COP5536- ADS- Programming Project Report

Name- Shromana Kumar

UFID- 70621103

UF Email account- shromana.kumar@ufl.edu

In this report, I will display the function structure and prototype for the Gator Taxi project. I have included the time and space complexity for the min heap and red-black tree implementations in the Ride Class details as it is a parent class for both data structures. We will see that the time complexity is (logn) for both data structures.

1. **Class**: gatorTaxi –

   This class is the main class of my project. It has 2 functions –

   i. **Function**: void main(String[]) –

   This function is the main function that calls the readInput() function, parses the function and parameter tokens received, and then further calls the corresponding insert, update, cancel, and print functions, etc from the Ride class.

   ii. **Function**: ArrayList readInput(String) –

   This function aims to read the input from the input text file and return an ArrayList holding the input functions and their respective parameters.

2. **Class**: Ride –

   This class gets the operation requests from the gatorTaxi class, then redirects them to the Min Heap and the RB Tree classes. It has 8 functions –

   i. **Function**: void Ride() –

   This constructor function creates one instance of the min heap and one instance of the red-black tree.

   Min Heap Time Complexity – O(1)

   Min Heap Space Complexity – O(1)

   Red-Black Tree Time Complexity – O(1)

   Red-Black Tree Space Complexity – O(1)

   ii. **Function**: int getPos(int) –

   This function calls the getPos function present in the RedBlackTree class. It takes a ride number as input and returns the position of the ride number in the min heap.

   Red-Black Tree Time Complexity – O(logn) where n is the number of nodes. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

iii. **Function**: void Insert(int, int, int) –

This function is used to insert new ride details into the min heap and red-black tree. It takes as input the ride number, ride cost, and trip duration, and then creates a node in the min heap and the red-black tree. If an already existing ride number is provided as input, then it returns "Duplicate RideNumber", throws an error, and terminates the program.

Min Heap Time Complexity – O(logn) where n is the number of nodes. This is because, in a binary heap, each level is fully filled except for the leaf level.

Min Heap Space Complexity – O(1)

Red-Black Tree Time Complexity – O(logn) where n is the number of nodes. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

iv. **Function**: void UpdateTrip(int, int) –

This function is used to update the trip duration of any existing ride. It takes as input the ride number, and new trip duration, and then checks if the ride number already exists or not. If it exists, then it updates the trip duration as per the guidelines provided.

I am reusing the insert and cancel functions in this function. So, the time complexity is the same as the insert and cancel functions.

Min Heap Time Complexity – O(logn) where n is the number of nodes. This is because, in a binary heap, each level is fully filled except for the leaf level.

Min Heap Space Complexity – O(1)

Red-Black Tree Time Complexity – O(logn) where n is the number of nodes. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

v. **Function**: void CancelRide(int) –

This function is used to cancel an existing arbitrary ride and delete its corresponding nodes from the min heap and the red-black tree. It takes as input the ride number, and then checks if the ride number already exists or not. If it exists, then it deletes the ride.

Min Heap Time Complexity – O(logn) where n is the number of nodes. This is because, in a binary heap, each level is fully filled except for the leaf level.

Min Heap Space Complexity – O(1)

Red-Black Tree Time Complexity – O(logn) where n is the number of nodes. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

vi. **Function**: void GetNextRide() –

This function is used to get the ride with the minimum cost (in case of multiple rides with the same cost, the one with the minimum duration is returned). It returns the ride details for the next ride and deletes its corresponding nodes from the min heap and the red-black tree.

Min Heap Time Complexity – O(logn) where n is the number of nodes. This is because, in a binary heap, each level is fully filled except for the leaf level.

Min Heap Space Complexity – O(1)

Red-Black Tree Time Complexity – O(logn) where n is the number of nodes. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

vii. **Function**: void Print(int) –

This function is used to print the ride details of any ride. It takes as input the ride number, calls the RedBlackTree class(as it is ordered by ride number), and returns the ride details for this ride.

The print is only implemented for the red-black tree as it is ordered by ride number and would have a O(logn) time complexity during printing. The min heap is not used as it is ordered by ride cost and trip duration and would have given an O(n) time complexity if used.

Red-Black Tree Time Complexity – O(logn) where n is the number of nodes. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

viii. **Function**: void Print(int, int) –

This function is used to print the ride details of any ride whose ride number falls between the input parameters. It takes as input the max and min ride numbers, calls the RedBlackTree class(as it is ordered by ride number), and returns the ride details for all the rides that fall in the input range.

The print is only implemented for the red-black tree as it is ordered by ride number and would have a O(logn) time complexity during printing. The min heap is not used as it is ordered by ride cost and trip duration and would have given an O(n) time complexity if used.

Red-Black Tree Time Complexity – O(logn + S) where n is the number of nodes and S is the number of ride details printed. This is because a red-black tree is a balanced binary tree (the height is logarithmic with respect to the number of nodes) and the number of rotations required during insertion and deletion is limited to a constant number.

