Final Project Report

Data Structure

Vincent Leon Ghanz
109006230

Table of Contents

Contents :

Chapter 1 : My algorithm

For the first step what I did after read all the introduction about the project, right away I made a Mind map for my algorithm to write all of my plans in the future and what should I do. Making a mind map is easier for me to recognize and make sure my procedure / steps working out. Also I only point out the main part of my algorithm like the big function that created my code works.

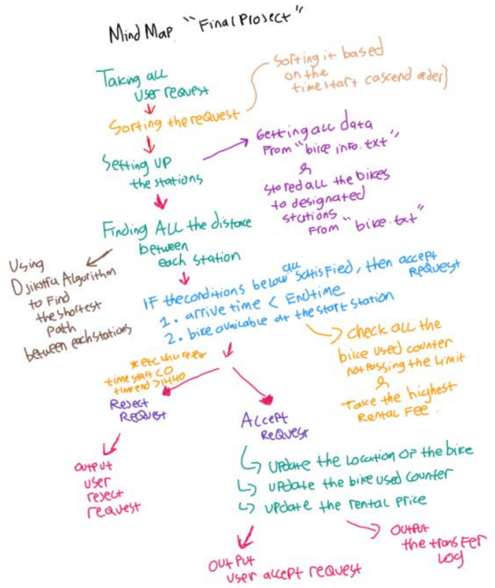
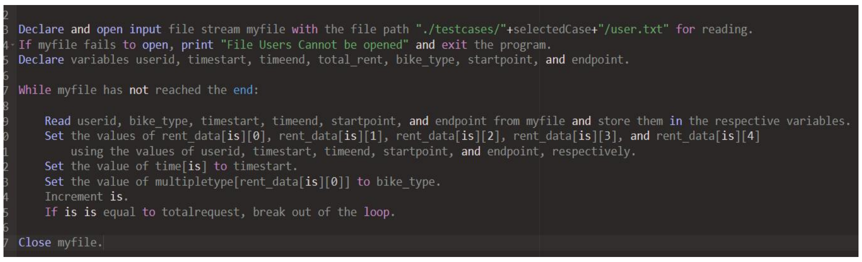

Mind Map "Final Project"
Taking all User request
Sorting it based on the time start cascend order)
Sorting the request
Setting up the stations
Getting all data from "bike info. txt" & stored all the bikes to designated stations from "bike. txt"
Finding All the distance between each station
Using Djikstra Algorithm to find the shortest path between each stations
IF the conditions below all satisfied, then accept Request
1. arrive time < Endtime
2. bike available at the start station
check all the bike used counter not passing the limit & Take the highest Rental Fee
time start < 0 time end > 1440
Reject Request
Accept Request
Update the location of the bike
Update the bike used counter
Update the rental price
Output the transfer log
Output user reject request
Output user accept request

Figure 1.1 <Mind-map *of my Algorithm*>

As you can see above in the *Figure 1.1*, My mind-map simply summarize this project for the Basic part. For the first step what I did is taking all the user request from the "*User.txt*" . And stores it in each 2d array with custom index that I have made only for my algorithm in Figure *1.2*. Why I am using 2d array to stored data? For me , I think its one of the simplest method for me to understand. I implemented my algorithm by accepting each request from the user first from the input file. And then I need to sort the user request based of its start time in ascending order which will be explained later. I assign the values of my 2d array based on the input values that I get from *User.txt* , for the column index 0 is the Id of the user, column index 1 is the time start of the user request, column index 2 is End time , column index 3 and 4 are for the departure and destination station. And also as you can see , 2 single array name *time* and multiple-type which will be very important for next process especially *Multipletype* cause it is a string array that carry a string contain user request bike types (Cause as we know the user can choose multiple bike type) so I stored it first as a string and I partitioned the string in the next process.

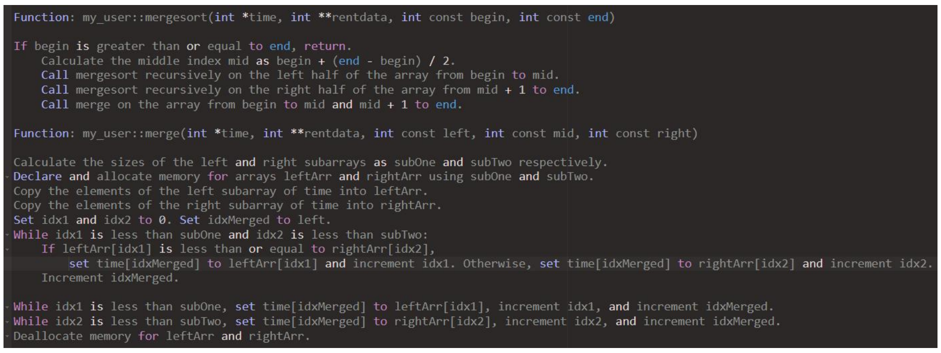


```
Declare and open input file stream myfile with the file path "./testcases/"+selectedCase+"/user.txt" for reading.
If myfile fails to open, print "File Users Cannot be opened" and exit the program.
Declare variables userid, timestart, timeend, total_rent, bike_type, startpoint, and endpoint.

While myfile has not reached the end:

    Read userid, bike_type, timestart, timeend, startpoint, and endpoint from myfile and store them in the respective variables.
    Set the values of rent_data[is][0], rent_data[is][1], rent_data[is][2], rent_data[is][3], and rent_data[is][4]
        using the values of userid, timestart, timeend, startpoint, and endpoint, respectively.
    Set the value of time[is] to timestart.
    Set the value of multipletype[rent_data[is][0]] to bike_type.
    Increment is.
    If is is equal to totalrequest, break out of the loop.

Close myfile.
```

