Nama : Ivan Adrian

NIM : 2602076323

Class : LB01

1. **RED BLACK TREE**

1a. Simulation:

A picture containing text, drawing, handwriting, diagram

Description automatically generated

1b. Implementation in C Pogram (Menggunakan VSCODE)

// RED BLACK TREE

// Nama : Ivan Adrian

// NIM : 2602076323

// Kelas : LB01

#include <stdio.h>

#include<string.h>

#include <stdlib.h>

enum Color{RED, BLACK};

struct Value{

    int value;

    Value\* parent;

    Value\* left;

    Value\* right;

    enum Color redOrBlack;

};

void leftRotation(Value\*\* root, Value\* current){

    Value \*childs = current->right;

    current->right = childs->left;

    if(childs->left != NULL){

        childs->left->parent = current;

    }

    childs->parent = current->parent;

    if(current->parent == NULL){

        \*root = childs;

    }

    else if(current == current->parent->left){

        current->parent->left = childs;

    }

    else{

        current->parent->right = childs;

    }

    childs->left = current;

    current->parent = childs;

}

void rightRotation(Value\*\* root, Value\*current){

    Value \*childs = current->left;

    current->left = childs->right;

    if(childs->right != NULL){

        childs->right->parent = current;

    }

    childs->parent = current->parent;

    if(current->parent == NULL){

        \*root = childs;

    }

    else if(current == current->parent->right){

        current->parent->right = childs;

    }

    else{

        current->parent->left = childs;

    }

    childs->right = current;

    current->parent = childs;

}

void violateRepairation(Value\*\* root, Value\* current){

    while(current->parent != NULL && current->parent->redOrBlack == RED){

        if(current->parent == current->parent->parent->left){

            Value\* uncle = current->parent->parent->right;

            if(uncle != NULL && uncle->redOrBlack == RED){

                current->parent->redOrBlack = BLACK;

                uncle->redOrBlack = BLACK;

                current->parent->parent->redOrBlack = RED;

                current = current->parent->parent;

            }

            else{

                if(current == current->parent->right){

                    current = current->parent;

                    leftRotation(root, current);

                }

                current->parent->redOrBlack = BLACK;

                current->parent->parent->redOrBlack = RED;

                rightRotation(root, current->parent->parent);

            }

        }

        else{

            Value\* uncle = current->parent->parent->left;

            if(uncle != NULL && uncle->redOrBlack == RED){

                current->parent->redOrBlack = BLACK;

                uncle->redOrBlack = BLACK;

                current->parent->parent->redOrBlack = RED;

                current = current->parent->parent;

            }

            else{

                if(current == current->parent->left){

                    current = current->parent;

                    rightRotation(root, current);

                }

                current->parent->redOrBlack = BLACK;

                current->parent->parent->redOrBlack = RED;

                leftRotation(root, current->parent->parent);

            }

        }

    }

    (\*root)->redOrBlack = BLACK;

}

void insertValue(Value\*\* root, int value){

    Value \*newItem = (Value\*)malloc(sizeof(Value));

    newItem->value = value;

    newItem->parent = NULL;

    newItem->left = NULL;

    newItem->right = NULL;

    newItem->redOrBlack = RED;

    Value\* current = \*root;

    Value\* parent = NULL;

    while(current != NULL){

        parent = current;

        if(newItem->value < current->value){

            current = current->left;

        }

        else{

            current = current->right;

        }

    }

    newItem->parent = parent;

    if(parent == NULL){

        \*root = newItem;

    }

    else if(newItem->value < parent->value){

        parent->left = newItem;

    }

    else{

        parent->right = newItem;

    }

    violateRepairation(root, newItem);

}

void printInorder(Value \*root){

    if (root == NULL){

        return;

    }

    else if(root != NULL){

        printInorder(root->left);

        printf("%d ", root->value);

        printInorder(root->right);

    }

}

int main() {

    Value \*root = NULL;

    int value[] = {41, 22, 5, 51, 48, 29, 18, 21, 45, 3};

    int nValue = sizeof(value) / sizeof(value[0]);

    int i = 0;

    while(i < nValue){

        insertValue(&root, value[i]);

        i++;

    }

    puts("Inorder Traversal of Created Tree");

    printInorder(root);

    return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Explanation of Code Red Black Tree:**

