COP 5536 – Project Report

Event Counter using Red Black tree

Sourav Kumar Parmar (UFID: 86511933 | souravkparmar@ufl.edu)

Instructions to run the program Build system: Linux (Ubuntu):Unzip the **Parmar_sourav.zip**

Submission details: "parmar_sourav.zip"

source file: bbst.cpp : implementation file**.**

makefile: makefile to compile code**.**

parmar_sourav_Report.pdf: Project report

- **Compile** :$ make
- **Clean**: $make clean
- **Running Instructions**:

  Create Red Black tree: $ ./bbst <filename>

  Interactive Commands: Command <param1> <*param2>
- Commands: $increase <event> <count>

  $reduce <event> <count>

  $count <event>

  $next <event>

  $previous <event>

  $inrange <event1> <event2>

  $levelorder : To display the level order traversal of the tree at any point.

  $quit: exit program from interactive mode

Compiler Used : gcc version: Ubuntu 4.8.4-2ubuntu1~14.04.1 (storm)

Language: C++

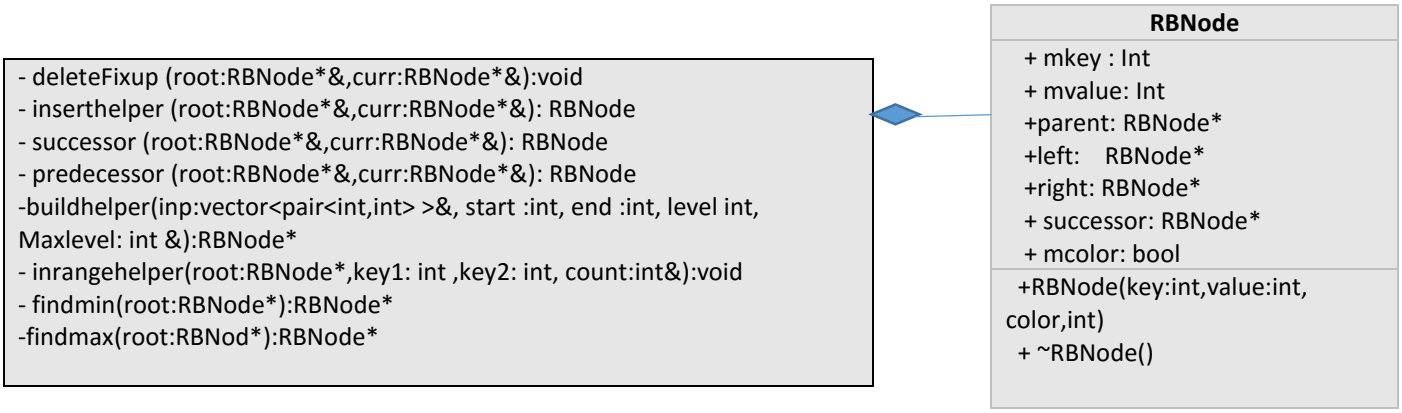
Class Diagram:

'-' implies private member (data/function)

+' implies public member (data /function)

Relationship: "Has a" composition (treemap composed of RBNode objects)

treemap
-root: RBNode* -nil :RBNode*
+buildtree(inp: vector<pair<int,int> >):int +colortree(maxlevel:int):void +inorder(root:RBNOde*, prev :RBNode*&, level:int ,maxlevel :int) :void +increase(key:int , value: int):void + insert(key:int , value: int):void +decrease(key:int , value: int):void +count(key:int):void +inrange(key1:int , key2: int):void +next(key:int):void +previous(key:int):void +levelorderprint():void +searchkey(root:RBNode*, key: int):RBNode* +findvalidnode(key:int, root: RBNode*, prev: RBNode*&, curr: RBNode* &, succ: bool):bool + deletenode(todelete:RBNode*&,root: RBNode*&,key: int):void + deletetree(root:RBNode*):void + getroot():RBNode* + rbnil():RBNode* +treemap() ~treemap() - rotateleft(root:RBNode*&,curr:RBNode*&):void - rotateright(root:RBNode*&,curr:RBNode*&):void - insertFixup (root:RBNode*&,curr:RBNode*&):void



Overview: Event counter is implemented using red black tree. A red-black tree is a binary search tree where each node has a colour attribute, the value of which is either * red or black

Invariants of a valid red-black tree:

1) The root is black.
2) All leaves are black.
3) Both children of each red node are black.
4) The paths from each leaf up to the root each contain the same number of black nodes.