Figure 1.2 <Pseudo-code *for Accepting User request*>

Then what I did in the second step is Sort the all the user request data that we just got from Figure *1.2* based on the time start and in the Ascending order. For the sorting algorithm, I choose algorithm that has the smallest time complexity for the big data cause we know the User <= 100.000 . So I am using Merge sort for this sorting (**The details will be in Chapter 2**) . I severe some problems when sorting the data based on time, because when I sort the time in the ascending order, then only the time data that sorted which is bad because the other data is stay still , and reminds that I stored this in Dynamic 2d array which is also the challenge. I solved it by creating another 2d dynamic array which has the sorted user request with the correct information of the request after sort. But before making the sorting algorithm , I make a **class** name ***my_user*** to make it easy for me to access and understand the flow of my code. So inside of that class have many **void function** and **int function** to help my algorithm works and some variables too.One of the important function inside **my_user** class is as you can see in the Figure *1.3* below, ***my_user::mergesort*** function is a recursive implementation of the merge sort algorithm, which is a divide and conquer sorting algorithm that works by recursively dividing an array into smaller pieces, sorting those pieces, and then merging them back together in a sorted order.

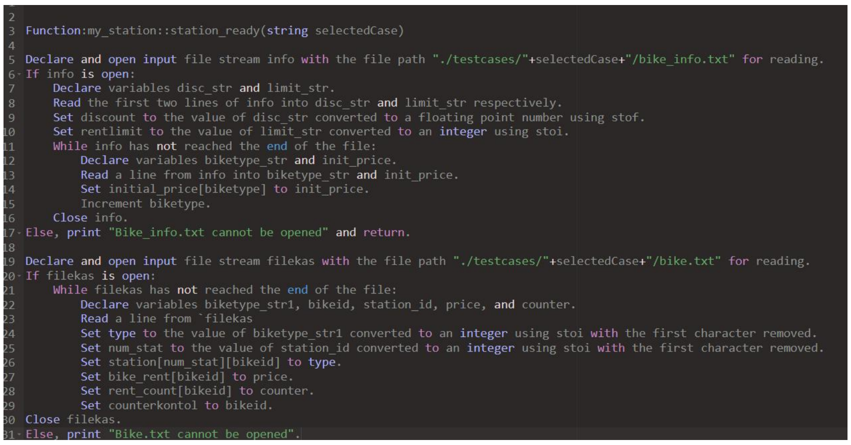
That function has 4 parameters which are **Time** is a pointer to an array of integers that represent the times of user requested the bike, **rentdata** is a pointer to a 2D array of integers that holds the data for each bike rental request. begin is an integer that represents the starting index of the portion of the time array that the function will operate on. **end** is an integer that represents the ending index of the portion of the time array that the function will operate on. The **my_user::merge function** is a helper function that is called by **my_user::mergesort** to merge two sorted arrays back together in a single sorted array and the parameters are basically the same like **my_user::mergesort.**



```
Function: my_user::mergesort(int *time, int **rentdata, int const begin, int const end)

If begin is greater than or equal to end, return.
    Calculate the middle index mid as begin + (end - begin) / 2.
    Call mergesort recursively on the left half of the array from begin to mid.
    Call mergesort recursively on the right half of the array from mid + 1 to end.
    Call merge on the array from begin to mid and mid + 1 to end.

Function: my_user::merge(int *time, int **rentdata, int const left, int const mid, int const right)

Calculate the sizes of the left and right subarrays as subOne and subTwo respectively.
Declare and allocate memory for arrays leftArr and rightArr using subOne and subTwo.
Copy the elements of the left subarray of time into leftArr.
Copy the elements of the right subarray of time into rightArr.
Set idx1 and idx2 to 0. Set idxMerged to left.
While idx1 is less than subOne and idx2 is less than subTwo:
    If leftArr[idx1] is less than or equal to rightArr[idx2],
        set time[idxMerged] to leftArr[idx1] and increment idx1. Otherwise, set time[idxMerged] to rightArr[idx2] and increment idx2.
    Increment idxMerged.

While idx1 is less than subOne, set time[idxMerged] to leftArr[idx1], increment idx1, and increment idxMerged.
While idx2 is less than subTwo, set time[idxMerged] to rightArr[idx2], increment idx2, and increment idxMerged.
Deallocate memory for leftArr and rightArr.
```

Figure 1.3 <*Pseudocode for Sort algorithm*>

Then following the next level of my mind-map is setting up all the station based on the input file that given. I also made another **class** named **my_station** which the purpose of this class to organize all my functions and variables that related to this data station and bike and so I can access and understand easily. And one of the functions that inside **my_station** class is **void my_station::station_ready**. This function is really important for my code cause it's one of the root pillars of my whole algorithm, basicllay the function of it is to get all the value / data from the input file that contains all the initial locations of the bike and other bike info. You can see the pseudo-code of this function below in Figure *1.4.*

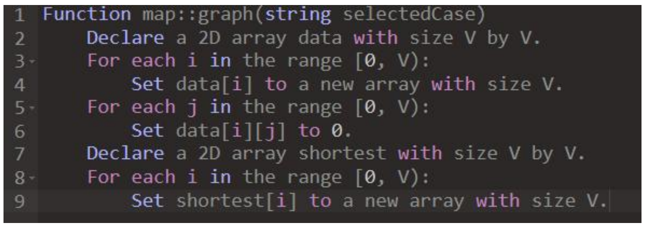


```

Function:my_station::station_ready(string selectedCase)

Declare and open input file stream info with the file path "./testcases/"+selectedCase+"/bike_info.txt" for reading.
If info is open:
    Declare variables disc_str and limit_str.
    Read the first two lines of info into disc_str and limit_str respectively.
    Set discount to the value of disc_str converted to a floating point number using stof.
    Set rentlimit to the value of limit_str converted to an integer using stoi.
    While info has not reached the end of the file:
        Declare variables biketype_str and init_price.
        Read a line from info into biketype_str and init_price.
        Set initial_price[biketype] to init_price.
        Increment biketype.
    Close info.
Else, print "Bike_info.txt cannot be opened" and return.

Declare and open input file stream filekas with the file path "./testcases/"+selectedCase+"/bike.txt" for reading.
If filekas is open:
    While filekas has not reached the end of the file:
        Declare variables biketype_str1, bikeid, station_id, price, and counter.
        Read a line from `filekas
        Set type to the value of biketype_str1 converted to an integer using stoi with the first character removed.
        Set num_stat to the value of station_id converted to an integer using stoi with the first character removed.
        Set station[num_stat][bikeid] to type.
        Set bike_rent[bikeid] to price.
        Set rent_count[bikeid] to counter.
        Set counterkontol to bikeid.
Close filekas.
Else, print "Bike.txt cannot be opened".
```

