Advance Data Structures (COP 5536)
Project

Name: Sitanshu Sukhmandir Lamba
UFID:5190-8991 Email: slamba@ufl.edu

1. Description

The aim of the project is to help Wayne Enterprises in developing a new city. This is a software that helps them keep track of the buildings under construction. The input file consists of three types of commands:-

1. Insert(building number,total time)
2. Print for a range of buildings or print one particular building.

The output file is written into in two cases:

1. A building completes execution
2. A print command arrives in the input file

Before each command line is the global time at which that command is arriving.

The data structures used in this project are:

1. MinHeap
2. Red black Tree
3. A custom data structure of each building consisting of building number(unique), executed time and total time

The programming language is JAVA. The input file has been given in the form a .txt file. The output is also provided in the form of a .txt file with write functions in the programs.

2. How To Execute:-

The zip file “Lamba_SitanshuSukhmandir.zip” contains a java file “risingCity.java”.
First compile the code using “javac risingCity.java”.
To run the code, run the command “java risingCity filename.txt”.

Output will be ready in the file “output_file.txt”.
Two input files have been included if required.
Any other input file needs to be added in the same folder as the java file after unzipping.

3. How the main function executes

The main function executes with the following steps

1. It first starts with creating the Minheap, Red Black Tree and a Queue for pending insertions. All data structures use the main Buildings data structure.
2. The **first instruction** is assumed to come at **time 0.** It is read and stored into a String. This string is split according to the regex \\W and stored into a String array of contents.
3. The value of Building Number and total time for completion are taken from the contents array and stored into a variable of the data structure “Buildings”. This is inserted into the MinHeap and the RedBlackTree.

4. Now it enters the loop of reading the input file till the end.

5. The simple working of this loop goes as follows

    - It checks if the size of the minheap is 0 then the pending insertions in the queue are added to the minheap and minHeapify() is called on the heap.
    - If the execution time of the element at the root becomes equal to its execution time, then the root is removed(from both heap and red black tree) and printed in the output file along with its global time of completion.
    - If the execution time of the root has completed 5 days of execution. The queue of remaining inserts is emptied into the minheap and minHeapify() is performed to decide the next Building to be worked on.
    - The global timer and the time are increased by 1 on each iteration.
    - If the Global time becomes equal to the time at which the command from the input file is arriving then the command is executed. If it is a print command then the necessary writing is done in the output file. If it is an Insert command, the new

Building is added to a queue and inserted into the Red Black Tree.
- After completing the reading from the file the minheap is worked on till it is empty using the same rules of execution.
- As Buildings get completed they are added to the output file along with the global time at which they were completed.

6. The file writer is closed and the output file is ready with the name "output_file.txt".

4. Classes And Methods Used (Program Structure):-

The main function has been explained earlier. Here are the remaining classes and their functions.

Class Buildings:

This is the main data structure used in the program. It consists of the details about the building relevant to the program:

Methods:

1. Buildings ():
    - *Description*: It is the basic constructor of the class that sets the values of buildingNum, executed_time and total_time.
    - *Parameters*: int buildingNum, int executed_time and int total_time.
    - *Return type:* void

Class MinimumHeap:

This class is the minheap of the buildings arranged according to building executed time and building number (if executed times are equal). The building to be worked on is always at the root of the heap.
size denotes the current size of the heap (1 <= size <=2000)
maxSize denotes the largest size possible
minheap[] is an array of Buildings storing the buildings being inserted.

Methods:

1. rootElement ():
    - *Description*: returns the root of the tree
    - *Parameters*: none
    - *Return type:* Buildings

2. minHeapify ():
    - *Description*: It creates the minheap according to the executed time of each building. The building with the lowest executed time is brought on top of the heap. In case when the executed times are equal then the Building with the lower buildingNum is given priority.
    - *Parameters*: int pos
    - *Return type:* void

3. remove ():
    - *Description*: Removes the element at the root of the minheap, decreases the size of the heap, replaces the root with the last element in the heap and minHeapifies the heap at its root
    - *Parameters*: none
    - *Return type:* void

4. Swap ():
    - *Description*: Swaps the values at the two positions of the heap given as input in the parameter.
    - *Parameters*: int fpos, int spos
    - *Return type:* void

5. lChild ():
    - *Description*: Returns the left child of the position given as parameter
    - *Parameters*: int p (position)
    - *Return type:* Integer

6. rChild ():
    - *Description*: Returns the right child of the position given as parameter
    - *Parameters*: int p (position)
    - *Return type:* Integer

7. isLeaf ():
    - *Description*: Checks if the Building at the given position is a leaf of the heap or not.
    - *Parameters*: int p (position)
    - *Return type:* Boolean

8. minHeap ():
    - *Description*: Calls minHeapify on all the buildings that are not leafs.
    - *Parameters*: none
    - *Return type:* void

9. insert ():
    - *Description*: Inserts a new building into the minheap and increases the size by 1.
    - *Parameters*: Buildings element (to be inserted)
    - *Return type:* void

Class RedBlackTree:

This class is the RedBlackTree that stores all the Buildings as a Red Black Tree according to their Building numbers.
This class is used when the print command arrives and the current state of the required buildings are then written into the output file.
The Buildings as passed by reference into the red black tree and therefore any updates make to each Building are visible during printing.
R = 0 (colour red)
B = 1 (colour blue)
This class consists of the class **RedOrBlackNode.**
It contains the Buildings value, left child pointer, right child pointer and parent pointer p. The garbage value is equilvalent to the null value.

