COP 5536 – Advanced Data Structures

Spring 2016

Programming Project

This project to develop an event counter using Red Black tree was developed using the Dev C++ IDE and the minGW32 compiler on a Windows 10 64-bit machine. I also used the Visual Studio IDE and Visual C++ compiler to debug and test the code.

The program initializes a red black tree from a sorted list of n events in O(n). Once the tree is initialized the following functions can be performed:

1. Increase(theID, m) : O(log n)
2. Reduce(theID, m) : O(log n)
3. Count(theID) : O(log n)
4. InRange(ID1, ID2) : O(log n + s) where s is the number of ID's in the range
5. Next(theID) : O(log n)
6. Previous(theID) : O(log n)

These time complexities ensured due to the fact that Red Black trees are balanced binary search tree with an upper bound of 2 log (n+1) on the height of the tree.

Program Structure and Function Prototypes:

To store the values and the properties of each of the node we use a struct node which has the following components:

1. long int ID : Key
2. int count : Event Count
3. int height : Height of Node
4. char color : Color of node
5. node *left : Pointer to left child
6. node *right : Pointer to right child
7. node *parent : Pointer to parent

The Red Black tree is stored as an object using the class RBtree which contains all the functions and a pointer to the root of the tree. When the object of class RBtree is initialized we initialize a sentinel node i.e. a node which is colored black and has the left, right and parent pointers pointing to itself.
So when the tree contains no element the root of the tree will be a sentinel node.

The program starts by reading the data from the input file and writing it into the 2-D array data. Data is of type long int has a size of [100000000][2]. Data is also declared as a global variable so that it does not use the memory on the program stack.

Once the variable Data is initialized we call the following function to initialize the Red Black tree.

- **node* initialize(long int arr[][2], long int start, long int end, node* father, int h, int check);**

  This function takes the following parameters as input:

1. long int arr[][2]: The array with the sorted list
2. long int start: The starting index of the array (initially 0)
3. long int end: The end index of the array (initially n-1)
4. node* father: Pointer to the parent node(initially null)
5. int h: Height of the node
6. int check: maximum height / height of last level of tree

This function builds an almost complete binary search tree by setting the middle element of the array as the root and then recursively building the left and right subtrees.
Once the binary search tree is complete we set the color of the nodes on the last level of the tree as red and color the rest of the nodes black. In this way all the properties of Red Black trees are satisfied.

- **void insert(long int key, long int count);**
  This function is used to insert a node in the Red Black tree. It takes in theID and the count of the node as parameters and performs a normal binary search tree insert. Once the insert is completed this function calls the following function to fix any violation caused due to this insertion.
  - **void insertfix(node *);**
    This function takes in the node that has been inserted as a parameter and performs required series of rotations or color flips depending upon the conditions and ensures that the Red Black tree properties are satisfied.

  Note: When we insert a node in the Red Black we set the height of the node as -1 as the value of height becomes insignificant after the tree is initialized and none of the other functions depend on the height.

- **void del(long int key);**
  This function is used to delete a node from the Red Black tree. It takes in theID of the node as parameter and performs a normal binary search tree delete. Once the deletion is completed and the node that was physically deleted is black node this function calls the following function to fix any violation caused due to this deletion. In case the deleted node is red no further process is required.
  - **void delfix(node *);**
    This function takes in the node that has been deleted physically as a parameter and performs required series of rotations or color flips depending upon the conditions and ensures that the Red Black tree properties are satisfied.

- **void leftrotate(node *);**

  This function is used by the “insertfix” and “delfix” functions takes in the node as a parameter and performs left rotate on that node

- **void rightrotate(node *);**

  This function is used by the “insertfix” and “delfix” functions takes in the node as a parameter and performs right rotate on that node.

- **node* successor(node *);**
  This function is used by both functions “del” and “search_pn” (explained later) to find and return the pointer to the successor the node passed as parameter to this function.

- **node* predecessor(node *);**

This function is analogous to the successor function and returns the pointer to the predecessor of the node passed as parameter. It is also used by “search_pn” (explained later).

- **void modifycounter(long int key, long int delta, int func);**

  This function is used to perform both Increase(theID, m) and Reduce(theID, m). It takes in theID, delta i.e. the value the count need to be increased or decreased by and another int func; if the value of func is 1 the Increase(theID, m) is performed and if it is 0 the Reduce(theID, m) function is performed.

  This function will first search for the node with theID. If the node is not found it is inserted using the “insert” function in case of Increase(theID, m) operation and the value of new count is printed; In case of Reduce(theID, m) “0” is printed.

  If the node is found the value of its count will be increased by delta in case of Increase(theID, m) operation and decreased by delta in case of Reduce(theID, m) and then the value of the new count is printed. If at any point the count of the theID becomes less than 0 the node is deleted by calling the “del” function and then “0” is printed.

- **void searchcount(long int x);**

  This function takes in theID of the node as a parameter and prints the value of its count. If the node is not found “0” is printed.

