# EXAMEN STRUCTURI DE DATE

**Subiectul 3**

**1.** Se consideră polinomul de o variabilă **P** ∈ **R**[X] cu reprezentarea sub formă de listă sortată (**SortArray**) P=(4,(7,5),(6,4),(5,3),(1,0)).

1p

Se cere:

0,5p a) Ce grad are **P** şi ce cantitate de memorie este folosită la stocarea lui **P** dacă sizeof(int)=2 şi sizeof(double)=6 ?

0,5p b) Care este reprezentarea lui P dacă s-ar folosi vectori de lungime 10 pentru stocarea tuturor coeficientilor $a_0$, $a_1$, ..., $a_9$ pentru polinoame de grad cel mult 9?

**2.** Un tip particular de matrice pătratică de ordinul n este matricea bandă de lăţime 3, de forma:

2p

$$\begin{bmatrix} x & x & & & & \\ x & x & x & & O & \\ & x & x & x & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & O & & & x & x & x \\ & & & & & x & x \end{bmatrix}$$

Scrieţi o formula matematica Adr:$\{1,2,...,n\}^2 \rightarrow \{1,2,...,3n-2\}$ care determină o reprezentare optimă a acestei matrici, memorând numai elementele nenule.

**3.** Arborele binar de căutare:

0,4p • exemplu,

0,4p • declaratia clasei **C++** (folosind Templates),

1,2p • implementarea metodei de cautare valoare.

**4.** Sortarea prin interclasare (mergesort):

2p

0,2p • exemplu,

0,4p • descrierea algoritmului,

1,0p • clasă C++ de implementare a algoritmului,

0,2p • ordin de operaţii (formula, fara calcule),

0,2p • ordin de locaţii (formula, fara calcule).

**5.** Considerăm următorul algoritm de generare a numerelor aleatoare: se porneşte cu un număr de două cifre $X_0 = \overline{AB}$ şi se ridică la pătrat, obţinându-se numărul de patru cifre $Y_0 = \overline{CDEF}$ din care se extrage următorul număr aleator $X_1 = \overline{DE}$ şi se repetă procedeul cu el.

0,5p i) Să se demonstreze că şirul $(X_n)_{n\in N}$ este periodic de perioadă $T_{X_0}$ oricare ar fi $X_0 \in \overline{1,\ 99}$.

0,5p ii) Să se determine o valoare pentru $X_0$ astfel încât şirul $(X_n)_{n\in N}$ să aibă perioada $T_{X_0}$ minimă si specificati valoarea perioadei.

**6.** Pentru un arbore binar, scrieţi o metodă:

1p

**int** BTree::getNumDegree2();

care determină numărul de noduri de grad 2 din arbore

**TOTAL:** 9p+1p oficiu=10p

# Subiectul 3

(2)

$$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{matrix}$$

| $a_{00}$ | $a_{01}$ | $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ | $a_{32}$ | $a_{33}$ |
|---|---|---|---|---|---|---|---|---|---|

$a_{00}$ are adresa $\alpha$

$i=j \Rightarrow a_{ij} = \alpha + 2 + 3(i-1) + 1 = \alpha + 3i$

$j=i-1 \Rightarrow a_{ij} = \alpha + 2 + 3(i-1) + \ldots = \alpha + 3i - 1$

$j=i+1 \Rightarrow a_{ij} = \alpha + 2 + 3(i-1) + 2 = \alpha + 3i + 1$

$\Rightarrow a_{ij} = \alpha + 2i + j$

(1) a)

sizeof(int) + 4 * (sizeof(double) + sizeof(int)) =

$= 2 + 4 \cdot (2+6) = 34.$

b) $10 \times$ sizeof(double) $= 60.$

(3) Arborele binar de căutare

Definiție

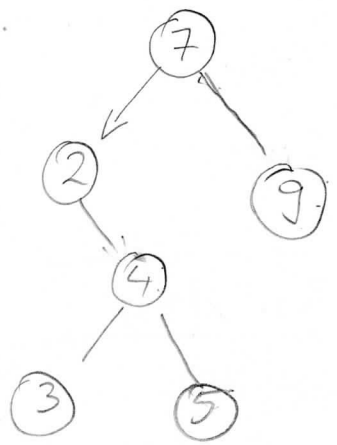
Un arbore de căutare binară este un arbore binar vid sau în care fiecare nod conține o cheie (după care se face ordonarea) ce satisface la următoarele condiții:

(i) toate cheile din subarborele stâng preced cheia din rădăcină.

(ii) cheia din rădăcină precede toate cheile din subarborele drept

(iii) subarborii stâng și drept sunt la rândul lor arbori de căutare.

