COP 5536 Spring 2023 – Advanced Data Structures
Programming Project

Name: Nikhitha Sudati
UFId: 9493-6209
UF Email: sudatinikhitha@ufl.edu

Project Description:

Implement Min-heap and Red-black-Tree for GatorTaxi which is an upcoming ride-sharing service.

Red-Black Tree (RBT) and min-heap have been used to implement this project altogether. A Red-Black Tree is a self-balancing Binary Search Tree (BST) in which each node is either Red or Black in color. A min-heap is a Binary Tree in which the root key must always be smaller than or equal to one of the keys of any existing children.

The various crucial Red-Black Tree actions, including insert, delete, search, balancing the tree after insert/delete, and finding an element in range, have all been implemented in this project.

Similar core actions, including insert, delete, update, heapify, and pop, have been developed for min-heap.

1. Print(rideNumber) – This prints the requested rideto print the required ride
2. Print(rideNumber1, rideNumber2) – This function prints all the rides in range between rideNumber1 and rideNumber2
3. Insert(rideNumber, rideCost, tripDuration) – This function insert a new ride and will not print anything in the output
4. GetNextRide() – This function is used to get next minimun cost ride from the min_heap
5. CancelRide(rideNumber) – This function deletes the ride
6. UpdateTrip(rideNumber, new_tripDuration) – Updates the ride with the new tripduration based on the conditions.

Implemented in Python.

Project File Structure:

I have implemented in python, so there is no makefile.

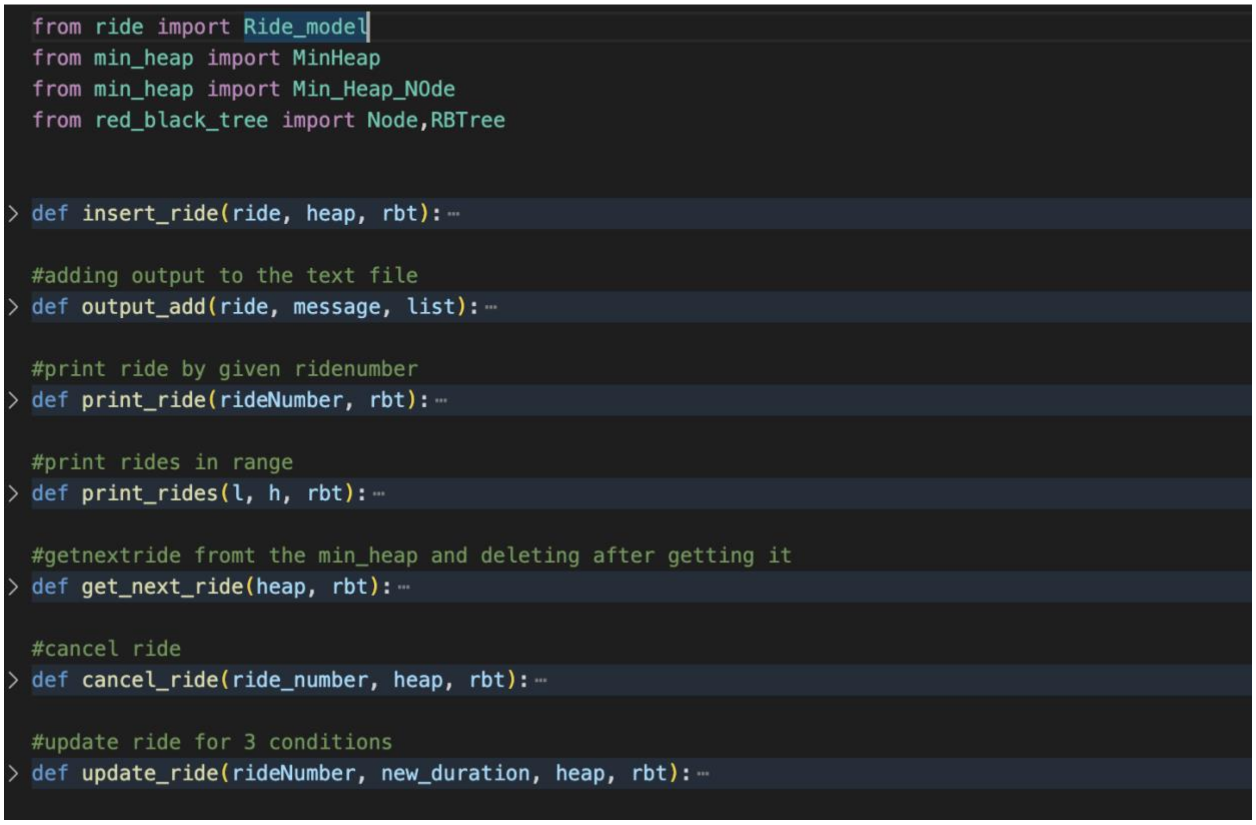
gator_taxi.py: contains all the main functions to insert_ride, print_ride, get_next_ride, cancel_ride, update_ride. Where each of these functions calls respective functions from min_heap.py and red_black_tree.py
min_heap.py: This file contains the implementation of min_heap, which is used to store(rideNumber, rideCost, tripDuration) triplets in the order of rideCost.This function returns the next available ride with minimun ride cost.
red_black_tree.py: This file contains the implementation of red_black_tree which is used to store(rideNumber,rideCost,tripDuration) triplets in the order of rideNumber.
ride.py: This file compares the cost and duration of two ride instances and return the cheaper among them
input.txt: This file is the input for testcase2
output_file.txt : This file is generated as output for input.txt which contains testcase2
input1.txt : this file is the input for testcase1
output1.txt: generated output file for input1.txt which contains data of testcase1

