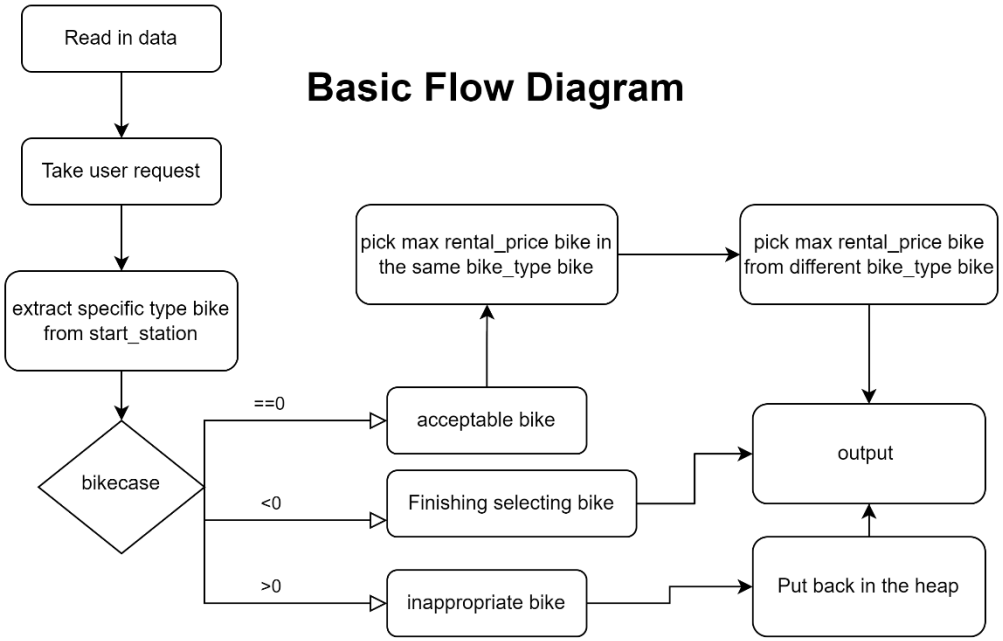
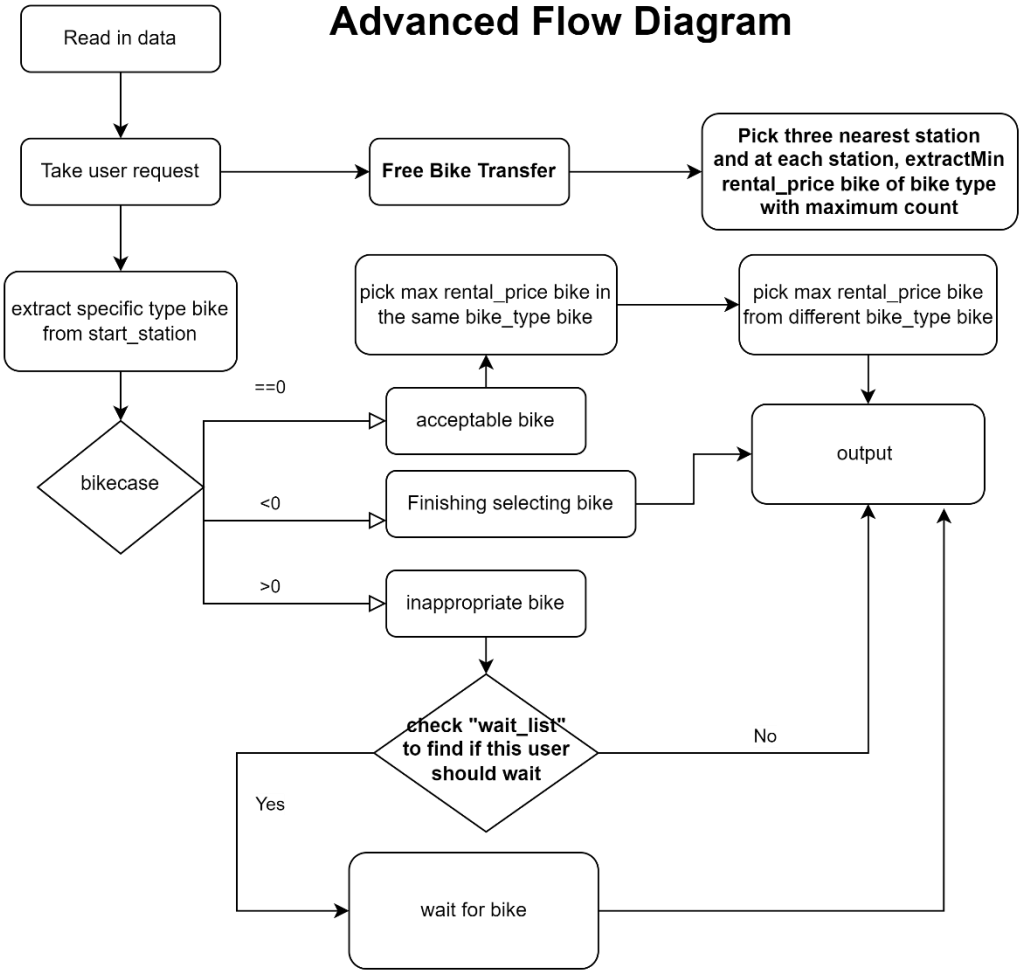
Data Structure Final Project

