Class definitions with function prototypes:

RideNode.java –

A java class for storing ride details.

Properties:

- **rideNumber** (int) – Unique integer identifier for each ride
- **rideCost** (int) – The estimated cost for the ride
- **tripDuration** (int) – total time needed to go to the destination from pickup.

Methods:

Contains getters and setters for the above-mentioned properties.

MinHeapNode.java –

A java class for representing a node in the MinHeap. It extends the RideNode class.

Properties:

- correspondingRBTNode(RBTNode) – contains the reference to the corresponding RBT node.

Methods:

Contains getters and setters for the correspondingRBTNode.

RBTNode.java –

A java class for representing a node in the Red-Black Tree. It extends the RideNode class.

Properties:

- **correspondingMinHeapNodeIndex**(int) – contains the index of the corresponding minHeapNode in the Heap.
- **leftChild**(RBTNode) – contains the reference of the left child of the node.
- **rightChild**(RBTNode) – contains the reference of the right child of the node.
- **parent**(RBTNode) – contains the reference of the parent of the node.
- **isRed**(boolean) – represents the color of the node, true if red, false otherwise.

Methods:

Contains getters and setters for the above-mentioned properties.

MinHeap.java –

A java class for Implementing the Min Heap with MinHeapNode as its nodes.

Properties:

- **heap**(MinHeapNode[]) – It is an array that represents the heap and is used to store the MinHeap nodes.
- **size**(int) – Integer to keep track of the size of the heap.

Methods:

- void **insert**(MinHeapNode) – Inserts the given node into the heap at the appropriate position maintaining the Min Heap property.
- MinHeapNode **arbitraryRemove**(int) – Deletes a node from the heap and calls the heapify method. It returns the deleted MinHeapNode.
- MinHeapNode **removeMin**() – Deletes the first element in the heap and calls heapify method. Returns the first MinHeapNode.
- void **heapify**(int) – It starts with the index given and recursively travels down the heap to place the given node in the appropriate index to satisfy Min Heap properties.
- void **updateTrip**(int) – It checks if the node at the given index is in appropriate position or not and performs heapifyUp or heapifyDown based on the scenario.
- Boolean **compare**(MinHeapNode, MinHeapNode) – It compares the 2 given min heap nodes to determines which ride should be served first. It returns true if the first node should be served first.

RBT.java –

A java class for implementing a Red-Black Tree using RBTNode as its node and contains all the methods for performing various required operations on the Red-Black tree.

Properties:

- **root**(RBTNode) – It is used to point to the root of the tree.

Methods:

- void **insert**(RBTNode) – Inserts the given node into the Red-Black tree, checks if the tree is balanced or not, if not calls, rebalanceRBTAfterInsert method.
- void **rebalanceRBTAfterInsert**(RBTNode, RBTNode, RBTNode) – Determines the type of rebalancing by classifying the type of imbalance(XYr) from the input provided needed and calls the appropriate rotation method to fix the properties.
- RBTNode **delete**(RBTNode, MinHeap) – Deletes the given node from the Red-Black tree by calling deleteAsIfItIsAStandardBST method, and then calls the rebalanceRBTAfterDelete method if rebalancing is needed. It returns the deleted node.
- RBTNode **deleteAsIfItIsAStandardBSt**(RBTNode, MinHeap) – This method is used for deleting the the given node from the tree. It returns null if no rebalancing is necessary or returns the sibling of the actually deleted node if rebalancing is necessary.
- void **rebalanceRBTAfterDelete**(RBTNode, RBTNode) – This method rebalances the Red-Black tree if a black node is deleted by classifying the type of imbalance(Xcn) calling the appropriate rotation method to fix the properties.

- void **rotateLL**(RBTNode, RBTNode, RBTNode) – Takes the 3 given nodes and performs the LL rotation.
- void **rotateLR**(RBTNode, RBTNode, RBTNode) – Takes the 3 given nodes and performs the LR rotation.
- void **rotateRL**(RBTNode, RBTNode, RBTNode) – Takes the 3 given nodes and performs the RL rotation.
- void **rotateRR**(RBTNode, RBTNode, RBTNode) – Takes the 3 given nodes and performs the RR rotation.
- void **specialRotationLR**(RBTNode, RBTNode, RBTNode, RBTNode) – This rotation is used to perform the LR rotation under different circumstances.
- void **specialRotationRL**(RBTNode, RBTNode, RBTNode, RBTNode) – This rotation is used to perform the RL rotation under different circumstances.
- int **getNumberOfRedChildren**(RBTNode) – this method returns the number of red children a given node has.
- RBTNode **getOtherChild**(RBTNode, RBTNode) – This method returns the sibling of the given node. Returns null if it doesn't exist.
- RBTNode **search**(int) – This method searches for the node, given the rideNumber and returns it.
- void **printRange**(int, int, StringBuilder) – Calls the recursive method printRange by passing root as the starting point.
- void **printRange**(RBTNode, int, int, StringBuilder) - Searches for all the nodes in the specified range of rideNumbers and appends to the given StringBuilder.

