Laborator 3:
structuri de date(II)

Întocmit de: Claudia Pârloagă

Îndrumător: Asist. Drd. Gabriel Danciu

I. NOŢIUNI TEORETICE

A. Arbori binari

Un **arbore binar** este o structură de date compusă din **noduri**.

PROPRIETĂŢI:

- fiecare nod are 0,1 sau 2 descendenţi(copii)
- **frunza** este un nod care nu are copii
- **dimensiunea** unui arbore este dată de numărul total de noduri
- **adâncimea** unui nod este egală cu lungimea drumului de la rădăcina arborelui la nod
- mulţimea nodurilor care au aceeaşi adâncime n se numeşte **nivelul** n al arborelui
- două noduri care au acelaşi părinte se numesc **fraţi**. Se disting unul faţă de celălalt prin poziţia lor faţă de părinte(stânga, dreapta)
- pentru un arbore cu înălţimea h numărul minim si cel maxim de noduri este dat de:

$$h + 1 \leq n \leq 2^{h+1} - 1$$

- înălţimea minimă şi cea maximă pentru un arbore cu n noduri este dată de:

$$|log_2(n + 1) - 1| \leq h \leq n - 1$$

- **arbore binar plin:** arbore binar care are exact $2^{h+1} - 1$ noduri
- **arbore binar complet:** un arbore plin în care toate frunzele au aceeaşi adâncime
- **numărul lui Catalan=**$\frac{(2n)!}{(n+1)!n!}$: determină câţi arbori distincţi cu n noduri se pot construi

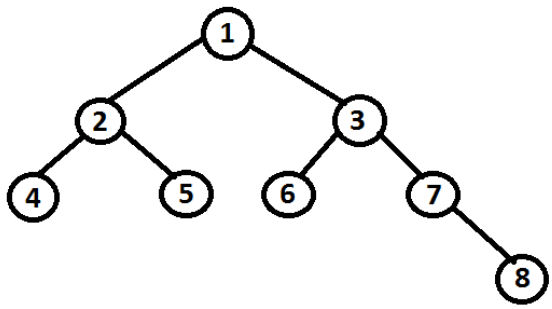

Figura 1: Arbore binar

B. Parcurgeri de arbori

1. Preordinea(RSD)

Preordinea presupune parcurgerea arborelui binar astfel:

- rădăcină(R)
- subarbore stâng(S)
- subarbore drept(D)

Fiecare subarbore este la rândul lui un arbore binar.

2. Inordinea(SRD)

Inordinea presupunea parcurgerea arborelui binar astfel:

- subarbore stâng
- rădăcina
- subarbore drept

3. Postordinea(SDR)

Postordinea presupunea parcurgerea arborelui binar astfel:

- subarbore stâng
- subarbore drept
- rădăcina

Pentru arborele binar prezentat în figura de mai sus vom scrie cele trei parcurgeri:

- preordine: 1, 2, 4, 5, 3, 6, 7, 8
- inordine: 4, 2, 5, 1, 6, 3, 7, 8
- postordine: 4, 5, 2, 6, 8, 7, 3, 1

C. Heap-uri

Un heap reprezintă un şir de obiecte care poate fi interpretat ca un arbore binar:

- unui nod din arbore îi corespunde un element din şir
- fiecare element din şir reţine valoarea unui nod al arborelui
- **valoarea fiecărui vârf este mai mare sau egală cu cea a fiecărui fiu al său**

```
PERCOLATE(T[1..n], i)
1.      ▷ se filtrează valoarea din T[i]
2.      k ← i
3.      repeat
4.          j ← k
5.          if j > 1 and T[j div 2] < T[k] then k ← j div 2
6.          interschimba T[j] și T[k]
7.      until j = k
```

Figura 2: Pseudocod

- dacă valoarea unui vârf creşte şi depăşeşte valoarea tatălui atunci se vor interschimba între ele aceste 2 valori până când este îndeplinită proprietatea de heap(operaţia se numeşte PERCOLATE)

- dacă valoarea unui vârf scade şi devine mai mică decât a unui fiu atunci schimbăm valoarea cu cea mai mare valoare a fiilor şi continuăm procedura până când este îndeplinită proprietatea de heap(operaţia se numeşte SHIFT_DOWN)

```
SIFT_DOWN(T[1..n], i)
1.      ▷ se cerne valoarea din T[i]
2.      k ← i
3.      repeat
4.          j ← k
5.          ▷ găsește fiul cu valoarea cea mai mare
6.          if 2j ≤ n and T[2j] > T[k] then k ← 2j
7.          if 2j < n and T[2j + 1] > T[k] then k ← 2j + 1
8.          interschimba T[j] și T[k]
9.      until j = k
```

Figura 3: Pseudocod

- metoda de creare a unui heap:

```
MAKE − HEAP(T[1..n])
1.      ▷ formează din T un heap
2.      for i ← (n div 2) to n do sift − down(T, i)
```

