COP 5536 Advanced Data Structure

Spring 2016

Programming Project Report

Implement an event counter using red-black tree

Bo Jian

0075-3810

bojian@ufl.edu

1. General Description

- Problem

1. Implement a red-black tree.
2. Implement increase, reduce, count, inrange, next ,previous method.
3. Use these methods to implement an event counter.

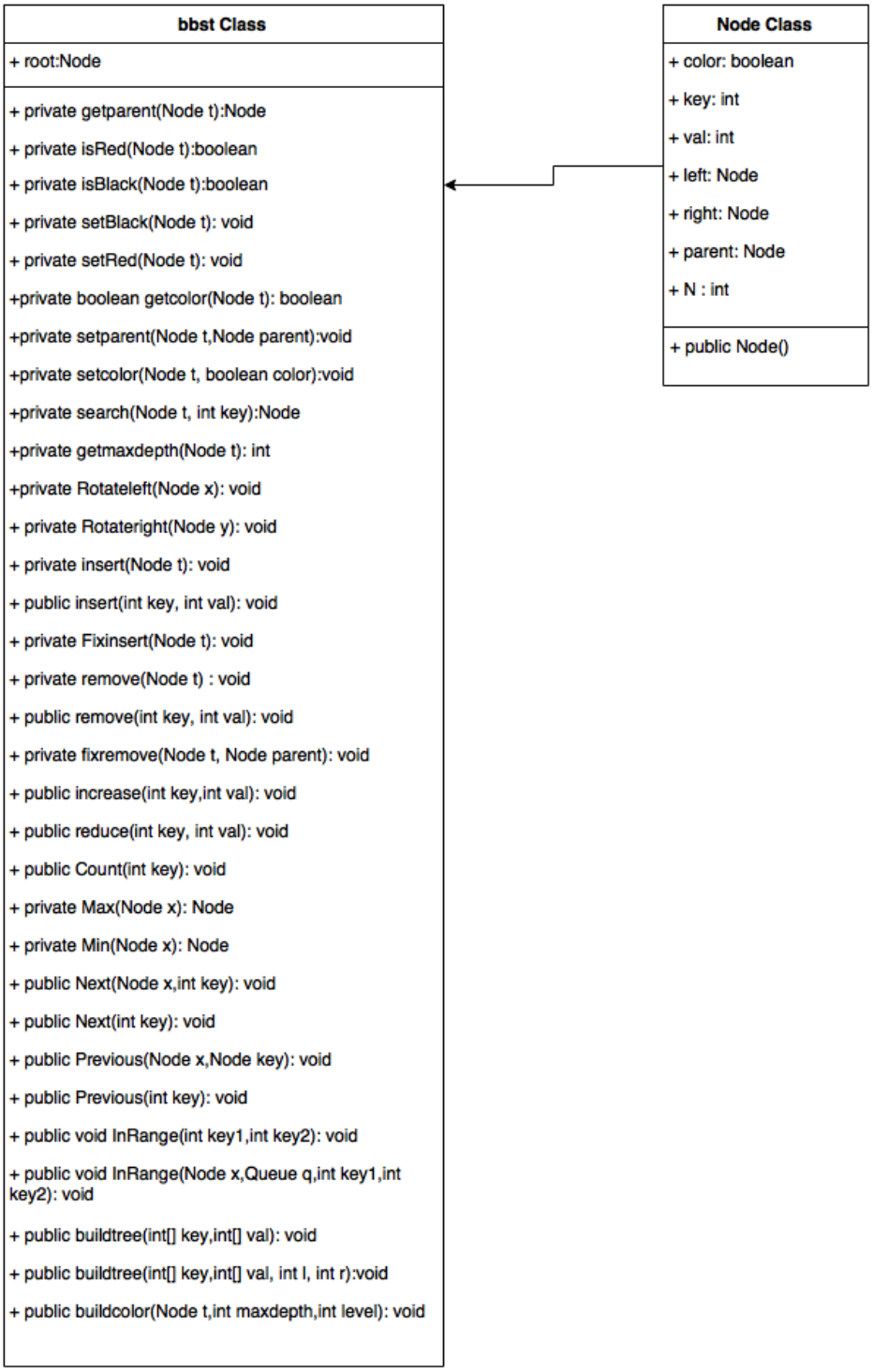
- Algorithm

1. Insert method and delete method for red-black tree
2. Insert fix method and delete fix method
3. Next, previous, count, inrandge method
4. Initialize red-black tree from sorted list method

- Programming Environment

1. Java Eclipse(Java Jdk8) on PC(using command line, need parameter)
2. Standard JDK on UNIX server(thunder.cise.ufl.edu)
   including Makefile file.