Execution Steps:

Unzip the file and run python3 gator_taxi.py input.txt, then the output will be generated in output_file.txt.

Function Prototypes:

gator_taxi:



```
from ride import Ride_model
from min_heap import MinHeap
from min_heap import Min_Heap_NOde
from red_black_tree import Node,RBTree


def insert_ride(ride, heap, rbt):⋯

#adding output to the text file
def output_add(ride, message, list):⋯

#print ride by given ridenumber
def print_ride(rideNumber, rbt):⋯

#print rides in range
def print_rides(l, h, rbt):⋯

#getnextride fromt the min_heap and deleting after getting it
def get_next_ride(heap, rbt):⋯

#cancel ride
def cancel_ride(ride_number, heap, rbt):⋯

#update ride for 3 conditions
def update_ride(rideNumber, new_duration, heap, rbt):⋯
```

The above picture contains all the functions. And the respective function from the required class is called.

insert_ride: Inserts new ride in both min heap and RBT and checks if the existence of the ride, if it exits return "Duplicate Ride Number"

print_ride: This prints the ride details with the given ridenumber. If the ride doesn't exits it returns "No active ride requests".

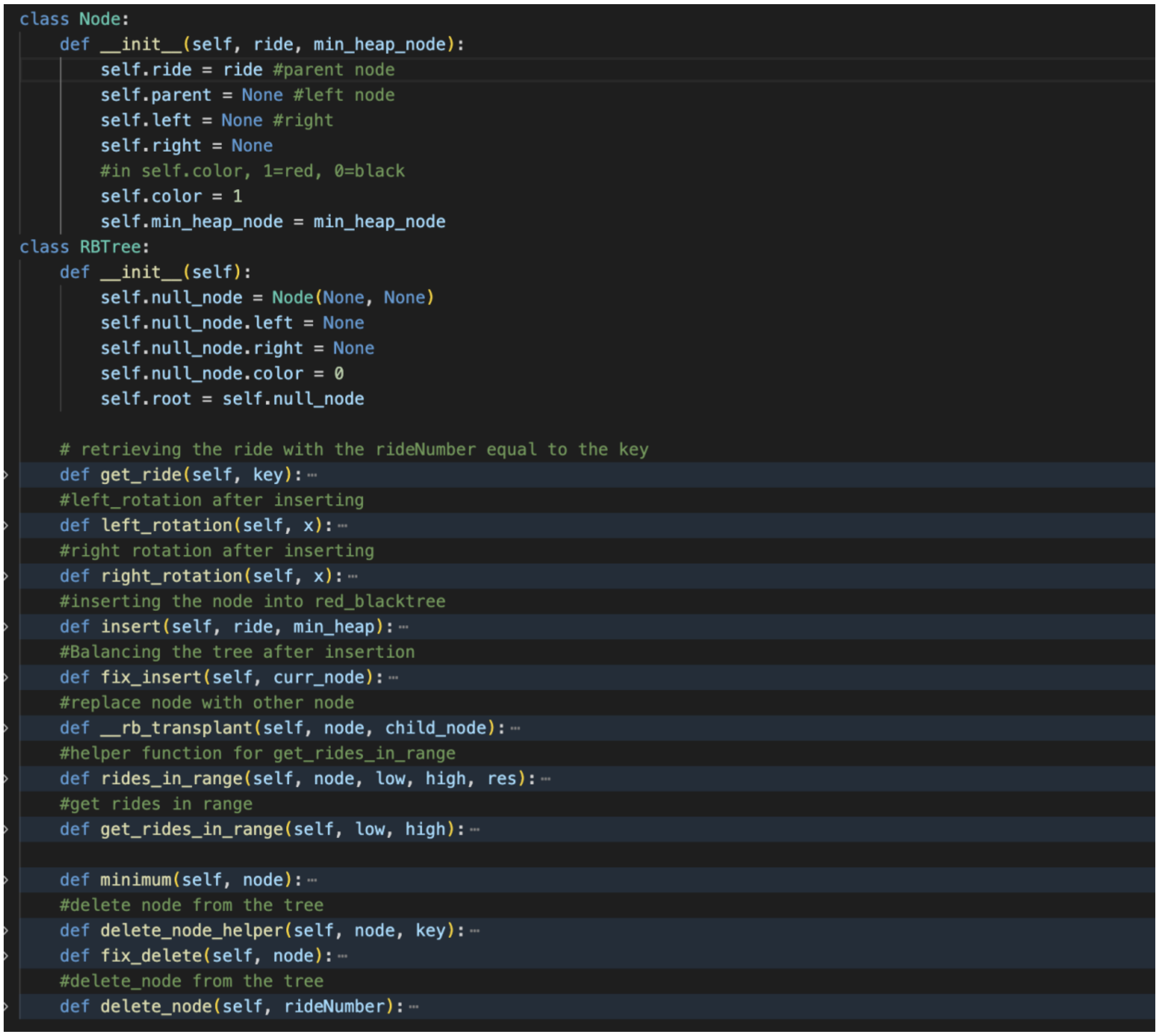
print_rides: This method checks for the rides in range between ridenumber1 and ridenumber2.

get_next_ride: This method gets the next ride from the min heap and removes the node from red black tree and prints the ride details. If no ride exits returns "No active ride requests"

