COP5536: Advanced Data Structures, Spring 2016

Project Report

Vikaasa Ramdas Thandu Venkat Kumar
UFID: 4400-5810
vikaasa@ufl.edu

Compiling Instructions:

The project has been written in java and can be compiled using standard JDK 1.8.0_60, with javac compiler.

Unzip the submitted folder. The folder contains the project files (.java files), the Makefile, the commands.txt file, and my output files (.txt). The input text files must be placed in the project directory before running the program.

To compile in the Unix environment, run the following command after changing the directory to the project directory:

```
$ make
```

To run the program such that it supports redirected input from a text-file "filename" that contains the initial sorted list, use the following command:

```
$ java bbst filename
```

To accept the commands from a text file and print to a text file, use the following command:

```
 $ java bbst filename < commands.txt > myoutput.txt
```

Note: In order to run the program for the test file "test_100000000", the heapspace will have to be set to 10 GB using the following command:

```
$ java -Xmx10000m bbst test_100000000 < commands.txt
```

Program Structure:

The project has three classes, "Node.java", "RedBlackTree.java", and "bbst.java", where "bbst.java" is the Main function.

`Node.java`

This class describes the structure of a node used in the Red Black Tree. It contains the following class variables:

`private int ID` – stores the ID value of the event.

`private int count` – stores the count of the number of active events with the given ID.

`private Node left` – links to the left child of the node.

`private Node right` – links to the right child of the node.

`private Node parent` – links to the parent of the node.

`private char color` – is used to identify the color of the node as 'r' or 'b'.

The class also contains a constructor function and getter and setter functions for ID, count, color, left, right, and parent.

`RedBlackTree.java`

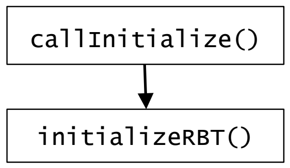
The class RedBlackTree is where all the operations of a Red Black Tree have been defined. The operations of this class are performed on instances of the class Node.

The functions defined in the class are given below:

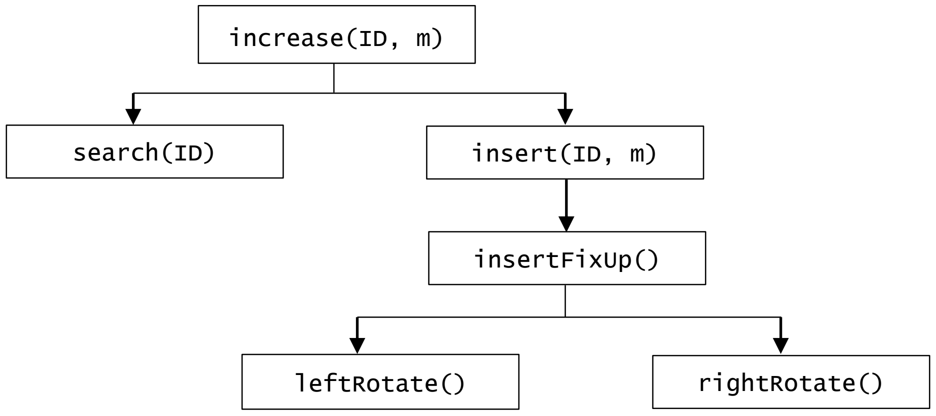
```
public void insert(int ID, int count)
public void insertFixUp(Node z)
public void leftRotate(Node x)
public void rightRotate(Node x)
public void transplant(Node u, Node v)
public void delete(Node z)
public void deleteFixUp(Node x)
public Node search(int x)
public void remove(int x)
public void increase(int ID, int m)
public void reduce(int ID, int m)
public void count(int ID)
public Node[] findNextPrev(Node root, Node[] res, int key)
public void next(int ID)
public void previous (int ID)
public int inRange(Node root, int ID1, int ID2)
public void callInRange(int ID1, int ID2)
public Node initializeRBT(int arr[][],int start,int end,int lvl,int h)
public void callInitialize(int arr[][],int start,int end,int n)
```

The various operations carried out by this class are diagrammatically represented below:

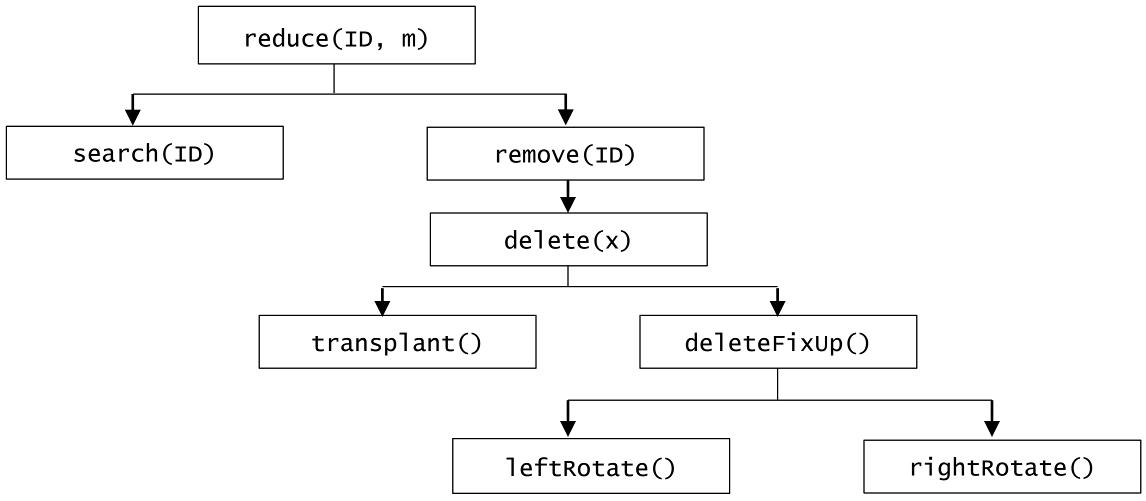
1. Initialize Red Black Tree:



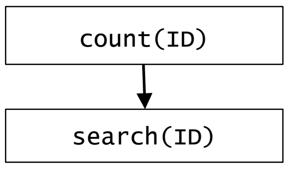
2. Increase:



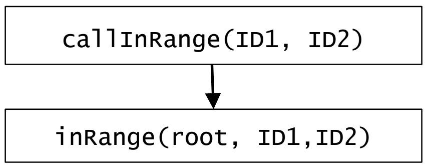
3. Reduce:



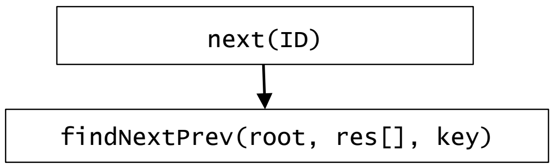
4. Count:



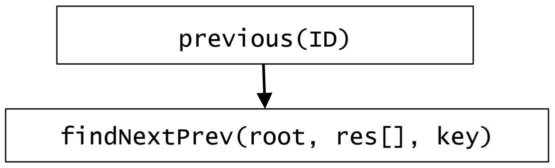
5. In Range:



6. Next:



7. Previous:



`bbst.java`

This class contains the main function that reads the input text file, initializes the input sorted array as a Red Black Tree, and then reads and executes user input commands by calling the relevant function.

Function prototypes:

`Node.java`

The function prototypes for the Node class are given below:

`public Node()`

This is a constructor for the Node class. It creates a sentinel node by setting parent, right child, left child to null, and coloring it as 'b'.

`public Node(int ID, int count)`

Parameters: int ID, int count
Description: This is a parameterized constructor for the Node class.

`public char getColor()`

Return type: char
Description: This function returns the color of the node.

`public void setColor(char color)`

Parameters: char color
Return type: void
Description: This function sets the color of the node to 'color'.

`public int getID()`

Return type: int
Description: This function returns the ID of the node.

`public void setID(int ID)`

Parameters: int ID
Return type: void
Description: This function sets the ID of the node to 'ID'.

`public int getCount()`

Return type: int
Description: This function returns the count of the node.

`public void setCount(int count)`

Parameters: int count
Return type: void
Description: This function sets the count of the node to 'count'.

`public Node getParent()`

Return type: Node
Description: This function returns the current node's parent.

`public void setParent(Node parent)`

Parameters: Node parent

Return type: void
Description: This function sets the current node's parent to 'parent'.

`public Node getLeft()`

Return type: Node
Description: This function returns the current node's left child.

`public void setLeft(Node left)`

Parameters: Node left
Return type: void
Description: This function sets the current node's left child to 'left'.

`public Node getRight()`

Return type: Node
Description: This function returns the current node's right child.

`public void setRight(Node right)`

Parameters: Node right
Return type: void
Description: This function sets the current node's right child to 'right'.

`RedBlackTree.java`

The function prototypes for the RedBlackTree class are given below:

`public void insert(int ID, int count)`

Parameters: int ID, int count
Return type: void
Description: A function to insert an event with two integer values, ID and count, in a Red Black Tree. It has a time complexity of O(log n).

`public void insertFixUp(Node z)`

Parameters: Node z
Return type: void
Description: This function fixes the possible violation of the red-black properties of the Red Black Tree when the insert() function is invoked. The possible violation is pushed up toward the root, until it reaches the root, which can just be made black.

`public void leftRotate(Node x)`

Parameters: Node x
Return type: void
Description: This function performs a left rotation around the node 'x'.

`public void rightRotate(Node x)`

Parameters: Node x
Return type: void
Description: This function performs a right rotation around the node 'x'.

