ADS - Spring 2023

Programming Project

Name: Bhakti Armish Kantariya
UFID: 3928 1182
Email: b.kantariya@ufl.edu

Problem:

GatorTaxi is a new application that handles the ride sharing requests and this software is a part of it. It handles the remaining ride requests one by one which are in the pending queue and performs operations as requested by the users.

Steps to run the code:

1. Change directory to Kantariya_BhaktiArmish
2. run ***make***
   or
   run ***python3 gatorTaxi.py <input_file.txt>***

The output gets generated in the **output_file.txt**

Structure of code:

gatorTaxi.py:

It is the only file that consists the implementation for the complete project with Red Black Tree and the Min heap. It has a global map declared named ridePointers, maintaining the location of the rides in min heap.

class MinHeap():

Implements the min heap and its functionality with the following given function prototypes.

1. **def __init__(self, maxtreesize: int) -> None**

   Initializes the tree properties when an object of this class is created. It initializes the leafride flag to 0 and ROOTNODE flag to 1. Leafride gets incremented after every insertion and decremented after every deletion.

2. **def parentRide(self, loc: int) -> int**

   It locates and returns the location of the parent of the node at the location given in the parameters.

3. **def leftChild(self, loc: int) -> int**

   It locates and returns the location of the left chilf of the node at the location ***loc*** given in the parameters.

4. **def rightChild(self, loc: int) -> int**
   It locates and returns the location of the right child of the node at the location ***loc*** given in the parameters.

5. **def isLeafRideNode(self, loc: int) -> Boolean**
   It checks if the node at location ***loc*** given in the parameters is a leaf node. It returns True if it a leaf node else False is returned.

6. **def swaprides(self, floc: int, sloc: int) -> None**
   It swaps the ride node at location ***floc*** with the ride node at location ***sloc.*** It also updates the ridePointers with the new location of the rides.

7. **def minRideHeapify(self, loc: int) ->None**
   This method heapifies the tree with root as the node at location ***loc.*** It checks the ridecost of the parent ride and the child ride, if both have same ridecost, it swaps the parent ride with the child having minimum ridecost. If the parent and child rides have same ridecost, it checks the tripduration and swaps if the tripduration of child is less than the parent.

8. **def insertMHeap(self, rideNumber: int, rideCost: int, tripDuration: int) -> None**
   it inserts the new ride at the leaf of the tree and keeps swaping the new node with the parent all the way to the root node whenever it comes across the condition where parent has more ridecost than new node or both have same ridecost but parent have more tripduration.

9. **def getNextRide(self) -> Dict [int, int, int]**
   It returns the root of min heap and deletes the root node by replacing it with the leaf node and running heapify over it.

10. **def cancelRide(self, location: int) -> None**
    It deletes the node at the given location and replaces it with the leaf node. Runs heapify function over the replaced node.

11. **def updateTrip(self, location: int, newTripDuration: int) -> None**
    it updates the ride at given location based on the value of new tripduration

Class RideNode()

It defines the node structure of a node in the red black tree.

1. **def __init__(self, maxtreesize: int) -> None**

It consists of node values such as nodecolor, left and right childs, parent ride, ridenumber, ridecost and rideduration.

ADS - Spring 2023

Programming Project

class RedBlackTree():

It implements the red black tree and various functionalities associated with it that are described below.

1. **def __init__(self, minheapInstance: MinHeap) -> None**
   it initializes the red black tree with its properties when an object of clas RedBlackTree is created. It also consists of an instance of the min heap for refering while performing operations.

2. **def insertRBT(self, rideNumber: int, rideCost: int, tripDuration: int) -> None**
   It performs insertion into the tree and handles rotation and recoloring after insertion.

3. **def rotationAndRecoloring_insertRBT(self, ride: RideNode) -> None**
   It performs rotation and recoloring based on the color and position of the nodes to ensure the tree satisfies all the constraints.

4. **def l_rotation(self, ride: RideNode) -> None**
   It performs left rotation at the given RideNode.

5. **def r_rotation(self, ride: RideNode) -> None**
   It performs the right rotation at the given RideNode.

6. **def print_ride(self, rideNumberFrom: int, rideNumberTo: int) -> None**
   It takes the help of print_ride_helper to find the rides and prints the ride triplets.

7. **def print_ride_helper(self, ride: RideNode, rideNumberFrom: int, rideNumberTo: int, rideDetails: int) -> None**
   It searches the single ride or the rides in the given range.

8. **def cancel_ride(self, rideNumber: int) -> None**
   It performs the deletion of the ride from the red black tree and the min heap.

9. **def remove_ride(self, rideNumber: int) -> None**
   It performs deletion of the ride from red black tree.

10. **def cancel_ride_helper(self, ride: RideNode, rideNumber: int) -> None**
    finds the ride with the given ridenumber, deletes it and performs rebalancing.

11. **def rbRebalancing(self, x: RideNode, y: RideNode) -> None**
    It replaces the x ridenode with y ridenode.

12. **def rotationAndRecoloring_deleteRBT(self, ride: RideNode) -> None**

It performs rotation and recoloring to satisfy the constraints.

13. **def minimum(self, ride: RideNode) -> RideNode**
    It returns the left child of the given ridenode

14. **def updateTrip(self, rideNumber: int, newTripDuration: int) -> None**
    Performs update operation on the ride with given ridenumber in both the trees.

15. **def update_ride_helper(self, ride, rideNumber, newTripDuration) -> Node**
    It is helper function that checks the conditions of new tripduration and updates the ride details accordingly.

16. **def getNextRide(self) -> None**
    It removes the ride from the red black tree with the rideNumber of the root node of min heap

If __name__ == "__main__"

It is the main function that handles the input file reading and calling the operations based on the inputs. It also handles printing the results to the output file.

Complexities:

Time:

Insert trip operation – Red black tree and min heap both take O(log(n)) time for inserting a new node in tree. Rebalancing and recoloring operation after insertion in red black tree takes the time of the height of the tree and similarly the heapify operation in min heap takes time of height of tree. Therefore, it's O(log(n))

Print single trip operation – If the ride is at the leaf then it would take O(log(n)) time.

Print multiple trip operation – It also takes O(log(n)) but with addition of X rides being printed. Therefore, time taken is O(log(n)+X)

Update trip, Get next ride and cancel ride operation – all of these takes O(log(n)) as the node to be operated on can be at the leaf.

Space:

- Red black tree has the space complexity of O(n),
- Min heap also has the space complexity of O(n),

where 'n' is the number of ride nodes in the tree.