Figure 1.4 <*Pseudocode for Setting up the stations algorithm*>

So I read the data from the *bikeinfo.txt* which contains the depreciation rate , rent limit of the bike and the initial price of each bike . if it is successfully opened, the first two lines are read into disc_str and limit_str respectively using getline. discount is then set to the value of disc_str converted to a floating point number using stof, and rentlimit is set to the value of limit_str converted to an integer using stoi.
The code then enters a loop that reads lines from info until the end of the file is reached. Each iteration of the loop reads a line from info into the variables biketype_str and init_price, and sets initial_price[types of bike] to init_price. biketype is then incremented .

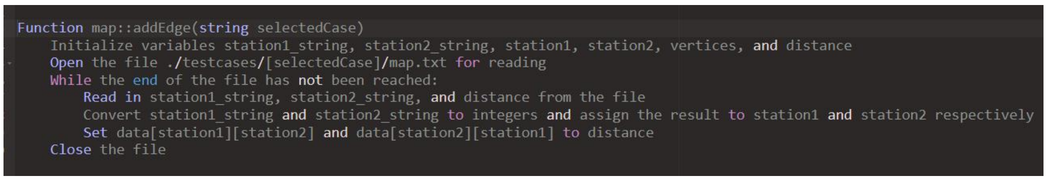
The next important part of my algorithm is to find the shortest path between all of the stations. Its important because if we know the shortest distance from the depart station to the destinations , then I can easily check if one of the request requirement is satisfy cause we know the distance is always one unit per minute for the simplicity to check. To know the shortest distance each stations, I use Djikstra Algorithm which is popular to find the shortest path. Before start, I made a **class** again named **map**. This class contains all the functions and variables of the Djikstra algorithm so it will work properly and organize pretty well. To make this algorithm work properly , I should create undirected graph first, then I created a **void map::graph(string selectedCase) .** The pseudocode of this function is in the below in Figure *1.5.*



```
Function map::graph(string selectedCase)
    Declare a 2D array data with size V by V.
    For each i in the range [0, V):
        Set data[i] to a new array with size V.
    For each j in the range [0, V):
        Set data[i][j] to 0.
    Declare a 2D array shortest with size V by V.
    For each i in the range [0, V):
        Set shortest[i] to a new array with size V.
```

Figure 1.5 <*Pseudocode for Undirected graph*>

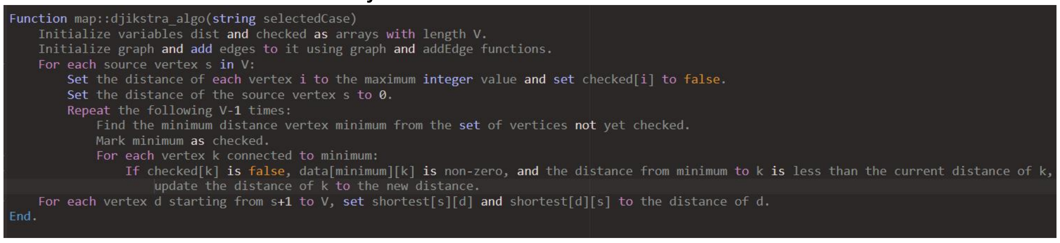
Then after initialize undirected graph then, I need to add edges to that undirected graph using the other function inside **map** class which is **void map::addEdge(string selectedCase) .** The function add the edge based on the input *map.txt* which provide all the distance between the stations. The pseudo-code shown below :



```
Function map::addEdge(string selectedCase)
    Initialize variables station1_string, station2_string, station1, station2, vertices, and distance
    Open the file ./testcases/[selectedCase]/map.txt for reading
    While the end of the file has not been reached:
        Read in station1_string, station2_string, and distance from the file
        Convert station1_string and station2_string to integers and assign the result to station1 and station2 respectively
        Set data[station1][station2] and data[station2][station1] to distance
    Close the file
```

Figure 1.6 <*Pseudocode for AddEdge*>

And then I can create the final Dijkstra main function which shown below:



```
Function map::djikstra_algo(string selectedCase)
    Initialize variables dist and checked as arrays with length V.
    Initialize graph and add edges to it using graph and addEdge functions.
    For each source vertex s in V:
        Set the distance of each vertex i to the maximum integer value and set checked[i] to false.
        Set the distance of the source vertex s to 0.
        Repeat the following V-1 times:
            Find the minimum distance vertex minimum from the set of vertices not yet checked.
            Mark minimum as checked.
            For each vertex k connected to minimum:
                If checked[k] is false, data[minimum][k] is non-zero, and the distance from minimum to k is less than the current distance of k,
                    update the distance of k to the new distance.
    For each vertex d starting from s+1 to V, set shortest[s][d] and shortest[d][s] to the distance of d.
End.
```

