Data Structure Final Project
Maximize Revenue for NTHU-Bike Company

Last Update: 2022/11/08

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 system to improve its workflow management. The company will provide a location map of available bike stations, initial bike allocations, and a rental fee schedule. The tool calculates the company's expected revenue and the final bike distribution for each scenario. Then, the tool shall also figure out how to earn more revenue if the company adopts different management policies.

Each data entry of the user input file is of the following form:

User_Id   Accept_Bike_Type   Start_Time   End_Time   Start_Point   End_Point

which describes that a user with **User_Id** wants to rent a bike of type **Accept_Bike_Type** from station **Start_Point** at **Start_Time** and to arrive at the destination station **End_Point** before **End_Time**. The rental fee is the **Rental_Price** multiplied by the rental time (your code should round down the rental fee to an integer). The rental time is assumed to be proportional to the shortest distance between the station **Start_Point** and **End_Point**, and the arrival time must be less than **End_Time**. If one of the requests cannot be fulfilled, e.g., no bike fits **Accept_Bike_Type** at the **Start_Point**, or the arrival time is later than **End_Time**, then the tool should reject the rental request.

Suppose the shortest path between a **Start_Point** s_1 and an **End_Point** s_2 is d_{s_1, s_2}, the formula of the rental fee is:

$$\text{Rental fee} = \lfloor \textbf{Rental\_Price} * d_{s_1, s_2} \rfloor$$

The "NTHU-Bike " is known for its rich choice of bike types and affordable prices. One of the company's policies is that the **Rental_Price** of every bike is discounted after each rental (to account for depreciation). For example, suppose the initial rental price of a new bike A is \$1 per minute, and the discount price is \$0.1 per rental. User Alice rents the brand new bike A for 200 minutes, and the company will charge Alice \$200. After that, another user, Bob, rents the (not brand new) bike A again, then his rental price is \$1 - \$0.1 = \$0.9 per minute. After Bob, if user Charlie also rents bike A, the price drops again to \$0.9 - \$0.1 = \$0.8 per minute. One thing to note is that once the rental count of a bike reaches a predetermined limit, the bike would be “retired” (not for rental anymore) for a better user experience. The rental count limit of each bike is defined in a file, which we will describe in the input format section. The formula of the per-minute rental price of each bike is shown below for your reference:

Actual Rental_Price = **Initial Rental_Price** - (depreciation_discount_price * **Rental_Count**)

"NTHU-Bike" also offers a popular feature called "free bike transfer service." The company transfers the rental bike to the user-designated station free of charge. You may apply this policy for optimal revenue. Note that the “free bike transfer service” application does not increase the Rental_Count of a bike.

For simplicity, we assume the biking speed and free bike transfer speed is always one unit distance per minute for all users and bike types, and for each rental request you shall find the shortest path to calculate the rental fee.

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

Example 1:

- U2 B1 23 89 S1 S3

The above line means user U2 requests to rent a type B1 bike from station S1 to S3, and user U2 will arrive at station S1 at time 23 and requires to arrive at station S3 before time 89.

If there is a bike with **Bike_Id** 17 and its **Bike_Type** is B1 at station S1 from time 23 to 89, and user U2 can arrive at station S3 before time 89, this rental request shall be accepted.

Once the rental request is accepted, your tool calculates the rental fee. Suppose that the bike is available at time 31 and the current rental price for the **Bike_Id** 17 is $1.7 per minute, and the shortest path d_{s_1, s_2} between S1 and S3 takes 25 minutes to travel, then user U2 shall be charged for $\lfloor 1.7*25 \rfloor = \lfloor 42.5 \rfloor =$ \$42. After user U2’s rental, the bike is returned to station S3 at time 56 (31+25), while the rental count of **Bike_Id** 17 increments by 1, and the total company revenue increases by $50. Note that no other user can rent **Bike_Id** 17 during U2’s rental time (31~56).

Example 2:

- U3 B1 23 89 S1 S3

The above line means user U3 requests to rent a type B1 bike from station S1 to S3, and user U3 will arrive at station S1 at time 23 and requires to arrive at station S3 before time 89.

If there is no B1 (**Accept_Bike_Type**) type bike at station S1 during time 23 to 89, or the user U2 cannot arrive at station S3 before time 89, this rental request shall be rejected.

