Maximize the Revenue for the NTHU-Bike Company

Last Update: 2021/12/12

1. Project Objective:

Apply the knowledge learned from the Data Structures course and implement it in a real-world scenario.

2. Project Description

You are assigned to help the "NTHU-Bike," a bike rental company, to design a rental analysis tool to improve its workflow management. The company will provide a location map of available bike stations, initial bike allocations, and rental fee schedule. The tool first calculates the company's expected revenue and the final bike distribution for each given scenario. Then, the tool shall also figure out how much more revenue can be gained if the company adopts different management policies.

The user data include both when and where a user rent or return bikes. Each data entry can be one of the following forms:

- rent **stationIdRent bikeType userId timeRent**
- return **stationIdReturn userId timeReturn**

The above bike renting and returning entries describe that **userId** rents a bike of type **bikeType**, and leave from station **stationIdRent** at **timeRent** to **stationIdReturn** at **timeReturn**. The rental fee charge is proportional to the total rental elapse time, i.e., $timeReturn - timeRent$.

The rental company offers a **discounted** and a **regular** per unit time rental rates. If the user returns the bike in the shortest time, then the discounted rate is applied, otherwise the regular rate is.

We use the following rental example to illustrate the fee calculation policy.

- rent s_1 **electric 00002 0**

  // user **00002** rents an **electric** bike from station s_1 at time **0**
- return s_2 **00002 20**

  // user **00002** returns the bike to station s_2 at time **20**

Suppose that the discounted and regular rates for electric bikes are 10 and 15, respectively. If the shortest time t_{s_1,s_2} required to travel between s_1 and s_2 is 20, then the user **00002** will be charged for 20*10 = $200, since the bike is returned in the shortest time. However, if t_{s_1,s_2}=10, then the user will be charged for 20*15=$300.

The tool should constantly keep track of the total company collected fees and the inventory status of each bike station. When a user rents a bike in s_i, the inventory of the type should decrease by one immediately; also, when a user returns a bike to s_j, the inventory should increase by one. Furthermore, the company assigns each bike a unique **bikeId**, and each station shall rent out the bike with the smallest bikeId.

We use an example to illustrate the inventory policy. Suppose the renter **00002** rents and returns as indicated in the last example, and there are initially two electric bikes at both station s_1 and s_2:

- s_1:

  electric: 100 101

  lady:

  road:

- s_2:

  electric: 200 201

  lady:

  road:

After the rental request of renter **00002** is exercised at time 0, the inventory status becomes:

- s_1:

  electric: 101

  lady:

  road:

- s_2:

  electric: 200 201

  lady:

  road:

Then when renter **00002** returns the bike at time 20, the inventory status becomes:

- s_1:

  electric: 101

  lady:

  road:

- s_2:

  electric: 100 200 201

  lady:

  road:

For the basic implementation, if bikes are not available, the rental request is simply rejected with no charge.

For the advanced version of the implementation, the company may apply a combination of the following options.

I. Reject the rental in case of shortage with no charge.

II. The renter can wait at the renting station s_i until a bike of the target type is available and the company pays a **waiting fee** to the renter as a compensation. The waiting fee shall be the per unit time waiting fee (to be specified in inputs) times the waiting time. Certainly, the waiting fee shall be deducted

from the revenue. Note that if a user waits to the closing time and no bike is available, the company should still pay the waiting fee but no other charges.

Note: Opening time is t=0, and closing time is t=1440. All rent and return must be done before closing time. Any operation or response whose processing would extend beyond the closing time is considered illegal.

III. Let the renter choose other type of bike available at the renting station s_i in case of shortage and charge the rental fee of the new bike type with a **reduced rate** (some % off.) For example, if the reduced rate is 0.8, the user will be charged 0.8 * (the normal fee).

IV. The company may pay a **transfer fee** (to be deducted from the revenue) to actively transfer bikes among stations at any time. Each transfer could transport **N** bikes of a single **type** from station s_j to s_i through the shortest path. The tool should then adjust the inventory of s_j immediately, but adjust the inventory of s_i only when the transferred bikes arrive. The transfer fee shall be the per unit time transfer fee (to be specified in inputs) times the shortest travel time between s_i and s_j, regardless of **N** or **type**. Note that if a user is waiting at s_i for the requested type of bike to arrive, the rule of waiting fee shall apply.

The goal of the advanced version is to maximize the company revenue. Your task is to design the optimization algorithm. You are recommended to first analyze the rental history, then decide whether to let renters wait or to transfer bikes, and tailor optimal responses to rental requests such that the company revenue can be improved.

3. Input Format

There will be four input text files: "map.txt", "station.txt", "fee.txt" and "user.txt". There should be **a single space** between each symbol or number and a **newline** at the end of each line. All the time ranges from 0 to 1440 (minutes in a day).

A. "map.txt": The map provided by the rental company is an **undirected graph**.

Each line represents an edge between two stations s_i and s_j with an positive integer e_{s_i,s_j} indicating the travel time required to cross the edge:

$$s_i \; s_j \; e_{s_i,s_j}$$

For simplicity, we assume each stationId is encoded as a unique integer.

