ADS Programming Project

Sudeep Rege

This report details the structure and working of bbst, a Java program creating an event counter using red-black trees as a data structure. This project was created in Java using Eclipse IDE over Windows and tested on a Linux VM. The specifications of the system and the compiler are mentioned below. The report also describes the functionality of the event counter, the function prototypes that I used and the general structure of the program.

System specifications

This program was run and tested over a Linux VM with 64 bit Ubuntu and a base memory of 1280 MB. I also tested this program on storm, as the input files test_10000000.txt and test_100000000.txt required additional resources my VM couldn't handle. The programs were run on a JVM utilizing the OpenJDK Runtime Environment on mixed mode (Java version 1.6.0_27) and on storm after allocating around 4 GB as the maximum heap size on JVM.

How to Run

To run the program, simply call the makefile in the folder to compile the source code and create the executable. One can then run the program by using terminal or cmd with the command

java bbst <input_file commands

The input file is taken as a string specifying the location of the file in the standard input stream (System.in) , and the commands file can be passed as an argument to the program.

General Structure

The program is structured as follows : The source file contains code to create three classes: Node, Tree and bbst, which is the main class containing the functionality of the program. The node class creates objects storing individual elements of the event<id, count>, while the tree class creates Red-Black Trees to store these nodes in sorted order.
The main class takes as input a sorted list of events to be entered in the tree and creates a RB tree in O(n) time. It also contains the functions to perform the following functions:

Increase(id,n) : Increases the count of the event with ID id by n. If not found, inserts an event with elements<id, n>. Prints the event with updated count.

Reduce(id,n) : Reduces the count of the event with ID id by n. If count goes below 0, deletes that event from the structure. Prints the event with updated count.

Count(id) : Prints the count of the event with ID id, if present. If not, prints 0.

InRange(id1,id2): Prints the total count of the events with IDs between id1 and id2. If none present, print 0.

Prev(id) : Prints the element previous to the Id given. If none present, prints “0 0”.

Next(id) : Prints the element after the Id given. If none present, prints “0 0”.

These functions are implemented in the bbst class and are called from the main() function.

Function prototypes

Some of the function prototypes detailed below have duplicate public and private member functions. Both functions have almost the same functionality, with the public functions calling their private counterparts when necessary. This was done to make the code easy to understand.

Class Node

Data members: id(Stores the ID as integer), count(Stores the count as integer), left(Reference to left child), right(Reference to right child) and parent(Reference to parent of node), color(flag for red 'r' or black 'b')

Member functions:
Node(): Constructor creating a Nil node with left, right and parent set to null, and id, count and color undefined.

Node(int id, int n): Constructor creating a Node with ID as id and count as n, and left, right and parent set to null.

isNil(): Boolean function to check whether node is a Nil node or not.

print(): Prints the elements of the node<id, count> .

Class Tree

Data members: Node root for storing the root of the tree.

Member functions:

insert(Node n): inserts Node n in the tree structure at the correct position.

l_rot(Node x): left rotates the subtree rooted at Node x.

r_rot(Node x): right rotates the subtree rooted at Node x.

insertFixup(Node n): Maintains the RB properties which may be violated after inserting Node x.

transplant(Node x,Node y): Transplants the subtree rooted at y to the one rooted at x.

delete(Node z): Deletes the node z from the red-black tree.

deleteFixup(Node x): Maintains the RB properties which may be violated after deleting Node z.

print(): Prints the tree in preorder fashion.

search(int id): Searches for the node with ID id passed to it. Returns the node if found, else it returns Nil node.

Class bbst

Member functions:

main(String args[]): Main function, creates the tree from the file in the InputStream, takes arguments passed to it, parses the resulting file for commands and applies them to the tree structure if available.

createTree(Tree t, int arr[][], int start, int end): Creates a red black tree from the given sorted array arr from positions start to end by recursively making the middle element the root node and the sublists start to middle and middle to end as left and right subtrees respectively. Returns the root of the tree .

increase(Tree t, int id, int m): Performs the functionality of the Increase operation detailed above.

reduce(Tree t, int id, int m): Performs the functionality of the Reduce operation detailed above.

count(Tree t, int id): Performs the functionality of the Count operation detailed above.

printNext(Tree t, int id):Performs the functionality of the Next operation detailed above.

next(Tree t, int id): Returns the successor of the node n with ID id, and Nil if not found.

printPrev(Tree t, int id):Performs the functionality of the Prev operation detailed above.

prev(Tree t, int id): Returns the predecessor of the node n with ID id, and Nil if not found.

inRange(Tree t, int id1, int id2): Returns the total count of all nodes between id1 and id2, and 0 otherwise.