COP 5536 Spring 2018

Programming Project:

Implementation of Job Scheduler
using
Min heap and Red Black Tree

REPORT

Anurag Gupta
UFID 9152-4969
anuraggupta@ufl.edu

Project Description:
Implemented a Job Scheduler using data structures - min heap and red black tree. Used executedTime as the key for Min heap and jobID as the key for Red Black Tree.

Programming Environment Used:
Java

How to Compile and Run:
Type below commands in the terminal after changing your directory to project directory:

```
make
java jobscheduler inputfile.txt
```

Structure of the program and Function Description:

There are 4 classes in this programming project:

RedBlackImplementation.java- As the name suggest, this class is used to implement Red Black Tree and uses jobID as the key.
This class has the following functions to operate on a Red Black Tree:

```
public void insert(int v): Inserts a new node to the Red Black Tree.

public void print(): It prints elements of a red black tree.

public java.lang.Integer next(int i): It returns the next element
of the red black tree.

public java.lang.Integer prev(int i): It returns the previous
element of the red black tree.

private void fixTree(Node node): When a node is inserted to or deleted
from Red Black Tree, a fixTree operation is performed in order to
balance the tree. This is achieved with the help of rotateLeft() or
rotateRight() operations.

private void rotateRight(Node node): This performs a right rotation
operation on the Red Black Tree.

private void rotateLeft(Node node): This performs a left rotation
operation on the Red Black Tree.

private static boolean isLeftChild(Node node): Checks whether a node is
left child or not.
```

private static boolean isRightChild(Node node):Checks whether a node is right child or not.

private static boolean isRed(Node uncle): Checks whether an uncle is Red or not.

private static Node getUnclenode(Node node): This return the uncle of a node.

MinHeapImplementation.java- Similarly, this class is used to implement min heap and uses executedTime as the key.
This class has the following functions to operate on a Min Heap:

private int parent(int pos): This function takes in the position of the current node and returns the position of the parent of the current node.

private int leftChild(int pos): This function takes in the position of the current node and returns the position of the left child of the current node.

private int rightChild(int pos): This function takes in the position of the current node and returns the position of the right child of the current node.

private boolean isLeaf(int pos): Checks whether a given node is a leaf or not.

private void swap(int a, int b): Swaps two nodes in the heap which is used in the minHeapify() function.

public int remove(): It removes a node from the heap.

public void insert(int element): inserts an element into the heap.

private void minHeapify(int pos): This operation is performed when an insertion or deletion operation is done on heap to rebalance the heap when the min heap property is violated.

public void MinHeapImplementation(): Forms a min heap by calling minHeapify function.

Job.java- This class consists of several set & get methods and also uses a constructor to initialize startseektime. Its functions are:

public int getJobID(): This function returns jobID.

public void setJobID(int jobID): This function sets the jobID.

public int getTotalTime(): This function returns total time.

public void setTotalTime(int totalTime): This function sets the total time.

public int getExecutedTime(): This function returns executed time.

public void setExecutedTime(int executedTime): This function sets the executiontime.

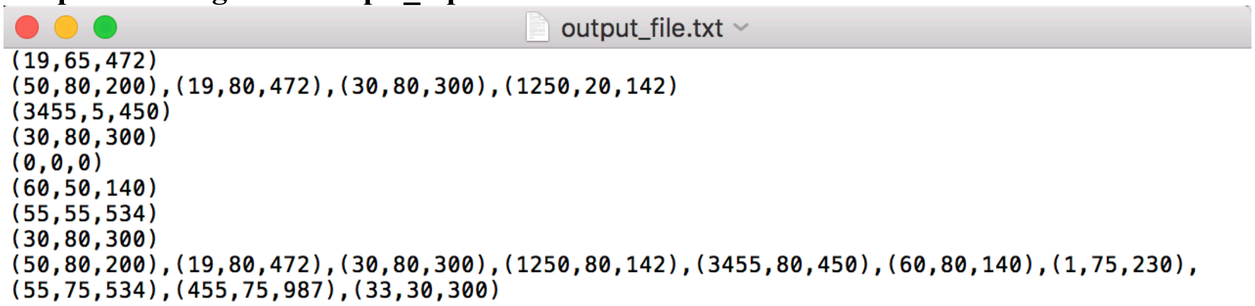
jobscheduler.java-

This is the main class. This class takes a text file as an input.

public static void main(String[] args):

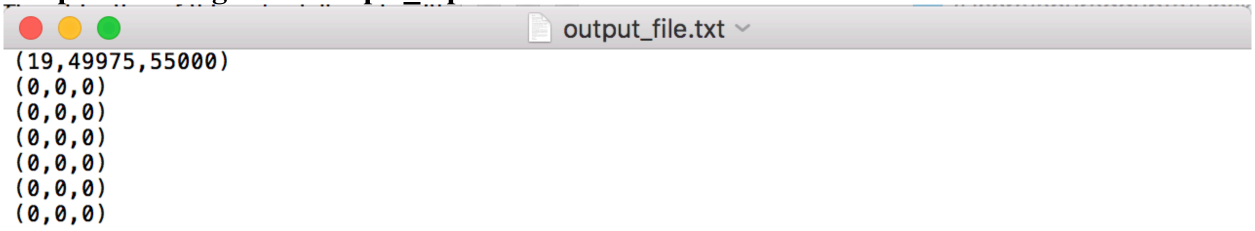
It reads the input file and convert the string in the input file and assigns it to suitable variables.
The jobscheduler assigns to it a job that has been run for the least amount of time so far. This job will run for the smaller of 5ms and the amount of remaining time it needs to complete. In case the job does not complete in 5ms it becomes a candidate for the next scheduling round.

Output for the given “sample_input1.txt” file:



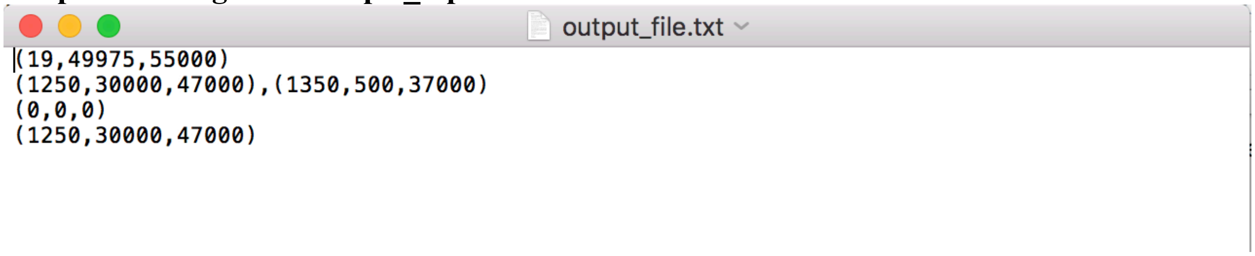
```
(19,65,472)
(50,80,200),(19,80,472),(30,80,300),(1250,20,142)
(3455,5,450)
(30,80,300)
(0,0,0)
(60,50,140)
(55,55,534)
(30,80,300)
(50,80,200),(19,80,472),(30,80,300),(1250,80,142),(3455,80,450),(60,80,140),(1,75,230),
(55,75,534),(455,75,987),(33,30,300)
```

Output for the given “sample_input2.txt” file:



```
(19,49975,55000)
(0,0,0)
(0,0,0)
(0,0,0)
(0,0,0)
(0,0,0)
(0,0,0)
```

Output for the given “sample_input3.txt” file:



```
(19,49975,55000)
(1250,30000,47000),(1350,500,37000)
(0,0,0)
(1250,30000,47000)
```