(An example of "map.txt")

```
1 2 25
2 3 60
```

B. "station.txt": The number of bikes ($b_{s_i,tp}$) initially available at a station s_i, $0 \leq b_{s_i,tp} \leq 100$.

The rental company has three types of bikes: electric, lady, and road. Each line indicates the initial inventory of a station:

s_i $b_{s_i,electric}$ $b_{s_i,lady}$ $b_{s_i,road}$

```
(An example of "station.txt")
1 6 3 4
2 30 100 9
3 0 10 10
```

The unique **bikeId** is then constructed according to the sequence number of the bikes initially deployed at the station $stationId$ using the following formula:

$stationId$ || (a two-digit sequence number starting from zero)

The "||" symbol means concatenation. For example, the first line of the above "station.txt" shows that initially at the station **1** there are 6 electric bikes, 3 lady bikes and 4 road bikes. Thereby, the **bikeId** for the electric bikes are numbered from 100 to 105, the lady bikes numbered 100 to 102, and the road bikes 100 to 103. Similarly, the lady bikes at the station 2, are numbered from 200 to 299, for a total of 100 lady bikes.

C. "fee.txt" specifies the rental fee schedule.

The first three lines specify the per unit rental time regular rate and discounted rate for each type of bike:

$bikeType$ $discountedRate$ $regularRate$

Thereafter, the three lines followed specify the per unit time waiting fee, the reduced rate for switching to other bike type, and the per unit time transferring fee.

```
// (An example of "fee.txt")
electric 30 40
lady 25 35
road 15 25
5          // rate of waiting fee
0.8        // the reduced rate for switching bike type
6          // rate of transferring fee
```

For the calculation in the basic implementation, ignore the last three lines.

D. "user.txt": Each line represents that either a user rents or returns a bike at what time.

rent stationIdRent *bikeType* userId timeRent

return stationIdReturn userId timeReturn

```
// (An example of "user.txt")
rent 3 lady 00002 0
rent 2 electric 00001 0
return 1 00001 35
```

```
return 1 00002 85
rent 3 electric 00003 90
return 1 00003 1440
```

The rental elapse time is $timeReturn - timeRent$, which shall be no smaller than the shortest transfer time. Also, for each rental request there shall be a corresponding return.

Note that the input from "user.txt" is only to describe the original schedule of renters. Under different policies and scenarios, the actual rent and return status may be different from "user.txt", your tool should conduct the accounting process according to the real status..

4. Output Format

Each execution of the tool shall output both the results of the basic implementation and the advanced version. The output files for the basic implementation are "part1_status.txt" and "part1_response.txt"; for the advanced version are "part2_status.txt" and "part2_response.txt".

A. "part1_status.txt" and "part2_status.txt": Output the final inventory of each station by first printing out "stationId:". The station shall be listed in an ascending order. Then sequentially list the inventory of each type of bikes in the sequence of the three types, electric, lady and road, by first printing out "bikeType: " in each category.

After each bikeType header, list the bike inventory in an ascending **bikeId** order. Leave blank after the bikeType header if the type ends up in shortage at the station.

For instance, the inventory of a station s_i should be:

s_i**:**

electric: $electirc\_bike\_ID_1$ $electirc\_bike\_ID_2$**…**

lady: $lady\_bike\_ID_1$ $lady\_bike\_ID_2$**…**

road: $road\_bike\_ID_1$ $road\_bike\_ID_2$**…**

After the inventory, print the total revenue from the rental run as the last line.

```
(An example of "part1_status.txt")
1:
electric: 100 101 102 103 104 105 200
lady: 100 101 102 300
road: 100 101 102 103
2:
……
3:
```

```
……
3525        // the last line is the total revenue
```

B. "part1_response.txt" and "part2_response.txt": Output the response to rental requests .

First copy each line from "user.txt" to the response file. Then each "rent" line is followed immediately with a line of a response to it. Also, insert any transfer decision as you see fit (as long as you keep each rental request and its response consecutively.) All output lines should be in chronological order.

As mentioned before, under different policies and scenarios, the actual rent and return status may be different from "user.txt". However, you should output the original rent and return line from "user.txt" regardlessly. Your tool should process the change of renter schedules internally. Furthermore, if there are muptiple users waiting or requesting rental at the same time, process their renting in the order the same as the input of "user.txt".

Your response shall be one of the following forms:

I. **accept**

// The renter gets a bike.

Note: The "accept" response is only valid when there is no shortage. If your tool accepts a rental request and there is no bike to give, your code would be considered incorrect.

II. **wait**

// Ask the renter to wait.

III. **discount *tp***

// Give the renter a discount to rent another type (***tp***) of bike.

Note: You can offer the renter discount even if there is no bike shortage in the advanced version.

IV. **reject**

// reject the rental request.

