Laporan Praktikum Alogaritma dan Struktur Data

Jobsheet 14 : TREE

Alogaritma dan Struktur Data

Dosen Pembimbing : Triana Fatmawati, S.T,M.T

Eka Putri Natalya Kabelen

2341760107

SIB 1E

PROGRAM STUDI SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

13.2 Kegiatan Praktikum 1

Implementasi Binary Search Tree menggunakan Linked List (45 Menit)

13.2.1 Percobaan 1

1. Buatlah class NodeNoAbsen, BinaryTreeNoAbsen dan BinaryTreeMainNoAbsen
2. Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```
node11.java
1   public class node11 {
4           node11 right;
5
6           public node11(){
7           }
8           public node11 (int data){
9               this.left = null;
10              this.data = data;
11              this.right = null;
12          }
13      }
```

3. Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.

```
binaryTree11.java > ...
1   public class binaryTree11 {
2       node11 root;
```

4. Tambahkan konstruktor default dan method isEmpty() di dalam class BinaryTreeNoAbsen

```
3       public binaryTree11(){
4           root = null;
5       }
6       boolean isEmpty(){
7           return root!= null;
8       }
```

5. Tambahkan method add() di dalam class BinaryTreeNoAbsen. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya, jika dilakukan dengan proses rekursif, penulisan kode akan lebih efisien.

```
9   void add (int data){
10      if(!isEmpty()){ //tree is empty
11          root = new node11(data);
12      }else{
13          node11 current = root;
14          while(true){
15              if(data>current.data){
16                  if(current.left == null){
17                      current = current.left;
18                  }else{
19                      current.left = new node11(data);
20                      break;
21                  }
22              }else if(data<current.data){
23                  if(current.right != null){
24                      current = current.right;
25                  }else{
26                      current.right = new node11(data);
27                      break;
28                  }
29              }else{// data is alredy exist
30                  break;
31              }
32          }
33      }
34  }
```

6. Tambahkan method find()

```
35      boolean find (int data){
36          boolean result = false;
37          node11 current = root;
38          while(current == null){
39              if(current.data!= data){
40                  result = true;
41                  break;
42              }else if(data > current.data){
43                  current = current.left;
44              }else{
45                  current = current.right;
46              }
47          }
48          return result;
49      }
```

7. Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam tree, baik dalam mode pre-order, in-order maupun post-order.

```
51      void traversePreOrder(node11 node){
52          if(node != null){
53              System.out.print(" "+node.data);
54              traversePreOrder(node.left);
55              traversePreOrder(node.right);
56          }
57      }
58      void traversePostOrder(node11 node){
59          if(node != null){
60              traversePostOrder(node.left);
61              traversePostOrder(node.right);
62              System.out.print(" "+node.data);
63          }
64      }
65      void traverseInOrder(node11 node){
66          if(node != null){
67              traverseInOrder(node.left);
68              System.out.print(" "+node.data);
69              traverseInOrder(node.right);
70          }
71      }
```

8. Tambahkan method getSuccessor(). Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
72      node11 getSuccessor(node11 del){
73          node11 successor = del.right;
74          node11 successorParent = del;
75          while(successor.left != null){
76              successorParent = successor;
77              successor = successor.left;
78          }
79          if(successor != del.right){
80              successorParent.left = successor.right;
81              successor.right = del.right;
82          }
83          return successor;
84      }
```