cancel_ride: Cancel the ride for the ridenumber and removes that ride from both min heap and red black tree.

update_ride: updates the ride duration, by checking three conditions.

Red-Black Tree Structure:

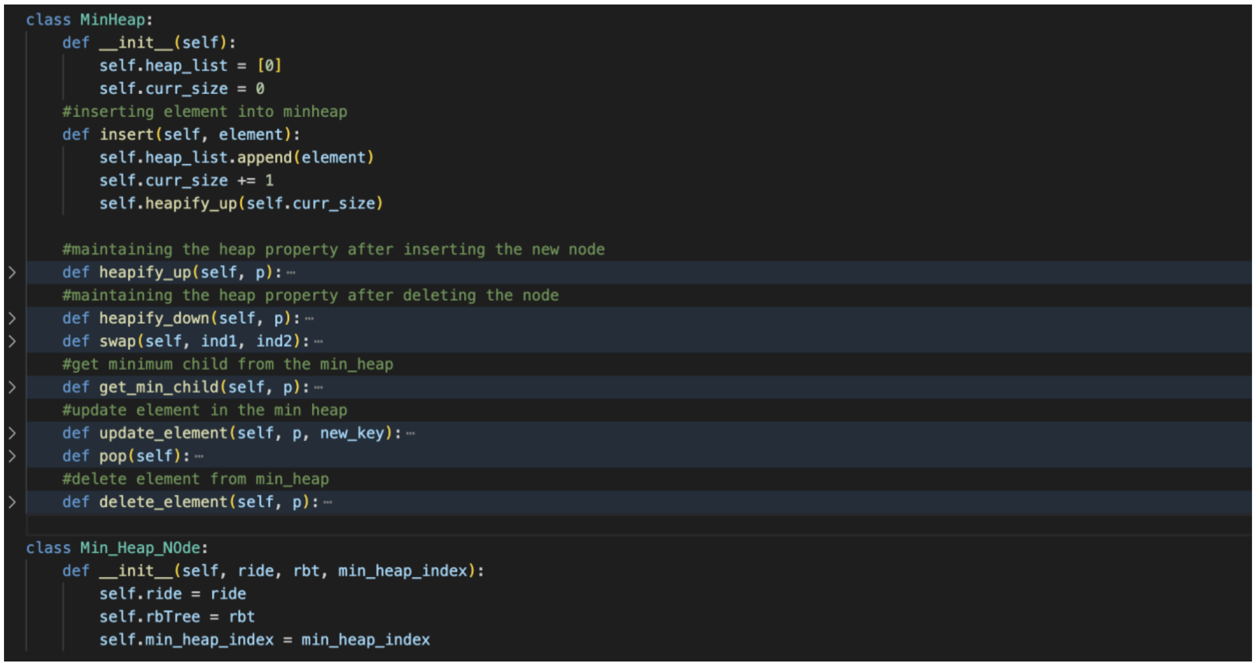


```python
class Node:
    def __init__(self, ride, min_heap_node):
        self.ride = ride #parent node
        self.parent = None #left node
        self.left = None #right
        self.right = None
        #in self.color, 1=red, 0=black
        self.color = 1
        self.min_heap_node = min_heap_node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

    # retrieving the ride with the rideNumber equal to the key
    def get_ride(self, key):⋯
    #left_rotation after inserting
    def left_rotation(self, x):⋯
    #right rotation after inserting
    def right_rotation(self, x):⋯
    #inserting the node into red_blacktree
    def insert(self, ride, min_heap):⋯
    #Balancing the tree after insertion
    def fix_insert(self, curr_node):⋯
    #replace node with other node
    def __rb_transplant(self, node, child_node):⋯
    #helper function for get_rides_in_range
    def rides_in_range(self, node, low, high, res):⋯
    #get rides in range
    def get_rides_in_range(self, low, high):⋯

    def minimum(self, node):⋯
    #delete node from the tree
    def delete_node_helper(self, node, key):⋯
    def fix_delete(self, node):⋯
    #delete_node from the tree
    def delete_node(self, rideNumber):⋯
```

