Laborator 1:
introducere în algoritmi

Întocmit de: Claudia Pârloagă

Îndrumător: Asist. Drd. Gabriel Danciu

I. NOŢIUNI TEORETICE

A. Sortarea prin selecţie

Date de intrare: un şir A, de date
Date de ieşire: şirul A, sortat crescător

```
SELECTION − SORT(A[1..n])
1.  for i ← 1 to n − 1 do
2.      minj ← i; minx ← A[i]
3.      ▷ caut poziţia finală a lui A[i] în şir
4.      for j ← i + 1 to n do
5.          if A[j] < minx then
6.              minj ← j
7.              minx ← A[j]
8.      ▷ şi schimb elementul actual cu cel mai mic găsit
9.      A[minj] ← A[i]
10.     A[i] ← minx
```

Figura 1: Sortarea prin selecţie

Cum funcţionează:

- pornind de la primul element din şir:
  - *minj* reţine poziţia celui mai mic element din şir
  - *minx* reţine cel mai mic element din şir, aflat pe poziţia *minj*
- se parcurge subşirul ce urmează elementului actual(linia 4 din pseudocod) şi se caută şi se reţine cel mai mic element şi poziţia acestuia; Se va interschimba acest element nou găsit cu elementul actual
- se continuă până când şirul este parcurs până la penultimul element, moment în care şirul este sortat.

B. Sortarea prin inserţie

Date de intrare: un şir A de n numere
Date de ieşire: şirul A, sortat

$INSERTION - SORT(A)$

1. **for** $j \leftarrow 2$ **to** $length[A]$
2. **do** $key \leftarrow A[j]$
3. $\triangleright$ *insert* $A[j]$ *into the sorted sequence* $A[i..j-1]$
4. $i \leftarrow j - 1$
5. **while** $i > 0$ *and* $A[i] > key$
6. **do** $A[i+1] \leftarrow A[i]$
7. $i \leftarrow i - 1$
8. $A[i+1] \leftarrow key$

Figura 2: Sortarea prin inserţie

Cum funcţionează:

- se parcurge şirul de la stânga la dreapta
- la fiecare iteraţie se selectează elementul $A[j]$
- începând cu poziţia $j-1$ elementele sunt mutate cu o poziţie la dreapta până când se va găsi o poziţie potrivită pentru inserarea elementului $A[j]$(liniile 4-8).

C. Calculul şirului lui Fibonacci

Şirul lui Fibonacci este definit prin recurenţa:

$$\begin{cases} f_0 = 0; f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \qquad \text{pentru } n \geq 2 \end{cases}$$

Metode de descriere:

- recursiv:

```
1 FIBONACCI1(n)
2 if(n<2)
3    return n
4 else
5    return FIBONACCI1(n-1)+FIBONACCI1(n-2)
```

- iterativ:

```
FIBONACCI2(n)
i<-1; j<-0
for k<-1 to n do
    j<-i+1
    i<-j-i
return j
```

D. Înmulţirea a două matrici

Condiţie:

- lăţimea primei matrici trebuie să fie egală cu înălţimea celei de-a doua matrici(dacă avem două matrici $A_{m\times n}$ şi $B_{p\times q}$ atunci, dacă dorim să le înmulţim trebuie ca $n = p$)

- Parcurgerea unei matrici

```
TRAVERSEZ(A)
for i=1 to m do
    for j=1 to n do
        => accesăm valoarea elementului A[i,j]
```

- Date două matrici $A_{m\times n}$ şi $B_{n\times p}$ produsul se va calcula astfel:

```
PRODUS(A,B)
for i=1 to m do
    for j=1 to p do
        for k=1 to n do
            C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}

return C
```

E. Algoritmul lui Euclid pentru calculul celui mai mare divizor comun

Algoritmul lui Euclid este o metodă de calcul a celui mai mare divizor comun a două numere. Prezentăm, mai jos, pseudocodul acestui algoritm.

EUCLID(m, n)

1. ***while*** $n \neq 0$ ***do***
2. $\quad temp \leftarrow n$
3. $\quad n \leftarrow$ *restul împărţirii lui* m *la* n
4. $\quad m \leftarrow temp$
5. ***return*** m

Figura 3: Algoritmul lui Euclid

F. Turnurile din Hanoi

Joc matematic în care: se dau 3 tije(numerotate tija1, tija 2 şi tija 3) şi un număr de N discuri de mărimi diferite ampasate pe tija 1.

