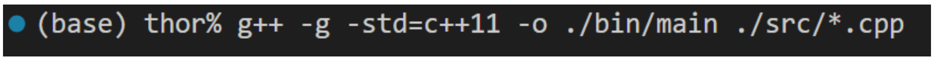
Name: 林育萱
ID : 111065546

# Final Project Report

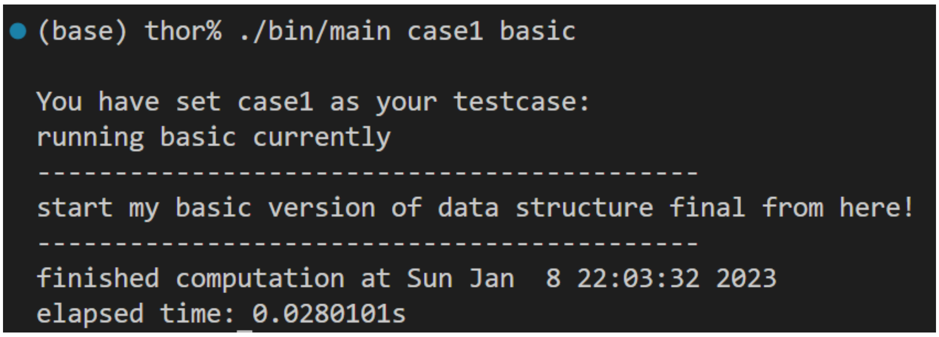
## 1. How to compile my program

Compile:



```
(base) thor% g++ -g -std=c++11 -o ./bin/main ./src/*.cpp
```

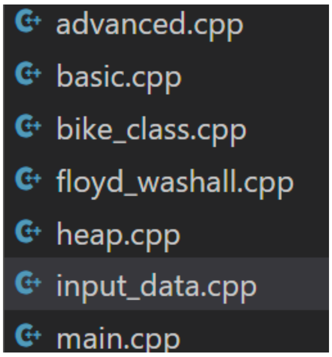
Execute:



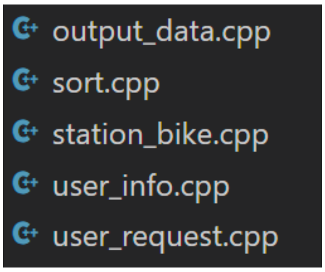
```
(base) thor% ./bin/main case1 basic

 You have set case1 as your testcase:
 running basic currently
 -------------------------------------------
 start my basic version of data structure final from here!
 -------------------------------------------
 finished computation at Sun Jan  8 22:03:32 2023
 elapsed time: 0.0280101s
```

## 2. The overall structure of my program

- Source files:



```
advanced.cpp
basic.cpp
bike_class.cpp
floyd_washall.cpp
heap.cpp
input_data.cpp
main.cpp
```

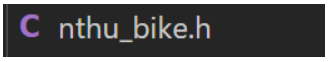


```
output_data.cpp
sort.cpp
station_bike.cpp
user_info.cpp
user_request.cpp
```

- main.cpp: the driver code of the program.
- basic.cpp: the basic version of the program.
- advance.cpp: the advance version of the program.
- input_data.cpp: write data from input files.
- output_data.cpp: write data to required files.
- user_request.cpp: for handling every user request based on arrive time.
- floyd_washall.cpp: for implementing floyd_washall algo.
- sort.cpp: implement Quick Sort for sorting user by their arrive time and
- bike_class.cpp: implement class of bike related data structure.

