

Data Structure Final Project Report

姓名：高曼琄

學號：111065538

- I. How to compile and execute your program and give an execution example.
 - A. 在 Makefile 裡設定 case、version (line 4, 5)，下圖範例為 case1, basic
 - B. 確認目前檔案路徑到 111065538_proj，在 terminal 輸入: make NTHU_bike
 - C. 輸出三個 txt 檔 (user_result.txt, station_status.txt, transfer_log.txt) 到 result 資料夾下對應的 case 資料夾內

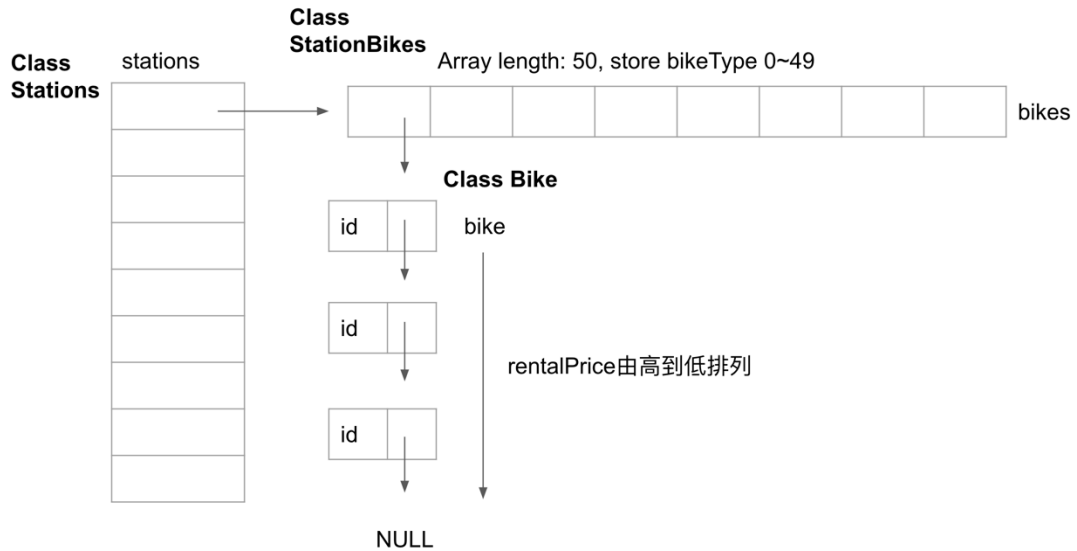
The screenshot shows a VS Code workspace for a project named '111065538'. The Explorer panel on the left shows the project structure with folders like 'bin', 'main.dSYM', 'info.txt', 'main', 'verifier', 'result', and 'case1'. The Source Explorer shows files like 'transfer_log.txt', 'user_result.txt', 'b_user.cpp', 'nthu_bike.h', 'b_station.cpp', 'main.cpp', 'b_graph.cpp', 'validation.cpp', and 'user_result.txt'. The Makefile is open in the editor, showing the 'NTHU_bike' target. The terminal window shows the command 'make NTHU_bike' being executed, resulting in the compilation of 'main.cpp' and the execution of the program. The output shows the program running with 'case1' and 'basic' version, displaying total revenue and user result match status.

```
Makefile
1 #!/bin/bash
2
3 # specify the case you want to test here!
4 case=case1
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
```

```
result > case1 > user_result.txt
1 U0 0 0 0 0 0
2 U1 1 18 564 606 1050
3 U2 1 28 246 397 4454
4 U3 0 0 0 0 0
5 U4 0 0 0 0 0
6 U5 1 5 440 508 1462
7 U6 0 0 0 0 0
8 U7 0 0 0 0 0
9 U8 0 0 0 0 0
10 U9 1 23 81 232 4454
11 U10 1 3 23 174 3246
12 U11 0 0 0 0 0
13 U12 1 19 941 1079 3174
14 U13 0 0 0 0 0
15 U14 0 0 0 0 0
16 U15 1 16 4 159 3642
17 U16 1 18 621 695 1813
18 U17 0 0 0 0 0
19 U18 1 1 357 453 864
20 U19 1 20 109 205 1920
21 U20 0 0 0 0 0
22 U21 1 3 811 962 3171
23 U22 1 26 1022 1064 756
```

```
terminal
(base) shuangjietizihuideMacBook-Pro 111065538_proj % make NTHU_bike
g++ -g -std=c++11 -o ./bin/main ./src/*.cpp
./bin/main case1 basic
You have set case1 as your test case:
running basic currently
start your basic version of data structure final from here!
total revenue for basic version: 47437
user_result.txt match!
finished computation at Wed Jan 4 23:30:20 2023
elapsed time: 0.016963s
(base) shuangjietizihuideMacBook-Pro 111065538_proj %
```

