MINGGU 7
Efficient Binary Tree

DESKRIPSI TEMA

(Apa yang akan dipelajari mahasiswa pada minggu ini)

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

1. Mahasiswa mampu menghapus data dari dalam Binary Search Tree
2. Mahasiswa mampu mengimplementasikan AVL Tree

PENUNJANG PRAKTIKUM

1. Aplikasi CodeBlocks
2. Aplikasi Dev-C++ (alternatif)

LANGKAH-LANGKAH PRAKTIKUM

A. Binary Search Tree

Pada pertemuan minggu lalu, kalian sudah belajar untuk membuat, memasukkan, dan menelusuri data dalam Binary Tree. Dalam pertemuan minggu ini, kalian akan belajar mengenai penghapusan data dari dalam Binary Tree.

- Tutorial 1.1 – Deleting Data Inside Binary Tree
  a. Copy Tutorial 1.1 yang terdapat pada Week 6 dan paste ke dalam file dengan nama W07_NIM_DBST.c
  b. Salin potongan code berikut ini ke dalam file tersebut.

```
60  // Function untuk delete node
61  Node* deleteNode(Node* root, int key){
62      // base case
63      if (root == NULL)
64          return root;
65
66      // Kalau key lebih kecil dari root, pindah ke kiri
67      if (key < root->key)
68          root->left = deleteNode(root->left, key);
69
70      // Kalau key lebih besar dari root, pindah ke kanan
71      else if (key > root->key)
72          root->right = deleteNode(root->right, key);
73
74      // Kalau key sama dengan root
75      else {
76          // node dengan 1 anak / tanpa anak
77          if (root->left == NULL) {
78              Node* temp = root->right;
79              free(root);
80              return temp;
81          }
82          else if (root->right == NULL) {
83              Node* temp = root->left;
84              free(root);
85              return temp;
86          }
87
88          // node dengan 2 anak
89          // ambil key terkecil di bawahnya
90          Node* temp = minValueNode(root->right);
91
92          // Copy nilai terkecilnya ke root
93          root->key = temp->key;
94
95          // Delete nilai terkecilnya
96          root->right = deleteNode(root->right, temp->key);
97      }
98      return root;
99  }
```

c. Ubah function main menjadi berikut.

```
101  int main(){
102    Node *root = NULL;
103
104    root = insert(root, 50);
105    insert(root, 50);
106    insert(root, 30);
107    insert(root, 20);
108    insert(root, 40);
109    insert(root, 70);
110    insert(root, 60);
111    insert(root, 80);
112
113    printf("Base Tree\n");
114    printf("Preorder   : "); printPreorder(root); printf("\n");
115    printf("Inorder    : "); printInorder(root); printf("\n");
116    printf("Postorder  : "); printPostorder(root); printf("\n\n");
117
118    printf("Deleting node 20\n");
119    deleteNode(root, 20);
120    printf("Preorder   : "); printPreorder(root); printf("\n");
121    printf("Inorder    : "); printInorder(root); printf("\n");
122    printf("Postorder  : "); printPostorder(root); printf("\n\n");
123
124    printf("Deleting node 20\n");
125    deleteNode(root, 30);
126    printf("Preorder   : "); printPreorder(root); printf("\n");
127    printf("Inorder    : "); printInorder(root); printf("\n");
128    printf("Postorder  : "); printPostorder(root); printf("\n\n");
129
130    printf("Deleting node 20\n");
131    deleteNode(root, 70);
132    printf("Preorder   : "); printPreorder(root); printf("\n");
133    printf("Inorder    : "); printInorder(root); printf("\n");
134    printf("Postorder  : "); printPostorder(root); printf("\n\n");
135
136    return 0;
137  }
```

B. AVL Tree

Merupakan sebuah self-balancing tree. Sama seperti binary tree, memiliki 2 leave per root, hanya saja, pada AVL tree terdapat sebuah Balance Factor. Terdapat 3 nilai pada Balance Factor, yaitu:

a) -1 : Tree lebih berat ke kanan
b) 0 : Tree memiliki tinggi yang sama pada masing-masing daun
c) 1 : Tree lebih berat ke kiri

Tree yang memiliki nilai balance factor di antara -1 hingga 1 merupakan Tree yang seimbang. Pemasukan data ke dalam AVL Tree sama dengan BST, namun terdapat beberapa step tambahan untuk melakukan Balancing.

