COP5536 ADVANCED DATA STRUCTURES

PROJECT FALL 2019

MEGHA SAI KAVIKONDALA

UFID: 4754-3974

UF-mail: mkavikondala@ufl.edu

Program Structure:

The project is divided into Four classes of-

1) risingCity.java
2) Storage.java
3) min_Heap.java
4) RBTree.java

The description for each class is as follows:

1) risingCity.java- This class mainly reads the inputs from the file. The global counter is also taken where if it is equal to the day given performs the given operation on the building. The limit for this operation is till the 5th day. When total time equal to the execution time then print the building number, the total time and execution time. Then remove the building from both Red black tree and min heap. The function prototypes used are:

```
main(String[] args) - Here, the variables are initialised. The inputs
from the file are also taken and stored. A global counter is maintained
in form of gcc which compares itself with the days that are incremented.
If both are equal then operations on building are performed. If not then
perform the operations on remaining previous buildings.

operation()- This method calls the operations to be performed based on
the Command line argument given which is either Insert or Print.

insert_Operation()- This method performs the insertion operation by
calling the Red Black tree and Min heap in order to insert the node in
it.

print_Operation()- This method performs the print operation. Based on
the arguments given to the print method, either the given building
number and its total time will be printed or a range of building numbers
present between the arguments are given.

split_cmd()- This method is used for splitting and replacing the white
spaces, brackets and commas present in the input line.

delimiter_remove()- This method removes the delimiter and divides the
input into two parts of day and operation to be performed.

 min_Heap getHeapObj()- This method is used for returning the minimum
heap object created in the class.

currentBuilding.getExecuteTime()- This method is used for getting the
executed time from the Storage class for the current building.
```

minHeap.getSize()- This method is used for getting the size of the minimum heap.

currentBuilding.getTotal_time()-This method is used for getting the total time from the Storage class for the current building.

currentBuilding.setExecutionTime()-This method is used for setting the executed time for the current building.

2) Storage.java- This mainly acts as a container for maintain the building number, total time and execution time. The red black tree and minimum heap contains its object for usage. The function prototypes used are:

   class Storage- This class is used for initialising the variables and the methods present.

   getExecuteTime ()- This method is used for getting the execution time for the current building.

   getBuildingNums ()- This method is used for getting the current building numbers.

   setExecuteTime ()- This method is used for setting the execution time for the current building.

   getTotal_time ()- This method is used for getting the total time for the current building.

3) Min_Heap.java- The minimum heap functionalities present all are executed in this class. It takes the object of Storage class for its usage. The function prototypes are :

   Class min_Heap- This class is used for initialising the variables and the methods present.

   getSize()- This method is used for getting the size of the minimum heap.

   parentNode(int)- This method is used for getting the position of the parent node in the minimum heap.

   rightChildNode(int)- This method is used for getting the position of the right child node in the minimum heap.

   leftChildNode(int)- This method is used for getting the position of the left child node in the minimum heap.

   leaf_node(int)- This method receives an index and it checks whether the node at that particular index is a leaf node or not.

   heapify(**int**)- This method takes the index and it will balance the minimum heap present with the property of heapify in order to maintain the heap property.

   insertion(Storage)- This method takes the node in order to insert it into the minimum heap.

minHeap()- This method is used for building the minimum heap through the heapify function mentioned.

remove(int)- This method takes building number and assigns the position in order to pass it as an argument to the position_removal().

position_removal(int)- This method takes the position and it is used for removing the minimum element present in the heap and returning it.

fetchNode(int)- This method takes the index and returns the node present in that index of the minimum heap.

4) RBTree.java- The red black tree functionalities all are executed in this class. It takes the help of the Storage class object for its usage. The function prototypes present are:

   class RBTree- This class consists of all the variables and the methods that are initialised.

   node_exist(RBNode, RBNode)- This method checks whether the given node is present in the Red Black Tree or not. This function is called upon when there is need to check during the delete function.

   insertion(Storage)- This method takes a node and call the insert() method.

   insert(RBNode, RBNode)- This method gets called by the insertion method and takes the Nodes as arguments and it is used for inserting the node into the Red Black Tree.

   RB_insfixup(RBNode)- This method takes the node in order to fix the red black tree property after any operation done on it.

   LR(RBNode)- This method takes the node in order to rebalance the tree through left rotation.

   RR(RBNode)- This method takes the node in order to rebalance the tree through right rotation.

   rbTransfer(RBNode,RBNode)- This method is used for transferring or replacing the sub tree with another sub tree.

   findBuildingNum(int)- This method takes the building number and is used for finding the building number in the red black tree.

   deleteNode(int)- This method is used for deleting the building number from the red black tree.

   deleteNode(RBNode)- This method is used for deleting the node.

   rb_DeletionFix(RBNode)- This method takes the node and it is used for the fixing the color and Red Black tree properties after deletion of node.

   minNode(RBNode)- This method takes the node and is used for finding the minimum node at the subtree root.

Search(RBNode)- This method is used for searching the node in the red black tree with the given id.

printBuilding(**int**)- This method is used for printing only the building number current and its total time and executed time.

printBuildRange(**int, int**)- This method is used for printing all the building numbers present between the given building numbers.

setptr(int, int)- This method sets a pointer between the Red black tree and minimum heap in the map.

deleteFromptr(**int**)- This method removes the given building number from the map where there is a pointer between the red black tree and minimum heap. Here, this is done after the removal of the element from both the trees.

getValueFromptr(**int**)- This method is used for getting the value from the map.

Instructions for Compilation-

- The given project has been compiled in the java compiler
- Execution:
  1) javac risingCity.java input.txt
  2) the output is given to the output_file.txt

Conclusion-

- The program that is implemented now is made efficient by time complexity by using minimum heap and red black tree. The nodes store the execution time and building number for individual trees given. Due to this it takes O(1) time to get the next building to work on.
- Through red black tree, the complexity is O(log n) for printing the building or printing the range of the building.