110030018 林芷儀

Basic Flow Diagram



Advanced Flow Diagram



Basic/Advanced

Read in data

- Define struct “UNode” to store user related data

```
class my_data
{
    // friend class my_DS; // not causing error ?
public:
    int get_station_num();
    Graph *read_map();
    bike_MaxHeap **read_bike();
    void read_bike_info();
    int read_user_num();
    void read_user();
    void sort_users(); //* merge sort user by start_time
    // void sort_bikes();                          //* merge sort bike by ID
    void mergeSort(UNode *&arr, int l, int r);     //* merge sort user by start_time
    void mergeSort(BMNode *&arr, int l, int r);    //* merge sort bike by ID
    void mergeSort(LNode *&arr, int l, int r);     //* merge sort bikeLog by ID
    void mergeSort(cUNode *&arr, int l, int r);    //* merge sort user_result by ID
    void merge(UNode *&arr, int p, int q, int r);  //* merge sort user by start_time
    void merge(BMNode *&arr, int p, int q, int r); //* merge sort bike by ID
    void merge(LNode *&arr, int p, int q, int r);  //* merge sort bikeLog by ID
    void merge(cUNode *&arr, int p, int q, int r); //* merge sort user_result by ID
```

```
// private:
//*
string select; // case?
//* for map
string start_station;
int num_start_station; // 車站的數字表示法
string end_station;
int num_end_station;   // 車站的數字表示法
int distance;          // required time between stations
int **shortest_record; //* 儲存計算過的最短路徑，避免重複計算
//* for station
int station_num = 0; // 總站數
//* for user
UNode *all_user_list;       // 把所有user都蒐集起來
int all_user_list_idx = 0; // 可以直接視為size來用
int user_num = 0;           // user總數
string user_ID;
int num_user_ID;
string AC_bike_type;    // 方便識別","相隔的bike_type
int *arr_AC_bike_type; // 用陣列儲存可用得bike type
int arr_index = 0;      // arr_AC_bike_type的index
int start_time;
int end_time;
string user_start_station;
int num_user_start_station; // user_start_station的數字表示
string user_end_station;
int num_user_end_station; // user_end_station的數字表示
//* for bike info
float depreciation;
int rental_limit;
int count_bike_type = 0;
//* for bike
int bike_total_num = 0; // bike總數
// 接收input
string bike_type;
int bike_id;
string station_id;
float rental_price;
int rental_count;
```

```
void my_data::read_user()
{

    string path = "./testcases/" + select + "/user.txt";
    ifstream ifs(path, ios::in);
    if (!ifs.is_open())
    {
        cout << "Failed to open user file.\n";
        return;
    }
    else
    {
        all_user_list = new UNode[user_num];
        while (ifs >> user_ID >> AC_bike_type >> start_time >> end_time >> user_s
        {
            num_user_ID = stoi(user_ID.erase(0, 1));
            num_user_start_station = stoi(user_start_station.erase(0, 1));
            num_user_end_station = stoi(user_end_station.erase(0, 1));
            // todo 陣列大小待優化，不然有很多浪費的空間!
            arr_AC_bike_type = new int[count_bike_type];
            for (int i = 0; i < count_bike_type; i++)
                arr_AC_bike_type[i] = -1;
            stringstream ss;
            ss.str(AC_bike_type);
            while (ss.good())
            {
                string substr;
                getline(ss, substr, ',');
                arr_AC_bike_type[arr_index] = stoi(substr.erase(0, 1));
                arr_index++;
            }
            /*...
            UNode newUNode;
            newUNode.user_ID = num_user_ID;
            newUNode.start_time = start_time;
            newUNode.end_time = end_time;
            newUNode.user_start_station = num_user_start_station;
            newUNode.user_end_station = num_user_end_station;
            // 確保進行deep copy
            int i = 0;
            // newUNode.AC_bike_type = new int[count_bike_type](); // set to 0
            newUNode.AC_bike_type = arr_AC_bike_type; // set to 0
            newUNode.len_AC = arr_index;
            // 歸零
            arr_index = 0;
            // 把UNode推到list中
            all_user_list[all_user_list_idx++] = newUNode;
```

```
typedef struct UserNode
{
    int user_ID;
    int *AC_bike_type;
    int len_AC = 0;
    int start_time;
    int end_time;
    int user_start_station;
    int user_end_station;
    //*
    int arrive_time;

} UNode;
```

