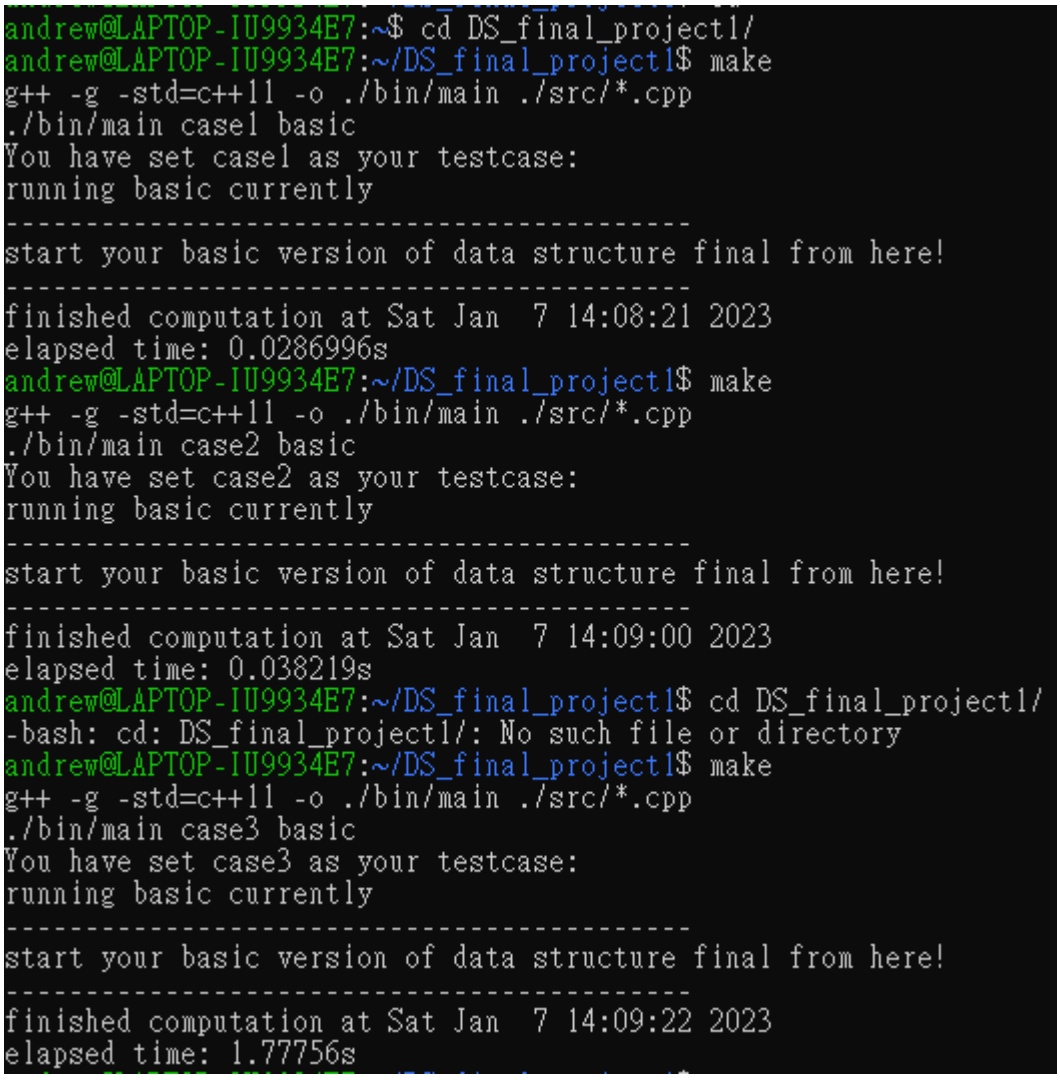
Final Project

Maximize Revenue for NTHU Bike Company

110062106 黎秉諺

1. execution example



先 cd 到所在的資料夾，也就是 DS_final_project1

然後打 make

(要先在 makefile 設定使用的 case 以及 version)

即可執行

2. The details of my data structures

首先是 bike:

```
struct bike
{
    int type;
    int id;
    int Rental_count;
    float Rental_price;
    int station;
    int cdtime=-1;
};
```

每台腳踏車我會先記錄它是哪個種類的腳踏車(B?)

