Report: GatorTaxi Ride-Share Service

Name Kushagra Sikka

UFID : 3979-0988

Introduction: GatorTaxi is a ride-sharing service that handles multiple ride requests every day. The company plans to develop new software to manage and keep track of their pending ride requests efficiently. Our task is to design and implement this software using a Red-Black Tree (RBT) and a min-heap data structure to store the ride information.

Design: I designed the software to store ride information in two data structures: RBT and min-heap. The RBT is used to store ride information ordered by ride number, while the min-heap is used to store ride information ordered by estimated cost and trip duration.

The software supports six operations:

1. Insert (rideNumber, rideCost, tripDuration)
2. Print(rideNumber)
3. Print (rideNumber1, rideNumber2)
4. UpdateTrip(rideNumber, newTripDuration)
5. GetNextRide()
6. CancelRide(rideNumber)

The RBT and min-heap data structures are maintained using pointers between corresponding nodes in each data structure.

Implementation: I implemented the software using Python and designed separate classes for RBT and min-heap data structures. I used the **enum** library for creating enumerated constants implemented in the RBT class.

For the Insert operation, we check if the rideNumber already exists in the RBT. If not, we insert the ride information into both the RBT and min-heap data structures.

For the Print operation, we search for the ride information using the rideNumber and print the triplet (rideNumber, rideCost, tripDuration) if it exists. If not, we print (0, 0, 0).

For the Print(rideNumber1, rideNumber2) operation, we traverse only the subtrees of the RBT that may possibly contain a ride in the specified range. We print all (rideNumber, rideCost, tripDuration) triplets separated by commas in a single line, including rideNumber1 and rideNumber2, if they exist. If there is no ride in the specified range, we print (0, 0, 0).

For the UpdateTrip operation, we first search for the ride information using the rideNumber. If it exists, we check the new_tripDuration and update the ride information in the min-heap and RBT accordingly.

For the GetNextRide operation, we select the ride with the lowest rideCost from the min-heap. If there are multiple rides with the same rideCost, we select the one with the lowest tripDuration. We then delete the ride information from both the RBT and min-heap.

For the CancelRide operation, we search for the ride information using the rideNumber and delete it from both the RBT and min-heap if it exists.

Conclusion: In conclusion, we successfully designed and implemented the GatorTaxi ride-sharing service software using a Red-Black Tree and min-heap data structure. The software efficiently manages and keeps track of pending ride requests while maintaining the necessary pointers between the two data structures.

Readme_file

To run the GatorTaxi program with a specific input file, use the following command in the terminal:

```
make run input_file=input.txt
```

else you can use the standard way of running it

```
python3 gatorTaxi.py <input.txt>
```

Replace `input.txt` with the name of your input file. The program will automatically generate the output file `output_file.txt` in the same directory.

The Makefile is configured to use Python 3 as the interpreter. If you have multiple versions of Python installed on your system, make sure to use the correct command to run Python 3. You may need to modify the Makefile to reflect the location of your Python 3 installation.

Testing is an important part of software development, as it helps to ensure that the code is working as expected and meeting the project requirements. For the GatorTaxi project, various test cases were created to test the functionality of the implemented code.

Some of the test cases that were performed include:

- Testing the Insert operation to ensure that ride numbers are unique and duplicate ride numbers are not inserted into the data structure.
- Testing the Print operation to verify that the correct triplet is printed when a ride number is given, and that (0, 0, 0) is printed if the ride number does not exist in the data structure.
- Testing the Print operation with a range of ride numbers to ensure that all triplets within the specified range are printed, and that (0, 0, 0) is printed if no rides exist within the specified range.
- Testing the UpdateTrip operation to verify that trip durations are updated correctly and that the ride is canceled with a penalty if the new trip duration is greater than the existing trip duration by a certain factor.
- Testing the GetNextRide operation to ensure that the ride with the lowest ride cost and trip duration is returned and deleted from the data structure.
- Testing the CancelRide operation to verify that the specified ride is deleted from the data structure and that no errors occur if the ride number does not exist.

These test cases were designed to cover different scenarios and edge cases, such as inserting rides with duplicate ride numbers, updating rides with different trip durations, and ensuring that the correct triplets are printed for different ranges of ride numbers. By performing these tests, we can ensure that the GatorTaxi program is working correctly and meeting the project requirements.

Complexity Analysis: The GatorTaxi system utilizes two main data structures to store and manage ride requests: a min-heap and a Red-Black Tree. The min-heap is used to maintain a priority queue of ride requests based on their rideCost and tripDuration, with the ride with the lowest cost and duration at the root of the heap. The Red-Black Tree, on the other hand, is used to store ride requests in a sorted order based on their rideNumber.

For each of the implemented operations, the time complexity is as follows:

1. Print(rideNumber): The time complexity of this operation is O(log(n)) as it involves searching the Red-Black Tree for the specified rideNumber. If the rideNumber exists, the triplet is printed; otherwise, (0,0,0) is printed.
2. Print(rideNumber1, rideNumber2): The time complexity of this operation is O(log(n) + S) where S is the number of triplets printed. To implement this operation, we only search the Red-Black Tree in the subtrees that may possibly have a ride in the specified range, which takes O(log(n)) time. Then, we print all the rides within the specified range, which takes O(S) time. If there are no rides in the specified range, (0,0,0) is printed.
3. Insert(rideNumber, rideCost, tripDuration): The time complexity of this operation is O(log(n)) as it involves inserting a new ride request into both the min-heap and the Red-Black Tree, both of which take O(log(n)) time.
4. GetNextRide(): The time complexity of this operation is O(log(n)) as it involves deleting the ride request with the lowest rideCost from the min-heap and updating the corresponding node in the Red-Black Tree, both of which take O(log(n)) time.
5. CancelRide(rideNumber): The time complexity of this operation is O(log(n)) as it involves deleting the ride request with the specified rideNumber from both the min-heap and the Red-Black Tree, both of which take O(log(n)) time. If there is no such ride, no action is taken.
6. UpdateTrip(rideNumber, new_tripDuration): The time complexity of this operation is O(log(n)) as it involves updating the trip duration of the ride request with the specified rideNumber in both the min-heap and the Red-Black Tree. The time complexity may increase to O(log(n)+log(k)), where k is the number of ride requests in the heap with rideCost greater than the updated ride request's cost, in the worst case where the ride request is cancelled and a new one is inserted with a penalty cost. However, this worst-case scenario is rare and occurs only when the updated trip duration is more than double the original duration.

Overall, the GatorTaxi system's time complexity is dominated by the min-heap and the Red-Black Tree, both of which have a worst-case time complexity of O(log(n)). Therefore, the system's overall time complexity for all operations is O(log(n)).

Please find attached Screenshots of test cases provided :