COP 5536
Programming project
Anirban Deb
UFID – 71056852
Email – deb.anirban@ufl.edu

Code Structure

Class MinHeapRide

The MinHeapRide class is a model class that represents a ride in a min heap data structure. It implements the Comparable interface to define the comparison logic for sorting rides in the min heap.

Data Members-

1. rideNumber (int): Represents the ride number of the ride.
2. rideCost (int): Represents the cost of the ride.
3. tripDuration (int): Represents the duration of the ride.
4. rbtRide (RedBlackTreeRide): An object of the RedBlackTreeRide class, which represents the minimum heap for rides.
5. positionInHeap (int): Represents the index of ride in the heap.

Constructor-

public MinHeapRide(int rideNumber, int rideCost, int tripDuration): Constructor to initialize the MinHeapRide object with the given ride number, ride cost, and trip duration.

Member Functions-

1. **@override public int compareTo(MinHeapRide ride):** This method compares two MinHeapRide objects based on their ride cost and trip duration and returns -1 if the current ride is smaller, 1 if the current ride is larger, and 0 if they are equal.
2. **public void print():** This method prints the ride details in the format of (rideNumber, rideCost, tripDuration).
3. **@override public String toString():** This method returns a string representation of the ride details in the format of (rideNumber, rideCost, tripDuration).

Class MinHeap

This class is an implementation of a min heap that provides methods for inserting rides into the heap, deleting the minimum ride from the heap, deleting an arbitrary ride from the heap, and heapifying the heap to maintain its properties.

Member functions-

1. **public MinHeap(int)**: Constructor for creating a MinHeap object with the specified maximum size.
2. **private int positionOfParentForNodeAtIndex(int index)**: Returns the position of the parent of a ride at a specific index.
3. **private int positionOfLeftChildForNodeAtIndex(int index)**: Returns the position of the left child (ride on the left) of a ride with a specific index.

4. **private int positionOfRightChildForNodeAtIndex(int index)**: Returns the position of the right child(ride on the right) of a ride with a specific index.
5. **private boolean isLeaf(int index)**: Returns true if the ride at the given index is a leaf node, false otherwise.
6. **private void positionSwapInHeap(int firstPosition, int secondPosition)**: Swaps the positions of two rides in the min heap.
7. **private void heapifyMinHeap(int index)**: Performs heapify on the heap starting from the ride at the given index.
8. **public void insertIntoMinHeap(MinHeapRide ride)**: Inserts a new ride into the min heap and does a bottom-up heapification.
9. **public MinHeapRide deleteMinFromMinheap()**: Deletes the minimum ride from the heap and returns it. Finally does top-down heapification to maintain heap properties.
10. **public MinHeapRide deleteArbitraryRideFromMinheap(MinHeapRide minHeapRide)**: Deletes an arbitrary ride from the heap and returns it. Also heapifies the heap to maintain properties.

- ## Class RedBlackTreeRide

The RedBlackTreeRide class is a model class for a red-black tree that represents rides. It implements the Comparable interface, which allows objects of this class to be compared based on their rideNumber property.

Data members-

1. rideNumber: An integer representing the ride number.
2. rideCost: An integer representing the cost of the ride.
3. tripDuration: An integer representing the duration of the trip in minutes.
4. minHeapRide: An object of the MinHeapRide class, which represents the minimum heap for rides.
5. parent: A reference to the parent node in the red-black tree.
6. left: A reference to the left child node in the red-black tree.
7. right: A reference to the right child node in the red-black tree.
8. colour: An integer representing the color of the node in the red-black tree.

Constructor-

public RedBlackTreeRide(int rideNumber, int rideCost, int tripDuration): Constructor for the RedBlackTreeRide class that initializes the data members rideNumber, rideCost, and tripDuration with the given values.

Member Functions-

1. **@override public int compareTo(RedBlackTreeRide ride):** Implementation of the compareTo method from the Comparable interface, which compares two RedBlackTreeRide objects based on their rideNumber property. Returns -1 if the current rideNumber is less than the given rideNumber, 1 if it is greater, and 0 if they are equal.
2. **public String print():** This method returns a string representation of the RedBlackTreeRide object in the format (rideNumber, rideCost, tripDuration).

3. **@Override public String toString():** Overrides the toString() method of the object class to return a string representation of the RedBlackTree object in the format (rideNumber, rideCost, tripDuration).

