găsiți numărul de telefon al cuiva pe telefonul dvs. sau adresa unei anumite companii dintr-o anumită țară. Acesta este motivul pentru care algoritmii de căutare sunt importanți. Fără ei, ar trebui să vă uitați la fiecare element de date - fiecare număr de telefon sau adresă de afaceri - individual, pentru a vedea dacă este ceea ce căutați. Într-o mulțime mare de date, va dura mult timp pentru a face acest lucru. În schimb, un algoritm de căutare poate fi folosit pentru a găsi elementele de date pe care le căutați.

Care este aplicabilitatea algoritmilor de căutare?

1. *Probleme în optimizare combinatorie, cum ar fi:*
   - *Problema de rutare a vehiculului, o formă a celei mai scurte probleme de cale (drum minim);*
   - *Problema rucsacului: având în vedere un set de articole, fiecare cu o greutate și o valoare, determinați numărul fiecărui articol care trebuie inclus într-o colecție, astfel încât greutatea totală să fie mai mică sau egală cu o limită dată și valoarea totală să fie la fel de mare pe cât posibil;*
   - *Problema de programare a asistentei medicale.*
2. *Probleme de satisfacție a constrângerilor, cum ar fi:*
   - *Problema colorării hărții;*
   - *Completarea unui sudoku sau cuvinte încrucișate.*
3. *În teoria jocurilor și, în special, în teoria combinatorie a jocurilor, alegerea celei mai bune mișcări de făcut (cum ar fi cu algoritmul minmax);*
4. *Găsirea unei combinații sau a unei parole din întregul set de posibilități;*
5. *Factorizarea unui număr întreg (o problemă importantă în criptografie);*
6. *Optimizarea unui proces industrial, cum ar fi o reacție chimică, prin schimbarea parametrilor procesului (cum ar fi temperatura, presiunea și pH-ul);*
7. *Preluarea unei înregistrări dintr-o bază de date etc.*

2.1 Căutare liniară. Căutare binară.

Obiective

- *Descrierea algoritmilor de căutare liniară și binară.*
- *Prezentarea situațiilor de aplicare a algoritmilor de căutare.*
- *Elaborarea algoritmilor de căutare conform specificațiilor propuse.*
- *Implementarea algoritmilor de căutare în limbajul de programare.*

*Căutarea și sortarea sunt două concepte principale în programarea computerului. Algoritmul **Linear Search** este foarte complex dacă doriți să căutați un număr dintr-o listă. Unul câte unul, fiecare element al listei este preluat și comparat cu elementul care trebuie căutat. Poate fi cel mai rău caz, dacă ultimul număr din listă este numărul care trebuie căutat. Dacă vorbim despre eficiență, eficiența este de câte ori programul trebuie să ruleze pentru a găsi numărul. Dacă găsim numărul pe care îl căutăm în prima poziție, atunci trebuie făcută o singură comparație și lucrurile sunt sortate, în caz contrar, se efectuiază multe comparații, astfel memoria este irosită. În medie, numărul de comparații va fi (n+1/2). Iar cea mai slabă eficiență a acestei tehnici este O (n), înseamnă ordinea de execuție.*

Exemplu pas cu pas:
Fie avem următorul array unidimensional de numere sortat ascendent:

i	0	1	2	3	4	**5**	6	7	8	9
A[i]	10	11	12	13	14	**15**	16	17	18	19

Să analizăm implementarea algoritmului:

- *Fie elementul ce trebuie căutat este K=15.*
- *Comparăm elementele A[i] cu K, până când A[i]=K, de unde obținem poziția i a elementului K în array.*
- *A[0]!=K, adică 10 !=15, trecem la elementul următor. A[1]!=K, adică 11!=15, trecem la elementul următor. A[2]!=K, adică 12!=15, trecem la elementul următor. A[3] !=K, adică 13!=15, trecem la elementul următor. A[4] !=K, adică 14 !=15, trecem la elementul următor. A[5] =K, adică 15=15, prin urmare elementul K=15 se află în array la indicele 5.*

*Algoritmul **Binary Search** este una dintre tehnicile de căutare utilizate pe scară largă. Poate fi folosit pentru sortarea array-urilor. Această tehnică de căutare urmează strategia de divizare și cucerire. Spațiul de căutare se reduce întotdeauna la jumătate în fiecare iterație. Algoritmul Binary Search este o tehnică foarte eficientă pentru căutare. Diferența dintre Linear Search și Binary Search este că în Linear Search fiecare element este verificat și comparat și apoi este sortat, în timp ce în Binary Search o listă care urmează să fie sortată este împărțită în două părți și apoi sortată.*

Exemplu pas cu pas:
Fie avem următorul array unidimensional de numere sortat ascendent:

i	0	1	2	3	4	**5**	6	7	8	9
A[i]	10	11	12	13	14	**15**	16	17	18	19

Să analizăm implementarea algoritmului:

- *Fie elementul ce trebuie căutat este K=15.*
- *Fie următoarele variabile: început=0, sfârșit=9 și mijloc=(început+ sfârșit)/2= (0+9)/2=4.5, deci mijloc=14. Deoarece K>mijloc, adică 15>14, vom căuta elementul în partea dreaptă a array-ului.*
- *Căutăm acum noua valoare a lui mijloc: început=5, sfârșit=9 și mijloc=(început+ sfârșit)/2= (5+9)/2=7, deci mijloc=17. Deoarece K<mijloc, adică 15<17, vom căuta elementul în partea stângă a array-ului.*

i	0	1	2	3	4	**5**	6	7	8	9
A[i]	10	11	12	13	14	**15**	16	17	18	19

- *Căutăm acum noua valoare a lui mijloc: început=5, sfârșit=6 și mijloc=(început+ sfârșit)/2= (5+6)/2=5.5, deci mijloc=15. Deoarece K=mijloc, adică 15=15, putem spune acum că am găsit elementul căutat al array-ului la poziția 5.*

i	0	1	2	3	4	**5**	6	7	8	9
A[i]	10	11	12	13	14	**15**	16	17	18	19

Implementarea recursivă a algoritmilor în limbajul C++

```
#include <iostream>
using namespace std;
// Variabilele globale
int V[] = {10,11,12,13,14,15,16,17,18,19},cheia=15;
int i,n=sizeof(V)/sizeof(V[0]);
// Subprogramul recursiv al algoritmului de căutare liniara
int LinearSearch(int X[], int n, int K){
    for (int i=0; i<n; i++){
        if (X[i]==K) return i;
    }
    return -1;
}
// Subprogramul recursiv al algoritmului de căutare binara
int BinarySearch(int X[],int stanga,int dreapta,int K){
    if (dreapta>=stanga){
        int mijloc =stanga+(dreapta-stanga) / 2;
        if (X[mijloc]==K) return mijloc;
        if (X[mijloc]>K) return BinarySearch(X,stanga,mijloc-1,K);
        return BinarySearch(X,mijloc+1,dreapta,K);
    }
    return -1;
}
// Subprogramul de afisare a array-ului unidimensional
void AfisareArray(){
    for (i=0;i<n;i++){
        cout<<"    "<<i;
    } cout<<endl;
    for (i=0;i<n;i++){
        cout<<"   "<<V[i];
    } cout<<endl;
}
// Programul principal
int main(){
    cout<<"Elementele array-ului initial sunt:"<<endl; AfisareArray();
    int r1=LinearSearch(V,n-1,cheia);
    (r1==-1)?cout<<"\nElementul nu este prezent in array!"
                    :cout<<"\nLinear Search:\n\tElementul "<<cheia<<"
se afla la indexul: "<<r1;
    int r2=BinarySearch(V,0,n-1,cheia);
    (r2==-1)?cout<<"\nElementul nu este prezent in array!"
                    :cout<<"\nBinary Search:\n\tElementul "<<cheia<<"
se afla la indexul: "<<r2;
}
```

Soluția obținută în urma execuției codului C++ este:

```
Elementele array-ului initial sunt:
    0    1    2    3    4    5    6    7    8    9
   10   11   12   13   14   15   16   17   18   19

Linear Search:
        Elementul 15 se afla la indexul: 5
Binary Search:
        Elementul 15 se afla la indexul: 5
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă a datelor în cazul când acestea vor fi litere majuscule. Aplicați ambii algoritmi de căuatare în același program pentru a căuta poziția unei litere introduse de la tastatură.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea descendentă a datelor în cazul când acestea vor fi litere minuscule, citite dintr-un fișier extern separate printr-un spațiu. Aplicați ambii algoritmi de căuatare în același program pentru a căuta poziția unei litere introduse de la tastatură.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei liste liniare simplu înlănțuite. Aplicați un algoritm de sortare și determinați poziția unui număr introdus de la tastatură.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei liste liniare dublu înlănțuite. Aplicați un algoritm de sortare și determinați poziția unei litere introdusă de la tastatură.

2.2 Căutare prin salt. Căutare prin interpolare.

Obiective

- *Descrierea algoritmilor de căutare prin salt și prin interpolare.*
- *Prezentarea situațiilor de aplicare a algoritmilor de căutare.*
- *Elaborarea algoritmilor de căutare conform specificațiilor propuse.*
- *Implementarea algoritmilor de căutare în limbajul de programare.*

*Algoritmul **Jump Search** este un algoritm relativ nou pentru căutarea unui element dintr-un array sortat. Ideea fundamentală din spatele acestei tehnici de căutare este de a căuta un număr mai mic de elemente în comparație cu algoritmul Linear Search (care scanează fiecare element din array pentru a verifica dacă se potrivește cu elementul căutat sau nu). Acest lucru se poate face prin omiterea unui număr fix de elemente din array sau sărind înainte cu un număr fix de pași în fiecare iterație.*

Exemplu pas cu pas:
Fie avem următorul array unidimensional de numere sortat ascendent:

i	0	1	2	3	4	**5**	6	7	8	9
A[i]	10	11	12	13	14	**15**	16	17	18	19

Să analizăm implementarea algoritmului:

- *Fie elementul ce trebuie căutat este K=15.*
- *Fie următoarele variabile:* $n=10, m=\sqrt{n}=\sqrt{10}=3.1622776$. *Deci valoarea întreagă a lui m este 3.*
- *Comparăm A[0] cu K, deoarece A[0]!=K și A[0]<K, trecem la următorul bloc. Comparăm A[2] cu K, deoarece A[2]!=K și A[2]<K, trecem la următorul bloc. Comparăm A[4] cu K, deoarece A[4]!=K și A[4]<K, trecem la următorul bloc. Comparăm A[6] cu K, deoarece A[6]!=K și A[6]>K, trecem la A[4] (începutul blocului curent) și aplicăm Linear Search.*
- *Comparăm A[4] cu K, deoarece A[4]!=K, trecem la A[5]. Comparăm A[5] cu K, deoarece A[5]==K, indexul 5 este tipărit ca locație validă și algoritmul se va termina.*

*Algoritmul **Interpolation Search** este o variantă îmbunătățită a Binary Search. Acest algoritm de căutare funcționează pe poziția de sondare a valorii solicitate. Pentru ca acest algoritm să funcționeze corect, colectarea datelor ar trebui să fie într-o formă sortată și distribuită în mod egal. Binary Search are un avantaj imens al complexității timpului față de Linear Search.*
Există cazuri în care locația datelor țintă poate fi cunoscută din timp. De exemplu, în cazul unei cărți de telefoane, să presupunem că dorim să căutăm numărul de telefon al persoanei Morar Alina. Aici, Linear Search și chiar Binary Search vor părea lente, deoarece putem sări direct în spațiul de memorie unde sunt stocate numele care încep de la litera „M".

Exemplu pas cu pas:
Fie avem următorul array unidimensional de numere sortat ascendent:

i	0	1	2	3	4	**5**	6	7	8	9
A[i]	10	11	12	13	14	**15**	16	17	18	19

Să analizăm implementarea algoritmului:
- *Fie elementul ce trebuie căutat este K=15 și următoarela formulă:*

$$poziția = început + \frac{sfârșit - început}{A[sfârșit] - A[început]} * (K - A[început]).$$

- *Aplicăm formula și obținem:*

$$poziția = 0 + \frac{9-0}{A[9]-A[0]} * (K - A[0]) = 0 + \frac{9}{19-10} * (15-10) = 1*5 = 5.$$

Fie avem următorul array unidimensional de numere sortat descendent:

i	0	1	2	3	**4**	5	6	7	8	9
A[i]	19	18	17	16	**15**	14	13	12	11	10

Să analizăm implementarea algoritmului:
- *Fie elementul ce trebuie căutat este K=15 și următoarela formulă:*

$$poziția = început + \frac{sfârșit - început}{A[început] - A[sfârșit]} * (K - A[sfârșit]).$$

- *Aplicăm formula și obținem:*

$$poziția = 0 + \frac{9-0}{A[0]-A[9]} * (K - A[9]) = 0 + \frac{9}{19-10} * (15-10) = 1*5 = 5.$$

- *Eliminăm ultimul indice și elementul său, aplicăm formula și obținem:*

$$poziția = 0 + \frac{8-0}{A[0]-A[8]} * (K - A[8]) = 0 + \frac{8}{19-11} * (15-11) = 1 * 4 = 4.$$

Implementarea recursivă a algoritmului în limbajul C++

```
#include <iostream>
#include<cmath>
using namespace std;
// Variabilele globale
int V[] = {10,11,12,13,14,15,16,17,18,19},cheia=15;
int i,n=sizeof(V)/sizeof(V[0]);
// Subprogramul recursiv al algoritmului de căutare prin salt
int JumpSearch(int X[],int n,int K){
    int i=0,m=sqrt(n); //initializam dimensiunea blocului de sarit
    while (X[min(m,n)-1]<K){
        i=m;m+=sqrt(n);
        if(i>=n) return -1;
    }
    //Linear Search in blocul curent
    while(X[i]<K){
        i++;
        if (i==min(m,n)) return -1;
    }
    if (X[i]==K) return i;
    return -1;
}
// Subprogramul recursiv al algoritmului de căutare prin interpolare
int InterpolationSearch(int X[],int n,int K){
    int a=0,b=(n-1); // indicii de inceput si de sfarsit
    while (a<=b && K>=X[a] && K<=X[b]){
        if (a==b){
            if (X[a] == K) return a;
            return -1;
        }
        int p=a+(((double)(b-a)/(X[b]-X[a]))*(K-X[a]));
        if (X[p]==K) return p;
        if (X[p]<K) a=p+1;
        else b=p-1;
    }
    return -1;
}
// Subprogramul de afisare a array-ului unidimensional
void AfisareArray(){
    for (i=0;i<n;i++){
        cout<<"    "<<i;
```

```
    } cout<<endl;
    for (i=0;i<n;i++){
        cout<<"   "<<V[i];
    } cout<<endl;
}
// Programul principal
int main(){
    cout<<"Elementele array-ului initial sunt:"<<endl; AfisareArray();
    int r1=JumpSearch(V,n,cheia);
    (r1==-1)?cout<<"\nElementul nu este prezent in array!"
                    :cout<<"\nJump Search:\n\tElementul "<<cheia<<" se
afla la indexul: "<<r1;
    int r2=InterpolationSearch(V,n,cheia);
    (r2==-1)?cout<<"\nElementul nu este prezent in array!"
                    :cout<<"\nInterpolation Search:\n\tElementul
"<<cheia<<" se afla la indexul: "<<r2;
}
```

Soluția obținută în urma execuției codului C++ este:

```
Elementele array-ului initial sunt:
    0    1    2    3    4    5    6    7    8    9
   10   11   12   13   14   15   16   17   18   19

Jump Search:
        Elementul 15 se afla la indexul: 5
Interpolation Search:
        Elementul 15 se afla la indexul: 5
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă a datelor în cazul când acestea vor fi litere majuscule. Aplicați ambii algoritmi de căuatare în același program pentru a căuta poziția unei litere introduse de la tastatură.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea descendentă a datelor în cazul când acestea vor fi litere minuscule, citite dintr-un fișier extern separate printr-un spațiu. Aplicați ambii algoritmi de căuatare în același program pentru a căuta poziția unei litere introduse de la tastatură.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei liste circulare simplu înlănțuite. Aplicați un algoritm de sortare și determinați poziția unui număr introdus de la tastatură.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei liste circulare dublu înlănțuite. Aplicați un algoritm de sortare și determinați poziția unei litere introdusă de la tastatură.

2.3 Căutare exponenţială. Căutare Fibonacci.

Obiective

- *Descrierea algoritmilor de căutare exponențială și Fibonacci.*
- *PPrezentarea situațiilor de aplicare a algoritmilor de căutare.*
- *Elaborarea algoritmilor de căutare conform specificațiilor propuse.*
- *Implementarea algoritmilor de căutare în limbajul de programare.*

*Algoritmul **Exponential Search** a fost creat pentru căutarea elementelor într-un array unidimensional de dimensiuni uriașe. Este un proces în doi pași. Mai întâi, algoritmul încearcă să găsească intervalul (L, R) în care este prezent elementul țintă și apoi folosește Binary Search în acest interval pentru a găsi locația exactă a țintei. Se numește Exponential Search, deoarece găsește elementul de menținere a intervalului căutând primul exponent k pentru care element la index pow (2, k) este mai mare decât ținta. Deși numele său este Exponential Search, complexitatea în timp a acestui algoritm este logaritmică. Este foarte util atunci când array-urile unidimensional sunt de dimensiuni infinite și converg către o soluție mult mai rapidă decât Binary Search.*

Exemplu pas cu pas:
Fie avem următorul array unidimensional de numere sortat ascendent:

i	0	1	2	3	4	**5**	6	7	8	9	10	11
A[i]	10	11	12	13	14	**15**	16	17	18	19	20	21

Să analizăm implementarea algoritmului:

- *Fie elementul ce trebuie căutat este K=15.*
- *Fie următoarele variabile: inițializăm variabila E cu valoarea 1.*
- *Deoarece A[0]=10 < K, atunci E va avea valoarea 2, deoarece A[1]=11 < K, atunci E va avea valoarea 4, A[2]=12 < K, atunci E va avea valoarea 8, A[3]=13 < K, atunci E va avea valoarea 16.*
- *Deoarece E>n, adică 16>12, apelăm la algoritmul Binry Search în intervalul E/2, adică de la 8 până la minim(E, n-1), adică minim(16, 11) = 11.*

i	0	1	2	3	4	**5**	6	7	8	9	10	11
A[i]	10	11	12	13	14	**15**	16	17	18	19	20	21

- *Fie variabile: început=E/2=8, sfârșit=minim(E,n-1)=11 și mijloc=(8+11)/2= 9.5, deci mijloc=19. Deoarece K<mijloc, adică 15<19, vom căuta elementul în partea stângă a array-ului unidimensional.*

i	0	1	2	3	4	**5**	6	7	8	9	10	11
A[i]	10	11	12	13	14	**15**	16	17	18	19	20	21

- *Căutăm acum noua valoare a lui mijloc: început=0, sfârșit=8 și mijloc= (0+8)/2=4, deci mijloc=14. Deoarece K>mijloc, adică 15>14, vom căuta elementul în partea dreaptă a array-ului unidimensional.*

i	0	1	2	3	4	**5**	6	7	8	9	10	11
A[i]	10	11	12	13	14	**15**	16	17	18	19	20	21

- *Căutăm acum noua valoare a lui mijloc: început=5, sfârșit=8 și mijloc= (5+8)/2=6.5, deci mijloc=16. Deoarece K<mijloc, adică 15<16, vom căuta elementul în partea stângă a array-ului unidimensional.*
- *Căutăm acum noua valoare a lui mijloc: început=5, sfârșit=5 și mijloc= (5+5)/2=5, deci mijloc=15. Deoarece K=mijloc, adică 15=15, putem spune acum că am găsit elementul căutat al array-ului la poziția 5.*

*Algoritmul **Fibonacci Search** este un algoritm de căutare eficient bazat pe principiul divizare și cucerire care poate găsi un element în array-ul unidimensional sortat cu ajutorul seriei Fibonacci în complexitatea timpului O (log N). Aceasta se bazează pe seria Fibonacci, care este o secvență infinită de numere care denotă un model care este capturat de următoarea ecuație: F(n+1)=F(n)+F(n-1), unde F(i) este numărul i din seria Fibonacci în care F(0) și F(1) sunt definite ca 0 și respectiv 1. Primele câteva numere Fibonacci sunt: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...*

Alte căutări, cum ar fi Binary Search, funcționează și pentru principiul similar privind împărțirea spațiului de căutare într-un spațiu mai mic, dar ceea ce face Fibonacci Search diferit este că împarte array-ul unidimensional în părți inegale și operațiile implicate în această căutare sunt doar adunarea și scăderea,

ceea ce înseamnă operații aritmetice ușoare și, prin urmare, reducerea sarcinii de lucru a mașinii de calcul.

Exemplu pas cu pas:
Fie avem următorul array unidimensional de numere sortat ascendent:

i	0	1	2	3	4	5	**6**	7	8	9	10	11
A[i]	10	11	12	13	14	15	**16**	17	18	19	20	21

Să analizăm implementarea algoritmului:
- *Fie elementul ce trebuie căutat este K=16.*
- *Inițializarea datelor: n=12, F_k=13 (un număr Fibonacci mai mare sau egal ca n) și P=8 (un număr Fibonacci precedent lui F_k), start=0 și finish=n-1=11.*
- *Verificăm: K=A[P+start], adică K=A[8] sau 16=18, deci avem fals, verificăm dacă 16>18, deci avem fals, verificăm dacă 16<18, deci avem adevărat. Valoarea start=0, finish=start+P-1=0+7=7, n=finish-start+1=7-0+1=8.*

i	0	1	2	3	4	5	**6**	7	8	9	10	11
A[i]	10	11	12	13	14	15	**16**	17	18	19	20	21

- *Acum start=0, finish=7, n=8 și P=5. Verificăm: K=A[P+start], adică K=A[5] sau 16=15, deci avem fals, verificăm dacă 16>15, deci avem adevărat, verificăm dacă 16<15, deci avem fals. Valoarea start=0+P+1=6, finish=7, n=finish-start+1=7-6+1=2.*

i	0	1	2	3	4	5	**6**	7	8	9	10	11
A[i]	10	11	12	13	14	15	**16**	17	18	19	20	21

- *Acum start=6, finish=7, n=2 și P=1 (P<=n). Verificăm: K=A[P+start], adică K=A[7] sau 16=17, deci avem fals, verificăm dacă 16>17, deci avem fals, verificăm dacă 16<17, deci avem adevărat. Valoarea start=6, finish=start+P-1=6+1-1=6, n=finish-start+1=6-6+1=1.*

i	0	1	2	3	4	5	**6**	7	8	9	10	11
A[i]	10	11	12	13	14	15	**16**	17	18	19	20	21

- *Acum start=6, finish=6, n=1 și P=0 (P<=n). Verificăm: K=A[P+start], adică K=A[6] sau 16=16, deci avem adevărat. Valoarea K=16 a fost găsită la indicele 6.*

Implementarea recursivă a algoritmului în limbajul C++

```
#include <iostream>
using namespace std;
// Variabilele globale
int V[] = {10,11,12,13,14,15,16,17,18,19},cheia=13;
int i,n=sizeof(V)/sizeof(V[0]);
// Subprogramul recursiv al algoritmului de căutare binara
int BinarySearch(int X[],int stanga,int dreapta,int K){
    if (dreapta>=stanga){
        int mijloc=stanga+(dreapta-stanga)/2;
        if (X[mijloc]==K) return mijloc;
        if (X[mijloc]>K) return BinarySearch(X,stanga,mijloc-1,K);
        return BinarySearch(X,mijloc+1,dreapta,K);
    }
    return -1;
}
// Subprogramul recursiv al algoritmului de căutare exponentiala
int ExponentialSearch(int X[],int n,int K){
    if (X[0]==K) return 0;
    int i=1;
    while (i<n && X[i]<=K) i=i*2;
    return BinarySearch(X,i/2,min(i,n-1),K);
}
// Subprogramul recursiv al algoritmului de căutare Fibonacci
int FibonacciSearch(int X[], int n, int K){
    int fbM2=0,fbM1=1,fbM=fbM2+fbM1,poz=-1;
    while (fbM<n){
        fbM2=fbM1; fbM1=fbM; fbM=fbM2+fbM1;
    }
    while (fbM>1){
        int i=min(poz+fbM2,n-1);
        if (X[i]<K){
            fbM=fbM1; fbM1=fbM2; fbM2=fbM-fbM1; poz=i;
        }
        else if (X[i]>K){
            fbM=fbM2; fbM1=fbM1-fbM2; fbM2=fbM-fbM1;
        }
        else return i;
    }
    if (fbM1 && X[poz+1]==K) return poz+1;
    return -1;
```

```
}
// Subprogramul de afisare a array-ului unidimensional
void AfisareArray(){
    for (i=0;i<n;i++){
        cout<<"    "<<i;
    } cout<<endl;
    for (i=0;i<n;i++){
        cout<<"   "<<V[i];
    } cout<<endl;
}
// Programul principal
int main(){
    cout<<"Elementele array-ului initial sunt:"<<endl; AfisareArray();
    int r1=ExponentialSearch(V,n-1,cheia);
    (r1==-1)?cout<<"\nElementul nu este prezent in array!"
                    :cout<<"\nExponential Search:\n\tElementul
"<<cheia<<" se afla la indexul: "<<r1;
    int r2=FibonacciSearch(V,n-1,cheia);
    (r1==-1)?cout<<"\nElementul nu este prezent in array!"
                    :cout<<"\nFibonacci Search:\n\tElementul
"<<cheia<<" se afla la indexul: "<<r1;

}
```

Soluția obținută în urma execuției codului C++ este:

```
Elementele array-ului initial sunt:
    0    1    2    3    4    5    6    7    8    9
   10   11   12   13   14   15   16   17   18   19

Exponential Search:
        Elementul 13 se afla la indexul: 3
Fibonacci Search:
        Elementul 13 se afla la indexul: 3
```

Nivelul 1

1. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă a datelor în cazul când acestea vor fi litere majuscule. Aplicați ambii algoritmi de căuatare în același program pentru a căuta poziția unei litere introduse de la tastatură.
2. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea descendentă a datelor în cazul când acestea vor fi litere minuscule, citite dintr-un fișier extern separate printr-un spațiu. Aplicați ambii algoritmi de căuatare în același program pentru a căuta poziția unei litere introduse de la tastatură.

Nivelul 2

3. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea ascendentă a datelor în cazul când aceste date vor fi numere de 2 cifre generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei stive. Aplicați un algoritm de sortare și determinați poziția unui număr introdus de la tastatură.
4. Elaborați un program în limbajul C/C++ care va prezenta câte un subprogram pentru sortarea descendentă a datelor în cazul când aceste date vor fi litere majuscule generate într-un fișier extern. Din fișierul extern datele vor fi citite sub forma unei cozi. Aplicați un algoritm de sortare și determinați poziția unei litere introdusă de la tastatură.

2.4 Recapitulare generală. Model de testare 1.

1 *Fiecare elev va sorta descendent elementele array-ului de mai jos conform unui algoritm de sortare studiat recent (Shell Sort, Merge Sort, Heap Sort, Quick Sort, Bucket Sort, Radix Sort, Cocktail Sort, Counting Sort și Comb Sort). După ce au fost sortate elementele array-ului, se va determina poziția elementului K=69 prin metoda Linear Search și Binary Search.*

26	34	77	20	69	83	36	62	25	75

2 *Fiecare elev va sorta ascendent elementele array-ului de mai jos conform unui algoritm de sortare studiat recent (Shell Sort, Merge Sort, Heap Sort, Quick Sort, Bucket Sort, Radix Sort, Cocktail Sort, Counting Sort și Comb Sort). După ce au fost sortate elementele array-ului, se va determina poziția elementului K=69 prin metoda Jump Search și Interpolation Search.*

50	25	31	20	45	69	75	86	85	88

3 *Fiecare elev va sorta descendent elementele array-ului de mai jos conform unui algoritm de sortare studiat recent (Shell Sort, Merge Sort, Heap Sort, Quick Sort, Bucket Sort, Radix Sort, Cocktail Sort, Counting Sort și Comb Sort). După ce au fost sortate elementele array-ului, se va determina poziția elementului K=' N ' prin metoda Fibonacci Search și Exponential Search.*

Z	X	C	V	B	N	M	L	K	J

4 *Fiecare elev va sorta descendent elementele array-ului de mai jos conform unui algoritm de sortare studiat recent (Shell Sort, Merge Sort, Heap Sort, Quick Sort, Bucket Sort, Radix Sort, Cocktail Sort, Counting Sort și Comb Sort). După ce au fost sortate elementele array-ului, se va determina poziția elementului K=' N ' prin metoda Fibonacci Search și Jump Search.*

P	O	I	U	J	K	L	M	N	B

5 *Implementați toți algoritmii de căutare studiați: Lineari, Binary, Jump, Interpolation, Fibonacci și Exponential pentru un array de numere întregi utilizând calculul tabelar (MS Excel sau LibreOffice Calc).*

1. Unități de conținut

Proba de autoevaluare va conține itemi creați în baza următoarelor unități de conținut conform unității de curs:

1. *Algoritmii de sortare prin interclasare și prin micșorarea incrementului;*
2. *Algoritmii de sortare rapidă și de sortare cu ansamble;*
3. *Algoritmii de sortare prin distribuție;*
4. *Algoritmii de sortare cu bule modificați;*
5. *Algoritmii de căutare liniară și căutare binară;*
6. *Algoritmii de căutare prin salt și căutare prin interpolare;*
7. *Algoritmii de căutare exponențială și căutare Fibonacci.*

2. Barem de notare

Nota	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	28-27	26-25	24-22	21-18	17-13	12-9	8-6	5-4	3-2

3. Barem orientativ de evaluare a corectitudinii rezultatelor

Item	***Explicații succinte***	***Punctaj***
1	*Pentru fiecare răspuns corect*	2*1p
2	*Pentru sortarea elementelor array-ului de numere și scrierea corectă a indicelui variabilei K*	2*1p
3	*Pentru sortarea elementelor array-ului de litere și scrierea corectă a indicelui variabilei K*	2*1p
4	*Pentru completarea fiecărei celule cu un răspuns corect*	6*1p
5	*Pentru crearea corectă a celor 4 subprograme cu pointeri* *Pentru apelarea corectă a celor 4 subprograme*	4*3p 4*1p
	TOTAL:	**28p**

Model de test de autoevaluare
Varianta I

<table>
<tr><th>Nr</th><th>Condiția itemului</th><th></th></tr>
<tr><td>1</td><td>Precizați valoarea de adevăr a următoarelor enunțuri:
<table>
<tr><td>Prima versiune pentru Shell Sort a fost publicată în 1959.</td><td>A / F</td></tr>
<tr><td>Interpolation Search nu este o optimizare pentru Binary.</td><td>A / F</td></tr>
</table></td><td>2p</td></tr>
<tr><td>2</td><td>Fie următorul array unidimensional de numere întregi:
<table>
<tr><td>34</td><td>29</td><td>32</td><td>39</td><td>42</td><td>49</td><td>23</td><td>93</td><td>24</td><td>94</td></tr>
</table>
Care va fi poziția variabilei K=49?</td><td>2p</td></tr>
<tr><td>3</td><td>Fie următorul array unidimensional de caractere:
<table>
<tr><td>X</td><td>E</td><td>D</td><td>L</td><td>T</td><td>K</td><td>Y</td><td>C</td><td>P</td><td>H</td></tr>
</table>
Care va fi poziția variabilei K='H' ?</td><td>2p</td></tr>
<tr><td>4</td><td>Efectuați o analiză comparativă a următoarelor două metode de sortare după complexitatea de timp. Completați următorul tabel:
<table>
<tr><th>Algoritmul</th><th>Timp eficient</th><th>Timp ineficient</th><th>Timp mediu</th></tr>
<tr><td>Shell Sort</td><td></td><td></td><td></td></tr>
<tr><td>Merge Sort</td><td></td><td></td><td></td></tr>
</table></td><td>6p</td></tr>
<tr><td>5</td><td>Generați un array unidimensional din literele majuscule. Aplicați algoritmul de sortare Shell Sort pentru a sorta Ascendent elementele arrayului de date. Apoi aplicați algoritmul de căutare Interpolation Search pentru a determina poziția elementului notat prin variabila X, X va fi numărul de ordine al lunii în care vați născut. Dacă vați născut în luna Mai, atunci veți determina poziția elementului al 5-lea din array-ul generat în cel obținut după sortare. Se vor realiza subprograme pentru generare, sortare, căutare și afișare. Toate subprogramele vor include variabile de tip pointer.</td><td>16p</td></tr>
</table>