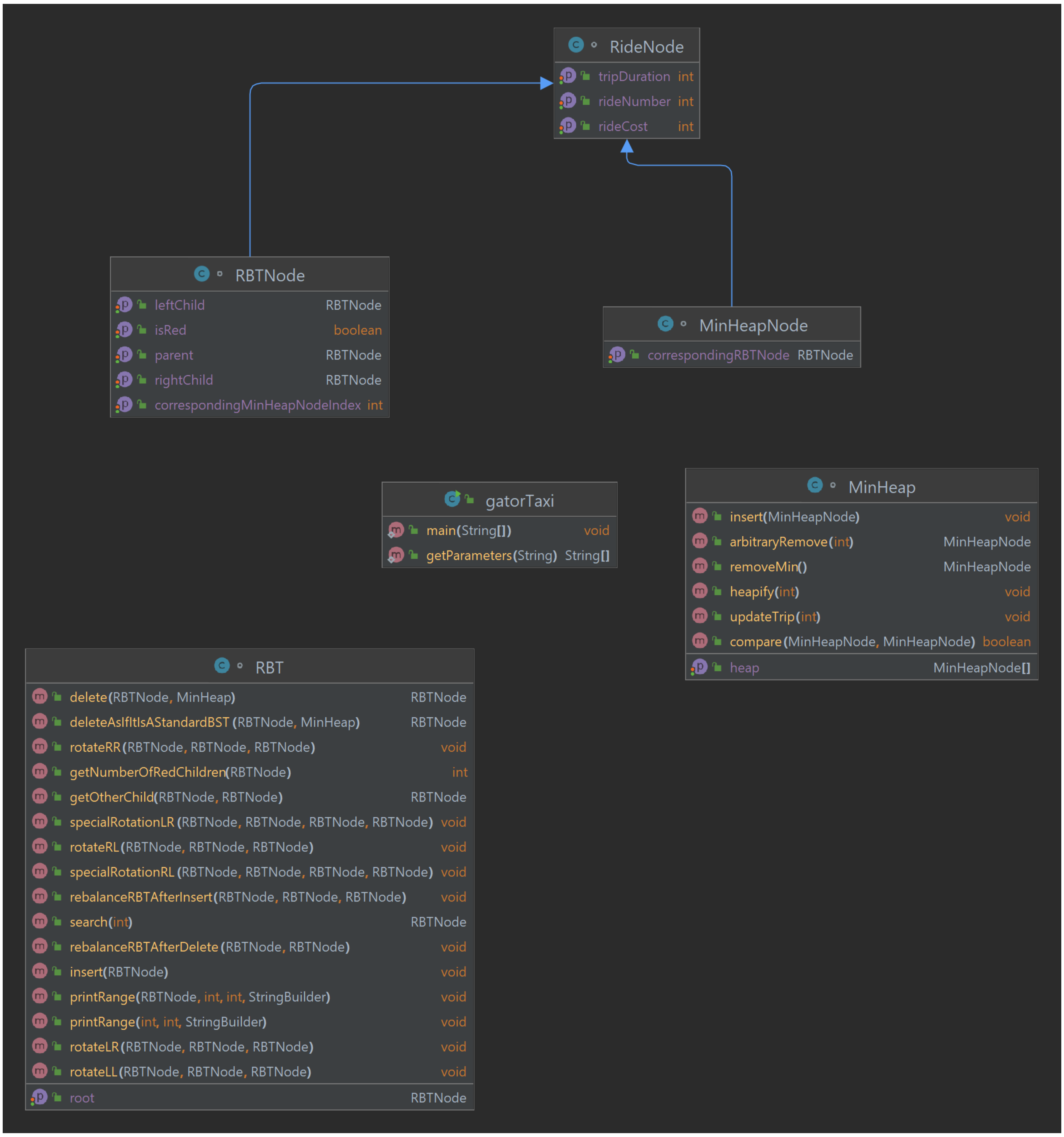
gatorTaxi.java –

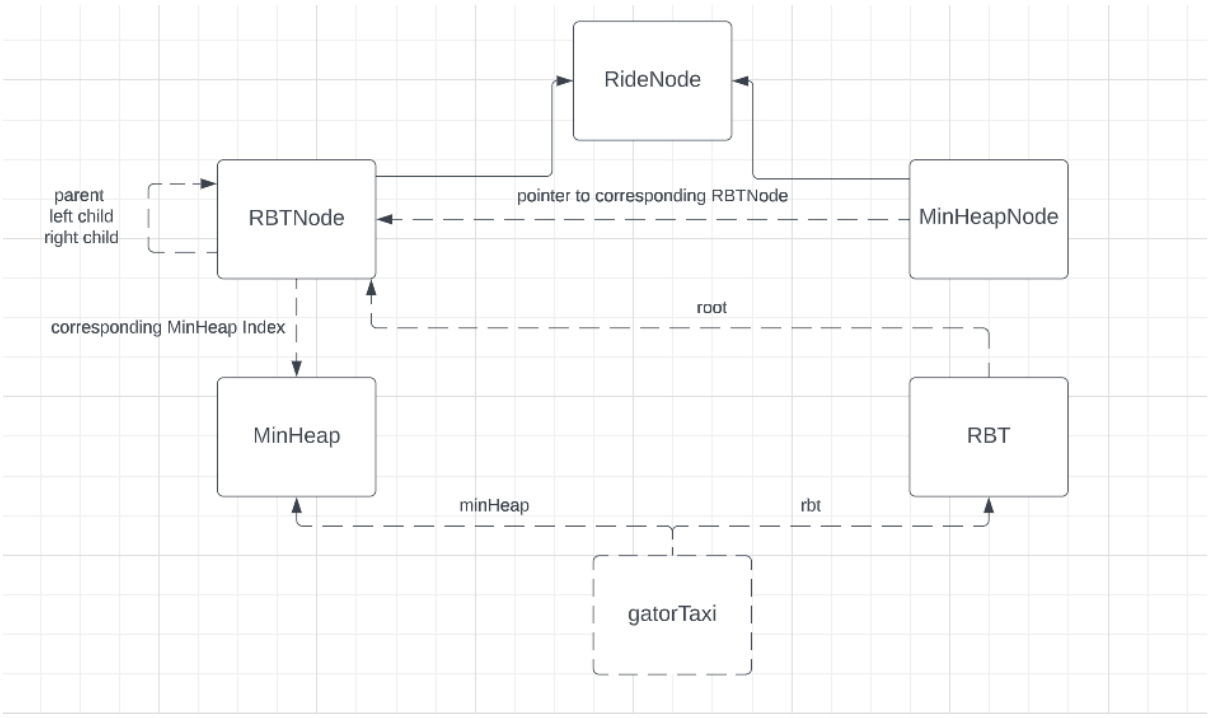
This is the class containing the main method and controls the main workflow of the program.

Methods:

String[] **getParemeters**(String) – this function returns the parameters given in the input as a string array.

Program Structure:


RideNode
tripDuration int
rideNumber int
rideCost int
RBTNode
leftChild RBTNode
isRed boolean
parent RBTNode
rightChild RBTNode
correspondingMinHeapNodeIndex int
MinHeapNode
correspondingRBTNode RBTNode
gatorTaxi
main(String[]) void
getParameters(String) String[]
MinHeap
insert(MinHeapNode) void
arbitraryRemove(int) MinHeapNode
removeMin() MinHeapNode
heapify(int) void
updateTrip(int) void
compare(MinHeapNode, MinHeapNode) boolean
heap MinHeapNode[]
RBT
delete(RBTNode, MinHeap) RBTNode
deleteAsIfItIsAStandardBST (RBTNode, MinHeap) RBTNode
rotateRR(RBTNode, RBTNode, RBTNode) void
getNumberOfRedChildren(RBTNode) int
getOtherChild(RBTNode, RBTNode) RBTNode
specialRotationLR (RBTNode, RBTNode, RBTNode, RBTNode) void
rotateRL(RBTNode, RBTNode, RBTNode) void
specialRotationRL (RBTNode, RBTNode, RBTNode, RBTNode) void
rebalanceRBTAfterInsert(RBTNode, RBTNode, RBTNode) void
search(int) RBTNode
rebalanceRBTAfterDelete (RBTNode, RBTNode) void
insert(RBTNode) void
printRange(RBTNode, int, int, StringBuilder) void
printRange(int, int, StringBuilder) void
rotateLR(RBTNode, RBTNode, RBTNode) void
rotateLL(RBTNode, RBTNode, RBTNode) void
root RBTNode



Time Complexities:

Insert – O(logn) – as maximum height of RBT is 2*log(n + 1), and that of a Min Heap is log(n), searching and inserting can be done in logarithmic time.

