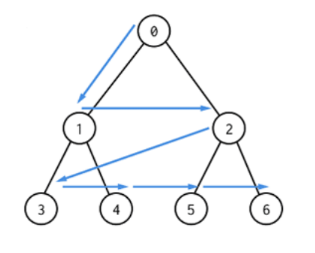
|  |  |
| --- | --- |
| Nama | : La Ode Muhammad Gazali |
| NIM | : 222212696 |
| Kelas | : 2KS2 |

**MODUL 9 PRAKTIUKUM STURKTUR DATA**

Sekarang, Anda telah memahami cara membuat AVL Tree. Untuk memperdalam pemahaman Anda mengenai AVL Tree, modifikasi BST untuk menyimpan nama mahasiswa yang ada pada program Praktikum8B.c menjadi AVL Tree.

1. Simpan ulang Praktikum9B.c dengan nama Praktikum9B.c, lalu lakukan modifikasi pada fungsi insert dan delete seperti yang kita lakukan pada kegiatan praktikum di atas.
2. Kemudian, tambahkan sebuah fungsi untuk menampilkan nama-nama mahasiswa yang ada pada tree dengan alur.

Level Order Trasversal :



* **Program**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct node

{

char data[30];

struct node \*left;

struct node \*right;

int height;

};

int height(struct node\* N)

{

if (N == NULL)

return 0;

return N->height;

}

// Hitung Balance factor untuk node N

int getBalanceFactor(struct node \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

int max(int a, int b)

{

return (a > b)? a : b;

}

struct node \*newNode(const char\* data)

{

struct node \*new\_node = (struct node\*)malloc(sizeof(struct node));

strncpy(new\_node->data, data, sizeof(new\_node->data));

new\_node->left = NULL;

new\_node->right = NULL;

new\_node->height = 1; // new node is initially added at leaf

return(new\_node);

}

struct node\* rightRotate(struct node \*y)

{

struct node \*x = y->left;

struct node \*T2 = x->right;

// Lakukan rotasi

x->right = y;

y->left = T2;

// Update height

y->height = max(height(y->left), height(y->right))+1;

x->height = max(height(x->left), height(x->right))+1;

// Return root baru

return x;

}

struct node \*leftRotate(struct node \*x)

{

struct node \*y = x->right;

struct node \*T2 = y->left;

// Lakukan rotasi

y->left = x;

x->right = T2;

// Update height

x->height = max(height(x->left), height(x->right))+1;

y->height = max(height(y->left), height(y->right))+1;

// Return root baru

return y;

}

struct node\* insert(struct node\* root, const char\* newData)

{

if (root == NULL) {

return(newNode(newData));

}

int compare = strcmp(newData, root->data);

if (compare < 0) {

root->left = insert(root->left, newData);

}

else if (compare > 0) {

root->right = insert(root->right, newData);

}

else

return root;

// 2. Update height dari node

root->height = 1 + max(height(root->left), height(root->right));

// 3. Hitung balance factor untuk menentukan apakah node unbalanced

int balance = getBalanceFactor(root);

// Jika tidak balanced, return hasil rotation

// Kasus 1: Left Left

if (balance > 1 && strcmp(newData, root->left->data) < 0)

return rightRotate(root);

// Kasus 2: Right Right

if (balance < -1 && strcmp(newData, root->right->data) > 0)

return leftRotate(root);

// Kasus 3: Right Left

if (balance < -1 && strcmp(newData,root->right->data) < 0)

{

root->right = rightRotate(root->right);

return leftRotate(root);

}

// Kasus 4: Left Right

if (balance > 1 && strcmp(newData,root->left->data)>0)

{

root->left = leftRotate(root->left);

return rightRotate(root);

}

return root;

}

struct node \* minValueNode(struct node\* node) //cari node minimum di suatu subtree

{

struct node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current->left != NULL)

current = current->left;

return current;

}

void displayPreorder(struct node\* node)

{

if (node == NULL)

return;

printf("%s ", node->data); // root

displayPreorder(node->left); // subtree kiri

displayPreorder(node->right); // subtree kanan

}

void displayInorder(struct node\* node)

{

if (node == NULL)

return;

displayInorder(node->left); // subtree kiri

printf("%s ", node->data); // root

displayInorder(node->right); // subtree kanan

}

void displayPostorder(struct node\* node)