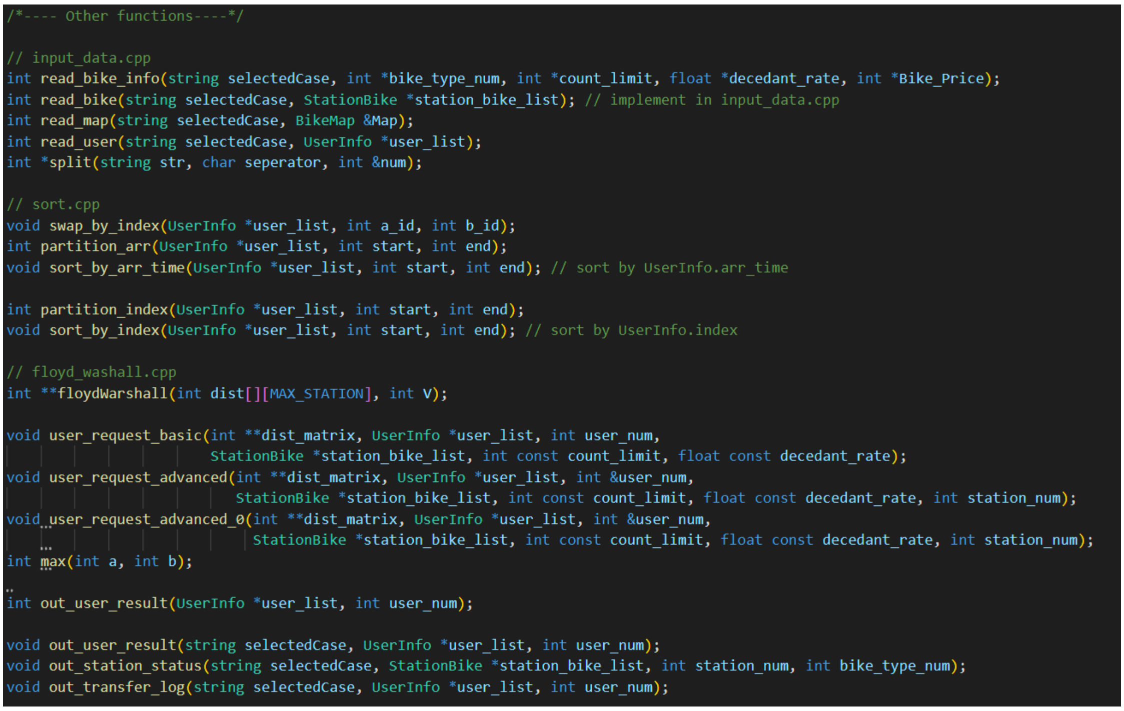
- user_info.cpp: implement class of user related data structure.
- heap.cpp: implement class of “heap of bike” data structure.
- station_bike.cpp: implement class of “array of heap of bike” data structure.

- Header files:



nthu_bike.h

- nthu_bike.h: declare all the classes and functions in the header file:



```
/*---- Other functions----*/

// input_data.cpp
int read_bike_info(string selectedCase, int *bike_type_num, int *count_limit, float *decedant_rate, int *Bike_Price);
int read_bike(string selectedCase, StationBike *station_bike_list); // implement in input_data.cpp
int read_map(string selectedCase, BikeMap &Map);
int read_user(string selectedCase, UserInfo *user_list);
int *split(string str, char seperator, int &num);

// sort.cpp
void swap_by_index(UserInfo *user_list, int a_id, int b_id);
int partition_arr(UserInfo *user_list, int start, int end);
void sort_by_arr_time(UserInfo *user_list, int start, int end); // sort by UserInfo.arr_time

int partition_index(UserInfo *user_list, int start, int end);
void sort_by_index(UserInfo *user_list, int start, int end); // sort by UserInfo.index

// floyd_washall.cpp
int **floydWarshall(int dist[][MAX_STATION], int V);

void user_request_basic(int **dist_matrix, UserInfo *user_list, int user_num,
                        StationBike *station_bike_list, int const count_limit, float const decedant_rate);
void user_request_advanced(int **dist_matrix, UserInfo *user_list, int &user_num,
                           StationBike *station_bike_list, int const count_limit, float const decedant_rate, int station_num);
void user_request_advanced_0(int **dist_matrix, UserInfo *user_list, int &user_num,
                             StationBike *station_bike_list, int const count_limit, float const decedant_rate, int station_num);
int max(int a, int b);

int out_user_result(UserInfo *user_list, int user_num);

void out_user_result(string selectedCase, UserInfo *user_list, int user_num);
void out_station_status(string selectedCase, StationBike *station_bike_list, int station_num, int bike_type_num);
void out_transfer_log(string selectedCase, UserInfo *user_list, int user_num);
```