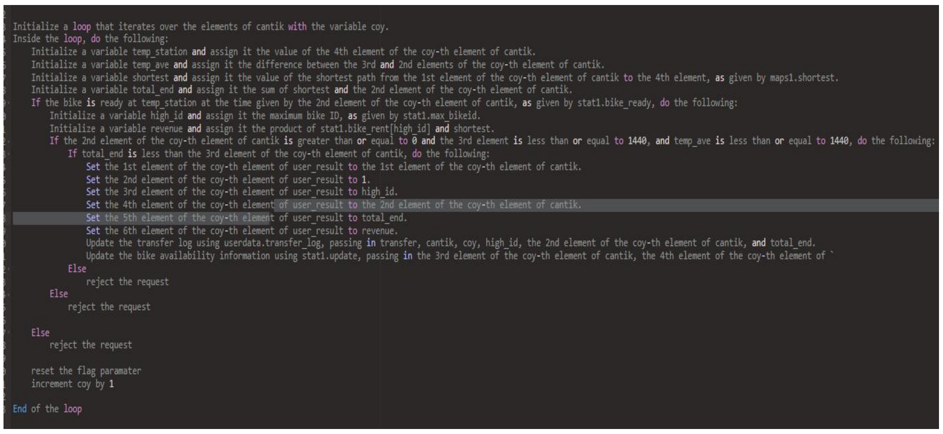
Figure 1.7 <*Pseudocode for Djikstra*>

The function first initializes two arrays, dist and checked, which will store the shortest distance from the source vertex to each vertex and whether a vertex has been checked, respectively. It then calls the graph and addEdge functions to create the graph and add the edges to it. Next, the function loops through all the vertices in the graph and sets the initial distance of each vertex to the maximum integer value, and sets the checked value of each vertex to false. The distance of the source vertex is set to 0. The function then enters a loop that runs V-1 times, where V is the number of vertices in the graph. On each iteration, it calls the minDistance function to find the vertex with the minimum distance value that has not been checked. The checked value of this vertex is then set to true.
The function then loops through all the vertices again, and updates the distance value of each vertex if the following conditions are met:
→ The vertex has not been checked
→ There is an edge between the minimum distance vertex and the current vertex
→ The distance of the minimum distance vertex is not the maximum integer value
→ The sum of the distance of the minimum distance vertex and the weight of the edge between it and the current vertex is less than the current distance of the current vertex.
After the inner loop completes, the shortest distance from the source to each vertex is stored in the shortest array. The function then repeats the process for each vertex as the source.

Then after getting all the data from those algorithms above, I can judge that the user request is acceptable or not according to our regulations. So I make a while loop for iterating all the sorted user request from the smallest time start until the end of the request. And inside this loop basically , I just assign 3 conditions for the user request can be accpeted :

1. The Requested Bike is ready at departure station
2. The time start is bigger than equals to 0 and The end start is smaller or equals than 1440 (which is the maximum time)
3. The arrive time cannot be later than the End time.

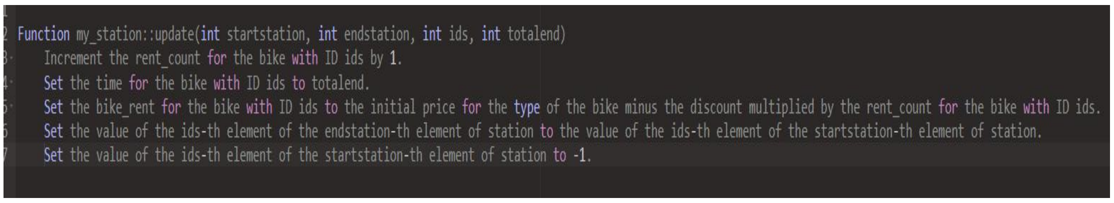
If all the conditions above are satisfy , then I accept the request. But If one of them not satisfy , then the system will reject the request . You can see in the figure *1.8.*



```
Initialize a loop that iterates over the elements of cantik with the variable coy.
Inside the loop, do the following:
    Initialize a variable temp_station and assign it the value of the 4th element of the coy-th element of cantik.
    Initialize a variable temp_ave and assign it the difference between the 3rd and 2nd elements of the coy-th element of cantik.
    Initialize a variable shortest and assign it the value of the shortest path from the 1st element of the coy-th element of cantik to the 4th element, as given by maps1.shortest.
    Initialize a variable total_end and assign it the sum of shortest and the 2nd element of the coy-th element of cantik.
    If the bike is ready at temp_station at the time given by the 2nd element of the coy-th element of cantik, as given by stat1.bike_ready, do the following:
        Initialize a variable high_id and assign it the maximum bike ID, as given by stat1.max_bikeid.
        Initialize a variable revenue and assign it the product of stat1.bike_rent[high_id] and shortest.
        If the 2nd element of the coy-th element of cantik is greater than or equal to 0 and the 3rd element is less than or equal to 1440, and temp_ave is less than or equal to 1440, do the following:
            If total_end is less than the 3rd element of the coy-th element of cantik, do the following:
                Set the 1st element of the coy-th element of user_result to the 1st element of the coy-th element of cantik.
                Set the 2nd element of the coy-th element of user_result to 1.
                Set the 3rd element of the coy-th element of user_result to high_id.
                Set the 4th element of the coy-th element of user_result to the 2nd element of the coy-th element of cantik.
                Set the 5th element of the coy-th element of user_result to total_end.
                Set the 6th element of the coy-th element of user_result to revenue.
                Update the transfer log using userdata.transfer_log, passing in transfer, cantik, coy, high_id, the 2nd element of the coy-th element of cantik, and total_end.
                Update the bike availability information using stat1.update, passing in the 3rd element of the coy-th element of cantik, the 4th element of the coy-th element of `
            Else
                reject the request
        Else
            reject the request

    Else
        reject the request

    reset the flag paramater
    increment coy by 1

End of the loop
```