Program Structure:

Build Tree:

STEP 1: Read the event, count pair from input file into a vector treevec
STEP 2: Use treevec to build a balanced Binary search tree
STEP 3: color the binary search **tree,** with all level except last coloured as black.

Function Prototype and Description:

void Buildtree(vector<pair<int,int> > &inp) :
Input vector is passed to helper program to build Balanced BST. Parent of root returned is made to point to sentinel nil node.

RBNode* buildhelper(vector<pair<int,int> > &inp, int begin, int end ,int level , int &maxlevel):
Build balanced BST from sorted input by creating mid node and recursively adding elements with index less than mid element in left subtree and elements with index greater than mid element in right subtree. Leaf nodes left and right child points to sentinel nil node.

void colortree(int maxlevel):
The balanced BST created in above step is converted to red black tree. The last level (maxlevel) is coloured red provided that the level is greater than 0 (root node is always black)

Commands:

- **void increase(int key, int value)**
  searchkey (key) is called to find if node is present in the tree. If the Node is present its count is increased by argument. If the Node is not present **void insert (int key, int value)** is called to insert node in balanced BST. The inserted node is coloured **red.** Insert calls **Void insertFixup (RBNode* &, RBNode*&)** to restore Red black tree invariant.

- **void decrease(int key, int value)**
  searchkey (key) is called to find if node is present in the tree**.** If the Node is not present it simply returns.
  If the node is present **deletenode(RBNode* &todelete, RBNode* &root, int key)** is called to delete node in balanced BST.
  If the deleted node is black **Void deleteFixup (RBNode* &, RBNode*&)** is called to restore Red Black tree invariant.

- **void count(int key)**
  searchkey (key) is called to find if node is present in the tree. If the node is present print its count else print zero.

- **void inrange(int key1, int key2)**
  It checks if key1 is less than key2. Function void **inrangehelper**(RBNode* root, int key1, int key2,int& count) is called . **inrangehelper** recursively traverse the left and right subtree if the node is in the range of [key1,key2] count is incremented with value of root, the count is passed as reference.

- **void next(int key)**
  checks if the key is less than minimum, if it is next is minimum. Otherwise, **calls bool findvalidnode(int key, RBNode *root, RBNode* &prev, RBNode* &curr,bool),** if the node is present in tree curr value is updated by that node

otherwise curr equals previous observed inorder node. Subsequently call to **RBNode* successor (RBNode*curr,RBNode* root)** is made which returns inorder successor.

- **void previous(int key)**
  checks if the key is greater than maximum, if it is previous is maximum. Otherwise, **calls bool findvalidnode(int key, RBNode *root, RBNode* &prev, RBNode* &curr,bool),** if the node is present in tree curr value is updated by that node otherwise curr equals current inorder node. Subsequently call to **RBNode* predecessor (RBNode*curr,RBNode* root)** is made which returns inorder predecessor.

Exception Handling:
Valid input file: If file name not valid throws exception
Command: if command is not a string Error is generated and user prompted to retry again.
Invalid param. If input command param is non integer value Error is generated and user is prompted to try again. for increase command if second param is less than equal to zero error is generated and user is prompted to enter integer value greater than zero.

Time Complexity:

Function	Running time	Description
Build RB Tree	**O(n)**	Build balanced BST , each element is visited only once
Color Tree	**O(n)**	Standard inorder traversal with last level coloured as red
Increase	**O(log n)**	Appropriate position is found in O (h) time (h is height of tree). If element is to be inserted insert fixup takes at worst O(h) time, overall running time O(h) i.e. O(log n).
Decrease	**O(log n)**	O (log n), to find the node to be deleted and deltefixup can take at worst O (log n), overall running time O (log n).
Inrange	**O(log n + s)**	O(log n)+s, O(log n) for search boundary elements of range, s is for number of elements within range, using modified inorder traversal only range elements are visited hence running time overall is O(log n)+s
Next	**O(log n)**	O(log n), find inorder successor running time is O(h) ,thereby O(log n)
Previous	**O(log n)**	O(log n), find inorder predecessor , running time is O(h) , thereby O(log n)
Count	**O(log n)**	O(log n), search in BST for the appropriate event, running time O(h), thereby O(log n)

Test Result:
Running time observed on storm.cise.ufl.edu server.

Number of elements	Running time
100	2 milli second
1000000	630 milli second
10000000	2 second
100000000	18 seconds

References:
Introduction to algorithms: CORMAN : Second Edition Chapter 13