2. Structure of Program



3. Function Prototypes

3.1 Node Class

Node structure		
Name	**Type**	**Description**
color	boolean	The color of the node
key	int	The key of the node(correspond to the id of the event)
val	int	The value of the node(correspond to the count of the event)
left	Node	Point to the left child of the node
right	Node	Point to the right child of the node.
parent	Node	Point to the parent of the node.

3.2 bbst Class

bbst data field:

Name	Type	Description
root	Node	pointer to root of the red-black tree.

Public functions:

1	**void buildtree(int[] key ,int[] val)**
	Build a Red-black tree from the input(sorted list), first it build a binary search tree, then assign color to each node;
2	**void buildtree(int[] key ,int[] val, int l int r)**
	Build a balanced binary search tree.
3	**void buildcolor(Node t, int maxdepth, int level)**
	Assign color to a balanced binary search tree to form a red black tree.
4	**Void inrange(int key1, int key2)**
	Print the total count of keys between key1 and key2(inclusively)
5	**Void inrange(Node x, Queue q, int key1,int key2)**
	Helper function, helpers to get all nodes which key is between key1 and key2
6	**Void Previous(int key)**
	Print the key and the val of the Node with the greatest key which is less than key
7	**Void Previous(Node x,int key)**
	Helper function, helpers to get the Node with the greatest key which is less than key
8	**Void Next(int key)**
	Print the key and the val of the Node with the lowest key that is greater than the key.

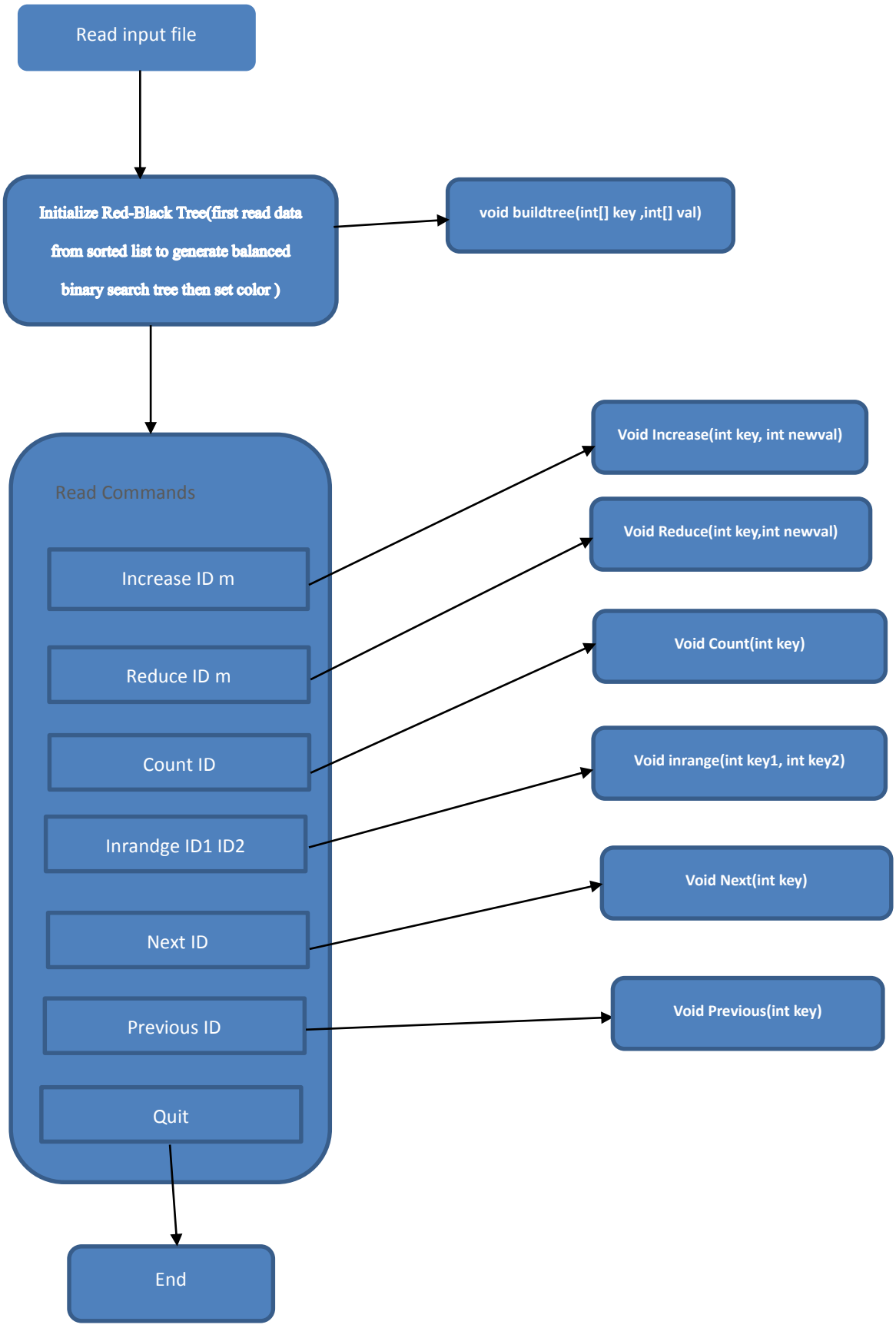
9	**Void Next(Node x, int key)**
	Helper function, helpers to get the Node with the lowest key which is greater than key
10	**Void Count(int key)**
	Given key, Print the val of key.
11	**Void Reduce(int key,int newval)**
	Decrease the val of key by newval. If the Node's val becomes less than or equal to 0, remove Node from the tree.
12	**Void Increase(int key, int newval)**
	Increase the val of the key by newval. If key is not in the tree, insert it.
13	**Void Insert(int key, int val)**
	Insert the Node with key and val into the tree.
14	**Void Remove(int key, int val)**
	Remove the Node with key and val from the tree.