Figure 1.8 <*Pseudocode*>

There are several important function that I havent mentioned yet like **my_station::update(int startstation, int endstation, int ids, int totalend),** This function is important because when I accept the user request , then the system need update the newest location of the bike, rental counter, the rental price etc. The recent location of the bike must be the End station of the user request and The bike cannot be used until the bike is at the station at certain time and no user use it.



```
Function my_station::update(int startstation, int endstation, int ids, int totalend)
    Increment the rent_count for the bike with ID ids by 1.
    Set the time for the bike with ID ids to totalend.
    Set the bike_rent for the bike with ID ids to the initial price for the type of the bike minus the discount multiplied by the rent_count for the bike with ID ids.
    Set the value of the ids-th element of the endstation-th element of station to the value of the ids-th element of the startstation-th element of station.
    Set the value of the ids-th element of the startstation-th element of station to -1.
```

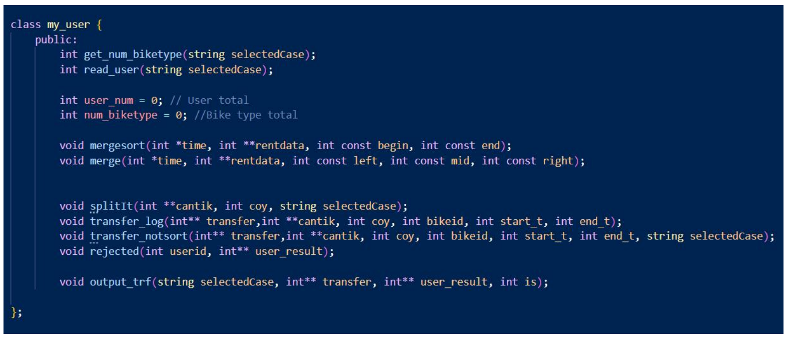
Figure 1.8 <*Pseudocode of Update function*>

And basically that’s all how the system works, and for the output part, i created 2d array special only for the output and the data sorted based on the user id in the ascending order, the Transfer_log also the same. For the station status. The system will tell the updated location of each bike after doing all the user request . I think that’s the main point for my algorithm , I cannot write all of the custom functions that I have used in my algorithm cause the limitation of space, but I think from what I write above, basically explain the whole system works.

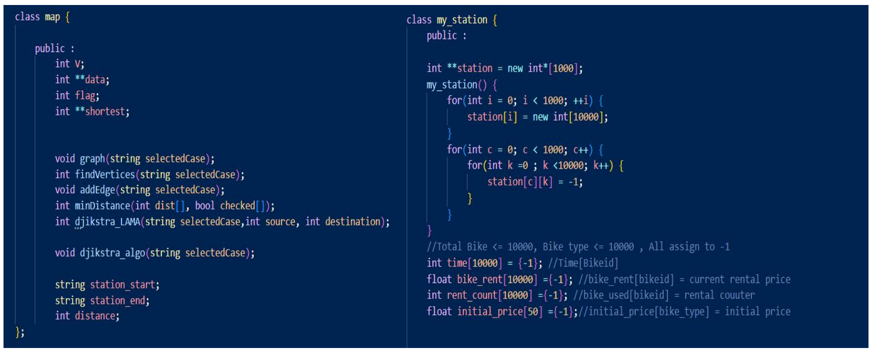
Chapter 2 : Details

This chapter I will explain more the detail of what kind of Data structures that i use from what we learn in the current course. The project require us to implement using the Data structure that we have learned without using any STL Data structures like Vector etc. So From the first impression, its pretty hard for me.

Starting from the first part, I am using **Class** to represent collection of data and function to operates with my system and It is easier using **Class** cause you can access the data easily just by calling out the data or the function that you have created before. The purpose of a class is to provide a structure for storing and organizing data in a program, and to provide a means of access and manipulation of that data. And also adding a clear and concise interface for interacting with the data, this would makes it more easier to apply data structure and ensure the data is used in a consistent way. I think we have learned using this **Class** in *0.1 C++ review*.



```
class my_user {
    public:
        int get_num_biketype(string selectedCase);
        int read_user(string selectedCase);

        int user_num = 0; // User total
        int num_biketype = 0; //Bike type total

        void mergesort(int *time, int **rentdata, int const begin, int const end);
        void merge(int *time, int **rentdata, int const left, int const mid, int const right);


        void splitIt(int **cantik, int coy, string selectedCase);
        void transfer_log(int** transfer,int **cantik, int coy, int bikeid, int start_t, int end_t);
        void transfer_notsort(int** transfer,int **cantik, int coy, int bikeid, int start_t, int end_t, string selectedCase);
        void rejected(int userid, int** user_result);

        void output_trf(string selectedCase, int** transfer, int** user_result, int is);

};
```