- II. The details of your data structures. What data structures did you use, and how did you implement those data structures.
 - A. 用 Adjacency Matrix：shortestPath[][]存任兩個 station 之間的距離
 - B. 用 Array of pointers, Linked list 存每個 station 裡 bikes 的情況，用 bike types 分類
 1. 用 Stations* stations 物件把站點相關的所有變數、function 包裝起來
 2. stations 物件底下用 StationBikes* station[NUM_STATIONS]把每個站點 bikes 配置資訊包裝起來
 3. StationBikes* station 物件裡有 Bike* bike[BIKETYPES]存放某站點內，所有 bikes 依照其 bike type 分類，同一 type 的 bikes 會用 linked list 串起來
 4. Bike* bikes[i]存 bike 相關資訊，例如：Bike type、Bike Id、Station Id、rentalPrice、rentalCount、指向下一個同 type 的 bike 的指標 (Bike* sameTypeNext)



- C. 用 array of pointers of class User 存 users 資訊，例如：user id、accept bike Type、start time、end time、start point、end point
- D. Output: 用 2-D int array 存 user_result，用 1-D Bike* array 存 station_status (按照 station Id、Bike Id 排列)
- E. 以下說明 Sorting method、finding all-pair shortest path 的實作，而其目的會在 Algorithm part 說明

1. SortUsers()用 **Bucket Sort** 實作

```

49 // sort users by startTime
50 // bucket sort since startTime, endTime at most 4 digits(1440)
51 // dealt with smaller userId first when startTime the same with SortUsersById()
52 void SortUsers(User** users, const int numOfUsers){
53     // sort users with bucket sort
54     User* bucket[10][numOfUsers]; // time is decimal, so base = 10 as num of rows
55     // in SortUsers(), the array "link" stores every round result
56     // need to initialize, or segmentation fault when link[j]->startTime (since no object and not NULL)
57     User* link[numOfUsers];
58     //put users into link
59     for(int j=0;j<numOfUsers;j++){
60         if(users[j]) {
61             link[j] = users[j];
62         }
63         else break;
64     }
65     // pow used to extract each digit from least significant digit (LSD)
66     int pow=1;
67     // 4 digits, so 4 rounds
68     for(int i=0;i<4;i++){
69         //clean the bucket
70         for(int j=0;j<10;j++){
71             for(int k=0;k<numOfUsers;k++){
72                 bucket[j][k] = NULL;
73             }
74         }
75         // put users into buckets
76         pow *=10;
77         for(int j=0;j<numOfUsers;j++){
78             int bucketNum = (link[j]->startTime)*10/pow%10;
79             int idx2=0;
80             while(bucket[bucketNum][idx2]){
81                 idx2++;

```

```

82     }
83     bucket[bucketNum][idx2] = link[j];
84 }
85 // store users' new order to array "link"
86 int idx2=0;
87 for(int j=0;j<10;j++){
88     for(int k=0;k<numOfUsers;k++){
89         if(bucket[j][k]){
90             link[idx2++] = bucket[j][k];
91         }
92     }
93 }
94 }
95 //put link (sorted result) back to users
96 for(int j=0;j<numOfUsers;j++){
97     if(link[j]) users[j] = link[j];
98     else break;
99 }
100
101 }