Model de test de autoevaluare
Varianta II

<table>
<tr><th>Nr</th><th>Condiția itemului</th><th></th></tr>
<tr><td>1</td><td>Precizați valoarea de adevăr a următoarelor enunțuri:
<table>
<tr><td>Quick Sort a fost publicată în 1961, de C. A. R. Hoare.</td><td>A / F</td></tr>
<tr><td>Cazul mediu de comparații pentru Linear Search: (n+1)/3.</td><td>A / F</td></tr>
</table></td><td>2p</td></tr>
<tr><td>2</td><td>Fie următorul array unidimensional de numere întregi:
<table>
<tr><td>30</td><td>25</td><td>28</td><td>34</td><td>38</td><td>45</td><td>23</td><td>90</td><td>19</td><td>74</td></tr>
</table>
Care va fi poziția variabilei K=45?</td><td>2p</td></tr>
<tr><td>3</td><td>Fie următorul array unidimensional de caractere:
<table>
<tr><td>Y</td><td>H</td><td>F</td><td>L</td><td>R</td><td>K</td><td>Z</td><td>C</td><td>P</td><td>J</td></tr>
</table>
Care va fi poziția variabilei K='P' ?</td><td>2p</td></tr>
<tr><td>4</td><td>Efectuați o analiză comparativă a următoarelor două metode de sortare după complexitatea de timp. Completați următorul tabel:
<table>
<tr><th>Algoritmul</th><th>Timp eficient</th><th>Timp ineficient</th><th>Timp mediu</th></tr>
<tr><td>Heap Sort</td><td></td><td></td><td></td></tr>
<tr><td>Quick Sort</td><td></td><td></td><td></td></tr>
</table></td><td>6p</td></tr>
<tr><td>5</td><td>Generați un array unidimensional din literele minuscule. Aplicați algoritmul de sortare QuickSort pentru a sorta Descendent elementele arrayului de date. Apoi aplicați algoritmul de căutare Jump Search pentru a determina poziția elementului notat prin variabila X, X va fi numărul de ordine al lunii în care vați născut. Dacă vați născut în luna Iunie, atunci veți determina poziția elementului al 6-lea din array-ul generat în cel obținut după sortare. Se vor realiza subprograme pentru generare, sortare, căutare și afișare. Toate subprogramele vor include variabile de tip pointer.</td><td>16p</td></tr>
</table>

Algoritmi de căutare
Itemi recomandaţi pentru lecţiile practice 5-8

1. *Implementați algoritmii de căutare cu afișarea rezultatului pas cu pas (Linear, Binary, Jump, Interpolation, Fibonacci și Exponential). Șirurile de date vor fi citite dintr-un fișier extern, apoi sortate ascendent. Afișați poziția primului element din array după ce acesta va fi sortat.*

Nr.	Partea 1	Nr.	Partea 2
1	82, 59, 34, 55, 16, 85, 53, 86, 57, 62	**9**	70, 37, 76, 43, 24, 11, 34, 22, 19, 48
2	39, 10, 85, 28, 66, 70, 79, 32, 19, 15	**10**	76, 11, 67, 49, 14, 43, 70, 45, 20, 73
3	86, 29, 78, 53, 85, 16, 70, 64, 21, 82	**11**	14, 48, 64, 80, 38, 61, 90, 82, 52, 41
4	34, 16, 81, 73, 24, 87, 33, 30, 18, 40	**12**	99, 35, 39, 34, 54, 33, 17, 62, 13, 24
5	73, 90, 36, 60, 21, 75, 76, 33, 80, 51	**13**	66, 56, 24, 34, 43, 57, 51, 65, 44, 69
6	80, 74, 13, 16, 82, 39, 89, 20, 96, 85	**14**	88, 23, 66, 55, 85, 15, 35, 52, 17, 90
7	49, 63, 64, 54, 98, 14, 67, 55, 60, 58	**15**	12, 51, 42, 92, 85, 82, 49, 56, 33, 47
8	20, 33, 45, 82, 66, 15, 57, 47, 69, 24	**16**	44, 97, 19, 18, 47, 99, 38, 65, 28, 77

2. *Implementați algoritmii de căutare cu afișarea rezultatului pas cu pas (Linear, Binary, Jump, Interpolation, Fibonacci și Exponential). Șirurile de date vor fi citite dintr-un fișier extern, apoi sortate descendent. Afișați poziția ultimului element din array după ce acesta va fi sortat.*

Nr.	Partea 1	Nr.	Partea 2
1	h z t e q m k n e m s o t l e u i a h c	**9**	n e m s o t l e u i a h c h z t e q m k
2	c u u i t u c h x i k b r h n q z m j x	**10**	k b r h n q z m j x c u u i t u c h x i
3	m b k c s m d m x w y x i y r r x z a y	**11**	w y x i y r r x z a y m b k c s m d m x

Nr.	Partea 1	Nr.	Partea 2
4	c e d l z d y t z u o k n q e o t c k f	**12**	z u o k n q e o t c k f c e d l z d y t
5	e m s o t l e u i a h c h z t e q m k n	**13**	e a h c h z t e q m k n m s o t l e u i
6	h x i k b r h n q z m j x c u u i t u c	**14**	n q z m j x c u u i t u c h x i k b r h
7	y x i y r r x z a y m b k c s m d m x w	**15**	a y m b k c s m d m x w y x i y r r x z
8	n q e o t c k f c e d l z d y t z u o k	**16**	k f c e d l z d y t z u o k n q e o t c

2. *Elaborați un program în limbajul C++ care va citi dintr-un fișier extern o listă de prenume de băieți, apoi va determina lungimea fiecărui prenume și îl va păstra într-un array unidimensional de numere întregi pozitive. Sortați ascendent array-ul obținut conform unei metode de sortare studiate, apoi determinați poziția primului element ce reprezintă partea întreagă de la media aritmetică a lungimii cuvintelor din fișier. Se vor aplica algoritmii Binary Search, Jump Search și Interpolation Search.*

Date de intrare	Date de ieșire
9 Dumitru Igor Tudor Ion Dorian Petru Florin Sergiu Cornel	lungime: 7 4 5 3 6 5 6 6 6 ascendent: 3 4 5 5 6 6 6 6 7 Primul indice al lui media=5 în array-ul sortat este 2.

3. *Elaborați un program în limbajul C++ care va citi dintr-un fișier extern o listă de prenume de fete, apoi va determina numărul de vocale ale fiecărui prenume și îl va păstra într-un array unidimensional de numere întregi pozitive. Sortați descendent array-ul obținut conform unei metode de sortare studiate, apoi determinați poziția primului element ce reprezintă partea întreagă de la media aritmetică a numărului de vocale ele cuvintelor. Se vor aplica algoritmii Linear Search, Fibonacci Search și Exponential Search.*

Date de intrare	Date de ieșire
8 Daniela Rebeca Eva Ramona Claudia Florica Sabina Lucia	Vocale: 4 3 2 3 4 3 3 3 descedent: 4 4 3 3 3 3 3 2 Primul indice al lui media=3 în array-ul sortat este 2.

4. *Elaborați în limbajul C++ un modul care va implementa toate metodele de căutare studiate. Creați următoarele subprograme asupra datelor unui array unidimensional de numere întregi:*
   a) *citirea;*
   b) *afișarea;*
   c) *sortarea ascendentă;*
   d) *sortarea descendentă;*
   e) *determinarea poziției elementului minim (maxim) par;*
   f) *determinarea poziției elementului maxim (minim) impar;*
   g) *determinarea poziției elementului maxim (minim) pătrat perfect;*
   h) *determinarea poziției elementului minim (maxim) cub perfect;*
   i) *determinarea poziției elementului maxim (minim) din seria Fibonacci.*

5. *Elaborați în limbajul C++ un modul care va implementa toate metodele de căutare studiate. Dar să presupunem că rotim o listă sortată în ordine crescătoare la un pivot necunoscut dumneavoastră în prealabil. De exemplu, 1 2 3 4 5 ar putea deveni 3 4 5 1 2. Determinați o modalitate de a găsi un element în lista rotită în timpul O (log n).*

Date de intrare	Date de ieșire
9 19 15 16 17 18 19 10 11 12 13	Elementul 19 are indicele 4

6. *Elaborați în limbajul C++ un modul care va implementa toate metodele de căutare studiate. Fie două mulțimi sortate A și B de dimensiunea n fiecare. Găsi mediana mulțimii obținute după îmbinarea celor două mulțimi de mai sus (adică mulțimea de lungime 2n). Complexitatea ar putea fi O (log (n)).*

Date de intrare	Date de ieșire
5 10 12 15 26 38 20 24 27 30 45	Mulțimea mare este: 10 12 15 20 24 26 27 30 38 45 Mediana=(24+26)/2=25

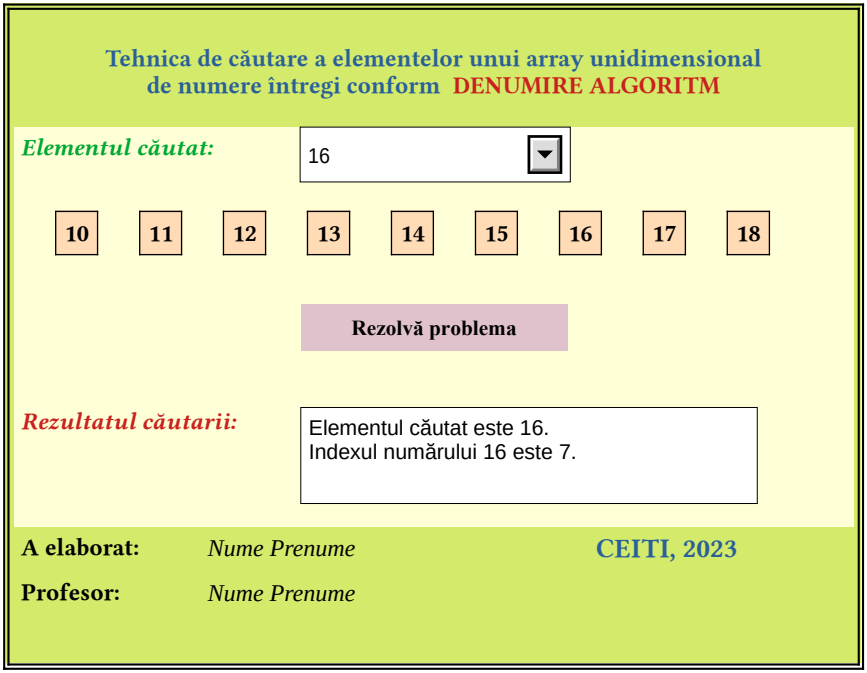
7. *Elaborați în limbajul C++ un modul care va implementa toate metodele de căutare studiate. Vi se oferă o listă de n numere întregi, elementele nu se pot repeta în listă. Găsiți primul și al doilea element minim al listei.*

Date de intrare	Date de ieșire
7 22 34 13 10 38 18 19	Elementele cautate sunt: 10 13

ELABORAREA UNEI APLICAȚII COMPLEXE IMPLEMENTÂND CONCEPTELE PROGRAMĂRII VIZUALE

1. Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: o componentă de tip Buton, o componentă de tip ComboBox, 9 componente de tip Panel, o componentă de tip MemoText și 8 componente Label.

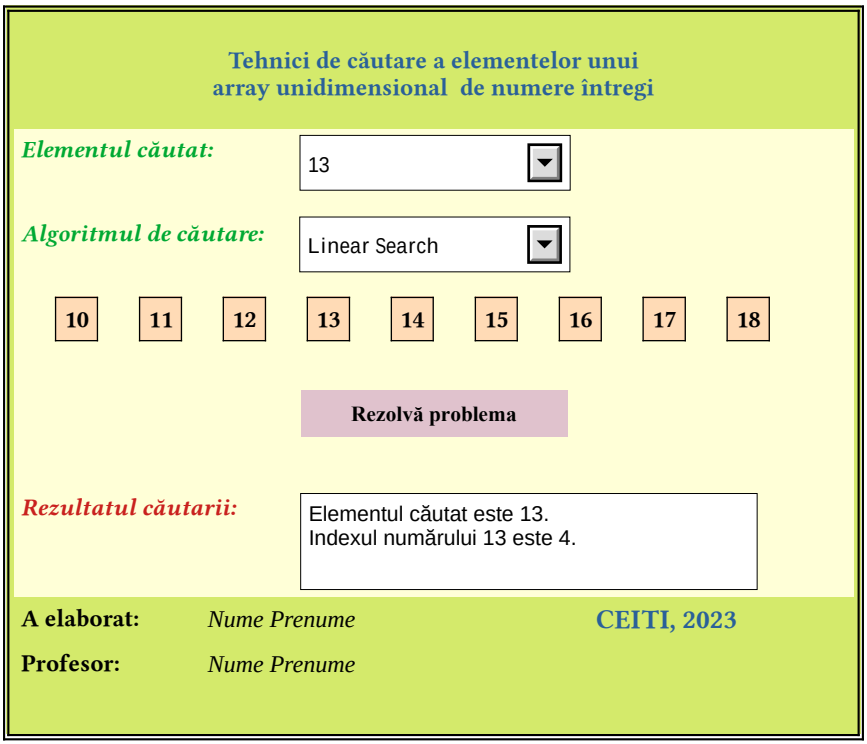
- ***ComboBox** va include elementul ce trebuie căutat (de la 1 la 1000).*
- ***Buton** va permite rezolvarea problemei conform algoritmului de căutare și va afișa în componeta **MemoText** datele corespunzătoare (vezi modelul).*



În baza modelului de aplicație vizuală prezentat recent, elaborați asemenea aplicație pentru fiecare din algoritmii de căutare studiați la această unitate de curs.

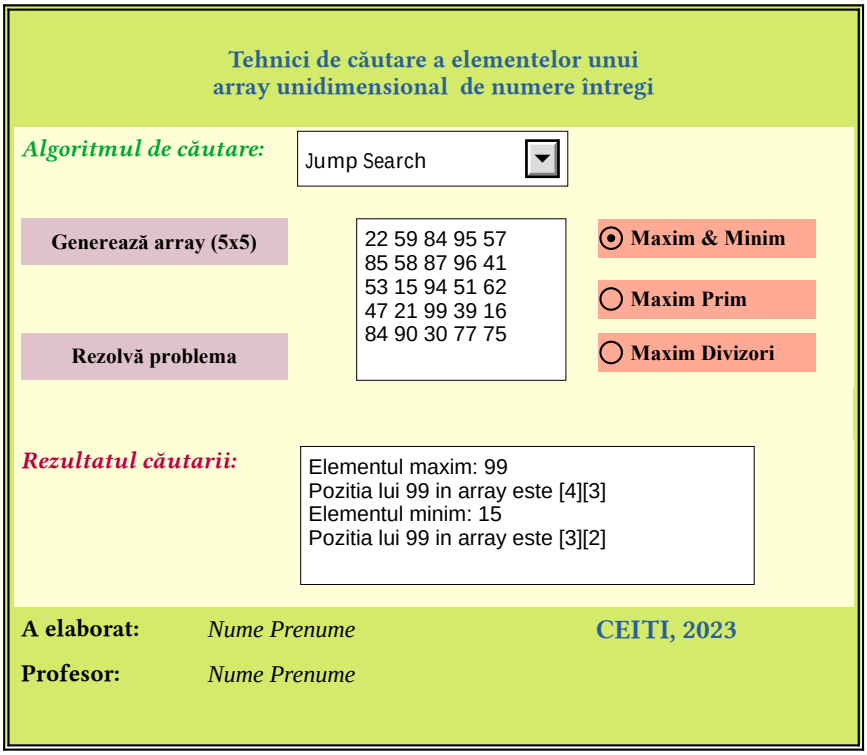
2. Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: o componentă de tip Buton, 2 component3 de tip ComboBox, 9 componente de tip Panel, o componentă de tip MemoText și 8 componente Label.

- ***ComboBox1** va include elementul ce trebuie căutat din intervalul [1, 1000]. **ComboBox2** va include denumirea algortmului de căutare a elementului: Linear Search, Binary Search, Jump Search, Interponation Search, Exponential Search sau Fibonacci Search.*
- ***Buton** va permite rezolvarea problemei conform algoritmului de căutare și va afișa în componeta **MemoText** datele corespunzătoare (vezi modelul).*



3. Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: 2 componente de tip Buton, o componentă de tip ComboBox, 3 componente de tip Radio Buton, 2 componente de tip MemoText și 8 componente de tip Label.

- ***ComboBox*** *va include denumirea algortmului de căutare a elementului: Linear Search, Binary Search, Jump Search, Interponation Search, Exponential Search sau Fibonacci Search.*
- ***Buton1*** *va permite generarea unui array bidimensional de numere întregi de două cifre, array-ul bidimensional este pătratic, de dimensiunea 5. Datele generate vor fi afișate în componenta* ***MemoText1****.* ***Buton2*** *va permite rezolvarea problemei conform algoritmului de căutare și va afișa în componeta* ***MemoText2*** *datele corespunzătoare (vezi modelul). Rezultatele vor fi afișate după ce se va genera array-ul bidimensional și selectarea opțiunii dintre grupul de butoane radio conform condițiilor. Pentru fiecare element găsit, se va afișa și coordonatele acestuia în array.*



Capitolul III

ALGORITMI DE PROGRAMARE DINAMICĂ

De ce avem nevoie de algoritmul SIMPLEX ?

În optimizarea matematică, algoritmul simplex al lui Dantzig este un algoritm popular pentru programarea liniară. Metoda simplex este o procedură iterativă de rezolvare a problemelor de programare liniară aduse la forma tabelară. Metoda simplex generează noi soluţii fezabile de bază care cresc valoarea funcţiei obiectiv (sau cel puţin o lasă neschimbată), prin generarea unor noi forme tabelare pentru sistemul de ecuaţii. Atunci când nu se mai poate aduce nici o îmbunătăţire a fost atinsă soluţia optimă.

O gamă largă de probleme de programare liniară deţin multe variabile atunci când ne propunem să-i găsim soluția, astfel metoda grafică de rezolvare a problemei respective nu poate fi aplicată. Voi prezenta aplicabilitatea algoritmului Simplex în soluționarea unei varietăți de probleme de programare liniară. Diverse aplicații industriale aplică algoritmul Simplex în soluționarea problemelor de programare liniară cu mii de restricții și variabile.

Care este aplicabilitatea algoritmului SIMPLEX ?

În activitatea de conducere, organizare și planificare se întâlnesc o serie de probleme privind repartizarea resurselor (a unor cantităţi de la unităţile furnizoare la cele consumatoare). Pentru elaborarea planului de transport intern la o întreprindere se pune problema de a repartiza anumite cantităţi privind un anumit material de la depozite (furnizori) la unităţi de producţie – secţii, ateliere (consumatori). Această repartizare trebuie făcută în aşa fel încât să se asigure costuri minime de transport.

Există o gamă foarte largă de fenomene economice care pot fi reprezentate prin modele de programare liniară de tip transport sau foarte asemănătoare cu acestea. Prezintam în continuare câteva dintre acestea: cu rute blocate, cu puncte intermediare, problema afectării, problema încărcării utilajelor și problema de transport a lui Koopmans.

3.1 Metoda grafică[1] de rezolvare a PPL.

Metodele grafice de rezolvare a problemelor de programare liniară (PPL) sunt aplicabile numai pentru modelele de programare liniară cu două variabile. În general, soluția grafică a unui sistem liniar de inecuații cu două necunoscute (variabile) este un domeniu de valori (un semiplan) denumit ***domeniu*** *(sau* ***regiune****)* ***admisibil(ă)*** *sau* ***acceptabil(ă)****. Este important, deoarece se definesc (și se vizualizează) tipurile de soluții: soluții admisibile (SA), soluții admisibile de baza (SAB), soluția optimă (SO), dacă există. Se precizează și cazurile în care: soluția optimă este unică, există mai multe soluții optime, nu există soluție optimă sau nu există un optim finit.*

Algoritmul metodei grafice

1. *Condițiile de nenegativitate:* $x_1>0,\ x_2>0$;
2. *Condițiile de nenegativitate sunt satisfacute în cadranul I;*
3. *Pentru a satisface toate restricțiile, pentru fiecare restricție se atașează:*
   - *o ecuație, ce reprezintă o dreaptă;*
   - *o inecuație strictă ce reprezintă un semiplan;*
   - *se alege semiplanul corespunzător;*
   - *deoarece toate restricțiile trebuie să fie satisfăcute, intersectează toate ariile.*
4. *Se obține astfel aria admisibilă. Coordonatele tuturor punctelor acestei arii verifică toate restricțiile și condițiile de nenegativitate.*
5. *Mulțimea soluțiilor admisibile (SA) este mulțimea coordonatelor* (x_1, x_2) *ale tuturor punctelor care satisfac toate restricțiile și condițiile de nenegativitate. Punctele se află în aria admisibilă și pe conturul ei.*
6. *Aria admisibilă este reprezentată haşurat. Aria admisibilă are o infinitate de puncte, deci mulțimea soluțiilor admisibile este în acest caz infinită.*
7. *În această mulțime trebuie să alegem acel punct ale cărui coordonate conferă funcției obiectiv valoare cea mai mare. Acel punct va reprezenta soluția optimă. Este clar că trebuie să restrângem mulțimea de puncte în care să căutăm soluția optimă, astfel ca această mulțime să fie finită. Aria admisibilă este o mulțime convexă.*

1 *Această temă se va studia doar la lecțiile teoretice, recomandăm aplicarea softului GeoGebra atunci când procesul de instruire este online.*

8. *Se cercetează numai vârfurile acestei mulțimi. Coordonatele vârfurilor poligonului convex, care înconjoară aria admisibilă constituie mulțimea (x_1, x_2) a soluțiilor admisibile de bază. Soluția optimă se află în unul din vârfurile poligonului.*
9. *Pentru determinarea soluției optime se poate proceda în două moduri:*
   - *Se calculează coordonatele vârfurilor și apoi se află valoarea funcției obiectiv în fiecare vârf. Soluția optimă este punctul care conferă funcției obiectiv valoarea optimă.*
   - *Se caută valoarea maximă a funcției obiectiv, care este tot liniară, deci acesteia i se poate atașa o dreaptă d. Apoi se deplasează d, paralel cu ea însăși, până la aria admisibilă și se alege punctul în care ordonata la origine, este cea mai mare. Punctul ales este un vârf al poligonului și reprezintă valoarea de maxim.*

Probleme rezolvate cu softul matematic GeoGebra Graphing Calculator[2]

1. Problema de transport

De la două depozite, A și B trebuie să fie distribuite, la trei piețe ale orașului fructe. Depozitul A stochează 10 tone de fructe pe zi, iar depozitul B stochează 15 tone de fructe pe zi. Primele două piețe au nevoie de 8 tone de fructe în fiecare zi, în timp ce al treilea are nevoie de 9 tone pe zi. Costul de transport de la fiecare depozit la fiecare piață este dat de tabelul de mai jos. Vom planifica transportul pentru a obține costul minim.

Depozit	Piața 1	Piața 2	Piața 3
A	10	15	20
B	15	10	10

Rezolvare:

Vom nota cu x cantitatea (în tone) care se transportă din depozitul A în piața 1, cu y cantitatea (în tone) care se transportă din depozitul A în piața 2. Vom obține următorul tabel:

Depozit	Piața 1	Piața 2	Piața 3
A	x	y	10-x-y
B	8-x	8-y	9-(10-x-y)=x+y

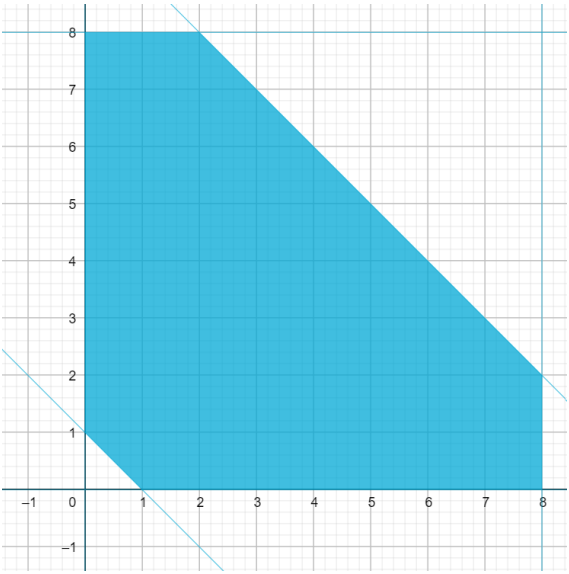
2 *Analizați **Anexa 1:** Implementarea softului GeoGebra la rezolvarea PPL*

Funcția obiectiv va fi:

$F(x,y)=10x+15y+20(10-x-y)+15(8-x)+10(8-y)+10(x+y-1)$, *deci vom avea:* $F(x,y)=-15x-5y+390$, *am obținut funcția obiectiv ce trebuie de minimizat conform sistemului de constrângeri:*

$$\begin{cases} x,y\leqslant 8 \\ x+y\leqslant 10 \\ x+y\geqslant 1 \\ x,y\geqslant 0 \end{cases}$$

Reprezentând grafic toate dreptele vom obține astfel regiunea admisibilă:



Apoi, rezolvând sistemele de ecuații corespunzătoare obținem vârfurile regiunii: A (0;1), B (0;8), C (2;8), D (8;2), E (8;0) și F (1;0).

Evaluând funcția obiectiv în vârfurile poligonului obținem: F(A)=385, F(B)=350, F(C)=320, F(D)=260, F(E)=270, F(F)=375.

Minimul se obține pentru x=8, y=2 și are valoarea de 260.

În acest caz, costul minim se realizează prin transportul:

- *de la depozitul A, 8 tone la piețele 1, 2 și nimic la 3;*
- *de la depozitul B, nimic la piața 1, 6 la 2 și 9 la piața 3.*

Putem spune că costul minim este atins cu această distribuție optimă.

2. Planul optim de producție

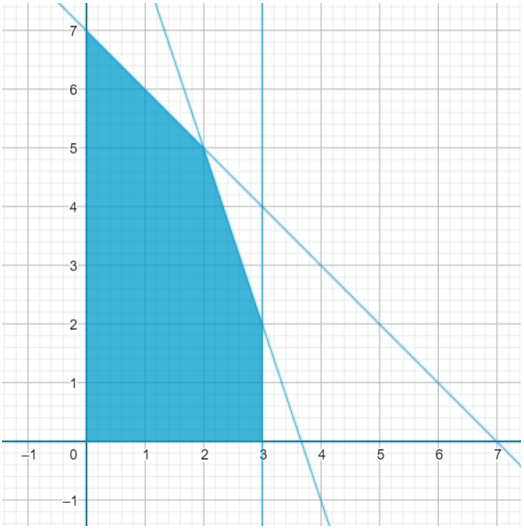
Pentru a produce sacouri și pantaloni, o fabrică utilizează trei mașini: de tăiat, de cusut și de vopsit. Pentru efectuarea unui sacou se utilizează mașina de tăiat - o oră, de cusut - trei ore, iar cea de vopsit - o oră, iar pentru pantaloni se folosește mașina de tăiat - o oră, de cusut - o oră, iar cea de vopsit – nici o oră. Mașina de vopsit poate fi utilizat timp de - trei ore, de cusut - unsprezece ore și cea de tăiat - șapte ore. Tot ceea ce este produs se vinde și se obține un profit de 8 euro pentru fiecare sacou și 5 euro pentru fiecare pereche de pantaloni. În ce mod vom folosi mașinile pentru a obține un profit maxim?

Rezolvare:

Fie x numărul de sacouri și y numărul de perechi de pantaloni. Funcția obiectiv va fi: F(x,y)=8x+5y și trebuie să o maximizăm conform următorului sistem de constrângeri:

$$\begin{cases} x \leqslant 3 \\ x+y \leqslant 7 \\ 3x+y \leqslant 11 \\ x, y \geqslant 0 \end{cases}$$

Reprezentând grafic toate dreptele vom obține astfel regiunea admisibilă:



Din reprezentare se observă că vârfurile poligonului sunt A(0;0), B(3;0), C(3;2), D(2;5) și E(0;7).

Valoarea maximă a funcției se atinge în punctul D (2;5).

Deci vom produce 2 sacouri și 5 perechi de pantaloni și vom avea un profit de 41 euro.

3. Amestec

Un fermier trebuie să asigure un minimum de 4 mg pe zi de vitamina A și 6 mg de vitamina B, în furajele pe care le dă vitelor. El are două tipuri de furaje, F_1 și F_2, al căror conținut de vitamină pe kilogram este reprezentat în tabelul de mai jos:

	Vitamina A	**Vitamina B**
Furaj F_1	2	6
Furaj F_2	4	3

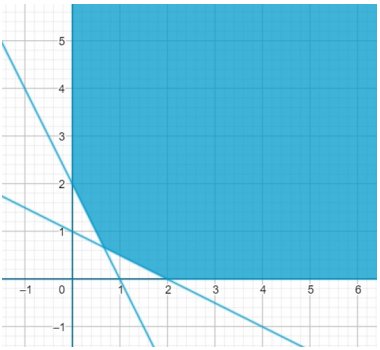
Kilogramul de furaj de tipul F_1 costă 0.4 €, iar Kilogramul de furaj de tipul F_2 costă 0.6 €. Cum ar trebui să se amestece cele 2 tipuri de furaje, pentru a obține o hrană mixtă cu vitaminele necesare bovinelor, astfel încât cheltuielile să fie minime?

Rezolvare:

Vom numi x kilogramele de furaje F_1 și y kilogramele de furaje F_2. Funcția obiectiv va fi: $F(x,y)=0.4x+0.6y$ și trebuie să o minimizăm conform următorului sistem de constrângeri:

$$\begin{cases} 2x+4y \geqslant 4 \\ x+2y \geqslant 2 \\ 6x+3y \geqslant 6 \\ 2x+y \geqslant 2 \\ x, y \geqslant 0 \end{cases}$$

Reprezentând grafic toate dreptele vom obține astfel regiunea admisibilă:



Punctul de minim $P\left(\frac{2}{3};\frac{2}{3}\right)$ *este atins în punctul de intersecție al drepelor:* 2x+y=2 și x+2y=2.

Prin urmare, ar trebui să fie amestecat $\frac{2}{3}$ *kg din furajul F_1 cu* $\frac{2}{3}$ *kg furaj F_2.*

4. Problema folosirii optime a resurselor

O patiserie face două tipuri de tarte: T_1 și T2, pentru care se utilizează trei ingrediente: A, B și C. Se dispune de 150 kg de ingredient A, 90 kg de ingredient B și de 150 kg de ingredient C. Pentru a face tarta T_1, se amestecă 1 kg de A, 1 kg de B și de 2 kg de C, în timp ce pentru a face tarta T_2, este nevoie de 5 kg de A, 2 kg de B și 1 kg de C.

a) *În cazul în care tarta T_1 se vinde la 10 lei, iar T_2 la 23 lei, ce cantități ar trebui fabricate de fiecare tip, pentru a maximiza veniturile?*

b) *Dacă prețul unei tarte de tipul T_1 este de 15 lei, care ar fi prețul unei tarte de tipul T_2 în cazul în care o soluție optimă este de a face 60 de tarte de tipul T_1 și 15 de tipul T_2?*

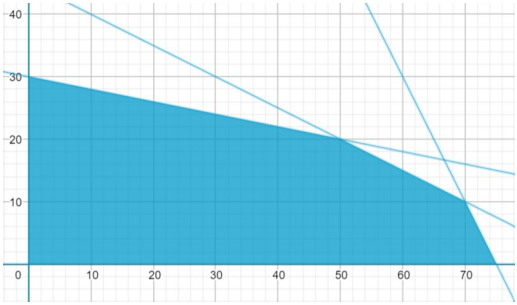
Rezolvare:

Vom numi x numărul de tarte T_1 și y numărul de tarte T_2.

a) *Funcția obiectiv va fi: F(x,y)=10x+23y și trebuie să o maximizăm conform următorului sistem de constrângeri:*

$$\begin{cases} x+5y \leqslant 150 \\ x+2y \leqslant 90 \\ 2x+y \leqslant 150 \\ x, y \geqslant 0 \end{cases}$$

Reprezentând grafic toate dreptele vom obține astfel regiunea admisibilă:



Valoarea maximă este atinsă la punctul P(50,20), obținut la intersecția dreptelor: x+5y=150 și x+2y=90.

De aceea, trebuie să fie tarte de tipul T_1 - 50 și tarte de tipul T_2 - 20 pentru a obține venitul maxim.

Venitul maxim obținut fiind: $10*T_1 + 23*T_2 = 10*50 + 23*20 = 500 + 460 = 960$ lei.

b) *Dacă numim p prețul tartei de tipul T_2, venitul ar fi dat de funcția obiectiv: G(x,y)=15x+py. Funcția G(x,y) atinge maximul în punctul P(60,15), care nu este vârf al regiunii admise. Asta se datorează faptului că există o infinite de soluții, iar dreptele 15x+py=0 și x+2y=90 vor fi paralele. Deci obținem egalitatea: $-5*2 = -1*p$, de aici obținem că p=30. Prețul pentru T_2 este 30 lei.*

3.2 Algoritmul SIMPLEX.

În practică, majoritatea problemelor de programare liniară au multe variabile, fapt ce face ca metoda soluției grafice de rezolvare să nu poată fi aplicată în cazul problemelor de programare liniar cu mai mult de două variabile. Algoritmul Simplex este utilizat în rezolvarea unei mari varietăți de probleme de programare liniară. În multe aplicații industriale, algoritmul Simplex este utilizat în rezolvarea problemelor de programare liniară cu mii de restricții și variabile. Aplicarea algoritmului Simplex presupune parcurgerea următoarelor etape:

- *aducerea problemei la forma standard;*
- *transformarea formei standard în forma tabelară;*
- *rezolvarea problemei cu ajutorul metodei Simplex.*

O problemă de programare liniară se spune că este în forma standard atunci când toate restricţiile devin ecuaţii şi toate variabilele nenegative. O inegalitate devine egalitate prin introducerea unei noi variabile, numită variabilă de compensare, ea reprezentând valoarea rămasă neutilizată din resursa respectivă.