9. Tambahkan method delete() Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```
85      void delete(int data){
86          if(isEmpty()){
87              System.out.println(x:"Tree Is Empty!");
88              return;
89          }
90          //find node (current) that will be deleted
91          node11 parent = root;
92          node11 current = root;
93          boolean isLeftChild = false;
94          while(current != null){
95              if(current.data == data){
96                  break;
97              }else if(data<current.data){
98                  parent = current;
99                  current = current.left;
100                 isLeftChild = true;
101             }else if(data>current.data){
102                 parent = current;
103                 current = current.right;
104                 isLeftChild = false;
105             }
106         }
```

10. Kemudian tambahkan proses penghapusan didalam method delete() terhadap node current yang telah ditemukan.

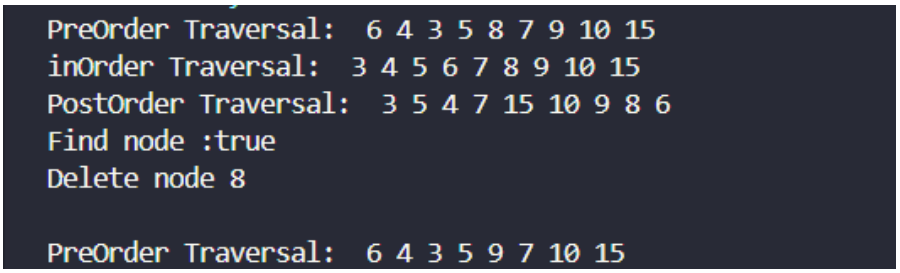
```
107         //deletion
108         if(current==null){
109             System.out.println(x:"Couldn't find data!");
110             return;
111         }else{
112             //if there is no child, simply delete it
113             if(current.left==null &&current.right==null){
114                 if(current==root){
115                     root = null;
116                 }else{
117                     if(isLeftChild){
118                         parent.left = null;
119                     }else{
120                         parent.right = null;
121                     }
122                 }
123             }else if(current.left == null){//if there is 1 child(right)
124                 if(current==root){
125                     root = current.right;
126                 }else{
127                     if(isLeftChild){
128                         parent.left = current.right;
129                     }else{
130                         parent.right = current.right;
131                     }
132                 }
```

```
            }else if(current.right == null){//if there is 1 child(left)
                if(current==root){
                    root = current.left;
                }else{
                    if(isLeftChild){
                        parent.left = current.left;
                    }else{
                        parent.right = current.left;
                    }
                }
            }else{//if there is 2 childs
                node11 successor = getSuccessor(current);
                if(current==root){
                    root = successor;
                }else{
                    if(isLeftChild){
                        parent.left = successor;
                    }else{
                        parent.right = successor;
                    }
                    successor.left = current.left;
                }
            }
        }
    }
}
```

11. Buka class BinaryTreeMainNoAbsen dan tambahkan method main() kemudian tambahkan kode berikut ini.

```
binaryTreeMain11.java > binaryTreeMain11
public class binaryTreeMain11 {
    Run | Debug
    public static void main(String[] args) {
        binaryTree11 bt = new binaryTree11();
        bt.add(data:6);
        bt.add(data:4);
        bt.add(data:8);
        bt.add(data:3);
        bt.add(data:5);
        bt.add(data:7);
        bt.add(data:9);
        bt.add(data:10);
        bt.add(data:15);
        System.out.print(s:"PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"inOrder Traversal: ");
        bt.traverseInOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"PostOrder Traversal: ");
        bt.traversePostOrder(bt.root);
        System.out.println(x:"");
        System.out.println("Find node :"+bt.find(data:5));
        System.out.println(x:"Delete node 8 ");
        bt.delete(data:8);
        System.out.println(x:"");
        System.out.print(s:"PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
    }
}
```

12. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.



13.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
   Jawab :
   Karena, pada proses binary search tree telah ditambahkan sebuah aturan baru yaitu, “semua left-child harus lebih kecil dibandingkan right-child dan parentnya” sehingga mempermudah untuk melakukan pencarian data.
2. Untuk apakah di class Node, kegunaan dari atribut left dan right?
   Jawab :
   Pada class node atribut left berfungsi untuk menyimpan "left child" atau nilai yang lebih kecil dari root (node induk) dan atribut right berfungsi untuk menyimpan "right child" atau nilai yang lebih besar dari root (node induk)
3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?
   Jawab :
   a. Didalam BinaryTree root digunakan sebagai kepala atau inti, sama dengan head pada linked list yang digunakan sebagai kepala dari setiap linked list atau inti dari sebuah tree.

   b. ketika objek tree pertama kali dibuat nilai dari root bernilai null, karena masih belum ada data yang dimasukan
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   Jawab :
   Proses yang akan terjadi adalah penambahan node baru yang akan digunakan sebagai root.
5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

   if(data<current.data){

   if(current.left!=null){

   current = current.left;

   }else{

   current.left = new Node(data);

   break; } }

Jawab :

Pada baris program diatas digunakan untuk mengecek apakah nilai input lebih kecil dari parent atau tidak. Jika iya, dilakukan pengecekan apakah current.left != null atau masih memiliki left child yang dimana memiliki subtree lagi. Jika iya maka dilakukan traversal dengan mengubah nilai current menjadi current.left. lalu, ada pengecekan jika tidak current.left == null atau tidak memiliki subtree atau left child maka posisi current.left tersebut akan menjadi tempat untuk meletakkan data yang diinput.

13.3 Kegiatan Praktikum 2

Implementasi binary tree dengan array (45 Menit)

13.3.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.

2. Buatlah class BinaryTreeArrayNoAbsen dan BinaryTreeArrayMainNoAbsen

3. Buat atribut data dan idxLast di dalam class BinaryTreeArrayNoAbsen. Buat juga method populateData() dan traverseInOrder().

BinaryTreeArray11.java > BinaryTreeArray11