Map using graph

- Using **adjacency list** to construct undirected graph

```cpp
// A structure to represent a node in adjacency list
typedef struct AdjListNode
{
    int dest;
    int weight;
    struct AdjListNode *next = NULL;
} node;

// A structure to represent an adjacency list
typedef struct AdjList
{
    // Pointer to head node of list
    struct AdjListNode *head = NULL;
} adjList;

class Graph
{
    int n;

public:
    adjList *bike_graph_List;
    int *dist_graph;
    Graph()
    {
        n = 0;
    }

    Graph(int nodeCount)
    {
        n = nodeCount;
        bike_graph_List = new adjList[n];
    }

    node *newAdjListNode(int dest, int weight);

    void addEdge(int source, int dest, int weight);
    int *&dijkstra(int src);

    // friend void dijkstra(Graph &Dgraph, int src);
    // friend class my_MinHeap;
};
```

```cpp
void Graph::addEdge(int source, int dest, int weight)
{
    node *newNode = newAdjListNode(dest, weight);

    newNode->next = bike_graph_List[source].head;
    bike_graph_List[source].head = newNode;

    // Since graph is undirected,
    // add an edge from dest to src also
    newNode = newAdjListNode(source, weight);
    newNode->next = bike_graph_List[dest].head;
    bike_graph_List[dest].head = newNode;
}
```

```cpp
// *graph
raph *my_data::read_map()
{
    // read station data
    // cout << "select: " << select << endl;
    string path = "./testcases/" + select + "/map.txt";
    ifstream ifs(path, ios::in);
    if (!ifs.is_open())
    {
        cout << "Failed to open map file.\n";
        return 0;
    }
    else
    {
        Graph *graph_ptr = new Graph(station_num); // create an object pointer
        // graph_ptr->initial_graph(station_num);
        while (ifs >> start_station >> end_station >> distance)
        { // overloading >> operator?
            // add edges into graph
            num_start_station = stoi(start_station.erase(0, 1)); //! remove first char;
            num_end_station = stoi(end_station.erase(0, 1));     //! remove first char;

            graph_ptr->addEdge(num_start_station, num_end_station, distance); // station

            cout << "start: " << num_start_station << " end: " << num_end_station << " di
            "\n";
        }
        // graph_ptr->print_graph();
        // cout << "min distance: " << graph_ptr->dijkstra(0, 5) << endl;
        ifs.close();
        return graph_ptr; // return the object pointer
    }
}
```

Dijkstra using graph_Minheap

- Using Dijkstra algorithm to get single source's shortest path to other destinations.
- Implemented with binary heap.
- Define struct "MNode" to store station and distance.

