Programming Project

Name: Sheel Taskar
UFID: 2753-4463
Email: sheel.taskar@ufl.edu

Program Structure

Classes

1. **Ride:**
   - This class represents an object of Gator Taxi. It contains the tuple rideNumber, rideCost, tripDuration along with pointer and color integer for Red Black Tree and index pointer for Heap.
   - It contains setter methods for setting the values of class variables.
   - Space complexity of the object is O(1) as it just has variables and pointers and no data structures.

2. **MinHeap:**
   - This class defines the blueprint of min Heap. It uses ArrayList for storing the objects of class Ride.
   - It contains methods for all the operations required in the Heap.
   - MinHeap's is based on primary key as rideCost and secondary key as tripDuration.
   - Space complexity of the MinHeap is O(n).

3. **RbTree:**
   - This class defines the blueprint of RbTree.
   - It contains methods for all the operations required in the Heap.
   - RbTree is based on primary key as rideNumber.
   - Space Complexity of RbTree is O(n).

4. **gatorTaxi:**
    - This class contains the main method. It creates instances of RbTree and MinHeap for implementing the assignment.
    - It contains logic for handling I/O operations.
    - Entire program takes O(n) space to store internal data structures.
    - Additional disk space is needed to store the output file.

Functions

Class: Ride

1. **Ride():**
    - Construction to initialize default value for variables.
    - Time complexity is O(1).

2. **Ride(int rideNumber, int rideCost, int tripDuration)**
    - Parameterized Constructor to intialize class variables with passed values.
    - Time complexity is O(1).

3. **setIndex(int heapIndex)**
    - Setter method for changing heapIndex.
    - Time complexity is O(1).

4. **setRideCost(int rideCost)**
    - Setter method for changing rideCost.
    - Time complexity is O(1).

5. **setTripDuration(int tripDuration)**
    - Setter method for changing tripDuration.
    - Time complexity is O(1).

Class: MinHeap

6. **MinHeap():**
    - Constructor to initialize ArrayList.
    - Time complexity is O(1).

7. **int size()**
    - Returns the size of Heap.
    - Time complexity is O(1).

8. **void insert(Ride r)**
    - Inserts the Ride object in the Heap.
    - Adds the Ride at the end of the list, then performs heapify, by comparing the rideCost of the parent.
    - Also has the tie breaking condition, when rideCost is same, it checks trip duration.
    - Time complexity is O(log(n)) as it inserts in worst time, equal to height of the heap.

9. **void decreaseDuration(int rideIndex, int tripDuration)**
    - Decreases the Duration of tripDuration.
    - This is called when trip duration is decreased when updateRide() method is called.
    - It is required because the minHeap's secondary key is tripDuration. Hence when duration is changes, we need to check the parent's trip duration.
    - In worst case, all the keys have same rideCost, ahe tripDuration is different.
    - Time complexity is O(log(n)), equal to height of the heap in worst case.

10. **Ride removeMin()**
    - Removes the smallest element in the heap which is at the top.
    - Takes the last elements, and puts at the top.
    - Performs heapify on 1st element.
    - Worst case time complexity is O(log(n)), equal to height of the heap.

11. **void heapify(int i)**
    - Heapify the element at index i.
    - Checks the left and right child rideCost.
    - Compares current index and left child cost, update smallest pointer
    - Compares current index and right child cost, update smallest pointer
    - Swaps the element if smallest is not index and again does heapify on smallest
    - Worst case time complexity is O(log(n)), equal to height of the heap.

12. **void remove(int heapIndex)**

- Arbitrary remove in min Heap, argument is heapIndex.
- Set the cost of the Ride at heapIndex as minimum cost-1.
- We get minimum cost form cost of min Element of the reap.
- We bubble the element at heapIndex to the top.
- Then performs removeMin.
- Time complexity is log(n) for bubbling up + log(n) for removeMin.
- Worst case time complexity is O(log(n)), equal to height of the heap.

13. **int right(int i)**
    - Return index of right child 2*i+2
    - Time complexity is O(1).

14. **int left(int i)**
    - Return index of right child 2*i+1
    - Time complexity is O(1).

15. **int parent(int i)**
    - Return index of parent i/2 or i-½
    - Time complexity is O(1).

16. **void swap(int i, int parent)**
    - Swap the objects at index i and parent.
    - While swapping, we also update heapIndex.
    - Time complexity is O(1).

Class: RbTree

17. **Ride searchHelper(Ride ride, int key)**
    - Helper method for search.
    - Search is similar to search in Binary Search Tree.
    - Go left or right based on values, return Ride if found.
    - Worst case time complexity is O(log(n)), equal to height of the tree.

18. **void deleteFix(Ride x)**
    - Balances tree after deletion
    - Perform the operation until x is not root or x's color is 0.
    - Use the rotateLeft and rotateRight Function for left and right rotation according to required condition.