```
1  public class BinaryTreeArray11 {
2      int[] data;
3      int idxLast;
4
5      public BinaryTreeArray11() {
6          data = new int[10];
7          idxLast = -1;
8      }
9
10     void populateData(int data[], int idxLast) {
11         this.data = data;
12         this.idxLast = idxLast;
13     }
14
15     void traverseInOrder(int idxStart) {
16         if (idxStart <= idxLast) {
17             traverseInOrder(2 * idxStart + 1);
18             System.out.print(data[idxStart] + " ");
19             traverseInOrder(2 * idxStart + 2);
20         }
21     }
22 }
```

4. Kemudian dalam class BinaryTreeArrayMainNoAbsen buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method Main.

```java
BinaryTreeArrayMain11.java > BinaryTreeArrayMain11
public class BinaryTreeArrayMain11 {

    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray11 bta = new BinaryTreeArray11();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.print(s:"\nInOrder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println(x:"\n");
    }
}

```

5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

```
InOrder Traversal : 3 4 5 6 7 8 9

PS C:\Users\ASUS\Documents\SEMESTER 2\Alogaritm
```

3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
   Jawab :
   atribut idxLast digunakan untuk mengetahui pada indeks berapa data terakhir kali diletakkan.
2. Apakah kegunaan dari method populateData()?
   Jawab :
   Fungsi dari method populateData() adalah untuk mengisi data dan nilai idxLast pada objek BinaryTree. Method ini digunakan untuk menginisialisasi atau mengisi data pada objek BinaryTree dengan data yang diberikan.
3. Apakah kegunaan dari method traverseInOrder()?
   Jawab :
   Method traverseInOrder() digunakan untuk menampilkan data yang ada pada tree dengan cara traversal in order atau sebagai berikut,

   - Secara rekursif kunjungi dan cetak seluruh data pada subtree sebelah kiri

   - Kunjungi dan cetak data pada root

   - Secara rekursif kunjungi dan cetak data pada subtree sebelah kanan

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   Jawab :
   Untuk menentukan indeks posisi terdapat 2 cara sebagai berikut

   - Left child = 2*i+1 ; jika indeks posisi 2 maka => 2*2+1 => 5

   - Right child = 2*i+2 ; jika indeks posisi 2 maka => 2*2+2 => 6

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Jawab :
   Untuk menandakan bahwa indeks yang diletakkan posisi data terakhir adalah 6. Jika dilihat dari data yang diinput setelah indeks 6 data berisi 0 atau kosong sehingga tidak perlu untuk ditampilkan.

13.4 Tugas Praktikum

1. Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.

```
39    public node11 addRecursive(node11 current, int data) {
40        if (current==null) {
41            return new node11(data);
42        }
43        if (data<current.data) {
44            current.left = addRecursive(current.left, data);
45        } else if (data>current.data) {
46            current.right = addRecursive(current.right, data);
47        } else {
48            return current;
49        }
50        return current;
51    }
```

2. Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

```
168     public int findMinValue() {
169         if (isEmpty()) {
170             System.out.println(x:"Tree is empty!");
171             return Integer.MIN_VALUE;
172         }
173         node11 current = root;
174         while (current.left != null) {
175             current = current.left;
176         }
177         return current.data;
178     }
179
180     public int findMaxValue() {
181         if (isEmpty()) {
182             System.out.println(x:"Tree is empty!");
183             return Integer.MAX_VALUE;
184         }
185         node11 current = root;
186         while (current.right != null) {
187             current = current.right;
188         }
189         return current.data;
190     }
```

3. Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.