- **long int inrange(node *, long int low, long int high);**

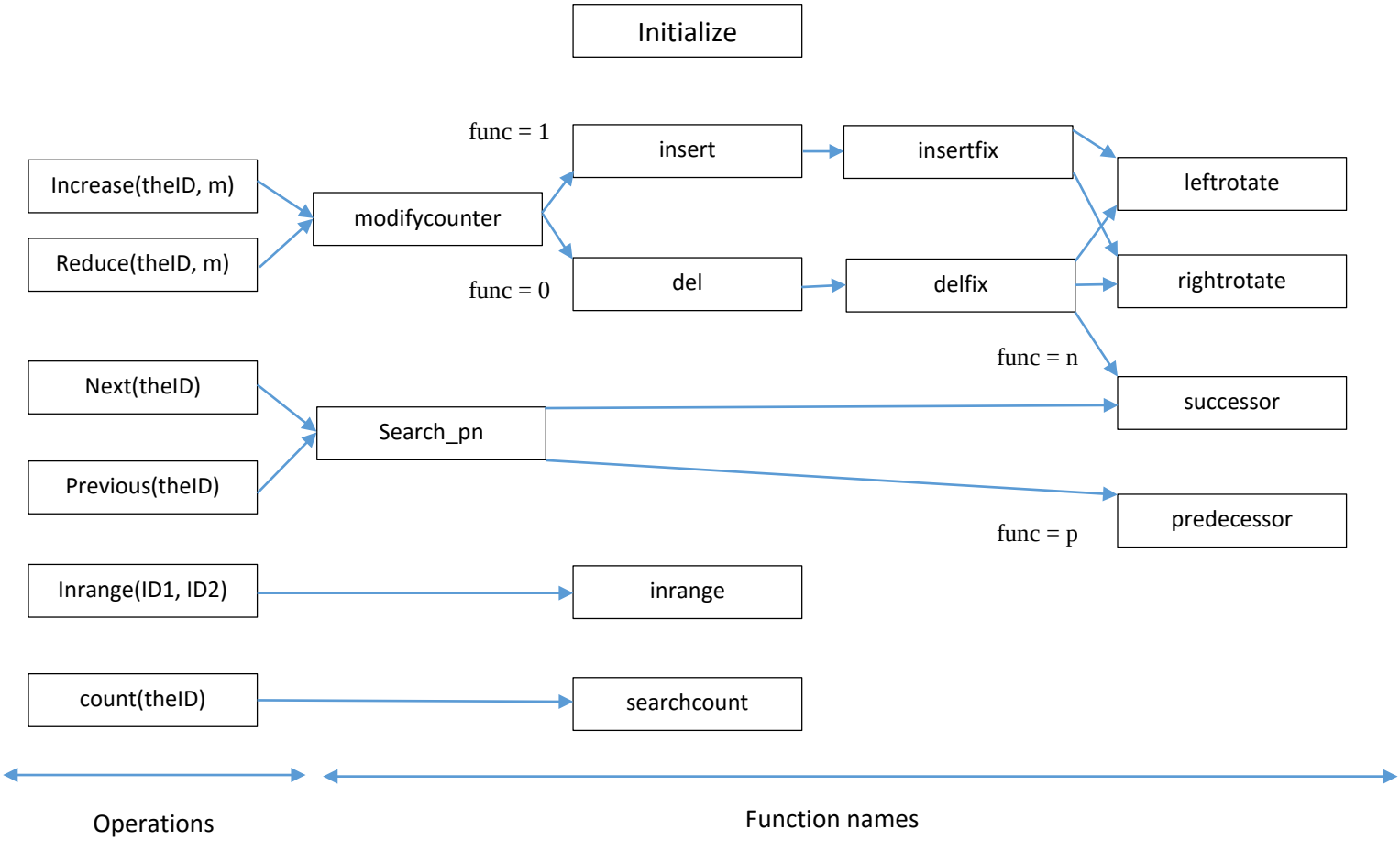
  This function takes in low and high values of a range as parameters along with a pointer to the root of the tree and returns the sum of all counts of theID’s that lie within that range. If any node is in range its count is included in the sum and then recursive calls to its left and right subtrees are made.

- **void search_pn(long int x, char func);**

  This function is used to perform both Next(theID) and Previous(theID) operations. It takes in theID of the node whose previous or next node needs to be found and a char func: if value of func is “n” the Next(theID) operation is performed and if it is “p” the Previous(theID) operation is performed.

  Next(theID) : If the node is found its successor is found using the “successor” function and the value of its ID and count is printed an in case the successor does not exist “0 0” will be printed. In case the node is not found the value of the count of theID and theID of the node which is the successor of the node whose theID values is just less than the input is printed.

  Previous(theID) : If the node is found its predecessor is found using the “predecessor” function and the value of its ID and count is printed an in case the predecessor does not exist “0 0” will be printed. In case the node is not found the value of the count of theID and theID of the node which is the predecessor of the node whose theID values is just greater than the input is printed.



References:

1. Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. Introduction to Algorithms (2nd ed.). McGraw-Hill Higher Education.
2. Coder's Hub: Android and Programming Tutorials: Red Black Tree : http://www.coders-hub.com/2015/07/red-black-tree-rb-tree-using-c.html
3. http://www.geeksforgeeks.org/print-bst-keys-in-the-given-range/
4. http://geeksquiz.com/c-program-red-black-tree-insertion/
5. https://github.com/zyi23/Red-Black-Tree
6. http://www.geeksforgeeks.org/sorted-array-to-balanced-bst/