O problemă de programare liniară este în formă tabelară atunci când toate constrângerile sunt ecuații având partea dreaptă mai mare sau egală cu zero și atunci când în fiecare ecuație apare o variabilă care are coeficientul egal cu +1 într-o singură ecuație și egal cu 0 în toate celelalte ecuații. Variabila cu coeficientul egal cu +1 într-o ecuație și într-o ecuație și 0 în celelalte se numește variabilă de bază asociată ecuației. Toate celelalte variabile care nu sunt variabile de bază se numesc variabile secundare.

În general, atunci când există mai multe variabile decât ecuații, există un număr infinit de soluții într-un sistem de ecuații. Dar chiar dacă există o infinitate de soluții, uneori este dificil de a găsi o singură soluție. Valoarea formei tabelare oferă cu ușurință găsirea unei soluții prin stabilirea valorilor variabilelor secundare la zero și a valorilor variabilelor de bază la valoarea corespunzătoare membrului drept al ecuației. Aceste soluții se numesc soluții fezabile de bază.

Metoda Simplex este o procedură iterativă de rezolvare a problemelor de programare liniară aduse la forma tabelară. Metoda Simplex generează noi soluții fezabile de bază care cresc valoarea funcției obiectiv (sau cel puțin o lasă neschimbată), prin generarea unor noi forme tabelare pentru sistemul de ecuații. Atunci când nu se mai poate aduce nici o îmbunătățire, atunci a fost atinsă soluția optimă.

În principal, metoda Simplex constă din 3 etape:

1. *Găsirea celei mai mari valori pozitive pentru C_j - F_j. Aceasta va desemna coloana pivot. Dacă nu există o astfel de valoare, atunci soluția optimă a fost deja găsit.*
2. *Pentru fiecare valoare pozitivă din coloana pivot se găsește raportul: (membrul drept)/(elementul corespunzător din coloana pivot). Raportul minim stabilește linia pivot. La intersecția coloanei pivot cu linia pivot se găsește elementul pivot.*
3. *Se generează noua formă tabelară astfel:*
   (a) *Se împarte linia pivot la elementul pivot;*
   (b) *Pentru toate celelalte linii, se înmulțește noua linie generată la punctul (a) prin elementul corespunzător din coloana pivot și se extrage din linia curentă;*
   (c) *Se completează celulele taloului și se trece la etapa 1.*

Implementarea algoritmului Simplex în formă tabelară

Fie următoarea problemă de programare liniară:

Funcția obiectiv	Sistemul de constrângeri
$F \rightarrow MAX$ $F = 2x + 3y + 4z$	$\begin{cases} x + y + z \leqslant 30 \\ 2x + y + 3z \geqslant 60 \\ x - y + 2z = 20 \\ x, y, z \geqslant 0 \end{cases}$

a) *Scrieți problema în forma standard.*
b) *Adăugați variabilele artificiale necesare pentru a obține prima formă tabelară și corectați corespunzător funcția obiectiv.*
c) *Rezolvați problema prin metoda Simplex, apoi verificați rezultatele cu ajutorul softului matematic portabil Maple 2020*[3]*;*
d) *Ce schimbare ar trebui făcută astfel încât să urmăriți minimizarea funcției obiectiv?*

3 *Analizați* ***Anexa 2:*** *Implementarea softului Maple 2020 la rezolvarea PPL*

Rezolvare:

a) *Pentru a aduce problema la forma standard se introduc variabilele de compensare s1 și s2, astfel încât restricțiile se transform din inegalități în egalități. Această formă nu este și tabelară, deoarece în a 2-a și a 3-a restricție nu există variabile cu coeficient +1 într-una singură și coeficientul 0 în celelalte.*

b) *Pentru a aduce problema la forma tabelară se adaugă variabilele artificiale a_1 pe linia a 2-a și respectiv, a_2 în linia a 3-a. Funcția obiectiv va fi și ea modificată prin adăugarea noilor variabile multiplicate cu coeficientul –M. Astfel forma tabelară va fi:*

Iterația 1:

Baza	C_j	x_1	x_2	x_3	s_1	s_2	a_1	a_2	
		2	3	4	0	0	-M	-M	
s_1	0	1	1	1	1	0	0	0	30
a_1	-M	2	1	3	0	–1	1	0	60
a_2	-M	1	–1	2	0	0	0	1	20
	F_j	–3M	0	–5M	0	M	-M	-M	–80M
	$C_j - F_j$	3M+2	3	5M+4	0	-M	0	0	

Deoarece a_2 este o variabilă artificială și va deveni o variabilă secundară, îi vom abandona coloana. Elementul ***2*** *se numește pivot. Cum a fost determinat acesta?*

c) *Rezolvăm problema prin metoda Simplex.*

Iterația 2:

Baza	C_j	x_1	x_2	x_3	s_1	s_2	a_1	
		2	3	4	0	0	-M	
s_1	0	1/2	3/2	0	1	0	0	20
a_1	-M	1/2	5/2	0	0	–1	1	30
x_3	4	1/2	–1/2	1	0	0	0	10
	F_j	-(1/2)M+2	-(5/2)M-2	4	0	M	-M	–80M
	$C_j - F_j$	(1/2)M+2	(5/2)M+5	0	0	-M	0	

*Deoarece a_1 este o variabilă artificială și va deveni o variabilă secundară, îi vom abandona coloana. Pivotul este **5/2**. Cum a fost determinat acesta în cazul dat?*

Iterația 3:

Baza	C_j	x_1	x_2	x_3	s_1	s_2	
		2	3	4	0	0	
s_1	0	1/5	1	0	1	15/4	11
x_2	3	1/5	0	0	0	−2/5	6
x_3	4	3/5	1	1	0	−1/5	13
F_j		3	3	4	0	−2	−30M+40
$C_j - F_j$		−1	0	0	0	2	

Iterația 4:

Baza	C_j	x_1	x_2	x_3	s_1	s_2	
		2	3	4	0	0	
s_2	0	1/3	1	0	5/3	1	55/3
x_2	3	1/3	0	0	2/3	0	40/3
x_3	4	2/3	1	1	1/3	0	50/3
F_j		11/3	3	4	10/3	0	320/3
$C_j - F_j$		−5/3	0	0	−10/3	0	

Astfel soluția optimă este $x_1=0$; $x_2=40/3$; $x_3=50/3$; $s_1=0$ și $s_2=55/3$, iar valoarea optimă este $F=320/3$.

d) *Problema se poate transforma într-o problemă de minimizare prin schimbarea valorii coeficienților variabilelor artificiale de la $-M$ la $+M$.*

Soluția acestei probleme poate fi verificată și online:

1. *Simplex method calculator (atozmath.com)*
2. *Online Calculator: Simplex Method (linprog.com)*

3.3 Algoritmul SIMPLEX DUAL.

Metoda Simplex pivotează de la un dicționar (vocabular al problemei) fezabil la un dicționar fezabil, care încearcă să ajungă la un dicționar a cărui rând are toți coeficienții săi pozitivi. În analiza sensibilității, anumite modificări ale uneii PPL vor duce la dicționare a căror linii „arată optim", dar care nu sunt fezabile. Pentru a profita de aceste dicționare, vom dezvolta o versiune duală a metodei Simplex. Apelați un dicționar dual fezabil dacă toți coeficienții din rândul său nu sunt pozitivi. Metoda Dual Simplex va pivota de la un dicționar dual fezabil la un dicționar dual fezabil care lucrează spre fezabilitate.

Această nouă strategie de pivotare se numește Metoda Dual Simplex, deoarece este la fel ca și metoda Simplex obișnuită pentru problema liniară duală. Acest lucru explică, de asemenea, termenul „dual feasible": fiecare dicționar pentru primal are un dicționar corespunzător pentru dual și un dicționar primal este dual fezabil exact atunci când dicționarul corespunzător pentru dual este fezabil (în sensul obișnuit). Cu toate acestea, nu vom profita cu adevărat de această corespondență: nu vom vorbi direct despre PPL duale în loc să explicăm cum să efectuați acești pivoți duali direct pe un dicționar dual fezabil pentru primal.

În principal, metoda Simplex Dual constă din 3 etape:

1. *Formulează problema*
   a) *formulați modelul matematic al problemei de programare liniară dată;*
   b) *dacă funcția obiectivă este de tip minimizare, atunci schimbați-o în tip de maximizare;*
   c) *toate „≥" constrâng la „≤" constrâng prin multiplicare cu −1;*
   d) *transformați fiecare constrângere „≤" într-o constrângere „=" adăugând o variabilă slack (auxiliară) la fiecare constrângere și atribuiți un coeficient de cost 0 în funcția obiectivului.*

2. *Aflați soluția de bază inițială*
   Găsiți soluția inițială de bază fezabilă prin setarea valorii zero la variabilele de decizie.

3. *Testare pentru optimalitate*
   a) *dacă toate valorile lui $X_B \geq 0$, atunci soluția curentă este soluția optimă, încheiați procesul;*
   b) *dacă există $X_B < 0$, selectați X_B minim negativ și acest rând se numește rând cheie, notată prin RK;*

c) *găsiți un raport* $\frac{C_j - F_j}{RK_j}$ *unde* $RC_j < 0$;

d) *găsiți un raport pozitiv minim, iar această coloană se numește coloană cheie, notată prin CK;*

e) *găsiți noul tabel de soluții și apoi repetați tot pasul 3.*

Implementarea algoritmului Simplex Dual în formă tabelară

Fie următoarea problemă de programare liniară:

Funcția obiectiv	Sistemul de constrângeri
$F \to MAX$ $F = -2x - y$	$\begin{cases} -3x - y \leqslant -3 \\ -4x - 3y \leqslant -6 \\ -x - 2y \leqslant -3 \\ x, y \geqslant 0 \end{cases}$

Rezolvare:

Problema este convertită în formă canonică prin adăugarea variabilelor slabe, excedentare și artificiale după cum este cazul:

a) *deoarece constrângerea 1 este de tip „≤" ar trebui să adăugăm variabila* S_1;

b) *deoarece constrângerea 2 este de tip „≤" ar trebui să adăugăm variabila* S_2;

c) *deoarece constrângerea 3 este de tip „≤" ar trebui să adăugăm variabila* S_3.

Iterația 1		C_j	**-2**	**-1**	**0**	**0**	**0**
B	C_B	X_B	**x**	**y**	S_1	S_2	S_3
S_1	**0**	**-3**	-3	-1	1	0	0
S_2	**0**	**-6**	-4	-3	0	1	0
S_3	**0**	**-3**	-1	-2	0	0	1
F	**0**	F_j	**0**	**0**	**0**	**0**	**0**
		$C_j - F_j$	**-2**	**-1**	**0**	**0**	**0**
		$\frac{C_j - F_j}{S_2}$	**1/2**	**1/3**	**---**	**---**	**---**

X_B negativ minim este –6 și indicele său de rând este 2. Deci, variabila de bază care pleacă este S2. Raportul pozitiv minim este 0,33 și indicele coloanei sale este 2. Deci, variabila de intrare este y. Elementul pivot este –3.

- *Intrare = y, Ieșire = S_2, Element cheie (pivot) = –3. R2 (nou) = R2 (vechi) ÷ (-3), R1 (nou) = R1 (vechi) + R2 (nou) și R3 (nou) = R3 (vechi) + 2×R2 (nou).*

Iterația 2		C_j	-2	-1	0	0	0
B	C_B	X_B	x	y	S_1	S_2	S_3
S_1	0	–1	–5/3	0	1	–1/3	0
y	–1	2	4/3	1	0	–1/3	0
S_3	0	1	5/3	0	0	–2/3	1
F	–2	F_j	–4/3	–1	0	1/3	0
		$C_j - F_j$	–2/3	0	0	–1/3	0
		$\frac{C_j - F_j}{S_1}$	2/5	---	---	1	---

X_B negativ minim este –1 și indicele său de rând este 1. Deci, variabila de bază care pleacă este S_1. Raportul pozitiv minim este 2/5, adică 0.4, iar indexul coloanei este 1. Deci, variabila de intrare este x. Elementul pivot este –5/3.

- *Intrare = x, Plecare = S_1, Element cheie = –5/3. R1 (nou) = R1 (vechi) × (–3/5), R2 (nou) = R2 (vechi) - 4/3×R1 (nou), R3 (nou) = R3 (vechi) - 5/3×R1 (nou).*

Iterația 3		C_j	-2	-1	0	0	0
B	C_B	X_B	x	y	S_1	S_2	S_3
x	–2	3/5	1	0	–3/5	1/5	0
y	–1	6/5	0	1	4/5	–3/5	0
S_3	0	0	0	0	1	–1	1
F	–12/5	F_j	–2	–1	2/5	1/5	0
		$C_j - F_j$	0	0	–2/5	–1/5	0
		Raport	---	---	---	---	---

Deoarece toate C_j-$F_j \leq 0$ și toate $X_{Bi} \geq 0$, astfel soluția curentă este soluția optimă. Prin urmare, soluția optimă se ajunge cu valoarea variabilelor ca: x = 3/5 și y = 6/5, unde obținem $F_{max} = -12/5$.

Implementarea algoritmului Simplex Dual în formă tabelară

Fie următoarea problemă de programare liniară:

Funcția obiectiv	Sistemul de constrângeri
$F \to MIN$ $F = 2x + 3y + 0z$	$\begin{cases} 2x - y - z \geq 3 \\ x - y + z \geq 2 \\ x, y, z \geq 0 \end{cases}$

Rezolvare:

Pentru a aplica metoda dual simplex, convertiți Min în Max și toate constrângerile ≥ în constrângere ≤ înmulțind −1. Problema este convertită în formă canonică prin adăugarea variabilelor slabe, excedentare și artificiale după cum este cazul:

a) *deoarece constrângerea 1 este de tip „≤" ar trebui să adăugăm variabila S_1;*
b) *deoarece constrângerea 2 este de tip „≤" ar trebui să adăugăm variabila S_2.*

Iterația 1		C_j	-2	-3	0	0	0
B	C_B	X_B	x	y	z	S_1	S_2
S_1	0	−3	−2	1	1	1	0
S_2	0	−2	−1	1	−1	0	1
F	0	F_j	0	0	0	0	0
		$C_j - F_j$	−2	−3	0	0	0
		$\frac{C_j - F_j}{S_1}$	1	---	---	---	---

X_B negativ minim este −3 și indicele său de rând este 1. Deci, variabila de bază de plecare este S_1. Raportul pozitiv minim este 1, iar indexul coloanei este 1. Deci, variabila de intrare este x. Elementul pivot este −2.

- *Intrare = x, Ieșire = S_1, Element cheie = –2, R1 (nou) = R1 (vechi) ÷ (–2), R2 (nou) = R2 (vechi) + R1 (nou).*

Iterația 2		C_j	-2	-3	0	0	0
B	C_B	X_B	x	y	z	S_1	S_2
x	–2	3/2	1	–1/2	–1/2	–1/2	0
S_2	0	–1/2	0	1/2	–3/2	–1/2	1
F	–3	F_j	–2	1	1	1	0
		$C_j - F_j$	0	–4	–1	–1	0
		$\frac{C_j - F_j}{S_2}$	---	---	2/3	2	---

X_B minim negativ este –1/2, iar indicele său de rând este 2. Deci, variabila de bază care pleacă este S_2. Raportul pozitiv minim este de 2/3, adică 0.(6), iar indexul coloanei este 3. Deci, variabila de intrare este z. Elementul pivot este –3/2.

- *Intrare = z, Ieșire = S_2, Element cheie = - 3/2.* R2 (nou) = R2 (vechi) × (–2/3), R1 (nou) = R1 (vechi) + 1/2×R2 (nou).

Iterația 3		C_j	-2	-3	0	0	0
B	C_B	X_B	x	y	z	S_1	S_2
x	–2	5/3	1	–2/3	0	–1/3	–1/3
z	0	1/3	0	–1/3	1	1/3	–2/3
F	–10/3	F_j	–2	4/3	0	2/3	2/3
		$C_j - F_j$	0	–13/3	0	–2/3	–2/3
		Raport	---	---	---	---	---

Deoarece toate C_j-$Z_j \leq 0$ și toate $X_{Bi} \geq 0$, astfel soluția actuală este soluția optimă. Prin urmare, soluția optimă se ajunge cu valoarea variabilelor ca: x = 5/3, y = 0, z =1/3. F_{Max} = –10/3, deci F_{Min} = 10/3.

Algoritmi de programare dinamică
Itemi recomandați pentru lecțiile practice 9-10

1. *Implementați metoda grafică de rezolvare pentru problemele:*

Nr.	*Constrângerile*	*Funcția obiectiv*	*Tipul de rezolvare*
1	$\begin{cases} x \leqslant 6 \\ 2x+3y \leqslant 6 \\ -x+y \leqslant 4 \\ x, y \geqslant 0 \end{cases}$	$F \to MAX$ $F = 4x+3y$	a) *manuală (varianta matematică);* b) *aplicați softul matematic GeoGebra.*
2	$\begin{cases} x+y \leqslant 10 \\ -x+y \leqslant 6 \\ x+2y \leqslant 16 \\ x, y \geqslant 0 \end{cases}$	$F \to MAX$ $F = 2x+5y$	a) *manuală (varianta matematică);* b) *aplicați softul matematic GeoGebra.*
3	$\begin{cases} x+y \leqslant 4 \\ 2x+y \geqslant 2 \\ x+2y \leqslant 16 \\ x, y \geqslant 0 \end{cases}$	$F \to MIN$ $F = 3x+2y$	a) *manuală (varianta matematică);* b) *aplicați softul matematic GeoGebra.*
4	$\begin{cases} -3x+2y \leqslant 4 \\ 2x-3y \leqslant 9 \\ x+2y \leqslant 16 \\ x, y \geqslant 1 \end{cases}$	$F \to MIN$ $F = 3x-y$	a) *manuală (varianta matematică);* b) *aplicați softul matematic GeoGebra.*
5	$\begin{cases} x+y \leqslant 10 \\ 3x+y \geqslant 7 \\ x, y \geqslant 0 \end{cases}$	$F \to MAX$ $F = -6x+4y$	a) *manuală (varianta matematică);* b) *aplicați softul matematic GeoGebra.*
6	$\begin{cases} 7x+2y \leqslant 11 \\ -x+5y \leqslant 4 \\ x, y \geqslant 0 \end{cases}$	$F \to MIN$ $F = x-3y$	a) *manuală (varianta matematică);* b) *aplicați softul matematic GeoGebra.*

2. *Implementați algoritmul SIMPLEX pentru următoarele probleme:*

Nr.	*Constrângerile*	*Funcția obiectiv*	*Tipul de rezolvare*
1	$\begin{cases} x+y+z\leqslant 30 \\ 2x+y+3z\geqslant 60 \\ x-y+2z=20 \\ x,y,z\geqslant 0 \end{cases}$	$F\rightarrow MAX$ $F=2x+3y+4z$	a) *manuală;* b) *aplicați calculul tabelar (MS Excel sau LibreOffice Calc).*
2	$\begin{cases} 3x+6y\leqslant 30 \\ 4x+2y+z\leqslant 20 \\ y+z\leqslant 10 \\ x,y,z\geqslant 0 \end{cases}$	$F\rightarrow MAX$ $F=2x+3y-z$	a) *manuală;* b) *aplicați calculul tabelar (MS Excel sau LibreOffice Calc).*
3	$\begin{cases} x+y+z\leqslant 50 \\ 2x+3y-z\leqslant 30 \\ x,y,z\geqslant 0 \end{cases}$	$F\rightarrow MAX$ $F=3x+4y+2z$	a) *manuală;* b) *aplicați calculul tabelar (MS Excel sau LibreOffice Calc).*
4	$\begin{cases} x+2y+z\geqslant 5 \\ 3y+2z\leqslant 12 \\ x,y,z\geqslant 0 \end{cases}$	$F\rightarrow MIN$ $F=2x-3y-4z$	a) *manuală;* b) *aplicați calculul tabelar (MS Excel sau LibreOffice Calc).*
5	$\begin{cases} -x+y-z\leqslant -16 \\ 2x-2z\geqslant 30 \\ x+2y\geqslant 8 \\ x,z\geqslant 0, y\leqslant 0 \end{cases}$	$F\rightarrow MIN$ $F=2x+2y-z$	a) *manuală;* b) *aplicați calculul tabelar (MS Excel sau LibreOffice Calc).*
6	$\begin{cases} x+y+z\leqslant 15 \\ 2x+y+3z\geqslant 30 \\ x-y+2z=15 \\ x,y,z\geqslant 0 \end{cases}$	$F\rightarrow MIN$ $F=2x-3y+4z$	a) *manuală;* b) *aplicați calculul tabelar (MS Excel sau LibreOffice Calc).*

3. *Implementați algoritmul SIMPLEX DUAL pentru problemele propuse la itemii 1 și 2.*
4. *Elaborați un program care va implementa biblioteca grafică în C/C++ pentru a verifica vizual soluțiile problemelor propuse la itemul 1, afișând la ecran sistemul de coordonate și punctele de intersecție a acestora.*
5. *Elaborați un program în C/C++ care va implementa algoritmii Simplex și Simplex Dual pentru o PPL urmând etapele de rezolvare a fiecărui algoritm.*

6. *Compania "MicroPC" are trei fabrici de asamblare a microprocesoarelor. Cea din San Francisco are o producţie lunară de 1700 de unităţi. Fabrica din Los Angeles produce 2000 unităţi pe lună, iar cea din Phoenix 1700 unităţi. Microprocesoarele sunt vândute în patru magazine. Magazinul din San Diego a emis o comandă de 1700 de unităţi pentru luna următoare, cel din Barstow are o cerere de 1000 unităţi, cel din Tucson 1500 de unităţi, iar pentru magazinul din Dallas cererea este de 1200 unităţi. Costul de transport al unui microprocesor de la fieecare fabrică până la fieecare magazin este prezentat în tabelul următor:*

Fabrici	**Magazine**			
	San Diego	**Barstow**	**Tucson**	**Dallas**
San Francisco	5	3	2	6
Los Angeles	4	7	8	10
Phoenix	6	5	3	8

În calitate de manager de distribuţie, formulaţi un model matematic pentru a găsi cel mai ieftin mod de distribuţie.

7. *În fiecare lună sunt tipărite câte 5000 de exemplare din revista AutoMagazin la 2 tipografiei: una la Bucureşti şi una la Cluj-Napoca. De aici revistele sunt transportate la centrele regionale de distribuţie. Luna aceasta centrul din Braşov a comandat 4000 de exemplare, centrul de la Piteşti a cerut 2000 de reviste, iar centrul de distribuţie de la Timişoara a cerut 2500 de copii. Costurile de transport de la fieecare tipografiee la centrele de distribuţie sunt date în tabelul de mai jos:*

Tipografii	**Centre regionale de distribuție**		
	Brașov	**Pitești**	**Timișoara**
București	0,07	0,05	0,10
Cluj-Napoca	0,03	0,11	0,04

În calitate de manager de distribuţie, formulaţi un model matematic pentru a găsi cel mai ieftin mod de distribuţie.

8. *Compania "EuroElectro" produce ventilatoare, cuptoare şi uscătoare. pentru a le comercializa. Contribuţia netă pentru aceste produse este estimată să fie: pentru un ventilator – 210, pentru un cuprot – 170 şi pentru un uscător – 45. Fiecare din aceste produse necesită timp de procesare în departamentele de fabricaţie, asamblare şi testare, după cum este prezentat mai jos:*

Produs	Timp necesar de lucru (în ore) pe domeniu		
	Fabricaţie	Asamblare	Testare
Ventilator	3	3	1
Cuptor	4	2	3/4
Uscător	1	1/2	1/2

Totalul orelor disponibile pentru procesare în cele trei departamente în săptămânile următoare sunt estimate a fi: pentru fabricaţie – 400 ore, pentru asamblare – 350 ore şi pentru testare – 200 ore. Utilizând metoda Simplex, determinaţi graficul de producţie pentru săptămânile următoare care maximizează totalul contribuţiei nete.

9. *Compania "Petroleum" cumpără ţiţei din trei surse distincte. Costul, capacităţile şi caracteristicile celor trei surse sunt oarecum diferite, aşa cum este arătat mai jos:*

Sursa	Cost pe baril (u.m.)	Capacitate anuală (mii barili)	Nivel	% maxim de benzină
Soamer	43	12.000	70	45
Northslope	52	5.000	85	50
Mideast	60	20.000	90	55

"Petroleum" estimează că este necesar să cumpere 20 milioane de barili de ţiţei în timpul anului următor, şi doreşte să determine programul de cumpărare care are cel mai mic cost. Decizia care se ia se bazează pe necesitatea ca media nivelului de ţiţei cumpărat să fie mai mică decât 80 şi cantitatea totală de ţiţei trebuie să deţină cel puţin 50% benzină. În plus, compania este obligată, sub contract, să cumpere o cantitate minimă de 5 milioane de barili de "ţiţei Mideast". Determinaţi costul minim al programului de cumpărare.

10. *Compania "Oţel-Prim" care produce oţel, are două mine de exploatare a zăcămintelor de fier, situate în Minnesota şi Virginia de West. Este necesar un total de 50 000 tone de zăcământ de fier pe săptămână de la aceste mine. Capacităţile săptămânale ale minelor şi costurile pe tonă de livrare spre întreprinderile de prelucrare sunt date în tabelul de mai jos:*

Denumire mină	Capacitate (tone pe săptămână)	Cost (u.m la tonă)
Soroca	35.000	87
Comrat	32.000	69

Zăcământul de minereu de fier la întreprinderea de prelucrare trebuie să conţină cel puţin 45% fier şi cel mult 1.3% cupru. Procentajele de fier şi cupru ale zăcămintelor celor două mine sunt:

Denumire mină	% Fier	% Cupru
Soroca	43	1.1
Comrat	67	1.4

Formulaţi modelul de programare liniară şi determinaţi cantitatea de fier (aplicând algoritmul Simplex) de la fiecare mină astfel încât costurile să fie minime.

11. *Compania "Euro-Fertilization" care produce fertilizatori, consideră că este mai important producerea unui nou fertilizator care trebuie să conţină mai mult decât o cantitate exactă de nitrogen, potasiu şi fosfat pe kilogram. Combinaţia specificată conţine minimum 15% din fiecare din ingredientele mai sus menţionate. Compania consideră patru ingrediente cu care să realizeze produsul. În tabelul de mai jos se află procentele din kilogram al fiecărui element chimic necesar, precum şi costul pe kilogram al fiecărui ingredient.*

Cost /Kg	0,30	0,60	0,15	0,20
Potasiu	40	6	1	10
Nitrogen	2	20	4	6
Fosfat	5	3	30	15

Formulaţi modelul de programare liniară care va determina cantitatea minimă necesară din cele trei elemente prin combinarea celor patru ingrediente.

12. *Compania "SmartOil" cumpără petrol brut din 3 surse: Orientul Mijlociu, Marea Nordului şi Rusia. Compania doreşte să cumpere petrol care să deţină cel puţin 50% benzină, un procentaj de sulf de minim 60% şi indicatorul alfa să aibe cea mai mică valoare de 7. Costul de transport este dat în tabelul de mai jos.*

Sursa	Cost (u.m.)	% benzină	% sulf	Indicatorul α
E.A.U.	3	80	40	2
R.A.S.	5	60	60	4
Fed. Rusă	7	40	20	12

Determinaţi modul în care trebuie să realizeze achiziţiile compania astfel încât să-şi minimizeze costurile, ţinând cont de restricţiile considerate. Precizaţi costul total. Modul de rezolvare este următorul: Transcrieţi problema în forma standard; Rezolvaţi problema prin metoda algoritmului Simplex.

13. *Compania "BestHome" a contractat producerea de locuinţe într-o localitate. Compania trebuie să ia o decizie referitoare la angajarea în echipa de finisare a unuia sau mai multor ucenici de zugravi (8 ore pe zi) sau să angajeze unul sau mai mulţi zugravi cu normă redusă (5 ore pe zi). Pentru ucenici compania plăteşte 4 u.m. pe oră, în timp ce pentru zugravii angajaţi cu normă redusă, plăteşte 9 u.m. pe oră. Compania doreşte să cheltuiască suplimentar cel mult 26 u.m. pe zi şi să nu folosească mai mult de 17 ore suplimentare pe zi. Analiza profitului indică faptul că fiecare zugrav cu normă redusă va mări profitul cu 3 u.m. pe zi, iar fiecare ucenic de zugrav cu 4 u.m. pe zi. Câte persoane va angaja compania astfel încât profitul să fie maxim.*

14. *Administratorul unei herghelii de cai doreşte să stabilească un regim alimentar pentru caii pe care îi are în îngrijire. În acelaşi timp, doreşte să menţină costurile cu întreţinerea animalelor la un nivel cât mai mic. Mixurile alimentare disponibile pentru hrana cailor sunt pe bază de ovăz, grâu şi produse minerale. Fiecare din aceste mixuri conţin cantităţi diferite din 5 ingrediente necesare pentru hrana zilnică a animalelor. Tabelul următor conţine*

cantitatea minimă necesară din fiecare ingredient per kilogram de amestec, precum şi costurile celor 3 mixuri alimentare.

Ingrediente	Mixul alimentar în kg			
	Ovăz	Grâu	Minerale	Cantitate minimă
A	2	3	1	6
B	1/2	1	1/2	2
C	3	5	6	9
D	1	3/2	2	8
E	1/2	1/2	3/2	5
Cost pe Kg	9	14	17	

În plus, proprietarul grajdului este conştient că un cal hrănit prea mult se mişcă mai greu. În consecinţă, el consideră că 6 kg. de hrană pe zi sunt maximul pe care trebuie să le primească un cal pentru a funcţiona corect. Care este amestecul optim dintre aceste 3 mixuri alimentare?

15. *Un mic atelier de mobilă produce 3 tipuri de mese A, B şi C. Fiecare model necesită un anumit timp pentru tăierea părţilor component, pentru asamblare şi pentru vopsire. Atelierul poate să vândă toate produsele pe care le fabrică şi mai mult, poate vinde modelul C şi nevopsit. Atelierul are câţiva angajaţi care lucrează în regim part-time, astfel timpul disponibil variază de la o lună la alta. Folosiţi datele din tabelul de mai jos pentru a formula o problemă de programare liniară care va ajuta conducerea atelierului să determine ce cantitate din fiecare produs să fabrice pentru a-şi maximiza profitul.*

Model	Decupare (ore)	Asamblare (ore)	Vopsire (ore)	Profit unitar
A	3	4	5	25 $
B	1	2	5	20 $
C	4	5	4	50 $
C (nevopsit)	4	5	0	30 $
Capacitate	150	200	300	

Capitolul IV

ALGORITMI ÎN GRAFURI

De ce avem nevoie de algoritmii aplicați în grafuri ?

Grafurile sunt structuri discrete formate din vârfuri și noduri care conectează aceste vârfuri. În lumea reală, există multe probleme care pot fi reprezentate cu ajutorul grafurilor. Grafurile au o deosebită importanță în mai multe domenii, inclusiv informatică, matematică pură, cercetare operațională, biochimie, sociologie și alte științe.

Care este aplicabilitatea în informatică a algoritmilor în grafuri ?