```

2. SortUsersById()、SortBikeById()用 Insertion Sort 實作

```

103 // after sort user.txt by startTime, sort by userID if two has the same startTime
104 // use insertion sort since most of entries are already sorted, nearly O(n) time
105 void SortUsersById(User** users,const int numOfUsers){
106     for(int i=1;i<numOfUsers;i++){
107         int target_id = stoi((users[i]->userId).substr(1));
108         User* target = users[i];
109         int idx = i-1;
110         // only if starttime is the same will they compare userid
111         while(idx>=0 && users[idx]->startTime == target->startTime \
112             && stoi((users[idx]->userId).substr(1)) > target_id){
113             users[idx+1] = users[idx];
114             idx--;
115         }
116         if(idx < 0){
117             users[0] = target;
118         }
119         else{
120             users[idx+1] = target;
121         }
122     }
123 }

```

```

200 // iterate through stations to collect all bikes and sort by bikeid within same station
201 // use insertion sort since already sorted by stations, most of them only move a little bit
202 // preparation for station_status.txt
203 void SortBikeById(Stations* stations){
204     int idx=0;
205     // iterate through stations to collect all bikes
206     for(int i=0;i<NUM_STATIONS;i++){
207         // assume at most 50 types of bike
208         for(int j=0;j<BIKETYPES;j++){
209             Bike* sameTypeCur = stations->station[i]->bikes[j];
210             if(sameTypeCur==NULL){
211                 continue;
212             }
213             while(sameTypeCur){
214                 stationStatus[idx++] = sameTypeCur;
215                 sameTypeCur = sameTypeCur->sameTypeNext;
216             }
217         }
218     }
219     // insertion sort
220     // iterate through all bikes
221     for(int i=1;i<NUM_BIKES;i++){
222         Bike* bikeCur = stationStatus[i];
223         int idx = i-1;
224         // find the correct place for bikeCur
225         while(idx>=0 && bikeCur && bikeCur->stationId == stationStatus[idx]->stationId \
226             && bikeCur->bikeId < stationStatus[idx]->bikeId){
227             // if station id the same and bikeCur id is smaller, move the compared bike one entry back
228             stationStatus[idx+1] = stationStatus[idx];
229             idx--;
230         }
231         // put bikeCur int the right place
232         stationStatus[idx+1] = bikeCur;
233     }
234 }

```

3. shortestPath[][]用 FloydWarshall()找出

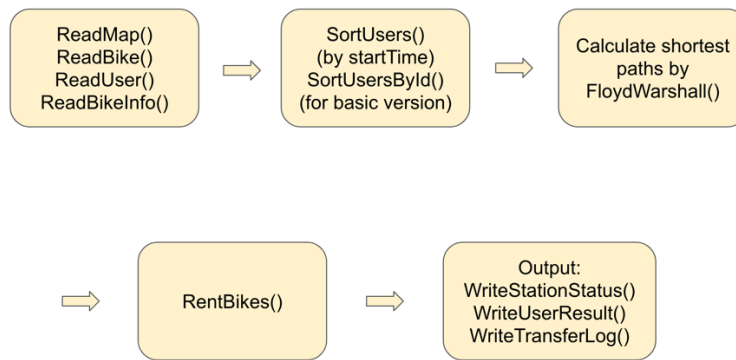
```

107 void Graph::FloydWarshall(){
108     for(int k=0;k<numOfStation;k++){
109         for(int i=0;i<numOfStation;i++){
110             for(int j=0;j<numOfStation;j++){
111                 // be careful with overflow
112                 if(shortestPath[i][k] != INT_MAX && shortestPath[k][j] != INT_MAX \
113                     && shortestPath[i][j] > shortestPath[i][k]+shortestPath[k][j]){
114                     shortestPath[i][j] = shortestPath[i][k]+shortestPath[k][j];
115                 }
116             }
117         }
118     }
119     //cout << "shortestPath[9][1]: " << shortestPath[9][1] << endl;
120 }

