COP 5536 Spring 2023 – Advanced Data Structures
Programming Project

Name: Sai Bhavana Nammi

UFId: 8950-5823

UF Email: snammi@ufl.edu

Project Description:

This project is about GatorTaxi which is a ride-sharing service. There are many ride requests that are catered to every day and hence, there is a need to create software which would help them keep a track of the ride requests that are pending.

The overall implementation of this project has been done in Python using Red-Black Tree (RBT) and min-heap. A Red-Black Tree is a self-balancing Binary Search Tree (BST) where all the nodes are colored either Red or Black. A min-heap is a Binary Tree where the key at the root must be always less than or equal among the keys of the children present.

In this project, the various key operations for Red-Black Tree have been implemented such as the insert, delete, search, balancing the tree after insert/delete, to find an element in range.

Similarly, for min-heap various key operations such as the insert, delete, update, heapify, pop have been implemented.

Using all the above implemented operations, various functions as per the requirements of the problem statement have been implemented.

1. Print(rideNumber) – to print the required ride
2. Print(rideNumber1, rideNumber2) – to print all the rides between rideNumber1 and rideNumber2
3. Insert(rideNumber, rideCost, tripDuration) – to insert a new ride for every different rideNumber
4. GetNextRide() – to get the ride with the lowest rideCost which is then deleted
5. CancelRide(rideNumber) – to delete the ride
6. UpdateTrip(rideNumber, new_tripDuration) – to update the ride if the destination is changed within a set of existing conditions

Here, the ride is identified by the triplet of rideNumber, rideCost and tripDuration.

Project File Structure:

As the project has been implemented in python, there is no makefile.

red_black_tree.py – Contains all the implementation for the operations to be performed on the red-black tree being used in the project.

min_heap.py – Contains all the implementation for the operations to be performed on the min-heap being used in the project.

ride_model.py – Contains the class Ride which is the definition for our Ride identified by the triplet rideNumber, rideCost and tripDuration and this file is responsible to make the comparison of cost and duration of two rides and returns the one which is the cheapest.

gator_taxi.py – Used to call the needed main functions mentioned in the problem statement like insert_ride, print_ride, delete_ride and these functions call the respective functions from the above mentioned min_heap.py and red_black_tree.py.

input.txt – Contains the operations to be performed on the red-black tree and min-heap.

output_file.txt – File that is generated as output.

Execution Steps:

Unzip the file and run the command as python gator_taxi.py input.txt which will generate the output in the output_file.txt.

Function prototypes:

Min-heap Structure -

```
class MinHeap:
    def __init__(self):
        self.heap_list = [0]
        self.curr_size = 0
    def insert(self, ele):...
    def heapify_up(self, p):...
    def swap(self, ind1, ind2):...
    def heapify_down(self, p):...
    def get_min_child_index(self, p):...
    def update_element(self, p, new_key):...
    def delete_element(self, p):...
    def pop(self):...

class MinHeapNode:
    def __init__(self, ride, rbt, min_heap_index):
        self.ride = ride
        self.rbTree = rbt
        self.min_heap_index = min_heap_index
```

The methods in our MinHeap class have been used for the following –

__init__: To initialize the heap with empty list and a current size set to 0.

insert: To insert element in the heap.

heapify_up: To maintain the property of the heap where the current node is swapped with the parent till parent is less than current node. It is called after the insert is done.

delete_element: To delete the element in the heap.

heapify_down: To maintain the property of the heap where the current node is swapped with the smallest child till the current node is less than both the child nodes.

update_element: To update the element in the heap. Maintaining the properties of the heap is done by heapify_up or heapify_down after updating the element.

swap: To swap two elements in a heap and also the index of these nodes are updated.

pop: To remove the root element of the heap and also to return it. The property of the heap is handled by calling heapify_down.

get_min_child: To get the index of the child which is the smallest for a node.

MinHeapNode is used to show the structure of a node in min-heap with a ride, rbTree and the index of the min-heap which keeps track of the position of any node in the heap.

Red-Black Tree Structure:

```
class RBTNode:
    def __init__(self, ride, min_heap_node):
        self.ride = ride
        self.parent = None  # parent node
        self.left = None  # left node
        self.right = None  # right node
        self.color = 1  # 1=red , 0 = black
        self.min_heap_node = min_heap_node
class RedBlackTree:
    def __init__(self):
        self.null_node = RBTNode(None, None)
        self.null_node.left = None
        self.null_node.right = None
        self.null_node.color = 0
        self.root = self.null_node
```

```
    # To retrieve the ride with the rideNumber equal to the key
    def get_ride(self, key):...
    # Balancing the tree after deletion
    def balance_tree_after_delete(self, node):...
    def __rb_transplant(self, node, child_node):...
    def delete_node_helper(self, node, key):...
    def balance_after_insert(self, curr_node):...
    def find_ride_in_range(self, node, low, high, res):...
    def get_rides_in_range(self, low, high):...
    def minimum(self, node):...
    def l_rotation(self, x):...
    def r_rotation(self, x):...
    def insert(self, ride, min_heap):...
    def delete_node(self, rideNumber):...
```

The RedBlackTree class has the entire implementation of the red-black tree with many of the methods to make the our program run as asked in the problem statement.

get_ride: It takes rideNumber as the parameter and the tree returns the node for the ride that exists in the tree.

insert: It takes ride and min_heap node as its parameters to create a new node and then insertion into the tree happens. But, after this to maintain the properties of the red-black tree, fix_insert method is called.

delete: It takes rideNumber as its parameter and it deletes the node from the tree. If there exists two child nodes, then the minimum node is replaced in the respective right subtree. To maintain the properties of a red-black tree, fix_delete is called.

get_rides_inrange: It takes low and high as parameters and gives the nodes lying in that specific range.

left_rotation, right_rotation: This is to balance the tree after performing the insert or delete operations on the tree.

gator_taxi.py -

```
# to import required functions from other files into our file
from ride_model import Ride
from min_heap import MinHeap
from min_heap import MinHeapNode
from reb_black_tree import RedBlackTree, RBTNode

# for insert command
def insert_ride(ride, heap, rbtree):...

# to update a ride
def update_ride(rideNumber, new_duration, heap, rbtree):...
# to get the next ride
def get_next_ride(heap, rbtree):...
# to print the triplet
def print_ride(rideNumber, rbtree):...

# to print all the triplets
def print_rides(l, h, rbtree):...

# to cancel a ride
def cancel_ride(ride_number, heap, rbtree):...

# to write the output to the output_file
def write_to_output(ride, mssg, list):...
```

The above picture shows all the functions required in the problem statement.

insert_ride: It inserts a new ride in both the min-heap and red-black tree and it also checks if there exists a ride already then it returns ‘Duplicate Ride Number’.

update_ride: It updates the duration of the ride by checking the three conditions that are given.

get_next_ride: It fetches the next ride from the min-heap and then proceeds to remove the node from the red-black tree and also prints the details of the ride. If such a ride does not exist then it returns “No active ride requests”.

print_ride: It is to print the details of the ride using the rideNumber. If such a ride does not exist then it returns “No active ride requests”.

print_rides: It is to print all the rides present between the range of rideNumber1 and rideNumber2.

cancel_ride: It is to cancel for a given rideNumber and it removes the ride from the min-heap and the red-black tree.

write_to_output: This is to write our generated output to the output_file.txt.