Advanced Data Structures (COP 5536) Fall 2019 Programming Project Report

Vishisth Chaturvedi
UFID 1787-6619
vchaturvedi@ufl.edu

PROJECT DESCRIPTION

The project is emulating the workings of Wayne Enterprises, a construction company. The project handles an object Building. Running the building process according to the required constraints.

It consists of two main parts. The Min Heap and The Red Black Tree. These two are used in combination to perform the required three operations:

A. Print Operation

B. PrintRange Operation

C. Insert Operation

The program reads the input file from the command prompt and then generates an output file named “output_file.text”.

STRUCTURE OF THE PROJCT AND FUNCTION DESCRIPTION

The implementation is based on five classes which are:

1. risingCity.java
2. HNode.java
3. MinHeap.java
4. RBTree
5. RBNode

Classes Description:

Below I have mentioned all the classes and some of their important functions:-

1. risingCity.java

This is the main driver class. It emulates the working of the project. It uses the MinHeap and RedBlackTree to store the present buildings. It keeps track of the global time.

Functions

A. **private void run():**- This is the main coordinater function. It takes the input commands. Works in cycles of 1 global unit time. It gives preference to reading the input commands over the current process execution. It calls the builder function to work on the building. When it detects input command it calls the insertBuilding function. On detecting print command it runs both print and print in range command.

B. **private void builder()**:- This the command which works on the building. It checks if the work on the building is complete or it has been working for 5 days.

C. **private void printBuilding(String[] values)**:- It performs both the print and print in Range command.

D. **private void insertBuilding(String[] values)**:- It inserts the building. It does so by inserting the building into both the heap and the tree.

2. HNode.java

This is the basic node class. It is the node which is the basic structure of each node in the heap.

Member variables

A. **public int executedTime** - The key in this heap is the executed time.

B. **public RBNode rbNode** - Each node has a reference to the equivalent RBNode;

3. MinHeap.java

Functions

A. **public int size() -** It return the size of the heap.

B. **public boolean isEmpty()** - Return if the heap is empty.

C. **public void add()** - Adds a new HNode to the heap.

D. **public void upHeapify()** - Heapify function used to maintain the heap property.

E. **public HNode remove()** - Remove function.

F. **public void swap()** - Swaps two nodes in the heap.

G. **public boolean isLarger()** - Returns the larger of the two nodes.

4. RBTree

Functions

A. **public void insert() -** Inserts a new node into the red black tree. It then uses the fixup function to maintain the Red Black Properties.

B. **public boolean delete() -** Deletes the node with the given key. Then uses the fixup function to correct the issue created due to the delete. Does this in a bottom to up manner.

C. **public RBNode search() -** Looks for the node with the given key.

D. **public List<RBNode> searchInRange() -** Looks for the nodes which are in the given range.

E. **public void rightRotate()** - Basic rotate function similar to the AVL tree.

F. **public void leftRotate()** - Left equivalent of the above function.

5. RBNode

This is the basic node structure of the RB Tree.

Member variables

A. **int Building_ID -** The unique id of each building.

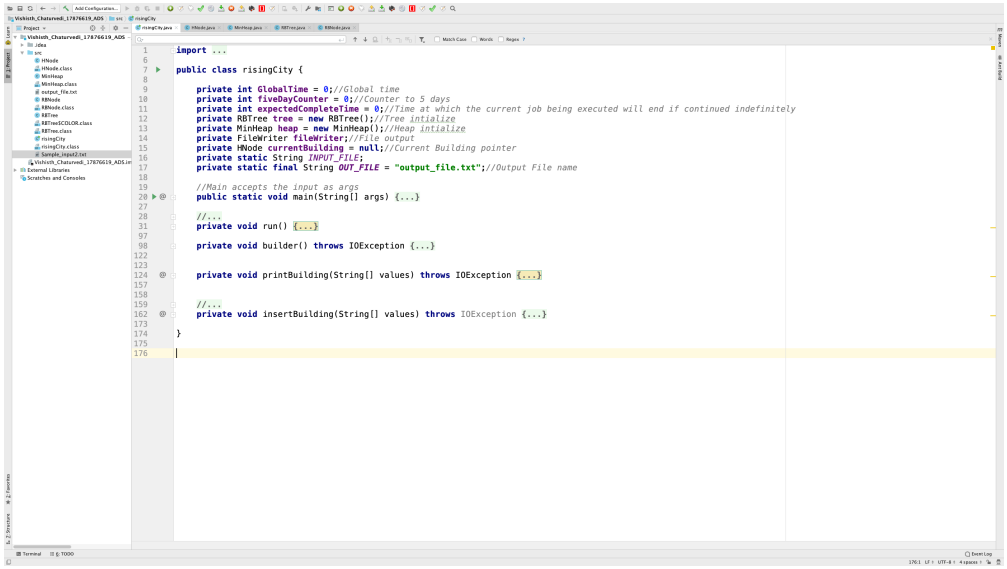
B. **RBNode left -** Points to the left child of the node.