```

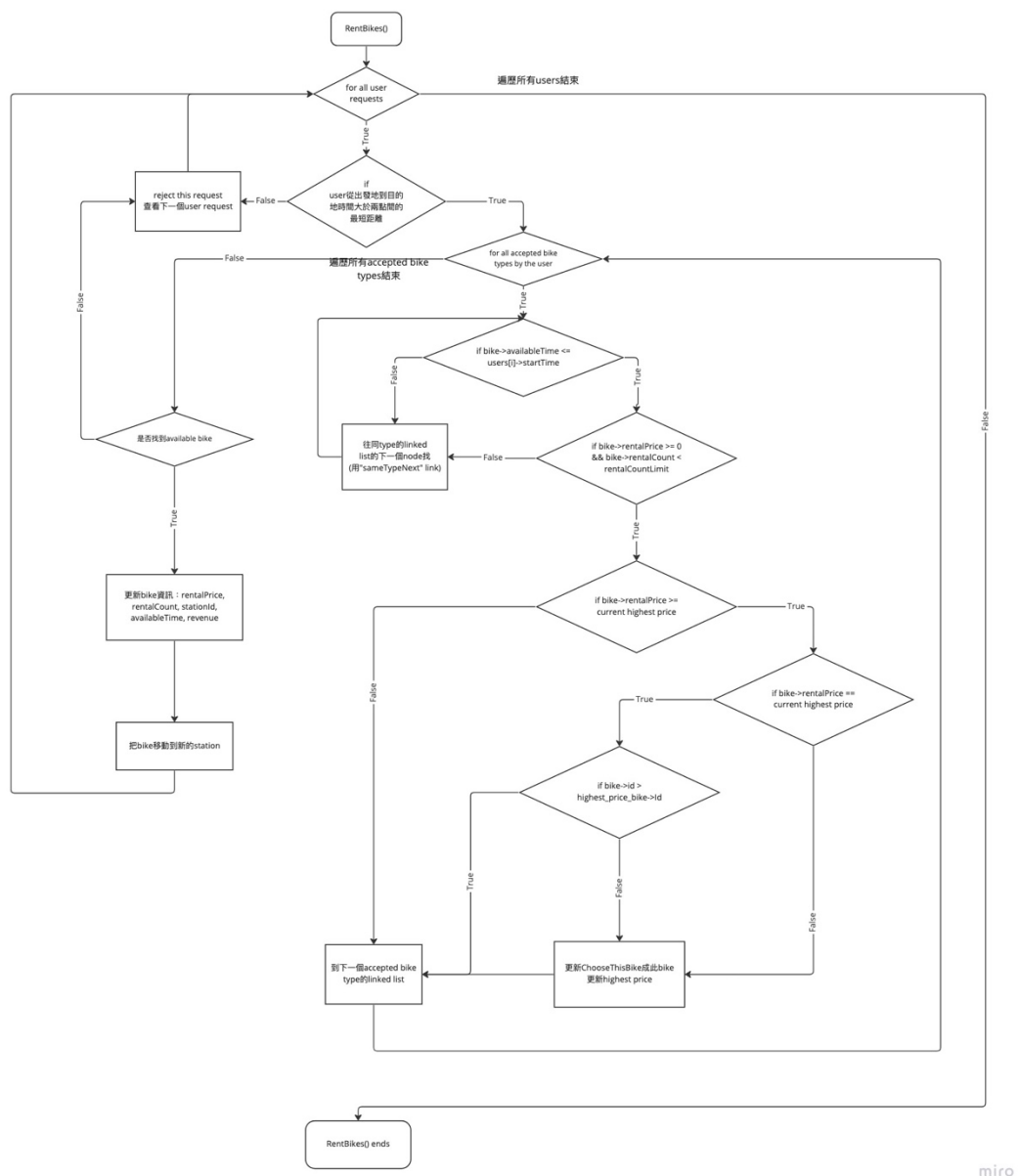
III. The details of your algorithm. You could use flow chart(s) and/or pseudo code to help elaborate your algorithm.