Methods:

1. insert ():
    - *Description*: inserts a new Building into the tree.
    - *Parameters*:

- *Return type:* Buildings

2. findAndPrint ():
    - *Description*: finds specific Building in the tree and returns it in the form of a String.
    - *Parameters*: int b (building number to be found)
    - *Return type:* String

3. findRedOrBlackNode ():
    - *Description*: finds whether a particular building is present in the red black tree from a particular Building.
    - *Parameters*: RedOrBlackNode n1,node
    - *Return type:* RedOrBlackNode

4. correctRbt ():
    - *Description*: performs the necessary changes in the tree after a building is inserted.
    - *Parameters*: RedOrBlackNode n1
    - *Return type:* void

5. rleft ():
    - *Description:* rotate the whole subtree/tree at the node to its left.
    - *Parameters*: RedOrBlackNode node
    - *Return type:* void

6. rRight ():
    - *Description*: rotate the whole subtree/tree at the node to its right.
    - *Parameters*: RedOrBlackNode n1
    - *Return type:* void

7. printRange ():
    - *Description*: checks and returns the nodes in the given range as parameter.
    - *Parameters*: int i ,int j
    - *Return type:* String

8. printRBTinRange ():

- *Description*: works as a supplementary function to printInBetween().
- *Parameters*: int i ,int j
- *Return type:* String

9. deleteAssist ():
    - *Description*: helps the delete function by changing the ranks and making the changes in the tree after a Building is deleted. Target is the node to be deleted.
    - *Parameters*: RedOrBlackNode target,with
    - *Return type:* String

10. delete ():
    - *Description*: the main delete function which uses a lot of other subroutines to make sure the building is deleted properly while maintining the properties of the tree th
    - *Parameters*: RedOrBlackNode target.
    - *Return type:* String

11. delete ():
    - *Description*: used as a supplementary function to printInBetween
    - *Parameters*: RedOrBlackNode flow int i, int j
    - *Return type:* Boolean

12. treeDeleteChanges ():
    - *Description*: used to change colour and rotate necessary subtrees after deletion has been performed.
    - *Parameters*: RedOrBlackNode flow int i, int j
    - *Return type:* void

13. functionAccess ():
    - *Description*: Decides which function needs to be performed on the tree which is either insertion or deletion. The choice parameter chooses the function and Buildings b is the Building to be inserted.
    - *Parameters*: RedOrBlackNode flow int i, int j
    - *Return type:* void

5. Test Cases:-

The test cases that were run on this code have been included in the zip file. Here are few of the screenshots for the input as well as the output.

Input:-

```
0: Insert(50,100)
45: Insert(15,200)
46: PrintBuilding(0,100)
90: PrintBuilding(0,100)
92: PrintBuilding(0,100)
93: Insert(30,50)
95: PrintBuilding(0,100)
96: PrintBuilding(0,100)
100: PrintBuilding(0,100)
135: PrintBuilding(0,100)
140: Insert(40,60)
185: PrintBuilding(0,100)
190: PrintBuilding(0,100)
195: PrintBuilding(0,100)
200: PrintBuilding(0,100)
209: PrintBuilding(0,100)
211: PrintBuilding(0,100)
```

Output:-

```
(15,1,200),(50,45,100)
(15,45,200),(50,45,100)
(15,47,200),(50,45,100)
(15,50,200),(30,0,50),(50,45,100)
(15,50,200),(30,1,50),(50,45,100)
(15,50,200),(30,5,50),(50,45,100)
(15,50,200),(30,40,50),(50,45,100)
(15,50,200),(30,45,50),(40,45,60),(50,45,100)
(15,50,200),(30,50,50),(40,45,60),(50,45,100)
(30,190)
(15,50,200),(40,50,60),(50,45,100)
(15,50,200),(40,50,60),(50,50,100)
(15,55,200),(40,54,60),(50,50,100)
(15,55,200),(40,55,60),(50,51,100)
(40,225)
(50,310)
(15,410)
```

The next input and output are large so the screenshot for the first few command have been provided in the report.

Input:-

```
0: Insert(16385,138)
4: Insert(19059,179)
32: Insert(5296,133)
130: Insert(13764,126)
220: Insert(15569,153)
283: Insert(4480,60)
304: Insert(13257,109)
325: Insert(10492,169)
402: Insert(15269,83)
466: Insert(3207,140)
480: Insert(18289,91)
495: Insert(5131,88)
563: Insert(4219,63)
571: Insert(19833,130)
605: Insert(802,142)
669: Insert(12593,198)
749: Insert(1428,128)
845: Insert(13213,185)
917: Insert(9390,91)
1004: Insert(18442,125)
1076: Insert(13320,32)
1101: Insert(13188,141)
1159: Insert(4172,197)
1235: Insert(2609,13)
1245: Insert(2116,48)
1293: Insert(19770,108)
1326: Insert(11644,180)
1372: Insert(12639,41)
1399: Insert(898,59)
1429: Insert(15864,191)
```

Output:-

```
(13320,1147)
(2609,1265)
(12639,1506)
(2116,1619)
(4727,1703)
(17218,1712)
(898,2181)
(16280,2794)
(15911,3138)
(7625,3348)
(13275,3407)
(11478,3540)
(9847,4124)
(6617,4400)
(2230,4873)
```