ADS PROJECT REPORT

NAME: SIVASUBRAMANIAN KANDASWAMI

UFID: 51447667

SOFTWARE SPECIFICATIONS

The JDK version1.8.0, update 73 (jdk1.8.0_73) was used to compile this project.

PROGRAM STRUCTURE

The program consists of two files with a '.java' extension

1. **bbst.java :** This file is the executable and contains the logic to read the test cases from a given test input file name. It reads the commands from the commands file provided the filenames are given in the command line arguments. The commands such as 'increase' and 'reduce' are rea from the file at this stage and the corresponding functions are called, which are located in the EventTree.java file.
2. **EventTree.java:** This file contains the definition of the Event node, its properties such as eventID, count and color. The EventTree class contains the definition of the functions that are required by the assignment such as 'increase' and 'reduce' and a host of other helper functions which are require to perform these operations.

Bbst.java Structure

->Required header files

-> Reading input from the file containing the test cases.

->Reading the input from the file containing the commands

->Calling the functions corresponding to the commands defined under EventTree.java

EventTree.java Structure and Function Prototypes

-> Required Header Files

-> Definition of the Event Node

-> Definition of the EventTree the Red Black tree.

->Increase(Event id, m)

->insert (Event e)

->insertFixup(Event e)

->Reduce(Event id, m)

->delete(Event e)

->deleteFixup(Event e)

->transplant(Event u, Event v)

->rotateLeft(Event e)

->rotateRight(Event e)

->Count(id)

->Inrange(id1,id2)

->totalKeyCount (start,end,Event e)

->Next(id)

->findSuccessor(Event e)

->closestNextEvent(Event e)

->Previous(id)

->findPredecessor(Event e)

->closestPreviousEvent(Event e)

->locateEvent(id)
->treeMin(Root)
->treeMax(Root)
->isRed(Event e)

The structure above describes the layout of the functions and the organization of the programs. The nested arrows indicate functions that are used to play a helping role in other functions.

Function Descriptions

1.Increase : The function increase increases the 'count' of the event specified by the id by 'm'. This operation has a time complexity of O(logn).
2.Insert : The insert method inserts an event whose key is assumed to have already been filled in, into the red-black tree. This operation has a time complexity of O(logn).

3. **insertFixup**: The insertFixup method deals with the following cases
    - Case 1: z's uncle y is red
    - Case 2: z's uncle y is black and z is a right child
    - Case 3: z's uncle y is black and z is a left child
4. **Reduce:** The function reduce decreases the 'count' of the event specified by the id by 'm'. This operation has a time complexity of O(logn).
5. **Delete:** This method deletes the Event passed as input to it. This operation has a time complexity of O(logn).
6. **DeleteFixup:** Performs the rotations and coloring required to balance the tree if required after the deletion.
    - Case 1: x's sibling w is red.
    - Case 2: x's sibling w is black, and both of w's children are black
    - Case 3: x's sibling w is black, w's left child is red, and w's right child is black.
    - Case 4: x's sibling w is black, and w's right child is red
7. **Transplant:** Transplant replaces one subtree as a child of its parent with another subtree.
8. **leftRotate:** The left rotation function allows the rebalancing of an unbalanced tree/subtree. This operation can be executed in constant time.
9. **rightRotate:** The right rotation function allows the rebalancing of an unbalanced tree/subtree. This operation can be executed in constant time.
10. **InRange**: The function inrange prints the total count of the events between the specified ids. The time complexity for this operation is O(log n + s) where s is the number of IDs in the range.
11. **totalKeyCount:** Determines the total keycount within the specified range .
12. **next:** The function next prints the 'count' the count of the event with the lowest ID that is greater that eventID and prints '0 0', if there is no next ID. This operation has a time complexity of O(logn).
13. **Previous:** The 'previous' function, prints the ID and the count of the event with greatest key that is less that eventID and prints '0 0', if there is no previous ID. This operation has a time complexity of O(logn).
14. **closestPreviousEvent:** Determines the closest event with a lower eventID.
15. **closestNextEvent:** Determines the closest event with a higher eventID.
16. **treeMax:** Finds the event with the highest id in the tree. This operation has a time complexity of O(logn).

17. **treeMin:** Finds the event with the lowest ID in the tree. This operation has a time complexity of O(logn).
18. **isRed:** Returns true if the event is colored red and false otherwise.