Advanced Data Structures (COP5536)

Fall 2019

Project Report

Name	Rachna Ahirwar
UFID	5929-9581
Email-id	rachna.ahirwar@ufl.edu

Introduction:

The aim of the project is to implement a software for a construction company, Wayne Enterprise. The software keeps track of the information regarding all the buildings which includes building number, total number of days spent on the building so far, and total number of days required to complete the construction of the building. The information is stored in form of triplet. The software is implemented by using minheap and red black tree. The minheap stores the building information triplet ordered by executed time. The red black tree stores the building information triplet ordered by building number. The input is taken as a text file from user through CLI which contains commands to work on a building, print information of buildings. The input is taken according to the global counter implemented.

Steps to compile and run:

1. Unzip the folder Ahirwar_Rachna

2. Open the folder named risingCity

3. Execute the make command

```
>> make
```

   Following will be generated:

```
storm:80% make
javac -g building.java
javac -g minheap.java
javac -g RBT.java
javac -g risingCity.java
```

4. Run the program by using command:

```
>> java risingCity <input_file>
```

   Example:

```
>> java risingCity Sample_input1.txt
```

Program Structure:

1. Class building.java

- **public class building**

```
{
    int buildingNum;    // Building number – type integer
    int executed time;  // Total number of days spent so far on the building – type integer
    int total_time;     // Total number of days required to complete the construction – type integer
}
```

2. Class RBT.java

- **class Node**

```
{
    building b;

    Node left,right,p;
    int color;
}
```

Methods

- **public int getNodeColor (Node rbnode)**

```
{
    // Returns the node color
    // Parameters - Node node
    // Return type - int
}
```

- **public void setNodeColor (Node rbnode, int color)**

```
{
    // Sets the color of a node - red/ black
    // Parameters - Node rbnode, int color
    // Return type - void
}
```

- **public Node insertRBT (Node nodeRoot, Node new_node)**

```
{
    // Inserts a new node
    If the same building number exists, it returns exception.
    If building number is less than the root, calls - insertBST(nodeRoot.left,new_node);
```

If building number is more than the root, calls- **insertBST(nodeRoot.right,new_node);**
// Parameters - Node rbnode, new_node
// Return type - Node
}

- **public void insertValue (Node rbnode)**

{
// Calls **insertBST** and **fixInsertRBTree**
// Parameters - Node rbnode
// Return type - void
}

- **public void rotateLeft (Node rbnode)**

{
// Performs the left rotation on the input node
// Parameters - Node rbnode
// Return type - void
}

- **public void rotateRight (Node rbnode)**

{
// Performs the right rotation on the input node
// Parameters - Node rbnode
// Return type - void
}

- **public void fixInsertRBTree (Node rbnode)**

{
// Fix the red black tree on node insertion
// Parameters - Node rbnode
// Return type - void
}

- **public void fixDeleteRBTree (Node rbnode)**

{
// Fix the red black tree on node deletion
// Parameters - Node rbnode
// Return type - void
}

- **public Node deleteRBT (Node nodeRoot, int buildingNum)**

{
    // Deletes a building number from the tree
    // Parameters - Node nodeRoot, int buildingNum
    // Return type - void
}

- **public void deleteValue (int buildingNum)**

  {
      // Calls **deleteBST** and **fixDeleteRBTree**
      // Parameters - int buildingNum
      // Return type - void
  }

- **public Node minValueNode (Node rbnode)**

  {
      // Returns the node with the minimum value
      // Parameters - Node rbnode
      // Return type - Node
  }

- **public Node maxValueNode (Node rbnode)**

  {
      // Returns the node with the maximum value
      // Parameters - Node rbnode
      // Return type - Node
  }

- **public int getBlackHeight (Node rbnode)**

  {
      // Returns the height of black nodes
      // Parameters - Node rbrnode
      // Return type - int
  }

- **public Node find (int buildingNum,Node rbnode)**

  {
      // Finds the given input Node
      // Parameters - Node rbnode
      // Return type - Node
  }

- **public void printBuilding (int buildingNum)**

```
{
    // Prints the building information as follows for a given building number as an input:
    // (building number, executed time, total time for execution)
    // If a building number does not exist, it prints (0,0,0)
    // Parameters - int buildingNum
    // Return type – void
}
```

- **public String printBuilding (Node rbnode,int buildingNum1, int buildingNum2)**

```
{
    // Prints all the buildings having value between buildingNum1 and buildingNum2
    // inclusive
    // Parameters - Node node, int buildingNum1, int buildingNum2
    // Return type - String
}
```

