Modul 10 : ADT AVL Tree

10.1 Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit,** dengan rincian sebagai berikut (misalkan):

a. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
b. 60 menit untuk penyampaian materi
c. 45 menit untuk pengerjaan tugas / Studi Kasus
d. 50 menit **Pengayaan**

10.2 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Mengetahui dan memahami konsep AVL Tree
2. Menerapkan konsep AVL Tree dengan menggunakan bahasa pemrograman
3. Menganalisis penerapan konsep AVL Tree

10.3 Alat & Bahan

1. Komputer
2. Java IDE

10.4 Dasar Teori

AVL Tree adalah salah satu varian dari binary tree tetapi dengan syarat tambahan:

- Data di subtree kiri suatu node selalu lebih kecil dari data di node tersebut dan data di subtree kanan suatu node selalu lebih besar .
- Beda tinggi kiri dan tinggi kanan suatu node tidak lebih dari 1. Jika hal ini terjadi maka dilakukan rotasi kiri atau rotasi kanan

ADT AVL Tree

Dari ilustrasi gambar di atas ADT antrian dapat direpresentasikan sebagai berikut

Node
int data int tinggi Node pKiri Node pKanan Node pInduk
Node(int, int, Node, Node, Node)

List
Node root;
AVLT() boolean cariDt(int dt) boolean sisipDt(int dt) int tinggi() int tinggi(Node node) int jumlahNode() void inOrderTraversal() int jumlahNode(Node node)

