ADS Project Report - 2019

Name: Pratik Shamsundar Loya
UFID: 31716903
Email: ploya@ufl.edu

Overview:

The Project consists of 4 classes.

1. **risingCity**: This is the main class that performs following operation:
    i. Initialize the objects of the other classes
    ii. Read the command from the Input file.
    iii. Runs the global counter and perform operation based on the input read from file
    iv. Call MinHeap function to perform construction on the current building
    v. Checks for the completion of construction of the town.

2. **Node:** This class is used to store the information of the building. It comprises of the following building data.
    i. **int BuildingNum** – Building Number
    ii. **int executed_time** – Execution Time on the building
    iii. **int total_time** – Total time required for building completion.

3. **MinHeap**: This class is used to perform operation related to Min Heap.
    i. Consists of MinheapNode comprising of
        1. **int execution_time** - Execution time on current building
        2. **Node node** - Pointer pointing the node in the RBT
    ii. Insertion and Remove Min from the Min Heap Data Structure
    iii. Increases the execution time of the building currently working on.

4. **RedBlackTree**: This class perform operation related to Red Black Tree.
    i. Insertion and Deletion from the Red Black Tree.
    ii. Write the building data into the file.

Function Prototype:

Class: risingCity

1. **Main Function:** This function initializes all the object and starts the global timer. The function reads the commands from the input file provided and performs operation when the global timer is equal to the time of the command. During each iteration it calls the work function (present in MinHeap

Class) to increase the execution time of the current building. When all the inputs are read and the RBT structure becomes empty then it exits the loop.

Class: MinHeap

Consist of following class

```
class MinheapNode {
        int executed_time;
        Node node = null;
}
```

Consist of ArrayList of MinheapNode: ArrayList<MinheapNode>

1. **public MinheapNode removeMin ():**
   This function returns the object present at the index 0 of the ArrayList. It replaces the element at index 0 with the last element and then perform comparison between the child and parent until the min heap property is satisfied.

2. **public int left_child (int parent):**
   Returns the index of the left child of the parent

3. **public int right_child (int parent):**
   Returns the index of the right child of the parent

4. **public void insert (int executed_time, Node node_pointer):**
   This function creates a new object of MinheapNode and initializes it with the executed_time and node_pointer provided and add it to the ArrayList and call the heapify function to check for the min heap property.

5. **private void heapify():**
   Starting from the last index, this function maintains the min heap property after insertion of the new node. It continuously compares the child with parent and swap the two when the execution time of the child is less than that of the parent. In case when execution time is same as that of the parent it checks for the lower building number.

6. **public MinheapNode work (MinheapNode CurrentWorkingBuilding):**
   This function accepts the CurrentWorkingBuilding as input. If CurrentWorkingBuilding is null then it removes the building with the lowest execution time from the ArrayList by calling the removeMin function. It then increments the execution time of the building in both the MinheapNode object and also in the corresponding node in the RedBlackTree. The CurrentWorkingBuilding is returned to the main function

Class: RedBlackTree

1. **public void checkRootNodeColor():**
   If the root color in the RBT Data Structure is Red, then flip it to Black.

2. **public void leftRotate(Node node) and public void rightRotate(Node node)**
   This function performs the corresponding rotation of the node given.

3. **public void flipColors(Node parent,Node grandParent,Node uncle):**
   This function is called when a new node is inserted and the uncle of the newly inserted node is Red. At that time the color of grandparent, parent and uncle are flipped.

4. **public void balanceTree(Node childNode)**
   This function is called when the newly inserted node is red and the parent of the node is also Red. This function maintains the property of RBT by performing rotation based on the position of the newly inserted node.

5. **public Node insert(int buildingNumber,int totalTime):**
   This function creates a new object of Node class and insert it in the data structure as per the insertion into the Binary search tree. After insertion, if the parent of newly inserted node is black then no operation is performed. But, if it is Red then it calls the balanceTree function in order to maintain the Red Black Tree property.

6. **public Color getColor(Node node):**
   This function return the color of the node.

7. **public void stabilizeDeletion(Node node,Node parent)**
   This function is called after deletion of the node in order to maintain the red black property. Depending on the color and position of the deleted node, there are 20 cases
   a. **When node deleted is red or child of deleted node is Red – no operation**
   b. **When the node is deleted from right sub tree**
      When the sibling is BLACK
         Case Rb0: When both the children of sibling is black.
            Case Rb0.1: When parent is Red. Change color of parent and sibling and DONE
            Case Rb0.2: When Parent is Black. Change color of parent and sibling and continue to check in the above sub tree

         Case Rb1: When one children of sibling is red.
            Case Rb1.1: When left child of sibling is red. Perform LL rotation
            Case Rb1.2: When right child of sibling is red. Perform LR rotation

         Case Rb2: When Both child of sibling is RED. Perform LR rotation

      When the sibling is RED
         Case Rr0: Here parent will be Black. Perform LL rotation and DONE
         Case Rr1: Here parent will be Black. One child of sibling is RED
            Case Rr1.1: When left child of right child of sibling is RED. Perform LR rotation
            Case Rr1.2: When right child of right child of sibling is RED. Perform LLR rotation
         Case Rr2: When both children of right child of sibling is Red. Perform LLR Rotation

   c. **When the node is deleted from left sub tree**
      All the above operation are performed exactly opposite.

8. **public Boolean delete(int buildingNumber):**
   The function deletes the node from RBT. If the node to be deleted has two children then it is replaced by its immediate predecessor and now its immediate predecessor node is deleted from RBT. If the deleted node is Red then the RBT property is still maintained. But, if the deleted node is black then it calls the stabilizeDeletion() function to maintain the RBT property.

9. **public String customised_inorder(Node node, int building_number_1, int building_number_2):**
   This function performs an inorder traversal through the tree and appends the details of the information that lies in between the two building numbers into the output file.

10. **public void PrintBuilding(int building_number, BufferedWriter writer):**
    This function outputs the details of the building

11. **public void PrintBuilding(int building_number1, int building_number2,BufferedWriter writer):**
    This function calls the customised_inorder() function to output the details of the building lying in the given range.

Project Flow:

The main function reads the first line of the input file and starts the global timer. When the global timer is equal to the timer of the input command it perform the corresponding operation.

1. When the command is Insertion it first calls the RBT.insert() function and inserts the node in RBT structure. Second it gets the reference from the function and pass it to MinHeap where the MinHeap creates a new MinheapNode and store the execution time and node reference into its ArrayList.
2. When command is PrintBuilding then it calls the function RBT.PrintBuilding depending on the number of the parameter passed.

The CurrentBuilding (object of MinheapNode) stores the information of the building on which the current work is going on. It is initially set to null. The main function calls the MinHeap.work() function that takes the CurrentBuilding as a parameter. When it is null, the work() function removes the object from ArrayList with the least execution time and stores it in the CurrentBuilding object and increment the execution time of that object and its corresponding RBT node and increase the count of days worked on the building by 1. After this the main function than checks for following two condition

1. if the execution time of the current building is equal to its total time than
    a. Output its completion date.
    b. Remove the node from the RBT (As the building is already removed from minheap)
    c. Reinitialize the CurrentBuilding object to null so that it can start working on the next building
    d. Reinitialize the count of days worked on current building to 0
2. If the days worked on current building is equal to 5 than
    a. Reinsert the CurrentBuilding object back into the ArrayList
    b. Reinitialize the CurrentBuilding object to null
    c. Reinitialize the count of days worked on current building to 0

When there is no input to read from the file and when the last node from the RBT is deleted then the project is completed.