A picture containing text, screenshot, software, number

Description automatically generated

A screenshot of a computer code

Description automatically generated with medium confidence

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer code

Description automatically generated with medium confidence

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer code

Description automatically generated with low confidence

A screenshot of a computer code

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with low confidence

A screenshot of a computer code

Description automatically generated with low confidence

A screenshot of a computer code

Description automatically generated with low confidence

A screenshot of a computer code

Description automatically generated with low confidence

A picture containing text, font, screenshot

Description automatically generated

A screenshot of a computer code

Description automatically generated with low confidence

1. **AVL TREE**

2a.simulation

A picture containing text, diagram, sketch, drawing

Description automatically generated

A picture containing text, drawing, sketch, black and white

Description automatically generated

**A picture containing sketch, drawing

Description automatically generated**

2b.Implementation in C program

// AVL TREE

// Nama : Ivan Adrian

// NIM : 2602076323

// Kelas : LB01

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<time.h>

#include<windows.h>

struct Value{

    int value;

    int height;

    int balanceFactor;

    Value\* left;

    Value\* right;

};

Value\* createNode(int value){

    Value\* newValue = (Value\*) malloc(sizeof(Value));

    newValue->value = value;

    newValue->right = newValue->left = NULL;

    newValue->height = 1;

    return newValue;

}

int max(int getHeightLeft, int getHeightRight){

    if(getHeightLeft > getHeightRight){

        return getHeightLeft;

    }

    return getHeightRight;

}

int getHeight(Value\* current){

    if(current == NULL){

        return 0;

    }

    return current->height;

}

int setBalanceFactor(Value\* current){

    if(current == NULL){

        return 0;

    }

    return getHeight(current->left) - getHeight(current->right);

}

int setHeight(Value\* current){

    if(current == NULL){

        return 0;

    }

    return max(getHeight(current->left), getHeight(current->right)) + 1;

}

Value\* updateNode(Value\* current){

    if(current == NULL){

        return current;

    }

    current->height = setHeight(current);

    current->balanceFactor = setBalanceFactor(current);

    return current;

}

Value\* leftRotation(Value\* current){

    Value \*pivot = current->right;

    Value \*childLeft = pivot->left;

    pivot->left = current;

    current->right = childLeft;

    current = updateNode(current);

    pivot = updateNode(pivot);

    return pivot;

}

Value\* rightRotation(Value\* current){

    Value \*pivot = current->left;

    Value \*childRight = pivot->right;

    pivot->right = current;

    current->left = childRight;

    current = updateNode(current);

    pivot = updateNode(pivot);

    return pivot;

}

Value\* rotation(Value\* current){

    if(current == NULL){

        return current;

    }

    // LL (Left-Left Case)

    if(current->balanceFactor > 1 && current->left->balanceFactor >= 0){

        return rightRotation(current);

    }

    // RR (Right-Right Case)

    else if(current->balanceFactor < -1 && current->right->balanceFactor <= 0){

        return leftRotation(current);

    }

    // LR (Left-Right Case)

    else if(current->balanceFactor > 1 && current->left->balanceFactor < 0){

        current->left = leftRotation(current->left);

        return rightRotation(current);

    }

    // RL (Right-Left Case)

    else if(current->balanceFactor < -1 && current->right->balanceFactor > 0){

        current->right = rightRotation(current->right);

        return leftRotation(current);

    }

    return current;

}

Value\* insertValue(Value\* root, int value){

    if(root == NULL){

        return createNode(value);

    }

    else if(value < root->value){

        root->left = insertValue(root->left, value);

    }

    else if(value > root->value){

        root->right = insertValue(root->right, value);

    }

    return rotation(updateNode(root));

}

Value\* searchValue(Value\* root, int value){

    if(root == NULL || root->value == value){

        return root;

    }

    else if(value < root->value){

        return searchValue(root->left, value);

    }

    else{

        return searchValue(root->right, value);

    }

}

Value\* deleteValue(Value\* root, int deletedValue){

    if(root == NULL){

        return root;

    }

    else if(deletedValue < root->value){

        root->left = deleteValue(root->left, deletedValue);

    }

    else if(deletedValue > root->value){

        root->right = deleteValue(root->right, deletedValue);

    }

    else if(deletedValue == root->value) {

        if(root->left == NULL && root->right == NULL){

            // Node to delete has no children

            free(root);

            root = NULL;

        }

        else if(root->left != NULL && root->right == NULL){

            // Node to delete has only left child

            Value \*temp = root->left;

            free(root);

            root = temp;

        }

        else if(root->left == NULL && root->right != NULL){

            // Node to delete has only right child

            Value \*temp = root->right;

            free(root);

            root = temp;

        }

        else if(root->left != NULL && root->right != NULL){

            // Node to delete has both left and right children

            Value \*temp = root->left;

            while (temp->right) {

                temp = temp->right;

            }

            root->value = temp->value;

            root->left = deleteValue(root->left, temp->value);

        }

    }

    return rotation(updateNode(root));

}

void preOrder(Value\* root){

    if(root){

        printf("%d ", root->value);

        preOrder(root->left);

        preOrder(root->right);