{

if (node == NULL)

return;

displayPostorder(node->left); // subtree kiri

displayPostorder(node->right); // subtree kanan

printf("%s ", node->data); // root

}

void search\_node(struct node\* root, const char\* data)

{

struct node\* cursor = root;

while (cursor != NULL) {

int compare = strcmp(data, cursor->data);

if (compare == 0) {

printf("\nNode %s ditemukan", data);

return;

}

else if (compare < 0) {

cursor = cursor->left;

}

else {

cursor = cursor->right;

}

}

printf("\nNode %s tidak ditemukan", data);

}

void levelOrderTraversal(struct node\* root) {

if (root == NULL)

return;

int maxNodes = 100;

struct node\* queue[100];

int front = 0, rear = 0;

queue[rear++] = root;

while (front < rear) {

struct node\* current = queue[front++];

printf("%s ", current->data);

if (current->left)

queue[rear++] = current->left;

if (current->right)

queue[rear++] = current->right;

}

}

struct node\* delete\_node(struct node\* root, const char\* deletedData)

{

// 1. Lakukan BST delete biasa

int compare = strcmp(deletedData, root->data);

if (root == NULL)

return root;

if (compare < 0)

root->left = delete\_node(root->left, deletedData);

else if(compare > 0)

root->right = delete\_node(root->right, deletedData);

else

{ //jika ditemukan node yang akan dihapus

// 1 CHILD atau NO CHILD

struct node\* cursor;

if (root->left == NULL)

{

cursor = root->right;

free(root);

root = cursor;

}

else if (root->right == NULL)

{

cursor = root->left;

free(root);

root = cursor;

}

//2 CHILDS

else

{

// cari minimum di subtree kanan

cursor = minValueNode(root->right);

strncpy(root->data, cursor->data, sizeof(root->data));

// Delete data yang telah dipindahkan sebagai root

root->right = delete\_node(root->right, cursor->data);

}

}

// Jika setelah dilakukan delete, tree kosong maka return root

if (root == NULL)

return root;

// 2. Update height dari node

root->height = 1 + max(height(root->left), height(root->right));

//3. Hitung balance factor untuk menentukan apakah root unbalanced

int balance = getBalanceFactor(root);

// Jika tidak balanced, return hasil rotation

// Kasus 1: Left Left

if (balance > 1 && getBalanceFactor(root->left) >= 0)

return rightRotate(root);

// Kasus 2: Right Right

if (balance < -1 && getBalanceFactor(root->right) <= 0)

return leftRotate(root);

// Kasus 3: Right Left

if (balance < -1 && getBalanceFactor(root->right) > 0)

{

root->right = rightRotate(root->right);

return leftRotate(root);

}

// Kasus 4: Left Right

if (balance > 1 && getBalanceFactor(root->left) < 0)

{

root->left = leftRotate(root->left);

return rightRotate(root);

}

// return root jika balanced

return root;

}

int main()

{

printf("====Identitas====\n");

printf("Nama : La Ode Muhammad Gazali\n");

printf("NIM : 222212696\n");

printf("Kelas: 2KS2\n\n");

struct node\* root = newNode("Jordan");

root = insert(root, "Dwinanda");

root = insert(root, "Atikah");

root = insert(root, "Gazali");

root = insert(root, "Syawal");

root = insert(root, "Rizky");

root = insert(root, "Zandik");

printf("====Tampilan node awal=====");

printf("\nPreorder : "); displayPreorder(root);

printf("\nInorder : "); displayInorder(root);

printf("\nPostorder : "); displayPostorder(root);

printf("\nLevel Order traversal : ");levelOrderTraversal(root);

printf("\n\n====Pencarian====");

search\_node(root, "Gazali");

search\_node(root, "Ilham");

root = delete\_node(root, "Syawal");

printf("\n\n===Setelah mengahapus Syawal===\n");

printf("Preorder : "); displayPreorder(root);

printf("\nInorder : "); displayInorder(root);

printf("\nPostorder : "); displayPostorder(root);

printf("\nLevel Order traversal : ");levelOrderTraversal(root);

return 0;

}

* **Output**



**AVL Tree sebelum delete Syawal**

Jordan

/ \

Dwinanda Syawal

/ \ / \

Atikah Gazali Rizky Zandik

**AVL Tree setelah delete Syawal**

Jordan

/ \

Dwinanda Zandik

/ \ /

Atikah Gazali Rizky