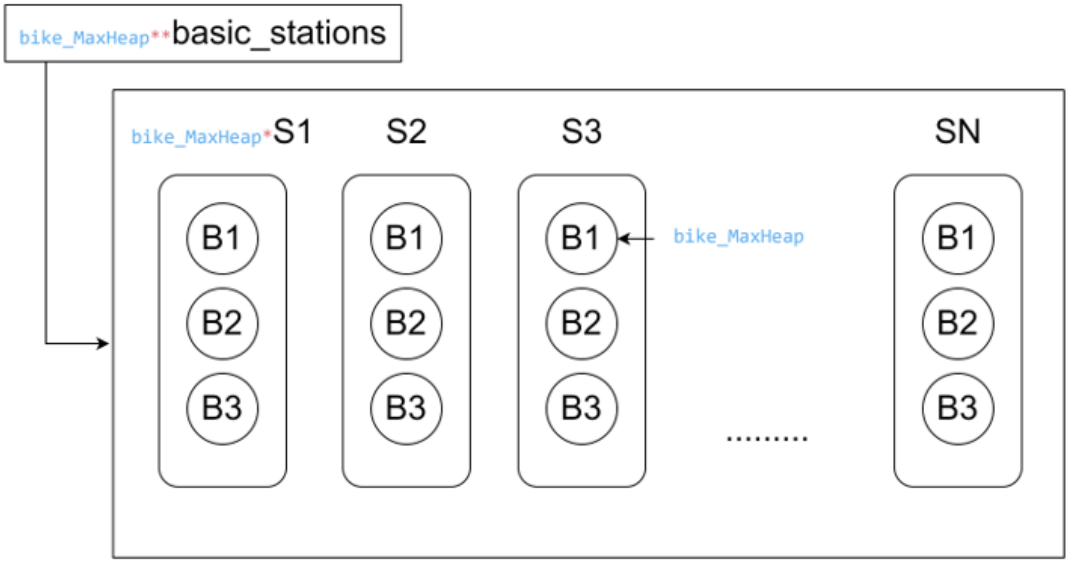
```
//! ------------------------graph_MinHeap----------
// Structure to represent a min heap node
typedef struct MinHeapNode
{
    int v;    // 站名
    int dist; // 距離
} MNode;

//! Structure to represent a min heap, for "graph"
class graph_MinHeap
{
public:
    // Number of heap nodes present currently
    int size = 0;

    // Capacity of min heap
    int capacity;

    // This is needed for decreaseKey()
    int *pos;
    MNode **array;

    graph_MinHeap(int cap)
    {
        capacity = cap;
        pos = new int[capacity];
        array = new MNode *[capacity];
    };
    // MHeap *createMinHeap(int capacity);
    MNode *newMinHeapNode(int v,
                          int dist);

    void swapMinHeapNode(MNode **a,
                         MNode **b);
    // to heapify a subtree with the root at given i
    void minHeapify(int idx);
    void decreaseKey(int v, int dist);

    // to extract(remove + return) the root which is
    MNode *extractMin();
    bool isInMinHeap(int v);
    int isEmpty();
    // void printArr(int dist[], int n);
    // void printHeapSort(ofstream &ofs);
    friend int *&Graph::dijkstra(int src);
};
```

-

```
int *&Graph::dijkstra(int src)
{
    // Get the number of vertices in graph
    int V = n;

    // dist values used to pick minimum weight edge in cut
    int *dist = new int[V];

    // minHeap represents set E
    graph_MinHeap minHeap(station_max_num);

    // Initially size of min heap is equal to V
    minHeap.size = V;
    // Initialize minheap with all vertices. dist value of all vertices
    for (int v = 0; v < V; ++v)
    {
        dist[v] = INT_MAX;
        minHeap.array[v] = minHeap.newMinHeapNode(v, dist[v]);
        minHeap.pos[v] = v;
    }

    // Make dist value of src vertex as 0 so that it is extracted first
    dist[src] = 0;
    minHeap.decreaseKey(src, dist[src]);
    while (!minHeap.isEmpty())
    {
        MNode *minHeapNode = minHeap.extractMin();
        /*...

        // Store the extracted vertex number
        int u = minHeapNode->v;

        // Traverse through all adjacent vertices of u (the extracted vertex) and
        node *pCrawl = bike_graph_List[u].head;

        // struct AdjListNode *pCrawl = graph->array[u].head;
        while (pCrawl != NULL)
        {
            int v = pCrawl->dest;
            // cout << "v:  " << v << endl;
            // If shortest distance to v is not finalized yet, and distance to v
            // through u is less than its previously calculated distance
            if (minHeap.isInMinHeap(v) &&
                dist[u] != INT_MAX &&
                pCrawl->weight + dist[u] < dist[v])
            {
                dist[v] = dist[u] + pCrawl->weight;

                // update distance value in min heap also
                minHeap.decreaseKey(v, dist[v]);
                /*
                cout << "v:  " << v << "  dist[v]: " << dist[v] << endl;*/
            }
            pCrawl = pCrawl->next;
        }
    }
    // todo: warning: reference to local variable dist' returned
    dist_graph = dist;

    // cout << "src: " << src << endl;...
    return dist_graph;
```