10.5 Prosedur Praktikum

	Program Latihan Praktikum 10.1
1	`class Node{`
2	`int data;`
3	`int tinggi; //tinggi node`
4	`Node pKiri;`
5	`Node pKanan;`
6	`Node pInduk;// pointer ke induk`
7	
8	`//constructor node`
9	`public Node(int dt, int tg, Node pKi, Node pKa, Node pI){`
10	`this.data = dt;`
11	`this.tinggi = tg;`
12	`this.pKiri = pKi;`

```
13              this.pKanan = pKa;
14              this.pInduk = pI;
15        }
16  }
17  public class AVLT {
18        private Node root;
19
20        public AVLT(){root = null;}
21
22        //cari dt di tree, mengembalikan true jika ditemukan
23        //dan false jika tidak
24        public boolean cariDt(int dt){
25              Node temp = root;
26
27              while(temp != null){
28                    if(dt == temp.data) return true;
29                    //cariDt subtree pKiri
30                    else if(dt < temp.data)    temp = temp.pKiri;
31                    //cariDt subtree pKanan
32                    else temp = temp.pKanan;
33              }
34              //dt tidak ditemukan
35              return false;
36        }
37        //sisip dt ke dalam tree, returns true if berhasil,
38     // false jika gagal
39        //tree diseimbangkan menggunakan algoritma AVL
40        public boolean sisipDt(int dt){
41              if(root == null){
42                    //sisip dt di root
43                    root = new Node(dt, 1, null, null, null);
44                    return true;
45              }
46              //tree tidak kosong
47              else {
48                    //mulai dari  root
49                    Node temp = root;
50                    Node prev = null;
51
52                    //cari lokasi penyisipan dt
53                    while(temp != null){
54                          if(dt == temp.data) return false;
55                          //sisip dt di subtree pKiri
56                          else if(dt < temp.data){
57                                prev = temp;
58                                temp = temp.pKiri;
59                          }
60                          //sisip dt di subtree pKanan
61                          else {
62                                prev = temp;
63                                temp = temp.pKanan;
64                          }
65                    }
66                    //buat node baru
67                    temp = new Node(dt, 1, null, null, prev);
68
69                    if(dt < prev.data) prev.pKiri = temp;//sisip di
70  pKiri                     else prev.pKanan = temp; //sisipDt at
71  pKanan
72                    //mulai dari node yang disisipkan dan
73                    //bergerak menuju root
74                    while(temp != null){
75                          //subtree pKiri dan pKanan memenuhi
76  kondisi AVL
77                          if(Math.abs(tinggi(temp.pKiri)-
78                       tinggi(temp.pKanan))  <=1){
79
80                                temp.tinggi =
```

```
81  Math.max(tinggi(temp.pKiri),
82
83  tinggi(temp.pKanan)) +1;
84                              }
85                              //kasus 1 algoritma AVL
86                              else if(tinggi(temp.pKiri)-
87                  tinggi(temp.pKanan)>= 2 &&
88                                                  tinggi(temp.pKiri.pKiri)
89  >=
90                  tinggi(temp.pKiri.pKanan))
91                              {
92                                      Node parent = temp.pInduk;
93                                      Node pKiri = temp.pKiri;
94
95                                      temp.pKiri = pKiri.pKanan;
96                                      if(temp.pKiri != null)
97  temp.pKiri.pInduk = temp;
98
99                                      pKiri.pKanan = temp;
100                                     temp.pInduk = pKiri;
101
102                                     pKiri.pInduk = parent;
103                                     if(parent == null) root = pKiri;
104                                     else
105 if(parent.pKiri==temp)parent.pKiri = pKiri;
106                                     else parent.pKanan = pKiri;
107
108                                     //hitung tinggi subtree pKanan
109                                     temp.tinggi =
110 Math.max(tinggi(temp.pKiri),
111
112 tinggi(temp.pKanan)) +1;
113
114                                     temp = pKiri;
115
116                                     //hitung tinggi dari root
117                                     temp.tinggi =
118 Math.max(tinggi(temp.pKiri),
119
120 tinggi(temp.pKanan)) +1;
121                             }
122                             //case 2 algoritma AVl
123                             else if(tinggi(temp.pKanan)-
124                  tinggi(temp.pKiri)>= 2 &&
125
126 tinggi(temp.pKanan.pKanan) >=
127                  tinggi(temp.pKanan.pKiri))
128                             {
129                                     Node parent = temp.pInduk;
130                                     Node pKanan = temp.pKanan;
131
132                                     temp.pKanan = pKanan.pKiri;
133                                     if(temp.pKanan != null)
134       temp.pKanan.pInduk = temp;
135
136                                     pKanan.pKiri = temp;
137                                     temp.pInduk = pKanan;
138
139                                     pKanan.pInduk = parent;
140                                     if(parent == null) root = pKanan;
141                                     else if(parent.pKanan == temp)
142                 parent.pKanan = pKanan;
143                                     else parent.pKiri = pKanan;
144
145                                     //hitung tinggi subtree pKanan
146                                     temp.tinggi =
147 Math.max(tinggi(temp.pKiri),
148
```

```
149 tinggi(temp.pKanan)) +1;
150
151                                 temp = pKanan;
152
153                                 //hitung tinggi dari root
154                                 temp.tinggi =
155 Math.max(tinggi(temp.pKiri),
156
157 tinggi(temp.pKanan)) +1;
158                         }
159                         //kasus 3 dari algoritma AVL
160                         else if(tinggi(temp.pKiri)-
161                 tinggi(temp.pKanan)>= 2 &&
162                                             tinggi(temp.pKiri.pKanan)
163 >=
164                 tinggi(temp.pKiri.pKiri))
165                         {
166                                 Node parent = temp.pInduk;
167                                 Node pKiripKanan =
168 temp.pKiri.pKanan;
169                                 temp.pKiri.pKanan =
170 pKiripKanan.pKiri;
171                                 if(temp.pKiri.pKanan!= null)
172                                         temp.pKiri.pKanan.pInduk =
173 temp.pKiri;
174
175                                 pKiripKanan.pKiri = temp.pKiri;
176                                 temp.pKiri.pInduk = pKiripKanan;
177
178                                 temp.pKiri = pKiripKanan.pKanan;
179                                 if(temp.pKiri != null)
180 temp.pKiri.pInduk = temp;
181
182                                 pKiripKanan.pKanan = temp;
183                                 temp.pInduk = pKiripKanan;
184
185                                 pKiripKanan.pInduk = parent;
186                                 if(parent == null) root =
187 pKiripKanan;
188                                 else if(parent.pKiri==temp)
189                 parent.pKiri = pKiripKanan;
190                                 else parent.pKanan = pKiripKanan;
191
192                                 //hitung tinggi subtree pKanan
193                                 temp.tinggi =
194 Math.max(tinggi(temp.pKiri),
195
196 tinggi(temp.pKanan)) +1;
197
198                                 temp = pKiripKanan;
199
200                                 //hitung tinggi dari root
201                                 temp.tinggi =
202 Math.max(tinggi(temp.pKiri),
203
204 tinggi(temp.pKanan)) +1;
205                         }
206                         //kasus 4 dari algoritma AVL
207                         else if(tinggi(temp.pKanan)-
208                 tinggi(temp.pKiri)>= 2 &&
209                                             tinggi(temp.pKanan.pKiri)
210 >=
211                 tinggi(temp.pKanan.pKanan))
212                         {
213                                 Node parent = temp.pInduk;
214                                 Node pKananpKiri =
215 temp.pKanan.pKiri;
216
```

```
                                    temp.pKanan.pKiri =
pKananpKiri.pKanan;
                                    if(temp.pKanan.pKiri!= null)
                                           temp.pKanan.pKiri.pInduk =
temp.pKanan;

                                    pKananpKiri.pKanan = temp.pKanan;
                                    temp.pKanan.pInduk = pKananpKiri;

                                    temp.pKanan = pKananpKiri.pKiri;
                                    if(temp.pKanan != null)
       temp.pKanan.pInduk = temp;

                                    pKananpKiri.pKiri = temp;
                                    temp.pInduk = pKananpKiri;

                                    pKananpKiri.pInduk = parent;
                                    if(parent == null) root =
pKananpKiri;
                                    else if(parent.pKanan == temp)
                                           parent.pKanan = pKananpKiri;
                                    else parent.pKiri = pKananpKiri;

                                    temp.tinggi =
Math.max(tinggi(temp.pKiri),

tinggi(temp.pKanan)) +1;

                                    temp = pKananpKiri;

                                    temp.tinggi =
Math.max(tinggi(temp.pKiri),

tinggi(temp.pKanan)) +1;
                             }
                             temp = temp.pInduk;
                      }
                      //penyisipan berhasil
                      return true;
               }
       }
       public int tinggi(){return root.tinggi;}
       private int tinggi(Node node){
              if(node == null)return 0;
              else return node.tinggi;
       }
       //hitung node-node dari tree
       public int jumlahNode() {
              return jumlahNode(root);
       }

       public void inOrderTraversal(){
              inOrder(root);
       }

       private void inOrder(Node r){
              if (r == null)return;
              inOrder(r.pKiri);
              System.out.printf("-%d",r.data);
              inOrder(r.pKanan);
       }

       //hitung node-node dari tree
       private int jumlahNode(Node node){
              if(node == null)    return 0;
              else
                     return 1 +jumlahNode(node.pKiri)
                +jumlahNode(node.pKanan);
```

```
283         }
284
285         public static void main (String[] args) {
286         AVLT t = new AVLT();
287         t.sisipDt(3); t.inOrderTraversal();System.out.println();
288         t.sisipDt(4); t.inOrderTraversal();System.out.println();
289         t.sisipDt(6); t.inOrderTraversal();System.out.println();
290         t.sisipDt(5); t.inOrderTraversal();System.out.println();
291         t.sisipDt(15); t.inOrderTraversal();System.out.println();
292         t.sisipDt(10); t.inOrderTraversal();System.out.println();
293         t.sisipDt(20); t.inOrderTraversal();System.out.println();
294         t.sisipDt(17); t.inOrderTraversal();System.out.println();
295         t.sisipDt(25);t.inOrderTraversal();System.out.println();
296         }
297 }
```