- *Teoria grafurilor este utilizată pe scară largă în reprezentarea sistemului de rețea. Teoria grafurilor în rețele poate fi observată în două categorii: Reprezentare grafică (Topologie) și Teoria rețelelor. Topologia este modalitatea de a reprezenta o structură de rețea în diferite formate care pot ajuta la ușurarea problemei și la obținerea unor rezultate mai precise. Termenul de rețea și graf sunt similare, deoarece ambele se referă la topologie (structură) în care sunt aranjate vârfurile și muchiile. Unele topologii de bază sunt topologia stea, topologia inel, topologia magistrală, topologia mesh și topologia arborescentă. Termenul de teorie a rețelelor reprezintă diferitele metodologii de analiză a unui graf și aplicarea teoriei rețelelor folosind o topologie.*
- *Teoria grafurilor este utilizată pentru a modela procesul de proiectare a siteului web, în care paginile web sunt reprezentate prin vârfuri, iar hiperlegăturile dintre ele sunt reprezentate prin muchii în graf. Acest concept este cunoscut sub numele de graf web. În teoria grafurilor, un astfel de graf este numit graf bipartit complet. Reprezentarea grafică ajută la găsirea tuturor componentelor conectate și folosind graful direcționat putem evalua utilitatea site-ului web și structura link-urilor.*
- *Structura grafului are un rol important în proiectarea bazei de date, deoarece oferă un proces de implementare rapid. Utilizează o bază de date de tip graf, care este reprezentată cu noduri, muchii și proprietăți pentru a reprezenta și stoca date. Oferă un sistem de stocare cu o listă de adiacență fără index și are un instrument robust, cum ar fi interogare. În plus, este ușor să descrii relația dintre date în reprezentarea bazei de date sub formă de graf.*

- *Un sistem de operare este un program care acționează ca o interfață între utilizator și hardware-ul computerului. Scopul sistemului de operare este de a oferi un mediu în care un utilizator poate executa programe într-un mod eficient și convenabil. Teoria grafurilor joacă un rol important în sistemul de operare în rezolvarea problemelor de programare a locurilor de muncă și alocarea resurselor. Conceptul de colorare a grafurilor este aplicat în problemele de programare a lucrărilor CPU. Lucrările sunt presupuse ca vârfuri ale unui graf și va exista o margine între două lucrări care nu poate fi executată simultan. Grafurile sunt, de asemenea, utilizate în algoritmii de programare a discurilor.*
- *Structurarea sau organizarea datelor în informații, astfel încât operațiuni precum parcurgerea, căutarea, sortarea, îmbinarea, inserarea, ștergerea etc. să devină ușoară, Un astfel de model logic și matematic este numit „Structuri de date". Alegerea modelului de structurare a datelor depinde de doi factori: (1) trebuie să descrie relația reală între date și (2) structura trebuie să fie simplă și ușor de prelucrat datele în informații, dacă este necesar. Reprezentarea neliniară a datelor în memorie este posibilă folosind teoria grafurilor. Relația arbitrară între date este reprezentată de un graf și de matricea de adiacență a acestuia. Mulți algoritmi ale grafurilor necesită parcurgerea sistematică a nodurilor și marginilor acestuia. Există două moduri standard de a parcurge un graf: lățimea (BFS) și adâncime (DFS).*
- *Teoria grafurilor joacă un rol important în exploatarea datelor (data mining) ca exploatarea grafurilor (graph mining). Exploatarea grafului descrie aspectul relațional al datelor. Diferitele abordări ale exploatării grafului sunt categoriile de subgraf, izomorfismul subgrafului, măsurile de exploatare (mining), metodele de soluție și invarianții.*

4.1 Algoritmi pentru circuite

Partea 1
Circuit Eulerian

Un graf este o colecție de vârfuri sau noduri și margini (arce) între unele sau toate vârfurile. Când există o cale care traversează fiecare margine exact o dată, astfel încât calea începe și se termină la același vârf, calea este cunoscută sub numele de circuit eulerian și graful este cunoscut sub numele de graf eulerian. Eulerian se referă la matematicianul elvețian Leonhard Euler, care a inventat teoria grafueilor în secolul al XVIII-lea.

Graful în care muchia poate fi parcursă în ambele direcții se numește graf nedirecționat.

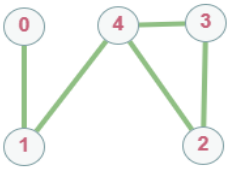
O cale euleriană este o cale din graf care vizitează fiecare margine exact o dată. Calea începe de la un vârf (nod) și trece prin toate marginile și ajunge la un alt nod la sfârșit. Există o demonstrație matematică care este folosită pentru a afla dacă calea euleriană este posibilă în graf sau nu, doar cunoscând gradul fiecărui vârf din graf.

Condițiile necesare sunt:

- *În primul rând, toate vârfurile cu grad > 0 ar trebui să fie în aceeași componentă (conectate).*
- *În al doilea rând, doar două margini trebuie să aibă un grad impar și toate marginile rămase trebuie să aibă grade pare.*

Să studiem cazul în detaliu. În primul rând, graful are întotdeauna un grad par, deoarece într-un graf nedirecționat, fiecare muchie adaugă 2 la gradul general al grafului. Deci, ar trebui să existe un număr par de vârfuri de nivel impar. În acest caz, să luăm în considerare graful cu vârful de doar 2 grade impare.

Să începem de la unul dintre vârfurile impar (de grade) și să trecem prin marginile rămase. La intrarea într-un vârf, gradul este redus cu 1, iar la ieșire, se reduce din nou cu 1. Deci, fiecare vârf intermediar trebuie să aibă un grad uniform. Nodurile de început și de sfârșit trebuie să facă una dintre cele două operații. Prin urmare, vor avea un grad ciudat.



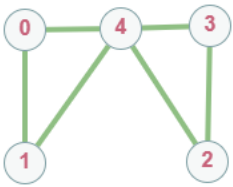
Mai sus este un exemplu de graf, pentru acest graf calea Euler este: 0-1-4-2-3-4.

Un circuit eulerian este o cale eulerianã în graf care începe și se termină la același vârf. Circuitul pleacă de la un vârf (nod) și trece prin toate muchiile și ajunge la același nod la sfârșit. Există, de asemenea, o dovadă matematică care este utilizată pentru a afla dacă un circuit eulerian este posibil în graf sau nu, doar prin cunoașterea gradului fiecărui vârf din graf.

Condițiile necesare sunt:

- *În primul rând, toate vârfurile cu grad > 0 ar trebui să fie în aceeași componentă conectate.*
- *În al doilea rând, toate marginile trebuie să aibă un grad uniform.*

Dovada este similară cu cea anterioară. Deci, să vedem implementarea pentru a afla dacă graful are o cale și un circuit Euler. Circuitul Euler pentru graful din imaginea din dreapta este: 0-1-4-2-3-4-0.



Teorema 1

Un graf va conține o cale Euler, dacă conține cel mult două vârfuri de grad impar. Un graf va conține un circuit Euler dacă toate vârfurile au grad egal.

Algoritm:

1. *Considerăm vârfurile cu grad > 0.*
2. *Verificăm dacă toate aceste vârfuri sunt conectate.*
3. *Privim și analizăm gradul tuturor vârfurilor:*
   - *Dacă toate vârfurile au grad par, Circuitul Euler este posibil.*
   - *Dacă doar 2 vârfuri au grad impar, Calea lui Euler este posibilă.*

Implementarea algoritmului în limbajul C++

```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;
vector<int> G[100];
ifstream fin("G1.txt");
int v,a,x,y;
// Functia de adaugarea arcului
void AdaugArc(){
    for(int i=0;i<a;i++){
        fin>>x>>y; // citim din fisierul G1
```

```
        G[x].push_back(y); G[y].push_back(x);
    }
}
// Algoritmul Depth-first search
void dfs(int curent, bool vizitat[]){
    vizitat[curent] = true;
    for(auto adj: G[curent])
        if(!vizitat[adj]) dfs(adj, vizitat);
}
//*****************************************
bool Conectat(int n, bool vizitat[]){
    int start=-1;
    for(int i=0; i<n; ++i){
        if(!vizitat[i] && G[i].size()>0){
            if(start==-1){
                dfs(i,vizitat); start=i;
            }
            else{
                cout<<"Graful nu este Eulerian!"<<endl;
                return false;
            }
        }
        vizitat[i]=true;
    }
    return true;
}
//*****************************************
void Verificare(int n){
    bool vizitat[n];
    fill(vizitat, vizitat+n, false);
    if(Conectat(n,vizitat)){
        int oddV = 0;   // Numararea varfurilor de grad impar
        for(int i=0; i<n; ++i)
            if(G[i].size()%2==1) oddV++;
        if(oddV == 0)
            cout<<"Graful este un circuit Euler!"<<endl;
        else if(oddV == 2)
            cout<<"Graful este o cale Euler!"<<endl;
        else
            cout<<"Graful nu este Euler!"<<endl;
    }
}
```

```
// Programul principal
int main(){
    cout<<"Citim numarul de varfuri si de arce din
fisier."<<endl;
    cout<<"Citim nodurile corespunzatoare arcelor grafului din
fisier."<<endl;
    fin>>v>>a; // citim din fisier
    cout<<"\nGraful are "<<v<<" varfuri si "<<a<<" arce!"<<en-
dl;
    AdaugArc(); Verificare(a);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Citim numarul de varfuri si de arce din fisier.
Citim nodurile corespunzatoare arcelor grafului din fisier.

Graful are 5 varfuri si 5 arce!
Graful este o cale Euler!
```

Fișierul de intrare G1.txt, include pe fiecare linie, separate prin spațiu următoarele perechi de numere: (5 5), (3 4), (4 2), (2 3), (4 1) și (1 0). Prima pereche de date reprezintă numărul de vârfuri și numărul de arce din graf.

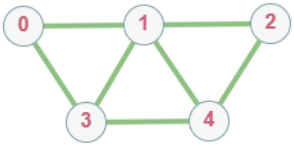
Cum ar trebui să fie modificat programul de mai sus pentru a afișa toate ciclurile Euleriene? Efectuați modificările necesare și afișați rezultatele.

Partea 2
Circuit Hamiltonian

Calea Hamiltoniană este un drum într-un graf direcționat sau nedirecționat care vizitează fiecare vârf exact o dată. Problema de a verifica dacă un graf (direcționat sau neorientat) conține o cale Hamiltoniană este NP-completă, așa este și problema găsirii tuturor Căilor Hamiltoniane într-un graf.

Ciclul sau circuitul hamiltonian este o cale hamiltoniană, deci există o muchie de la ultimul vârf la primul vârf.

Vom încerca să determinăm dacă un graf conține sau nu un ciclu hamiltonian. Și când este prezent un ciclu hamiltonian, imprimăm acest ciclu. Fie un graf G= (V,M), iar V={1,2,3,4,5} și M={(0 1),(1 2),(2 4),(4 1),(1 3), (0 3),(3 4)}. Să alcătuim pentru acest graf matricea de adiacență.



Matricea de adiacență este:

```
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 1 1 1 0
```

Exemple de căi Hamiltoniene:

a) `0 1 2 4 3 0`
b) `0 3 4 2 1 0`

Teorema 2

După Bellman și Held-Karp:
Calea hamiltoniană poate fi determinată (găsită) în timp $O(2^n)$ și complexitatea spațială este O(1).

Proprietăți:

1. *Orice ciclu hamiltonian poate fi convertit într-un drum hamiltonian prin eliminarea uneia din muchiile sale, dar un drum hamiltonian poate fi extins la un ciclu hamiltonian numai dacă extremitățile sale sunt adiacente.*
2. *Toate grafurile hamiltoniene sunt **biconexe**, dar un graf biconex nu este obligatoriu și hamiltonian (de exemplu, **graful Petersen**).*
3. *Un **graf eulerian** G (graf conex în care fiecare nod are grad par) are în mod necesar un ciclu eulerian, un drum închis care parcurge fiecare muchie din G exact o dată. Acest drum corespunde unui ciclu Hamiltonian în **graful linie** L(G), astfel încât graful linie al fiecărui graf eulerian este hamiltonian. Grafurile linie pot avea alte cicluri hamiltoniene care nu corespund ciclurilor euleriene, în special graful linie L(G) din orice graf hamiltonian G este în sine hamiltonian, indiferent dacă graful G este eulerian.*
4. *Un **graf turneu** (cu mai mult de două noduri) este hamiltonian **dacă și numai dacă** este **tare conex**.*

5. *Numărul de cicluri hamiltoniene diferite într-un graf neorientat complet de n noduri este (n–1)! /2 și într-un graf orientat complet de n noduri este (n–1)!. Aceste afirmații se bazează pe presupunerea că ciclurile care sunt aceleași cu excepția punctului de plecare nu sunt luate în calcul separat.*

Implementarea algoritmului în limbajul C++

```
#include <iostream>
#define N 5 // numarul de varfuri din graf
using namespace std;
void AfisareSolutie(int cale[]);
// Verificam siguranța, functia data este folosită pentru a ve-
rifica daca un varf este adiacent varfului adăugat anterior și
care deja nu a fost adaugat.
bool Sigur(int n,bool g[N][N],int cale[],int poz) {
    if (g[cale[poz-1]][n]==0) return false;
    for (int i=0;i<poz;i++){
        if (cale[i]==n) return false;
    }
    return true;
}
// Verificăm posibilitatea construirii unei cai Hamilton
bool TestCale(bool g[N][N],int cale[],int poz) {
    //Verificam daca toate varfurile sunt incluse
    if (poz==N) {
        if (g[cale[poz-1]][cale[0]]==1) return true;
        else return false;
    }
    for (int n=1;n<N;n++) {
        //Verificam daca varful dat poate fi adaugat
        if (Sigur(n,g,cale,poz)) {
            cale[poz]=n;
            //Repetam pentru a construi restul caii Hamilton
            if (TestCale(g,cale,poz+1)==true) return true;
            //Eliminam varful daca nu conduce la solutie
            cale[poz] = -1;
        }
    }
    return false;
}
// Construirea ciclului Hamiltonian
```

```
bool CicluHamiltonian(bool g[N][N]) {
    int *cale = new int[N];
    for (int i=0; i<N; i++)
    cale[i]=-1;
    // Punem varful 0 ca primul varf din cale. Daca exista un
ciclu Hamiltonian, atunci calea poate fi inceputa din orice
punct al ciclului, deoarece graful este nedirectionat.
    cale[0] = 0;
    if (TestCale(g,cale,1)==false){
        cout<<"\nCiclul Hamiltonian nu exista!"<<endl;
        return false;
    }
    AfisareSolutie(cale); return true;
}
void AfisareSolutie(int cale[]) {
    cout<<"Exista ciclu Hamiltonian!"<<endl;
    cout<<"Exemplu de ciclu: ";
    for (int i = 0; i < N; i++){
        cout<<cale[i]<<" ";
    }
    cout<<cale[0]<<endl;
}
// Programul principal
int main() {
    bool g[N][N] = { // Introducem matricea de adiacenta
        {0,1,0,1,0}, {1,0,1,1,1}, {0,1,0,0,1},
        {1,1,0,0,1}, {0,1,1,1,0}
    };
    CicluHamiltonian(g);
}
```

Soluția obținută în urma execuției codului C++ este:

```
Exista ciclu Hamiltonian!
Exemplu de ciclu: 0 1 2 4 3 0
```

Cum ar trebui să fie modificat programul de mai sus pentru a afișa toate ciclurile Hamiltoniene? Efectuați modificările necesare și afișați rezultatele.

4.2 Algoritmi de optimizare

Partea 1
Problema vânzătorului călător (TSP -The Travelling Salesman Problem)

Problema vânzătorului călător este provocarea de a găsi cea mai scurtă, dar cea mai eficientă rută pentru care o persoană să poată lua o listă de destinații specifice. Este o binecunoscută problemă algoritmică în domeniul informaticii și al cercetării operaționale. Există, evident, o mulțime de rute diferite de ales, dar găsirea celei mai bune - cea care va necesita cea mai mică distanță sau cost - este ceea ce matematicienii și informaticienii au petrecut decenii încercând să rezolve.

Problema vânzătorului călător a atras atât de multă atenție, deoarece este atât de ușor de descris, dar atât de dificil de rezolvat. De fapt, acest tip de problemă aparține clasei de probleme de optimizare combinatorie cunoscute sub numele de NP-complete. Aceasta înseamnă că problema noastră este clasificată ca NP-hard, deoarece nu are o soluție „rapidă" și complexitatea calculării celei mai bune rute va crește atunci când adăugați mai multe destinații la problemă.

Problema poate fi rezolvată prin analiza fiecărei rute dus-întors pentru a o determina pe cea mai scurtă. Cu toate acestea, pe măsură ce numărul destinațiilor crește, numărul corespunzător de dus-întors depășește capacitățile chiar și ale celor mai rapide computere. Cu 10 destinații, pot exista mai mult de 300.000 de permutări și combinații dus-întors. Cu 15 destinații, numărul de rute posibile ar putea depăși 87 de miliarde.

Soluții populare pentru problema vânzătorului călător[4]

1. *Abordarea Forței Brute, cunoscută și sub numele de Abordare Naivă, calculează și compară toate permutările posibile ale traseelor sau căilor pentru a determina cea mai scurtă soluție unică. Pentru a rezolva problema folosind abordarea Brute-Force, trebuie să calculați numărul total de rute și apoi să desenați și să enumerați toate rutele posibile. Calculați distanța fiecărui traseu și apoi alegeți-l pe cel mai scurt - aceasta este soluția optimă.*
2. *Metoda ramificației și legăturii (Branch & Bound) descompune o problemă care trebuie rezolvată în mai multe subprobleme. Este un sistem pentru rezolvarea unei serii de subprobleme, fiecare dintre acestea putând avea mai multe soluții posibile și în care soluția selectată pentru o problemă poate*

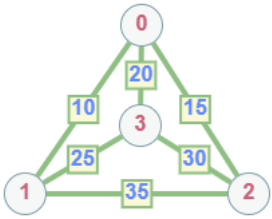
4 *Analizați* ***Anexa 3:*** *Metode de rezolvare a problemei vânzătorului călător pas cu pas.*

avea un efect asupra soluțiilor posibile ale subproblemelor ulterioare. Pentru a rezolva problea folosind metoda Branch & Bound, trebuie să alegeți un nod de pornire și apoi să setați legat la o valoare foarte mare (să spunem infinit). Selectați cel mai ieftin arc între nodul nevizitat și cel curent și apoi adăugați distanța la distanța curentă. Repetați procesul în timp ce distanța curentă este mai mică decât cea legată. Dacă distanța actuală este mai mică decât limita, ați terminat. Acum puteți adăuga distanța astfel încât legătura să fie egală cu distanța curentă. Repetați acest proces până când toate arcurile au fost acoperite.

3. *Metoda celui mai apropiat vecin. Cheia acestei metode este să vizitați întotdeauna cea mai apropiată destinație și apoi să vă întoarceți în primul oraș când sunt vizitate toate celelalte orașe. Pentru a rezolva problema folosind această metodă, alegeți un oraș aleatoriu, apoi căutați cel mai apropiat oraș nevizitat și mergeți acolo. După ce ați vizitat toate orașele, trebuie să vă întoarceți în primul oraș.*

Exemplu de problemă

Fie un set de orașe și distanța dintre fiecare pereche de orașe, problema este de a găsi cel mai scurt traseu posibil care să viziteze fiecare oraș exact o dată și să revină la punctul de plecare.



Rezolvare:
Observați diferența dintre ciclul Hamiltonian și problema vânzătorului călător (PVC). Problema ciclului Hamiltoninan este de a găsi dacă există un tur care vizitează fiecare oraș exact o dată.

Aici știm că Hamiltonian Tour există (deoarece graficul este complet) și, de fapt, există multe astfel de ture, problema este de a găsi o greutate minimă a ciclului Hamiltonian. De exemplu, luați în considerare graficul prezentat în figura din partea dreaptă. Un tur pentru PVC din graf este 1-2-4-3-1. Costul turului este de 10 + 25 + 30 + 15, adică 80.

***Soluție naivă** (Complexitatea timpului: Θ (n!))**:*
- *luați în considerare orașul 1 ca punct de plecare și de sfârșit;*
- *generați toate (n-1)! Permutații ale orașelor;*

- *calculați costul fiecărei permutări și țineți evidența permutării costurilor minime;*
- *întoarceți permutația cu un cost minim.*

Soluție cu implementarea programării dinamice:

Fie setul de vârfuri dat să fie {1, 2, 3, 4, ...n}. Să considerăm 1 ca punct de pornire și de sfârșit al ieșirii. Pentru fiecare alt vârf i (altul decât 1), găsim calea costului minim cu 1 ca punct de plecare, i ca punct final și toate vârfurile care apar exact o dată. Să fie costul acestei căi costul (i), costul ciclului corespunzător ar fi costul (i) + dist (i, 1) unde dist (i, 1) este distanța de la i la 1. În cele din urmă, returnăm minim dintre toate valorile [cost (i) + dist (i, 1)]. Acest lucru pare simplu până acum. Acum întrebarea este cum se obține costul (i)?

Pentru a calcula costul (i) utilizând programarea dinamică, trebuie să avem o relație recursivă în ceea ce privește subproblemele. Să definim un termen C (S, i) să fie costul căii costului minim care vizită fiecare vârf din mulțimea S exact o dată, începând cu 1 și terminând cu i. Începem cu toate subseturile de mărime 2 și calculăm C (S, i) pentru toate subseturile unde S este subsetul, apoi calculăm C (S, i) pentru toate subseturile S de dimensiunea 3 și așa mai departe. Rețineți că 1 trebuie să fie prezent în fiecare subset.

*Pentru un set de dimensiuni n, considerăm n-2 subseturi fiecare de dimensiunea n-1 astfel încât toate subseturile să nu aibă nth în ele. Folosind relația de recurență de mai sus, putem scrie o soluție bazată pe programare dinamică. Există cel mult O (n * 2^n) subprobleme și fiecare are nevoie de timp liniar pentru a rezolva. Prin urmare, timpul total de funcționare este O (n^2 * 2^n). Complexitatea timpului este mult mai mică decât O (n!), Dar totuși exponențială. Spațiul necesar este, de asemenea, exponențial. Deci, această abordare este, de asemenea, imposibilă chiar și pentru un număr ușor mai mare de vârfuri.*

Implementarea naivă a algoritmului în limbajul C++

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define V 4
// Implementarea algoritmului pentru PVC (minim)
int PVC(int graph[][V], int s){
    // Stocam toate varfurile in afara de varful sursa
    vector<int> varf;
    for (int i=0; i<V; i++){
```

```
        if (i!=s) varf.push_back(i);
    }
    // Depozitam greutatea minima a ciclului hamiltonian.
    int MinTraseu = INT_MAX;
    do {
        // Stocam greutatea traseului curent (GTC),adica costul
        int GTC=0;
        // Calculam greutatea curenta a traseului
        int k=s;
        for (int i=0; i<varf.size(); i++) {
            GTC+=graph[k][varf[i]]; k=varf[i];
        }
        GTC+=graph[k][s];
        // Actualizam valoarea minima
        MinTraseu = min(MinTraseu, GTC);
    }
    while (next_permutation(varf.begin(),varf.end()));
    return MinTraseu;
}
// Programul principal
int main(){
    // Reprezentam graful sub forma de array 2D
    int graph[][V] = { { 0, 10, 15, 20 },{ 10, 0, 35, 25 },
{ 15, 35, 0, 30 },{ 20, 25, 30, 0 } };
    int s=0;
    cout<<"Costul minim al traseului va fi: ";
    cout<<PVC(graph, s)<<" $."<<endl;
}
```

Soluția obținută în urma execuției codului C++ este:

```
Costul minim al traseului va fi: 80 $.
```

Cum ar trebui să fie modificat programul de mai sus pentru a afișa toate traseele de cost minim (sau maxim)? Efectuați modificările necesare și afișați rezultatele.

Partea 2
Problema colorării vârfurilor unui graf

Colorarea grafului nu este altceva decât un mod simplu de etichetare a componentelor acestuia, cum ar fi vârfurile, marginile și regiunile, sub anumite constrângeri. Într-un graf, nu există două vârfuri adiacente, margini adiacente sau regiuni adiacente, acestea trebuie să fie colorate cu un număr minim de culori. Acest număr se numește număr cromatic, iar graful se numește graf corect colorat. În timp ce colorarea grafului, constrângerile care sunt stabilite pe graf sunt culorile, ordinea de colorare, modul de atribuire a culorii etc. O colorare este dată unui vârf sau unei anumite regiuni. Astfel, vârfurile sau regiunile având aceleași culori formează seturi independente.

- ***Colorare Vîrf.*** *Colorarea vârfului este o atribuire a culorilor vârfurilor unui graf „G", astfel încât niciun vârf adiacent să nu aibă aceeași culoare. Pur și simplu, nu există două vârfuri ale unei margini care să aibă aceeași culoare.*
- ***Număr cromatic.*** *Numărul minim de culori necesar pentru colorarea vârfurilor grafului „G" este numit numărul cromatic al lui G, notat cu X (G).*

X (G) = 1 dacă și numai dacă „G" este un graf nul. Dacă „G" nu este un graf nul, atunci X(G) ≥ 2.

Soluții populare pentru problema colorării vârfurilor unui graf

1. *Abordare naivă: Generați toate configurațiile posibile de culori. Deoarece fiecare nod poate fi colorat folosind oricare dintre cele m culori disponibile, numărul total de configurații de culoare posibile este m^V. După generarea unei configurații ale culoreilor, verificați dacă vârfurile adiacente au sau nu aceeași culoare. Dacă sunt îndeplinite condițiile, imprimați combinația și întrerupeți bucla.*
2. *Backtracking: Ideea este de a atribui culori unul câte unul la vârfuri diferite, începând de la vârful 0. Înainte de a atribui o culoare, verificați siguranța luând în considerare culorile deja atribuite vârfurilor adiacente, adică verificați dacă vârfurile adiacente au sau nu aceeași culoare . Dacă există o atribuire de culoare care nu încalcă condițiile, marcați atribuirea de culoare ca parte a soluției. Dacă nu este posibilă nicio atribuire de culoare, retrogradează și returnează false.*
3. *Utilizarea BFS (Breadth First Search): Abordarea aici este de a colora fiecare nod de la 1 la n inițial cu culoarea 1. Și începe să călătoriți BFS de la un nod de pornire nevizitat pentru a acoperi toate componentele conectate dintr-o*

singură dată. Când ajungeți la fiecare nod în timpul traversării BFS, efectuați următoarele:

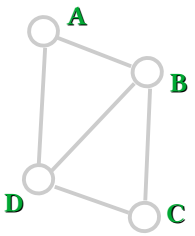
- *verificați toate marginile nodului dat;*
- *pentru fiecare vârf conectat la nodul nostru printr-o margine:*
  - *verificați dacă culoarea nodurilor este aceeași, dacă este același, măriți culoarea celuilalt nod (nu curentul) cu unul;*
  - *verificați dacă a fost vizitat sau nevizita, dacă nu este vizitat, marcați-l ca vizitat și puneți-l într-o coadă;*
- *verificați starea maxColors până acum, dacă depășește o valoare M, returnează fals.*

După ce ați vizitat toate nodurile, returnați adevărat (deoarece nu a putut fi găsită nicio condiție de încălcare în timpul călătoriei).

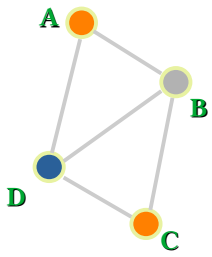
4. *Greedy: Din păcate, nu există un algoritm eficient disponibil pentru colorarea unui graf cu un număr minim de culori, deoarece problema este o problemă NP Completă cunoscută. Există totuși algoritmi aproximativi pentru a rezolva problema. Mai jos este algoritmul Greedy de bază pentru a atribui culori. Nu garantează utilizarea de culori minime, dar garantează o margine superioară a numărului de culori. Algoritmul de bază nu folosește niciodată mai mult de d+1 culori, unde d este gradul maxim al unui vârf din graf:*
   - *colorează primul vârf cu prima culoare, procedăm astfel și pentru următoarele V-1 vârfuri rămase;*
   - *luăm în considerare vârful ales în prezent și-l colorăm cu valoarea cea mai mică numerotată pentru culoare care nu a fost utilizată anterior pentru vârfurile colorate adiacente acestuia;*
   - *dacă toate culorile folosite anterior apar pe vârfurile adiacente lui v, atunci îi vom atribui o nouă culoare.*

Exemplu de problemă

Fie un grup de prieteni reprezentat prin următorul graf. Poți fi un prieten cu cineva, doar dacă există între voi o relație de prietenie (prietenii sunt uniți printr-un arc). Fiecare persoană din grupul vrea să poarte o căciulă de o anumită culoare. De ce culoare ar trebui să fie căciula fiecărei persoane, dacă fiecare prieten trebuie să aibă o căciulă de culoare diferită față de ceilalți prieteni ai săi și numărul de culori folosite trebuie să fie minim.



Rezolvare:



Pașii necesari pentru a colora un graf G cu n vârfuri sunt după cum urmează:

1. *aranjați vârfurile grafului într-o anumită ordine;*
2. *alegeți primul vârf și colorați-l cu prima culoare;*
3. *alegeți următorul vârf și colorați-l cu cea mai mică numerotată culoare care nu a fost colorată pe niciun vârf adiacent acestuia. Dacă toate vârfurile adiacente sunt colorate cu această culoare, atribuiți-i o nouă culoare. Repetați acest pas până când toate vârfurile sunt colorate.*

În figura de mai sus, la primul vârf A este colorat în orange. Deoarece vârfurile adiacente ale vârfului A sunt din nou adiacente, vârful B și vârful D sunt colorate cu culori diferite, respectiv gri și albastru. Apoi, vârful C este colorat la fel de orange, deoarece niciun vârf adiacent al lui c nu este colorat în orange.
Prin urmare, am putea colora graful nostru cu 3 culori. Prin urmare, numărul cromatic al grafului G este 3.

***Soluție naivă** (Complexitatea timpului: $\Theta(m^V)$):*

1. *Creați o funcție recursivă care ia indicele curent, numărul de vârfuri și array-ul de culori de ieșire.*
2. *Dacă indicele curent este egal cu numărul de vârfuri, atunci efectuați dacă configurația culorii de ieșire este sigură, adică verificați dacă vârfurile adiacente nu au aceeași culoare. Dacă sunt îndeplinite condițiile, tipăriți configurația și întrerupeți.*
3. *Atribuiți o culoare unui vârf (de la 1 la m).*
4. *Pentru fiecare culoare atribuită apelați recursiv funcția cu următorul index și numărul de vârfuri.*
5. *Dacă orice funcție recursivă returnează adevărat, atunci întrerupeți bucla și returnați adevărat.*

***Soluție backtracking** (Complexitatea timpului: $\Theta(m^V)$):*

1. *Creați o funcție recursivă care ia graful, indicele curent, numărul de vârfuri și array-ul de culori de ieșire.*
2. *Dacă indicele curent este egal cu numărul de vârfuri, atunci imprimați configurația culorii în array-ul de ieșire.*
3. *Atribuiți o culoare unui vârf (de la 1 la m).*
4. *Pentru fiecare culoare atribuită, verificați dacă configurația este sigură (adică verificați dacă vârfurile adiacente nu au aceeași culoare) apelați recursiv funcția cu următorul index și numărul de vârfuri.*

5. *Dacă orice funcție recursivă returnează adevărat, întrerupeți bucla și reveniți la adevărat (returnați true).*
6. *Dacă nicio funcție recursivă nu returnează true, atunci returnează false.*

Implementarea Greedy a algoritmului în limbajul C++

```
#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>
using namespace std;
int v,a,i,j;
vector <vector<int>> G; // Graful
vector <int> C; // Culoare
bool vis[100011];
ifstream fin("arce.txt");
// Algoritmul Greedy de rezolvare a problemei
void ColorVarfGreedy(){
    C[0]=0;
    for (i=1;i<v;i++){
        C[i]=-1;
    }
    bool VN[v]; // Varf nevizitat
    for (i=0;i<v;i++){
        VN[i]=0;
    }
    for (i=1;i<v;i++){
        for (j=0;j<G[i].size();j++)
            if (C[G[i][j]]!=-1) VN[C[G[i][j]]]=true;
        int cr;
        for (cr=0;cr<v;cr++)
            if (VN[cr] == false) break;
        C[i] = cr;
        for (j=0;j<G[i].size();j++)
            if (C[G[i][j]] != -1) VN[C[G[i][j]]]=false;
    }
}
int main(){
    int x,y;
    cout<<"Citim numarul de varfuri si de arce din
fisier."<<endl;
    fin>>v>>a; // citim din fisier
```

```
    G.resize(v); C.resize(v); memset(vis,0,sizeof(vis));
    cout<<"Citim varfurile celor "<<a<<" arce (muchii) din fi-
sier."<<endl;
    for(i=0;i<a;i++){
        fin>>x>>y; // citim din fisier
        x--; y--; G[x].push_back(y); G[y].push_back(x);
    }
    cout<<"\nApelam subprogramul creat pentru colorarea varfu-
rilor: "<<endl;
    ColorVarfGreedy(); // Apelam subprogramul creat
    for(i=0;i<v;i++){
       cout<<"\tVarful "<<i+1<<" este colorat cu "<<C[i]+1;
       cout<<endl;
    }
}
```

Explicația programului

1. *Utilizatorul trebuie să introducă mai întâi numărul de vârfuri, v, apoi numărul de muchii, a, din graf, apoi ar trebui să fie urmat de linii a, care notează punctele A și B, dacă există un arc între punctele A și B.*
2. *Graficul este stocat ca listă de adiacență, apoi tuturor vârfurilor li se atribuie culori conform algoritmului Greedy, verificând care este cel mai bine posibil să fie atribuit în acel moment.*

Soluția obținută în urma execuției codului C++ este:

```
Citim numarul de varfuri si de arce din fisier.
Citim varfurile celor 5 arce (muchii) din fisier.