The RBTree class contains the complete implementation of the red-black tree algorithm. This contains several methods

get_ride: This method takes in ride number as its parameter and if the ride exits in the tree it returns the node containing the ride.
left_rotation and right_rotation: Used to balance the tree after insertion or deletion from the tree.
insert:This method takes two parameters ride,min_heap node. It creates a new node with these values and inserts it into tree. The fix_insert method is called after the node is inserted to maintain the red-black tree properties.
delete: This method deletes the node from the tree. If the node has two children it replaces with a minimum node in the right subtree. Once deleted, fix_delete is called to maintain the property of tree.
get_rides_inrange: This takes two parameters and returns node in that specific range.

Min-heap Structure:



```
class MinHeap:
    def __init__(self):
        self.heap_list = [0]
        self.curr_size = 0
    #inserting element into minheap
    def insert(self, element):
        self.heap_list.append(element)
        self.curr_size += 1
        self.heapify_up(self.curr_size)

    #maintaining the heap property after inserting the new node
    def heapify_up(self, p):⋯
    #maintaining the heap property after deleting the node
    def heapify_down(self, p):⋯
    def swap(self, ind1, ind2):⋯
    #get minimum child from the min_heap
    def get_min_child(self, p):⋯
    #update element in the min heap
    def update_element(self, p, new_key):⋯
    def pop(self):⋯
    #delete element from min_heap
    def delete_element(self, p):⋯

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

The above MinHeap class few methods:

__init__ : This method initializes the heap with an empty list and a current size with 0
insert: This method insert an element into the heap by maintaining the heap property by calling the heapify_up method.
heapify_up : This method is called in the Insert method to maintain the heap property by swapping the current node with its parent until the parent is smaller than the current node
heapify_down: This method is called in the delete method, to maintain the heap property by swapping the current node with its smallest child until the node is smaller than both of its children.
swap: This method is used to swap the position of 2 elements in the heap and updates their respective indices.
get_min_child: This method gives the index of the smallest child of the node.
update_element: Updates the element and also maintain the heap property by calling heapify_up or heapify_down.
pop: Pop method removes and returns the heap's root node and maintains heap property by heapify_down
delete_element : This method deletd the node from the heap and maintains the heap property by calling heapify_down.

This code also defines an other class Min_Heap_Node, which represents a node in the minimum heap. And each node consists of a ride, an RBTree object, and a minimum heap index.
The minimum heap index is used to keep track of the node’s position in the heap.