Stations and Bikes using bike_MaxHeap

- In order to extract maximum-rental-priced bike efficiently, using heap as main data structure.
- Define struct “BMNode” to store bike’s related data.




```
//! -------------------------bike_MinHeap-----------------
//! A class for Min Heap，for "bike_type"
typedef struct BMaxHeapNode
{
    string bike_type;
    int id = -1;
    float rental_price = -1;
    int rental_count;
    //*
    int returned_time = 0;
} BMNode;

class bike_MaxHeap
{
public:
    int capacity = bike_max_num;             // maximum possibl
    int heap_size = 0;                       // Current number
    BMNode *harr = new BMNode[bike_max_num]; // pointer to arra
    // int *pos = new int[capacity];            // 紀錄每個node的

    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return (2 * i + 1); }  // to get index of
    int right(int i) { return (2 * i + 2); } // to get index of
    BMNode getMax() { return harr[0]; }      // Returns the min

    void BMNode_swap(BMNode *x, BMNode *y)
    {
        BMNode temp = *x;
        *x = *y;
        *y = temp;
    }

    //* Inserts a new node "把整個bike_node都丟到heap中，方便比較r
    void insertKey(BMNode &newNode);
    void MINinsertKey(BMNode &newNode);
    // to heapify a subtree with the root at given index
    void MaxHeapify(int i);
    BMNode extractMax();    // 回傳最大rental_price的BMNod
    // void MinHeapify(int i);
    // BMNode extractMin();    //  algo
    bool isEmpty();
```

Output of station_status.txt (final bikes inventory)

- Using BMNode array called “Barray” to store bikes in the ascending order of station and bike_id.

```
for (int i = 0; i < read_data.station_num; i++)
{
    ss << i;
    string station_id = "S" + ss.str();
    // cout << "here!" << endl;
    station_heap_size = 0;
    for (int j = 0; j < read_data.count_bike_type; j++)
    {
        cout << "heap.size " << basic_stations[i][j].heap_size << endl;
        station_heap_size += basic_stations[i][j].heap_size;
    }
    cout << "station_heap_size: " << station_heap_size << endl;

    BMNode *Barr = new BMNode[station_heap_size];
    //! 把單一station的bike都蒐集起來放在Barr
    for (int k = 0; k < read_data.count_bike_type; k++)
    {
        // Pointer arithmetic is done in units of the size of the pointer type.
        BMNode *ptr;
        ptr = basic_stations[i][k].harr;
        /* for (int m = 0; m < basic_stations[i][k].heap_size; m++)…
    }
    // 把單一station的bike用ID進行排序小到大
    read_data.mergeSort(Barr, 0, station_heap_size - 1);
    // cout << "here!--3" << endl;

    string bikeB;
    for (int q = 0; q < station_heap_size; q++)
    {
        bikeB = "B" + Barr[q].bike_type;
        cout
            << station_id << " " << Barr[q].id << " " << bikeB << " " << Barr[q]
            rental_price << " " << Barr[q].rental_count << endl;

        ofs_status << station_id << " " << Barr[q].id << " " << bikeB << " " <<
        Barr[q].rental_price << " " << Barr[q].rental_count << endl;
    }
    Barr_idx = 0;
    delete[] Barr;
    ss.str("");
    ss.clear();
}
```

