

COP 5536 Spring 2018 Programming Project

Name-Mehar Singh

EXECUTION FORMAT

The program files are compatible to run on any java environment- both Linux and windows.

There are 3 java files and corresponding class files

1. Jobheap.java
2. Rbtree.java
3. jobscheduler.java

Compile- `javac jobscheduler.java`

Run-`java jobscheduler`

DESCRIPTION OF CODE STRUCTURE

PROGRAM FLOW

Jobheap is the program that maintains minimum heap structure to store the executed time for the jobs. **Rbtree** is the program that stores the job fields- where Jobid is the key of the tree.

JobScheduler is the controller class that implements job scheduling and dispatches job to the processor. The program takes user input of input_file which contains the commands to be executed. String array "al" stores the parsed user commands-(Arrival time, command, Job fields). Objects of Jobheap and Rbtree class are created. The input string is sent as an argument to the **command(string)** method which compares the character values and then decides upon the action to be taken-(Insert job, Print job, Print next, Print previous, Print jobs in a given range). Depending on the value returned from the command function, strings are added to the "al" array. Variable **totalRuntime** keeps track of the total job time of all the inserted jobs. **systemTime** variable is the global time counter, loop runs until systemTime is less than total time of all the inserted jobs. PreviousJob keeps track of the current executing job. The **commands[]** array stores parsed string from "al". If system time is equal to arrival time of job commands[1] is checked to determine the action to be taken. if(commands[1].equals("insert")): Job is inserted into the heap and the Rbtree.

if(commands[1].equals("single")): Job fields of the given Jobid are printed
if(commands[1].equals("range")): Job fields of the job with job id in the given range are printed.

if(commands[1].equals("next")): job fields of the next job with smallest id greater than given id printed

if(commands[1].equals("prev")): Job fields of the previous job with greatest job id smaller than id printed.

Each time a job is executed, it is removed from the minheap (extractmin) and the rbtree and after execution again reinserted into the structures-heap. insertKey(previousJob
rbtree.insert(previousJob);

FUNCTION PROTOTYPES

Jobheap.java

- **private void MinHeapify(int idx)**- function to heapify job. Compare the executed time of job of the index with its left and right child's executed time and deciding the minimum node
- **public Job extractMin()**-Function to extract the minimum Root node with minimum execution time
- **private void decrease(int idx,int val)**-Function to decrease the value of "idx" to "val"
- **public void deleteKey(int idx)**-first decrease the execution time, then compare and swap with parents untill the node becomes the root node. Extract the node from the root with extractMin()
- **private void increaseSize()**-Function to increase the size of jobArray array . new capacity is twice the capacity of the old array. Copy the contents of the old array using newHeapArray[]
- **public void insertKey(Job job)**-Function to insert new values in the heap. Comparing the values of the newly inserted node with the parent value and swapping untill parent.executed_time<child.executed_time

jobscheduler.java

- **private static int command(String string)**- function to read the input string and extract the command to be executed
- **private static String getText(String string)**- function to split input string and store the integer input values of (JOB Id to insert, total time, range of job ids to print next, previous or the job with given id) if two integer inputs like insert or print range commands then values seperated by " "
- Other implementations of the program takes place in main function of the class.

Rbtree.java

- **public Node findNode(Node findNode, Node node):** Searches for the node with the job id of the given node by comparing the values with left and right children
- **public void insert(Job job):** Inserts the given node in the rbtree. Transfers the control to fixTree function.
- **private void fixTree(Node node):** Takes the newly inserted node as the input parameter and then fixes the color and pointers of the node.
- **void rotateLeft(Node node):** Performs the Left Rotation of the subtree with input node as the root node.
- **void rotateRight(Node node):** Performs the Left Rotation of the subtree with input node as the root node.
- **void deleteTree():** Deletes the entire Rbtree
- **void transplant(Node target, Node with):** Function called from delete function to replace the the target node value with the new value
- **boolean delete(Job jb):** Deletes the argument job from the tree.
- **void deleteFixup(Node x):** After deletion of the job fixes up the color and the pointers of the nodes. Transfers control to rotateLeft and rotateRight functions
- **Node treeMinimum(Node subTreeRoot):** Returns the minimum of the subtree passed as argument.
- **Job findUtil(int node):** Calls the findNode function to search for a node.
- **boolean hasNext(Node node):** Check if the node has left or right children
- **Node inorderSuccessor(Node root, Node curr):** Gets the inorder successor of the node
- **Node minValue(Node node):** Returns the minimum of the subtree passed as Node argument.

- **public Node findGreaterNode(Node findNode, Node node)**-Finds the node with jobid just greater to the given node.
- **Node nextNode(int job)**-calls findGreaterNode to determine the next node in sequence of executed time. Returns the node of the rbtree.
- **Job next(int job)**- calls findGreaterNode to determine the next node in sequence of executed time. Returns the job object of that node of the rbtree.
- **Node inorderPredecessor(Node root,Node curr)**-gets the inorder predecessor of the node.
- **Node findSmallerNode(Node findNode,Node node)**-Find the node with node id just smaller than the given node.
- **Job prev(int job)**-Find the previous job of the given job.Calls the findSmallerNode function.

EXECUTION RESULTS

```


D:\ufl sem8\ads\project_not_final\Job 2.0\sample_input3.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
sample_input3.txt x output.txt x JobScheduler.java Jobheap.java Rbtree.java
1 0: Insert(50,60000)
2 49950: Insert(19,55000)
3 99950: Insert(30,58000)
4 125900: PrintJob(19)
5 199500: Insert(1250,47000)
6 229500: Insert(1350,37000)
7 230000: PrintJob(30,5200)
8 235000: NextJob(1350)
9 236000: PreviousJob(1350)

```

Fig 1. Input file

D:\ufl sem8\ads\project_not_final\Job 2.0\output_file.txt - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help



The screenshot shows a Sublime Text editor window with the title bar 'D:\ufl sem8\ads\project_not_final\Job 2.0\output_file.txt - Sublime Text (UNREGISTERED)'. The menu bar includes 'File', 'Edit', 'Selection', 'Find', 'View', 'Goto', 'Tools', 'Project', 'Preferences', and 'Help'. The editor has four tabs: 'sample_input3.txt', 'output_file.txt', '.Jobheap.java', and 'Rbtree.java'. The 'output_file.txt' tab is active, displaying the following text:

```
1 (19 49975 55000)
2 (1250 30000 47000),(1350 500 37000)
3 (0 0 0)
4 (1250 30000 47000)
5
```

Fig2. Output file