- Worst case time complexity is O(log(n)), equal to height of the tree.

19. **void Transform(Ride u, Ride v)**
    - RB Transplant for ride u and ride v.
    - If parent is null, sets root to v.
    - Else check u with left of parent and set it as v.
    - Else right of parent is v.

20. **void deleteRideHelper(Ride ride, int key)**
    - Helper class for deletion.
    - Remove is similar to normal binary search tree.
    - After deletion, check the colors and conditions for rb transplant and call Transform function.
    - If color is 0, then only perform DeleteFix.

21. **void insertFix(Ride k)**
    - Balances the tree after insertion.
    - Similar to deleIteFix function.
    - Use the rotateLeft and rotateRight Function for left and right rotation according to required condition.
    - Worst case time complexity is O(log(n)), equal to height of the tree.

22. **RbTree()**
    - Constructor to initialize tree with default values.
    - Time Complexity is O(1).

23. **String getRides(int rideNumber1, int rideNumber2)**
    - Return ride from rideNumber1 to rideNumber2.
    - Calls printBetweenRides Helper method.

24. **String printBetweenRides(int rideNumber1, int rideNumber2, Ride ride)**
    - Helper Method for getRides.
    - Does the inorder traversal from rideNumber1 to rideNumber2.
    - Goes left till null found, right the left.
    - Then returns node and the then goes right till the null. Returns right.
    - This is performed recursively.

25. **Ride searchTree(int k)**
    - Calls the searchHelper method.

26. **Ride minimum(Ride ride)**
    - Return the minimum rideNumber object.
    - Goes left of the tree to find the minimum.
    - Worst case time complexity is O(log(n)), equal to height of the tree.

27. **Ride maximum(Ride ride)**
    - Return the maximum rideNumber object.
    - Goes right of the tree to find the minimum.
    - Worst case time complexity is O(log(n)), equal to height of the tree.

28. **void rotateLeft(Ride x)**
    - Performs left rotation.
    - Checks left, right and parents left and right.
    - Apply the left rotation.

29. **void rotateRight(Ride x)**
    - Performs right rotation.
    - Checks left, right and parents left and right.
    - Apply the right rotation.

30. **void insert(Ride ride)**
    - Sets parent of the ride as null.
    - Sets the left and right child of ride as NULL node.
    - Sets color as 1.
    - Performs insert operations as BST.
    - Checks the parents and grandparents
    - After insert calls the insertFix method on the inserted node.
    - Worst case time complexity is O(log(n)), equal to height of the tree.

31. **Ride getRoot()**
    - Returns the root element.
    - Time Complexity is O(1).

32. **void deleteRide(int data)**
    - Calls the deleteRide Helper method.

Class: gatorTaxi

33. **String insert(int rideNumber,int rideCost,int tripDuration,MinHeap minHeap,RbTree rbt)**
    - Creates object of Ride with values rideNumber, rideCost and tripDuration.
    - Calls insert method of Minheap and Rbtree.
    - Hence, time complexity is O(log(n)).

34. **String print(int rideNumber, RbTree rbt)**
    - Prints the tuple of ride with given rideNumber.
    - Calls the search method in RbTree.
    - Hence, time complexity is O(log(n)).

35. **String print(int rideNumber1, int rideNumber2, RbTree rbt)**
    - Prints the tuple of all the rides between rideNumber1 and rideNumber2.
    - Calls the getRide Method in Rbtree.
    - Returns comma separated tuples from rideNumber1 to rideNumber2.

36. **String getNextRide(MinHeap minHeap,RbTree rbt)**
    - Prints the tuple of ride with minimum cost and tripDuration.
    - Calls removeMin operation on minHeap.
    - Calls deleteRide operation on RbTree.
    - Hence, time complexity is O(log(n)).

37. **void updateTrip(int rideNumber, int tripDuration, MinHeap minHeap, RbTree rbt)**
    - Executes the three conditions for UpdateTrip.
    - If new tripDuration is less, then calls decreaseDuration on MinHeap.
    - If it is less than equal to 2* current tripDuration, calls deleteRide on Rbtree and remove on MinHeap, calls insert method of this class with new ride with ride cost +10.
    - If new TripDuration is > 2* current, we delete ride from RbTree and MinHeap.
    - Hence, time complexity is O(log(n)).

38. **void cancelRide(int rideNumber, MinHeap minHeap, RbTree rbt)**
    - Calls remove operation on minHeap for arbitrary remove.
    - Calls deleteRide operation on RbTree.
    - Hence, time complexity is O(log(n)).

39. **public static void main(String arg[])**
    - Main method for the program.

- Initializes MinHeap and RbTree.
- Reads the input file and parses the commands.
- Executes the commands and writes it in the output file

Steps to run the Program

1. Unzip the folder.
2. run make
3. java gatorTaxi <filename.txt>

Input file: <any_name_allowed>
Output file: output_file.txt