Laborator 5:
CĂUTAREA BINARĂ. CODUL GRAY

Întocmit de: Claudia Pârloagă

Îndrumător: Asist. Drd. Gabriel Danciu

I. NOŢIUNI TEORETICE

A. Căutarea binară

Algoritmul de **căutare binară** găseşte poziţia unui anumit element într-un şir de numere sortat.
Date de intrare: un şir de numere sortat crescător A. Fie x, o valoare dată.
Date de ieşire: poziţia pe care se află x în şirul A
Cum funcţionează:

- **acţiunile algoritmului:** la fiecare pas, se compară valoarea dată x cu elementul care se află la mijlocul şirului de numere. Dacă sunt egale, s-a găsit elementul şi se returnează poziţia pe care s-a găsit.
- altfel:
  - dacă x este mai mic decât elementul aflat la mijlocul şirului atunci se repetă acţiunile algoritmului în subşirul aflat în stânga mijlocului şirului
  - altfel, dacă x este mai mare decât elementul aflat la mijlocul şirului atunci se repetă acţiunile algoritmului în subşirul aflat în dreapta mijlocului şirului
  - dacă şirul în care se face căutarea ajunge la dimensiunea zero atunci elementul nu se găseşte în şirul dat.

Matematic:

- dacă $x = \left[\frac{n}{2}\right]$ adică dacă este egal cu valoarea din mijlocul şirului atunci am găsit elemetul x pe poziţia $n/2$
- altfel:
  - dacă $x < \left[\frac{n}{2}\right]$ caut în vector între poziţiile $\left[0, \frac{n}{2} - 1\right]$
  - dacă $x > \left[\frac{n}{2}\right]$ caut în vector între poziţiile $\left[\frac{n}{2} + 1, n - 1\right]$
- căutarea în unul din cele 2 intervale se face în mod similar: compar cu jumătatea intervalului. Dacă elementul este egal cu x, stop, altfel verific în care parte se poate afla x.

1. Descrierea algoritmului

1. **Recursiv**

```
1
2  CautareBinara(A,x,st,dr)
3  {
4   //testăm dacă şirul este gol
5   if(dr < st)
6        return "nu_s-a_găsit_x"
7   else
8     {
9       //calculăm mijlocul şirului
10      mid = (st + dr)/2
11      if(A[mid] > x)
12         // x se află în subşirul stâng
13          CautareBinara(A,x,st,mid-1);
14        else if(A[mid] < x)
15                  //x se află în subşirul drept
16                  CautareBinara(A,x,mid+1,dr);
17                else
18                  //s-a găsit x
19                       return mid
20
21    }
22 }
```

Iniţial: $st = 0$ şi $dr = n - 1$

La fiecare pas se caută între poziţia st şi dr. Se urmează paşii algoritmului de căutare binară.

2. **Iterativ**

```
1  CautareBinara(A,x,st,dr)
2  {
3    //se face căutarea cât timp şirul nu este gol
4    while(dr >= st)
5      {
6        //calculăm mijlocul şirului
7        mid = (st + dr)/2
8
9        //
10       if(A[mid] < x)
11          //căutarea se va face în subşirul drept
12          //modificăm mid
13          mid = mid + 1
14        else if(A[mid] > x)
15                  //căutarea se va face în subşirul stâng
16                  mid = mid - 1
17                else
18                  //x=mid
19                  return mid;
20     }
21   //nu s-a găsit x
22   return "nu s-a găsit x"
23 }
```

B. Problema comis-voiajorului

Un comis-voiajor trebuie să viziteze un număr de n oraşe. Iniţial, el se află în oraşul 1. Comis-voiajorul nu trebuie să treacă de două ori prin acelşi oraş iar la întoarcere să revină în oraşul de unde a plecat.

Dacă se cunosc drumurile între oraşe să se determine toate posibilităţile de efectuare a parcursului.

Problema poate fi descrisă astfel:

Fie $G = (V, E)$ un graf neorientat în care oricare două vârfuri ale grafului sunt unite printr-o latură căreia îi este asociat un cost strict pozitiv. Cerinţa este de a determina un ciclu care începe de la un nod aleatoriu a grafului, trece exact o singură dată prin toate celelalte noduri şi se întoarce la nodul iniţial, cu condiţia ca ciclul să aibă un cost minim.

Problema poate fi rezolvată folosind metoda backtracking sau strategia greedy.

Exemplu:

Figura 1: Problema comis-voiajorului

În imaginea de mai sus, în graful necolorat vârfurile reprezintă oraşele iar muchiile sunt drumurile între oraşe, cu un cost asociat.

Considerăm întâi ciclul A-B-D-C-E-A(marginile colorate în verde) al cărui cost este $1+4+1+2+5=13$.

Dacă vom considera ciclul B-A-C-D-E-B(marginile colorate în roşu) costul va fi $1+2+1+3+3=10$.

Astfel, al doilea ciclu are costul mai mic, deci este mai bun.

C. Codul Gray

Codul Gray sau **"reflected binary code"** este un sistem de numeraţie binar, numit după Frank Gray, cel care l-a patentat în 1953.

Un cod Gray de ordin n este un şir care conţine toate numerele de la 0 la 2^{n-1} astfel încât orice două numere consecutive din şir să difere, în reprezentarea lor binară, prin exact un bit.