- Tutorial 2.1 – Balancing Tree
  a. Buat file dengan nama W07_NIM_AVL1.c
  b. Salin potongan code di bawah ini.

```c
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

typedef struct Node{
    // Tambahan height untuk menyimpan tinggi dari node
    int key, height;
    struct Node *left, *right;
} Node;

// Function untuk menghitung tinggi Tree
int height(struct Node *N){
    if (N == NULL)
        return 0;
    return N->height;
}

// Function untuk mencari nilai terbesar
int max(int a, int b){
    return (a > b)? a : b;
}

Node *newNode(int item){
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->key = item;
    temp->left = temp->right = NULL;
    temp->height = 1; // Tambah ini
    return temp;
}
```

```
31  // Function untuk memutar ke kiri nilai subtree dengan nilai x
32  Node *leftRotate(Node *x){
33      Node *y = x->right;
34      Node *T2 = y->left;
35
36      // Perform rotation
37      y->left = x;
38      x->right = T2;
39
40      //  Update heights
41      x->height = max(height(x->left), height(x->right))+1;
42      y->height = max(height(y->left), height(y->right))+1;
43
44      // Return new root
45      return y;
46  }
47
48  // Function untuk memutar ke kanan nilai subtree dengan nilai x
49  Node *rightRotate(Node *y){
50      Node *x = y->left;
51      Node *T2 = x->right;
52
53      // Perform rotation
54      x->right = y;
55      y->left = T2;
56
57      // Update heights
58      y->height = max(height(y->left), height(y->right))+1;
59      x->height = max(height(x->left), height(x->right))+1;
60
61      // Return new root
62      return x;
63  }
64
65  // Mengambil nilai balance suatu node
66  int getBalance(Node *N){
67      if (N == NULL)
68          return 0;
69      return height(N->left) - height(N->right);
70  }
```

```
72  Node *insert(Node *node, int key){
73      /* 1.  Lakukan insertion seperti BST */
74      if (node == NULL) return(newNode(key));
75
76      if (key < node->key)
77          node->left  = insert(node->left, key);
78      else if (key > node->key)
79          node->right = insert(node->right, key);
80      else // Equal keys are not allowed in BST
81          return node;
82
83      /* 2. Update height (tinggi) node sebelumnya */
84      node->height = 1 + max(height(node->left),
85                             height(node->right));
86
87      /* 3. Hitung Balance Factor untuk menentukan
88            apakah Tree ini merupakan Tree yang balance */
89      int balance = getBalance(node);
90
91      // Ada 4 case saat Tree tidak balance
92
93      // Left Left Case
94      if (balance > 1 && key < node->left->key)
95          return rightRotate(node);
96
97      // Right Right Case
98      if (balance < -1 && key > node->right->key)
99          return leftRotate(node);
100
101     // Left Right Case
102     if (balance > 1 && key > node->left->key){
103         node->left =  leftRotate(node->left);
104         return rightRotate(node);
105     }
106
107     // Right Left Case
108     if (balance < -1 && key < node->right->key){
109         node->right = rightRotate(node->right);
110         return leftRotate(node);
111     }
112
113     return node;
114 }
```

```
116  void printInorder(Node *node){
117      if (node == NULL) return;
118      printInorder(node->left);
119      printf("%d ", node->key);
120      printInorder(node->right);
121  }
122
123  void printPreorder(Node* node){
124      if (node == NULL) return;
125      printf("%d ", node->key);
126      printPreorder(node->left);
127      printPreorder(node->right);
128  }
129
130  void printPostorder(Node* node){
131      if (node == NULL) return;
132      printPostorder(node->left);
133      printPostorder(node->right);
134      printf("%d ", node->key);
135  }
136
137  Node* minValueNode(Node* node){
138      Node* current = node;
139
140      /* looping ke node terbawah di kiri */
141      while (current && current->left != NULL)
142          current = current->left;
143
144      return current;
145  }
```

```
147  // Function untuk delete node
148  Node* deleteNode(Node* root, int key){
149      // base case
150      if (root == NULL)
151          return root;
152
153      // Kalau key lebih kecil dari root, pindah ke kiri
154      if (key < root->key)
155          root->left = deleteNode(root->left, key);
156
157      // Kalau key lebih besar dari root, pindah ke kanan
158      else if (key > root->key)
159          root->right = deleteNode(root->right, key);
160
161      // Kalau key sama dengan root
162      else {
163          // node dengan 1 anak / tanpa anak
164          if (root->left == NULL) {
165              Node* temp = root->right;
166              free(root);
167              return temp;
168          }
169          else if (root->right == NULL) {
170              Node* temp = root->left;
171              free(root);
172              return temp;
173          }
174
175          // node dengan 2 anak
176          // ambil key terkecil di bawahnya
177          Node* temp = minValueNode(root->right);
178
179          // Copy nilai terkecilnya ke root
180          root->key = temp->key;
181
182          // Delete nilai terkecilnya
183          root->right = deleteNode(root->right, temp->key);
184      }
185      return root;
186  }
```

```c
int main(){
  Node *root = NULL;

  root = insert(root, 50);
  root = insert(root, 30);
  root = insert(root, 20);
  root = insert(root, 40);
  root = insert(root, 70);
  root = insert(root, 60);
  root = insert(root, 80);

  printf("Base Tree\n");
  printf("Preorder  : "); printPreorder(root); printf("\n");
  printf("Inorder   : "); printInorder(root); printf("\n");
  printf("Postorder : "); printPostorder(root); printf("\n\n");

  printf("Deleting node 20\n");
  deleteNode(root, 20);
  printf("Preorder  : "); printPreorder(root); printf("\n");
  printf("Inorder   : "); printInorder(root); printf("\n");
  printf("Postorder : "); printPostorder(root); printf("\n\n");

  printf("Deleting node 30\n");
  deleteNode(root, 30);
  printf("Preorder  : "); printPreorder(root); printf("\n");
  printf("Inorder   : "); printInorder(root); printf("\n");
  printf("Postorder : "); printPostorder(root); printf("\n\n");

  printf("Deleting node 70\n");
  deleteNode(root, 70);
  printf("Preorder  : "); printPreorder(root); printf("\n");
  printf("Inorder   : "); printInorder(root); printf("\n");
  printf("Postorder : "); printPostorder(root); printf("\n\n");

  return 0;
}
```

REFERENSI