EMERTXE TRAINING PROJECT DOCUMENTATION FRAMEWORK

REQUIREMENTS & DESIGN DOCUMENT

Module – Data Structures

Red Black Tree

Contents

1 Abstract 1
2 Requirements 2
3 Prerequisites 3
4 Design 4
5 Sample Output 6
6 Artifacts 8
6.1 Skeleton Code 8
6.2 References 8

1 Abstract

A red–black tree is a kind of self-balancing binary search tree. Each node of the binary tree has an extra bit, and that bit is often interpreted as the color (red or black) of the node. These color bits are used to ensure the tree remains approximately balanced during insertions and deletions.

Balance is preserved by painting each node of the tree with one of two colors (typically called ‘red’ or ‘black’) in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

2 Requirements

In addition to the requirements imposed on a binary search tree the following must be satisfied by a red–black tree:

1. Each node is either red or black.
2. The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa, this rule has little effect on analysis.
3. All leaves (NIL) are black.
4. If a node is red, then both its children are black.
5. Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes. Some definitions: the number of black nodes from the root to a node is the node's black depth; the uniform number of black nodes in all paths from root to the leaves is called the black-height of the red–black tree.

Operations to be Implemented:

1. Insertion
2. Deletion
3. Search
4. Find Min
5. Delete Min
6. Find Max
7. Delete Max

•

3 Prerequisites

- Pointers, Structures and Dynamic Memory Handling
- Trees

4 Design

Function	Insert
Prototype	int insert(tree_t **root, data_t item);
Input Parameters	• root – Pointer to the root node of the Red Black tree. • item – New data to be inserted into the Red Black tree.
Return Values	Status (SUCCESS / FAILURE)

Function	Delete
Prototype	int delete(tree_t **root, data_t item);
Input Parameters	• root – Pointer to the root node of the Red Black tree. • item – Data to be deleted from the Red Black tree.
Return Values	Status (SUCCESS / FAILURE)

Function	Find Minimum
Prototype	int find_minimum(tree_t **root, data_t *min);
Input Parameters	• root – Pointer to the root node of the Red Black tree. • min – Minimum data present in the tree is collected via this pointer.
Return Values	Status (SUCCESS / FAILURE)

Function	Find Maximum
Prototype	int find_maximum(tree_t **root, data_t *max);
Input Parameters	• root – Pointer to the root node of the Red Black tree. • min – Maximum data present in the tree is collected via this pointer.
Return Values	Status (SUCCESS / FAILURE)

Function	Delete Minimum
Prototype	int delete_minimum(tree_t **root);
Input Parameters	• root – Pointer to the root node of the Red Black tree.
Return Values	Status (SUCCESS / FAILURE)

Function	Delete Maximum
Prototype	int delete_maximum(tree_t **root);
Input Parameters	• root – Pointer to the root node of the Red Black tree.
Return Values	Status (SUCCESS / FAILURE)

5 Sample Output

```
user@emertxe] gcc red_black_tree.c -o red_black_tree.exe
./red_black_tree.exe
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 1
Enter the element : 18

Do u want to contiune(y/n) : y

Enter the element : 12

Before Balancing Red Black Tree is :
 (12)--(RED->0)  (18)--(BLACK->1)
Now Tree is balance
 (12)--(RED->0)  (18)--(BLACK->1)
Do u want to contiune(y/n) : y

Enter the element : 89

Before Balancing Red Black Tree is :
 (12)--(RED->0)  (18)--(BLACK->1)  (89)--(RED->0)
Now Tree is balance
 (12)--(RED->0)  (18)--(BLACK->1)  (89)--(RED->0)
Do u want to contiune(y/n) : y

Enter the element : 14

Before Balancing Red Black Tree is :
 (12)--(RED->0)  (14)--(RED->0)  (18)--(BLACK->1)  (89)--(RED->0)
Now Tree is balance
 (12)--(BLACK->1)  (14)--(RED->0)  (18)--(BLACK->1)  (89)--(BLACK->1)
Do u want to contiune(y/n) : y

Enter the element : 28

Before Balancing Red Black Tree is :
 (12)--(BLACK->1)  (14)--(RED->0)  (18)--(BLACK->1)  (28)--(RED->0)  (89)--(BLACK->1)
Now Tree is balance
 (12)--(BLACK->1)  (14)--(RED->0)  (18)--(BLACK->1)  (28)--(RED->0)  (89)--(BLACK->1)
Do u want to contiune(y/n) : n
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 2
Display (12)--(BLACK->1)  (14)--(RED->0)  (18)--(BLACK->1)  (28)--(RED->0)  (89)--(BLACK->1)
Enter the choice : 3
Enter the element which u want to search from the red black tree : 14
Node is found  (14)--(RED->0)
```

Fig 5 1: Tree Creation, Display and Search Node

```
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit

Enter the choice : 4
Maximum node in the given Red Black Tree is : (89)--(BLACK->1)
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit

Enter the choice : 5
Minimum node in the given Red Black Tree is : (12)--(BLACK->1)
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 6
Enter the node which u want to delete: 18
successor data : 28
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 2
Display (12)--(BLACK->1)  (14)--(RED->0)  (28)--(BLACK->1)  (89)--(BLACK->1)
```

Fig 5 2: Finding Minimum, Maximum and Deletion of Node

```
1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 7
Tree Before deleting the minimum node:
(12)--(BLACK->1)  (14)--(RED->0)  (28)--(BLACK->1)  (89)--(BLACK->1)
Tree After deleting the minimum node:
(14)--(BLACK->1)  (28)--(BLACK->1)  (89)--(BLACK->1)

1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 8
Tree Before deleting the maximum node:
(14)--(BLACK->1)  (28)--(BLACK->1)  (89)--(BLACK->1)
Tree After deleting the maximum node
(14)--(BLACK->1)  (28)--(BLACK->1)

1.Create a tree
2.Display
3.Search a node
4.Find Maximum node in Tree
5.Find Minimum node in Tree
6.Deletion
7.Delete Minimum node
8.Delete Maximum Node
9.Exit
Enter the choice : 9
user@emertxe]
```

Fig 5 3: Delete Maximum Node

6 Artifacts

6.1 Skeleton Code

- www.emertxe.com/content/data-structures/code/redblacktree_src.zip

6.2 References

- https://en.wikipedia.org/wiki/Red%E2%80%93black_tree