For this project, you should implement two versions of the tool. We have pre-set rules for rejecting or accepting a user request for the first basic implementation. The rules are

I. Free bike transfer service is forbidden.

II. Please sort the user request first in the ascending order of **Start_time** and if you encounter the same **Start_time** then follow the ascending order of **User_ID**.

III. If no bike is available at **Start_time**, the rental request should be rejected with no charge. (we assume users do not wait for bikes.)

IV. If several bikes are available, pick the bike with the highest Rental price.

V. If several bikes are available and with the same highest price, pick the bike with the smallest **Bike_ID**. (every bike is assigned a unique ID.)

VI. If several users arrive at the station at the same time, handle the rental request from the smaller **User_ID** first.

In the advanced version, the tool should maximize the company's revenue. It may help if you decide when to call "free bike transfer service" and whether to reject some rental requests and thus tailor a sequence of optimal decisions to the user rental requests. For the advanced version of the implementation, the requirements are

I. With the same user requests, the final total revenue should be more than the basic version.

II. Free bike transfer service is allowed.

III. If no bike is available at **Start_time**, users could wait for bikes. Make sure that the user still needs to arrive at **End_Point** before **End_Time**.

IV. You could reject some requests or pick any available bike (of the same request type) as you see fit.

3. Input Format

There will be four input text files: "map.txt," "bike.txt," "user.txt," and "bike_info.txt." Use a single space to separate each symbol or number and a "new line" character at the end of each line. No empty lines are allowed.

A. **map.txt:**

The map provided by the rental company is an **undirected graph**.

In map.txt, each line represents an edge between two stations s_i and s_j with a positive integer d_{s_i, s_j} indicating the travel time required to cross the edge:

```
(An example of "map.txt")
S1 S2 25
S2 S3 60
…
```

Note: Station_Id does not always start from 0 or in increasing order. You should carefully construct the map.

B. bike.txt:

The following terms in each line indicate each bike's initial status:

1. **Bike_Type:**

   The type of bike, string type
2. **Bike_Id:**

   The id of the bike, integer type
3. **Station_Id:**

   The id of the station where the bike is, string type
4. **Rental_price:**

   The current rental price of the bike. The unit is "dollars per minute."
5. **Rental_Count:**

   The number of times this bike has been rented out before. The bike should be retired if the Rental_Count reaches or exceeds the rental count limit (defined in **bike_info.txt**.)

```
(An example of "bike.txt")
// Bike_Type, Bike_Id, Station_Id, Rental_Price, and Rental_Count
B0 0 S1 10 0
B0 2 S3 10 0
B1 3 S2 20 0
…
```

Note:

1. Each Bike_Type, Bike_Id, and Station_Id is of a random name or number.
2. Calculate each bike's rental fee using the formula mentioned above.
3. Once a bike is retired, users should not rent the bike.

C. **user.txt:**

In user.txt, the following terms in each line specify the rental requests of users:

1. **User_Id:**

   The id of the requesting user, string type

2. **Accept_Bike_Type:**
   The acceptable bike types of the user, string type. If there are multiple choices, use "," to separate between choices
3. **Start_Time:**
   The time the user plans to pick up the bike at the Start_Point, integer type
4. **End_Time:**
   The deadline the user must arrive at the End_Point (user shall arrive before End_time), integer type
5. **Start_Point:**
   Where the user will pick up the bike, string type
6. **End_Point:**
   The target location that the user is to return the bike, string type

```
(An example of "user.txt")
// User_Id, Accept_Bike_Type, Start_Time, End_Time, Start_Point, End_Point
U1 B1,B2 3 100 S1 S3
U2 B1 0 7 S3 S2
U3 B2 1 100 S1 S3
U4 B1 1 5 S3 S2
…
```

Note:

1. The numbers in **User_Id** and **Accept_Bike_Type** do not always start from 0 or in increasing order.
2. The minimum **Start_Time** is 0, and the largest **End_Time** is 1440. (A day has 1440 minutes.)
3. Each **User_Id** occurs only once.
4. We limit the maximum number of users to 100,000.

D. **bike_info.txt:**
The file defines detailed bike rental parameters, including depreciation discount price, rental count limit to retire, and the initial rental price of each bike type.

```
(An example of "bike_info.txt")
//depreciation discount price
0.5
//the rental count limit
10
// Bike_Type, Initial_Price(Rental_count = 0)
B1 20
B2 26
…
```

Note: The depreciation discount price will be in floating point type.

