**COP 5536 – Advanced Data Structures**

**SPRING 2023  
  
Programming Project**

**GATOR TAXI**

**PROJECT REPORT**

**Name: Adithya Koti Narasimhachar Guruachar**

**UFID: 9342-2539**

**Email:** [**adithyakotinaras@ufl.edu**](mailto:adithyakotinaras@ufl.edu)

**Problem statement:**

GatorTaxi is an up-and-coming ride-sharing service. They get many ride requests every day and are planning to develop new software to keep track of their pending ride requests.

A ride is identified by the following triplet:

rideNumber: unique integer identifier for each ride.

rideCost: The estimated cost(in integer dollars) for the ride. tripDuration: the total time (in integer minutes) needed to get from pickup to destination.

**The needed operations are:**

1. **Print(rideNumber)** prints the triplet (rideNumber, rideCost, tripDuration).

2. **Print(rideNumber1, rideNumber2)** prints all triplets (rx, rideCost, tripDuration) for which rideNumber1 <= rx <= rideNumber2.

3. **Insert (rideNumber, rideCost, tripDuration)** where rideNumber differs from existing ride numbers.

4. **GetNextRide()** When this function is invoked, the ride with the lowest rideCost (ties are broken by selecting the ride with the lowest tripDuration) is output. This ride is then deleted from the data structure.

5. **CancelRide(rideNumber)** deletes the triplet (rideNumber, rideCost, tripDuration) from the data structures, can be ignored if an entry for rideNumber doesn’t exist.

6. **UpdateTrip(rideNumber, new\_tripDuration)** where the rider wishes to change the destination, in this case,

a) if the new\_tripDuration <= existing tripDuration, update the ride with the new trip duration.

b) if the existing\_tripDuration < new\_tripDuration <= 2\*(existing tripDuration), the driver will cancel the existing ride and a new ride request would be created with a penalty of 10 on existing rideCost . We update the entry in the data structure with (rideNumber, rideCost+10,new\_tripDuration)

c) if the new\_tripDuration > 2\*(existing tripDuration), the ride would be automatically declined and the ride would be removed from the data structure.

**Project Structure:**

The project consists of five classes.

1. minHeapNode.java

2. RBTNode.java

3. MinHeap.java

4. RBT.java

5. gatorTaxi.java

**Implementation**:

To implement gatorTaxi we make use of a min heap and red-black tree (RBT). The min heap priorities rides with lower ride cost. If two or more rides have the same ride cost, then the ride with minimum trip duration is given more priority. The RBT is implemented with ride number as the search key.

To perform an insert operation, we create a minheap node and a RBT node, then establish a link between them by storing the pointer to the minheap in the corresponding RBT node and vice versa. This is required to perform operations such as getNextRide() and cancelRide(rideNumber).

Operation on rideNumber such as Print(rideNumber), Print(rideNumber1, printNumber2) are done on the RBT tree while the other operations are done by modifying both the RBT tree and min heap.

For performing a remove operation at arbitrary location in min heap we store the index in the min heap node and after removing we heapify in the upward or downward direction. This is done to ensure that the time complexity of the cancelRide() is O(log(N)).  
  
The input file name is read as a command line argument and we loop across the each statement in the input file, perform the required operation and write the result to the output file named as output\_file.txt.

**Class Definitions and function prototypes:**

**1. minHeapNode.java**

This class is used to create objects for gatorTaxi and to be pushed into min heap. It contains ride number, ride cost, trip duration, index, and pointer to corresponding RBT Node.

**Instance variables:**

* public int rideNumber – Every ride has a unique ride number and is stored in this variable.
* public int rideCost - Ride cost of a ride is stored in this variable.
* public int tripDuration - The trip duration is stored in this variable.
* public int index - The index of the current node in the minheap array is stored in this variable.
* public RBTNode rbtPointer - This is a pointer to the corresponding node in RBT data structure.