Private Functions

1	**Node getparent(Node t)**
	Get the parent of the Node t;
2	**boolean isRed(Node t)**
	Whether the Node's color is Red.
3	**boolean isBlack(Node t)**
	Whether the Node's color is Black.
4	**Void setBlack(Node t)**
	Set the color of the Node to black
5	**Void setRed(Node t)**
	Set the color of the Node to red.
6	**Void setparent(Node t, Node pa)**
	Set the parent of Node t to pa.
7	**boolean getcolor(Node t)**
	Get the color of Node
8	**void setcolor(Node t, Boolean color)**
	Set the Node's color
9	**Node search(Node t,int key)**
	Given key, search the Node which has the key in the tree, if it is not in the tree, return null.
10	**Int getmaxdepth(Node t)**
	Get the depth of the tree;
11	**Void RotateLeft(Node x)**
	Rotate Left helper function in red-black tree.
12	**Void RotateRight(Node y)**

	Rotate Right helper function in red-black tree.
13	**Void insert(Node t)**
	Insert Node t to tree,helper function
14	**Void Fixinsert(Node t)**
	After insert, fix the tree in order to meet the request of red-black tree.
15	**Void Remove(Node t)**
	Remove Node t from tree, helper function.
16	**Void fixreomve(Node t, Node parent)**
	After remove, fix the tree in order to meet the request of red-black tree.
17	**Node Min(Node x)**
	Get the Node which has the maximum key in the tree.
18	**Node Max(Node x)**
	Get the Node which has the minimum key in the tree.

4. Program Flowchart



5. Performance Evaluation

4.1 Performance Requirement:

Command	Desctiption	Time complexity
Increase(*theID, m*)	Increase the *count* of the event *theID* by *m*. If *theID* is not present, insert it. Print the *count* of *theID* after the addition.	O(log *n*)
Reduce(*theID, m)*	Decrease the *count* of *theID* by *m*. If *theID*'s *count* becomes less than or equal to *0*, remove *theID* from the counter. Print the *count* of *theID* after the deletion, or 0 if *theID* is removed or not present.	O(log *n*)
Count(*theID*)	Print the count *of theID*. If not present, print 0.	O(log *n*)
InRange(*ID1, ID2*)	Print the total count for *ID*s between *ID1* and *ID2* inclusively. Note, *ID1* ≤ *ID2*	O(log *n* + *s*) where *s* is the number of *ID*s in the range.
Next(*theID*)	Print the *ID* and the *count* of the event with the lowest *ID* that is greater that *theID*. Print "0 0", if there is no next *ID*.	O(log *n*)
Previous*(theID)*	Print the *ID* and the *count* of the event with the greatest key that is less that *theID*. Print "0 0", if there is no previous *ID*.	O(log *n*)

5.2 Run time Analysis

Test Environment: Windows 7, Java 8, using command line.

Test file size : 100, 1000000, 10000000(numbers of events)

Result: I test the program with the test_100.txt, test_1000000.txt and test_10000000.txt , and the output of the first two files is the same as the sample result. The time required is divided into two parts: initialize time (including file reading time) and command time(which is the commands file TA offers, including 20 different commands).

n value (number of events)	Initialize time(millisecond)	Command time(millisecond)
100	5	2
1000000	1317	2
10000000	14207	2

Comments:

1. From the result showed above, we can see the time to initialize a red-black tree is approximately linearly increasing, but the time of the command is nearly the same because the number of the commands is only 20, if we execute more commands, we can see the difference.
2. We can see most of the time is spent on the initialization part, because read file from disk will spend a lot of time when the file is large.