Class RedBlackTree

This class is an implementation of a red-black tree that provides methods for inserting rides into the tree, deleting rides from the tree, and also transforming the tree to maintain red-black tree properties.

Constructor-

public RedBlackTree(): Constructor to create a red-black tree node by initializing a new node and setting the left and right child to null.

Member functions-

1. **public RedBlackTreeRide searchRideNumberInTree(int rideNumber)**: This method is used to search for a ride using the ride number and return the ride.
2. **private RedBlackTreeRide searchRideNumberInTreeHelper (RedBlackTreeRide rbtRide, int rideNumber)**: This is a helper method for the searchRideNumberInTree method.
3. **public int insertIntoRbt(RedBlackTreeRide rbtRide):** This method inserts a new ride into the red-black tree.
4. **private void repositionAfterInsert(RedBlackTreeRide rbtRide):** This method repositions the rides in the red-black tree through rotations to maintain the red-black tree property.
5. **public List<String> printBetweenRide1AndRide2(int rideNumber1, int rideNumber2)**: This method returns a list of all the rides between two specific ride numbers.
6. **public void printBetweenRide1AndRide2Helper(RedBlackTreeRide rbtRide, int rideNumber1, int rideNumber2, List<String> result)**: This is a helper method for the printBetweenRide1AndRide2 method.
7. **public RedBlackTreeRide minimumRideNumber(RedBlackTreeRide rbtRide):** This method traverses the red-black tree and returns the ride with the minimum ride number.
8. **public RedBlackTreeRide removeFromRBT(int rideNumber):** This method removes the ride with the ride number specified in the argument.
9. **private RedBlackTreeRide removeMinRBTHelper(RedBlackTreeRide rbtRide, int rideNumber):** This is the helper method for removeFromRBT method.
10. **public void removeMinWithPointerRBT(RedBlackTreeRide rbtRide):** This method removes the ride in the argument which we get after removal from minheap.
11. **private void removeMinWithPointerRBTHelper(RedBlackTreeRide rideToBeRemoved):** This is the helper method for removeMinWithPointerRBT method.
12. **private void repositionAfterDelete(RedBlackTreeRide rbtRide)**: This method is executed after deletion to ensure that the red-black tree properties are maintained which is done via rotations.
13. **private void redBlackTransplant(RedBlackTreeRide rideA, RedBlackTreeRide rideB):** This method is used to interchange the position of two rides in a red-black tree to maintain the red-black tree property.

14. **public void rotationLeft(RedBlackTreeRide rbtRide)**: This method performs a left rotation on the given node rbtRide in the tree.
15. **public void rotationRight(RedBlackTreeRide rbtRide)**: This method performs a right rotation on the given node rbtRide in the tree.
16. **public void updateTrip(int rideNumber, int new_tripDuration):** This method updates the ride details based on the new trip duration. If the trip duration reduces, just the trip duration gets updated with the new duration. If the new duration increases by less than double the previous duration, the ride is canceled and again inserted into the data structures with the new duration updated, and also the cost increases by 10. If the new duration is more than double the previous duration the ride is canceled.

- ## Class gatorTaxi

This class represents the main driver code that runs the taxi service where rides are added, deleted, updated and managed. Its done using a combination of red-black tree and min heap.

Data Members-

1. **rbTree:** A static instance of RedBlackTree class, which represents a red-black tree used to store the ride requests.
2. **heap:** A static instance of MinHeap class, which represents a min heap used to store the ride requests.

Member Functions-

1. **public static String insert(int rideNumber, int rideCost, int tripDuration):** This method is used to insert a ride into the red-black tree and min heap, and create a pointer link between them. It returns a string "Duplicate RideNumber" if the ride number already exists in the tree, otherwise, an empty string.
2. **public static String print(int rideNumber):** This method returns the ride details for the given rideNumber and returns a string containing the ride details in the format "(rideNumber, rideCost, rideDuration)".
3. **public static String print(int rideNumber1, int rideNumber2):** This method returns all the rides between rideNumber1 and rideNumber2, and returns a string containing the list of ride details in the format "(rideNumber, rideCost, rideDuration)" separated by commas.
4. **public static String getNextRide():** This method removes the ride with the lowest cost from the min heap and subsequently from the red-black tree, and returns the ride details as a string. If there are no active ride requests, it returns a message "No active ride requests".
5. **public static void cancelRide(int rideNumber):** This method cancels (removes) the ride with the given rideNumber from the red-black tree and subsequently from the min heap.
6. **public static void updateTrip(int rideNumber, int new_tripDuration):** This method updates the trip details of the ride with the given rideNumber. The ride cost changes according to the new tripDuration.
7. **public static void main(String[] args):** This is the main method where the driver code resides. This function takes a file as input, executes all the method calls in it and writes the output to a output file.