**Instance methods:**

* public String toString() - This methods overrides default toString method of Object class and returns the all the instance variables with primitive data types as a string concatenated by a space.

**2. RBTNode.java**

This class is used to create objects for gatorTaxi similar to minHeapNode and is used for pushing into RBT. It contains ride number, ride cost, trip duration and pointer to corresponding minheap node.

**Instance variables**:

* public int rideNumber - Every ride has a unique ride number and is stored in this variable.
* public int rideCost - Ride cost of a ride is stored in this variable.
* public int tripDuration - The trip duration is stored in this variable.
* public int index - The index of the current node in the minheap array is stored in this variable.
* public minHeapNode heapNode - This is a pointer to the corresponding node in RBT data structure.

**Instance methods:**

* public String toString() - This methods overrides default toString method of Object class and returns the all the instance variables with primitive data types as a string concatenated by a space.

**3. MinHeap.java**

This class contains the implementation of min heap. The min heap is ordered by minimum ride cost and if there are multiple rides with the same ride cost, then the min heap is ordered by trip duration.

**Instance variables:**

* public minHeapNode minHeap[] – Array for storing minHeapNodes.
* public int size – Integer which stores number of node present in the minHeap.

**Instance methods:**

* public minHeapNode getParentNode(int index) – Method that takes the index of the minHeapNode as the input and returns its parent node.
* public minHeapNode getLeftChildNode(int index) – Method that takes the index of the minHeapNode and returns its left child.
* public minHeapNode getRightChildNode(int index) – Method that takes the index of the minHeapNode and returns it’s right child.
* public int getParentIndex(int index) – Method that takes the index of the minHeapNode and returns the index of the parent.
* public int getLeftChildIndex(int index) – Method that takes the index of the minHeapNode and returns the index of the left child
* public int getRightChildIndex(int index) – Method that takes the index of the minHeapNode and returns the index of the right child
* public boolean isNodeALeaf(int index) – Method that takes the index of the minHeapNode and returns if the node at that index is a leaf node or not a leaf node.
* public boolean hasLeftChild(int index) – Method that takes the index of the minHeapNode and checks if the node at this index has a left child.
* public boolean hasRightChild(int index) – Method that takes the index of the minHeapNode and checks if the node at this index has a right child
* public void swapNodes(int index1, int index2) – Utility function that is used to swap two nodes present at indexes index1 and index2.
* public void insert(minHeapNode newNode) – Method to insert a new minHeapNode into the min heap.
* public void heapifyUp(int currentIndex) – Method to heapify the min heap in the upward direction form the node at currentIndex
* public void heapifyDown(int index) – Method to heapify the min heap in the downward direction from the node at index
* public minHeapNode removeMin() – Method used to remove and return the root of the min heap.
* public void removeNode(minHeapNode node) – Method used to remove the node passed as a parameter from the min heap.

**4. RBT.java**

This class contains the implementation of Red Black Tree. The search key in RBT is the ride number.

**Class variables:**

* static RBTNode externalNode – External node of the RBT

**Instance variables:**

* RBTNode root – Root node of the RBT

**Instance methods:**

* public void insert(RBTNode newNode) – Method to insert a new RBTNode inside the red black tree.
* public void restoreRbtOnInsert(RBTNode node) – Method to restore the RBT properties after insertion of a new node.
* public void rotateRight(RBTNode node) – Method to rotate the given node to the right.
* public void rotateLeft(RBTNode node) – Method to rotate the given node to the left.
* public RBTNode getUncleNode(RBTNode parent) – Method to get the uncle node of the given parent node
* public RBTNode inOrderSuccessor(RBTNode node) – Method to get the in-order successor of a given node.
* public void deleteRBTNode(int rideNumber) – Method to delete a RBTNode with the given ride number.
* public void restoreRbtOnDelete(RBTNode fNode) – Method to restore the RBT properties after a node is deleted. The node which violates the properties of the RBT is passed as a parameter.
* public boolean checkRideNumber(int rideNumber) – Method to check whether there exists any node in RBT with the given ride number.
* public RBTNode getNodeFromRideNumber(int rideNumber) – Method to get the node from RBT with the given ride number.

