Data Structure Final Project Report

姓名：高曼瑄

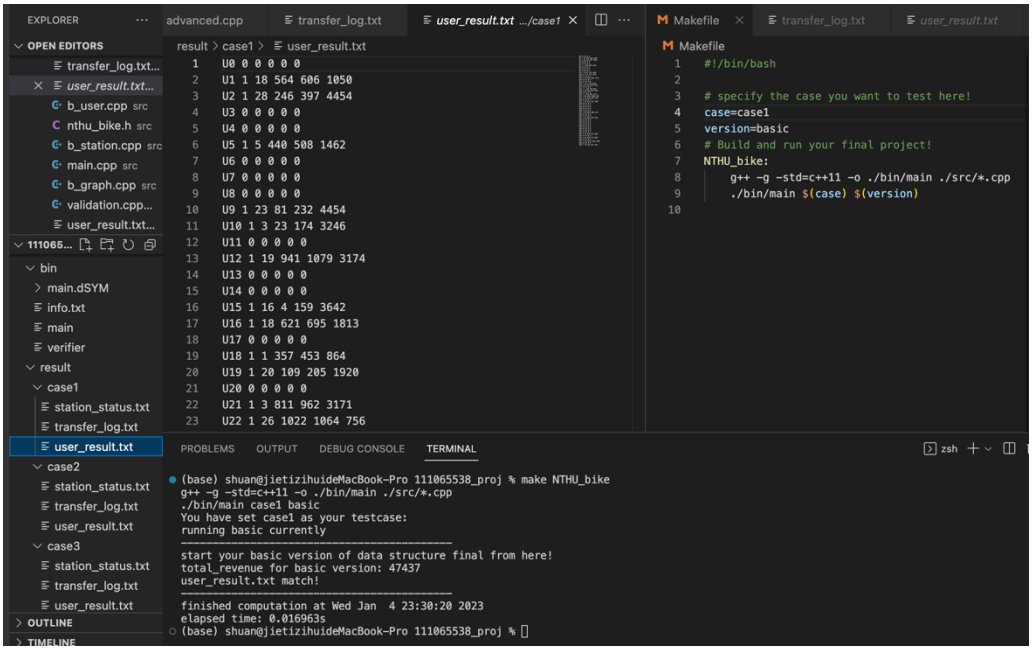
學號：111065538

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

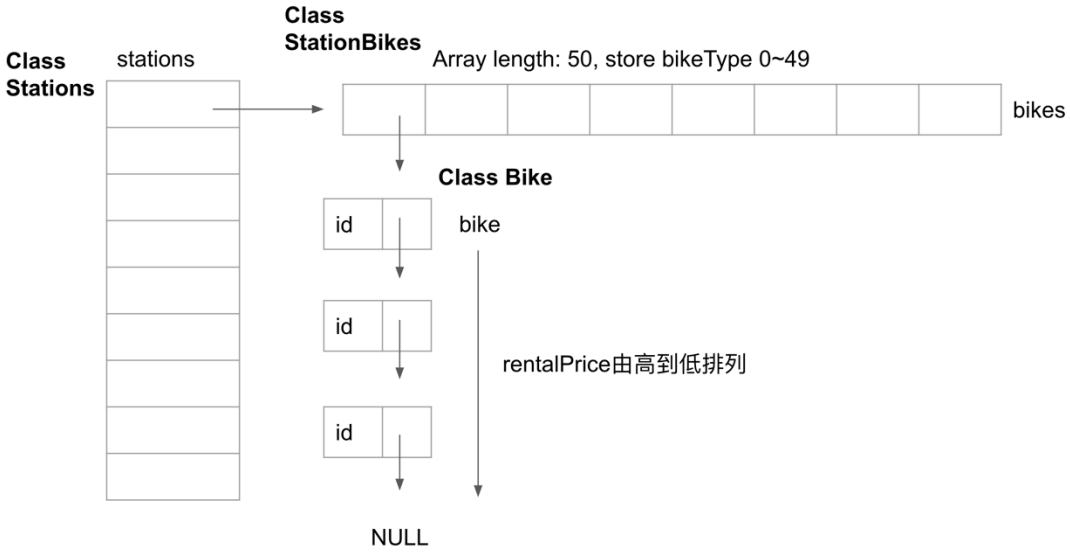


II. The details of your data structures. What data structures did you use, and how did you implement those data structures.