`public void transplant(Node u, Node v)`

Parameters: Node u, Node v
Return type: void

Description: This function replaces the subtree rooted at node u with the subtree rooted at node v.

```
public void delete(Node z)
```

Parameters: Node z
Return type: void
Description: A function to delete the node 'z' from a red-black tree. It has a time complexity of O(log n).

```
public void deleteFixUp(Node x)
```

Parameters: Node x
Return type: void
Description: This function fixes the possible violation of the red-black properties caused by deleting a node using the delete() function.

```
public Node search(int x)
```

Parameters: int x
Return type: Node
Description: This function searches the Red Black Tree for a node with ID = 'x', and returns the node if found. The time complexity of this function is O(log n).

```
public void remove(int x)
```

Parameters: int x
Return type: void
Description: This function calls the search() function to find a node with ID = 'x'. If found, it then calls the delete() function passing the node that was returned by the search() function as an argument. The time complexity of this function is O(log n).

```
public void increase(int ID, int m)
```

Parameters: int ID, int m
Return type: void
Description: This function calls the search() function to find a node with ID = 'ID'. If found, it then increases the count of that node by 'm', and prints the count after increasing. Else, it calls the insert() function passing 'ID' and 'm' as arguments. The time complexity of this function in O(log n).

```
public void reduce(int ID, int m)
```

Parameters: int ID, int m
Return type: void
Description: This function calls the search() function to find a node with ID = 'ID'. If found, it then decreases the count of that node by 'm', and prints the count after decreasing. However, if the count <= 0 after decreasing, the delete function() is called passing the node as an argument. If the node is not present or is removed, then '0' is printed. The time complexity of this function in O(log n).

```
public void count(int ID)
```

Parameters: int ID
Return type: void

Description: This function calls the search() function to find a node with ID = ‘ID’. If found, the function then prints the count of the found node. Else, it prints ‘0’. This function has a time complexity of O(log n).

`public Node[] findNextPrev(Node root, Node[] res, int key)`

Parameters: Node root, Node[] res, int key
Return type: Node[]
Description: This function is a recursive function that returns an array of two nodes which identify the next and previous nodes with value of ‘ID’ around the value of ‘key’, even when node with ID = key is not present in the tree. The time complexity of this function is O(log n).

`public void next (int ID)`

Parameters: int ID
Return type: void
Description: This function calls the findNextPrev() function, passing ‘root’, ‘res’, and ‘ID’ as parameters, and prints the returned next node’s ID and count, if a next node was found. If no next node was found, then “0 0” is printed. The time complexity of this function is O(log n).

`public void previous (int ID)`

Parameters: int ID
Return type: void
Description: This function calls the findNextPrev() function, passing ‘root’, ‘res’, and ‘ID’ as parameters, and prints the returned previous node’s ID and count, if a previous node was found. If no previous node was found, then “0 0” is printed. The time complexity of this function is O(log n).

`public int inRange(Node root, int ID1, int ID2)`

Parameters: Node root, int ID1, int ID2
Return type: int
Description: This function is a recursive function that returns the sum of the counts of the nodes with IDs within the range of ID1 and ID2, even when nodes with ID1 and ID2 are not present in the tree. This function has a time complexity of O(log n + s) where ‘s’ is the number of IDs in the range.

`public void callInRange(int ID1, int ID2)`

Parameters: int ID1, int ID2
Return type: void
Description: This function calls the inRange() function and prints the returned value, which is the sum of the counts of all the nodes with IDs within the range of ID1 and ID2. This function has a time complexity of O(log n + s) where ‘s’ is the number of IDs in the range.

`public Node initializeRBT(int arr[][],int start,int end,int lvl,int h)`

Parameters: int arr[][], int start, int end, int lvl, int h
Return type: Node
Description: This function initializes the Red Black Tree from an array of events containing ID and count and sorted by ID. This is done by making the middle element of the sorted array as the root of the Red Black Tree, and then recursively doing it for the

left half and the right half. The root of the entire Red Black Tree is then returned. All nodes other than the immediate parents of external nodes are colored black, while the immediate parents of external nodes are colored red. The time complexity of this function is O(n).

```
public void callInitialize(int arr[][],int start,int end,int n)
```

Parameters: int arr[][], int start, int end, int n)
Return type: void
Description: This function calls the initializeRBT() function, which then initializes the Red Black Tree from the sorted array of events. The returned node is then assigned as root of the Red Black Tree, and its parent is set to nil. The time complexity of this function is O(n).

Output:

The outputs for all test input files are included in the project directory as op1.txt, op2.txt, op3.txt, and op4.txt.

The time taken for the different test files are given below:

```
test_100.txt
real    0m0.108s
user    0m0.000s
sys     0m0.015s

test_1000000.txt
real    0m0.406s
user    0m0.000s
sys     0m0.000s

test_10000000.txt
real    0m6.766s
user    0m0.000s
sys     0m0.000s

test_100000000.txt
real    2m53.150s
user    0m0.000s
sys     0m0.000s
```