它的 id

它的 rental_count(借幾次)

它的 rental_price(現在的價格)

它在哪一站(S?)

以及他上次歸還的時間

```
struct userrequest2
{
    int id;
    int oknum=0;
    int okay[1010]={0};
    int start_time;
    int end_time;
    int start_pt;
    int end_pt;
    int bike_startpt;
    int bike_transferstartpt;
    int bike_transferendpt;
    int bike_arrive=0;
    int act_end=0;
    int money=0;
    int bikeid;
    int fulfill=0;
    bike2 ubike;
};
```

而 user 的部分我會記錄他的 id

Oknum 是能接受腳踏車種類的數量

Okay 是儲存了他所有能接受的腳踏車種類

Start_time 是他發出 request 的時間

End_time 是能接受最晚的抵達時間

Start_pt 和 end_pt 是起點以及終點

Bike_startpt 是紀錄腳踏車原本在哪一站(transferstartpt 跟 transferendpt 是 debug 用)

Bike_arrive 是紀錄 user 幾分等得到腳踏車(free bike transfer)

如果為 0 代表沒有經過 free bike transfer

Act_end 是實際到達的時間

Money 是所花的錢

Bikeid 是 bike 的 id

Fulfill 是有沒有借到車

Ubike 是借到的車子

```
struct station2
{
    int id;
    int* cap;
    bike2** bk;
    int* bknum;
};
```

Station 的部份的話:

Id 顧名思義(S?)

Cap 是一維的 int 陣列 紀錄每個 type 腳踏車目前的最大容量

Bk 則記錄站裡的所有腳踏車(bk[種類][第幾輛])

Bknum 紀錄每一種車子當前的數量

```
station2* stat2;
userrequest2 user_request2[100010];
bike2 all_bike2[10010];
bike2 all_bike_by_id2[10010];
bike2 unusable2[10010];
```

一些拿來記錄腳踏車以及 user_request

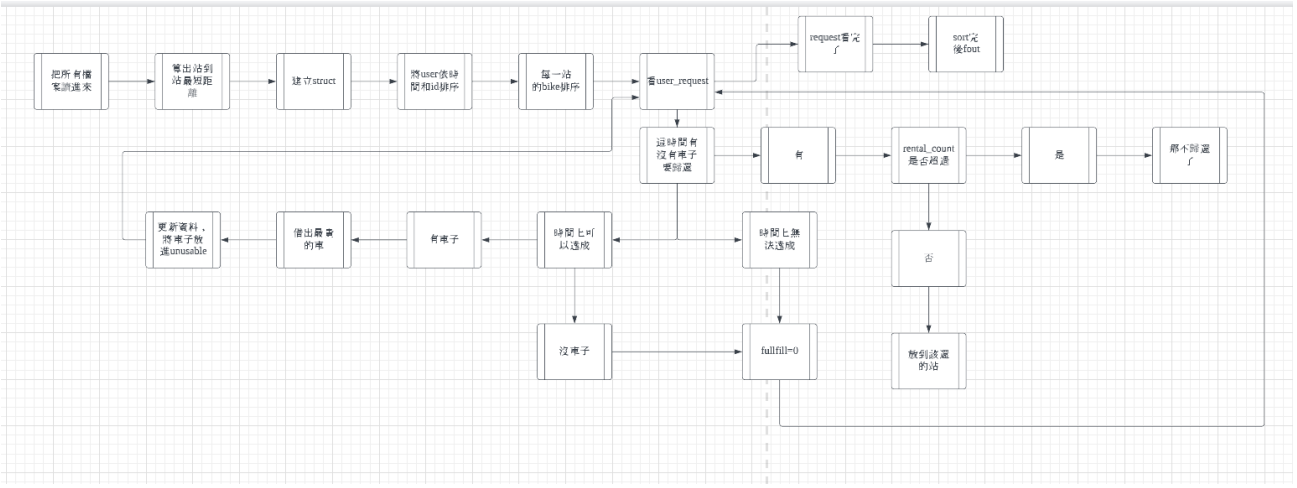
這之中的 unusable 是記錄著所有被借出但尚未歸還的車子

```
int distance[1001][1001];
int min_distance[1001][1001];
```

