ADS Programming Project 1 - GatorTaxi ride-sharing service

Arshdeep Kaur

79001937

Arshdeep.kaur@ufl.edu

Problem Statement:

We want to build a ride sharing service named gatorTaxi that can help track the pending ride requests. Each ride is identified by a triplet – rideNumber(unique identifier for each ride), rideCost(estimated cost for the ride in integer dollars) and tripDuration(total time to get from source to destination in integer minutes).

We want the service to be able perform some operations as mentioned below:

1. **Print (rideNumber)** – this function should print the ride details
2. **Print (rideNumber1, rideNumber2)** – this function should print all rides details whose ride numbers lies between the given rideNumbers
3. **Insert(rideNumber)** – it allows to insert a new ride into the system and if that rideNumber already exists, we stop the program execution
4. **getNextRide()** – this function allows to get the next ride with the lowest cost, or if the cost is same, then the lowest tripDuration
5. **CancelRide(rideNumber)** – this function allows to cancel a ride and delete these ride details from the service
6. **UpdateTrip(rideNumber,new_tripduration)** – this function allows to update the ride if new duration is within the range of current duration and twice the current duration with a cost penalty of 10$. If the new trip duration is higher than that, the trip is cancelled/ride deleted.

Program Structure:

The program has been built using the encapsulation and abstraction concepts of Object Oriented programming. Separate classes have been created for Ride, Node, Node color, Min heap, RedBlack Tree and the main project implementation. The project structure is discussed below:

Classes:

Ride – To identify a ride with a triplet which are represented by class variables:

- rideNumber (int)
- rideCost (int)
- tripDuration (int)

Node – This class is created to identify a single node of a redblack tree.

- rideNumber(int)
- ride(Ride) – points to a Ride
- leftChild(Node) – points to left child of this node
- rightChild(Node) – points to right child of this node
- parent(Node) – points to the parent of this node
- color (NodeColor) – represents the color of the Node i.e. BLACK/RED

NodeColor: This is an enum class to identify the color of a node. It has 2 constants : RED, BLACK

MinHeap:

MinHeap stores the all the rides in an array list following min heap properties ordered by RideCost. If two rides have same cost, they are further prioritized based on the tripDuration. The class variables for minheap include:

- ArrayList<Ride> heapArray - List of all ordered Rides
- Map<Integer, Integer> - Maps the rideNumber to the index at which that ride is stored in the heapArray. This makes it easier to build heap again when a ride is updated or deleted. It also gives an easy access to check if a ride is present of not.
- maxSize (int) – this variable refers to the maximum size of the heap which is 2000 given in requirement.

Apart from the class variables, there are some class functions and helper functions that are used to check if heap is empty and to insert or remove ride from heap and to get the next minimum ride. We will discuss these functions in detail in next section.

RedBlackTree:

Red-Black tree is a self-balancing binary search tree in which each node contains an extra bit for denoting the color of the node, either red or black. This class represents the redBlack Tree data structure, where the root points to the root of this tree which is a Node. The Node will have the value and the pointers to other children, parent and node's color.

Class variables:

- Root root (Node) – Points to root of a tree.

It supports functions to check the node's color, if the tree is empty, and to search is ride from its number or in between a range of numbers, to insert and deleted a node in the red black tree. The node further is linked to the ride itself. It consists of various other helper function which help in insertion and deletion of nodes.

gatorTaxiRide:

This is the class which has pointers to both the minheap and red black trees and implements the logic for all the functions as mentioned in the requirement.