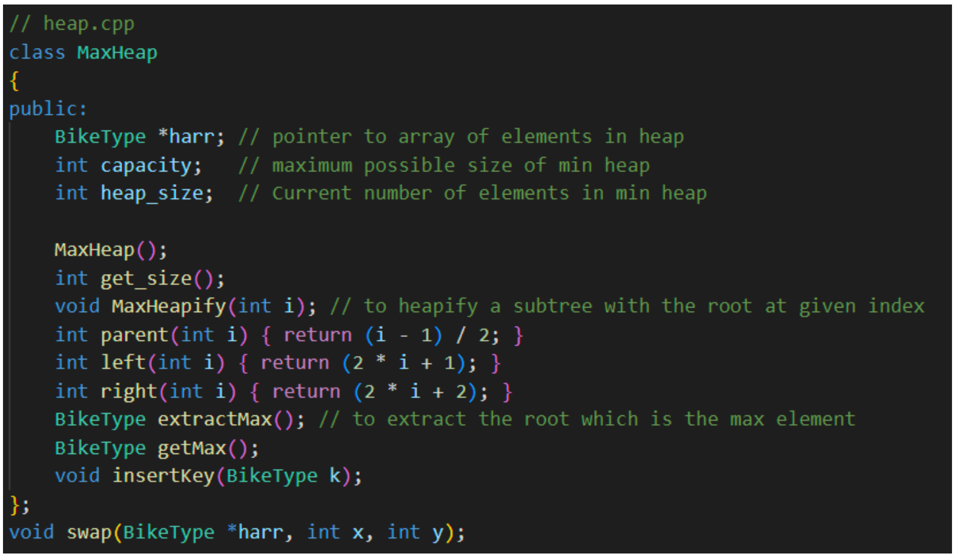
## 3. The details of my Data Structures

### A. Heap

I use max heap to store bike according to their rental price. Every bike type has their own heap, and every station contains multiple heap. Every time a user requests a bike, the heap will extract the bike with max price from the heap.

How I implement it:

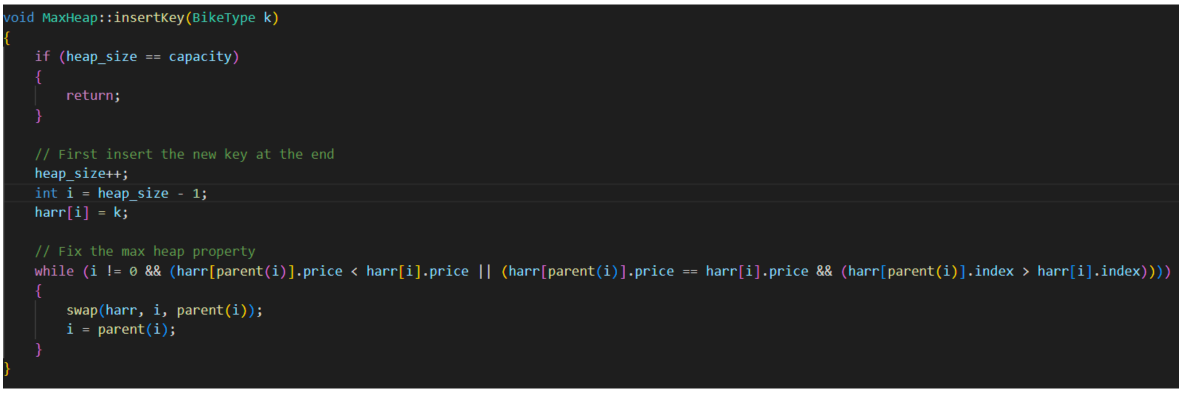
Declaration:



```
// heap.cpp
class MaxHeap
{
public:
    BikeType *harr; // pointer to array of elements in heap
    int capacity;   // maximum possible size of min heap
    int heap_size;  // Current number of elements in min heap

    MaxHeap();
    int get_size();
    void MaxHeapify(int i); // to heapify a subtree with the root at given index
    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return (2 * i + 1); }
    int right(int i) { return (2 * i + 2); }
    BikeType extractMax(); // to extract the root which is the max element
    BikeType getMax();
    void insertKey(BikeType k);
};
void swap(BikeType *harr, int x, int y);
```

Important function implementation:

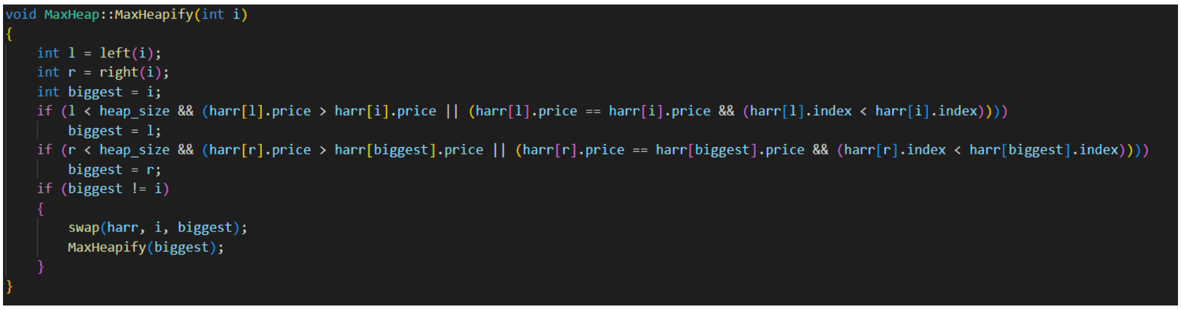
- Insert key:



```
void MaxHeap::insertKey(BikeType k)
{
    if (heap_size == capacity)
    {
        return;
    }

    // First insert the new key at the end
    heap_size++;
    int i = heap_size - 1;
    harr[i] = k;

    // Fix the max heap property
    while (i != 0 && (harr[parent(i)].price < harr[i].price || (harr[parent(i)].price == harr[i].price && (harr[parent(i)].index > harr[i].index))))
    {
        swap(harr, i, parent(i));
        i = parent(i);
    }
}
```

Insert object of class BikeType and adjust the node according to bike’s price and index.

- Heapify



```
void MaxHeap::MaxHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int biggest = i;
    if (l < heap_size && (harr[l].price > harr[i].price || (harr[l].price == harr[i].price && (harr[l].index < harr[i].index))))
        biggest = l;
    if (r < heap_size && (harr[r].price > harr[biggest].price || (harr[r].price == harr[biggest].price && (harr[r].index < harr[biggest].index))))
        biggest = r;
    if (biggest != i)
    {
        swap(harr, i, biggest);
        MaxHeapify(biggest);
    }
}
```

Maintain the max heap’s attribute, according to bike’s price and index.

## 4. The details of my Algorithm

A. Quick sort:

I use Quick sort to sort the user with their arrive time/user index respectively.

Name: 林育萱
ID : 111065546

Quick sort pseudo code:

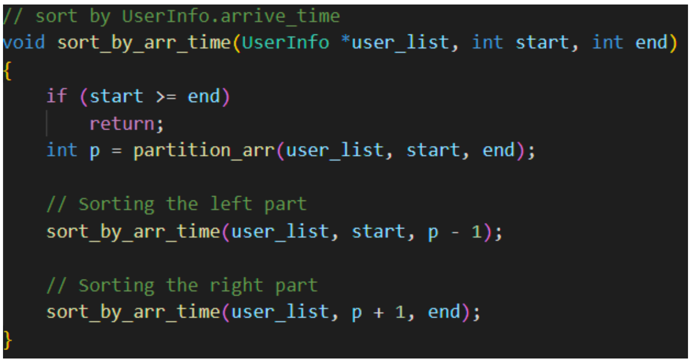
-Sort:

```
If the array is bigger than zero:
        find the partition index by Partition;
        sort the array of left to partition index;
        sort the array of right to partition index;
```

-Partition:

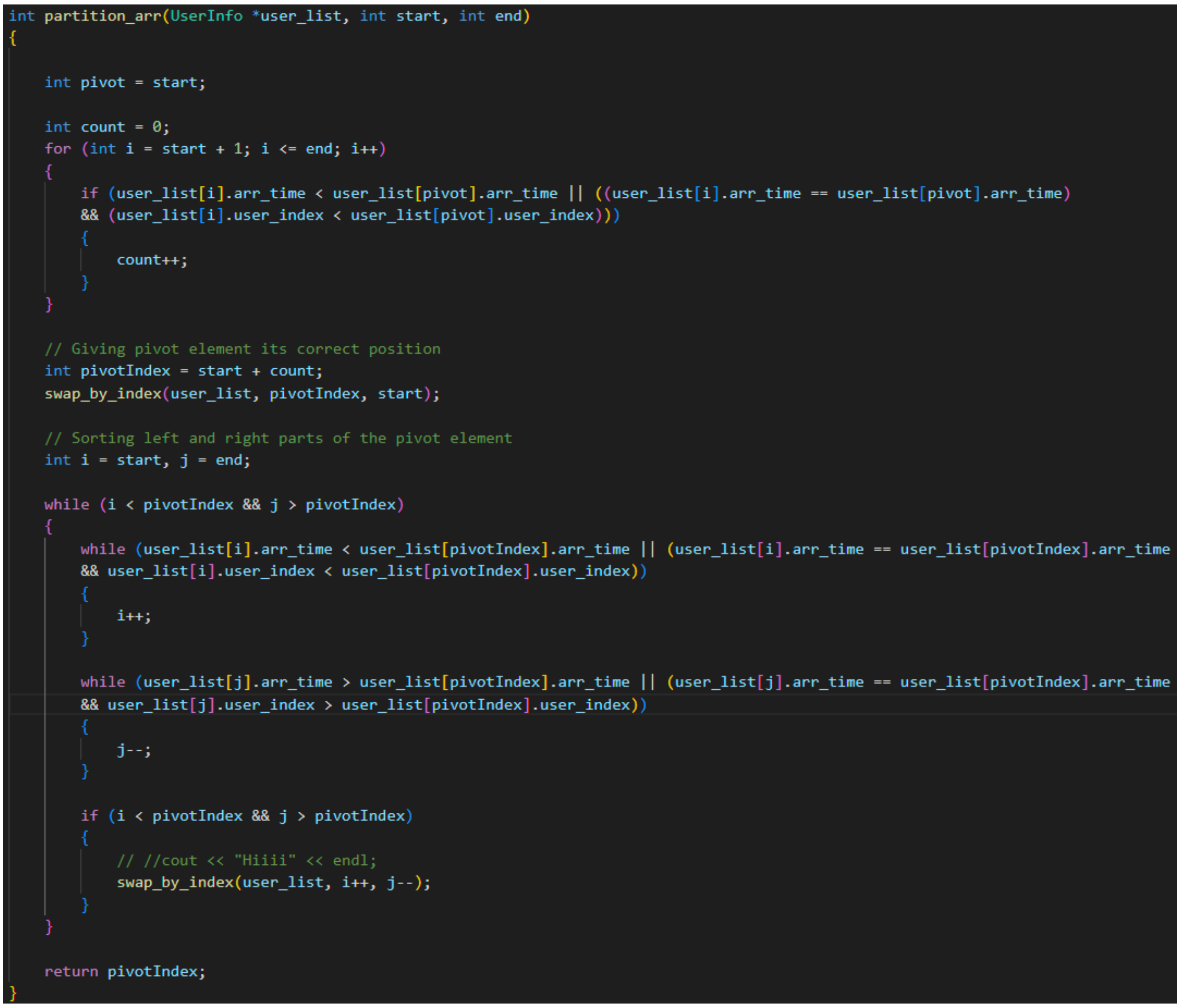
```
Find the position(index) of pivot;
pivot to the found position;
until all element right to the pivot is bigger and left to the pivot is smaller:
        swap left and right element;
```

How I implement it:



```
// sort by UserInfo.arrive_time
void sort_by_arr_time(UserInfo *user_list, int start, int end)
{
    if (start >= end)
        return;
    int p = partition_arr(user_list, start, end);

    // Sorting the left part
    sort_by_arr_time(user_list, start, p - 1);

    // Sorting the right part
    sort_by_arr_time(user_list, p + 1, end);
}
```

Name: 林育萱
ID : 111065546



```
int partition_arr(UserInfo *user_list, int start, int end)
{

    int pivot = start;

    int count = 0;
    for (int i = start + 1; i <= end; i++)
    {
        if (user_list[i].arr_time < user_list[pivot].arr_time || ((user_list[i].arr_time == user_list[pivot].arr_time)
        && (user_list[i].user_index < user_list[pivot].user_index)))
        {
            count++;
        }
    }

    // Giving pivot element its correct position
    int pivotIndex = start + count;
    swap_by_index(user_list, pivotIndex, start);

    // Sorting left and right parts of the pivot element
    int i = start, j = end;

    while (i < pivotIndex && j > pivotIndex)
    {
        while (user_list[i].arr_time < user_list[pivotIndex].arr_time || (user_list[i].arr_time == user_list[pivotIndex].arr_time
        && user_list[i].user_index < user_list[pivotIndex].user_index))
        {
            i++;
        }

        while (user_list[j].arr_time > user_list[pivotIndex].arr_time || (user_list[j].arr_time == user_list[pivotIndex].arr_time
        && user_list[j].user_index > user_list[pivotIndex].user_index))
        {
            j--;
        }

        if (i < pivotIndex && j > pivotIndex)
        {
            // //cout << "Hiiii" << endl;
            swap_by_index(user_list, i++, j--);
        }
    }

    return pivotIndex;
}
```