Construirea unui cod Gray pe n-biţi:

- acesta poate obţinut recursiv astfel:
  - se reflectă lista codului Gray pentru $n-1$-biţi;
  - intrările din lista iniţială se prefixează cu 0-binar;
  - intrările din lista obţinută în urma reflectării se prefixează cu 1-binar;
  - se concatenează cele doua liste obţinute în urma prefixărilor.

Exemplu:

- pentru $n=1$:
  - codul Gray pentru 1 bit este: G_1=0, 1
- pentru $n=2$:
  - lista iniţială: 0, 1
  - lista reflectată: 1, 0
  - prefixarea cu 0: 00, 01
  - prefixarea cu 1: 11, 10
  - concatenarea: G_2=00, 01, 11, 10
- pentru $n=3$:
  - lista iniţială: 00, 01, 11, 10
  - lista reflectată: 10, 11, 01, 00
  - prefixarea cu 0: 000, 001, 011, 010
  - prefixarea cu 1: 110, 111, 101, 100
  - concatenarea: G_3=000, 001, 011, 010, 110, 111, 101, 100

Proprietăţi:

- fiecare element al codului Gray este un şir de n biţi;

- G_n este considerat a fi prima jumătate a lui G_{n+1}
- fiecare element din G_n apare exact o singură dată în listă;
- oricare 2 elemente consecutive diferă prin exact 1 bit;
- ultimul element din G_n diferă de primul element prin exact 1 bit; avem, astfel, **ciclitate** pentru codul Gray;
- **adiacenţa**: trecerea de la o cifră zecimală la următoare sau la precedenta necesită modificarea unui singur bit din elementul de cod.

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Căutarea binară

Codul de mai jos exemplifică implementarea algoritmului de căutare binară:

- recursiv

```java
package cautari;

import java.util.Scanner;

public class CautareBinara {


   public static int search(int[ ] a,  int key, int imin, int imax)
   {
      int middle;

      if (imax <= imin)
         return -1;
      else
      {
         middle = (imin+imax)/2;

         if ( a[middle] > key)
            // Valoarea cautate este mai mica decat a[middle], cautarea se face inainte de middle.
            return search(a, key, imin, middle-1);
         else  if( a[middle] < key)
            // Valoarea cautate este mai mare decat a[middle], cautarea se face dupa middle.
            return search(a, key, middle+1, imax);
         else
               return middle;
      }
   }

   public static void main(String[] args) {
      Scanner s=new Scanner(System.in);
      int[] a={ 2, 4, 6, 8, 10, 12, 14 };
      System.out.println("Introduceti numarul pe care vreti sa il cautati: ");
      int key=s.nextInt();

      System.out.println("Se incepe cautarea...");
      System.out.print("Este " + key + " in sirul dat? ");
      int answer = search(a, key, 0, a.length);
      if (answer == -1)
            System.out.println("Nu.");
       else
            System.out.println("Da, pe pozitia [" + answer + "].");
   }
}
```

- iterativ

```java
package cautari;

import java.util.Scanner;

public class CautareBinara {


   public static int search(int[ ] a,  int key, int imin, int imax)
   {
      int middle;

      if (imax <= imin)
         return -1;
      else
      {
         middle = (imin+imax)/2;

         if ( a[middle] > key)
            // Valoarea cautate este mai mica decat a[middle], cautarea se face inainte de middle.
            return search(a, key, imin, middle-1);
         else  if( a[middle] < key)
            // Valoarea cautate este mai mare decat a[middle], cautarea se face dupa middle.
            return search(a, key, middle+1, imax);
         else
               return middle;
      }
   }

   public static void main(String[] args) {
      Scanner s=new Scanner(System.in);
      int[] a={ 2, 4, 6, 8, 10, 12, 14 };
      System.out.println("Introduceti numarul pe care vreti sa il cautati: ");
      int key=s.nextInt();

```

```
35          System.out.println("Se incepe cautarea...");
36          System.out.print("Este " + key + " in sirul dat? ");
37          int answer = search(a, key, 0, a.length);
38          if (answer == -1)
39                System.out.println("Nu.");
40           else
41                System.out.println("Da, pe pozitia [" + answer + "].");
42      }
43  }
```

III. TEMĂ

Pentru fiecare din următoarele probleme se va scrie pseudocodul şi apoi codul în Java corespunzător.

Problema 1:

Fie graful $G = (V, E)$ care este reprezentat printr-o matrice de adiacenţă. Scrieţi o implementare pentru algoritmul lui Prim, pentru acest caz, care rulează într-un timp $O(V^2)$.

Problema 2:

Fie un graf neorientat $G = (V, E)$ conex cu funcţia cost $w : E \to R$ şi $|E| \geq |V|$.

- fie T un arbore parţial al lui G şi max[u,v]= muchia de cost maxim pe drumul unic între u şi v din T, pentru oricare două noduri $u, v \epsilon V$. Implementaţi un algoritm care primeşte T şi calculează max[u,v] pentru toate $u, v \epsilon V$.
- Scrieţi un algoritm eficient care calculează al doilea cel mai bun arbore parţial de cost minim al lui G.