Apelam subprogramul creat pentru colorarea varfurilor:
        Varful 1 este colorat cu 1
        Varful 2 este colorat cu 2
        Varful 3 este colorat cu 1
        Varful 4 este colorat cu 3
```

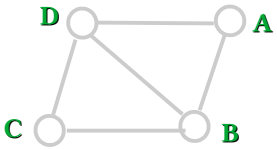
Fișierul de intrare arce.txt, include pe fiecare linie, separate prin spațiu următoarele perechi de numere: (4 5), (1 2), (1 4), (2 3), (2 4) și (3 4).

Partea 3
Problema colorării arcelor unui graf

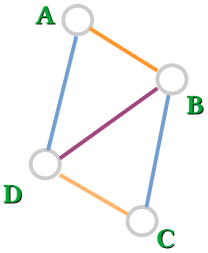
În teoria grafurilor, o colorare a arcului unui graf este o atribuire de „culori" la arcul grafului, astfel încât două arce adiacente să nu aibă aceeași culoare. Problema este foarte asemănătoare cu cea de colorare a vârfurilor unui graf.

Exemplu de problemă

Fie un număr de orașe și drumul național ce unește o pereche de orașe. Găsi o modalitate de a forma un număr minim de grupe care va trasa marcajul rutier în oraș, astfel încât drumurile ce vor fi marcate de o echipă de muncitori să nu fie adiacente. Care este numărul minim e echipe?



Rezolvare:



Soluția la problemă:

1. *orice două arce conectate la același vârf vor fi adiacente;*
2. *luăm un vârf și oferim culori diferite, tuturor marginilor conectate, eliminăm aceste margini din graf (sau le marcăm ca colorate);*
3. *traversăm una dintre marginile sale și repetăm pasul 2 cu noul vârf.*

Implementarea BFS a algoritmului în limbajul C++

```
#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>
#include <queue>
#include <set>
using namespace std;
ifstream fin("arce.txt");
int v,a,i,j;
vector<vector<pair<int,int> > > G; // Graful
vector<int> C; // Culoare
```

```
bool vis[100011];
// Algoritmul Greedy de rezolvare a problemei
void ColorArceBFS(int nod){
	queue<int> q; int c=0;
	set<int> AmColorat;
	if(vis[nod])return;
	vis[nod]=1;
	for(i=0;i<G[nod].size();i++){
		if(C[G[nod][i].second]!=-1){
			AmColorat.insert(C[G[nod][i].second]);
		}
	}
	for(i=0;i<G[nod].size();i++){
		if(!vis[G[nod][i].first]){
			q.push(G[nod][i].first);
		}
		if(C[G[nod][i].second]==-1){
			while(AmColorat.find(c)!=AmColorat.end())
			c++; C[G[nod][i].second]=c;
			AmColorat.insert(c);
		}
	}
	while(!q.empty()){
		int temp=q.front(); q.pop(); ColorArceBFS(temp);
	}
	return;
}
// Programul principal
int main(){
    int x,y; set<int> empty;
    cout<<"Citim numarul de varfuri si de arce din
fisier."<<endl;
    fin>>v>>a; // citim din fisier
    G.resize(v); C.resize(a,-1);
    cout<<"Citim varfurile celor "<<a<<" arce (muchii) din fi-
sier."<<endl;
    memset(vis,0,sizeof(vis));
    for(i=0;i<a;i++){
        fin>>x>>y; // citim din fisier
        x--; y--;
        G[x].push_back(make_pair(y,i));
        G[y].push_back(make_pair(x,i));
```

```
    }
    cout<<"\nApelam subprogramul creat pentru colorarea arce-
lor: "<<endl;
    ColorArceBFS(0); // Apelam subprogramul creat
    for(i=0;i<a;i++){
       cout<<"\tArcul "<<i+1<<" este colorat cu "<<C[i]+1<<"\
n";
    }
}
```

Explicația programului

1. *Utilizatorul trebuie să introducă mai întâi numărul de vârfuri, v, apoi numărul de muchii, v, din graf, apoi ar trebui să fie urmat de linii a (arce), care notează punctele A și B, dacă există un arc între puctele A și B.*
2. *Graful este stocat ca listă de adiacență, Apoi BFS este implementat folosind coada și culorile sunt alocate fiecărui arc.*
3. *Au fost folosite numere în locul culorilor, pentru simplitatea codului sursă.*

Soluția obținută în urma execuției codului C++ este:

```
Citim numarul de varfuri si de arce din fisier.
Citim varfurile celor 5 arce (muchii) din fisier.

Apelam subprogramul creat pentru colorarea arcelor:
        Arcul 1 este colorat cu 1
        Arcul 2 este colorat cu 2
        Arcul 3 este colorat cu 2
        Arcul 4 este colorat cu 3
        Arcul 5 este colorat cu 1
```

Fișierul de intrare arce.txt, include pe fiecare linie, separate prin spațiu următoarele perechi de numere: (4 5), (1 2), (1 4), (2 3), (2 4) și (3 4).

Cum ar trebui să fie modificat programul de mai sus pentru a afișa toate modalitățile de colorare a vârfurilor și a arcurilor? Efectuați modificările necesare și afișați rezultatele.

4.3 Algoritmi de aproximare

Partea 1
Arbori minimi de acoperire (AMA) - algoritmul Kruskal

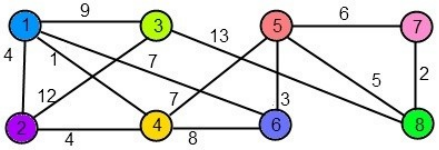
Găsirea unui arbore minim de acoperire pentru un graf are aplicaţii în domenii cât se poate de variate:

1. *Reţele (de calculatoare, telefonie, cablu TV, electricitate, drumuri): se doreşte interconectarea mai multor puncte, cu un cost redus şi atunci este utilă cunoaşterea arborelui care conectează toate punctele, cu cel mai mic cost posibil. STP(Spanning Tree Protocol) este un protocol de rutare care previne apariţia buclelor într-un LAN, şi se bazează pe crearea unui arbore de acoperire. Singurele legături active sunt cele care apar în acest arbore, iar astfel se evită buclele.*
2. *Segmentarea imaginilor: împărţirea unei imagini în regiuni de pixeli cu proprietăţi asemănătoare. E utilă mai apoi în analiza medicală a unei zone afectate de o tumoare de exemplu.*
3. *Algoritmi de aproximare pentru probleme NP-dure: problema comis-voiajorului, arbori Steiner.*
4. *Clustering: pentru detectarea de clustere cu forme neregulate.*

Algoritmul a fost dezvoltat în 1956 de Joseph Kruskal. Determinarea arborelui minim de acoperire se face prin reuniuni de subarbori minimi de acoperire. Iniţial, se consideră că fiecare nod din graf este un arbore. Apoi, la fiecare pas se selectează muchia de cost minim care uneşte doi subarbori disjuncţi, şi se realizează unirea celor doi subarbori. Muchia respectivă se adaugă la mulţimea muchiilor din arborele minim de acoperire.

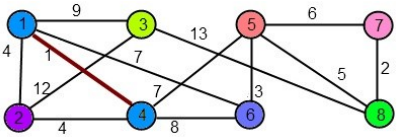
Exemplu de problemă

Se consideră graful din figura următoare:

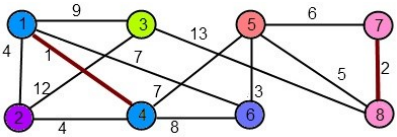


Fiecare subarbore va fi colorat diferit. Cum iniţial fiecare nod reprezintă un subarbore, nodurile au culori diferite. Pe măsură ce subarborii sunt uniţi, nodurile aparţinând aceluiaşi subarbore vor fi coloraţi identic. Costurile muchiilor sunt sortate în ordine crescătoare[5].

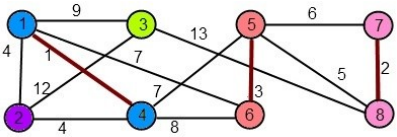
Pas 1: *Se alege prima muchie, (1,4). Se observă că uneşte subarborii {1} şi {4}, deci muchia e adăugată la MuchiiAMA, iar cei doi subarbori se unesc.*



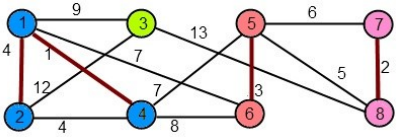
Pas 2: *Următoarea muchie este (7,8), care uneşte {7} şi {8}. Se adaugă la MuchiiAMA şi se unesc cei doi subarbori.*



Pas 3: *Următoarea muchie este (5,6), care uneşte {5} şi {6}. Se adaugă la MuchiiAMA şi se unesc cei doi subarbori.*

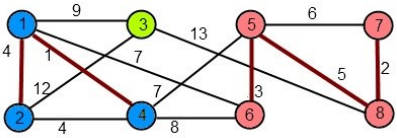


Pas 4: *Următorul cost este 4. Se observă că muchiile (1,2) şi (2,4) au costul 4 şi unesc {2} cu {1,4}. Se adaugă la MuchiiAMA una dintre cele două muchii, fie ea (1,2), şi se unesc cei doi subarbori. Alegerea muchiei (2,4) va duce la găsirea unui alt AMA.*



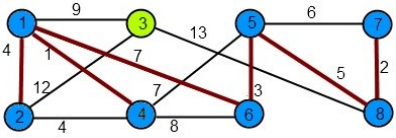
5 *Aici se poate analiza pas cu pas în mod vizual algoritmul Kruskal: https://www.cs.usfca.edu/~galles/visualization/Kruskal.html*

Pas 5: *Următoarea muchie de cost minim este (5,8), care unește {5,6} și {7,8}. Se adaugă la MuchiiAMA și se unesc cei doi subarbori, rezultând {5,6,7,8}.*



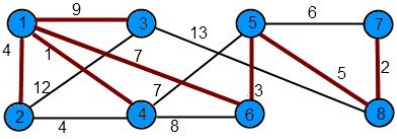
Pas 6: *Muchia (5,7), care are cel mai mic cost actual, are ambele extremități în subarborele {5,6,7,8}. În consecință, nu se efectuează nicio schimbare. MuchiiAMA = {(1,4),(7,8),(5,6),(1,2),(5,8)}.*

Pas 7: *Următorul cost este 7. Se observă că muchiile (1,6) și (4,5) au costul 7 și unesc subarborii {1,2,4} și {5,6,7,8}. Se adaugă la MuchiiAMA (1,6), și se unesc cei doi subarbori. Alegerea muchiei (4,5) va duce la găsirea unui alt AMA.*



Pas 8: *Muchia (4,6) de cost 8 are capetele în același subarbore, deci nu se produc schimări. MuchiiAMA = {(1,4),(7,8),(5,6),(1,2),(5,8),(1,6)}.*

Pas 9: *Muchia (1,3) de cost 9 unește cei doi subarbori rămași, {1,2,4,5,6,7,8} și {3}. Deci după unire obținem un singur arbore. (1,3) se adaugă la MuchiiAMA, care va conține acum 7 muchii, iar algoritmul se oprește.*



Arborele minim de acoperire obținut este {(1,4), (7,8), (5,6), (1,2), (5,8), (1,6), (1,3)}. Costul său se calculează însumând costurile tuturor muchiilor: Cost(MuchiiAMA) = 1 + 2 + 3 + 4 + 5 + 7 + 9 = 31.

Alți arbori minimi de acoperire pentru exemplul propus sunt:

- *{(1,4), (7,8), (5,6), (1,2), (5,8), (4,5), (1,3)};*
- *{(1,4), (7,8), (5,6), (2,4), (5,8), (1,6), (1,3)};*
- *{(1,4), (7,8), (5,6), (2,4), (5,8), (4,5), (1,3)}.*

Implementarea algoritmului în limbajul C++

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
#include <fstream>
using namespace std;
ifstream fin("arce.txt");
typedef  pair<int, int> pereche;
int V,E,u,v,w;
// structura grafului
struct Graph{
    int V, E;// varfuri si arce
    vector< pair<int, pereche> > arc;
    Graph(int V,int E){
        this->V=V;this->E=E;
    }
    // Nodul de inceput - nodul de sfarsit - costul
    void AdaugArc(int u,int v,int w){
        arc.push_back({w,{u,v}});
    }
    // Algoritmul Kruskal
    int KruskalAMA();
};
// Structura multimii de arce
struct MultimeDisjuncta{
    int *parinte,*grad,n;
    MultimeDisjuncta(int n){
        this->n=n; parinte=new int[n+1];
        grad=new int[n+1];
        for (int i=0;i<=n;i++){
            grad[i]=0; parinte[i]=i;
        }
    }
    // Gasim perechea potrivita
    int Gasit(int u){
        if (u!=parinte[u]) parinte[u]=Gasit(parinte[u]);
        return parinte[u];
    }
    // Combinam arcele
    void Combin(int x,int y){
        x=Gasit(x),y=Gasit(y);
        if (grad[x]>grad[y]) parinte[y]=x;
        else parinte[x]=y;
        if (grad[x]==grad[y]) grad[y]++;
    }
};
```

```
int Graph::KruskalAMA(){
    int CostAMA = 0;
    sort(arc.begin(), arc.end());
    MultimeDisjuncta dis(V);
    vector< pair<int, pereche> >::iterator it;
    for (it=arc.begin(); it!=arc.end(); it++){
        int u=it->second.first,v=it->second.second;
        int u1=dis.Gasit(u),v1=dis.Gasit(v);
        if (u1!=v1){
            cout<<u<<"-"<<v<<"   ";
            CostAMA+=it->first; dis.Combin(u1,v1);
        }
    }
    return CostAMA;
}
//Programul principal
int main(){
    // Citim din fisier numarul de varfuri si de arce
    cout<<"Citim numarul de varfuri si de arce din
fisier."<<endl;
    fin>>V>>E; // varfuri si arce
    Graph g(V,E); // construim structura grafului
    // Adaugam arcele si costurile acestora
    cout<<"Citim varfurile arcelor si costurile din
fisier."<<endl;
    for(int i=0;i<E;i++){
        fin>>u>>v>>w; g.AdaugArc(u,v,w);
    }
    cout<<"\nArcele arborelui minim de acoperire sunt: "<<endl;
    int CostAMA=g.KruskalAMA();
   cout<<"\n\nCostul arborelui minim de acoperire este: "<<Cos-
tAMA;
}
```

Soluția obținută în urma execuției codului C++ este:

```
Citim numarul de varfuri si de arce din fisier.
Citim varfurile arcelor si costurile din fisier.
Arcele arborelui minim de acoperire sunt:
1-4   7-8   5-6   1-2   5-8   1-6   1-3
Costul arborelui minim de acoperire este: 31
```

Partea 2
Debitul maxim - algoritmul Ford-Fulkerson

Imaginați-vă că vi se dă sarcina de a găsi o soluție eficientă pentru a produce programe aeriene pentru mii de rute în fiecare zi. Este posibil să trebuiască să luați în considerare mai mulți factori, cum ar fi utilizarea echipamentului, alocarea echipajului, satisfacția clienților, unele condiții meteorologice imprevizibile, defecțiuni etc. Asemenea problemă pare a fi foarte complicat de a-i găsi soluția, însă puteți să vă gândiți la aceasta ca la o problemă de debit maxim și să o rezolvați folosind metoda (algoritmul) Ford-Fulkerson.

Cu toate acestea, modelarea acestei probleme vă va oferi o înțelegere cuprinzătoare a metodei Ford-Fulkerson, astfel încât să o puteți utiliza oriunde este cazul. Dar înainte de asta, să înțelegem problema debitului maxim.

Metoda Ford-Fulkerson, cunoscută și sub numele de „algoritm de creștere a căii" este o abordare eficientă pentru a rezolva problema debitului maxim. Metoda Ford-Fulkerson depinde de două concepte principale și acestea sunt: Rețea reziduală și Creșterea căilor.

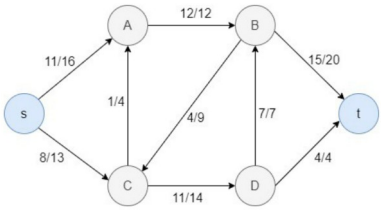
Etapa 1 a algoritmului

Este un graf, care are aceleași vârfuri ca și rețeaua originală, cu una sau două muchii pentru fiecare muchie din rețeaua originală. Și indică fluxul suplimentar posibil prin rețea. Pentru fiecare margine, vom calcula debitul suplimentar după cum urmează: Flux = capacitatea marginii - fluxul marginii.

Deci, pentru a crea o rețea reziduală, vom lua rețeaua noastră originală și vom actualiza capacitatea fiecărei margini cu fluxul suplimentar corespunzător acelei margini. Apoi vom adăuga și margini inverse pentru a indica cantitatea de flux care trece în prezent peste marginile rețelei originale.

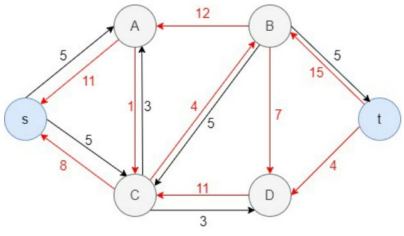
Exemplu:

Pentru a face acest lucru mai clar, să ne uităm la următorul graf.



Pentru fiecare muchie, prima valoare reprezintă valoarea debitului, iar a doua reprezintă capacitatea muchiei.
De exemplu, să luăm în considerare muchia CD. Aici debitul este 11 și capacitatea marginii este 14. Deci, dacă calculăm debitul suplimentar pentru această margine, obținem 14-11 = 3.

Deci, în rețeaua reziduală, putem actualiza capacitatea muchiei CD ca 3 și apoi adăugați o muchie inversă cu valoarea 11 (valoarea debitului rețelei originale) între C și D. De asemenea, putem repeta acest lucru pentru toate marginile și iată ce obținem ca rețea reziduală.



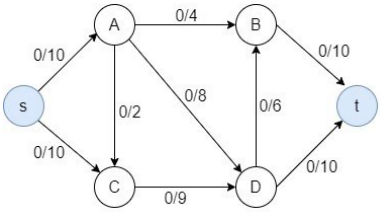
Etapa 2 a algoritmului

Având în vedere o rețea de curgere G=(V, E), calea de creștere este o cale simplă de la s la t în graful rezidual corespunzător al rețelei de curgere.

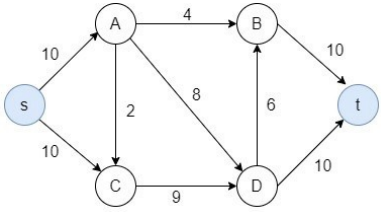
Metoda Ford-Fulkerson:

- *inițial, setați fluxul „f" al fiecărei margini la 0;*
- *în timp ce există o cale de creștere „p" în rețeaua reziduală măriți fluxul „f" de-a lungul lui „p";*
- *returați fluxul „f".*

Pasul 1: *Setați fluxul fiecărei margini la 0.*

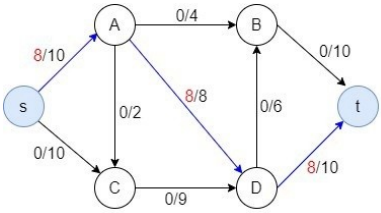


Rețeaua de flux

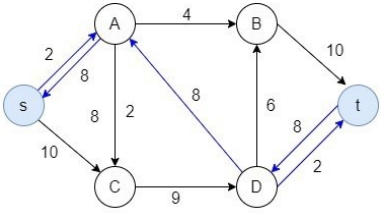


Rețeaua reziduală

Pasul 2: *Acum, găsiți o cale de creștere în rețeaua reziduală. Aici, selectez calea s -> A -> D -> t. Apoi trebuie să identificăm capacitatea de blocaj (adică debitul maxim pentru acea cale) pentru calea selectată. După cum puteți vedea, de-a lungul acestei căi, capacitatea de blocaj este de 8. Acum, fără a încălca constrângerile de capacitate, actualizați valorile de curgere ale marginilor din calea de creștere. Apoi veți obține următoarea rețea de flux și rețeaua reziduală.*

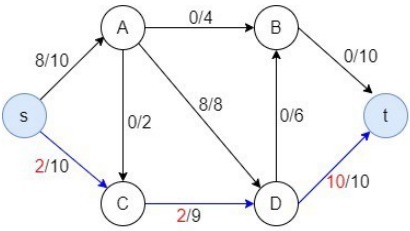


Rețeaua de flux

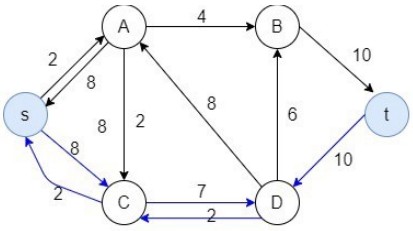


Rețeaua reziduală

Pasul 3: *Apoi selectez calea de creștere s -> C -> D -> t. Acum capacitatea de blocaj este de 2.*

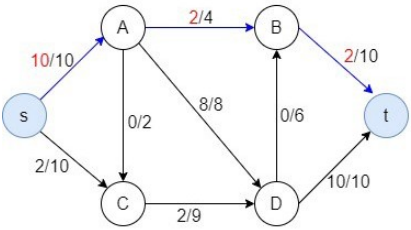


Rețeaua de flux

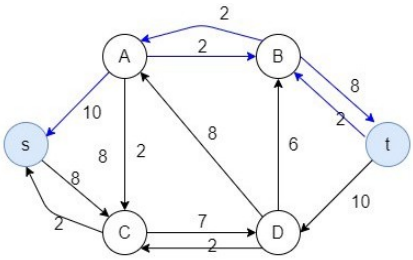


Rețeaua reziduală

Pasul 4: *Calea de creștere s -> A -> B->t, capacitatea de blocaj este 2.*

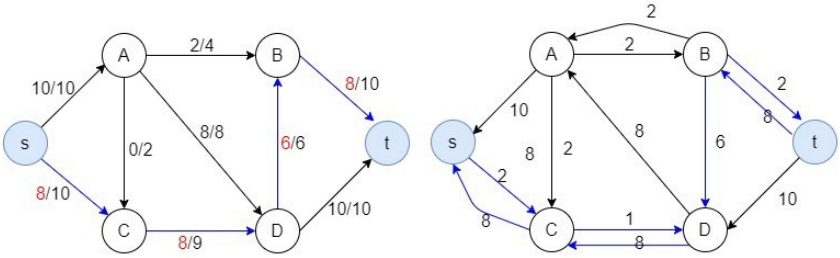


Rețeaua de flux

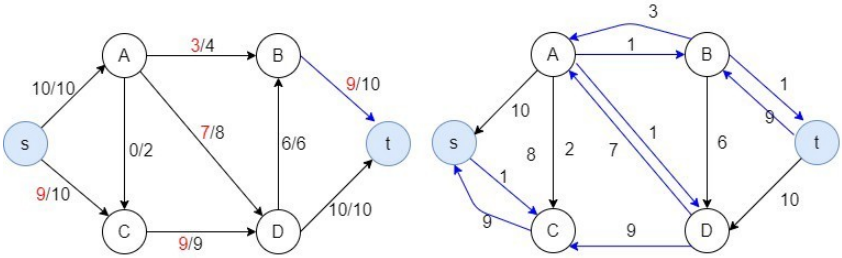


Rețeaua reziduală

Pasul 5: *Calea de creștere s -> C -> D -> B->t, capacitatea de blocaj este de 6.*



Pasul 6: *Calea de creștere s -> C -> D -> A-> B->t, capacitatea de blocaj este 1.*



Acum nu există căi rămase de la s la t în graful rezidual. Deci, nu există posibilitatea de a adăuga flux. Aceasta înseamnă că metoda Ford-Fulkerson este completă și suntem gata să găsim debitul maxim. Deoarece debitul maxim este egal cu debitul care iese din sursă, în acest exemplu, debitul maxim este 10+9 = 19.

Implementarea algoritmului în limbajul C++

```
#include <iostream>
#include <climits>
#include <cstring>
#include <queue>
using namespace std;
#define V 6 // numarul de noduri din graf
// Aplicam BFS ca si algoritm de cautare
bool BFS(int rG[V][V],int s,int t,int parinte[]) {
```

```
    bool vizitat[V]; memset(vizitat,0,sizeof(vizitat));
    queue<int> q; q.push(s); vizitat[s]=true; parinte[s]=-1;
    while (!q.empty()) { // cat timp nu este coada goala
        int u=q.front(); q.pop();
        for (int v=0;v<V;v++){
            if (vizitat[v]==false && rG[u][v]>0){
                q.push(v); parinte[v]=u; vizitat[v]=true;
            }
        }
    }
    return (vizitat[t]==true);
}
// Aplicam algoritmul Ford-Fulkerson
int FordFulkerson(int g[V][V], int s, int t){
    int u,v,rG[V][V];
    for (u=0;u<V;u++){
        for (v=0;v<V;v++){
            rG[u][v]=g[u][v];
        }
    }
    int parinte[V],FluxMaxim = 0;
    // Actualizarea valorilor reziduale ale muchiilor
    while (BFS(rG,s,t,parinte)) {
        int FluxCale=INT_MAX;
        for (v=t; v!=s; v=parinte[v]) {
            u=parinte[v];FluxCale=min(FluxCale, rG[u][v]);
        }
        for (v=t; v!=s; v=parinte[v]) {
            u=parinte[v]; rG[u][v]-=FluxCale;
            rG[v][u]+=FluxCale;
        }
        // Adaugarea fluxurilor de cale
        FluxMaxim+=FluxCale;
    }
    return FluxMaxim;
}
// Programul principal
int main() {
    // matriciea retelei reziduale
    int g[V][V] = {
        //S A B C D T - nodurile grafului
        {0,10,0,10,0,0},
```

```
        {0,0,4,2,8,0},
        {0,0,0,0,0,10},
        {0,0,0,0,9,0},
        {0,0,6,0,0,10},
        {0,0,0,0,0,0}
    };
    cout<<"Fluxul maxim este: "<<FordFulkerson(g,0,5)<<endl;
}
```

Soluția obținută în urma execuției codului C++ este:

```
Fluxul maxim este: 19
```

Aplicații Ford-Fulkerson:

- *Conducta de distributie a apei;*
- *Problemă de potrivire bipartită;*
- *Circulație cu cereri.*

4.4 Recapitulare generală. Model de testare 2.

1. *Implementați metoda grafică de rezolvare pentru PPL cu GeoGebra. Fie litera A numărul de ordine din registrul grupei.*

Obiectiv	**Constrângerile**
$F \to MAX$ $F = x + 2y$	$\begin{cases} Ax + (A-2)y \leqslant 6 \\ (A-1)x + (A-4)y \leqslant 10 \\ -(A+3)x + (A-4)y \leqslant 12 \\ x, y \geqslant -10 \end{cases}$

Obiectiv	**Constrângerile**
$F \to MIN$ $F = 2x + y$	$\begin{cases} -Ax + (A+2)y \leqslant -15 \\ (A+10)x - (A+4)y \geqslant 30 \\ Ax + (-A+2)y \geqslant 7 \\ x, y \geqslant 0 \end{cases}$

2. *Implementați algoritmul SIMPLEX de rezolvare pentru PPL. Fie literele A și B numere naturale ce reprezintă data și luna nașterii. Pentru data de 22 mai vom avea: A=22 și B=5.*

Obiectiv	**Constrângerile**
$F \to MAX$ $F = 5x - 2y$	$\begin{cases} Ax + By \leqslant 6 \\ (A-1)x + (B-4)y \leqslant 10 \\ -(A+3)x + (B-4)y \leqslant 12 \\ x, y \geqslant 0 \end{cases}$

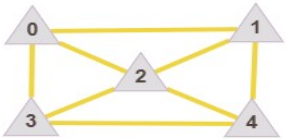
Obiectiv	**Constrângerile**
$F \to MIN$ $F = 2x - 5y$	$\begin{cases} -Bx + (A+2)y \leqslant -15 \\ (A+10)x - (B+4)y \geqslant 30 \\ Bx + (-A+2)y \geqslant 7 \\ x, y \geqslant 0 \end{cases}$

3. *Implementați algoritmul SIMPLEX DUAL de rezolvare pentru PPL.*

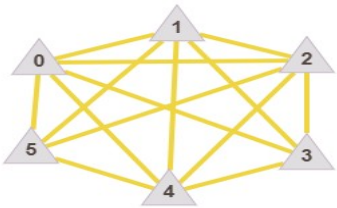
Funcția obiectiv	**Constrângerile**
$F \to MAX$ $F = 4x + 3y$	$\begin{cases} x \leqslant 6 \\ 2x + 3y \leqslant 6 \\ -x + y \leqslant 4 \\ x, y \geqslant 0 \end{cases}$

Funcția obiectiv	**Constrângerile**
$F \to MAX$ $F = 4x + 3y$	$\begin{cases} x \leqslant 6 \\ 2x + 3y \leqslant 6 \\ -x + y \leqslant 4 \\ x, y \geqslant 0 \end{cases}$

4. *Determinați dacă următoarele grafuri reprezintă curcuite Euleriene / Hamiltoniene. Scrieți pentru fiecare graf toate căile Euler / Hamilton posibile.*

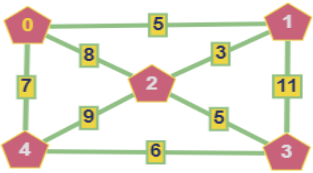


Care este diferența dintre o cale Euler și una Hamilton?

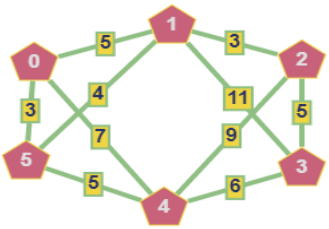


Care este diferența dintre un circuit Euler și unul Hamilton?

5. *Rezolvați următoarele probleme ale călătorului vânzător, stabilind costul minim al traseului său. Orașele (localitățile) vor fi reprezentate sub formă de grafuri neorientate cu ponderi.*

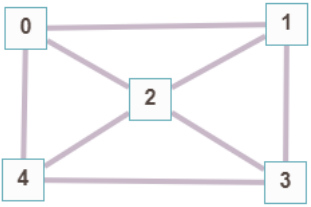


Distanța este: _____
Traseul corespunzător: _____

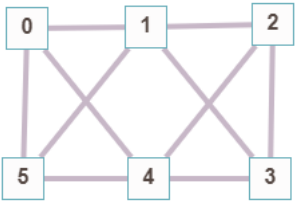


Distanța este: _____
Traseul corespunzător: _____

6. *Stabiliți cum vor fi colorate vârfurile / muchiile următoarelor grafuri, astfel încât să se aplice un număr minim de culori. Vârfurile / muchiile adiacente trebuie să fie de culori diferite.*



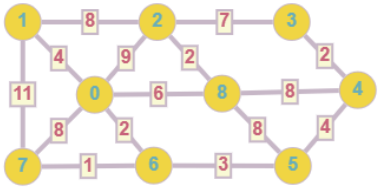
Scrieți numărul minim de culori: ____
Stabiliți ce culoare are fiecare nod.



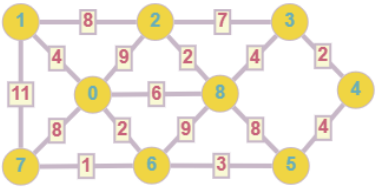
Scrieți numărul minim de culori: ____
Stabiliți ce culoare are fiecare nod.

