COP 5536 Advanced Data Structure

Project Report

Name: Abhinav Aryal

UFID: 3327-1507

UF-Email: abhinavaryal@ufl.edu

Programming language used: Python

1. Project Description:

Creating software to track the ride request with the help of Red-black Tree and Min Heap. It will have features like Insert, Delete, GetNextRide, UpdateTrip, CancelRide, Print. The input of the software will use are rideN-> RideNumber, rideC -> cost of the ride, tripD -> Duration of the trip.

Data Structure implemented: Red Black Tree and Min Heap

2. About the Implemented Data Structures:

Red-Black-Tree:

A Red-Black tree is a binary search tree that has benefits like being a self-balancing binary tree, so every external node to the root node will contain the same amount of black nodes. It also has features likes, all new nodes will be red nodes and no parent and child of a red node can be red. This enables us to have a height of the RBT between log(n+1) and 2log(N+1)
These properties are maintained by using the rotations:

LeftRotate: **rotateleft(node)** *O(1) ->Time complexity*

RightRotate:**rotateright(node)** *O(1)-> Time complexity*

These functions are basically there to balance the tree and this is done by swapping children of nodes and swapping the parent to the child as per the condition depending on the function. They are called by functions like **rebalanceinsert(node)** and **rebalanceDel(node)** which are called when an **insert(int, int, int)** and **deleteN(node)** respectively. These two balancing functions are making required rotations and colour changes to ensure the property of the RB Tree. This RBT helps us to update, insert, delete, and search in log(n) time. (Not if we are printing nodes in a range)

Min Heap:

A Min Heap is implemented, which will have the minimum element always in the root node and all the other properties of Min Heap. Here we have a list that has been used in order to store nodes that has values like rideN, rideC, tripD and the corresponding node in the RBT tree. And an additional dictionary has been used that is being used for index mapping in the previous list with help of the provided rideN. The properties of the Min heap are preserved with the help of heapify functions, **heapify(int)** and **revheapify(int).** They both take the index as the parameter from which the heapify process should be done. The **heapify()** function ensures that the parent node is always smaller than its children and starts from the top. While the **revheapify(int)** function is used to do the same work but it starts working from the bottom.
Both these data structures can be found in the AdvanceDS.py file.

3. Input process:

An input file will be read by the Program and when it detects the operation to be done, it will initially remove the unwanted symbols like ("\((.*?)\)") and will initiate a split operation to obtain, the individual detail information like ride number, ride cost and trip duration if it is needed. The values for example in the case of the insert will be Insert20,30,40, after removing the "(" and ")"

4. Functions implemented:

Insert:

Format for insert in Input file => Insert (rideN, rideC, tripD)

The aim of this is to insert a new node in the Red-Black tree and the Min Heap. The operation will be done in **O (logn)**

The ride details are used to call the **insertN (int, int, int)** function of the red-black tree and if the insertion is successful another insertion takes place in the min heap by calling the function **insertN(int, int, int, node)** of the minheap, which will have the ride details and the RBTnode reference.

The expected output in the output file =>

if insertion is successful:

No output expected

if insertion is of duplicate rideNr:

Write "Duplicate RideNumber" in the output file

After the Insertion is Completed in RBT: **rebalanceinsert(node)** is called in order to fix the RBT as every new node is red and two red nodes can't be kept consecutively. This function will fix the issue by making the necessary rotations mentioned above

CancelRide:

Format for CancelRide in Input file => CancelRide(rideN)

This function aims to delete a node from the trees. It is done in O(logn) time complexity.

We will have to find the node that contains the ride number and this is done by calling the function **find(int)** in the RBTtree, If there are no such rides, it will just continue with the program. Otherwise, It will process the deletion from the RBT and Min heap by using the function **deleteN(node)** in both the RBTree and Min Heap. Both will use the instance of the rbtnode to delete the node. In RBT, we will call the **balanceDel(int)** this will start the balancing process by making the necessary rotations mentioned above.
While in the min Heap, after deleting the arbitrary node, we will call the **revheapify(int)** to initiate the heapify process from that index node.
No Output is expected in any condition.

GetNextRide:

Format for GetNextRide in Input file => GetNextRide(). *No data needed to be passed. parameters*

The aim of this call is to get the Next Ride with minimum cost, and if the ride cost is the same, trip duration acts as the tiebreaker. This function will run in **O(logn).**

After we are inside the case, the function **GetMin()** will run in the min Heap and the node will be popped out before which it will be stored in a temp variable and returned to delete the node from RBT later by calling the function **deleteN(node)** in the RBT. After deleting the node with the minimum cost which will be in the root will need fixing and this will be done by the function **revheapity(int),** the passed value will be the index to initiate the heapify process which will always be 0 in case of deleting the min node.