B. Floyd Washall Algorithm

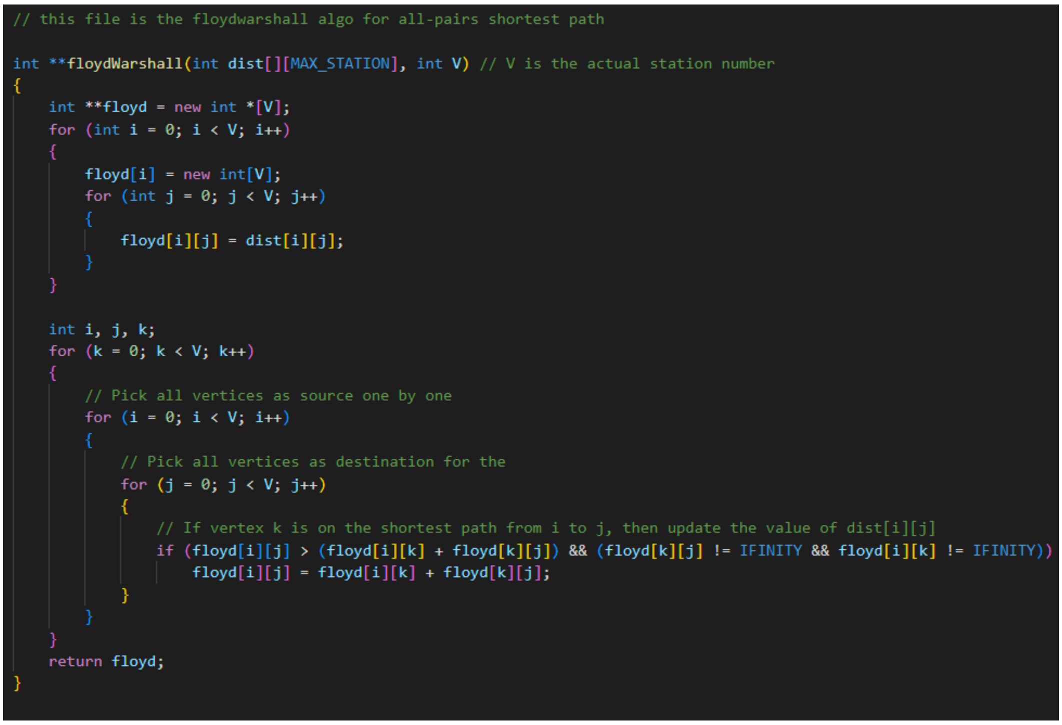
I use Floyd Washall to calculate the shortest distance between all the station pair, and then store the calculate result in a 2D array.

Floyd Washall Pseudo code:

```
$n = no\ of\ vertices$
$A = matrix\ of\ dimension\ n*n$
$for\ k = 1\ to\ n$
    $for\ i = 1\ to\ n$
        $for\ j = 1\ to\ n$
            $A^k[i, j] = min\ (A^{k-1}[i, j], A^{k-1}[i, k] + A^{k-1}[k, j])$
```

How I implement it:

Name:  林育萱
ID : 111065546



```
// this file is the floydwarshall algo for all-pairs shortest path

int **floydWarshall(int dist[][MAX_STATION], int V) // V is the actual station number
{
    int **floyd = new int *[V];
    for (int i = 0; i < V; i++)
    {
        floyd[i] = new int[V];
        for (int j = 0; j < V; j++)
        {
            floyd[i][j] = dist[i][j];
        }
    }

    int i, j, k;
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as destination for the
            for (j = 0; j < V; j++)
            {
                // If vertex k is on the shortest path from i to j, then update the value of dist[i][j]
                if (floyd[i][j] > (floyd[i][k] + floyd[k][j]) && (floyd[k][j] != IFINITY && floyd[i][k] != IFINITY))
                    floyd[i][j] = floyd[i][k] + floyd[k][j];
            }
        }
    }
    return floyd;
}
```