7. *Verificați după posibilitate în regim online itemul 4 accesând platforma ce permite implementarea unor algoritmi specifici pentru analiza grafurilor: https://graphonline.ru/en/ .*
8. *Verificați după posibilitate în regim online itemul 5 accesând platofrma ce permite implementarea algoritmului pentru rezolvarea pe pași a problemei: https://linprog.com/en/main-traveling-salesman-problem .*
9. *Verificați după posibilitate în regim online itemul 6 accesând platforma ce permite implementarea unor algoritmi specifici pentru algoritmi în grafuri: https://graphonline.ru/en/ .*

10. *Determinați arborele minim de acoperire ale următoarelor grafuri aplicând algoritmul lui Kruskal. Scrieți toate iterațiile posibile până ajungeți la sfârșitul algoritmului.*

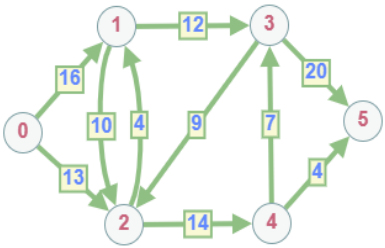


a) *Construiți arborele minim de acoperire.*
b) *Stabiliți numărul de frunze ale acestui arbore.*

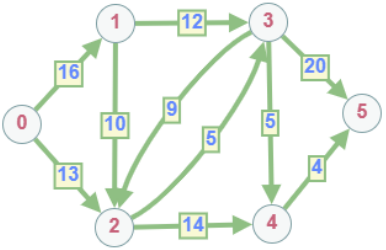


a) *Construiți arborele minim de acoperire.*
b) *Stabiliți numărul de frunze ale acestui arbore.*

11. *Determinați debitul maxim al următoarelor grafuri aplicând algoritmul lui Ford-Fulkerson. Scrieți toate iterațiile posibile până ajungeți la sfârșitul algoritmului.*



a) *Stabiliți care va fi debitul maxim pentru fiecare iterație.*
b) *Debitul maxim este:* _____



a) *Stabiliți care va fi debitul maxim pentru fiecare iterație.*
b) *Debitul maxim este:* _____

1. Unități de conținut

Proba de autoevaluare va conține itemi creați în baza următoarelor unități de conținut conform unității de curs:

1. *Algoritmii pentru circuite în grafuri;*
2. *Algoritmii de optimizare în grafuri;*
3. *Algoritmii de aproximare în grafuri;*
4. *Metoda grafică de rezolvare a PPL;*
5. *Metoda Simplex Primar de rezolvare a PPL;*
6. *Metoda Simplex Dual de rezolvare a PPL.*

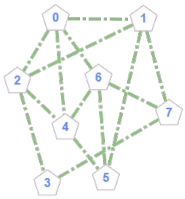
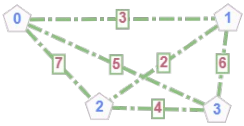
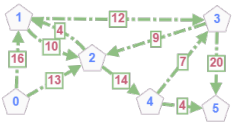
2. Barem orientativ de evaluare a corectitudinii rezultatelor

Item	***Explicații succinte***	***Punctaj***
1	*Pentru scrierea corectă a funcției obiectiv și a sistemului de constrângeri conform condiției problemei*	2*1p
2	*Pentru colorarea corectă a numărului de vârfuri V ale grafului prin utilizarea unui număr minim de culori N.*	2*2p
3	*Pentru scrierea datelor în formă de array bidimensional* *Pentru scrierea corectă a rezultatelor după iterația 1 și 2* *Pentru scrierea corectă a costului distanței și a traseului*	2p 2*2p 2p
4	*Pentru scrierea corectă a primilor 4 pași conform algoritmului problemei (Kruskal / Ford-Fulkerson).*	4*2p
5	*Pentru scrierea corectă a primelor 2 iterații conform algoritmului Simplex primar / Simplex dual:* • *Pentru determinarea corectă a elementului pivot* • *Pentru calcularea corectă a valorilor lui F_j* • *Pentru calcularea corectă a valorilor lui C_j-F_j* • *Pentru scrierea valorii funcției obiectiv după iterația 2*	 2*2p 2*2p 2*2p 1p
	TOTAL:	**35p**

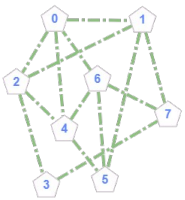
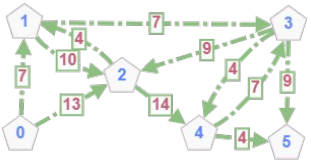
3. Barem de notare

Nota	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Punctaj	35-34	33-30	30-27	26-22	21-17	16-12	11-7	6-4	3-2

Model de test de autoevaluare
Varianta I

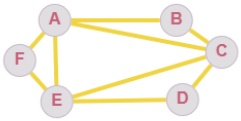
Nr	Condiția itemului	
1	*Transcrieți matematic următoarea problemă (funcția obiectiv și sistemul de constrângeri):* *Un atelier de construcții metalice dispune de țevi cu lungimea de 5m, din care trebuie să taie cel puțin 35 țevi în lungime de 2m, 25 țevi de 2,5m și 10 țevi de 3m lungime. Cum trebuie procedat astfel ca să se realizeze consumuri minime de material?*	**2p**
2	 *Fie un grup de 8 prieteni reprezentat prin următorul graf. Poți fi un prieten cu cineva, doar dacă există între voi o relație de prietenie. Fiecare persoană din grupul vrea să poarte o căciulă de o anumită culoare. De ce culoare ar trebui să fie căciula fiecărei persoane, dacă fiecare prieten trebuie să aibă o căciulă de culoare diferită față de ceilalți prieteni ai săi și numărul de culori folosite trebuie să fie minim.*	**4p**
3	 *Fie un set de orașe și distanța dintre fiecare pereche de orașe reprezentată printr-un graf. Problema este de a găsi cel mai scurt traseu posibil care să viziteze fiecare oraș exact o dată și să revină la punctul de plecare. Scrieți costul distanței și traseul parcurs.*	**8p**
4	*Aplicați algoritmul Kruskal pentru a determina arborele minim de acoperire. Scrieți primele 4 iterații de implementare pas cu pas a algoritmului împreună cu explicațiile corespunzătoare.* 	**8p**
5	*Aplicați algoritmul Simplex Primar pentru rezolvarea tabelară a problemei de la itemul 1. Scrieți primele două iterații și rezultatul funcției obiectiv după iterația a 2-a, pentru fiecare iterație elementul pivot se va încercui în tabel.*	**13p**

Model de test de autoevaluare
Varianta II

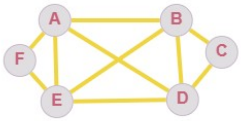
Nr	Condiția itemului	
1	*Transcrieți matematic următoarea problemă (funcția obiectiv și sistemul de constrângeri):* *Un tâmplar dispune de pari de lemn cu lungimea de 5m, din care trebuie să taie cel puțin 35 pari în lungime de 2m, 25 pari de 2,5m și 15 pari de 3m lungime. Cum trebuie procedat astfel ca să se realizeze consumuri minime de material?*	**2p**
2	 *Fie un număr de 8 orașe și drumul național ce unește o pereche de orașe. Găsi o modalitate de a forma un număr minim de grupe care va trasa marcajul rutier în oraș, astfel încât drumurile ce vor fi marcate de o echipă de muncitori să nu fie adiacente. Care este numărul minim e echipe?*	**4p**
3	 *Fie un set de orașe și distanța dintre fiecare pereche de orașe reprezentată printr-un graf. Problema este de a găsi cel mai scurt traseu posibil care să viziteze fiecare oraș exact o dată și să revină la punctul de plecare. Scrieți costul distanței și traseul parcurs.*	**8p**
4	*Aplicați algoritmul Ford-Fulkerson pentru a determina fluxul maxim. Scrieți primele 4 iterații de implementare pas cu pas a algoritmului împreună cu explicațiile corespunzătoare.* 	**8p**
5	*Aplicați algoritmul Simplex Primar pentru rezolvarea tabelară a problemei de la itemul 1. Scrieți primele două iterații și rezultatul funcției obiectiv după iterația a 2-a, pentru fiecare iterație elementul pivot se va încercui în tabel.*	**13p**

Algoritmi în grafuri
Itemi recomandaţi pentru lecţiile practice 11–14

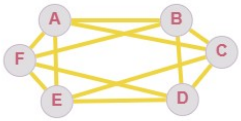
1. *Stabiliți pentru următoarele grafuri un exemplu de cale Euler:*



Exemplu: _____

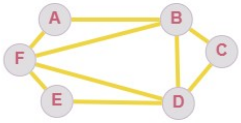


Exemplu: _____

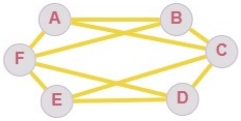


Exemplu: _____

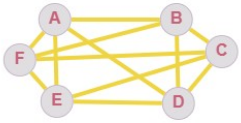
2. *Stabiliți pentru următoarele grafuri un exemplu de circuit Euler:*



Exemplu: _____

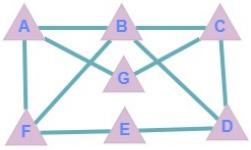


Exemplu: _____

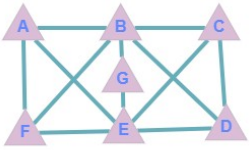


Exemplu: _____

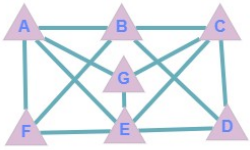
3. *Stabiliți pentru următoarele grafuri un exemplu de cale Hamilton:*



Exemplu: _____

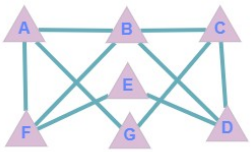


Exemplu: _____

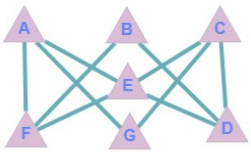


Exemplu: _____

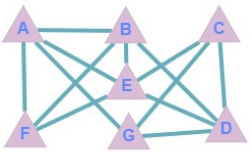
4. *Stabiliți pentru următoarele grafuri un exemplu de circuit Hamilton:*



Exemplu:



Exemplu:



Exemplu:

5. *Scrieți o secvență de cod C/C++ care va afișa toate căile Euler posibile pentru fiecare graf de la itemul 1.*
6. *Scrieți o secvență de cod C/C++ care va afișa toate circuitele Euler posibile pentru fiecare graf de la itemul 2. Datele de intrare (numărul de vârfuri, numărul de arce și perechile de arce) se for citi dintr-un fișier text, iar rezultatul se va afișa la ecran.*
7. *Scrieți o secvență de cod C/C++ care va afișa toate căile Hamilton posibile pentru fiecare graf de la itemul 2.*
8. *Scrieți o secvență de cod C/C++ care va afișa toate circuitele Hamilton posibile pentru fiecare graf de la itemul 4. Datele de intrare (numărul de vârfuri, numărul de arce și perechile de arce) se for citi dintr-un fișier text, iar rezultatul se va afișa la ecran.*
9. *Elaborați o secvență de cod C++ care va efectua subprograme pentru următoarele operațiuni:*
   a) *citirea datelor unui graf (matricea de adiacență) dintr-un fișier extern;*
   b) *afișarea la ecran a datelor citite din fișierul de intrare;*
   c) *afișarea la ecran a tuturor căilor Euler posibile din graf în raport cu un anumit vârf al grafului, vârful este citit de la tastatură;*
   d) *afișarea la ecran a tuturor căilor Hamilton posibile din graf în raport cu un anumit vârf al grafului, vârful este citit de la tastatură;*
   e) *afișarea la ecran a tuturor circuitelor Euler posibile din graf în raport cu un anumit vârf al grafului, vârful este citit de la tastatură;*
   f) *afișarea la ecran a tuturor circuitelor Hamilton posibile din graf în raport cu un anumit vârf al grafului, vârful este citit de la tastatură.*

10. *Determinați soluția problemei vânzătorului călător care trebuie să viziteze un număr de orașe. El dorește să înceapă dintr-un anumit oraș, să viziteze*

fiecare oraș o singură dată și apoi să se întoarcă la punctul său de plecare. Costul călătoriei fiecărui oraș dintr-un anumit oraș este prezentat în tabel.

	A	B	C
A	X	10	15
B	10	X	35
C	15	35	X

Costul=60

	A	B	C
A	X	10	15
B	5	X	5
C	15	5	X

Costul=20

	A	B	C
A	X	7	14
B	7	X	21
C	14	21	X

Costul=35

11. *Determinați soluția problemei vânzătorului călător care trebuie să viziteze un număr de orașe. El dorește să înceapă dintr-un anumit oraș, să viziteze fiecare oraș o singură dată și apoi să se întoarcă la punctul său de plecare. Costul călătoriei fiecărui oraș dintr-un anumit oraș este prezentat în tabel.*

	A	B	C	D	E
A	X	1	2	3	4
B	4	X	1	2	3
C	4	3	X	1	2
D	4	3	2	X	1
E	4	3	2	1	X

Costul=8

	A	B	C	D	E
A	X	1	2	3	4
B	4	X	2	3	4
C	4	3	X	3	4
D	4	3	2	X	4
E	4	3	2	1	X

Costul=12

	A	B	C	D	E
A	X	2	4	6	8
B	19	X	10	12	14
C	17	15	X	16	18
D	13	11	9	X	20
E	7	5	3	1	X

Costul=43

12. *Pentru itemii 10-11, verificați rezultatele obținute după următoarea strategie de rezolvare a problemei:* ***Online Calculator (TSP)****.*

13. *Determinați cel puțin o modalitate de colorare a vârfurilor unui graf, cu condiția de a aplica un număr minim de culori, astfel încât vârfurile adiacente să fie colorate diferit. Pentru fiecare graf avem numărul de vârfuri, numărul de arce și matricea de incidență.*

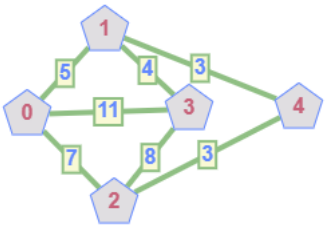
Cazul A	***Cazul B***
4 6 1, 1, 1, 0, 0, 0 0, 1, 0, 1, 1, 0 1, 0, 0, 1, 0, 1 0, 0, 1, 0, 1, 1	5 10 1, 1, 1, 0, 0, 0, 1, 0, 0, 0 0, 1, 0, 1, 1, 0, 0, 1, 0, 0 1, 0, 0, 1, 0, 1, 0, 0, 1, 0 0, 0, 1, 0, 1, 1, 0, 0, 0, 1 0, 0, 0, 0, 0, 0, 1, 1, 1, 1

14. *Determinați cel puțin o modalitate de colorare a arcelor unui graf, cu condiția de a aplica un număr minim de culori, astfel încât arcele adiacente să fie colorate diferit. Pentru fiecare graf avem numărul de vârfuri, numărul de arce și matricea de adiacență.*

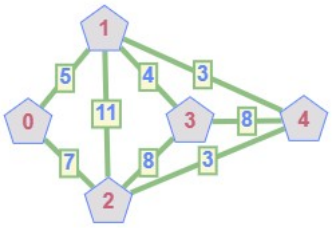
Cazul A	***Cazul B***
6 15 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0.	7 15 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0.

15. *Elaborați o secvență de cod C++ care va efectua subprograme pentru următoarele operațiuni:*
    - *citirea datelor pentru fiecare tabel din itemul 10, dintr-un fișier extern;*
    - *afișarea la ecran a tuturor costurilor minime corespunzătoare fiecărei probleme datele căreia s-au citit dintr-un fișier textual extern.*
16. *Elaborați o secvență de cod C++ care va efectua subprograme pentru următoarele operațiuni:*
    - *citirea datelor pentru fiecare tabel din itemul 11, dintr-un fișier extern;*
    - *afișarea în același fișier a tuturor costurilor minime corespunzătoare fiecărei probleme datele căreia s-au citit dintr-un fișier textual extern.*
17. *Elaborați o secvență de cod C++ care va efectua subprograme pentru următoarele operațiuni:*
    - *citirea matricii de incidență pentru fiecare graf din itemul 13, dintr-un fișier extern, fiecare graf va fi citit dintr-un fișier aparte;*
    - *citirea matricii de adiacență pentru fiecare graf din itemul 14, dintr-un fișier extern, fiecare graf va fi citit dintr-un fișier aparte;*
    - *afișarea în același fișier a tuturor modalităților de colorare a vârfurilor fiecărui graf;*
    - *afișarea în același fișier a tuturor modalităților de colorare a arcelor fiecărui graf;*
    - *afișarea la ecran a numărului de culori aplicate pentru a colora vârfurile fiecărui graf;*
    - *afișarea la ecran a numărului de culori aplicate pentru a colora arcele fiecărui graf.*

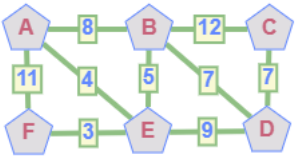
18. *Determinați arborele minim de acoperire ale următoarelor grafuri aplicând algoritmul lui Kruskal.*



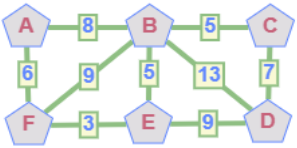
Soluția: _____
_____



Soluția: _____
_____

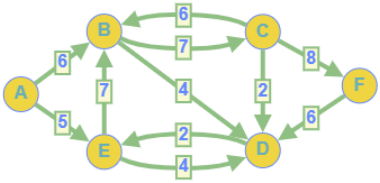


Soluția: _____
_____

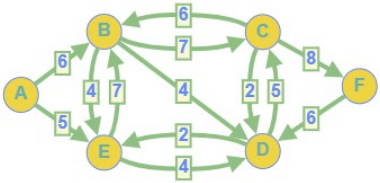


Soluția: _____
_____

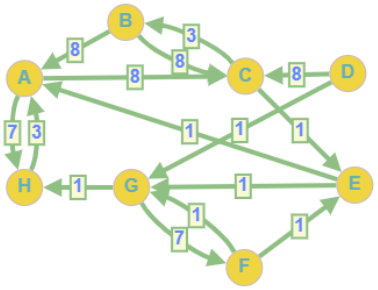
19. *Determinați debitul maxim al următoarelor grafuri aplicând algoritmul lui Ford-Fulkerson.*

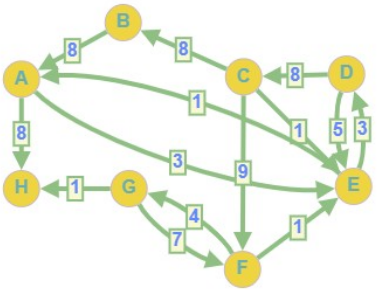


Soluția: _____
_____



Soluția: _____
_____





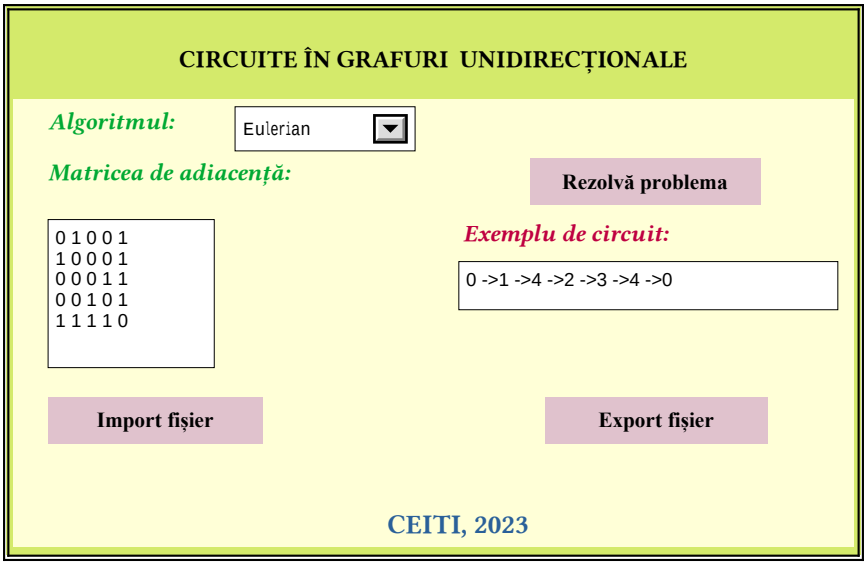
Soluția: _____
_____

Soluția: _____
_____

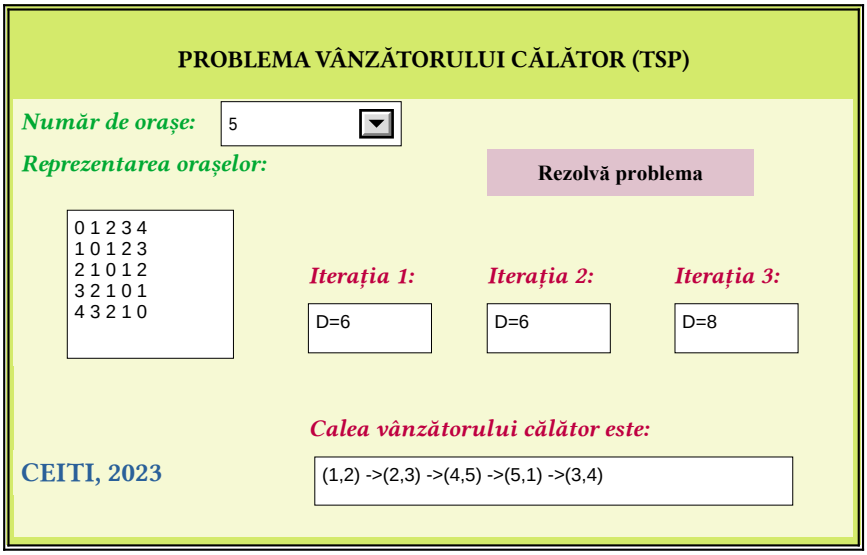
20. *Verificați după posibilitate în regim online itemul 18 accesând platforma ce permite implementarea unor algoritmi specifici pentru grafuri. Din fereastra care apare vom accesa blocul* ***Min Spanning Tree,*** *iar în partea stângă vom alege din meniul propus, algoritmul Kruskal și vom vizualiza soluția pe pași a problemei după ce am creat graful nostru, accesând* ***Draw Graph****, link-ul platormei este: https://visualgo.net/en.*

ELABORAREA UNEI APLICAȚII COMPLEXE IMPLEMENTÂND CONCEPTELE PROGRAMĂRII VIZUALE

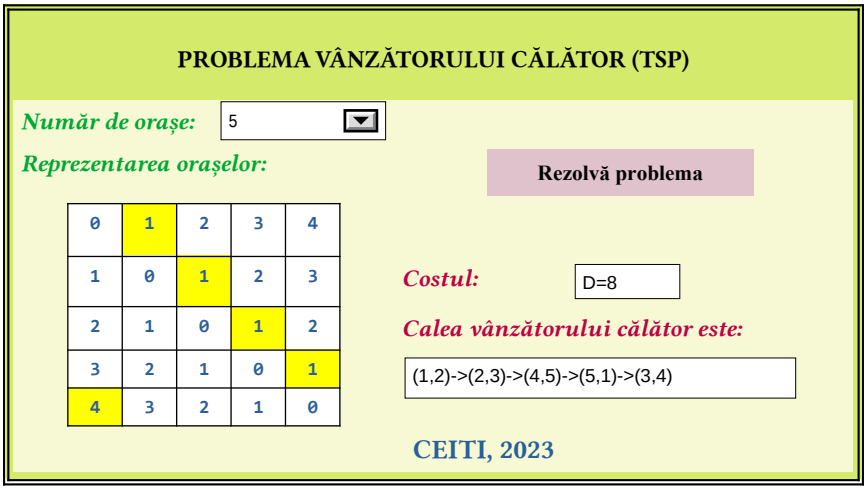
1. *Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: 3 componente de tip Buton, o componentă de tip ComboBox, 9 componente de tip Panel, 2 componente de tip MemoText și 9 componente Label.* ***ComboBox*** *va include tipul de algoritm implementat (Euler sau Hamilton).* ***Buton1*** *va permite importarea matricii de adiacență a grafului neorientat, după importare, datele vor apărea în componenta* ***MemoText1.*** ***Buton2*** *va permite rezolvarea problemei conform algoritmului ales din componenta ComboBox și va afișa în componenta* ***MemoText2*** *un mesaj care confirm prezența sau lipsa unei căi și a unui circuit în raport cu tipul de algoritm ales din componenta* ***ComboBox.*** ***Buton3*** *va permite scrierea datelor problemei într-un fișier în raport cu rezultatul din componenta* ***MemoText2.*** *Dacă graful are o cale conform algoritmului ales, atunci afișăm toate căile posibile din graf. Dacă graful are un circuit conform algoritmului ales, atunci afișăm toate circuitele posibile din graf. La sfârșit se va descărca fișierul text.*



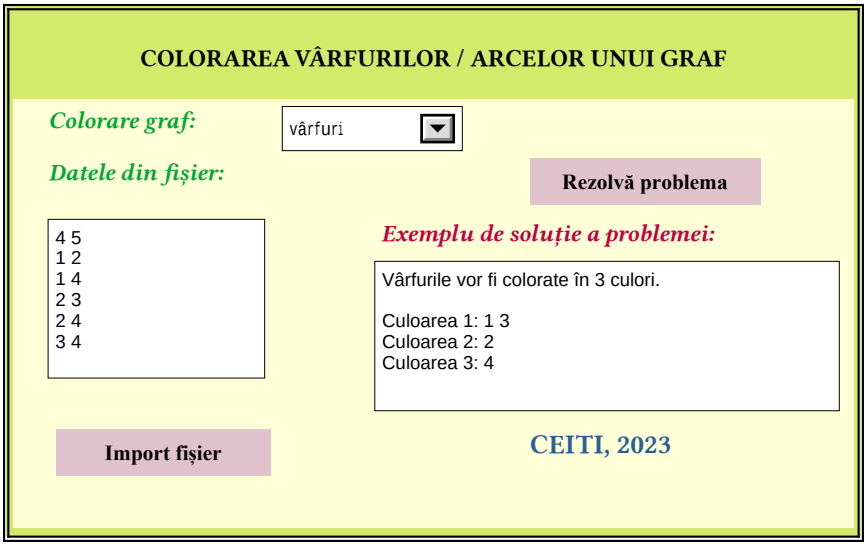
2. *Elaborați o aplicație care va conține în fereastra principală următoarele componente vizuale: o componentă de tip Buton, o componentă de tip ComboBox, 2 componente de tip MemoText, 3 componente de tip Text și 9 componente Label.* ***ComboBox*** *va include numărul de orașe care trebuie traversate. În* ***MemoText1*** *se va introduce manual matricea orașelor și a costurilor acestora.* ***Buton*** *va permite rezolvarea problemei conform datelor numărului de orașe, scriind în cele 3 casete de tip Text distanța obținută după fiecare iterație, dacă a 3-a iterație nu va exista la vre-o problemă, atunci în caseta a 3-a se va scrie cuvântul „LIPSĂ”. După completarea acestor 3 casete, se va completa în același timp și componenta* ***MemoText2*** *cu datele despre traseul parcurs conform distanței minime de la ultima iterație.*



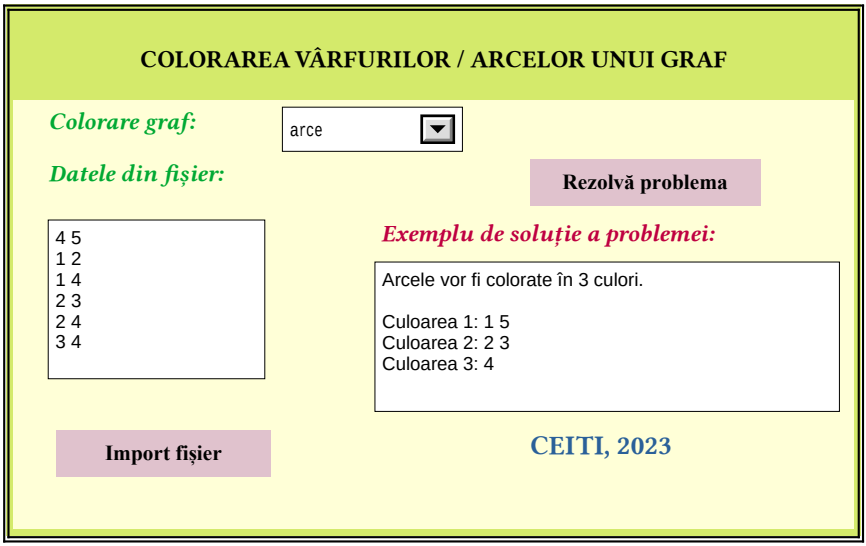
3. *Elaborați o aplicație care va conține în fereastra principală componentele vizuale în conformitate cu repartizarea acestora în imaginea de mai jos.* ***ComboBox*** *va include numărul de orașe care trebuie traversate. În* ***StringGrid*** *se va introduce manual matricea orașelor și a costurilor acestora.* ***Buton*** *va permite rezolvarea problemei conform datelor numărului de orașe, colorând celulele din* ***StringGrid*** *ce prezintă distanța conform costului minim obținut. Se va completa în același timp și componenta* ***MemoText*** *cu datele despre traseul parcurs conform costului minim de la ultima iterație.*



4. *Elaborați o aplicație care va conține în fereastra principală componentele vizuale în conformitate cu repartizarea acestora în imaginea de mai jos. Fișierul de intrare arce.txt, include pe fiecare linie, separate prin spațiu următoarele perechi de numere: (4 5), (1 2), (1 4), (2 3), (2 4) și (3 4).*



În dependență de condiția aleasă din ComboBox, se vor afișa și rezultatele problemei. Pentru același graf, colorarea arcelor ar fi conform imaginii:



Capitolul V

SINTEZA GENERALĂ A UNITĂȚII DE CURS

MODELE DE ITEMI PROPUȘI PENTRU LECȚIA DE RECAPITULARE A CUNOȘTINȚELOR TEORETICE ȘI PRAXIOLOGICE.

Notații importante:

- C_B – *cea mai bună complexitate a timpului;*
- C_M – *complexitatea medie a timpului;*
- C_R – *cea mai rea complexitate a timpului;*
- C_{SR} – *cea mai rea complexitate a spațiului.*

Complexitatea algoritmilor de sortare:

Algoritm	C_B	C_M	C_R	C_{SR}
Bubble	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Insertion	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge	O(n*log n)	O(n*log n)	O(n*log n)	O(n)
Quick	O(n*log n)	O(n*log n)	$O(n^2)$	O(log n)
Heap	O(n*log n)	O(n*log n)	O(n*log n)	O(n)
Shell	O(n)	$O((n*\log n)^2)$	$O((n*\log n)^2)$	O(1)
Bucket	O(n+k)	O(n+k)	$O(n^2)$	O(n)
Counting				
Radix	O(n*k)	O(n*k)	O(n*k)	O(n+k)
Cocktail				
Comb				

1. *Completați casetele din tabelul de mai sus cu informația corespunzătoare complexității algoritmilor de sortare. Pentru aceasta puteți studia mai multe resurse webografice.*
2. *Completați casetele din tabelul de mai jos cu informația corespunzătoare complexității algoritmilor de căutare. Pentru aceasta puteți studia mai multe resurse webografice.*

Complexitatea algoritmilor de căutare:

Algoritm	C_B	C_M	C_R	C_{SR}
Linear	O(1)	O(n)	O(n)	O(1)
Binary	O(1)	O(log n)	O(log n)	O(1)
Jump				
Iterpolation				
Exponential				
Fibonacci				

3. *Elaborați un program în limbajul C++ care va genera un șir de numere întregi pare. Aplicați unul din algoritmii de sortare studiați și aranjați elementele șirului ascendent.*
4. *Elaborați un program în limbajul C++ care va genera un șir de numere întregi prime. Aplicați fiecare din algoritmii de sortare și aranjați elementele șirului descendent.*
5. *Elaborați un program în limbajul C++ care va genera un array 2D de numere întregi. Afișați elementele array-ului 2D sup formă de spirală (la dorință alegeți una din cele 8 modalități posibile). Aplicați unul din algoritmii de sortare studiați și aranjați elementele array-ului 1D (obținut după afișarea sub formă de spirală) ascendent.*
6. *Elaborați un program în limbajul C++ care va genera un array 2D de numere întregi. Afișați elementele array-ului 2D sup formă de diagonală (la dorință alegeți una din cele 8 modalități posibile). Aplicați unul din algoritmii de sortare studiați și aranjați elementele array-ului 1D (obținut după afișarea sub formă de diagonală) descendent.*
7. *Elaborați un program în limbajul C++ care va genera un array 2D de numere întregi. Afișați elementele array-ului 2D sup formă de șarpe (la dorință alegeți una din cele 8 modalități posibile). Aplicați unul din algoritmii de*

sortare studiați și aranjați elementele array-ului 1D (obținut după afișarea sub formă de șarpe) ascendent.

8. *Elaborați un program în limbajul C++ care va genera un array 2D de numere întregi. Afișați elementele array-ului 2D sup formă de spirală (la dorință alegeți una din cele 8 modalități posibile). Aplicați unul din algoritmii de sortare studiați și aranjați elementele array-ului 1D (obținut după afișarea sub formă de spirală) ascendent. Determinați poziția în array-ul 1D a elementului A[0][0] din array-ul 2D generat, se va aplica la dorință un algoritm de căutare studiat.*
9. *Elaborați un program în limbajul C++ care va genera un array 2D de numere întregi. Afișați elementele array-ului 2D sup formă de diagonală (la dorință alegeți una din cele 8 modalități posibile). Aplicați unul din algoritmii de sortare studiați și aranjați elementele array-ului 1D (obținut după afișarea sub formă de diagonală) descendent. Determinați poziția în array-ul 1D a elementului A[1][1] din array-ul 2D generat, se va aplica la dorință un algoritm de căutare studiat.*
10. *Elaborați un program în limbajul C++ care va genera un array 2D de numere întregi. Afișați elementele array-ului 2D sup formă de șarpe (la dorință alegeți una din cele 8 modalități posibile). Aplicați unul din algoritmii de sortare studiați și aranjați elementele array-ului 1D (obținut după afișarea sub formă de șarpe) ascendent. Determinați poziția în array-ul 1D a elementului A[2][2] din array-ul 2D generat, se va aplica la dorință un algoritm de căutare studiat.*
11. *Un fermier trebuie să asigure un minimum de 14 mg pe zi de vitamina A și 16 mg de vitamina B, în furajele pe care le oferă bovinelor. El are două tipuri de furaje, F_1 și F_2, al căror conținut de vitamină pe kilogram este reprezentat în tabelul de mai jos:*

	Vitamina A	**Vitamina B**
Furaj F_1	12	16
Furaj F_2	14	13

Kilogramul de furaj de tipul F_1 costă 0.45 €, iar Kilogramul de furaj de tipul F_2 costă 0.75 €. Cum ar trebui să se amestece cele 2 tipuri de furaje, pentru a obține o hrană mixtă cu vitaminele necesare bovinelor, astfel încât cheltuielile să fie minime? Rezolvați problema prin metoda grafică (cu softul GeoGebra), prin implementarea algoritmului Simplex Primar și algoritmul Simplex Dual.