4. Output Format

Each execution shall output the revenue result, the status of all bikes (which station each bike ends up at), and all responses to all user requests. The output files are "user_result.txt," "station_status.txt," and "transfer_log.txt."

A. “station_status.txt”: Output the final bike inventory, first in the ascending order of each station and within each station in the ascending order by bike_id

Each line should contain the **station_id**, **bike_id**, **bike_type**, **rental_price**, and **rental_count**, where rental_count represents the number of times this bike has been rented out before, and rental_price represents the current rental price of the bike.

Here is an example of “station_status.txt.”

```
// station_id, bike_id, bike_type, rental_price, and rental_count
S2 0 B1 9.95 1
S2 1 B1 20 0
S3 2 B1 30 0
```

B. “user_result.txt”: The responses to the user requests.

Output the status of accepted or rejected of each user request and also the revenue collected from the request. Each record of "User_result.txt" has seven columns, and the definition of these columns is listed below:

1. **User_Id**:

   The id of the user

2. **AcceptOrNot**:

   The decision to the user request: 0 means “reject,” 1 means “accept.”

3. **Bike_Id**:

   The id of the bike

4. **Bike_Start_Time**:

   The bike rental start time. If the system rejects the rental, put 0 (number zero) here.

5. **Bike_End_Time**:

   The bike rental end time. If the system rejects the rental, put 0 (number zero) here.

6. **Revenue**:

   The revenue earned by the company from this user request. If the system rejects the rental, put 0 (number zero) here. Remember to round down your result.

(An example of “user.txt”)

```
U1 B1,B2,B3 0 1 S1 S2
U2 B1,B2,B3 3 6 S1 S2
```

(An example of “user_result.txt”)

```
U1 0 0 0 0 0  // if the system rejects the request
U2 1 0 3 4 100  // if the system accepts the request
```

C. “transfer_log.txt” :

The log records the bike transfers. This log contains the bike movement originating from users’ bike ridings and bike transfers due to the "free bike transfer service" by the company. Each line of the "transfer_log" has six columns, as listed below:

1. **Bike_Id**:

   The id of the bike
2. **Start_Station**:

   The start point of the bike
3. **End_Station**:

   The endpoint of the bike
4. **Transfer_Start_Time**:

   The bike transfer start time
5. **Transfer_End_Time**:

   The bike transfer end time
6. **Bike_Rider**:

   Represent who caused the bike transfer/movement. If a user rides the bike, insert the user_id here. If the bike is transferred by the “NTHU-bike,” insert -1.

(An example of “user.txt”)

```
U1 B1,B2,B3 0 1 S1 S2
U2 B1,B2,B3 3 6 S1 S2
```

(An example of “user_result.txt”)

```
U1 0 0 0 0 0 0 // if the system rejects the request
U2 1 0 3 4 100 2 // if the system accepts the request
```

(An example of "transfer_log.txt")

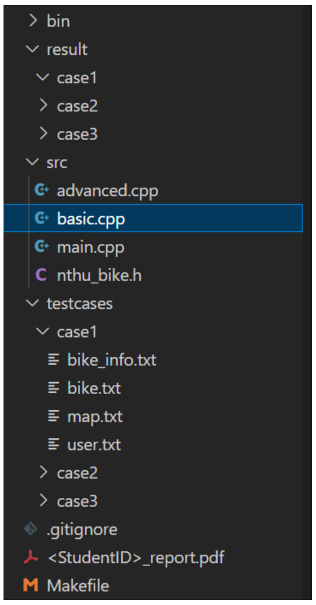
```
0 S1 S2 3 4 U2
```

5. Submission

Submit a single compressed zip file "**<StudentID>_proj.zip**" and upload 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 a subfolder named "**<StudentID>_proj.**" All files of this project should be put in this subfolder.

For the final project, you are to implement two versions of code, a basic version, and an advanced version. TAs will provide a template folder including test cases, a makefile, and a designated file structure. Write your basic version at "src/basic.cpp", and your advanced version at "src/advanced.cpp".