C. **RBNode right -** Points to the right child of the node.

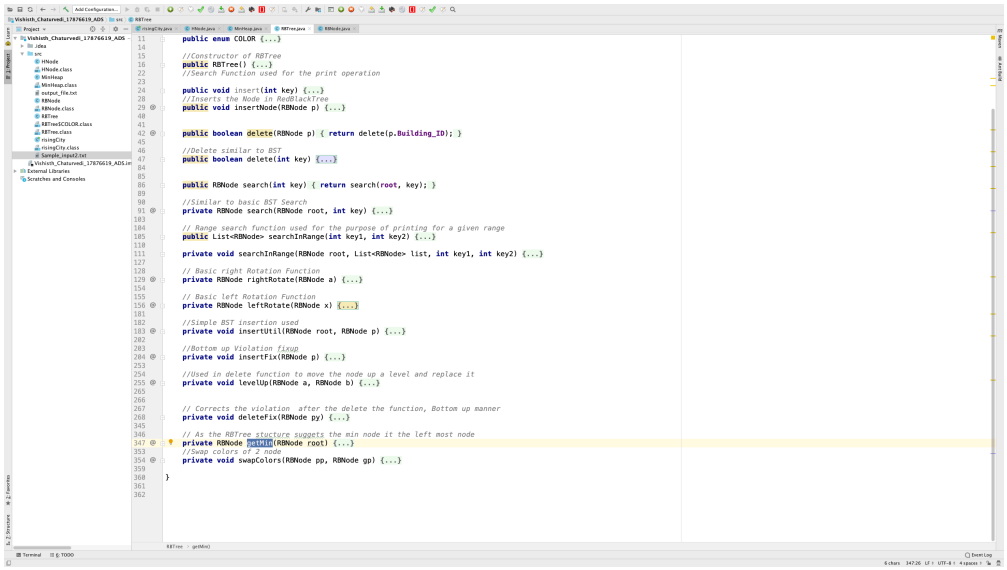
D. **RBNode parent** - Points to the parent of the node.