Exemplu:




Pentru implementare vom construi o clasă btree cu datele: val, *st, *dr
și metodele: constructor implicit destructor, inserare, citire, inordine, preordine, postordine, transversal. Menționăm că la aceste metode mai pot fi adăugate și altele

Implementare:

```
class btree
{
public
int val;
btree  *st, *dr;
 btree();     constructor
~btree();     destructor.
 btree* inserare ( btree * &, int); // inserează
 btree * citire ();                  // citire
 void inordine ( btree * bt) // parcurgere în ordine.
 void preordine (btree *bt) // parcurgere în preordine
 void postordine (btree *bt) // parcurgere în postordine
 void transversal ( btree *bt) // parcurgere transversală
};
 btree ::btree ()
     {
      st =NULL;
      dr=NULL;
     }
 btree ::~btree()
      {
       destroy (this)
      }.
```

Căutarea unei chei în arborele binar de căutare.
returnează NULL dacă nu găsește cheia sau
pointer la nodul cu valoarea cheii.
Căutarea se face recursiv sau iterativ.

```
btree * btree::cautaRe ( btree * bt, int info)
{
 if (bt==NULL)
        return NULL;
 if (bt->val == info)
                return bt;
 if (bt->val > info
        return cautaRe (bt->st, info ); // caută în
                                           stânga
 else
     return cautaRe (bt->dr, info ); // caută în
                                        dreapta
}

btree * btree:: caută iterativ (btree *bt, int info )
{
btree * temp // variabilă cu care se parcurge arborele
btree * aux = NULL // păstrează adresa nodului în care se
 if (bt == NULL)      găsește cheia.
        return NULL;
 if ( bt->val ==info)
        return bt;
 temp = bt;
 while ( temp != NULL)
     { if (temp->val==info)
        aux = temp;
       if (temp->val < info)
                temp = temp -> dr ; // caută în dreapta
       else
                temp = temp -> st ; // căuta în stânga.
     }
 return aux;
}
```

(4) Sortarea prin interclasare.

Este o metodă care are $T(n) \in \Theta(n \log_2 n)$. Această metodă se bazează pe următoarea idee: se împarte tabela de $N$ elemente în două tabele care se sortează separat și apoi se interclasează cele două tabele.
Pentru această metodă avem următorul algoritm în Pseudocod.

```
procedure MergeSort(S)
begin
  n = |S|
  if n<=1 then return;
  {
   Împarte S={s1, ..., sn} în două subsecvențe.
        S1={s1, ... s[n/2]} și S2={s[n/2]+1, ..., sn}
  }
MergeSort(S1);
MergeSort(S2);
interclasăm clasele.
end
```

Această metodă are ordinul de locații.

$$L(n) = n \log_2 n$$

și ordinul de operații.

$$T(n) = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1, \forall n \geq 2.$$

care se obține din relația de recurență

CODUL în C++ (de căutat).

Pentru implementarea algoritmului avem nevoie de următoarele funcții:

```
void divide ( int v[100], int ls, int ld)
  {
    if (ld - ls <= 1)
        sort (ls, ld);
    else // înseamnă card > 1
    {
      divide (v, ls, (ls+ld)/2 ); //partea stângă
      divide (v, (ls+ld)/2+1, ld); // partea dreaptă
      merge (v, ls, (ls+ld)/2, ld); // interclasarea
    }.
```

```
void sort (int v[100], int ls, int ld) // ld - ls ≤ 1
                                       // înțeleasă
  {
    int aux;
    if v[ls] > v[ld]
        {
        aux = v[ls];
        v[ls] = v[ld];
        v[ld] = aux;
        }
  }.
```

```
void merge (int v[100], int ls, int mijl, int ld)
  {
  int aux[100];
  int i, j, k; // i pentru parcurgere prima parte
               // j pentru parcurgere a-II-a parte
               // k pentru parcurgere vector auxiliar
  i = ls; // începutul primei secvențe.
  j = mijl + 1; // începutul celei de-a II-a secv.
  k = 0; // începutul vectorului sortat.
```

(6).

```
int BTree::getNumDegree2(BTree * bt)
{
  if (bt!=NULL)
  {
    if (bt->st!=NULL)&&(bt->dr!=NULL)
        return 1+getNumDegree2(bt->st)+getNumDegree2(bt->dr)
    else
    getNumDegree2(bt->st)+getNumDegree2(bt->dr).
  }.
}.
```

---

```
while((i<=mijl)&&(j<=ld)) // parcurgem și comparăm elementele din cele două secvențe. simultan.
  }
  if(V[i]<V(j))
          {
          aux[K]=V[i]
           i++;
           K++;
          }
  else {aux[K]=V[j];
        j++;
        K++;
       }.
  }
// testăm care secvență s-a terminat.
if (i<mijl) // mai sunt elemente în prima secv.
    while (i<=mijl
          {
          X[K]=V[i];
           i++;
           K++;
          }
else
  while (j<= ld) // mai sunt elemente în a-j-a secvență
          {
          X[K]=V[j]; j++; V++; }
// Elementele sortate sunt în aux. afișare(aux[100];
```