UFID-11714071

Project by : Tirth Doshi

Email Id: tirth.doshi@ufl.edu

Under the guidance of : Prof. Dr. Sartaj Sahni

Rising City Project Report

Chapter 1: Node Structure and Assumptions

A) Node Structure:

There are 2 nodes created in this project. One is a node for the RedBlackTree and one is a node for the min heap.

1. **RedBlackNode** : This node contains the below fields:
    a. **Executed Time:** This is the amount of time that the building has been currently worked on.
    b. **Total Time:** This is the amount of time required to complete the building.
    c. **Building Number :** This is the building identifier through which one identifies the building.
    d. **Color:** This is the color of the node in the RedBlack Tree
    e. **Left and Right pointer:** These are the child pointers in the RedBlackTree
    f. **Parent Pointer:** There is a parent pointer as well in this node
    g. **Pointer to Minheap :** A pointer to minheap is maintained in the redblack node.
2. **MinHeapNode :** This node contains the below fields:
    a. **Executed time:** The definition for this is same as above.
    b. **Total Time:** This is the total time required to complete the building
    c. **Building Number:** This is the building identifier which is unique for every building.
    d. **Pointer to RedBlackTree:** This field stores a pointer to its corresponding node in the redblack tree.

B) Assumptions

Depending on the understanding the following are the assumptions made during the project:

1. Whenever there is an insert command and the global counter is equal to the current time then we insert into the MinHeap and the RedBlackTree irrespective of what is currently being executed.
2. If there is a print statement on the day when the building construction is completed, the print statement works prior to the print of the constructed building.
3. Also, the code does not handle duplicate building numbers.

Chapter 2: Rising City Class

This class contains the main function and has various elements to be understood.

1. **Significance of variables**:

a. Filename: This variable stores the name of the file coming from the command line argument.

b. Command: At any point of time, this variable will contain the next command required to be executed.

c. current_time: This is the time given in the command variable. For example, "0: Insert (15,20)" would mean that the current time would be 0.

d. globalcounter: This is a counter that starts from 0 and increments after every iteration of the while loop.

e. checkprint : In case if there is a print statement on the day that the building gets executed, the print function will get executed first and checkprint will make sure that the print command does not get executed again in the while loop.

f. Totaltime: At any point the value of this variable would be the sum of all the total times inserted.

g. picknextbuilding: Would be 0 when we need to continue the work on 1 building, would be 1 when we need to pick the next building.

h. no_of_days_worked: At any point this variable gives the value of how many days consecutively have we worked on the current building getting executed.

i. Minheapnode x1: x1 is the current building we are working on. If x1 is null it means we are not working on any building.

j. rb: This is a redblack tree object.

k. mp: This is a minheap object.

l. b: This variable always stores the time of the next day command. For example if we have " 0:Insert(15,20)\n 15: Insert(20,30) " b will hold the value 15 since we do not want the execution to stop if there are new queries.

2. Execution of the while loop:

Algorithm:

a. If the current_time is equal to the global counter,read the next line and store the previous line in temporary variable temp.
b. If temp contains an insert then insert it into the minheap and the redblacktree.
c. If temp contains a print statement, print the required building in the required format.
d. If picknextbuilding is 1 then we do the extract min from the minheap. Basically, we are picking the building to work on such that its executed time is less & in case of a clash, we pick a building that has minimum building number.
e. If executed time is less than the total time and no of days worked for is less than 5 we increment the executed time by 1 in both the structures, i.e the minheap and the RedBlackTree. We also increment the no of days worked by 1.

f. If the no of days worked becomes equal to 5 and the executed time is not equal to the total time, then we reinsert that building into the minheap.
g. If the executed time at any point becomes equal to the total time, and if there is a print query for that building we print the building first and then output the building.

Chapter 3: MinHeap & its functions

Minheap : The minheap is implemented using the array of objects.

The following are the functions implemented in minheap:

a. **private void minheapify(int i) :** This function takes a position as its argument. This function compares its right and left child and swaps the one that is having smaller executed time. In case of a tie, we compare the building numbers and swap the one with smaller building number.
b. **public void insert(minheapnode node) :** This is simply used to insert the minheapnode into the minheap.
c. **public void insertheapify(int child):** This function compares the executed time of its parent and if the parent execution time is greater than we swap the nodes. We recursively call the function with the argument child/2.
d. **public void insertminheap(int a, int b , int c) :** This is a function that simply creates the object and calls the insert function.
e. **public minheapnode extractmin() :** Here the minheapnode is popped and replaced by the last node in the minheap. After that we call the minheapify at the first position.

Variables of minheap:

a. **Size:** This variable gives the size of the minheap at any point of time
b. **Capacity :** This is initialized to 2000 according to the problem statement.
c. **First:** This will always hold 1.

Chapter 4: RedBlackTree & its functions

RedBlackTree: A basic RedBlackTree is created using the RedBlackNode described above. The redblacktree is created on the basis of building numbers as given in the problem statement. Below are the functions of the RedBlackTree:

a. **public void printTree(RedBlackNode root):** This function simply prints the tree in inorder fashion along with its corresponding color.
b. **Private RedBlackNode Search(RedBlackNode findRedBlackNode, RedBlackNode temproot):** This function is used to search a particular node in the redblack tree and if it finds that node it returns that node else it returns a null pointer.
c. **Private void insert(RedBlackTree insertnode):** This function inserts the node just like we insert into a binary search tree and later calls the fix function to fix it according to the redblack tree.
d. **Private void fix(RedBlackNode insertednode) :** This function is used to fix the redblack tree along with its properties.
e. **Void rotateleft(RedBlackNode insertednode):** This function is used to do the rotation anticlockwise direction wrt insertednode.
f. **Void rotateright(RedBlackNode insertednode):** This function is used to do the rotation in clockwise direction wrt insertednode.
g. **void changeparent(RedBlackNode py, RedBlackNode y):** This function is used to change the parent pointers in case of a deletion.
h. **Boolean delete(RedBlackNode deletednode):** This function is used to do the deletion in the redblack tree. There are 6 cases of deletion which are handled in the delete fix function.
i. **Void deletefix(RedBlackNode deficientnode):** This function is used to fix the deletion since delete causes one branch to be deficient in terms of the number of black nodes. There are 6 cases of deletion that are handled by this function.
j. **RedBlackNode successor(RedBlackNode node):** This function returns the leftmost child of the right subtree.
k. **RedBlackNode insertprint(int a,int b,int c) :** This function is used to do the insert of a particular node and returns that node after inserting.
l. **Public void rbdelete(int a,int b, int c):** This function is used to delete the node from the redblack tree. It makes a node with values a, b, & c and calls the delete function.
m. **Public void printsingle(int a):** This function is used while a print query comes of a specific node.
n. **Public void printsinglewithroot(int bno,RedBlackNode temproot):** This function is used for finding the building number returning that building.
o. **Public void printrange(int b1, int b2):** This function is used to print the range of building numbers between b1 & b2. All these nodes are stored in the array list.

p. **Public void printwithroot(int b1, int b2, ArrayList<RedBlackNode>res):** This functions stores the range in the array list result.