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++;
```

```
            }
            bucket[bucketNum][idx2] = link[j];
        }
        // store users' new order to array "link"
        int idx2=0;
        for(int j=0;j<10;j++){
            for(int k=0;k<numOfUsers;k++){
                if(bucket[j][k]){
                    link[idx2++] = bucket[j][k];
                }
            }
        }
    }
    //put link (sorted result) back to users
    for(int j=0;j<numOfUsers;j++){
        if(link[j]) users[j] = link[j];
        else break;
    }

}
```

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

```
// after sort user.txt by startTime, sort by userID if two has the same startTime
// use insertion sort since most of entries are already sorted, nearly O(n) time
void SortUsersById(User** users,const int numOfUsers){
    for(int i=1;i<numOfUsers;i++){
        int target_id = stoi((users[i]->userId).substr(1));
        User* target = users[i];
        int idx = i-1;
        // only if starttime is the same will they compare userid
        while(idx>=0 && users[idx]->startTime == target->startTime \
        && stoi((users[idx]->userId).substr(1)) > target_id){
            users[idx+1] = users[idx];
            idx--;
        }
        if(idx < 0){
            users[0] = target;
        }
        else{
            users[idx+1] = target;
        }
    }
}
```

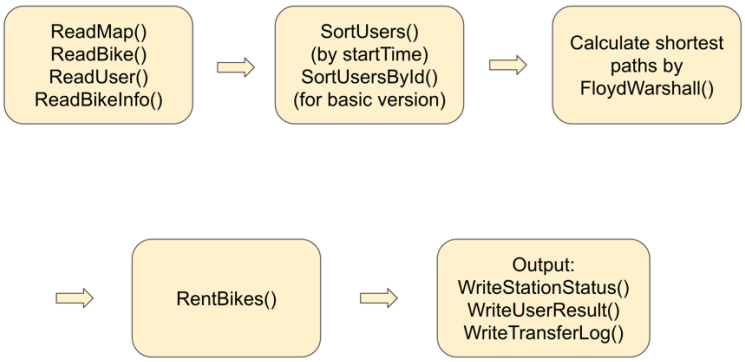
```
// iterate through stations to collect all bikes and sort by bikeid within same station
// use insertion sort since already sorted by stations, most of them only move a little bit
// preparation for station_status.txt
void SortBikeById(Stations* stations){
    int idx=0;
    // iterate through stations to collect all bikes
    for(int i=0;i<NUM_STATIONS;i++){
        // assume at most 50 types of bike
        for(int j=0;j<BIKETYPES;j++){
            Bike* sameTypeCur = stations->station[i]->bikes[j];
            if(sameTypeCur==NULL){
                continue;
            }
            while(sameTypeCur){
                stationStatus[idx++] = sameTypeCur;
                sameTypeCur = sameTypeCur->sameTypeNext;
            }
        }
    }
    // insertion sort
    // iterate through all bikes
    for(int i=1;i<NUM_BIKES;i++){
        Bike* bikeCur = stationStatus[i];
        int idx = i-1;
        // find the correct place for bikeCur
        while(idx>=0 && bikeCur && bikeCur->stationId == stationStatus[idx]->stationId \
        && bikeCur->bikeId < stationStatus[idx]->bikeId){
            // if station id the same and bikeCur id is smaller, move the compared bike one entry back
            stationStatus[idx+1] = stationStatus[idx];
            idx--;
        }
        // put bikeCur int the right place
        stationStatus[idx+1] = bikeCur;
    }
}
```

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