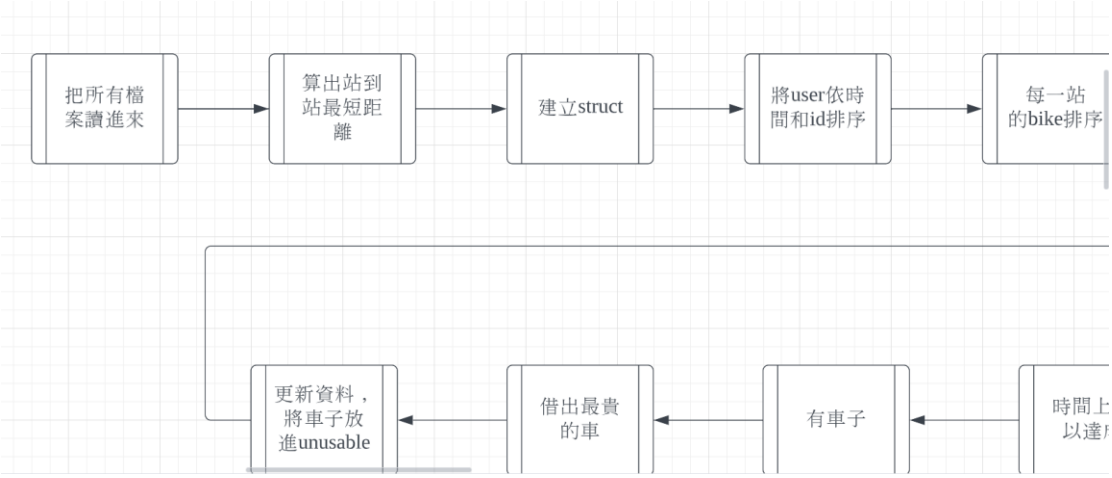
Distance 以及 min_distance 分別是站到站的距離以及最短距離

Detail，flowchart:

Basic_flowchart



整個 project 的主要概念如上圖




```
for (int i = 0; i<=num; i++)
{
    for (int j = 0; j<=num; j++)
    {
        min_distance[i][j]= distance[i][j];
    }
}
for (int k= 0; k<=num; k++)
{
    for (int i= 0; i<=num; i++)
    {
        for (int j= 0; j<=num; j++)
        {
            if((min_distance[i][k]+min_distance[k][j])<min_distance[i][j])
            {
                min_distance[i][j] = min_distance[i][k] + min_distance[k][j];
            }
        }
    }
}
```

首先是算出最短路徑的部分

我使用了 Floyd-Warshall’s Algorithm 來達成

跑一些 for 迴圈即可得到站到站的最短路徑

Code 實作如上圖

```
inline void QuickSort_user(userrequest *a, const int left, const int right)
{
    bool bFlag = false;
    if (left < right) {
    int i = left;
    int j = right + 1;
    userrequest pivot = a[left];

    // if (left==997)
    //   bFlag = true;

    do
    {
       do i++; while ( ( (a[i].start_time < pivot.start_time) || ( (a[i].start_time == pivot.start_time) && (a[i].id<pivot.id) ) )&&i<user_num);
       do j--; while ( ( (a[j].start_time > pivot.start_time) || ( (a[j].start_time == pivot.start_time) && (a[j].id>pivot.id) ) )&&j>0);
    //    if (bFlag)
    //        cout<<i<<" "<<j<<endl;

       if (i < j)
       {
        userrequest tmp=a[i];
        a[i]=a[j];
        a[j]=tmp;
       }
    } while (i < j);

    userrequest tmp=a[left];
    a[left]=a[j];
    a[j]=tmp;
     QuickSort_user(a, left, j - 1);
//    cout<<left<<" "<<j<<endl;
    QuickSort_user(a, j + 1, right);
}
}
```