A. Overview:



1. ReadBike()：read bike 時呼叫 AddBike()，AddBike()會 new 一個 Bike 型別的物件，先看放在哪個 station，再依其 bike type 分類，同 type 的 bikes 用 linked list 連接 (用來連接的物件指標：Bike* sameTypeNext)，並且依照 rentalPrice 高到低排列
2. ReadUser()、SortUsers()、SortUsersById()：
 - a. SortUsers()用 **Bucket Sort** 實作，因為用 startTime 排，而 startTime 已知最大值為 1440，僅 4 位數，Bucket Sort 只需 4 回合，所以採 Bucket Sort 已達到 $O(n)$ time
 - b. SortUsersById()用 **Insertion Sort** 實作，因為此 function 是針對 startTime 相同的 user 需按照 userId 小到大排列，大部分已排好，比較不會有大範圍的資料搬移，所以用 Insertion Sort
 - c. basic version：user 按照 startTime 由小到大排，若 startTime 相同，則用 UserId 由小到大排
 - d. advance version：user 按照 startTime 由小到大排，若 startTime 相同，則用 endTime 由小到大排，讓要比較早到達目的地的 user 先借，目的是希望能提高總共 accept 的 request 數量
3. FloydWarshall()：把任兩點最短距離存入 Adjacency Matrix “shortestPath[NUM_STATIONS][NUM_STATIONS]”
4. RentBikes()：實作租借 bike 之邏輯，包含判斷 request 是否接受、將 bike 從出發站點移動到目的站點，兩個 version 分開討論
 - a. 判斷 request 接受的條件：
 1. Basic version：
 - a. user 從出發地到目的地的時間需大於兩點之間的最短距離： $users[i] \rightarrow endTime - users[i] \rightarrow startTime > shortestPath[startPoint][endPoint]$
 - b. user 可接受的 biketypes 中，要有 available 的 bike，所以遍歷 user 出發地所有可接受的 biketypes： $while(users[i] \rightarrow acceptBikeType[j] != "") \{$

- i. $\text{bike} \rightarrow \text{availableTime} \leq \text{users}[i] \rightarrow \text{startTime}$
- ii. rentalPrice 要大於等於 0, $\text{rentalCount} < \text{rentalCountLimit}$
- iii. 不斷更新有最高 rentalPrice 的 bike 物件和最高 rentalPrice : chooseThisBike 、 highestPrice
 - a. 如果有兩 bike 的 rentalPrice 都是最高價格，則選 userId 較小者
- iv. 每次在 linked list 間移動都要記錄目前 $\text{bike}(\text{sameTypeBikeCur})$ 的前一台 $\text{bike}(\text{prevBike})$ ，因為要記錄 chooseThisBike 的前一台 $\text{bike}(\text{prevChosenBike})$ ，才能在 chooseThisBike 從 linked list 刪除，並移動到目的站點時，把 chooseThisBike 的前一台 bike 和後一台 bike 連接起來
- c. 如果遍歷完 user 所有可接受的 biketypes 仍找不到 chooseThisBike ，表示沒有 bike 是 available，所以 request rejected
- d. 如果有找到 chooseThisBike ，則要移動 bike 並更新 bike 資訊，包含 stationId , rentalPrice , rentalCount , availableTime , 此次租借產生的 revenue
 - i. 移動 bike 方式：用更新後的資訊 $\text{AddBike}()$ 產生新物件，並 delete 原物件(chooseThisBike)
 - ii. $\text{availableTime} = \text{users}[i] \rightarrow \text{startTime} +$ 出發站點到目的站點的最短距離 (用 $\text{users}[i] \rightarrow \text{startTime}$ 計算而不是 bike 原本的 availableTime 是因為 bike 變成 available 不代表馬上會被借走)