```
void Graph::FloydWarshall(){
    for(int k=0;k<numOfStation;k++){
        for(int i=0;i<numOfStation;i++){
            for(int j=0;j<numOfStation;j++){
                // be careful with overflow
                if(shortestPath[i][k] != INT_MAX && shortestPath[k][j] != INT_MAX\
                && shortestPath[i][j] > shortestPath[i][k]+shortestPath[k][j]){
                    shortestPath[i][j] = shortestPath[i][k]+shortestPath[k][j];
                }
            }
        }
    }
    //cout << "shortestPath[9][1]: " << shortestPath[9][1] << endl;
}
```

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

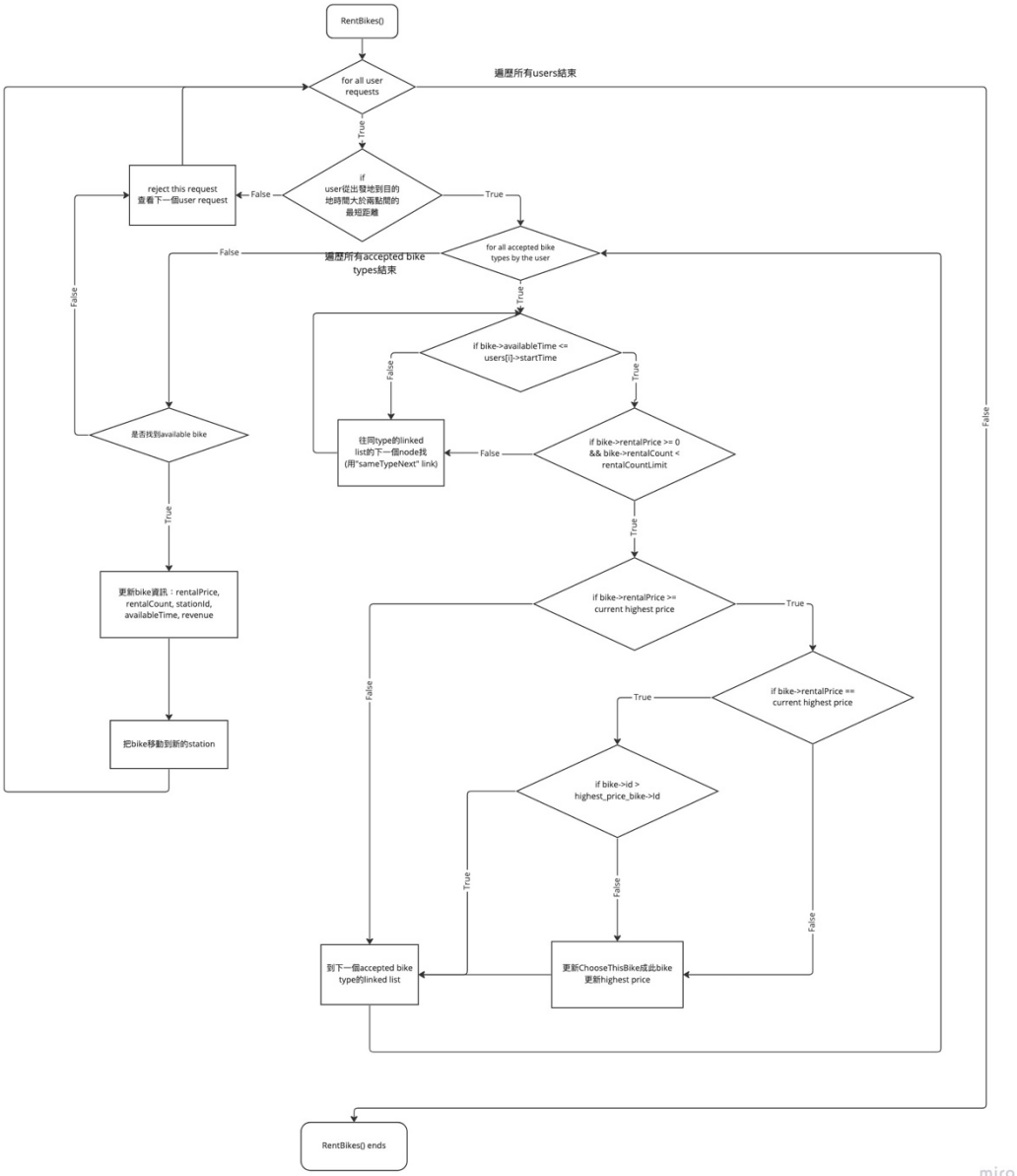
A. Overview:



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]->endTime - users[i]->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]->acceptBikeType[j]!=""){}

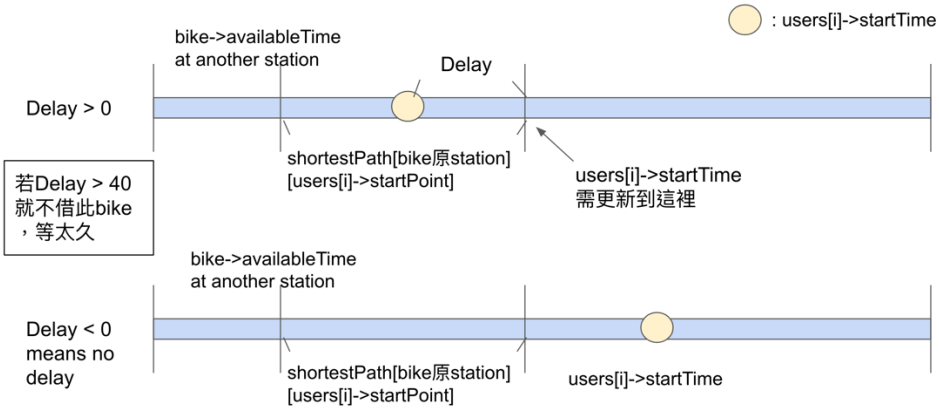
i. bike->availableTime <= users[i]->startTime
ii. rentalPrice 要大於等於 0，rentalCount < rentalCountLimit
iii. 不斷更新有最高 rentalPrice 的 bike 物件和最高 rentalPrice：chooseThisBike、highestPrice
    a. 如果有兩 bike 的 rentalPrice 都是最高價格，則選 userId 較小者
iv. 每次在 linked list 間移動都要記錄目前 bike(sameTypeBikeCur)的前一台 bike(prevBike)，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. availableTime = users[i]->startTime + 出發站點到目的站點的最短距離 (用 users[i]->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)



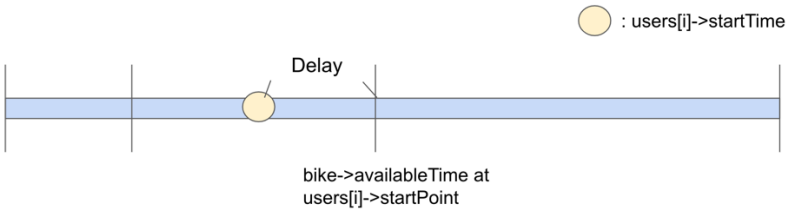
2. Advance version (與 basic version 不同處用紅色標示)：