**5. gatorTaxi.java**

This is the class with the main method and it contains the code for taking input from the file, performing insert, delete and update operations on both the min heap and rbt and outputting the results to the file.

**Class variables:**

* static int printCount – Print count stores the already printed rides in print operation. It is used to separate multiple rides by comma.

**Class methods**:

* public static boolean **insert**(int rideNumber, int rideCost, int tripDuration, MinHeap mHeap, RBT rbt) –

This method implements **the Insert (rideNumber, rideCost, tripDuration)** requirement.

This method is used for insertion into min heap and RBT. First, it’s checks whether a ride is already present in the RBT and if it’s not present then it inserts a new Node with given node details into both min Heap and RBT. The time complexity is O(log(N)) for checking if the ride already exists and O(log(N)) for insertion into min heap and O(log(N)) for insertion into RBT. So, the total time complexity is **O(log(N))**. On successful insertion it returns true, or else false.

* public static String **printRideNumber**(int rideNumber, RBT rbt) –

This method implements **Print(rideNumber)** requirement.

This method is used for returning the node with the given ride number. The RBTNode from the given ride number is obtained in the complexity order of height of the RBT which is **O(log(N))**. If the node with the given ride number doesn’t exist it returns (0,0,0).

* public static int printRideNumberWithInRange(int min, int max,RBTNode rbt, FileWriter outputWriter) –

This method implements **Print(rideNumber1, rideNumber2)** requirement.

This method is used for printing nodes whose ride number ranges in between min and max both included. We perform an inorder traversal recursively in the RBT and print the nodes that are in the required range.

The time complexity of this operation is **O(log(N) + S)**. O(log(N)) to find the node with ride number greater than or equal to the minimum ride number and S for getting next S successors.

This function returns the number of nodes within the range along with printing the node details.

* public static String getNextRide(MinHeap mHeap, RBT rbt) –

This method implements **GetNextRide()** requirement.

This method checks if there are any active rides present from the min heap and if it’s present then performs removeMin from min heap and deleteRBTNode from RBT tree and returns the node details. If there are no active rides, then it returns “No active ride requests”.

The time complexity of removing minimum element from min heap is O(log(N)) and removing the corresponding node from the RBT also take O(log(N)) time. So, the overall complexity if O(log(N)).

* public static void cancelRide(int rideNumber, MinHeap mHeap, RBT rbt)

This method implements **CancelRide(rideNumber)** requirement.

This method checks if there is a node with given ride number in the RBT. If yes, then removes this node from the RBT. Using the heapNode pointer present in the RBT node, we remove the corresponding node from the min heap and perform heapify.  
The complexity of checking and removing a node from RBT is O(log(N)) and removing a node from min heap is O(log(N)). We store the index of the min heap node, to achieve O(log(N)) complexity for removal of an arbitrary node. So, the overall complexity of cancelRide is **O(log(N)).**

* public static void updateTripDuration(int rideNumber, int modifiedTripDuration, RBT rbt, MinHeap mHeap)

This method implements **UpdateTrip(rideNumber,newTripDuration)** requirement.

This method is used for updating the trip duration of a ride with the given ride number. First we check and get the RBT node from the RBT tree with the given ride number. This takes O(log(N)) complexity.

Next, if the current trip duration is greater than or equal to the new modified trip duration, then we simply update the trip duration of the node in both RBT and minheap and heapify min heap if required.

If the existing trip duration is less than the new modified trip Duration and the modified trip Duration is less than or equal to 2 times the existing trip duration then cancel the ride and create a new ride with cost penalty of 10. This takes the time complexity of O(log(N)). In all other conditions we simply cancel the ride. This takes the time complexity of O(log(N)). So the overall cost of this operation is **O(log(N)).**

All operations mentioned in the problem statement are implemented in O(log(N)) complexity.