Print(rideNumber) – O(logn) – Search for the rideNumber takes O(logn) time.

Print(rideNumber1, rideNumber2) – O(log(n) + S) - where S is the number of nodes with ridenumbers in the given range. As we are only entering the subtrees that we are interested in, i.e., if it contains any number between rideNumber1 and rideNumber2, we are only visiting the interested 'S' nodes. So, this only takes logarithmic time.

CancelRide()– O(logn) – This takes O(logn) time as delete operation takes O(logn) in both RBT and Min Heap.

UpdateTrip()– O(logn) – This takes O(logn) time, as it only involves a deletion in both data structures only once in the worst case.

GetNextRide() – O(logn) – This takes O(logn) as heapify and delete operations take O(logn) time in the worst case.

Program flow:

Get a line from the input file, stop the program if end of file is reached

↓

Parse the input

↓

In case of Insert, search if the ride number already exists, stop the program, else create MinHeapNode and RBTNode, set corresponding pointers, and call insert method of respective data structures.

In case of Print, check the number of parameters and call the printRange method in RBT if there are two parameters, else search for the node and print the tuple.

In case of UpdateTrip, check the updateTrip conditions, and call the appropriate methods for either updating or deleting.

In case of CancelRide, search for the node in RBT, perform arbitrary delete in MinHeap and delete in RBT.

In case of GetNextRide, call removeMin method on MinHeap, get the corresponding RBTNode and delete in RBT.

↓

Flush and close the bufferedWriter, close the bufferedWriter

↓

End program