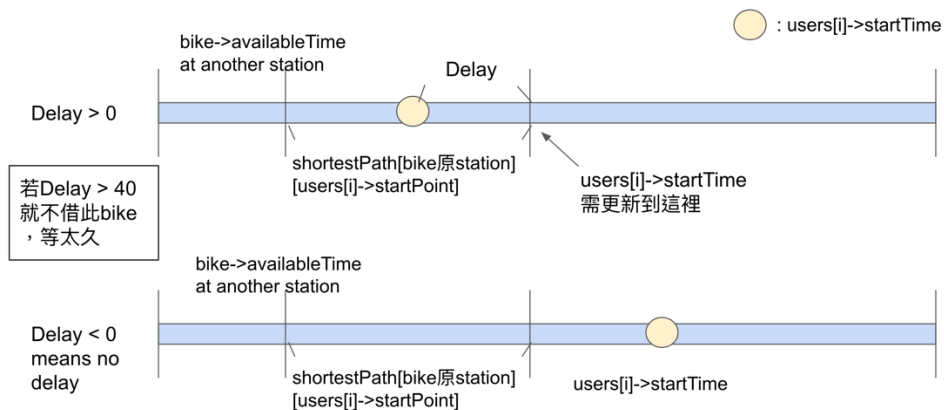


2. Advance version (與 basic version 不同處用紅色標示)：

- user 從出發地到目的地的時間需大於兩點之間的最短距離：

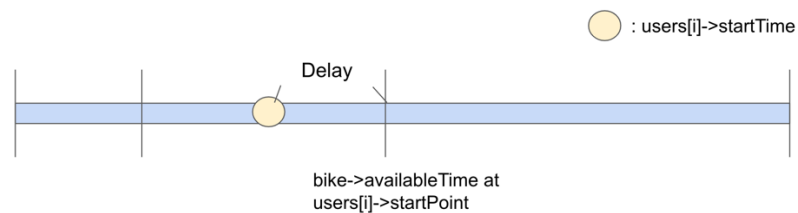
$$users[i] \rightarrow endTime - users[i] \rightarrow startTime > shortestPath[startPoint][endPoint]$$
- 為了找其他 station 可能有 rentalPrice 更高的 bike 可以借，所以遍歷所有 stations，每個 station 再遍歷所有 user 接受的 biketypes
 - 計算 $delay = bike \rightarrow availableTime + free\ transfer\ 花\ 的\ 時間 - users[i] \rightarrow startTime$
 - 檢查 $delay \leq 40 \ \&\& \ user\ 從\ 開始\ 到\ 結束\ 時長 \geq \ 最短\ 距離 + delay$

Delay = bike->availableTime + free transfer花的時間 - users[i]->startTime



若不用transfer，user一樣最多等40秒

Delay = bike->availableTime + free transfer花的時間 - users[i]->startTime
 此時free transfer花的時間 = 0
 Delay <= 40 才考慮藉此bike