E. **HNode heapnode** - Reference to the equivalent HNode in the heap.

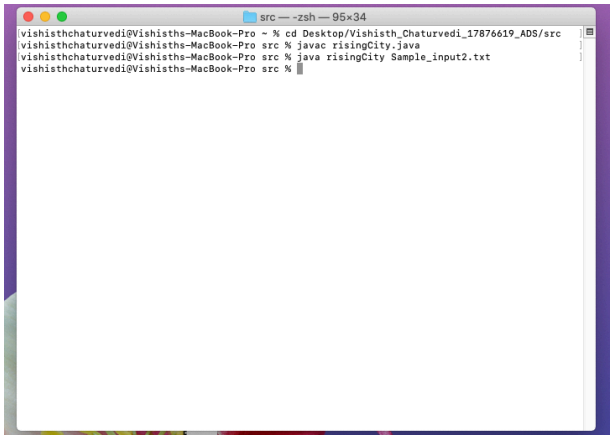
SCREENSHOTS

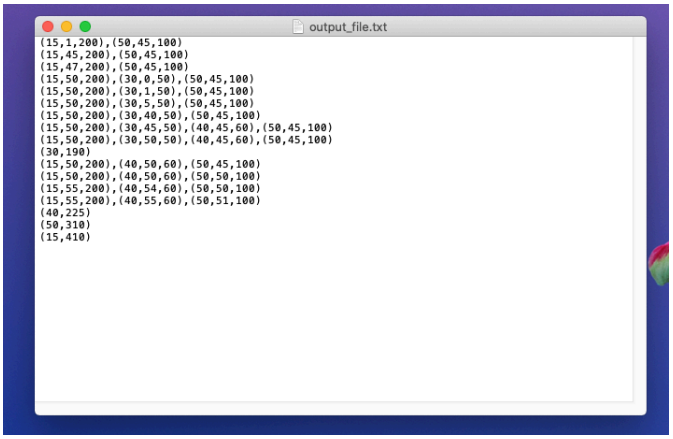


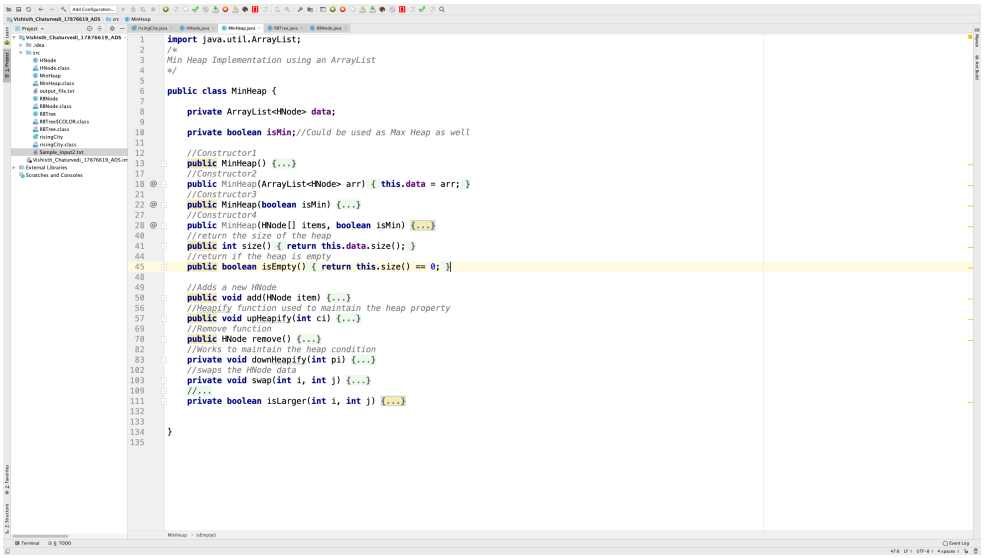
risingCity.java



RBTree.java

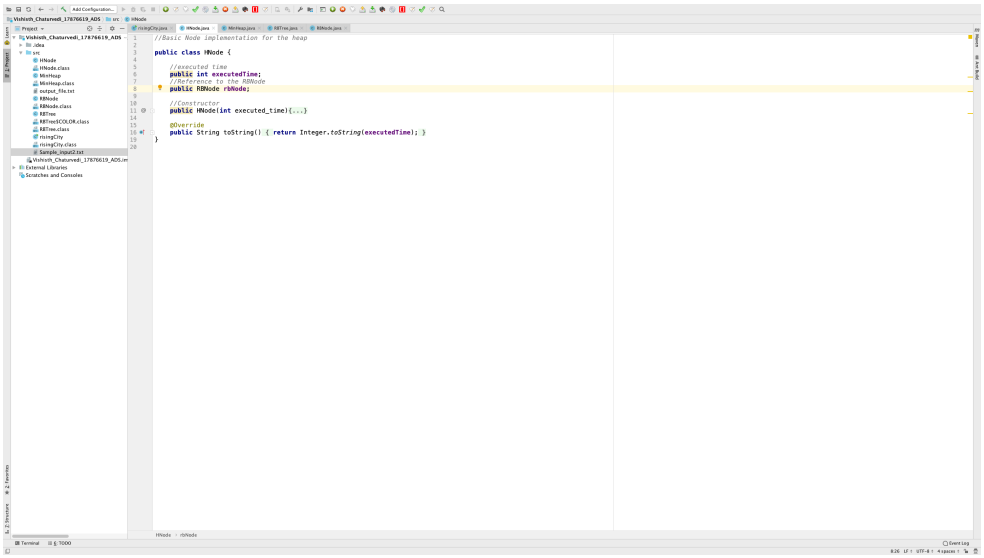






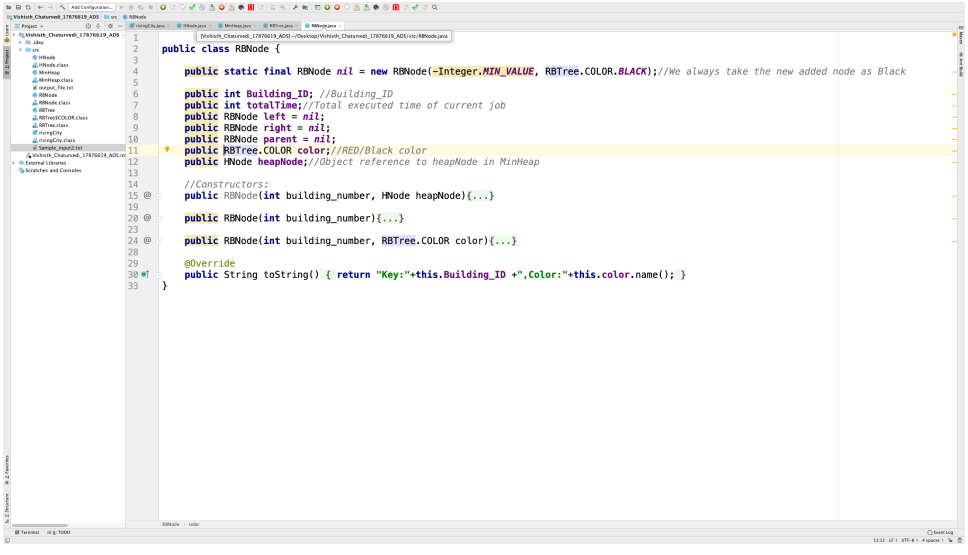
```java
import java.util.ArrayList;
/*
Min Heap Implementation using an ArrayList
*/

public class MinHeap {

    private ArrayList<HNode> data;

    private boolean isMin;//Could be used as Max Heap as well

    //Constructor1
    public MinHeap() {...}
    //Constructor2
    public MinHeap(ArrayList<HNode> arr) { this.data = arr; }
    //Constructor3
    public MinHeap(boolean isMin) {...}
    //Constructor4
    public MinHeap(HNode[] items, boolean isMin) {...}
    //return the size of the heap
    public int size() { return this.data.size(); }
    //return if the heap is empty
    public boolean isEmpty() { return this.size() == 0; }

    //Adds a new HNode
    public void add(HNode item) {...}
    //Heapify function used to maintain the heap property
    public void upHeapify(int ci) {...}
    //Remove function
    public HNode remove() {...}
    //Works to maintain the heap condition
    private void downHeapify(int pi) {...}
    //swaps the HNode data
    private void swap(int i, int j) {...}
    //...
    private boolean isLarger(int i, int j) {...}


}
```

MinHeap.java



```java
//Basic Node implementation for the heap