Advanced-only

Free Bike Transfer Implementation

- At each user request,
  - Find the nearest three station
  - At each station, find bike_type with **maximum heap_size**(i.e. max number of bikes)
  - If the heap_size exceeds certain number (called FBT_magic_number, depends on the total number of bikes), then extract bike of **minimum rental_price at the station**

```
//! decide FBT magic number
int FBT_magic_number;
if ((0 <= read_data.bike_total_num) && (read_data.bike_total_num <= 30))
    FBT_magic_number = 1;
else if ((30 < read_data.bike_total_num) && (read_data.bike_total_num <= 60))
    FBT_magic_number = 2;
else if ((60 < read_data.bike_total_num) && (read_data.bike_total_num <= 90))
    FBT_magic_number = 3;
else if ((90 < read_data.bike_total_num) && (read_data.bike_total_num <= 1500))
    FBT_magic_number = 5;
else if ((1500 < read_data.bike_total_num) && (read_data.bike_total_num <= 3000))
    FBT_magic_number = 6;
else if ((3000 < read_data.bike_total_num) && (read_data.bike_total_num <= 5000))
    FBT_magic_number = 7;
else if ((5000 < read_data.bike_total_num) && (read_data.bike_total_num <= 7000))
    FBT_magic_number = 8;
else if ((7000 < read_data.bike_total_num) && (read_data.bike_total_num <= 9000))
    FBT_magic_number = 9;
else if ((9000 < read_data.bike_total_num) && (read_data.bike_total_num <= 10000))
    FBT_magic_number = 10;
```

```
if (max_type_size[i] > FBT_magic_number) // 多於FBT_magic_number車再FBT
{
    // // check min
    // for (int q = 0; q < basic_stations[nearest_stations[i]][max_bike_type[i]].heap_size; q++)
    // {
    //     cout << " id: " << basic_stations[nearest_stations[i]][max_bike_type[i]].harr[q].id << " price
    // }
    // todo 可以看看extractMax/extractMin的效果
    BMNode tmp;
    if (1500 < read_data.bike_total_num)
        tmp = findMinimumElement(basic_stations[nearest_stations[i]][max_bike_type[i]], basic_stations[ne
    else
        tmp = basic_stations[nearest_stations[i]][max_bike_type[i]].extractMax();

    // //! reach rental limit
    if (tmp.rental_count >= read_data.rental_limit)
    {
        basic_stations[nearest_stations[i]][max_bike_type[i]].insertKey(tmp);
        continue;
    }

    cout
        << "transfered id: " << tmp.id << endl;
    cout << "nearest_stations[i] : " << nearest_stations[i] << endl;
    // 求出nearest_stations[i]站點的距離
    if (!read_data.shortest_record[nearest_stations[i]])
    {
        // 回傳single source 的dist array
        read_data.shortest_record[nearest_stations[i]] = basic_graph.dijkstra(nearest_stations[i]);
    }
    int transfer_path = read_data.shortest_record[nearest_stations[i]][tuser_start_station];
    cout << "path: " << transfer_path << endl;
    //* 把bike的returned time加上轉運時間
    // 所以下面抓target的時候，FBT的bike已經會在start station，而returned time已經加上轉運時間
    // 剩下user的start time要處理
    //! 這些FBT的BIKE，要修改的只有RETURN TIME，且會被插入到START STATION
    int transfer_start_time = tmp.returned_time;
    tmp.returned_time += transfer_path;
    cout << "transfered_bike-id: " << tmp.id << "  transfered_bike returned time:" << tmp.returned_time <
    /* todo 要做出extractmin
    // if (transfer_start_time == tmp.returned_time)
    // {
```