    }

}

void inOrder(Value\* root){

    if(root){

        inOrder(root->left);

        printf("%d ", root->value);

        inOrder(root->right);

    }

}

void postOrder(Value\* root){

    if(root){

        postOrder(root->left);

        postOrder(root->right);

        printf("%d ", root->value);

    }

}

void printTraversal(Value\* root){

    printf("Preorder: ");

    preOrder(root);

    puts("");

    printf("Inorder: ");

    inOrder(root);

    puts("");

    printf("Postorder: ");

    postOrder(root);

    puts("");

}

void exitScreen(){

    system("cls");

    puts("Thank You");

    printf("Press Enter to exit");

    getchar();

    exit(0);

}

void menuList(){

    int inputNumber;

    Value \*root = NULL;

    do{

        system("cls");

        puts("1. Insertion");

        puts("2. Deletion");

        puts("3. Traversal");

        puts("4. Exit");

        printf("Choose: ");

        scanf("%d", &inputNumber);

        getchar();

        int valueToInput;

        int valueToDelete;

        switch(inputNumber){

            case 1:

                printf("Insert: ");

                scanf("%d", &valueToInput);

                getchar();

                root = insertValue(root, valueToInput);

                printf("Value %d was inserted\n", valueToInput);

                break;

            case 2:

                printf("Delete: ");

                scanf("%d", &valueToDelete);

                getchar();

                if(searchValue(root, valueToDelete) != NULL){

                    root = deleteValue(root, valueToDelete);

                    puts("Data Found");

                    printf("Value %d was deleted\n", valueToDelete);

                }

                else if(searchValue(root, valueToDelete) == NULL){

                    puts("Data not found");

                }

                root = deleteValue(root, valueToDelete);

                break;

            case 3:

                printTraversal(root);

                break;

            case 4:

                exitScreen();

                break;

        }

        puts("Press enter to continue...");

        getchar();

    }while(inputNumber >= 1 && inputNumber <= 4);

}

int main(){

    menuList();

    return 0;

}

**i.Main Menu**

**A screen shot of a computer

Description automatically generated with medium confidence**

**ii.Insertion**

**A screen shot of a computer

Description automatically generated with medium confidence**

Insert 6

**A screen shot of a computer

Description automatically generated with low confidence**

Insert 27

**A screen shot of a computer

Description automatically generated with medium confidence**

Insert 19

**A screen shot of a computer

Description automatically generated with low confidence**

Insert 11

**A black screen with white text

Description automatically generated with low confidence**

Insert 36

**A black screen with white text

Description automatically generated with low confidence**

Insert 14

**A screen shot of a computer

Description automatically generated with medium confidence**

Insert 81

**A screen shot of a computer

Description automatically generated with low confidence**

Insert 63

**A black screen with white text

Description automatically generated with low confidence**

Insert 75

**iii.Deletion**

**A screen shot of a computer

Description automatically generated with medium confidence**

Delete 14

**A screen shot of a computer

Description automatically generated with medium confidence**

Delete 75

**A screen shot of a computer

Description automatically generated with medium confidence**

Delete 36

**A screen shot of a computer

Description automatically generated with medium confidence**

Delete 19

**A screen shot of a computer

Description automatically generated with medium confidence**

Delete 11

**A screen shot of a computer

Description automatically generated with medium confidence**

Data Not Found

**iv.Traversal**

**A screenshot of a computer screen

Description automatically generated with low confidence**

After Insertion

**A screenshot of a computer

Description automatically generated with medium confidence**

After Deletion

**v.Exit**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Explanation of Code AVL TREE:**

**A screen shot of a computer

Description automatically generated with medium confidence**

**A picture containing text, font, number, screenshot

Description automatically generated**

**A screen shot of a computer code

Description automatically generated with low confidence**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A picture containing text, font, screenshot, line

Description automatically generated**

**A picture containing text, font, screenshot, line

Description automatically generated**

**A picture containing text, font, screenshot

Description automatically generated**

**A picture containing text, screenshot, font, number

Description automatically generated**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A screenshot of a computer program

Description automatically generated with low confidence**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A screenshot of a computer code

Description automatically generated with medium confidence**

**A screenshot of a computer code

Description automatically generated with medium confidence**

**A screen shot of a computer code

Description automatically generated with low confidence**

**A picture containing text, screenshot, font

Description automatically generated**

**A picture containing text, screenshot, font, line

Description automatically generated**

**A picture containing text, screenshot, line

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A picture containing text, screenshot, font, line

Description automatically generated**

**A screenshot of a computer program

Description automatically generated with low confidence**

**A screenshot of a computer program

Description automatically generated with low confidence**

**A picture containing text, screenshot, number

Description automatically generated**