Class variables:

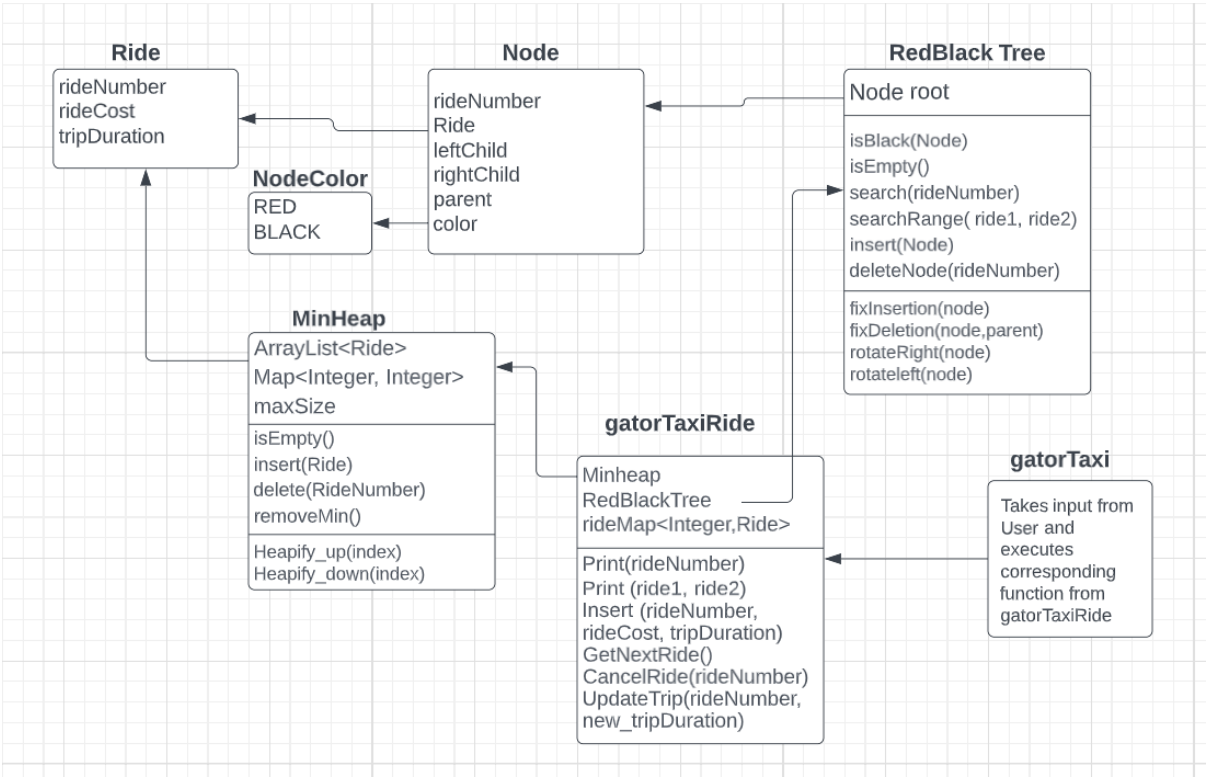
- Minheap (minHeap)
- redBlacktree(RedBlack Tree)
- rideMap<Integer,Ride> - maps ride number to its Ride.

The functions will be discussed in detail in the next section of function prototypes.

gatorTaxi:

This is the main class which takes in inputs from the users and calls the gatorTaxiRide functions based on the input command from the user.

The class diagram is shown as below:



Function Prototypes and Complexity:

Class MinHeap:

1. **boolean isEmpty() –**

It returns True if size of heap is 0, else returns False

Time Complexity – O(1)

2. **void insert(Ride ride)**

This function adds the new ride at end of the array and calls heapify_up function which builds the heap from bottom to up to ensure the heap properties are met, going up from child to parent till it reaches the root. This is the only path where the heap property could have been changed upon insertion.

It also adds an entry onto the map that identifies rideNumber to the index of the ride.

Time Complexity – Since the heapify_up function goes up only one path , the complexity is equal to the height of the tree = log(n). Thus, the insert function also has the same complexity = log(n)

3. **void delete(int RideNumber)**

This function uses the map to identify the index at which the Ride is stored in the array. Thus it will get the Ride given its ride number in O(1) time. Once it gets the ride, it will delete the node using the heapify_down operation. The node to be deleted will be replaced with data from last node and last node is deleted. Then, we perform heapify down starting from this node.

