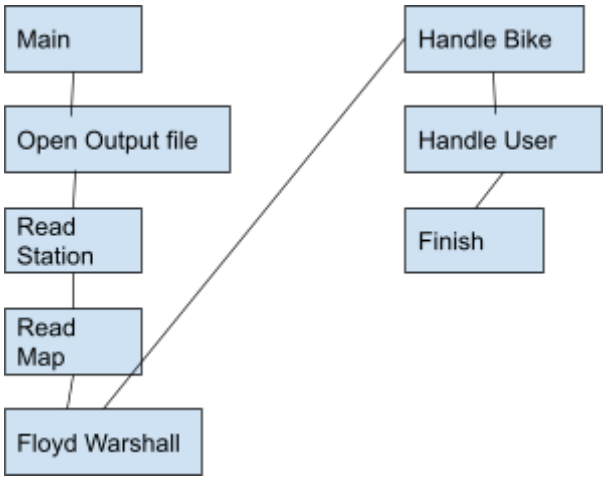
Data Structure Final Project Report
109006201 周志偉

1. I did not do the advance part this is only basic
2. How to compile?
    a. I uses the makefile to compile my program
    b. g++ -std=c++11 -o ./bin/main ./src/main.cpp
    c. ./bin/main $(case) $(version)
    d. Select the case and version you want to compile
        i. Type "make"
        ii. Select the case and version you want to compile
        iii.
```
Davis-2:DS_final_project davis$ make
g++ -std=c++17  -o ./bin/main ./src/main.cpp
./bin/main case1 basic
You have set case1 as your testcase:
running basic currently
----------------------------------------
----------------------------------------
finished computation at Sun Jan  8 00:11:56 2023
elapsed time: 7.09897s
```
        iv.
```
Davis-2:109006201_proj davis$ ./bin/verifier case1
You have set case1 as your path:
----------------------------------------
start load result
bike_deprecation_rate : 0.500000 max_rental_count : 40
----------------------------------------
Total Revenue : 47437
----------------------------------------
finished computation at Sun Jan  8 17:45:20 2023
elapsed time: 0.012404s
```

3. Program Structure
    a.


    b. Figure 1 is the flow chart of main. We will first read the input from the text file and from map.txt data we will use Floyd Warshall to calculate the shortest path. We will then process the user and bike from min_time to max_time and output the data to the result in the end.
        i. Import_files base on case
            1. Input map
                a. Initialize the distances first

        b. Extract the int of station 1 and station 2
    2. Input bike_info
        a. Read the depreciation_discount_price and rental_count_limit;
        b. Extract id from Bike type and set it to id.
        c. Record bike_info[id] = price;
    3. Input User
        a. Input the user and record user in priority queue
        b. Sort the user in start time first then sort the user in ID
    4. Input Bike
        a. read and record each bike entry
        b. record the bike entry in the station it is initially in
        c. Put it in the priority queue, sort it from highest price then smallest id

ii. floyd-warshall algorithm to find shortest path between two stations
    1. shortest_paths()

iii. Handle Bike()
    1. Check first if there is bike exit if no bike
    2. Pop bike from not_yet_bikes priority queue
    3. If the rental count is more than the bike count limit retired the bike(sort by id)
    4. Else just normally insert the bike in the corresponding priority queue, put it in the priority queue, sort it from highest price then smallest id

iv. Handling User(int time)
    1. loop till all bikes arriving now are extracted
    2. exit if no users arrived, or no users at all
    3. Pop bike from priority queue
    4. Jump to handling user (user u)

v. Handling User(user u)
    1. int distance = distances[u.start_point][u.end_point];
    2. int user_time = u.end_time - u.start_time;
    3. Check if we can get there on time
    4. for every desired bike_type check if Bike exists at station and choose the bike with highest rental price and smallest bike id
    5. If it not resolve then it will reject the bike
    6. Remove the bike from the station
    7. Print to user output and transfer output

vi. Print_bikes
    1. Basically for outputting the station status
    2. I have priority queue for outputting the station
    3. Sort by bike id using the output bike , both station bike and retired bike

4. Data structure
    a. The details of your data structures. What data structures did you use, and how did you implement those data structures?
    b. I use a priority queue in my program

i. a priority Queue is an extension of the queue with the following properties. Every item has a priority associated with it. An element with high priority is dequeued before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.
    1. The element with the highest value is assigned the highest priority and the element with the lowest value is assigned the lowest priority.
ii. In my priority queue class I will have to scale up and scale down to double or halve the size of the array.
iii. Function enqueue is to insert a new element
iv. Function dequeue is to remove the element with the highest priority
v. Function Function peek is to check the top element

c. Heap will be use
    i. Heapify Up
        1. add elements one by one to the sorted array and then adjust the position by moving it up the heap as much as needed (“HEAPIFY UP”).
    ii. Heapify Down
        1. if we move the nodes downward instead, the number of swaps decreases drastically since now the first layer with the least nodes needs the most swaps.