Red-Black Tree Space Complexity – O(1)

3. **Class**: MinHeap –

This class gets the operation requests from the Ride class, then performs the requested operation on the Min Heap. It has 11 functions –

i. **Function**: void MinHeap() –
This constructor function creates one instance of the Node class.

ii. **Function**: int parent(int) –
This function takes as input the position of a ride in the heap and returns the position of the parent of the ride.

iii. **Function**: int leftChild(int) –
This function takes as input the position of a ride in the heap and returns the position of the left child of the ride.

iv. **Function**: int rightChild(int) –
This function takes as input the position of a ride in the heap and returns the position of the right child of the ride.

v. **Function**: boolean isLeaf(int) –
This function takes as input the position of a ride in the heap and returns true if the node is a leaf node in the min heap and returns false otherwise.

vi. **Function**: boolean exists(int) –
This function takes as input the position of a ride in the heap and returns true if the node exists and returns false otherwise.

vii. **Function**: void swap(int, int) –
This function takes as input the position of 2 rides in the heap and swaps the position of the 2 rides in the min heap.

viii. **Function**: void minHeapify(int) –
This function takes as input the position of a ride in the heap and checks and corrects that the subtree rooted at the position of the ride follows the principles of a min heap.

ix. **Function**: Node Insert(int, int, int) –
This function is used to insert new ride details into the min heap. It takes as input the ride number, ride cost, and trip duration, and then creates a node at the end of the min

heap. It then checks and corrects that the subtree of the new ride follows the principles of a min heap.

x. **Function**: void GetNextRide() –
This function is used to get the ride with the minimum cost (in case of multiple rides with the same cost, the one with the minimum duration is returned) from the min heap. It returns the ride details of the root of the min heap, deletes the root, and replaces it with the last element in the min heap. It then checks and corrects that the subtree of the new ride follows the principles of a min heap.

xi. **Function**: void CancelRide(int, int) –
This function is used to cancel an existing arbitrary ride and delete its corresponding nodes from the min heap. It takes as input the ride number and the position of a ride in the heap. It deletes the ride from the heap and then it checks and corrects that the subtree of the new ride follows the principles of a min heap.

4. **Class**: RedBlackTree –
This class gets the operation requests from the Ride class, then performs the requested operation on the Min Heap. It has 8 functions –

i. **Function**: void RedBlackTree() –
This constructor function creates a root variable with NULL assigned to it.

ii. **Function**: RBNode RR(RBNode) –
This function takes as input an RBNode variable and performs RR rotation on the node and then returns the node which takes its position in the red-black tree.

iii. **Function**: RBNode LL(RBNode) –
This function takes as input an RBNode variable and performs LL rotation on the node and then returns the node which takes its position in the red-black tree.

iv. **Function**: void balanceInsert(RBNode) –
This function takes as input an RBNode variable and checks if the balancing criteria for the red-black tree are met for that node, and if not then it corrects the balance.

v. **Function**: void balanceDelete(RBNode) –
This function takes as input an RBNode variable and checks if the balancing criteria for the red-black tree are met for that node, and if not then it corrects the balance.

vi. **Function**: void Insert(Node, int) –
This function is used to insert new ride details into the red-black tree. It takes as input the min heap Node for the ride and the ride number. It creates an RBNode variable for the ride, traverses through the tree to find its position, then performs balancing so that the tree still follows the principles of a red-black tree.

vii. **Function**: void CancelRide(int) –
This function is used to cancel an existing arbitrary ride and delete its corresponding nodes from the red-black tree. It takes as input the ride number. It searches for the node in the red-black tree, deletes the ride, replaces it with the minimum ride from the right subtree, then performs balancing so that the tree still follows the principles of a red-black tree.

viii. **Function**: int getPos(int) –

This function takes a ride number as input, searches for the ride number in the red-black tree, and then returns the position of the ride number in the red-black tree.

ix. **Function**: void Print(int) –

This function is used to print the ride details of any ride. It takes as input the ride number, searches for the ride number in the red-black tree, and returns the ride details for this ride.

x. **Function**: void Print(int, int) –

This function is used to print the ride details of any ride whose ride number falls between the input parameters. It takes as input the max and min ride numbers, searches for the ride numbers in the red-black tree, and returns the ride details for all the rides that fall in the input range.

5. **Class**: WriteOutput –

This class is used to write the output into a file. It has 1 function –

i. **Function**: void writeOutputToFile(String, String) –

This function takes as input a string content and the name of the of output file and appends the input string content into the output file.

6. **Class**: Node –

This class is used to hold the nodes of the min heap. It has 1 function –

ii. **Function**: void Node(int, int, int, int) –

This constructor function takes as input the ride number, ride cost, trip duration, and position of a node in the heap and creates a Node variable that would hold the details of that node.

7. **Class**: RBNode –

This class is used to hold the nodes of the red-black tree. It has 1 function –

iii. **Function**: void RBNode(Node, int) –

This constructor function takes as input the ride number and the corresponding Node variable for the ride, and creates a RBNode variable which would hold the details of that node along with parent-child relationship and colour.