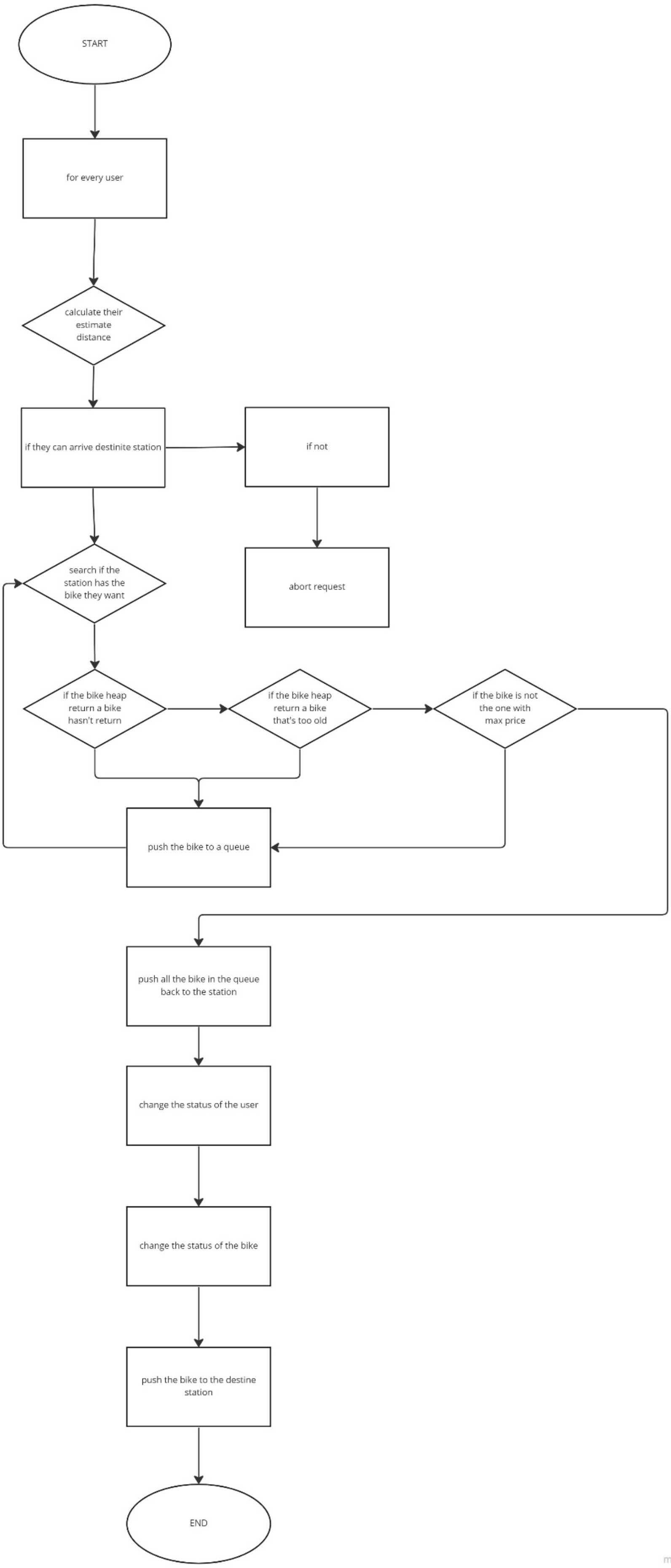
C. The user_request algorithm(self-design):

The flow chart :

(next page)

Name:  林育萱

ID : 111065546


START
for every user
calculate their estimate distance
if they can arrive destinite station
if not
abort request
search if the station has the bike they want
if the bike heap return a bike hasn't return
if the bike heap return a bike that's too old
if the bike is not the one with max price
push the bike to a queue
push all the bike in the queue back to the station
change the status of the user
change the status of the bike
push the bike to the destine station
END

# 5. Feedback

A. I would like to suggest a few progress check during the semester.

B. Assignment Spec seemed to be modified too many times, if someone finished the work early, may not see those changes.