   a. user 從出發地到目的地的時間需大於兩點之間的最短距離：users[i]->endTime - users[i]->startTime > shortestPath[startPoint][endPoint]
   b. 為了找其他 station 可能有 rentalPrice 更高的 bike 可以借，所以遍歷所有 stations，每個 station 再遍歷所有 user 接受的 biketypes
      i. 計算 delay = bike->availableTime + free transfer 花的時間 - users[i]->startTime
      ii. 檢查 delay <= 40 && user 從開始到結束時長 >= 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

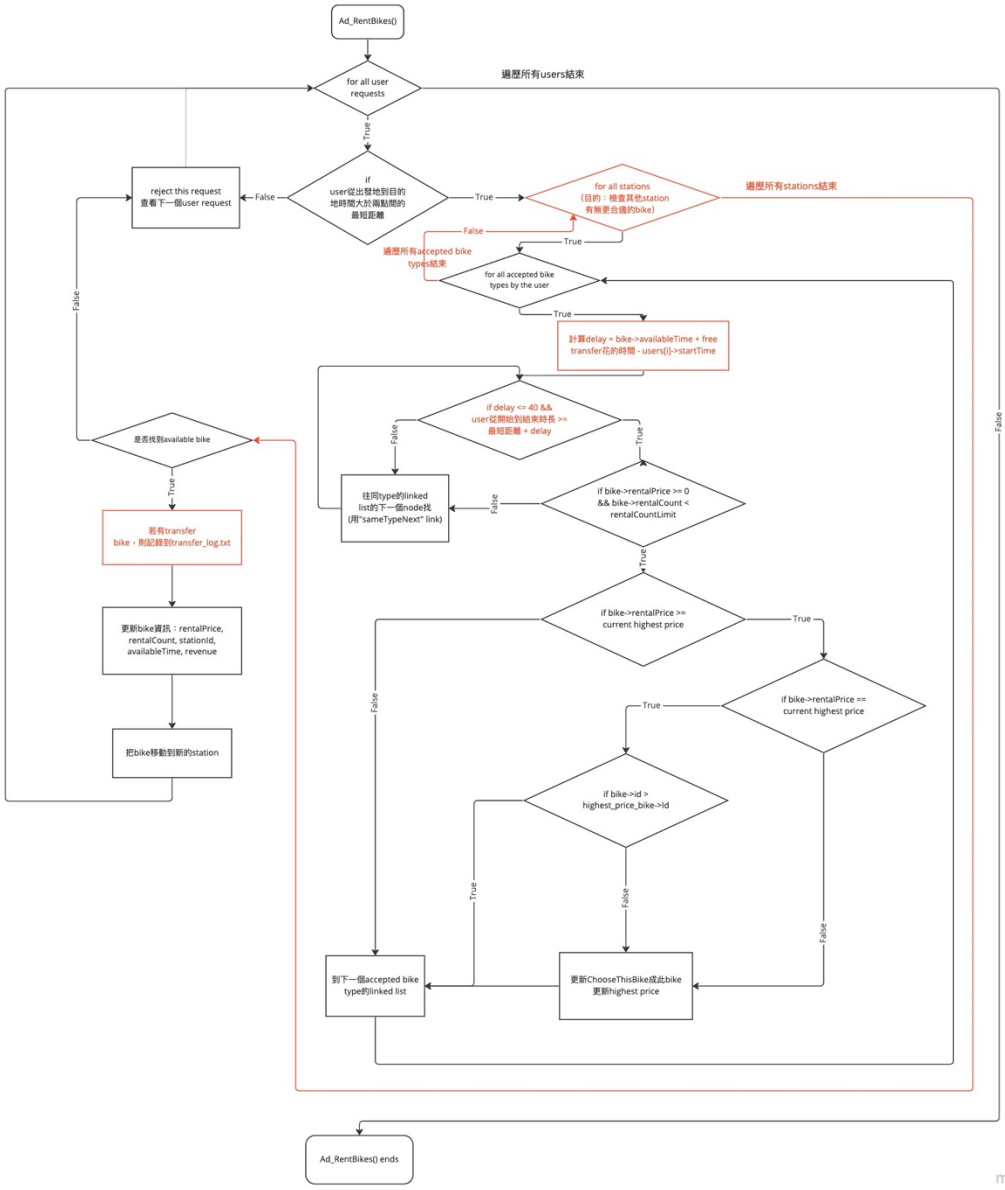
c. 如果遍歷完 user 所有可接受的 biketypes 仍找不到 chooseThisBike，表示沒有 bike 是 available，所以 request rejected
d. 如果有找到 chooseThisBike，則要移動 bike 並更新 bike 資

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. availableTime = users[i]->startTime + 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