Figura 4: Pseudocod

- procedura de sortare a unui şir folosind heap:

```
HEAP_SORT(T[1..n])

1.      ▷ sortăm tabloul T
2.      MAKE_HEAP(T)
3.      for i ← n downto 2 do
4.          interschimbă T[1] şi T[i]
5.          ▷ SIFT − DOWN(T[1..i − 1], 1)
```

Figura 5: Pseudocod

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Arbori binari

Codul de mai jos prezintă un mod de implementare al unui arbore binar, împreună cu cele 3 tipuri de parcurgeri: inordine, preordine, postordine:

```
1  package btn;
2
3  public class BTNode
4  {
5    public Object data;//informatia dintr-un nod
6    private BTNode left, right;
7
8    public BTNode (Object initialData, BTNode initialLeft, BTNode initialRight)
9    {
10     data = initialData;
11     left = initialLeft;
12     right = initialRight;
13   }
14
15   public Object getData () { return data; }
16   public BTNode getLeft () { return left; }
17   public BTNode getRight () { return right; }
18
19   public void setData (Object newData) { data = newData; }
20   public void setLeft (BTNode newLeft) { left = newLeft; }
21   public void setRight (BTNode newRight) { right = newRight; }
22
23   public Object getLeftMostData ()
24   {
25     if( left == null )
26       return data;
27     else
28       return left.getLeftMostData ();
29   }
30
31   public Object getRightmostData ()
32   {
33     if( right == null )
34       return data;
35     else
36       return right.getRightmostData ();
37   }
38
39 //traversarea in inordine
40   public void inorderPrint (int depth)
41   {
42     if( left != null )
43       left.inorderPrint (depth + 1);
44     for( int i = 0 ; i < depth ; i ++)
45       System.out.print ("___");
46     System.out.println (data);
47     if( right != null )
48       right.inorderPrint (depth + 1);
49
50   }
51
52     //metoda determina daca un nod este sau nu frunza
53    //=> true daca nodul este frunza
54    //=> false altfel
55   public boolean isLeaf ()
56   {
57     return (left == null) && (right == null);
58   }
59
60 //postordine
61   public void postorderPrint (int depth)
62   {
63     if( left != null )
64       left.postorderPrint (depth + 1);
65     if( right != null )
66       right.postorderPrint (depth + 1);
67     for( int i = 0 ; i < depth ; i ++)
68       System.out.print ("___");
69     System.out.println (data);
70   }
71
72 //traversarea in preordine
73   public void preorderPrint (int depth)
74   {
75     for( int i = 0 ; i < depth ; i ++)
76       System.out.print ("___");
77     System.out.println (data);
78     if( left != null )
79       left.preorderPrint (depth + 1);
80     if( right != null )
81       right.preorderPrint (depth + 1);
82   }
83
84
85 //numara cate noduri sunt in arborele binar
86   public static int treeSize (BTNode root)
87   {
88     if( root == null )
```

```
89        return 0;
90      else
91        return 1 + treeSize (root.left) + treeSize (root.right);
92    }
93  }
```

Clasa de testare:

```
1  package btn;
2
3  public class Demo
4  {
5    public static void main (String[] args)
6    {
7      /*BTNode a = new BTNode (new Character('O'), null, null);
8      BTNode b = new BTNode (new Character('R'), null, null);
9      BTNode c = new BTNode (new Character('L'), a, b);
10     a = new BTNode (new Character('T'), null, null);
11     b = new BTNode (new Character('G'), null, a);
12     a = new BTNode (new Character('A'), c, b);*/
13
14     BTNode a = new BTNode (15, null, null);
15     BTNode b = new BTNode (18, null, null);
16     BTNode c = new BTNode (12, a, b);
17     a = new BTNode (20, null, null);
18     b = new BTNode (7, null, a);
19     a = new BTNode (1, c, b);
20
21     System.out.println ("Inordine:");
22     a.inorderPrint (0);
23     System.out.println ("---");
24     System.out.println ("Preordine:");
25     a.preorderPrint (0);
26     System.out.println ("---");
27     System.out.println ("Postordine:");
28     a.postorderPrint (0);
29   }
30 }
```

III. TEMĂ

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

Problema 1:

Implementaţi procedura HEAP_INCREASE_KEY(A, i, k) care atribuie $A[i] \leftarrow max(A[i], k)$ şi actualizează structura heap-ului în mod corespunzător, într-un timp de $O(lgn)$.

Problema 2:

Operaţia HEAP_DELETE(A, i) şterge elementul din nodul i al heap-ului A.
Implementaţi operaţia HEAP_DELETE pentru un heap de $n - elemente$.

Problema 3:

Scrieţi un algoritm care uneşte(merge) k liste sortate într-o singură listă sortată