INFORMATION
TECHNOLOGY
UNIVERSITY

Software Engineering Department - ITU
SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 27/11/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 13. Red Black Trees

Name	Roll number	Report (out of 35)

Checked on: ______________________________

Signature: ___________________________________

1.1 Objective

The objective of this lab is to practice problems related to the implementation of Red Black Trees.

1.2 Equipment and Component

Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

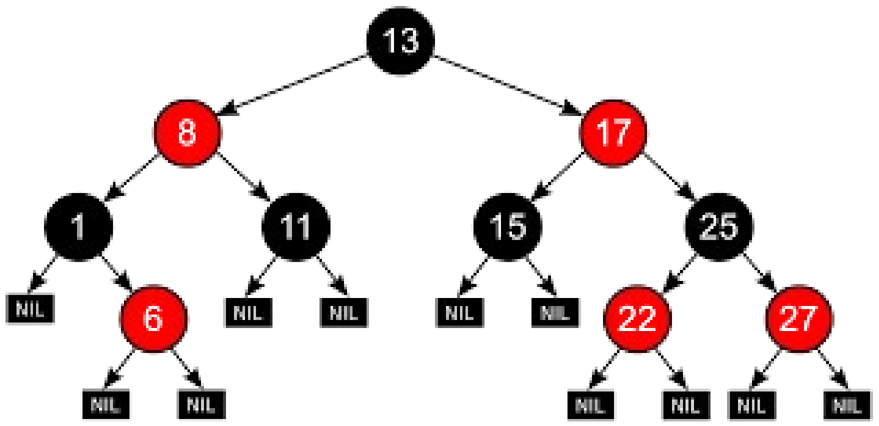
1.4 Theory and Background

A **Red-Black Tree** is a self-balancing binary search tree used in computer science to maintain sorted data while ensuring logarithmic time complexity for common operations like insertion, deletion, and search. The tree is named for the two types of nodes it contains: **Red** and **Black**, which enforce balance through a set of properties.

Properties of Red-Black Trees

To ensure balance and maintain logarithmic depth, a Red-Black Tree adheres to the following properties:

1. **Node Color**: Each node is either **Red** or **Black**.
2. **Root is Black**: The root of the tree is always black.
3. **Red Node Rule**: Red nodes cannot have red children (no two consecutive red nodes).
4. **Black Height Rule**: Every path from a node to its descendant NULL pointers (leaf nodes) has the same number of black nodes.
5. **Leaf Nodes**: All leaf nodes (NULL pointers) are considered black.



Lab Tasks

Task 1:

You will implement a **Red-Black Tree** using the following class structure. Each node in the tree will have a color (Red or Black) and pointers to its parent, left, and right children.

Class: Node

Data Members

1. **int key**: The value stored in the node.
2. **bool color**: Represents the color of the node (true for red, false for black).
3. **Node* left**: Pointer to the left child node.
4. **Node* right**: Pointer to the right child node.
5. **Node* parent**: Pointer to the parent node.

Member Functions

1. **Constructor:**
   - **Node(int key, bool color)**: Initializes the node with the given key and color, and sets child and parent pointers to nullptr.
2. **Getters and Setters:**
   - **int getKey():** Returns the key of the node.
   - **bool getColor()**: Returns the color of the node.
   - **void setColor(bool color):** Sets the color of the node.
   - **Node* getLeft():** Returns the pointer to the left child.
   - **Node* getRight()**: Returns the pointer to the right child.
   - **Node* getParent()**: Returns the pointer to the parent.
   - **void setLeft(Node* left**Child): Sets the left child pointer.
   - **void setRight(Node* rightChild)**: Sets the right child pointer.
   - **void setParent(Node* parentNode)**: Sets the parent pointer.

Class: RedBlackTree

Data Members

1. **Node* root**: Pointer to the root of the Red-Black Tree.

Member Functions

1. **Constructor:**
   - **RedBlackTree():** Initializes the tree with root as nullptr.
2. **Insertion:**

   - **void insert(int key)**: Inserts a new key into the tree while maintaining Red-Black Tree properties.
   - **void balanceInsert(Node* node)**: Helper function to fix violations after insertion.
3. **Search:**
   - **Node* search(int key)**: Searches for a node with the given key and returns a pointer to it.
4. **Rotation:**
   - **void leftRotate(Node* node):** Performs a left rotation around the given node.
   - **void rightRotate(Node* node):** Performs a right rotation around the given node.
5. **Utility Functions:**
   - **void printTree()**: Prints the Red-Black Tree in an in-order traversal.

Lab Instructions

1. **Implement the Node Class**:
   - Begin by implementing the Node class and its associated member functions.
2. **Set Up the Tree**:
   - Implement the RedBlackTree class and initialize the root as nullptr.
3. **Insertions**:
   - Add the insert function to handle the insertion of a new key.
   - Use the balanceInsert function to maintain Red-Black properties after an insertion.
4. **Tree Rotations**:
   - Implement the leftRotate and rightRotate functions to handle tree re-balancing.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

Assessment Rubric for Lab

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					