# ADS PROJECT REPORT

NAME : Y.S.Siddharth
UFID : 1361-7920
UF-Email : siddhart.yellurs@ufl.edu

## Implementation/Understanding/Structure of the Problem :

Based on my understanding of the problem statement, I have used a min-heap data structure and a Red-Black Tree in order to meet the desired output requirements. I have first defined a class called Building which has the following attributes which is buildingNumber, executedTime, totalTime and another variable called the tempExectime. The buildingNumber stores the building number, executedTime keeps a track of the executed time with respect to the global time, totalTime contains the value of the total time required to finish the building and tempExectime is a variable we have used to make the functioning of the executedTime easier.

According to the input statement, the building number and the total time taken is given along with the global time at which it has to be inserted. Also a building is executed only when its executed time is the lowest with respect to others. A building is executed for a global time of 5 and then the condition is checked. In case the executed time equals the total time, then the building doesn't have to run for a global time of 5.

We have used the minheap data structure in order to store the building and we use all its attributes from the building class. In the min heap, we make sure that the root of the node is the building with the smallest executedTime value. The other nodes are inserted as the left and the right child of this node. We also have a pointer called minBuilding which always points to the root node of the min heap. After a global time of 5, we check for the smallest executedTime in all the nodes and the node with smallest value automatically moves to the root. Whenever the building gets completed, it is removed from the min heap and the Red Black tree (executedTime == totalTime). It immediately gets printed when it is completed. In case the executed time of both the buildings are the same, we then check for the building number (bn) and select the building with the lower building number.

In case of the Red Black tree, the buildings are inserted into the Red Black Tree as and when it is added in the input. And when a building is completed, the node is entirely removed from the RedBlack Tree. We keep track of the global Time and use it accordingly. We print the output at the multiples of 5 in the global time and keep printing till all the buildings are completely finished. In case of the print command, we print all the buildings between the specified range of the building number . We have also initialized the building array to a max limit of 2000. Thus the limit for the number of buildings is 2000. We read an external input file and the output is written into an output file which is stored outside. Thus, we get the required output when the input file is given. The project has been implemented in Java.

**Functions used to implement Project** :

I have used various function to implement the project. We shall look at the function used in the min heap

```
1)                  public void insert(Building element)
```

This function is used to insert a node into the heap, it checks if the executed time of the current heap is lesser than the current executed time of the parent and also checks if the buildingNumber of the current node in the heap is lesser than the root node. If it is true, then we perform a swap between the current node and the parent node.

```
2)                            private int parent(int pos)
```

This function returns the position of the right child for the heap node which is currently at position value. It returns (pos value * 2) + 1.

```
3)                           private int leftChild(int pos)
```

This function returns the position of the left child for the heap node which is currently at position value. It returns (pos value * 2).

```
4)                           private boolean isLeaf(int pos)
```

This returns a Boolean value and checks if the passed node is a leaf node or not. It compares it with the size(height of the min heap) and returns true if it is greater than size. This means that the passed node is a leaf node.

```
5)                           private void minHeapify(int pos)
```

Function is used to perform the heapify function. First it checks if the node is a non-leaf node and is greater than any of its child nodes. If the condition is true it swaps with the left child and performs the heapify function. If the condition is not true, then it swaps with the right child and performs the heapify function.

```
6)                            public void print()
```

This function is used to print all the contents of the heap. It is put into a loop and it prints the Parent and its left and right child.

```
7)                          public Building remove()
```

This function is used to remove and return the minimum element from the heap. We pop the element to be removed and move the next smallest element to the root which we call the heap front.  If the size is not 0, then we pass the root element as an argument and call the minheapify function.

RED BLACK TREE :

We define a class for the red black tree with the public variables red, black. We then get the building with its attributes from the rising_City file as the node.

```
1)                       private void insert(Node node)
2)                       private void fixTree(Node node)
3)                       void rotateRight(Node node)
4)                        void rotateLeft(Node node)
```

These function is used to insert nodes into the Red black Tree. It checks for the conditions of the red black tree insertion and performs the functions as such. In case the root node has the value nil, then the node is made the black node. If the node is to be inserted in the middle it is given the red value and the different cases and conditions for the red black tree insertion is checked. Once the  nodes are inserted, the different test cases are checked in fix tree which performs the required rotations based on the test conditions. In case of rotate right, the function is called and the same goes for left rotate.

```
5)                         public boolean delete(int bn)
6)                         void deleteFixup(Node x)
```

This function is used to delete the tree from the Red Black Tree. In this case the first function just deletes the tree from the red black tree. Once that is done, it calls the deleteFixup function which in turn performts the required operations in order to maintain the balance in the redBlack tree. It checks for the number of black nodes in each path and checks if there are equal number of black nodes in every path.

```
7)                   public void printTree(Node node)
```

This function is used to print the contents of the tree. It checks if the node is not nill and then prints the node of the tree using this function.

```
8) private List<Building> findNodes(int bn1, int bn2, Node node, List<Buildin
   g> list
```

This function is used to find nodes based and we pass the arguments bn1, bn2 and node. We use this to find which building number has the lower value amongst the two buildings which are being compared. It traverses through the red black tree and finds the node required.

RISING CITY

In this main file, we perform the main functions and function calls required to implement the program in the file. We initialize the variables heap and tree in order to use it for function calls to the Heap and the RBT in the program. We also initialize arrays such as buildingstoRemove, BuildingsWaiting and global time along with a changer in this class.

```
1)       public void insertBuilding(int buildingNo, int time)
```

This function is used to insert a new building number and it takes in arguments such as buildingNo and time(Total Execution Time). We then initialize a node b with the attributes buildingNo, 0 ,time.

```
2)                   boolean passOneDay() throws IOException
```

This function is used to increment the temp_time every time the global time increases. At multiples of 5 global time, we equate the value of temp time to executed time. We then call the heap.remove() to remove the building from the node and check for the building which has the least executed time. It automatically gets updated to the root node. We already have a pointer to the root node which we call minBuilding. Thus we use this pointer to insert the new building into the root. We also check if the executed time equals the total time. If that is the case, then we exit that the building with its attributes from the tree and call a print function and perform a close.

```
3) public void printBuildings(int buildingNo1, int buildingNo2) throws IOExce
   ption
```

This function is used to print the building node b at multiples of 5 with respect to global time. It prints all the buildings and its current status at that particular global time between the the two values buildinfNo1 and buildingNo2. It uses these two values and uses their building numbers as the range. Using that function, it throws all the values that lie in this range. Thus we get the buildings between the given two building bumbers with all its attributes.

```
4)              public static void main(String[] args)
```

This is the main function. We read the input from an external file in this used the BufferedReader command and then use the readLine command to check and parse the values. When the time is greater than the global time, we perform the passOneDay command in order to increment the global time. We perform a few iinteger parsing commands and then write the ouput to an external file which we name output_file which is a text file. It calls the printBuilding command when the string 'i' meets the various conditions. Once the loop has been broken, we close the file reader.

## Time Complexity :

The program uses RBT and MinHeap and thus follows the time complexity of these data structures. It executes the time complexity in O(log(n)+S) where n is number of active buildings and S is the number of triplets printed. I have complied to the search conditions specified in the question where only those subtrees are entered which is in the possible building range. All the other operations take O(logn) time.