12. *Creați 3 exemple de grafuri ce conțin minim 5 noduri (vârfuri), care să conțină un circuit Euler, dar să nu conțină un circuit Hamilton, aplicând platforma online din următorul link: https://graphonline.ru/en/ Pentru fiecare graf obținut afișați: matricea de adiacență, matricea de incidență și matricea distanței.*
13. *Creați 3 exemple de grafuri ce conțin minim 10 noduri (vârfuri), care să conțină un circuit Euler. Colorați nodurile sau arcele acestui graf aplicând platforma online din următorul link: https://graphonline.ru/en/ Pentru fiecare graf obținut afișați una din următoarele 3 matrici: matricea de adiacență, matricea de incidență și matricea distanței.*
14. *Creați 3 exemple de grafuri ce conțin minim 10 noduri (vârfuri), care să conțină un circuit Hamilton. Colorați nodurile sau arcele (muchiile) acestui graf aplicând platforma online din următorul link: https://graphonline.ru/en/ Pentru fiecare graf obținut afișați una din următoarele 3 matrici: matricea de adiacență, matricea de incidență și matricea distanței.*
15. *Creați un exemplu de graf poneral ce conține 6 noduri (vârfuri) și suma totală a ponerilor (Sp) este cuprinsă în intervalul [45; 100]. Să presupunem că nodurile acestui graf reprezintă orașele R. Moldova, problema este de a găsi cel mai scurt traseu posibil pentru a vizita fiecare oraș exact o dată și să revină la punctul de plecare. Determinați costul drumului unui călător, rezolvând problema cu ajutorul platformei online: https://linprog.com/ Pentru a vizualiza pașii de rezolvare, alegem din meniul stâng al platformei câmpul corespunzător pentru* **Traveling Salesman Problem.** *Analizați pașii de rezolvare și efectuați concluziile personale.*

Probleme suplimentare[6]:

a) *Generați o matrice de adiacență pentru un graf neorientat fară ponderi. Rezultatul se va păstra într-un fișier text extern.*
b) *Generați o matrice de adiacență pentru un graf neorientat cu ponderi. Rezultatul se va păstra într-un fișier text extern.*
c) *Generați o matrice de adiacență pentru un graf orientat cu ponderi. Rezultatul se va păstra într-un fișier text extern.*

6 *În baza matricilor generate se va putea construi un graf cu ajutorul platformei: https://graphonline.ru/en/*

Capitolul VI

STUDIUL INDIVIDUAL GHIDAT DE PROFESOR

Dispoziții generale[7]

Studiul individual (SI) este o activitate educațională şi de cercetare, care vizează dezvoltarea competențelor necesare şi se desfășoară cu îndrumarea metodică a profesorului, dar fără participarea sa directă.

SI este o formă importantă de instruire orientată spre pregătirea unui specialist care va opera eficient cu competențele profesionale, fiind capabil de o dezvoltare continuă pe parcursul vieții.

Volumul SI (activități individuale, independente a elevului este stabilit prin Planul de învățământ, iar conținutul acestuia prin Curriculumul unității de curs.

Studiul individual al elevului ghidat de profesor (SIGP) poate include şi studiul materialelor suplimentare din cadrul disciplinei/unității de curs, consultații suplimentare pentru elevii cu o reușită scăzută, care întâmpină dificultăți în înțelegerea şi la realizarea sarcinilor de studiu, organizarea activităților didactice cu utilizarea diverselor forme interactive, inclusiv a discuțiilor; realizarea evaluărilor curente; verificarea eseelor, referatelor, rapoartelor, portofoliilor, studiilor de caz elaborate de către studenți etc.

SIGP constituie parte din norma didactică a profesorului şi se include în orarul procesului didactic, constituind partea componentă a volumului de lucru al elevului pe săptămână, suplimentar la numărul de ore de contact direct incluse în planul de învățământ. SIGP este prevăzut pentru toate disciplinele/modulele din planul de învățământ.

Scopul şi sarcinile principale ale studiului individual

SIGP se desfășoară în scopul dobândirii de către elevi a unor competențe profesionale generale şi speciale, inclusiv capacitatea de a utiliza cunoştințele dobândite într-un domeniu profesional.

Un rezultat important al SIGP este formarea la aceştia a capacității de a se instrui individual pentru obținerea de cunoştințe, abilități, competențe în scopul dezvoltării profesionale continue.

---

7 *Aceste informații au fost preluate și adaptate din „Regulamentul privind organizarea studiului individual al studenților la Universitatea Tehnică a Moldovei”, link: Regulament-privind-organizarea-studiului-individual-al-studenților-la-UTM.pdf*

Sarcini principale:

- *crearea abilităților de căutare şi utilizare a informațiilor normative, legislative, de informare şi speciale;*
- *însușirea calitativă şi sistematizarea cunoștințelor teoretice obținute, dezvoltarea acestora şi utilizarea prin intermediul relațiilor multidisciplinare;*
- *formarea abilităților de aplicare a cunoștințelor teoretice în practică (în activitatea profesională);*
- *dezvoltarea abilităților cognitive, formarea independenței în gândire;*
- *dezvoltarea inițiativei creative, a responsabilității şi organizării;*
- *formarea capacității de autodezvoltare;*
- *dezvoltarea abilităților științifice şi de cercetare etc.*

Eficiența studiului individual al studenților este asigurată prin:

- *existența literaturii metodice (indicații metodice, instrucțiuni, îndrumare etc.);*
- *resurse informaționale (manuale, site-uri, SOFT-uri speciale etc.);*
- *materiale de control (teste, chestionare);*
- *consultații;*
- *posibilitatea de prezentare publică a rezultatelor studiului individual (conferințe video, concursuri, seminare publice etc.).*

Principii de organizare a studiului individual al studenților:

- *formularea scopurilor studiului individual, informarea elevilor privind acestea, stimularea atitudinii pozitive şi a interesului studenților pentru acest studiu;*
- *organizarea sistematică şi sporirea graduală a complexității conținutului şi tipurilor de activități de studiu individual,*
- *evidența gradului de pregătire şi a caracteristicilor individuale ale elevilor, diferențierea sarcinilor;*
- *stabilirea clară a finalităților studiului prin competențe şi abilități;*
- *elaborarea indicațiilor metodice pentru studiul individual.*

Sarcina pentru SI va conține:

- *scopul;*
- *conținutul minim al studiului, cu indicarea tipului acestuia;*
- *indicații privind forma de raportare a rezultatelor studiului: referat, raport, rezultate ale experimentului, chestionarului, diagrame, scheme, tabele etc.*

- *termenul de raportare (în cadrul orelor auditoriale, în cadrul unei lucrări de control planificate, prezentarea la control cadrului didactic, raportarea la un seminar, conferință etc.);*
- *forma de evaluare şi baremul.*

Tipuri de activități ale SIGP

Planificarea conținutului SIGP va ține cont de specificul disciplinei/modulului. De exemplu la disciplinele speciale (de specialitate) înafara celor menționate anterior, accentul se va pune pe analiza unei situații specifice cu elaborarea unei soluții.

În mod convențional SIGP poate fi împărțit în două: obligatoriu (SIO) şi controlat (SIC).

- *SIO asigură pregătirea elevului pentru activitatea la clasă. Rezultatele acestei pregătiri se manifestă prin activitatea elevului la ore şi calitatea răspunsului în cadrul evaluărilor. Notele pentru activitatea în clasă sunt parte componentă a rezultatului evaluării curente.*
- *SIC este orientat spre aprofundarea cunoștințelor teoretice, dezvoltarea competențelor analitice în cadrul disciplinelor de studiu. Evaluarea rezultatelor acestui studiu se realizează în cadrul orelor de consultații şi de ghidare a elevilor şi pot face parte din notele de la evaluările curente.*

LUCRAREA INDIVIDUALĂ NR.1

Studiu individual nr.1 la unitatea de curs
Utilizarea tehnicilor clasice de programare
cu genericul
Algoritmi de sortare și căutare în C/C++

Profesor NUME PRENUME

Nume Prenume Elev	***Grupa***	***Link-ul de la secvența video explicativă***	***Nota***

Abilități:

1. *Descrierea algoritmilor de sortare eficientă, prin distribuție și cu bule modificați.*
2. *Descrierea algoritmilor de căutare.*
3. *Elaborarea algoritmilor de sortare eficientă, prin distribuție și cu bule modificați conform specificațiilor propuse.*
4. *Elaborarea algoritmilor de căutare conform specificațiilor propuse.*
5. *Implementarea algoritmilor de sortare și de căutare în limbajul de programare.*

Algormitmii de sortare studiați	***Algormitmii de căutare studiați***
1) *Merge;* 2) *Quick;* 3) *Heap;* 4) *Shell;* 5) *Bucket;* 6) *Counting;* 7) *Radix;* 8) *Cocktail;* 9) *Comb.*	1) *Linear;* 2) *Binary;* 3) *Jump;* 4) *Interpolation;* 5) *Exponential;* 6) *Fobonacci.*

1. *Fiecare elev va genera un array 1D de numere întregi și va determina în acest array poziția unui element care se va introduce de la tastatură. Fiecare elev va aplica algoritmii săi conform variantei (tipul de sortare, algoritmul de sortare și algoritmul de căutare). Elevii vor fi repartizați pe variante conform listei de la repartizarea pe subgrupe.*

1	Ascendent; Merge; Linear.	**9**	Ascendent; Bucket; Exponential.
2	Descendent; Comb; Jump.	**10**	Descendent; Comb; Linear.
3	Ascendent; Quick; Binary.	**11**	Ascendent; Counting; Fibonacci.
4	Descendent; Merge; Fibonacci.	**12**	Descendent; Merge; Binary.
5	Ascendent; Heap; Jump.	**13**	Ascendent; Radix; Linear.
6	Descendent; Quick; Fibonacci.	**14**	Descendent; Shell; Exponential.
7	Ascendent; Shell; Interpolation.	**15**	Ascendent; Cocktail; Binary.
8	Descendent; Heap; Exponential.	**16**	Descendent; Bucket; Jump

2. *Fiecare elev va genera un array 2D (NxM) de numere întregi și va afișa elementele acestui array conform unei metode de parcurgere a elementelor unui array 2D. Rezultatul parcurgerii de va păstra într-un array 1D. Pentru array-ul 1D obținut se va determina poziția unui element care se va introduce de la tastatură. Fiecare elev va aplica algoritmii săi conform variantei de la itemul precedent, iar tipul de parcurgere ale elementelor array-ului 2D este mai jos.*

1	Spirală din NV pe orizontală.	**9**	Șerpuită din SV pe orizontală.
2	Diagonală din NV pe verticală.	**10**	Spirală din SE pe verticală.
3	Șerpuită din NV pe orizontală.	**11**	Diagonală din SE pe orizontală.
4	Spirală din NE pe verticală.	**12**	Șerpuită din SE pe verticală.
5	Diagonală din NE pe orizontală.	**13**	Spirală din NV pe orizontală.
6	Șerpuită din NE pe verticală.	**14**	Diagonală din NV pe verticală.
7	Spirală din SV pe orizontală.	**15**	Șerpuită din NE pe orizontală.
8	Diagonală din SV pe verticală.	**16**	Spirală din NE pe verticală.

3. *Fiecare elev va genera un array 2D (NxM) și va afișa elementele acestui array conform unei metode de parcurgere. Rezultatul parcurgerii de va păstra într-un array 1D. Pentru array-ul 1D obținut se va determina poziția unui element care se va introduce de la tastatură (algoritmul de căutare este la alegere). Fiecare elev va aplica algoritmii săi conform variantei individuale: tipul de parcurgere, tipul de sortare, algorimul de sortare și condiția de sorare este prezentată în tabelul de mai jos.*

Nr	Generare array 2D ce conține	Parcurgere array 2D, tipul și algoritmul de sortare	Condiția de sortare
1	Litere majuscule	Spirală NV-orizontală Ascendent ShellSort	*De sortat toate consoanele, fără a modifica array-ul unidimensional obținut.*
2	Numere de 2 cifre	Spirală NE-orizontală Descendent CocktailSort	*De sortat doar elementele impare, fără a modifica array-ul unidimensional obținut.*
3	Litere majuscule	Spirală SV-orizontală Ascendent RadixSort	*De sortat toate consoanele de pe poziții pare, fără a modifica array-ul unidimensional obținut.*
4	Numere de 2 cifre	Spirală SE-orizontală Descendent MergeSort	*De sortat doar elementele divizibile cu 5, fără a modifica array-ul unidimensional obținut.*
5	Litere majuscule	Spirală NV-verticală Ascendent QuickSort	*De sortat toate consoanele de pe poziții impare, fără a modifica array-ul unidimensional obținut.*
6	Numere de 2 cifre	Spirală NE-verticală Descendent CombSort	*De sortat doar elementele pare, fără a modifica array-ul unidimensional obținut.*
7	Litere majuscule	Spirală SV-verticală Ascendent BucketSort	*De sortat toate consoanele de pe poziții prime, fără a modifica array-ul unidimensional obținut.*
8	Numere de 2 cifre	Spirală SE-verticală Descendent HeapSort	*De sortat doar elementele prime, fără a modifica array-ul unidimensional obținut.*

Nr	Generare array 2D ce conține	Parcurgere array 2D, tipul și algoritmul de sortare	Condiția de sortare
9	Litere majuscule	Șerpuită NV-orizontală Ascendent ShellSort	*De sortat toate consoanele, fără a modifica array-ul unidimensional obținut.*
10	Numere de 2 cifre	Șerpuită NE-orizontală Descendent CocktailSort	*De sortat doar elementele pătrate perfecte, fără a modifica array-ul unidimensional obținut.*
11	Litere majuscule	Șerpuită SV-orizontală Ascendent RadixSort	*De sortat toate consoanele de pe poziții pare, fără a modifica array-ul unidimensional obținut.*
12	Numere de 2 cifre	Șerpuită SE-orizontală Descendent MergeSort	*De sortat doar elementele cuburi perfecte, fără a modifica array-ul unidimensional obținut.*
13	Litere majuscule	Circulară NV-verticală Ascendent QuickSort	*De sortat toate consoanele de pe poziții impare, fără a modifica array-ul unidimensional obținut.*
14	Numere de 2 cifre	Circulară NE-verticală Descendent CombSort	*De sortat doar elementele seriei Fibonacci, fără a modifica array-ul unidimensional obținut.*
15	Litere majuscule	Circulară SV-verticală Ascendent BucketSort	*De sortat toate consoanele de pe poziții prime, fără a modifica array-ul unidimensional obținut.*
16	Numere de 2 cifre	Circulară SE-verticală Descendent HeapSort	*De sortat doar elementele care nu au suma cifrelor un număr par, fără a modifica array-ul unidimensional obținut.*

Barem de notare

Nota	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Puncte									

Barem de corectare[8] (model orientativ)

Nr.item	Explcații	Punctaj
1	• *Pentru crearea unui subprogram ce va permite generarea unui array 1D de numere întregi.* • *Pentru crearea unui subprogram ce va permite afișarea datelor unui array 1D de numere întregi.* • *Pentru crearea unui subprogram ce va permite aplicarea algoritmului de sortare conform sarcinii individuale și tipului de sortare (ascendent sau descendent).* • *Pentru crearea unui subprogram ce va permite aplicarea algoritmului de căutare conform sarcinii individuale, afișând la ecran indicele (poziția) elementului căutat.*	2p 2p 2p 2p
2	• *Pentru crearea unui subprogram ce va permite generarea unui array 2D de numere întregi.* • *Pentru crearea unui subprogram ce va permite afișarea datelor unui array 2D de numere întregi.* • *Pentru crearea unui subprogram ce va permite afișarea datelor unui array 2D de numere întregi sub forma unui array 1D conform metodei de parcurgere (șerpuită, circulară sau diagonală) și punctul de start.* • *Pentru crearea unui subprogram ce va permite aplicarea algoritmului de sortare conform sarcinii individuale și tipului de sortare (ascendent sau descendent).* • *Pentru crearea unui subprogram ce va permite aplicarea algoritmului de căutare conform sarcinii individuale, afișând la ecran indicele (poziția) elementului căutat în array-ul 1D obținut după sortare.*	2p 2p 2p 2p 2p
3	• *Pentru crearea unui subprogram ce va permite generarea unui array 2D de numere întregi.* • *Pentru crearea unui subprogram ce va permite afișarea datelor unui array 2D de numere întregi.*	2p 2p

8 *Recomandăm pentru fiecare lucrare individuală propusă pentru elevi, profesorul să realizeze un barem de corectare și un barem de notare orientativ.*

Nr.item	Explcații	Punctaj
	• *Pentru crearea unui subprogram ce va permite afișarea datelor unui array 2D de numere întregi sub forma unui array 1D conform metodei de parcurgere (șerpuită, circulară sau diagonală) și punctul de start.*	4p
	• *Pentru crearea unui subprogram ce va permite aplicarea algoritmului de sortare conform sarcinii individuale suplimentare și tipului de sortare (ascendent sau descendent).*	2p
	• *Pentru crearea unui subprogram ce va permite aplicarea algoritmului de căutare conform sarcinii individuale, afișând la ecran indicele (poziția) elementului căutat în array-ul 1D obținut după sortare.*	2p
TOTAL:		**30p**

LUCRAREA INDIVIDUALĂ NR.2

Studiu individual nr.2 la unitatea de curs
Utilizarea tehnicilor clasice de programare
cu genericul
Algorimul simplex primar și simplex dual la rezolvarea PPL

Profesor NUME PRENUME

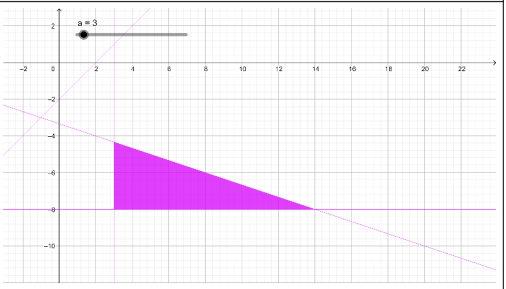
Nume Prenume Elev	***Grupa***	***Link-ul de la secvența video explicativă***	***Nota***

Abilităţi:

1. *Descrierea algoritmilor de programare dinamică pentru rezolvarea problemelor de programare iniară (PPL).*
2. *Rezolvarea PPL prin metoda grafică.*
3. *Elaborarea algoritmilor Simplex Primar și Simplex Dual conform specificațiilor propuse.*
4. *Implementarea algoritmului SIMPLEX și SIMPLEX DUAL de programare liniară în limbajul de programare.*

Forme ale soluției algoritmului SIMPLEX
(1) Metoda grafică (doar pentru 2 necunoscute); (2) Metoda Simplex de bază; (3) Metoda bazei artificiale; (4) Metoda fazei duble; (5) Metoda modificată.

1. *Elevii vor introduce în locul literei A, numărul său de ordine din registrul grupei +1. Rezolvați următoarea PPL cu ajutorul softului GeoGebra. Elevul cu numărul de ordine 2 va avea valoare lui A=2+1, adică A=3, ș.a.m.d.*

Constrângerile și Obiectivul	***Reprezentare grafică model***
$\begin{cases} x \geqslant A \\ -x+y \leqslant A-5 \\ -x-ay \geqslant 10 \\ y \geqslant -5-A \end{cases}$ $F=4x+Ay \rightarrow MAX$ $G=Ax-y \rightarrow MIN$	

2. *Elevii vor introduce în locul literei A, numărul său de ordine din registrul grupei. Rezolvați următoarea PPL cu ajutorul algoritmului Simplex Primar și Simplex Dual.*

Constrângerile și Obiectivul	***Sistem de constrângeri model***
$\begin{cases} (A+3)x+(A+4)y+(A+7)z \leqslant (A+16) \\ -(A+1)x+(A+5)y+(A+3)z \geqslant (A+9) \\ (A+4)x-(A+2)y-(A+6) \leqslant (A+19) \\ x,y,z \geqslant 0 \end{cases}$ $F=(A+1)x-(A+3)y+(A+3)z \rightarrow MIN$ $G=(A+1)x-(A+3)y+(A+3)z \rightarrow MAX$	Pentru A=3, vom obține: $\begin{cases} 6x+7y+10z \leqslant 19 \\ -4x+8y+6z \geqslant 12 \\ 7x-5y-9z \leqslant 21 \\ x,y,z \geqslant 0 \end{cases}$ $F=4x-7y+6z \rightarrow MIN$ $G=4x-6y+6z \rightarrow MAX$

Implementați acești algorimi utilizând calculul tabelar Microsoft Excel sau LibreOffice Calc, scrieți toate formulele necesare.

LUCRAREA INDIVIDUALĂ NR.3

Studiu individual nr.3 la unitatea de curs
Utilizarea tehnicilor clasice de programare
cu genericul
Algorimi în grafuri

Profesor NUME PRENUME

Nume Prenume Elev	***Grupa***	***Link-ul de la secvența video explicativă***	***Nota***

Abilităţi:

1. *Descrierea algoritmilor de optimizare, de aproximare și pe circuite pentru grafuri.*
2. *Prezentarea situațiilor de aplicare a algoritmilor de optimizare, de aproximare și pe circuite pentru grafuri.*
3. *Elaborarea algoritmilor de optimizare, de aproximare și pe circuite pentru grafuri conform specificațiilor propuse.*
4. *Elaborarea algoritmilor de optimizare, de aproximare și pe circuite pentru grafuri conform specificațiilor propuse.*
5. *Implementarea algoritmilor de optimizare pentru grafuri în limbajul de programare.*
6. *Implementarea algoritmilor de aproximare pentru grafuri în limbajul de programare.*
7. *Implementarea algoritmilor pentru circuite în grafuri în limbajul de programare.*

1. *Stabiliți pentru fiecare graf ce este reprezentat printr-o matrice:*
   a) *toate circuitele Euler din fiecare vârt al grafului;*
   b) *toate circuitele Hamilton din fiecare vârf al grafului.*

Subgrupa 1 (1-16) *Elaborați un program în C++ ce va genera într-un fișier o matrice de incidență a unui graf ce conține minim 5 vârfuri. În baza matricii construiți graful și rezolvați problema manual.*	***Subgrupa 2 (17-32)*** *Elaborați un program în C++ ce va genera într-un fișier o matrice de adiacență a unui graf ce conține minim 5 vârfuri. În baza matricii construiți graful și rezolvați problema manual.*

2. *Stabiliți pentru fiecare graf ce este reprezentat printr-o matrice:*
   a) *costul minim conform metodei celui mai apropiat vecin și o cale corespunzătoare acestui cost (drum);*
   b) *costul maxim conform metodei celui mai apropiat vecin și o cale corespunzătoare acestui cost (drum).*

Subgrupa 1 (1-16) *Elaborați un program în C++ ce va genera într-un fișier o matrice de incidență a unui graf ponderat neorientat ce conține minim 5 vârfuri. În baza matricii construiți graful și rezolvați problema vânzătorului călător (Comisului voiajor) manual.*	***Subgrupa 2 (17-32)*** *Elaborați un program în C++ ce va genera într-un fișier o matrice de incidență a unui graf ponderat orientat ce conține minim 5 vârfuri. În baza matricii construiți graful și rezolvați problema vânzătorului călător (Comisului voiajor) manual.*

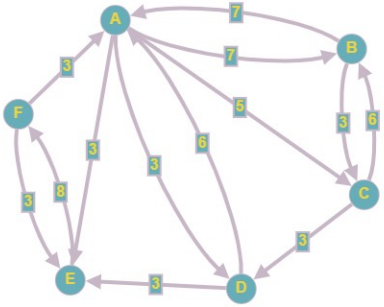
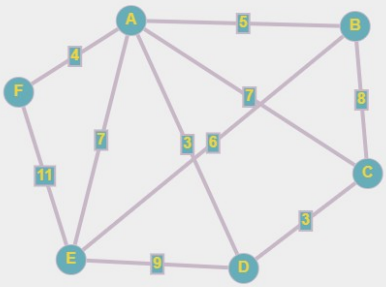
3. *Stabiliți pentru fiecare graf ce este reprezentat printr-o matrice:*
   a) *modul de colorare a vârfurilor sale;*
   b) *modul de colorare a arcelor sale.*

Subgrupa 1 (1-16) *Elaborați un program în C++ ce va genera într-un fișier o matrice de incidență a unui graf neorientat ce conține minim 5 vârfuri. În baza matricii construiți graful și rezolvați problema manual.*	***Subgrupa 2 (17-32)*** *Elaborați un program în C++ ce va genera într-un fișier o matrice de incidență a unui graf orientat ce conține minim 5 vârfuri. În baza matricii construiți graful și rezolvați problema manual.*

4. *Stabiliți pentru fiecare graf ce este reprezentat printr-o matrice:*

Subgrupa 1 (1-16)	***Subgrupa 2 (17-32)***
Elaborați un program în C++ ce va genera într-un fișier o matrice de adiacență a unui graf ponderat orientat ce conține minim 8 vârfuri. În baza matricii construiți graful și aplicăm algoritmul Ford-Fulkerson.	*Elaborați un program în C++ ce va genera într-un fișier o matrice de adiacență a unui graf ponderat neorientat ce conține minim 8 vârfuri. În baza matricii construiți graful și aplicăm algoritmul Kruskal.*

Matricea de adiacență:
`0, 7, 5, 3, 3, 0,`
`7, 0, 3, 0, 0, 0,`
`0, 6, 0, 3, 0, 0,`
`6, 0, 0, 0, 3, 0,`
`0, 0, 0, 0, 0, 8,`
`3, 0, 0, 0, 3, 0,`
	Matricea de adiacență:
	`0, 5, 7, 3, 7, 4,`
	`5, 0, 8, 0, 6, 0,`
	`7, 8, 0, 3, 0, 0,`
	`3, 0, 3, 0, 9, 0,`
	`7, 6, 0, 9, 0, 11,`
	`4, 0, 0, 0, 11, 0,`

Exemplu de graf cu 6 vârfuri: 	***Exemplu de graf cu 6 vârfuri:*** 

BIBLIOGRAFIE

1. Changder N., "Sorting Algorithms and Their Performance Analysis", ISBN: 9789390954506, 2022, India (Mumbai), 126 pag.
2. Pandey D. K., "Efficient Sorting Algorithms for Large Data Sets", ISBN: 9789390354818, 2021, India (Mumbai), 104 pag.
3. Cristian A., "Sortare eficientă: algoritmi, implementări și analiză" , ISBN: 9786069785901, 2021, România (Cluj-Napoca), 160 pag.
4. Pellea S., "Algoritmi de sortare eficiente", ISBN: 9786069782948, 2020, România (Cluj-Napoca), 168 pag.
5. Srivastava D., "Data Structures & Algorithms: Sorting Algorithms in Java", ISBN: 9789390188008, 2020, India (Mumbai), 138 pag.
6. Hillier F. S., Lieberman G. J., "Introduction to Operations Research. - 10th ed.", ISBN 978-0-07-352345-3, 2014, SUA (New York), 1022 pag.
7. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К., "Алгоритмы: построение и анализ. - 2-е изд.", ISBN 978-5-8459-1794-0, 2013, Россия (Москва), 1312 с.
8. Knuth D. E., "Arta programarii. Volumul 3. Sortare si căutare", ISBN 978-973-0-11951-9, 2011, România (București), 832 pag.
9. Sedgewick R., Wayne K., "Algorithms, 4th Edition", ISBN 978-0-321-57351-3, 2011, SUA (New Jersey, Upper Saddle River), 992 pag.
10. Кнут Д. Э., "Искусство программирования. Том 3. Сортировка и поиск. - 2-е изд.", ISBN 978-5-8459-1392-8, 2010, Россия (Москва), 832 с.
11. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C., "Introduction to Algorithms. - 3rd ed.", ISBN 978-0-262-03384-8, 2009, România (București), 1312 pag.
12. Bărbulescu L., "Programare liniară", ISBN 978-973-27-1558-3, 2008, România (București), 184 pag.
13. Dumitrescu D., Măndoiu I., "Grafuri și aplicații", ISBN 973-685-739-5, 2004, România (București), 192 pag.
14. Iordache S., Popa A., "Optimizare combinatorică", ISBN 978-973-667-417-1, 2008, România (București), 320 pag.

ANEXE

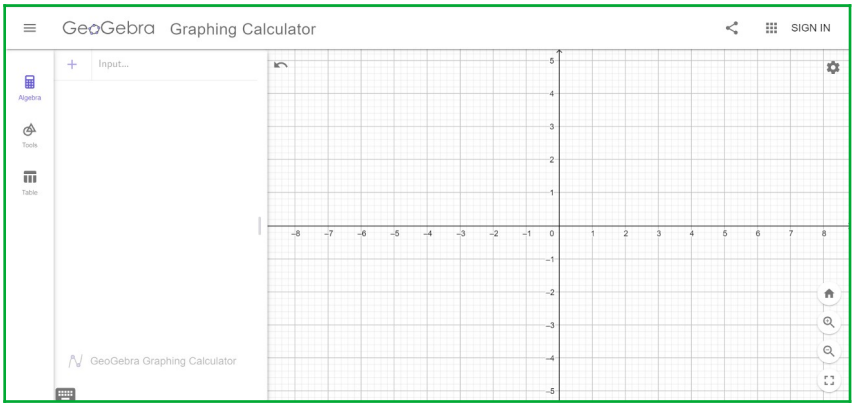
Anexa 1
Implementarea softului GeoGebra pentru rezolvarea PPL

Condiția problemei	***Funcția obiectiv***	***Constrângerile***
Fie următoarea PPL descrisă prin funcția obiectiv și sistemul său de constrângeri. Rezolvați problema aplicând softul matematic GeoGebra.	$F \to MAX$ $F = 4x - 3y$	$\begin{cases} x + 3y \leqslant 7 \\ x - 7y \geqslant 0 \\ x, y \geqslant 0 \end{cases}$

Rezolvare:

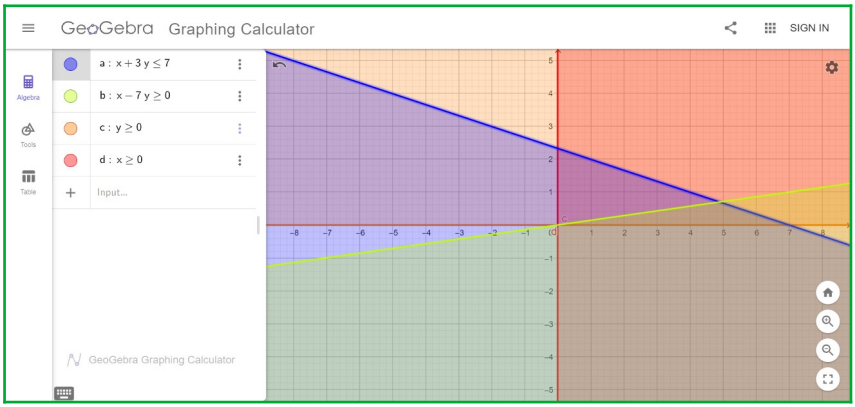
- *Vom descărca softul* ***GeoGebra Graphing Calculator*** *pe calculator accesând următorul link:* https://www.geogebra.org/?lang=en *sau îl vom accesa în regim online:* https://www.geogebra.org/graphing

- *Fereastra aplicației este prezentată în imaginea de mai jos:*

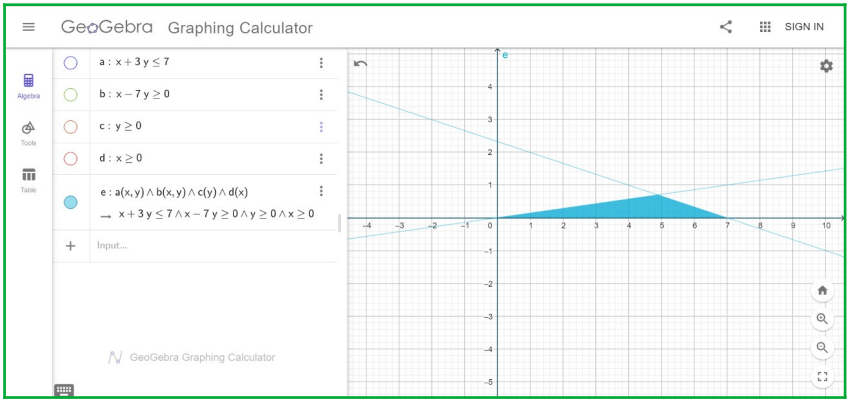


- *În partea stângă sus avem spațiul de introducere a constrângerilor problemei (inegalitățile). Vom introduce constrângerile problemei noastre și vom vizualiza planele obținute conform fiecărei constrângeri, aceste planuri vor avea culori individuale. Pentru a înțelege mai bine cum se introduc datele și cum*

se poate de analizat rezultatele obținute, analizați imaginea de mai jos:



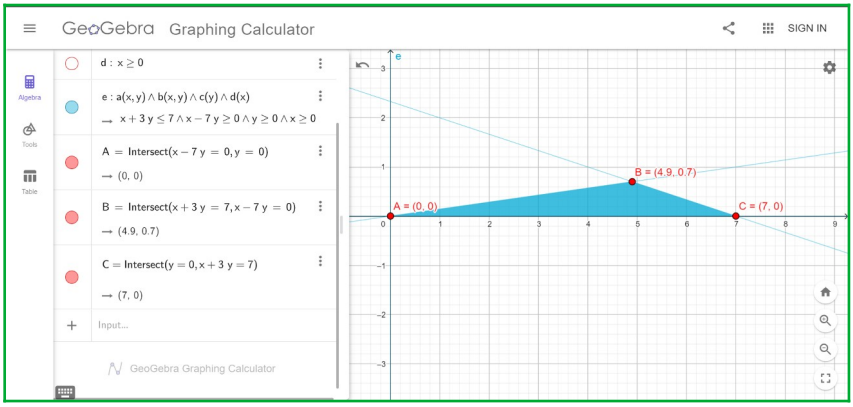
- *Vom stabili intersecția acestor plane, adică vom obține o figură geometrică sau posibil și un plan. Pentru aceasta vom scrie o instrucțiune care ne va permite vizualizarea intersecției. Din imaginea de mai sus se observă că intersecția va fi un triunghi, ceea ce am și obținut în imaginea de mai jos, debifând toate planurile inițiale:*



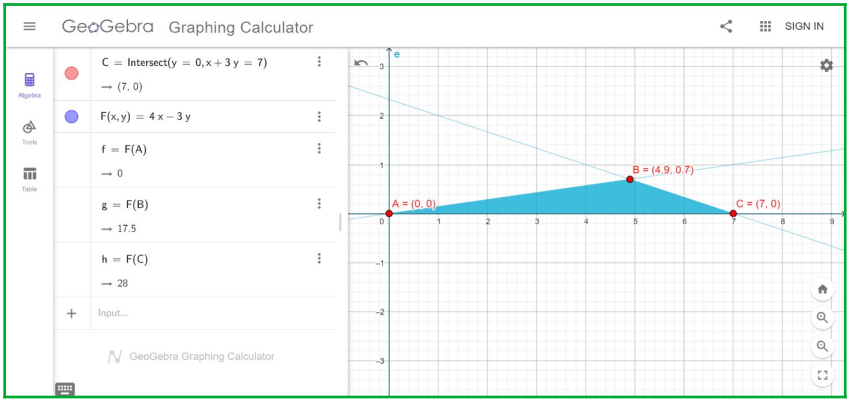
- *Vom stabili punctele de intersecție ale dreptelor, adică vom transforma fiecare inegalitate într-o egalitate. Conform datelor de mai sus, observăm că vom*

avea 3 puncte, deoarece planul de intersecție a tuturor constrângerilor reprezintă un triunghi.

