EXAMEN STRUCTURI DE DATE

Subiectul 1

1. Se consideră secvenţele $S_1=\{2,7,8,10\}$ şi $S_2=\{1,3,4,5\}$.

0,2p a) Care este rezultatul interclasării lui S_1 cu S_2 ?

0,4p b) Care este ordinul de operaţii pentru interclasarea a două mulţimi de n elemente fiecare?

0,4p c) Care este ordinul de operaţii pentru căutarea unei valori într-o mulţime sortată cu n elemente?

2. Fie **B** un arbore binar cu **n** noduri în care, pentru fiecare nod, subarborele stâng are acelaşi număr

2p de noduri cu subarborele drept. Să se arate că **n+1** este o putere a lui **2**.

3. Structura de date vector unidimensional sortat (**SortArray**):

0,2p • exemplu

0,2p • declaratie structură de date **C++** (cu **Template**),

0,8p • implementarea metodei de căutare

0,8p • analiza algoritmului de cautare

4. Sortarea prin selecţie:

0,4p • exemplu,

1,0p • implementare C++,

0,6p • ordinul de operaţii (fara demonstratie)

5. Fie T_{i+1} numărul de biţi care se schimbă în reprezentarea binară a lui **i** când se generează succesorul

1,0p său **i+1**. Să se determine $\sum_{i=0}^{n-1} T_i$.

6. Pentru un arbore binar, scrieţi o metodă:

1,0p

int BTree::getNumDegree1();

care determină numărul de noduri de grad 1 din arbore

TOTAL: 9p+1p oficiu=10p

Subiectul 1 ✓

0,2 (1) a) Rezultatul interclasării este: {1, 2, 3, 4, 5, 7, 8, 10}

b) $O(n)$

0,4 c) $O(\log_2 n)$

2p (2) Numărul maxim de noduri dintr-un arbore de adâncime k (rădăcină are k nivele) este $2^k - 1$

Din datele problemei avem nr. de noduri $= n \Rightarrow$

$\Rightarrow n = 2^k - 1 \Rightarrow (n+1 = 2^k)$

(3)

Un exemplu de vector sortat este $V = (3, 4, 7, 8, 12, 20, 27)$

Clasa SortArray nu are nicio dată nouă ci doar adaugă metode noi clasei Array pe care o moștenește.

Aceste metode sunt:

(i) adaugă o valoare constantă

(ii) șterge o valoare

(iii) interclasează vectorul curent cu un vector sa.

(iv) inserează o valoare în vectorul curent.

(v) inserează succesiv toate valorile unui vector sortat „sa" în vectorul curent

(vi) elimină o valoare din vectorul curent

(vii) elimină succesiv toate valorile vectorului „sa" din vectorul curent

se implementează astfel:

```
template <class T>
class SortArray: public Array<T>
{
public:
SortArray & addValue (T const & val);
SortArray & RemoveVal.( T const & val);
SortArray & Merge ( SortArray<T> const & sa);
SortArray & operator+( T const & val);
SortArray & operator + (SortArray<T> cons & sa);
```

```
SortArray& operator-(T const& val);
SortArray& operator-(SortArray<T>& sa);
int findValue(T const& val);
};
```

Obs. Toate metodele din SortArray au $T(n) \in O(n)$.

Metoda findValue caută binar în vectorul sortat valoarea val și returnează poziția acesteia dacă o găsește și base-1 în caz contrar.

Căutarea se face astfel

left=0 mid = (left+right)/2 right = lengt-1

Dacă val ∈ [left, mid) atunci right devine mid-1

Dacă val ∈ (mid, right] atunci left devine mid+1

Algoritmul se continuă până right < left.

```
template <class T>
int SortArray<T>::findValue(T const& val)
  {
  int left=0, right=lenght-1, mid;
    while (left<=right)
      {
      mid =(left+right)/2;
      if (data[mid]<val
                  left = mid+1;
      else
            {if (data[mid]>val)
                  right=mid-1;
             else
               return mid;
            }
      }
  return base-1; // nu s-a găsit valoarea.
};
```