User wait for bike user

- At each user request, we also accept bikes whose returned_time exceeds _start_time, and store them into an array called “wait_list”
  - Although it’s called “list”, my implementation actually do not store every bike matching the condition. Instead, when new node comes in, I compare it with old node inside the array, and pick the one with lowest returned_time. In this way, we can not only reduce required memory space but also minimize bike transfer time(which produces 0 revenue!)
- Scenario1: First not find (choosing all available bikes except bike in “wait_list”)
  - When we search for the first time and find that there is no proper bike to pick under certain conditions
  - Then we pick bike in “wait_list”
    - If we do have bike in the “wait_list”, then user successfully find bike, then output
- Scenario2: Truly not find
  - When there is no available bike in “wait_list”, it’s considered as failure, then output.

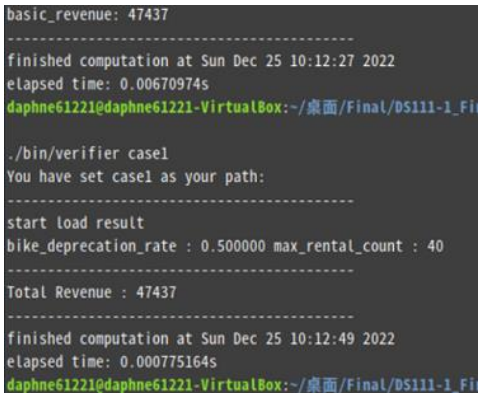
```
else
{
    if (target.returned_time > tstart_time)
    {
        //! user wait for bike when no bike is available
        // todo 要wait 哪一些bike?
        // 目前是等rental price最多的那一台
        cout << "user wait for bike" << endl;
        //* WAIT
        {
            BMNode tmp;
            // tstart_time = target.returned_time;
            if (!wait_list.isEmpty())
            {
                tmp = wait_list.extractMax();
                //! 改變標準: 轉運車越早到越好
                if (target.returned_time < tmp.returned_time)
                {
                    cout << "new waited bike target-id: " << target.id << endl;
                    // 把tmp放回去，不等tmp
                    basic_stations[tuser_start_station][stoi(tmp.bike_type)].insertKey(tmp);
                    // 等target
                    wait_list.insertKey(target);
                }
                else
                {
                    cout << "original waited bike tmp-id: " << tmp.id << endl;
                    // basic_stations[tuser_start_station][stoi(target.bike_type)].insertKey(
                    store_BMNode[tmp_idx++] = target;
                    wait_list.insertKey(tmp);
                }
            }
            else
                wait_list.insertKey(target);

            cout << "~~~~~put into wait-list~~~~~" << endl;
            target = basic_stations[tuser_start_station][tAC_bike_type[i]].extractMax();
            continue;
        }
    }
```