Scopul este de a muta discurile de pe tija 1 pe tija 3, utilizând tija 2 ca intermediar.

Metoda iterativă de rezolvare presupune:

```
HANOI(n,A,B,C)
if n ≠ 0 then
    HANOI(n-1, A,B,C)
      afisez <<mută discul de pe>> A <<pe >>C
    HANOI(n-1,A,B,C)
```

G. Înmulţirea a la russe

Metoda nu presupune folosirea înmulţirii ci divizarea cu 2 şi adunarea. Algoritmul este:

```
RUSSE(A, B)
1.  arrays X, Y
2.  X[1] ← A; Y[1] ← B
3.  i ← 1
4.  ▷ se construiesc cele două coloane
5.  while X[i] > 1 do
6.      X[i + 1] ← X[i] div 2
7.      Y[i + 1] ← Y[i] + Y[i]
8.      i ← i + 1
9.  prod ← 0
10. ▷ se calculeaza produsul final
11. while i > 0 do
12.     if X[i] este impar then prod ← prod + Y[i]
13.     i ← i − 1
14. return prod
```

Figura 4: Inmultirea a la russe

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Înmulţirea a la russe

Codul de mai jos exemplifică modul de implementare al algoritmului:

```
import java.util.Scanner;

public class Russe {

   public static int arusse(int a, int b)
   {
      int[] x,y;
      int i=1, prod=0;
      x=new int[1000];
      y=new int[1000];
      x[1]=a;
      y[1]=b;

      while(x[i]>1)
      {
         x[i+1]=x[i]/2;
         y[i+1]=y[i]+y[i];
         i=i+1;
      }
      while(i>0)
      {
         if((x[i]%2)!=0) prod=prod+y[i];
         i=i-1;
      }
      return prod;

   }


   public static void main(String[] args) {
      Scanner s=new Scanner(System.in);
      System.out.println("Se efectueaza inmultirea a doua numere:");
      System.out.println("Primul numar:");
      int a=s.nextInt();
      System.out.println("Al doilea numar:");;
      int b=s.nextInt();
      System.out.println("Produsul celor 2 numere este "+arusse(a,b));

   }

}
```

Observatie!

Cele două numere a căror produs se calculeaza vor fi introduse de către cel care rulează programul.

B. Algoritmul lui Euclid pentru calculul celui mai mare divizor comun

Implementarea algoritmului:

```
import java.util.Scanner;

public class AlgEuclid {

   public static int cmmdc(int m, int n)
   {
      int temp;

      while(n!=0)
      {
         temp=n;
         n=m%n;
         m=temp;
      }
      return m;
   }

   public static void main(String[] args) {
      Scanner s=new Scanner(System.in);
      System.out.println("Calculul celui mai mare divizor comun folosind algoritmul lui Euclid");
      System.out.println("Primul numar:");
      int m=s.nextInt();
      System.out.println("Al doilea numar:");
      int n=s.nextInt();
      System.out.println("Cel mai mare divizor comun este:"+cmmdc(m,n));
   }
}
```

III. TEMĂ

Problema 1: Se dau:

Date de intrare: o secvenţă de n numere $A = (a_1, a_2, ..., a_n)$ şi o valoare v
Date de ieşire: poziţia i astfel încât $v = A[i]$ sau -1 în cazul în care v nu apare în A.
Scrieti o aplicaţie pentru căutarea secvenţială, care caută în secvenţa de numere pentru a-l găsi pe v.

Problema 2:

Să se adune 2 numere întregi în formă binară pe n-biţi, reţinute în 2 vectori A şi B de $n-$elemente.
Suma celor 2 numere se va reţine în formă binară, într-un vector C de $n+1-$elemente.

Problema 3:

Se consideră problema evaluării unui polinom.
Se dau n coeficienţi $a_0, a_1, ..., a_{n-1}$ şi un număr real x să se calculeze $\sum_{i=0}^{n-1} a_i x^i$.
Implementţi un algoritm de complexitate $\Theta(n^2)$ pentru această problemă.
Implementaţi un algoritm de complexitate $\Theta(n)$ care utilizează următoarea metodă(numită regula lui Horner) pentru a rescrie polinomul:
$\sum_{i=0}^{n-1} a_i x^i = (\cdots(a_{n-1}x + a_{n-2})x + \cdots + a_1)x + a_0$
Pentru schema lui Horner: http://ro.wikipedia.org/wiki/Schem%C4%83_Horner