Analiza algoritmului de căutare

Dacă notam cu k_2 numărul de căutări și k_1 constanta cu previne am algoritmul de sortare HeapSort atunci în cazul folosirii algoritmului de căutare binară avem

$$T_B(n) = (k_1 n + k_2) \cdot \log_2 n.$$

Dacă nu am fi folosit căutarea binară atunci

$$T_S(n) = k_2 * \frac{n}{2}$$

Observăm că dacă $k_2 \cdot \frac{n}{2} > (k_1 n + k_2)$ atunci este preferabilă căutarea Binară

(4) Sortarea prin selecție

Presupune cunoașterea șirului înainte de a începe sortarea: Principiul este următorul:

Pas 1 se află minimul șirului $\{a_1, a_2, \dots, a_n\}$ și se trece acest șir pe poziția 1

Pas 2 se află minimul șirului $\{a_1, a_2, \dots a_n\}$ | minimul anterior și se trece pe poziția a-II-a.

Se continuă algoritmul până la epuizarea șirului

Acest algoritm are $T(n) \in O(n^2)$

Exemplu: Fie șirul $a = \{8, 3, 5\}$

Pas 1

$\min\{8,3,5\} = 3 \Rightarrow poz[0] = 3$

$\min\{8,5\} = 5 \Rightarrow poz[1] = 5$

$poz[2] = 8.$

$poz = (3, 5, 8)$ vectorul sortat.

Acest algoritm are următoarea implementare.

```
void Sort (Array<T> &V)
{ int j, k, max, n = v.getLenght();
  T temp; // pentru interschimbare.
```

```
for (j=n-1; j>0; j--)
  {
   max=j;
   for (k=j-1; k>=0; k--)
        if (v[k] > v[max])
                max=k; // se muta elementul pe poz k
                temp=v[max];
                v[max]=v[j];
                v[j]=temp;
   }.
}
};// sfârşitul clasei;
```

Finalul va fi $v[j] = \max_{k=\overline{0,j}} v[k]$, $j = \overline{0, n-1}$

$v[0] = v[0]$

$v[1] = \max_{k=\overline{0,1}} \{v[0], v[1]\}$.

$v[2] = \max_{k=0,2} \{v[0], v[1], v[2]\}$.

⋮

$v[n-1] = \max_{k=\overline{0,n-1}} \{v[0], v[1], \dots v[n-1]\}$

(6)
```
int getdegree 1 (BTree * bt)
  {if (bt!=NULL)
        {
        if (bt->st==NULL) && (bt->dr!=NULL)
                        return 1;
        else
        {if(bt->st!=NULL) && (bt->dr==NULL);
                        return 1+ getdegree (bt->st);
        else  // noduri de grad 0 sau 2.
                        return getdegree(bt->st)+getdegree(bt->dr);
        }
        }
  }.
```

```
#include<iostream.h>
#include<stdlib.h>
#include<string.h>
template<class T>
class vector
{
protected:
T* data; //pointer la o zona de memorie cu elemente de tip T
unsigned int lungime;//lungimea vectorului
char* nume;//adresa sirului de caractere ce contine numele vectorului
public:
vector(char*text,int n)//constructor de initializare;
      {
      lungime=n;
      data=new T[n];
      nume=new char(strlen(text)+1);
      strcpy(nume,text);
      }
vector();//constructor implicit
vector(const vector &);//costructor de copiere a vectorului v in obiectul curent

void citire();//citirea unui vector

void afisare();//afisare unui vector
};//inchiderea clasei
template<class T> void vector<T>::citire()
      {
      cout<<"citire vectorului "<<nume<<endl;
      for(int i=0;i<lungime;i++)
            {
            cout<<nume<<"["<<i<<"]= ";
            cin>>data[i];
            }
      }
template<class T>void vector<T>::afisare()
{
if(lungime>0)
      {
      cout<<"elementele vectorului " <<nume<<":"<<endl;
      for(int i=0;i<lungime;i++)
      cout<<nume<<"["<<i<<"]="<<data[i]<<endl;
      }
}
template<class T>vector<T>::vector( const vector<T> &v)
      {
      lungime=v.lungime;
      data=new T[lungime];
      for(int i=0;i<lungime;i++)
            data[i]=v.data[i]; //copierea elementelor
      nume=new char(strlen(v.nume));
      strcpy(nume,v.nume);
      }


void main()//functia principala
{
vector<float> a("A",2);//s-a apelat constructorul
a.citire();
a.afisare();
}
```