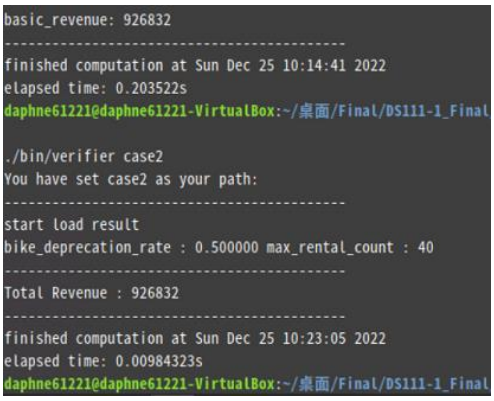
```
    //* heap已經為空，price=-10
    if (target.rental_price < -1)
    {
        bike_case = -1;
        cout << " no bike  " << endl;
        // continue;
    }
    //* 這邊抓出來的bike要放回去
    // todo 最後要優化，把這些條件放在一起檢查
    else if (target.rental_count >= read_data.rental_limit)
    {
        bike_case = 1;
        cout
            << "target.rental_count >= read_data.rental_limit" << e
    }
    else if (tstart_time + shortest_path >= tend_time)
    {
        bike_case = 1;
        cout << "(tstart_time + shortest_path > tend_time)" << endl
    }
    // else if (target.returned_time > tstart_time)
    else if (target.returned_time + shortest_path >= tend_time)
    {
        bike_case = 1;
        cout
            << "target.returned_time + shortest_path > tend_time" <<
    }
```

```
else
{
    cout << "--------------First not find -----------------" << endl;

    //! implement wait
    // todo 如果沒車可以等? 一定每次都要等嗎? 何時要等何時不用等?
    // 目前的做法: 找不到就等
    BMNode waited_bike;
    waited_bike = wait_list.extractMax();
    // int bike_start_time = 0;
    if (waited_bike.id != -10) // 不是空的
    {
        cout << "*****waited bike*****"
             << "id: " << waited_bike.id << endl;
        tstart_time = waited_bike.returned_time;
        // waited_bike.returned_time = tstart_time + shortest_path;
        target = waited_bike;

        goto FIND;
    }

    cout << "--------------Truly not find -----------------" << endl;

    cUNode user_sort;
    user_sort.user_ID = tuser_ID; // num
    user_sort.AC = 0;
    user_sort.bike_ID = 0;
    user_sort.bike_start_time = 0;
    user_sort.bike_end_time = 0;
    user_sort.revenue = 0;
    check_user_output[check_user_idx++] = user_sort;
    // todo 記得把正確的形式改回來

    ofs_user
        << user_id << " " << 0 << " " << 0 << " " << 0 << " " << 0 << " " << 0 << " " <<
        0 << endl;
}
while (!wait_list.isEmpty()) // 清空
    wait_list.extractMax();
```

Execution result

Basic testcase1/ testcase2/ testcase3

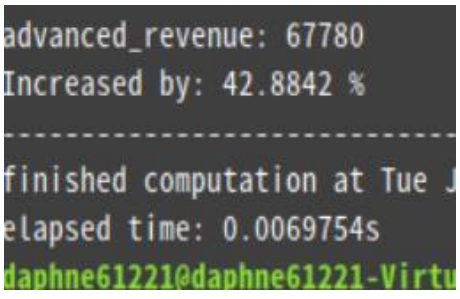


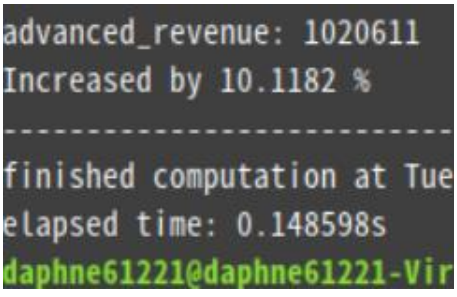


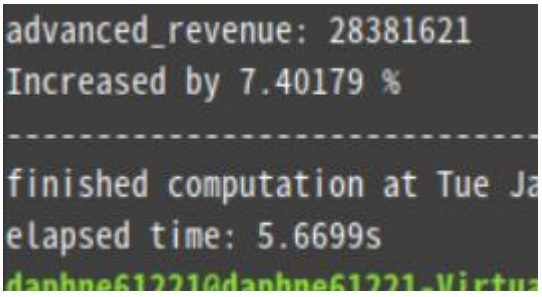
Advanced testcase1/ testcase2/ testcase3

```cpp
// test
cout << "basic_revenue: " << basic_revenue << endl;
selectedCase.erase(0, 4);
int tc = stoi(selectedCase);
switch (tc)
{
case 1:
    cout << "Increased by: " << (basic_revenue - 47437.0) / 47437.0 * 100 << " %" << endl;
    break;
case 2:
    cout << "Increased by " << (basic_revenue - 926832.0) / 926832.0 * 100 << " %" << endl;
    break;
case 3:
    cout << "Increased by " << (basic_revenue - 26425651.0) / 26425651.0 * 100.0 << " %" << endl;
    break;
default:
    break;
}
```

Increased by 42.8842 % / Increased by 10.1182% / Increased by 7.40179 %







Execution command

Using make file:
make

Using g++ and bin:
g++ -g -std=c++11 -o ./bin/main ./src/*.cpp
./bin/main <case> <version>

Environment

Virtual Machine (VM) for Windows

- VM: VirtualBox
- OS: Linux Mint
- Version: Ubuntu.
-