- *În partea stângă a următoarei imagini se observă în intersecția căror drepte se obțin cele 3 puncte A, B și C. Pentru fiecare punct putem stabili o culoare individuală, dar aceasta o să analizăm mai târziu.*



- *Vom scrie funcția obiectiv și vom determina valoarea funcției în fiecare din cele 3 puncte obținute în urma intersecțiilor:*



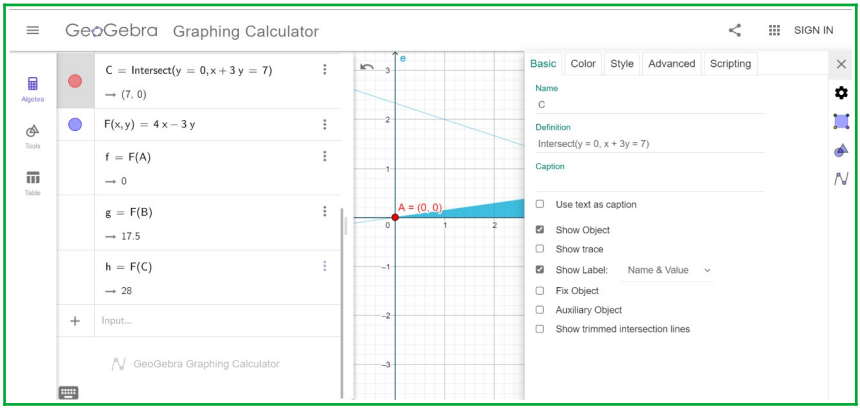
- *Conform condițiilor problemei, funcția noastră obiectiv tinde spre o valoare maximă, această valoare maximă este atinsă în punctul C(7,0), unde*

F(C)=28. Dacă funcția noastră obiectiv era să tindă spre o valoare minimă, atunci această valoare era atinsă în punctul A(0,0), unde F(A)=0.

Important

La rezolvarea PPL, în cazul unei funcții obiective de minimizare se poate obține și o valoare negativă. Rezultatul negativ obținut indiferent de condiția de minimizare sau de maximizare reprezintă un deficit, o pierdere.

- *Vă propun să analizați posibilitățile de editare, pentru aceasta alegeți punctul C obținut în urma intersecției și dați un click pe cele 3 puncte verticale, apoi din meniul vertical alegeți **Settings**, iar ca rezultat va apărea:*

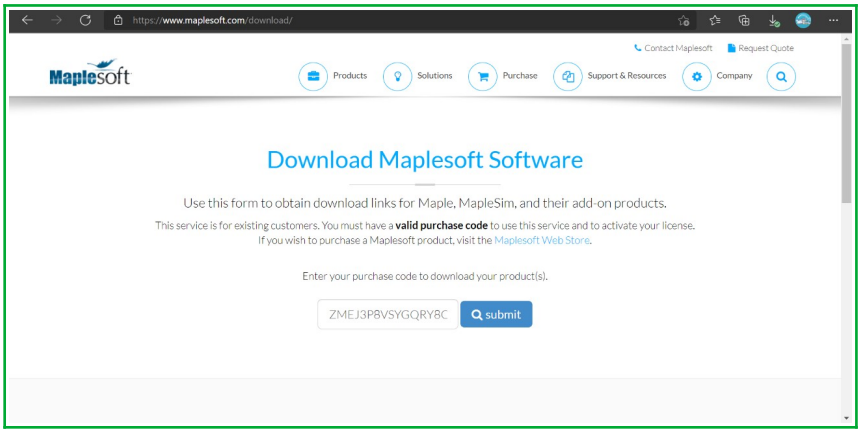


Important

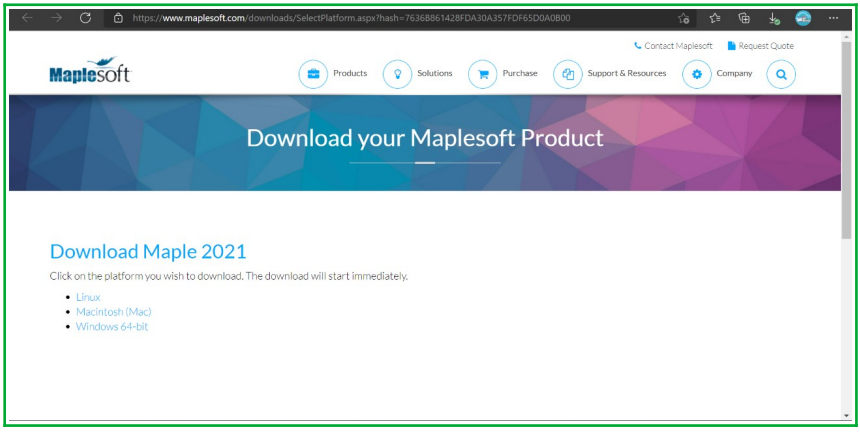
- *Softul GeoGebra este bine structurat, intuitiv în cele mai multe situații. Celelalte submeniuri de editare: Color, Style, Advanced și Scripting puteți să le analizați în mod individual.*
- *În partea stângă aveți și alte meniuri importante ale aplicației: Algebra, Tools și Table, acestea le veți studia de asemenea în mod individual.*

Anexa 2
Implementarea softului Maple 2021 pentru rezolvarea PPL

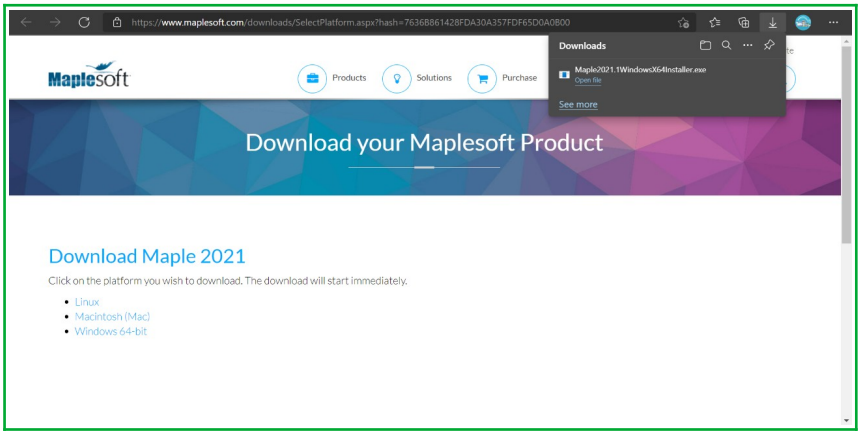
*Vom descărca softul **Maple 2021** pe calculator după ce am efectuat plata pentru licență, pentru aceasta accesăm link-ul: https://www.maplesoft.com/download/ , apoi în spațiul rezervat vom introduce chia de licență și apăsăm **Submit**.*



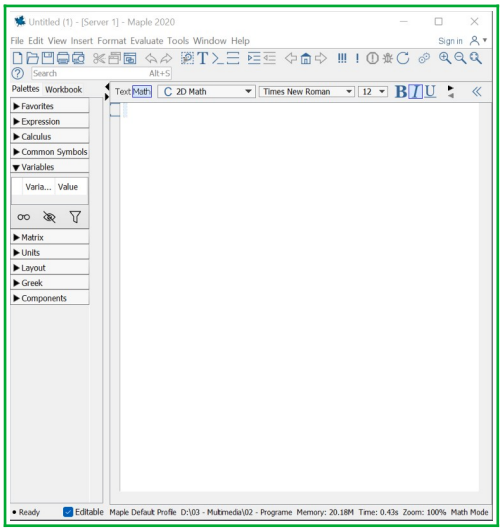
După aceasta vom alege versiunea sistemului nostru de operare și automat se va începe procesul de descărcare a fișierului care ne va permite instalarea softului.



După ce s-a descărcat încep procesul de instalare, procesul cuprinde câțiva pași foarte simpli. Cheia de licență se obține după ce efectuați procedura de înregistrare a profilului personal. Eu am descarcat o versiune de probă a aplicației.



Eu însă deja am instalată pe calculator versiunea 2020, pe care o utilizez de ceva timp. Interfața aplicației arată în felul următor:



Pentru a crea un proiect nou, accesăm meniul: File - > New -> Worksheet Mode. În continuare vom analiza secvența de instrucțiuni pentru rezolvarea unei PPL:

a) obiectiv maxim realizat;

```
> with(simplex) :
> cnsts := {x + y + z ≤ 30, 2 x + y + 3 z ≥ 60, x − y + 2 z = 20} :
> obj := 2 x + 3 y + 4 z :
> maximize(obj, cnsts union {0 ≤ x, 0 ≤ y, 0 ≤ z})
```

$$\left\{x = 0, y = \frac{40}{3}, z = \frac{50}{3}\right\} \quad (1)$$

```
> sol1 := maximize(obj, cnsts) :
> assign(sol1) : obj;
```

$$\frac{320}{3} \quad (2)$$

Am făcut apel la pachetul de date simplex, apoi am introdus constrângerile problemei noastre (cnsts), apoi am aplicat funcția obiectiv (obj). După aceasta am aplicat funcția de minimizare și am obținut valorile variabilelor x, y și z. Acum în variabila sol1 vom salva valoarea funcției în punctele obținute x,y și z, apoi valoarea obținută o afișăm prin apelul assign. Cazul b este cu minimizare, puteți analiza mai jos ce se modifică.

b) obiectiv minim realizat;

```
> with(simplex) :
> cnsts := {x + y + z ≤ 30, 2 x + y + 3 z ≥ 60, x − y + 2 z = 20} :
> obj := 2 x + 3 y + 4 z :
> minimize(obj, cnsts union {0 ≤ x, 0 ≤ y, 0 ≤ z})
```

$$\{x = 10, y = 10, z = 10\} \quad (1)$$

```
> sol2 := minimize(obj, cnsts) :
> assign(sol2) : obj;
```

$$90 \quad (2)$$

Anexa 3
Metode de rezolvare a problemei vânzătorului călător (ambulant)

Un vânzător ambulant trebuie să viziteze 4 orașe. El dorește să înceapă dintr-un anumit oraș, să viziteze fiecare oraș o singură dată și apoi să se întoarcă la punctul său de plecare. Costul călătoriei fiecărui oraș dintr-un anumit oraș este prezentat în tabelul alăturat.

Graf neorientat

	A	B	C	D
A	X	10	15	20
B	10	X	35	25
C	15	35	X	30
D	20	25	30	X

Graf orientat

	A	B	C	D
A	X	10	15	20
B	5	X	35	25
C	10	25	X	30
D	15	20	25	X

Rezolvare 1:

◆ ***Folosind metoda celui mai apropiat vecin***

1. *Dacă începem de la A, atunci calea este A⟶B=10, B⟶D=25, D⟶C=30, C⟶A=15, iar distanța totală = 80.*
2. *Dacă începem de la B, atunci calea este B⟶A=10, A⟶C=15, C⟶D=30, D⟶B=25, iar distanța totală = 80.*
3. *Dacă începem de la C, atunci calea este C⟶A=15, A⟶B=10, B⟶D=25, D⟶C=30, iar distanța totală = 80.*
4. *Dacă începem de la D, atunci calea este D⟶A=20, A⟶B=10, B⟶C=35, C⟶D=30 și distanța totală = 95.*

Răspuns: *Distanța minimă este 80.*

Rezolvare 2:

◆ ***Folosind metoda celui mai apropiat vecin***

1. *Dacă începem de la A, atunci calea este A⟶B=10, B⟶D=25, D⟶C=25, C⟶A=10, iar distanța totală = 70.*
2. *Dacă începem de la B, atunci calea este B⟶A=5, A⟶C=15, C⟶D=30, D⟶B=20, iar distanța totală = 70.*
3. *Dacă începem de la C, atunci calea este C⟶A=10, A⟶B=10, B⟶D=25, D⟶C=25, iar distanța totală = 70.*
4. *Dacă începem de la D, atunci calea este D⟶A=15, A⟶B=10, B⟶C=35, C⟶D=30 și distanța totală = 90*

Răspuns: *Distanța minimă este 70.*

Anexa 4
Model de testare finală asistată de calculator

Competenţe specifice la unitatea de curs:

1. *Prelucrarea tipurilor dinamice de date în cadrul aplicațiilor de consolă.*
2. *Utilizarea structurilor dinamice de date pentru problemele întâlnite în activitatea profesională.*
3. *Gestionarea eficientă a memoriei interne a calculatorului.*
4. *Utilizarea tehnicilor clasice de programare pentru problemele întâlnite în activitatea profesională.*
5. *Utilizarea grafurilor pentru problemele întâlnite în activitatea profesională.*
6. *Alegerea tehnicii clasice de programare adecvate problemei.*

Unităţi de competenţă la unitatea de curs:

1. *Utilizarea algoritmilor de sortare în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*
2. *Utilizarea algoritmilor de căutare în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*
3. *Utilizarea algoritmilor de programare dinamică în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*
4. *Utilizarea algoritmilor combinatorici în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*

Model de examen realizat pe platforma Moodle

Testarea finală (examenul nu va depăși 180 min, profesorul stabilește timpul probei teoretice și al celei practice) va fi asistat de calculator, acesta va include două probe: Proba teoretică (25 min) și Proba practică (155 min).

***Proba teoretică** va include un test grilă din minim 20 itemi dintr-o bancă de itemi de minim 60 de itemi propuși spre realizare conform fiecărei unități de competență în mod aleator.*

***Proba practică** va include un test grilă (itemi de tip eseu) din minim 5 itemi (probleme de rezolvat în limbajul de programare C++) dintr-o bancă de itemi de minim 20 de itemi propuși spre realizare (itemii se vor selecta în același mod ca la proba teoretică).*

*Nota finală la această probă de examinare se va calcula astfel: NT = 0.4 * nota obținută la proba teoretică, NP = 0.6 * nota obținută la proba practică, astfel nota finală va fi: N = NT + NP. Exemplu: nota la proba teoretică a fost 8 și la proba practică - 6, atunci nota finală la examen va fi: 0.4*8 + 0.6*6 = 3.2+3.6 = 6.8 (adică va avea 7).*

Banca de itemi (model orientativ)

C = *Cunoaștere;* A = *Aplicare (înțelegere funcțională);* I = *Integrare (soluționarea problemelor).*

	Unităţi de competenţă	*Numărul de itemi după nivele cognitive*			Total itemi
		C (30%)	A (40%)	I (30%)	
1	*Utilizarea algoritmilor de sortare în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*	9 (2)	8 (2)	9 (2)	26 (6)
2	*Utilizarea algoritmilor de căutare în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*	12 (2)	8 (2)	12 (2)	32 (6)
3	*Utilizarea algoritmilor de programare dinamică în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*	8 (2)	8 (2)	12 (2)	28 (6)
4	*Utilizarea algoritmilor combinatorici în cadrul aplicațiilor de consolă și a aplicațiilor vizuale.*	8 (3)	24 (6)	12 (3)	44 (12)
	Total itemi: / Total itemi în test:	37 (9)	48 (12)	45 (9)	130 (30)

Barem de notare pentru proba teoretică

Proba teoretică *include cei 21 itemi conform nivelurilor cognitive notate cu* ***C*** *și* ***I****. Fiecare item din nivelul* ***C*** *va fi apreciat cu 1p (punct), iar cele din nivelul* ***I*** *- cu 2p (puncte). În total vor fi acumulate: 9+24=33 puncte.*

Analizați următorul barem procentual:

%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5	4-1
Nota	10	9	8	7	6	5	4	3	2	1

În contextul desfășurării procesului educațional în regim online, fiecare cadru didactic va realiza un test asistat de calculator prin intermediul unei platforme educaționale propuse de instituția de învățământ. Modele de itemi pentru fiecare categorie cognitivă:

Itemi de cunoaștere	Itemi de înțelegere funcțională
• *itemi cu alegere duală;* • *itemi cu alegere multiplă;* • *itemi de tip pereche.*	• *itemi cu răspus scurt;* • *itemi cu răspuns de completare;* • *itemi de tip întrebare structurată.*

MODELE DE ITEMI DIN CATEGORIA C (cunoaștere):

UC 1	1. *Metoda Shell începe prin sortarea perechilor de elemente aflate la o distanță una de alta, apoi reducând progresiv decalajul dintre elementele care trebuie comparate. (Adevărat / Fals)* 2. *Cine este autorul algoritmului de sortare Heap ?* ○ J. W. J. Williams ○ Donald Shell ○ C. A. R. Hoare ○ John von Neumann
UC 2	1. *Ideea fundamentală din spatele Jump Search este de a căuta un număr mai mic de elemente în comparație cu algoritmul Linear Search. (Adevărat / Fals)* 2. *Care este denumirea algoritmului de sortare ce este mai rapid decât Lineary Search dar mai încet decât Binary Search ?* ○ Jump ○ Exponential ○ Fibonacci ○ Interpolation
UC 3	1. *Metodele grafice de rezolvare a problemelor de programare liniară (PPL) sunt aplicabile numai pentru modelele de programare liniară*

	cu două variabile. (Adevărat / Fals) 2. *Când a fost demonstrat că algoritmul simplex este un algoritm în timp exponențial, iar pe de altă parte, algoritmul se comportă în timp polinomial pentru rezolvarea problemelor din viața reală?* ◯ 1972 ◯ 1973 ◯ 1971 ◯ 1974
UC 4	1. *Calea hamiltoniană poate fi determinată (găsită) în timp O(n) și complexitatea spațială este O(n). (Adevărat / Fals)* 2. *Când a fost dezvoltat algoritmul lui Joseph Kruskal pentru determinarea arborelui minim de acoperire în graf?* ◯ 1956 ◯ 1955 ◯ 1957 ◯ 1960

MODELE DE ITEMI DIN CATEGORIA A (înțelegere funcțională):

UC 1	1. *Care va fi numărul de pași (se va număra fiecare operațiune, nu iterațiile) pentru sortarea ascendentă a următorului șir de numere: 4, 7, 8, 2, 9? Pentru sortare se va aplica algoritmul Quick Sort. {=45 ~35 ~25 ~55}* ◯ 45 ◯ 25 ◯ 35 ◯ 55 2. *Care va fi complexitatea cea mai rea (mare) a algoritmului Quick Sort în cazul unui array unidimensional de 8 numere întregi ?* ◯ 8^2 (este 64) ◯ 2^8 ◯ 8^3 (este 512) ◯ 3^8
UC 2	1. *Fie următorul array unidimensional de numere întregi pozitive: 19, 37, 30, 17, 40, 11, 43, 36, 21, 15. Stabiliți care va fi poziția numărului 37 conform algoritmului Linear Search, dacă indicele de început este 0 ?*

	○ 7 ○ 2 ○ 1 ○ 8 2. *Fie următorul array unidimensional de litere minuscule: r, o, j, v, h, m, g, l, i, f. Stabiliți care va fi poziția caracterului „m” conform algoritmului Exponential Search, dacă indicele de început este 0 ?* ○ 6 ○ 3 ○ 4 ○ 5
UC 3	1. *Soluția următoarei probleme de programare liniară conform sistemului de constrângeri: 2x+y<=5, x+3y<=11, x>=0, y>=0 și următoarea funcție obiectiv de minimizare: F(x,y)=5x-y.* ○ -4 ○ -4.5 ○ -0.6 ○ -0.8 2. *Soluția următoarei probleme de programare liniară conform sistemului de constrângeri: 3x-7y>=15, –4x+6y<=20 și următoarea funcție obiectiv de minimizare: F(x,y)=3x+4y. Determinați elementul pivot al primei iterații conform algoritmului Simplex.* ○ 3 ○ 9 ○ 7 ○ 5
UC 4	1. *Un graf neorientat și fără ponderi este reprezentat prin următoarea matrice de adiacență* [ ] *un circuit Euler:* *0, 0, 1, 1,* *0, 0, 1, 1,* *1, 1, 0, 0,* *1, 1, 0, 0.* *În spațiul rezervat scrieți una din varinatele: conține / nu conține.* 2. *Fie un graf neorientat și fără ponderi reprezentat prin următoarea matrice de adiacență:* *0, 1, 0, 1, 0,* *1, 0, 1, 1, 1,* *0, 1, 0, 0, 1,* *1, 1, 0, 0, 0,*

	0, 1, 1, 0, 0. *Ce nod va avea aceeași culoare precum nodul 2?* *Răspuns:* [ ] *(nodurile se vor separa doar prin spațiu, fără virgulă, nodul de început este considerat 0).*

MODELE DE ITEMI DIN CATEGORIA I (integrare):

UC 1	1. *Elaborați un program în limbajul C++ care va sorta ascendent un șir de numere întregi conform metodei Merge Sort. Șirul de numere se va genera, iar rezultatul sortării se va afișa la ecran.* 2. *Elaborați un program în limbajul C++ care va sorta ascendent un șir de numere întregi conform metodei Cocktail Sort. Șirul de numere se va genera, iar rezultatul sortării se va afișa la ecran.*
UC 2	1. *Elaborați un program în limbajul C++ care va sorta descendent un șir de numere întregi conform unui algoritm de sortare la alegere, apoi se va introduce de la tastatură un element poziția căruia trebuie de căutat. Pentru căutarea elementului se va aplica algoritmul Linear Search. Șirul de numere se va genera, iar rezultatul sortării se va afișa la ecran, apoi se va afișa elementul căutat, precum și poziția acestuia în șirul deja sortat.* 2. *Elaborați un program în limbajul C++ care va sorta descendent un șir de caractere conform unui algoritm de sortare la alegere, apoi se va introduce de la tastatură un element poziția căruia trebuie de căutat. Pentru căutarea elementului se va aplica algoritmul Exponential Search. Șirul de caractere se va genera, iar rezultatul sortării se va afișa la ecran, apoi se va afișa elementul căutat, precum și poziția acestuia în șirul deja sortat.*
UC 3	1. *Rezolvați următoarea problemă de programare liniară (PPL) conform sistemului de constrângeri: x+y>12, 38x+45y<505, x>=0, y>=0 și următoarea funcție obiectiv de minimizare: F(x,y)=x+3y. Scrieți valoarea funcției obiectiv. Se va aplica metoda grafică.* 2. *Rezolvați următoarea problemă de programare liniară (PPL) conform sistemului de constrângeri: 2x+y<50, x+3y<118, x>=0, y>=0 și următoarea funcție obiectiv de maximizare: F(x,y)=5x-y. Scrieți valoarea funcției obiectiv. Se va aplica algoritmul SIMPLEX DUAL.*
UC 4	1. *Stabiliți dacă următorul graf reprezentat prin următoarea matrice de adiacență include un circuit Eulerian:*

	0, 1, 0, 1, *1, 0, 0, 1,* *0, 0, 0, 1,* *1, 1, 1, 0.* *Scrieți un exemplu de circuit.* 2. *Fie un set de orașe și distanța dintre fiecare pereche de orașe, problema este de a găsi cel mai scurt traseu posibil care să viziteze fiecare oraș exact o dată și să revină la punctul de plecare. Mulțimea de orașe este reprezentată prin următoarea matrice de adiacență a unui graf ponderal:* *0, 6, 3, 7,* *6, 0, 0, 8,* *3, 0, 0, 5,* *7, 8, 5, 0.* *Explicați rezolvarea problemei conform metodei celui mai apropiat vecin.*

Anexa 5
Repere metodologice, privind predarea / învățarea unității de curs

Proiectarea didactică *este o activitate complexă, un proces deliberativ de fixare a paşilor ce vor fi parcurşi în realizarea instrucţiei şi educaţiei. Proiectarea didactică - o anticipare a ceea ce doreşte cadrul didactic să realizeze împreună cu elevii în cadrul unei activităţi, a unei zile, a unei luni, pe parcursul unui an, ţinînd cont de obiectivele curriculare.*

Proiectarea didactică de lungă durată *presupune realizarea asocierilor dintre competențele specifice, unități de competențe, obiective și conținuturi; împărțirea conținuturilor instructiv educative în unități de învățare; stabilirea succesiunii de parcurgere a unităților de învățare și detalierea conținuturilor tematice ale fiecărei unități de învățare incluse în Curriculumul unității de curs „Utilizarea tehnicelor clasice de programare" (UTCP); alocarea resurselor de timp considerate necesare pentru fiecare unitate de învățare, în concordanță cu cerințele administrării unității de curs. În procesul proiectării didactice se va asigura respectarea succesiunii logice a unităților de conținut. Administrarea unității de curs / Repartizarea modulelor pe unități de timp – pentru fiecare unitate de conținut / de învățare se va indica numărul de ore alocate, numărul de evaluări; numărul de ore pe semestru.*

Proiectarea didactică de scurtă durată *sau* ***Proiectul de lecție*** *este un instrument de lucru pentru cadrul didactic. Este un plan prin care se asigură faptul că toate activitățile constituie ating scopul urmărit, în care există o succesiune logică a activităților și în care sunt antrenați toți elevii.*

În alegerea strategiilor didactice şi a tehnologiilor pot fi utilizate sugestiile recomandate de Curriculum, Ghidul profesorului şi alte surse, decizia finală privind organizarea şi desfăşurarea activităţilor aparţinând profesorului. Activităţile de învăţare şi evaluare recomandate sunt grupate pe nivele de complexitate şi vizează, atât competenţele cu caracter de înţelegere şi aplicare (rezolvarea exerciţiilor, rezolvarea de probleme), cât şi competenţele cu caracter de integrare (studiul de caz, experimentul, proiectul). Pentru elaborarea sarcinilor didactice se va utiliza în special taxonomia lui Bloom clasică sau cea revizuită.

Metodele recomandate la disciplina UTCP sunt: expunerea de material teoretic, lucrul la calculator, individual şi/sau sub conducerea cadrului didactic, rezolvarea de probleme, lucrarea practică, lucrarea de laborator, studiu individual ghidat de profesor.

Profesorii, în proiectele didactice, specifică unitățile de competență semnificative pentru lecția respectivă și în baza acestora formulează obiectivele lecției.

În afară de mediul fizic de învăţare, se recomandă crearea un mediu de învăţare virtual, pentru gestionarea resurselor de învăţare, a sarcinilor, a feedback-ului şi pentru evaluarea rezultatelor învăţării. Resurse care pot fi folosite la crearea unui mediu de colaborare online: Moodle - www.moodle.org, Google Meet - https://meet.google.com etc.

Implementarea Curriculumului se va realiza prin evaluarea inițială a nivelului de stăpânire de către elevi a competențelor parțial formate la disciplinele studiate anterior. Se recomandă organizarea evaluării inițiale, prin test, realizat pe platforma educațională (Moodle).

În scopul eficientizării procesului de predare-învățare, se recomandă utilizarea diverselor surse electronice, instrumente digitale. Pentru realizarea sarcinilor propuse se recomandă utilizarea limbajului de programare C++, C# sau Java. Limbajul de programare se poate utiliza folosind un mediu de dezvoltare (Code Blocks, Visual Studio e.t.c.). Pentru implementarea metodelor grafice de rezolvare a problemelor de programare liniară (PPL) se recomandă utilizarea softului matematic GeoGebra Graphing Calculator 2. Rezolvarea PPL prin utilizarea algoritmului Simplex poate fi realizată cu ajutorul unui procesor abelar (MS Excel, Libre Office Calc e.t.c.), iar verificarea soluției prin Simplex method calculator (atozmath.com), Online Calculator: Simplex Method (linprog.com).

Modelele de evaluare urmează să fie adaptate la strategiile didactice utilizate de fiecare cadru didactic. Sarcinile propuse în cadrul suportului de curs la disciplina UTCP, poartă un caracter de recomandare. Ca urmare, profesorul, la discreția sa, poate modifica sarcinile în conformitate cu Curriculumul disciplinei. Vom evidenția următoarele tipuri de evaluare:

a) evaluarea inițială (se va realiza sub forma unui test grilă);

b) evaluarea curentă (se va realiza prin intermediul platformei MOODLE);

c) evaluarea finală (se va realiza prin intermediul platformei MOODLE).

La fiecare început de semestru, cadrele didactice, vor instrui elevii cu privire la normele de securitate și protecție a muncii, pentru desfășurare în condiții de siguranța a orelor în laborator. Vor informa elevii asupra accidentelor care se pot produce și asupra modului de acordare a primului ajutor acolo unde este cazul.

Recomandări de aplicații / platforme pentru crearea prezentărilor :

1. *Prezi (vezi link: https://prezi.com/);*
2. *Genial (vezi link: https://genial.ly/);*
3. *Sway (descărcați softul de la acest link: https://www.microsoft.com/en-us/p/sway/9wzdncrd2g0j?activetab=pivot:overviewtab);*
4. *Slidebean (vezi link: https://slidebean.com/);*
5. *Knovio (vezi link: https://knovio.com/).*

Recomandări de aplicații / platforme pentru crearea testelor grilă:

1. *ProProfs Quiz Maker (vezi link: https://www.proprofs.com/quiz-school/);*
2. *Survey Anyplace (vezi link: https://surveyanyplace.com/);*
3. *Survey Sparrow (vezi link: https://surveysparrow.com/);*
4. *Survey Monkey (vezi link: https://www.surveymonkey.com/);*
5. *Survey Gizmo (vezi link: https://www.alchemer.com/);*
6. *Woorise (vezi link: https://woorise.com/);*
7. *Wufoo (vezi link: https://www.wufoo.com/);*
8. *Ask Nicely (vezi link: https://www.asknicely.com/);*
9. *Brandquiz (vezi link: https://www.brandquiz.io/);*
10. *Microsoft Forms (vezi link: https://www.microsoft.com/en-us/microsoft-365/online-surveys-polls-quizzes).*

Recomandări de aplicații / platforme pentru obținerea de feedback:

1. *TrustPilot -Website: www.trustpilot.com;*
2. *ReeVoo -Website: www.reevoo.com;*
3. *Bugherd - Website: www.bugherd.com;*
4. *UserReport -Website: www.userreport.com.*

Recomandări de interogare a elevilor prin intermediul Microsoft Excel

1. *Creați lista elevilor în celulele A1 – A32;*
2. *Într-o celulă separată (spre exemplu B2) afișați aleator un elev care va răspunde tema pentru acasă, care va explica rezolvarea unei probleme sau care va efectua partajarea ecranului atunci când procesul educațional este în regim online.*
3. *În celula B2 vom scrie una din instrucțiuni pentru primii 10 elevi:*
   `=INDEX($A$1:$A$10,RANDBETWEEN(1,COUNTA($A$1:$A$10)),1);`
   `=INDEX($A$1:$A$10,RANDBETWEEN(1,ROWS($A$1:$A$10)),1).`
4. *În celula B3 vom scrie una din instrucțiuni pentru următorii 10 elevi:*
   `=INDEX($A$11:$A$20,RANDBETWEEN(1,COUNTA($A$11:$A$20)),1);`
   `=INDEX($A$11:$A$20,RANDBETWEEN(1,ROWS($A$11:$A$20)),1).`