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



```
> bin
v result
  v case1
  > case2
  > case3
v src
  advanced.cpp
  basic.cpp
  main.cpp
  nthu_bike.h
v testcases
  v case1
    bike_info.txt
    bike.txt
    map.txt
    user.txt
  > case2
  > case3
.gitignore
<StudentID>_report.pdf
Makefile
```

i. Write the "basic.cpp," "advanced.cpp," and all related code in the "src" folder.

ii. TA provides a Makefile template. If needed, you may modify the Makefile and explain how you use it to compile the codes in your report.

iii. Generate an executable file into the "bin" folder.

iv. Three open test cases are provided in "testcases" folder.

v. Executed results should be put in the "result" folder.

vi. Name your report "<StudentID>_report.pdf".

B. How to execute the program?

i. Compiling:

```
g++ -g -std=c++11 -o ./bin/main ./src/*.cpp
```

The above instruction will compile all the files in “src” folder and generate an executable file “main ”in the “bin” folder

ii. Execution:

```
./bin/main <case> <version>
```

The above instruction needs to specify the parameters (case and version) to run the compiled executable file. The options of parameters are listed below:

- case: case1, case2, case3
- version: basic, advanced

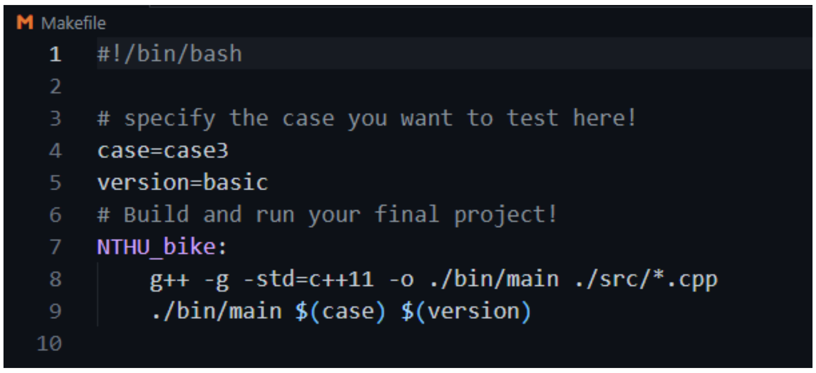
For instance, to test the basic version on case1, type “`./bin/main case1 basic`”; To test the advanced version on case1, type “`./bin/main case1 advanced`”

iii. Makefile

By specifying the case and version in Makefile, type “make NTHU_bike,” and then Makefile will compile and execute the program.

There are two parameters in Makefile that you can modify at lines 4 and 5 to test your program, and the options of parameters are listed below:

- case: case1, case2, case3
- version: basic, advanced



```
1   #!/bin/bash
2
3   # specify the case you want to test here!
4   case=case3
5   version=basic
6   # Build and run your final project!
7   NTHU_bike:
8       g++ -g -std=c++11 -o ./bin/main ./src/*.cpp
9       ./bin/main $(case) $(version)
10
```

To learn more about Makefile, you can refer to this link

6. Grading

A. Environment

i. Ubuntu 20.04

ii. Standard C++ 11

iii. The execution time for each test case is limited to **one** minute.

B. Incorrect 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::queue, std::vector, etc., are not allowed.) Note that our testing platform is a straightforward machine that supports only standard CPUs, supports no GPU or other non-CPU instructions, and is not connected to the internet.

C. Testcase (80%)

   i. Correctness (60%)
      - There are 3 open test cases and 7 hidden test cases.
      - 6 points for each test case. TA will run the basic version to test your result.

   ii. Performance - Revenue (20%)
      - It is graded by the revenue of your program on 3 open test cases and 7 hidden test cases. TA will run the advanced version to get your revenue result. Incorrect results will not earn any points.
      - 2 points for each test case. Suppose M students output correctly and **improve the final total revenue** (versus the basic version result). In that case, TA will rank the final revenues of these M students from the highest to the lowest. If your revenue ranks at the k-th place and there are a total of M students, you will receive 2% * ($\frac{M+1-k}{M}$) points.

D. Report (10%)

Your report must contain the following contents, and you can add more.

   i. Your name and student ID
   ii. How to compile and execute your program and give an execution example.
   iii. The details of your data structures. What data structures did you use, and how did you implement those data structures.
   iv. The details of your algorithm. You could use flow chart(s) and/or pseudo code to help elaborate your algorithm.
   v. [Optional] If you have any feedback, please write it here. Such as comments for improving the spec of this assignment, etc.

Note: The project report is limited to 10 pages.

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

E. 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.