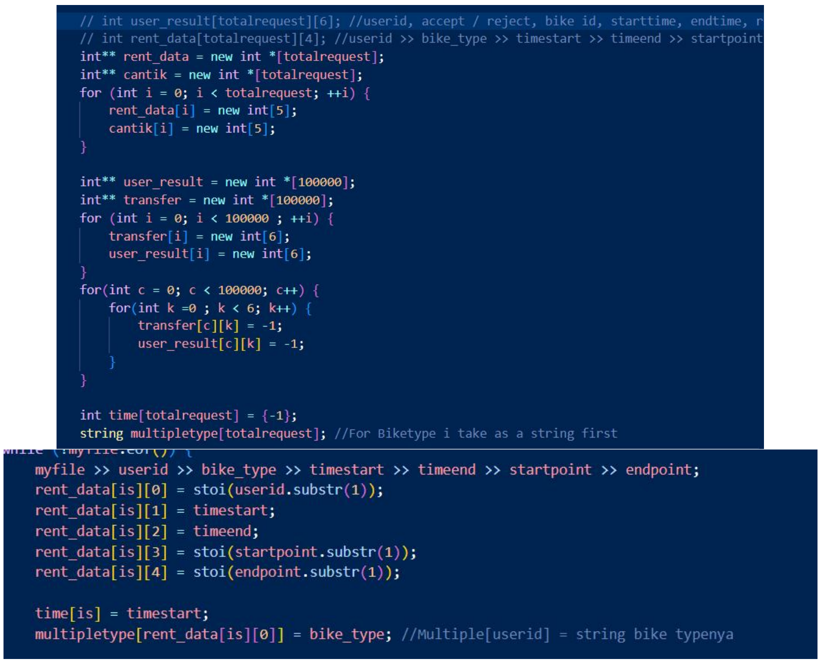


```
class map {

    public :
        int V;
        int **data;
        int flag;
        int **shortest;


        void graph(string selectedCase);
        int findVertices(string selectedCase);
        void addEdge(string selectedCase);
        int minDistance(int dist[], bool checked[]);
        int djikstra_LAMA(string selectedCase,int source, int destination);

        void djikstra_algo(string selectedCase);

        string station_start;
        string station_end;
        int distance;
};
```

```
class my_station {
    public :

    int **station = new int*[1000];
    my_station() {
        for(int i = 0; i < 1000; ++i) {
            station[i] = new int[10000];
        }
        for(int c = 0; c < 1000; c++) {
            for(int k =0 ; k <10000; k++) {
                station[c][k] = -1;
            }
        }
    }
    //Total Bike <= 10000, Bike type <= 10000 , All assign to -1
    int time[10000] = {-1}; //Time[Bikeid]
    float bike_rent[10000] ={-1}; //bike_rent[bikeid] = current rental price
    int rent_count[10000] ={-1}; //bike_used[bikeid] = rental couuter
    float initial_price[50] ={-1};//initial_price[bike_type] = initial price
```

Figure 2.1 <Example of Class In my code>

The second data structure that I used is **Array**. This is the most important ones I think in the system because of all the data from the input until output stored in the Array. Ofcourse, without the Array means Without data which means the system wont work cause theres nothing to process. As we know from what we learn, An array in data structures is a linear data structure that stores a sequence of elements of the same data type. Each element in the array is identified by an index, which is a numerical value that corresponds to the position of the element in the array. Why I am using array? I think for me Array is the one that popular inside my head and easy to understand, Its useful for storing and manipulate large amount of data cause Array allow for fast and efficient access to the value of it. I used 2 types of array in my code which are Common array , and Dynamic array. For the dynamic array in my code, are implemented by combining a dynamically allocated array and an array of pointers, where each element of the array points to a dynamically allocated array. A 2D array is an array of arrays, with each element of the array being itself an array. Dynamic 2D array is the one that I implemented a lot in my code.



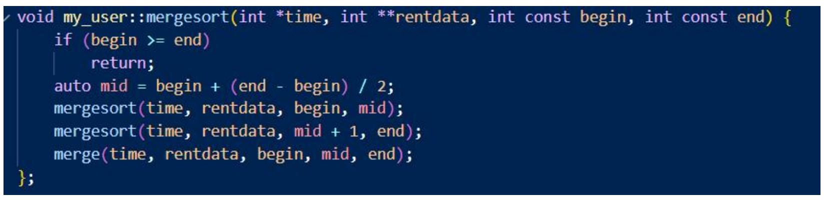
```cpp
// int user_result[totalrequest][6]; //userid, accept / reject, bike id, starttime, endtime, r
// int rent_data[totalrequest][4]; //userid >> bike_type >> timestart >> timeend >> startpoint
int** rent_data = new int *[totalrequest];
int** cantik = new int *[totalrequest];
for (int i = 0; i < totalrequest; ++i) {
    rent_data[i] = new int[5];
    cantik[i] = new int[5];
}

int** user_result = new int *[100000];
int** transfer = new int *[100000];
for (int i = 0; i < 100000 ; ++i) {
    transfer[i] = new int[6];
    user_result[i] = new int[6];
}
for(int c = 0; c < 100000; c++) {
    for(int k =0 ; k < 6; k++) {
        transfer[c][k] = -1;
        user_result[c][k] = -1;
    }
}

int time[totalrequest] = {-1};
string multipletype[totalrequest]; //For Biketype i take as a string first
```

```cpp
    myfile >> userid >> bike_type >> timestart >> timeend >> startpoint >> endpoint;
    rent_data[is][0] = stoi(userid.substr(1));
    rent_data[is][1] = timestart;
    rent_data[is][2] = timeend;
    rent_data[is][3] = stoi(startpoint.substr(1));
    rent_data[is][4] = stoi(endpoint.substr(1));

    time[is] = timestart;
    multipletype[rent_data[is][0]] = bike_type; //Multiple[userid] = string bike typenya
```

Figure 2.2 <Array Implementation>

As you can see in the Figure 2.2, Those are some examples the way I implemented Array in my code. And don't forget to delete the Array after processing all of the data (Not wasting any memory). In the given code snippet, an array called rent_data is being used to store data about bike rentals. rent_data is a 2D array of integers with a size of 100x5, meaning it can store 100 sets of data with each set containing 5 integers. The extracted and converted values are then stored in the rent_data array at the index is. rent_data[is][0] stores the user ID, rent_data[is][1] stores the start time, rent_data[is][2] stores the end time, rent_data[is][3] stores the start point, and rent_data[is][4] stores the end point. This code uses an array to store data about bike rentals in a structured and organized way, allowing the data to be easily accessed and manipulated.