Note: You may deliberately reject a rental request for revenue consideration in the advanced version.

V. **transfer** s_j s_i ***tp*** **number time**

// at **time**, start transferring the **number** of type ***tp*** bikes from s_j to s_i.

Note: In the advanced version, you may make a transfer decision at any time.

```
(An example of "part1_response.txt")
rent 3 lady 00002 0
```

```
accept
rent 2 electric 00001 0
accept
return 1 00001 35
return 1 00002 85
rent 3 electric 00003 90
reject
return 1 00003 1440   // the original return of rejected rents should still be outputted
```

(An example of "part2_response.txt")

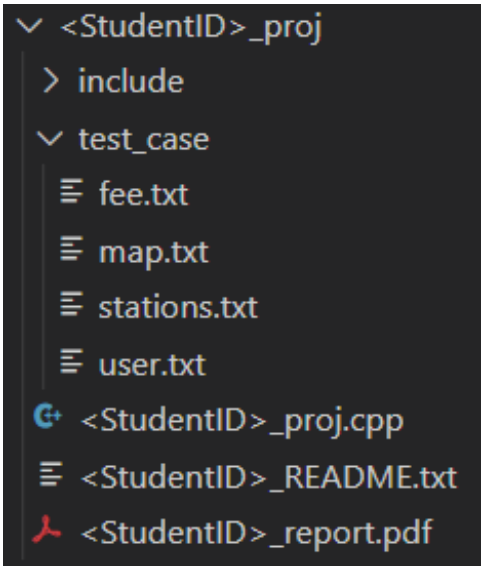
```
rent 3 lady 00002 0
accept
rent 2 electric 00001 0
accept
transfer 2 3 electric 1 10
return 1 00001 35
return 1 00002 85
rent 3 electric 00003 90
accept
return 1 00003 1440
```

Note that for the basic version, there are only two valid types of responses: "**accept**" and "**reject**".

5. Submission

Submit a single compressed zip file "**<StudentID>_proj.zip**" and submit it to the **eeclass** platform before the deadline. For example, if your StudentID is 109000000, the name of your zip file should be "109000000_proj.zip". Your zip file should contain only one subfolder named "**<StudentID>_proj**". All files of this project should be put in this subfolder.

A. "**<StudentID>_proj**" directory structure:



i. Put your main function in a cpp file named **"<StudentID>_proj.cpp"**.

ii. Put the output file in the same directory as the <StudentID>_proj.cpp.

iii. Put all other files to be included in the **"include"** folder.

iv. Expect test cases be put in the **"test_case"** folder.

v. Your report should be named **"<StudentID>_report.pdf"**.

vi. Optional **"<StudentID>_README.txt"** for any additional information.

6. Grading

A. Environment

i. Ubuntu 20.04

ii. Standard C++ 11

iii. Compiling:
"g++ -g *.cpp ./include/*.cpp -o <StudentID>_proj -std=c++11"

iv. The execution time for each test case is limited to at most **1** minutes.

B. Illegal implementations receive zero scores: You get no score if your program cannot compile or execute on the specified testing platform and command. **All data structures, except arrays (from C language), should be implemented by yourself** (std::string is allowed. Other STL data structures, such as std::array, std::vector, etc, are not allowed.) Note that our testing platform is a simple machine that supports only standard CPUs, supports no GPU nor other non-CPU instructions, and is not connected to the internet.

C. Part1 (60%)

i. Basic test cases (30%): TA will provide three open test cases. For each test case, if both the "part1_status.txt" and "part1_response.txt" are correct, you receive 10% credit.

ii. Advanced test cases (30%): TA will test with multiple hidden test cases. If there are total N hidden test cases and you pass p test cases, then you receive $30\% * \left(\frac{p}{N}\right)$ credit.

D. Part2 (20%)

i. Correctness: For each test case, the final bike distribution and revenue number in "part2_status.txt" should match your "part2_response.txt."

ii. Revenue maximization (20%): For each test case, If M students output correctly and **improve the final revenue** (versus part1's result), TA will rank the final revenue of these M students from the highest to the lowest. Suppose your final revenue ranks at the kth place; you will receive $\left(\frac{20\%}{N}\right) * \left(\frac{M+1-k}{M}\right)$ credit, where N is the total number of test cases.

E. Report (10%)

Your project report is recommended to follow this outline:

1) Part1 Design
    1-1) Program Flow Chart
    1-2) Detailed Description
2) Part2 Design
    2-1) Detailed Description of how you choose your responses

Note: The project report is limited to 10 pages.

Note: Your report can be either in Chinese, in English, or mixed.

F. Demo (10%)

i. Each person should reserve a 15-min demo with TA.

ii. Be prepared to explain your implementation in detail to TAs.

iii. During the 1-on-1 demo, TAs will download your code from **eeclass** and run it on TAs' machines.

Etiquette

a. Do not plagiarize others' work, or you will fail this course.

b. No acceptance of late homework.

c. Please frequently check the class website announcements for possible updates to the project.