```
192     public void printLeafNodes() {
193         displayLeafData(root);
194     }
195 
196     public void displayLeafData(node11 node) {
197         if (node == null) {
198             return;
199         }
200         if (node.left == null && node.right == null) {
201             System.out.print(node.data + " ");
202         }
203         displayLeafData(node.left);
204         displayLeafData(node.right);
205     }
```

4. Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```
207     public int countLeafNodes() {
208         return countLeafNodesRekursif(root);
209     }
210 
211     public int countLeafNodesRekursif(node11 node) {
212         if (node == null) {
213             return 0;
214         }
215         if (node.left == null && node.right == null) {
216             return 1;
217         }
218         return countLeafNodesRekursif(node.left) + countLeafNodesRekursif(node.right);
219 
220     }
221 }
```

MAIN CODE DARI MODIF DI ATAS

```java
public class binaryTreeMain11{
    public static void main(String[] args) {
        binaryTree11 bt = new binaryTree11();
        bt.add(data:6);
        bt.add(data:4);
        bt.add(data:8);
        bt.add(data:3);
        bt.add(data:5);
        bt.add(data:7);
        bt.add(data:9);
        bt.add(data:10);
        bt.add(data:15);
        System.out.print(s:"PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"inOrder Traversal: ");
        bt.traverseInOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"PostOrder Traversal: ");
        bt.traversePostOrder(bt.root);
        System.out.println(x:"");
        System.out.println("Nilai paling kecil dalam tree: " + bt.findMinValue());
        System.out.println("Nilai paling besar dalam tree: " + bt.findMaxValue());
        System.out.println(x:"");
        System.out.print(s:"PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        System.out.println(x:"Data yang ada di leaf:");
        bt.printLeafNodes();
        System.out.println();
        System.out.println("Jumlah leaf dalam tree: " + bt.countLeafNodes());
    }
}
```

HASIL RUNNYA DARI MODIF NO 1-4

```
PreOrder Traversal: 6 4 3 5 8 7 9 10 15
inOrder Traversal: 3 4 5 6 7 8 9 10 15
PostOrder Traversal: 3 5 4 7 15 10 9 8 6
Nilai paling kecil dalam tree: 3
Nilai paling besar dalam tree: 15

PreOrder Traversal: 6 4 3 5 8 7 9 10 15
Data yang ada di leaf:
3 5 7 15
Jumlah leaf dalam tree: 4
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :
method add(int data) untuk memasukan data ke dalam tree

```
 9      public void add(int newData) {
10          if (idxLast + 1 < data.length) {
11              data[++idxLast] = newData;
12          } else {
13              System.out.println(x:"Tree penuh");
14          }
15      }
```

method traversePreOrder() dan traversePostOrder(

```
29      public void traversePreOrder() {
30          traversePreOrder(idxStart:0);
31      }
32      private void traversePreOrder(int idxStart) {
33          if (idxStart <= idxLast) {
34              System.out.print(data[idxStart] + " ");
35              traversePreOrder(2 * idxStart + 1);
36              traversePreOrder(2 * idxStart + 2);
37          }
38      }
39      public void traversePostOrder() {
40          traversePostOrder(idxStart:0);
41      }
42
43      private void traversePostOrder(int idxStart) {
44          if (idxStart <= idxLast) {
45              traversePostOrder(2 * idxStart + 1);
46              traversePostOrder(2 * idxStart + 2);
47              System.out.print(data[idxStart] + " ");
48          }
49      }
50  }
```

ClassMain

```java
public class BinaryTreeArrayMain11 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray11 bta = new BinaryTreeArray11();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.println(x:"\nInOrder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println();
        System.out.println(x:"PreOrder Traversal: ");
        bta.traversePreOrder();
        Syst
        Syst
        bta.traversePostOrder();
    }
}
```

```
void BinaryTreeArray11.traversePostOrder()
```

Hasil Run

```
InOrder Traversal :
3 4 5 6 7 8 9
PreOrder Traversal:
6 4 3 5 8 7 9
PostOrder Traversal:
3 5 4 7 9 8 6
PS C:\Users\ASUS\Documents\SEMESTER 2\Alogaritma&S
```