- # Time and space complexities
  - ## Insert(rideNumber, rideCost, tripDuration)

    Time complexity: O(log n)

    In a red-black tree a new ride is inserted into the leaf after finding its parent and then rotations are performed to maintain the red-black tree properties. All the steps except finding the parent of the new ride are constant time operations. And since red black tree is a search tree, we are reducing the tree size by half at each step, so the complexity is O(log n), .

    Space complexity: O(1)

    The space complexity here is O(1) since the method is independent of n.
  - ## print(int rideNumber)

    Time complexity: O(log n)

    The method for searching a target rideNumber in a red-black tree has a time complexity of **O(log n)**, where n represents the total number of rides in the tree. The search starts from the root of the tree and involves comparing the target rideNumber with the value of the current node. If the target rideNumber is smaller, the search moves to the left subtree, and if it is larger, the search moves to the right subtree. Here we can see that once we move to one side of the tree the other side is completely out of the game. So at each step, half of the remaining tree is not considered, resulting in O(log n).

    Space complexity: O(1)

    The space complexity here is **O(1)** since the method is independent of n.
  - ## Print(rideNumber1, rideNumber2)

    Time complexity: O(log(n) + S)

    The time complexity of this operation is **O(log(n) + S)** where n is the number of rides in the tree currently and S is the number of rides that fall in the range of rideNumber1 and rideNumber2.

    - Here we search for both the rides rideNumber1 and rideNumber2 which will take O(log(n)) for both rides.
    - Next we need to traverse the subtree whose root is rideNumber1 and print all the rides which fall in range. This will take O(s) time as we need to print S rides, assuming we have S rides in range.
    - So, finally the overall time complexity is O(2* log n + S) which is O(log (n) +S)

    Space complexity: O(S)

    The space complexity here is **O(S)** since we are storing S rides which are in range
  - ## GetNextRide()

    Time complexity: O(log n)

    - Considering a min heap, the root stores the ride with the cheapest ride cost. To remove the root element, it is removed and replaced with the last leaf element. And then, to maintain the min-heap property heapification happens which is transferring the smallest element to the root which happens stepwise, resulting in a logarithmic**(O(log n))** time complexity.

- For the red-black tree, we get the pointer to the red-black tree from the min heap. Once we get the ride, the ride is removed from the tree and subsequently, rotations and color switches are performed to maintain the tree's properties. All these operations take constant time, so the overall complexity for red-black remove is O(1).
- Finally taking both we see that the time complexity is O(log n)

Space complexity: O(1)

The space complexity here is O(1) since the method is independent of n.

- ## CancelRide(rideNumber)

Time complexity: O(log n)

- Considering **red-black tree**, we first search for the ride with rideNumber specified. This step takes O(log n) since we are only vising one subtree at each step and the height of the tree is log n. Then we actually delete the ride and perform rotations and color switches to maintain the tree properties. These operations are constant time. So deletion from red-black tree has a time complexity of O(log n).
- Using the pointer correspondence, we can find the **min heap** ride to be deleted from the red black ride. Once we find the ride, it is deleted in constant time. Then, to maintain the heap properties, heapify is performed which works level wise to store the minimum element in the root of each subtree. This step takes O(log n) time.
- So, overall the time complexity is O(log n).

Space complexity: O(1)

The space complexity here is O(1) since the method is independent of n

- ## UpdateTrip(rideNumber, new_tripDuration)

Time complexity: O(log n)

Here we need to search for the node to be updated using the rideNumber. This takes O(log n) time. Depending on the new duration, in the worst case we have to do an insert and a cancel. We have seen that insert and cancel takes O(log n) each. Therefore, the overall time complexity is O(log n).

Space complexity: O(1)

The space complexity here is O(1) since the method is independent of n

- ## Overall space complexities of Red Black tree and min heap

  - ### Red-black tree

    The space complexity of a red-black tree is O(n), where n is the number of rides in the tree. All the rides stored in the tree need constant space, so the total space required will be in the order of O(n).

  - ### Min Heap

    The space complexity of a min heap is O(n), where n is the number of rides in the tree. All the rides stored in the min heap need constant space, so the total space required will be in the order of O(n).