public class HNode {

    //executed time
    public int executedTime;
    //Reference to the RBNode
    public RBNode rbNode;

    //Constructor
    public HNode(int executed_time){...}

    @Override
    public String toString() { return Integer.toString(executedTime); }
}
```

HNode.java



```java
public class RBNode {

    public static final RBNode nil = new RBNode(-Integer.MIN_VALUE, RBTree.COLOR.BLACK);//We always take the new added node as Black

    public int Building_ID; //Building_ID
    public int totalTime;//Total executed time of current job
    public RBNode left = nil;
    public RBNode right = nil;
    public RBNode parent = nil;
    public RBTree.COLOR color;//RED/Black color
    public HNode heapNode;//Object reference to heapNode in MinHeap

    //Constructors:
    public RBNode(int building_number, HNode heapNode){...}

    public RBNode(int building_number){...}

    public RBNode(int building_number, RBTree.COLOR color){...}

    @Override
    public String toString() { return "Key:"+this.Building_ID +",Color:"+this.color.name(); }
}
```

CONCLUSION

All the three required functions have been implemented accurately . All of them run in the required time complexities.

1. Insert = O(log n)

2. Print(key) = O(log n)

3. Print(key1, key2) = O(log(n)+S)

The insert function makes use of both the Min Heap implementation and the Red Black Tree implementation. Both of the Print Functions make use of the Red Black Tree.