110065528 張云瑄

1. **How to compile and execute**
    a. change the path of the test case in the program
    b. type “make” in the terminal

2. **Details of the algorithm**
    a. **5 objects in the project**
        i. **User:** a request represent an User
            - id
            - start/end time
            - start/end station
            - operator overloading (=, >, <)
        ii. **Request_mgmt:** a controller to do the operations of Users
            - A minHeap which stores all Users
            - some member functions (Create, run_request, … etc)
        iii. **Bike:** a bike object means a real bike
            - id
            - type
            - station
            - price
            - count
            - start time (available start time)
            - available (retire or not)
            - operator overloading (=, >, <)
        iv. **Bike_Mgmt:** a controller to do the operations of Bikes
            - 2 array of AVL_Trees: storing “available” bikes and “retire” bikes
            - count limit
            - discount
            - array of price of all bike types
            - some member functions (Create, getBike, moveBike, … etc)
        v. **IdString:** store the strings with User.id, aim to sort the logs in files
            - res (string)
            - id (User.id)

    b. **steps**

i. Create the **"minHeap" of all User** (small start time first → small id first)

ii. Create **all bikes** objects and insert them into the **"available AVL_Tree"**

iii. Create a **matrix "minCost"** (n*n)
- store the min cost of every two stations
- using "Floyd-Warshall" algorithm

iv. Create **two AVL_Tree of "IdString"** (user_result and transfer_log)
- storing the results need to be written into "user_result.txt" and "transsfer_log.txt" separately.

v. **Deal** with **all User requests**
- Pop the User from the minHeap
- Search suitable bikes from the "available AVL_Tree" in the Bike_mgmt (pick the price highest one)
- if found available bike
    a. add the IdString of the result into the "user_result" AVL_Tree
    b. add the IdString of the result into the "transfer_log" AVL_Tree
    c. move bike and update bike status
        i. delete the bike from the org station AVLTree
        ii. update the status of the bike
        iii. check the bike is available or retired
            1. insert the bike into the new station AVLTree (available or retired)
- if not found available bike
    a. add the IdString of the result into the "user_result" AVL_Tree

vi. **Write results** into three files
- user_result.txt
    a. Inorder traversal the "user_result" "AVLTree of IdString"
- transfer_log.txt
    a. Inorder traversal the "transfer_logt" "AVLTree of IdString"
- station_status.txt
    a. search the bikes station by station
        i. Create an "result AVLTree" of bike

ii. traverse the bikes in the “available tree” and “retire tree” of the station and Insert into the “result AVLTree”

iii. Inorder traversal the “result AVLTree”

3. Using data structure

a. minHeap

i. where to use in the project

- store all User
- store available bike types of each user

ii. how to construct

- create template class “MinHeap”
- some member functions (details in the code)

b. AVLTree

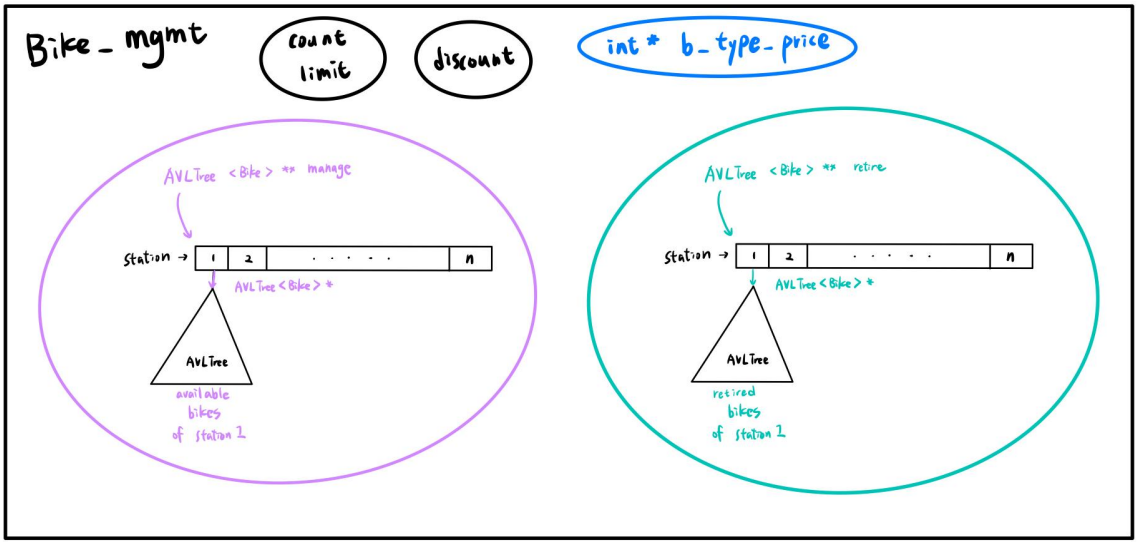
i. where to use in the project

- store all bikes in the “Bike_mgmt”
- store the results which need to be written into three files

ii. how to construct

- create template class “AVLTree”
- some member functions (details in the code)

c. details of how to store bikes in the “Bike_mgmt”



4. Feedback

Thanks all TAs.