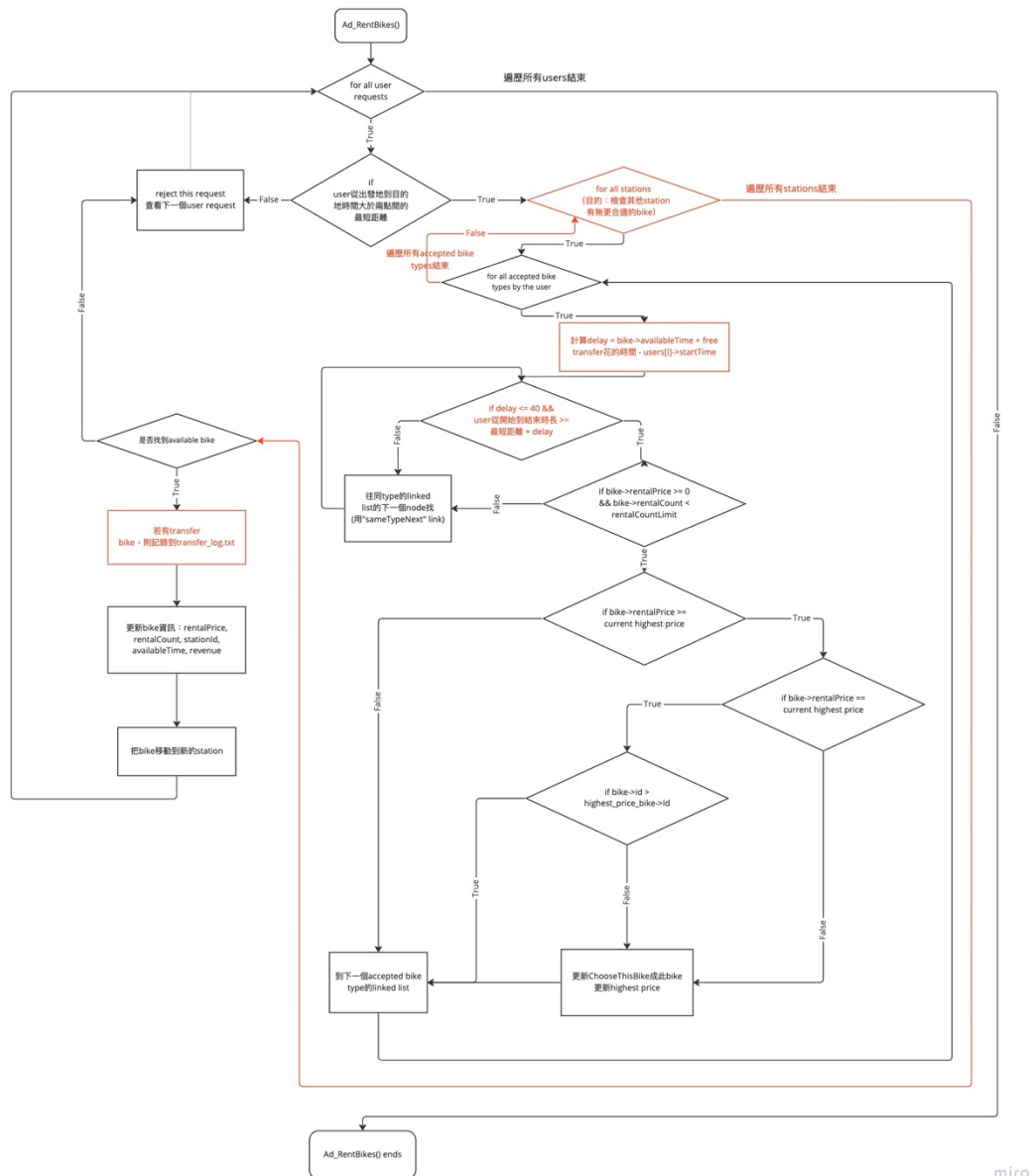


- iii. rentalPrice 要大於等於 0，rentalCount < rentalCountLimit
- iv. 不斷更新有最高 rentalPrice 的 bike 物件和最高 rentalPrice：chooseThisBike、highestPrice
 - a. 如果有兩 bike 的 rentalPrice 都是最高價格，則選 userId 較小者
- v. 每次在 linked list 間移動都要記錄目前 bike(sameTypeBikeCur)的前一台 bike(prevBike)，因為要記錄 chooseThisBike 的前一台 bike(prevChosenBike)，才能在 chooseThisBike 從 linked list 刪除，並移動到目的站點時，把 chooseThisBike 的前一台 bike 和後一台 bike 連接起來
- c. 如果遍歷完 user 所有可接受的 biketypes 仍找不到 chooseThisBike，表示沒有 bike 是 available，所以 request rejected
- d. 如果有找到 chooseThisBike，則要移動 bike 並更新 bike 資

訊，包含 stationId, rentalPrice, rentalCount, availableTime, 此次租借產生的 revenue

- i. 移動 bike 方式：用更新後的資訊 AddBike() 產生新物件，並 delete 原物件(chooseThisBike)
- ii. $\text{availableTime} = \text{users}[i] \rightarrow \text{startTime} + \text{出發站點到目的站點的最短距離}$

e. 若有 free bike transfer，則記錄到 transfer_log.txt



5. Output：用 user_result[][], station_status[] 紀錄所需資訊，transfer_log.txt 跟 user_result.txt 共用 user_result[][] (因為有許多重複資訊，但在 advance version 中 transfer_log 會在 RentBike () 中直接輸出，不和 user_result 共用)

a. WriteStationStatus() 前須先呼叫 SortBikeById()：

1. 用意：因為 station 裡的 bike 是按照 biketypes 排列，biketypes 內按照 rentalPrice 由高到低排列，並不是按照 bikeId 排列，與 station_status.txt 排列方式不同，因此實作 SortBikeById()
2. SortBikeById()用 **Insertion Sort** 實作，因為 Bike 按照 stations 順序取出時，station 順序正確，只是要重新排列同 station 內的 bikes，比較不會有大範圍的資料搬移，因此用 Insertion Sort