3. Class minheap.java

Methods

- **void insert (building b)**

```
{
    // Inserts a building in the heap
    // Parameters - building b
    // Return type - void
}
```

- **building removetop ()**

```
{
    // Remove the root element, then move the last element of heap to root and heapify the tree
    // Parameters - []
    // Return type - building
}
```

- **building seetop ()**

```
{
    // Returns the top element of heap without removing it
    // Parameters - []
    // Return type - building
}
```

- **void UPheapify ()**

```
{
    // Heapifies from bottom to top
    // Parameters - []
    // Return type - void
}
```

- **void Downheapify ()**

```
 {
    // Heapifies from top to bottom
    // Parameters - []
    // Return type - void
 }
```

- **void execute (building b,Integer rem)**

```
 {
    // Works on the building and update the execution time
    // Removes it from the heap if execution time = total time
    // Parameters - building b, Integer rem
    // Return type - void
 }
```

4. Class risingCity.java

Methods

- **public static void increaseGlobalTimer (List<Integer> li, minheap heap, RBT red_black)**

```
{
    // Increases the global timer and calls the execute function if first element of input = counter
    // Arguments - List<Integer> li , minheap heap , RBT red_black
    // Return type - void
}
```

- **public static void execute (List<Integer> li, minheap heap, RBT red_black)**

```
{
    // Inserts new building in the heap and rbt if it is a new value
    // Arguments - List<Integer> li , minheap heap , RBT red_black
    // Return type - void
}
```

- **private static List<Integer> funParse(String line)**

```
{
```

// Parses the input file and checks whether we have to Insert/Print
// Gets the counter value, building number, execution time
// Arguments – String line
// Return type - List<Integer>
}

- **public static void main (String[] args)**
{
// Main function
// Arguments – String[] args
// Return type - void
}

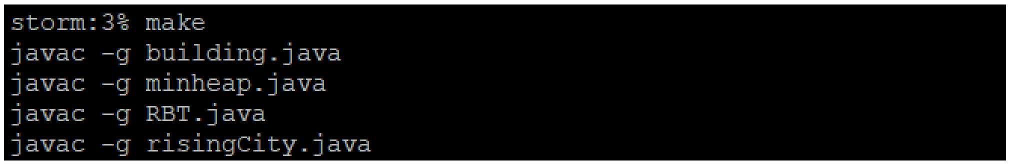
Variables:

- reader = new BufferedReader (new FileReader(fileName)) => initializes new buffered reader
- line = reader.readLine() => line reader
- li=funParse(line) => parsing a line
- RBT red_black=new RBT() => initializing new RBT
- minheap heap = new minheap() => initializing new minheap
- building remember=new building(0,0,0) => new building
- boolean flag=false
- int track=0 => To keep track of the building execution

The main function takes the input file name as an argument. If valid file is not given in the input, it throws an exception. Also, if no argument is given, it throws an exception. There is an output file “output_file.txt” and all the output is redirected to that file. BufferReader is initialized to read from the input file. The global counter runs which decides at what time the work on a building starts. The executed time for each building is initialized with zero. Whenever an input is read from the file, it checks if the first value in the input file line equals the global counter value. If it is equal, it checks whether the command is to insert or print. In insert, if the building already exists, it starts execution on that building number, else inserts a new building. If print for a building, it prints the building execution information. If the print is between two buildings, it prints the information of all the building numbers between those two buildings. It starts the execution on the building number which it reads from the file. If not, it increments the global counter. At a time, it works or five days on a building till it is not more than the total execution time. If the work is done for five days and the total execution time is more than the executed time, it puts back the building in the minheap. Also, it updates the execution time into the red black tree. When the executed time equals total execution time, the building is removed from the minheap and the red black tree. It the company is free to work, takes the next building to work on from the minheap and starts executing. Is the minheap is not having any element, it exits. This runs in a loop until the program exits, when all the work on all the building finishes.