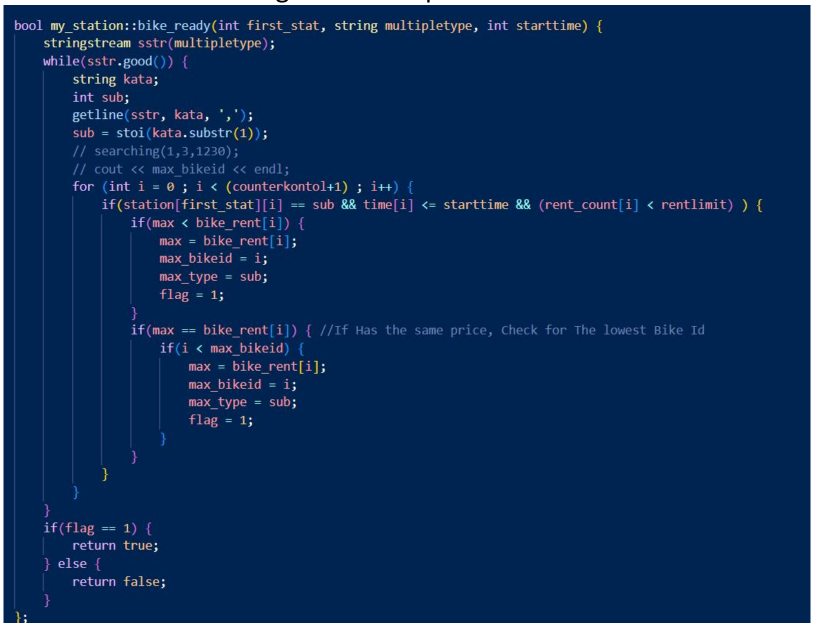
The third data structure that I used is **Merge Sort**. What Is Merge sort? It's a efficient sorting algorithm that divides an array / set data into two halves and sorting each half and combine them again into one. Why I choose Mergesort ? From what I have learned, Merge Sort is one of the fastest sort algorithm in Data structures, which has the time complexity O(N * Log N) either Worst or Best case scenario. O(N * Log N) is the best sort computing time . Because of that time complexity it is very efficient to handle the large inputs of data. And also, It's a stable sort which the system need when the starting time request are the same , then the system will took by the smallest user id . The project require the system to sort out the user request based on the time start.



```cpp
void my_user::mergesort(int *time, int **rentdata, int const begin, int const end) {
    if (begin >= end)
        return;
    auto mid = begin + (end - begin) / 2;
    mergesort(time, rentdata, begin, mid);
    mergesort(time, rentdata, mid + 1, end);
    merge(time, rentdata, begin, mid, end);
};
```

Figure 2.3 <MergeSort Implementation>

The fourth data structure that I implement in my code is **Linear Search.** I think for me Linear search is the simplest search algorithm to understand and suits perfectly with my system. Why i choose Linear search? Because For my system I tried using Binary search but it wont work cause Binary search can be use only on sorted list, and I know the fact that the time complexity is the main problem **O(N)** but After try using Linear search , I think its still possible using it , and If the list is not sorted and is expected to change frequently, it may not be worth the time and effort to keep it sorted just for the sake of using binary search. In such cases, linear search can be a good choice. My algorithm really suits with how linear search work cause my system need to search for multiple elements in a list, linear search may be more efficient, as system can stop the search as soon as you find the first element. With binary search, system would have to continue searching through the entire list to find all the elements. One of the example I am using Linear Search in My code for finding Bikes that suitable for the user and has the highest rental price.



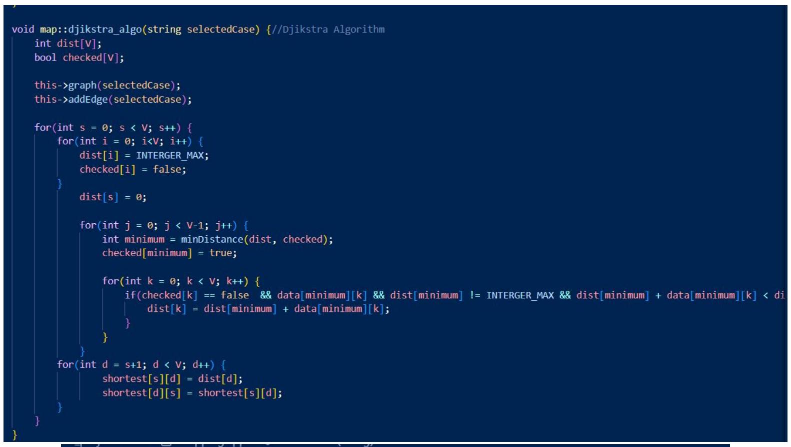
```
bool my_station::bike_ready(int first_stat, string multipletype, int starttime) {
    stringstream sstr(multipletype);
    while(sstr.good()) {
        string kata;
        int sub;
        getline(sstr, kata, ',');
        sub = stoi(kata.substr(1));
        // searching(1,3,1230);
        // cout << max_bikeid << endl;
        for (int i = 0 ; i < (counterkontol+1) ; i++) {
            if(station[first_stat][i] == sub && time[i] <= starttime && (rent_count[i] < rentlimit) ) {
                if(max < bike_rent[i]) {
                    max = bike_rent[i];
                    max_bikeid = i;
                    max_type = sub;
                    flag = 1;
                }
                if(max == bike_rent[i]) { //If Has the same price, Check for The lowest Bike Id
                    if(i < max_bikeid) {
                        max = bike_rent[i];
                        max_bikeid = i;
                        max_type = sub;
                        flag = 1;
                    }
                }
            }
        }
    }
    if(flag == 1) {
        return true;
    } else {
        return false;
    }
};
```