10.6 Hasil Percobaan

Tulislah hasil dari Program Latihan Praktikum 10.1

10.7 Analisis Hasil

1. Jelaskan apa berbedaan dari 4 kasus yang ada pada Program Latihan Praktikum 10.1

   ……

   ……

   ……

2. Dalam proses penyisipan data kedalam pohon AVL jika kondisi pohon tidak seimbang maka harus dilakukan rotasi kiri, rotasi kanan, rotasi kiri kanan, atau rotasi kanan dan kiri. Dari program di atas pada baris berapakah dilakukan masing-masing proses ini?

   ……

   ……

10.8 Kesimpulan

10.9 Latihan

1. Tambahkan method putar Kiri pada Program Latihan Praktikum 10.1 untuk menyeimbangkan AVL Tree
2. Tambahkan method putar Kanan pada Program Latihan Praktikum 10.1untuk menyeimbangkan AVL Tree
3. Tambahkan method putar Kiri Kanan pada Program Latihan Praktikum 10.1 untuk menyeimbangkan AVL Tree
4. Tambahkan method putar Kanan Kiri pada Program Latihan Praktikum 10.1 untuk menyeimbangkan AVL Tree

10.10 Tugas

Buatlah Program Latihan Praktikum 10.1 dengan menambahkan method untuk menghapus suatu node pada pohon AVL dengan 3 kondisi yaitu jika node yang dihapus adalah daun, jika node yang dihapus mempunyai satu anak dan jika node yang dihapus mempunyai 2 anak

10.11 DAFTAR PUSTAKA

- Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, “Data Structures andAlgorithms Using Java 6 edition”, Wiley, USA, 2014.
- John R. Hubbard, “Scaum’s Outline of Data Structures With Java second Edition”, McGraw-Hill, New york, 2007.
- Robert Lafore, “Data Structures and Algorithm in Java second Edition”, Sams Publishing, Indiana, 2003