The expected output in the output file =>

if the Tree is not empty:

Write the next ride details in the output file in the (rideN, rideC, tripD).

if the Tree is empty:

Write "No Active ride requests"

UpdateTrip:

Format for UpdateTrip in Input file => UpdateTrip(rideN, newTripD).

The aim of this call is update the ride details of a particular ride. This function will run in **O(logn).**

When we enter the case, we will have to do the same process as in delete and initially find the node with the help details provided, then the instance of the node is then provided to **Updation(node, int)** in both the RBT and the min Heap, which will have a reference to the node found after using the details and the newTripD. The **Updation(node, int)** function in both has a similar way of working.

If the new duration is more than twice the current duration we call the **deleteN(int).**

Else we update the trip duration.

In addition to updating the ride duration, we also add $10 to the rideC if the current ride duration is more than the new ride duration and less than twice the current ride duration. In min Heap as the heap is stored ordered by the rideC and tiebreaker tripD, we will have to fix the tree and run the **heapity(int), revheapify(int)** functions.

This Function is not expected to give any output.

Print:

Format for Print in Input file => Print(rideN) or Print(RideN1, rideN2)

The aim of this call is to print the ride details that have been requested to be printed.. This function will run in **O(logn)** if we are printing ride details of only one ride but takes **O(k+ logn)** if we have to run multiple rides.

When we enter the case to process print, we will check two cases:

1) **If there is only one rideN**

We will call the **Display(int, int)** both being the ride numbers in the RBT. This function will again call the **Range(list, node, int, int),** it gives an empty list, the root node and the two rideNs as the parameters. In the **Range** function, we will initially check if we have to go either the left child or the right, and this is done again and again with the help of recursion with the node as either child as per the condition and for every rideN that lies in the range will be appended in the initial empty array which will be finally printed.

2) **If there is a range provided**

Here we will simply print the ride number with the help of the node value that we obtain with the help of **find(int)** in the RBtree.

The expected output in the output file =>

If there is no such rideN or no rideN in the range:

Write "(0,0,0)"

if such node or nodes present:

Write "(rideN, rideC, tripD) * number of ride details present"

5. Implementation of provided test cases

TestCase1:

input.txt

```
Insert(25,98,46)
GetNextRide()
GetNextRide()
Insert(42,17,89)
Insert(9,76,31)
Insert(53,97,22)
GetNextRide()
Insert(68,40,51)
GetNextRide()
Print(1,100)
UpdateTrip(53,15)
Insert(96,28,82)
Insert(73,28,56)
UpdateTrip(9,88)
GetNextRide()
Print(9)
Insert(20,49,59)
Insert(62,7,10)
CancelRide(20)
Insert(25,49,46)
UpdateTrip(62,15)
GetNextRide()
Print(1,100)
Insert(53,28,19)
Print(1,100)
```

output_file.txt

```
(25, 98, 46)
No active ride requests
(42, 17, 89)
(68, 40, 51)
(9, 76, 31),(53, 97, 22)
(73, 28, 56)
(0,0,0)
(62, 17, 15)
(25, 49, 46),(53, 97, 15),(96, 28, 82)
Duplicate RideNumber
```

TestCase2:

input.txt

```
Insert(5,50,120)
Insert(4,30,60)
Insert(7,40,90)
Insert(3,20,40)
Insert(1,10,20)
Print(2)
Insert(6,35,70)
Insert(8,45,100)
Print(3)
Print(1,6)
UpdateTrip(6,75)
Insert(10,60,150)
GetNextRide()
CancelRide(5)
UpdateTrip(3,22)
Insert(9,55,110)
GetNextRide()
UpdateTrip(6,95)
Print(6)
Print(5,9)
GetNextRide()
CancelRide(7)
Print(7)
Insert(11,70,170)
GetNextRide()
Insert(12,80,200)
Print(12)
UpdateTrip(11,210)
GetNextRide()
CancelRide(14)
UpdateTrip(12,190)
Insert(13,70,220)
GetNextRide()
Insert(14,100,40)
UpdateTrip(14,100)
CancelRide(12)
Print(11,14)
GetNextRide()
Insert(15,20,35)
Print(14)
```

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)
(12, 40, 30),(16, 80, 82),(20, 16, 75)
(20, 16, 75)
(0,0,0)
(8, 60, 97)
(11, 80, 210)
(16, 90, 124)
(15, 90, 85)
(23, 49, 46)
(1, 56, 85),(7, 125, 54)
(7, 125, 54)
No active ride requests
(17, 12, 37)
(0,0,0)
Duplicate RideNumber
```