Sort 的部分我主要是用 quick sort

讓我可以按照自己想要的優先級進行排序

例如上圖我就先考慮 start_time 再考慮 id

是十分方便的

```
if(stat[statid].bknum[bktype]>=stat[statid].cap[bktype])
{
    bike* tmp=new bike[(stat[statid].cap[bktype])*2+1];
    for(int j=0;j<=stat[statid].bknum[bktype];j++)
    {
        tmp[j]=stat[statid].bk[bktype][j];
    }
    delete[] stat[statid].bk[bktype];
    stat[statid].bk[bktype]=tmp;
    stat[statid].cap[bktype]=stat[statid].cap[bktype]*2;
}
```

除此之外我覺得值得一提的是 station 中 bike 的儲存方式

因為如果一開始就在 station 中開 1000*10000 的 bike 會太大超出記憶體限制

所以我讓它是以一種 malloc 的方式來實作

一開始開小一點

當儲存超過 cap 時(超出目前大小)

我就讓 cap 來乘以 2

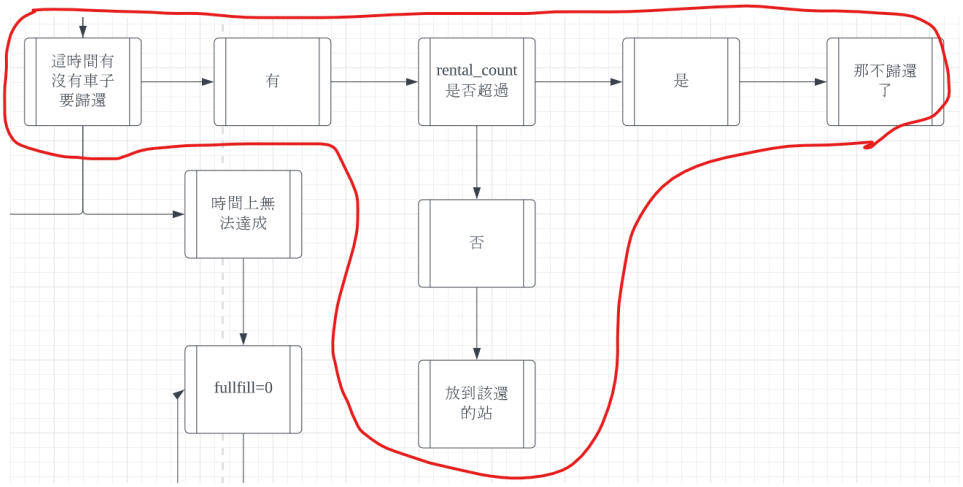
藉由這樣來節省記憶體

```
int thenewbike=unsable_num-1;
while(thenewbike>0)
{
    if(all_bike_by_id[unusable[thenewbike].id].cdtime>all_bike_by_id[unusable[thenewbike-1].id].cdtime||(all_bike_by_id[unusable[thenewbike].id].cdtime==all_bike_by_id[un
    {
        swap2(&unusable[thenewbike],&unusable[thenewbike-1]);
        thenewbike--;
    }
    else
    {
        break;
    }
}

int e=unsable_num-1;
while(all_bike_by_id[unusable[e].id].cdtime<=user_request[i].start_time&&e>=0)
{
    if(all_bike_by_id[unusable[e].id].Rental_count<count_limit)
    {
        if(stat[all_bike_by_id[unusable[e].id].station].bknum[all_bike_by_id[unusable[e].id].type]>=stat[all_bike_by_id[unusable[e].id].station].cap[all_bike_by_id[unusab
        {

            int statid=all_bike_by_id[unusable[e].id].station;
            int bktype=all_bike_by_id[unusable[e].id].type;
            bike* tmp2=new bike[(stat[statid].cap[bktype])*2+1];
            for(int j=0;j<=stat[statid].bknum[bktype];j++)
            {
                tmp2[j]=stat[statid].bk[bktype][j];
            }
            delete[] stat[statid].bk[bktype];
            stat[statid].bk[bktype]=tmp2;
            stat[statid].cap[bktype]=stat[statid].cap[bktype]*2;
        }

        stat[all_bike_by_id[unusable[e].id].station].bk[unusable[e].type][stat[all_bike_by_id[unusable[e].id].station].bknum[unusable[e].type]]=all_bike_by_id[unusable[e]
        stat[all_bike_by_id[unusable[e].id].station].bknum[unusable[e].type]++;
        station tmp=stat[all_bike_by_id[unusable[e].id].station];
        QuickSort_bike(tmp.bk[unusable[e].type],0,tmp.bknum[unusable[e].type]-1);
        for(int y=0;y<tmp.bknum[unusable[e].type];y++)
        {
            stat[all_bike_by_id[unusable[e].id].station].bk[unusable[e].type][y]=tmp.bk[unusable[e].type][y];
        }
    }
    unsable_num--;
    e--;
}
```