Time Complexity - Since the heapify_down operation goes from the index down to a leaf, fixing heap properties, its complexity if equal to the height of that node. The complexity will be worst if the node deleted is root = O(log n). If the node deleted is a leaf, it will be done in O(1) time. The average time complexity if referred to as O(log n).

4. **Node removeMin()**

This function just returns the value present at the 0^{th} index of the heap i.e the next ride. We are not performing node deletion in this function, but calling the delete function specifically if we will need to delete the min node.

Time Complexity – Since the data access in an arraylist is constant time, the complexity of this function is O(1).

Class RedBlackTree:

1. Boolean isBlack(Node)-

This function returns true if the node is Black or null, and false otherwise.

Time Complexity – O(1) – constant time to compare color.

2. Boolean isEmpty()

Returns true if root is null else returns false.

Time Complexity – O(1)

3. Node search(Int rideNumber)

This function iterates through the tree to find the node with the same rideNumber, going to the right subtree if rideNumber is greater or to left subtree is rideNumber is smaller. It only traverses one particular path until it finds the ridenumber.

Time Complexity – O(log n)

4. **ArrayList<Ride> search(Int rideNumber1, int rideNumber2)**

Similar to the previous function it traverses the redblack tree in an inorder traversal fashion and ignores any subtree greater than rideNumber2 or smaller than rideNumber1 returning the list of the rides in this range.

Time Complexity – O(log n)+S (where S is the number of rides in that range)

5. **void insert(Node)**

While inserting a new node, the new node is inserted as a RED node. After insertion of a new node, we call the **fixInsertion()** method which checks if there is any red-red violation such that the newly inserted red node's parent is also a red node. This is fixed using recoloring and required rotations**(leftRotate(),rightRotate())** to regain the red-black tree properties.

Time Complexity – O(log n)

6. **deleteNode(int rideNumber)**

Deleting a node may or may not disrupt the red-black properties of a red-black tree. If the deleted node is black, the number of black nodes in each path of read-black tree may not remain same anymore. A fixing algorithm is used to regain the red-black properties which is implemented in the **fixDeletion()** method and necessary recoloring of nodes and rotations**(leftRotate(),rightRotate())** are performed.

Time Complexity – O(log n)

Class gatorTaxiRide:

This class implements all the main functions that the gatorTaxi service needs by performing necessary checks and calling the functions of heap and red-black tree simultaneously.

1. void Print(int rideNumber)

This function calls the search function of the red black tree and prints the ride. The time complexity is same as the search function i.e – O(log n)

2. void Print(int rideNumber1, int rideNumber2)

This function calls the searchRange function of the red black tree and prints the rides returned in the list of rides. The time complexity is same as the searchRange function i.e – O(log n)+S

3. void Insert (int rideNumber, int rideCost, int tripDuration)

This function checks if ride exits using the map in O(1) time and exits after printing “Duplicate RideNumber” . If the ride is not present, it inserts the ride in minheap and the redblack tree and updates the map to store the new ride. The time complexity of this function is same as the insert function of heap and red black tree i.e. O(log n).

4. Ride GetNextRide()

This function calls the getMin function of the minheap and gets the next ride in O(1) time. It these deletes the ride from the minheap, redblack tree and the map. The time complexity of this function is same as the delete function of heap and red black tree i.e. O(log n).

5. void CancelRide(int rideNumber)

This function deletes the ride from the minheap , redblack tree and the map. The time complexity of this function is same as the delete function of heap and red black tree i.e. O(log n).

6. void UpdateTrip(int rideNumber, int new_tripDuration)

- If the new_tripDuration is less than the existing trip duration, we get the ride from the map and just update it’s new trip duration in O(1) time complexity. No other operation is needed.
- Else If the new_tripDuration is less than twice the existing trip duration, we get the ride from the map and delete it from the heap and the red black tree and insert the ride again with new cost with penalty of 10 and new duration. The node is reinserted because just updating the cost and duration would have resulting in violating the heap property. Another method is to call heapify_up/heapify_down on the updated ride. The time complexity of this function is same as the delete/insert function of heap and red black tree i.e. O(log n).
- Finally, if the new duration of trip is even greater than twice the current trip duration, we just delete the ride from the heap, redblack tree and it’s references from the map. It is same as the CancelRide function, which can also be called here. The time complexity of this function is same as the delete function of heap and red black tree i.e. O(log n).