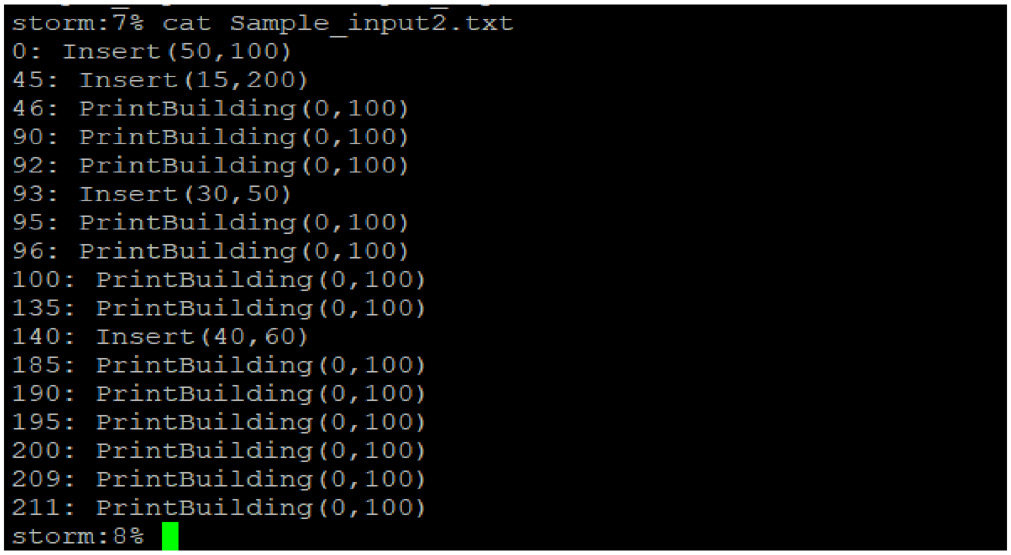
Results:

1. **Make**



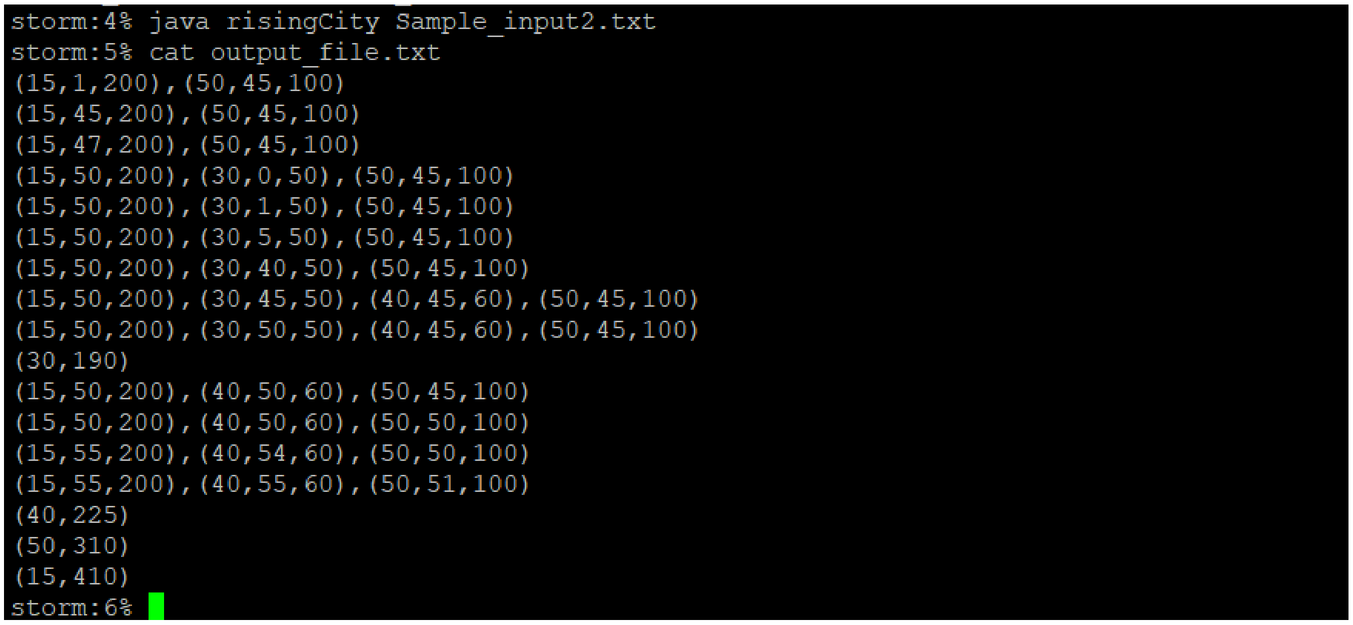
```
storm:3% make
javac -g building.java
javac -g minheap.java
javac -g RBT.java
javac -g risingCity.java
```

2. **Input file:**



```
storm:7% cat Sample_input2.txt
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)
storm:8%
```

3. **java risingCity Sample_input2.txt**



```
storm:4% java risingCity Sample_input2.txt
storm:5% cat output_file.txt
(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)
storm:6%
```