Figure 2.4 <Linear search Implementation>

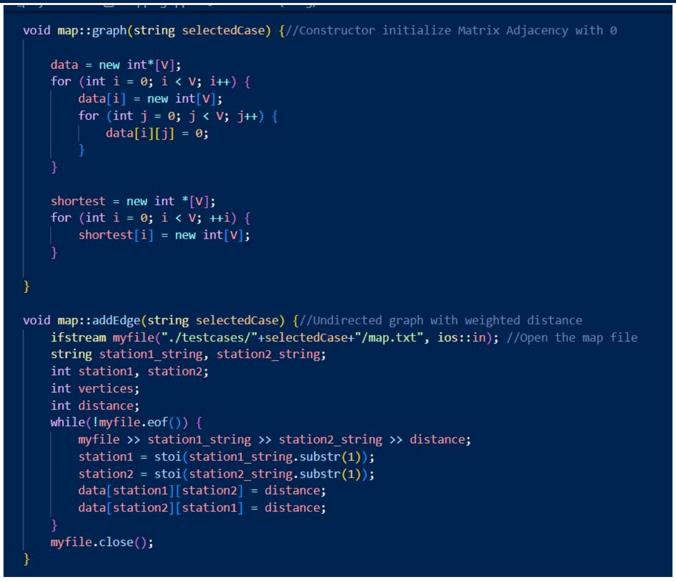
As you can see in the Figure *2.4,* I am using Linear search to find the suitable bikes that meet all the conditions like Not passing the rent limit, Theres no one using the bike , and Has the highest price between all the user choices.

The last data structure I implemented is **Dijkstra Algorithm (with Undirected Graphs).** As we know in the Input data file that given, the distance between Each station is all positive edge weight and non negatiive weight , so I just directly thinking choose Dijkstra Algorithm to find the guaranteed shortest path problem. Dijkstra algorithm is also efficient because has the time complexity **O(V log V + E)** which is suitable for use on large graphs. The way I implement it is first I need also to create **a Undirected graph** map for mapping all the stations location and the distance between each of them , As we know Graph consists of a finite set of vertices (also called nodes) and a set of edges connecting these vertices. And the way I represent **Graph** is in Adjacency Matrix way where each row and column represents a vertex, and the value at a particular row and column . Its powerful and easy to use.

After creating a **Graph** and initialize all of them with the Weight of each edges from the input file that given, then the system can start the process of calculating the shortest path between all of those station . The result of Dijkstra are stored in a dynamic 2d Array which the index of row is for the Departure station and the index of column is for the destination station. Both way are the same distance. Why I stored in array? Because the way I implement the program to start counting the shortest path of all possibilities of each stations and so when the user request from station A to station B, then the system don't need to calculating again for the entire request. Just only need one process.



```cpp
void map::djikstra_algo(string selectedCase) {//Djikstra Algorithm
    int dist[V];
    bool checked[V];

    this->graph(selectedCase);
    this->addEdge(selectedCase);

    for(int s = 0; s < V; s++) {
        for(int i = 0; i<V; i++) {
            dist[i] = INTERGER_MAX;
            checked[i] = false;
        }
            dist[s] = 0;

            for(int j = 0; j < V-1; j++) {
                int minimum = minDistance(dist, checked);
                checked[minimum] = true;

                for(int k = 0; k < V; k++) {
                    if(checked[k] == false  && data[minimum][k] && dist[minimum] != INTERGER_MAX && dist[minimum] + data[minimum][k] < di
                        dist[k] = dist[minimum] + data[minimum][k];
                    }
                }
            }
        for(int d = s+1; d < V; d++) {
                shortest[s][d] = dist[d];
                shortest[d][s] = shortest[s][d];
        }
    }
}
```

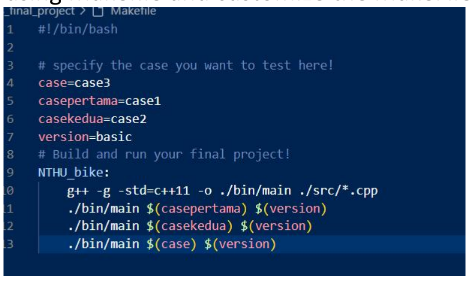


```cpp
void map::graph(string selectedCase) {//Constructor initialize Matrix Adjacency with 0

    data = new int*[V];
    for (int i = 0; i < V; i++) {
        data[i] = new int[V];
        for (int j = 0; j < V; j++) {
            data[i][j] = 0;
        }
    }

    shortest = new int *[V];
    for (int i = 0; i < V; ++i) {
        shortest[i] = new int[V];
    }

}

void map::addEdge(string selectedCase) {//Undirected graph with weighted distance
    ifstream myfile("./testcases/"+selectedCase+"/map.txt", ios::in); //Open the map file
    string station1_string, station2_string;
    int station1, station2;
    int vertices;
    int distance;
    while(!myfile.eof()) {
        myfile >> station1_string >> station2_string >> distance;
        station1 = stoi(station1_string.substr(1));
        station2 = stoi(station2_string.substr(1));
        data[station1][station2] = distance;
        data[station2][station1] = distance;
    }
    myfile.close();
}
```

Figure 2.5 <Djikstra and Graph Implementation>

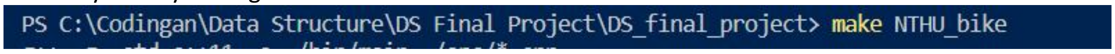
Chapter 3 : Execute the file

To execute my program, I am using Makefile and customize the MakeFile that given by the TA little bit.



```makefile
#!/bin/bash

# specify the case you want to test here!
case=case3
casepertama=case1
casekedua=case2
version=basic
# Build and run your final project!
NTHU_bike:
	g++ -g -std=c++11 -o ./bin/main ./src/*.cpp
	./bin/main $(casepertama) $(version)
	./bin/main $(casekedua) $(version)
	./bin/main $(case) $(version)
```

By Using the Makefile above, to run the program we can just call / input in the terminal " **make NTHU_bike** ". Then the program will automatically test for that 3 testcase and don't need to do it manually one by one again.



```
PS C:\Codingan\Data Structure\DS Final Project\DS_final_project> make NTHU_bike
```