Class gatorTaxi :

This is the main class which takes the input arguments as the name of the input file and reads the commands from the input file using the BufferedReader.

Based on the input files commands, it extracts the function name and the argument and calls the corresponding function from the gatorTaxiRide class.

It writes any output print statement into an output text file and uses Buffered Output and PrintStream to do so.

Steps to run the file:

Unzip the project and add it to the directory. The ***make*** command will create an executable gatorTaxi class file.

```
thunder:~/ADSProj> ls -l
total 41
-rw-rw-r-- 1 arshdeep.kaur grad   606 Apr 11 21:44 ABC.txt
-rw-rw-r-- 1 arshdeep.kaur grad 33091 Apr 11 21:27 gatorTaxi.java
-rw-rw-r-- 1 arshdeep.kaur grad  1503 Apr  4 11:04 input.txt
-rw-rw-r-- 1 arshdeep.kaur grad   207 Apr 11 21:41 Makefile
-rw-rw-r-- 1 arshdeep.kaur grad   385 Apr  9 12:41 sample_input.txt
thunder:~/ADSProj> make
rm -rf *.class
rm -rf gatorTaxi output*
javac gatorTaxi.java
```

2. The command ***java gatorTaxi input_file_name.txt*** will run the function and write the output the the output_file.txt.

```
thunder:~/ADSProj> java gatorTaxi input.txt
thunder:~/ADSProj> cat output_file.txt
(0,0,0)
(3,20,40)
(1,10,20),(3,20,40),(4,30,60),(5,50,120),(6,35,70)
(1,10,20)
(3,20,22)
(6,55,95)
(6,55,95),(7,40,90),(8,45,100),(9,55,110)
(4,30,60)
(0,0,0)
(8,45,100)
(12,80,200)
(6,55,95)
(9,55,110)
(11,80,210),(13,70,220)
(10,60,150)
(0,0,0)
(11,80,210),(13,70,220),(15,20,35)
(15,20,35)
(13,70,30)
(13,70,30)
(0,0,0)
(17,70,25)
(11,80,210)
(16,70,60)
(0,0,0)
```

```
(0,0,0)
Duplicate RideNumber
thunder:~/ADSProj> ls -l
total 64
-rw-rw-r-- 1 arshdeep.kaur grad   606 Apr 11 21:44 ABC.txt
-rw-r--r-- 1 arshdeep.kaur grad  3244 Apr 11 22:03 gatorTaxi.class
-rw-rw-r-- 1 arshdeep.kaur grad 33091 Apr 11 21:27 gatorTaxi.java
-rw-r--r-- 1 arshdeep.kaur grad  3159 Apr 11 22:03 GatorTaxiRide.class
-rw-rw-r-- 1 arshdeep.kaur grad  1503 Apr  4 11:04 input.txt
-rw-rw-r-- 1 arshdeep.kaur grad   207 Apr 11 21:41 Makefile
-rw-r--r-- 1 arshdeep.kaur grad  2497 Apr 11 22:03 MinHeap.class
-rw-r--r-- 1 arshdeep.kaur grad   497 Apr 11 22:03 Node.class
-rw-r--r-- 1 arshdeep.kaur grad   777 Apr 11 22:03 NodeColor.class
-rw-r--r-- 1 arshdeep.kaur grad   569 Apr 11 22:04 output_file.txt
-rw-r--r-- 1 arshdeep.kaur grad  3901 Apr 11 22:03 RedBlackTree.class
-rw-r--r-- 1 arshdeep.kaur grad   324 Apr 11 22:03 Ride.class
-rw-rw-r-- 1 arshdeep.kaur grad   385 Apr  9 12:41 sample_input.txt
thunder:~/ADSProj>
```

3. Running the command ***make clean*** will delete any output or class files created at the end.