還有一個部分是我會記錄所有借出中的 bike

按到站時間排序

每次有新的 user_request 進來時(user_request 已經照時間以及 id 排序)

我會先查看它來的時間

然後看哪些借出中的車子可以還回去

實作如上

```
user_request[i].fulfill=0;
if(min_distance[user_request[i].start_pt][user_request[i].end_pt]+user_request[i].start_time>=user_request[i].end_time)
{
    user_request[i].fulfill=0;
}
else
{
    station startstation=stat[user_request[i].start_pt];
    float max=-1;
    int flag=0;
    bike candi;
    for(int j=0;j<user_request[i].oknum&&!flag;j++)
    {
        if(startstation.bknum[user_request[i].okay[j]]>0) flag=1;
    }
    if(flag==0)
    {
        user_request[i].fulfill=0;
    }
    else
    {
       for(int j=0;j<user_request[i].oknum;j++)
       {
        int the_type=user_request[i].okay[j];
        if(startstation.bknum[user_request[i].okay[j]]!=0)
        {
          if(startstation.bk[the_type][startstation.bknum[the_type]-1].Rental_price>max||(startstation.bk[the_type][startstation.bknum[the_type]-1].Rental_price==max&&
          {
            candi=startstation.bk[the_type][startstation.bknum[the_type]-1];
            max=startstation.bk[the_type][startstation.bknum[the_type]-1].Rental_price;
          }
        }
       }
    user_request[i].fulfill=1;
    user_request[i].bikeid=candi.id;
    user_request[i].act_end=user_request[i].start_time+min_distance[user_request[i].start_pt][user_request[i].end_pt];
    user_request[i].money=(user_request[i].act_end-user_request[i].start_time)*candi.Rental_price;
    stat[user_request[i].start_pt].bknum[candi.type]--;
    all_bike_by_id[candi.id].Rental_count++;
    all_bike_by_id[candi.id].Rental_price-=discount;
    all_bike_by_id[candi.id].station=user_request[i].end_pt;
    all_bike_by_id[candi.id].cdtime=user_request[i].act_end;
    unusable[unsable_num]=all_bike_by_id[candi.id];
    unsable_num++;
    user_request[i].ubike=candi;
    }
```

借車的地方我是會先看從時間上我有沒有機會滿足這個 request

如果沒可能我就直接回絕

如果有機會的話我就查看在 user 可接受的車種中

初始站有沒有剩下的車

如果沒有我就回絕

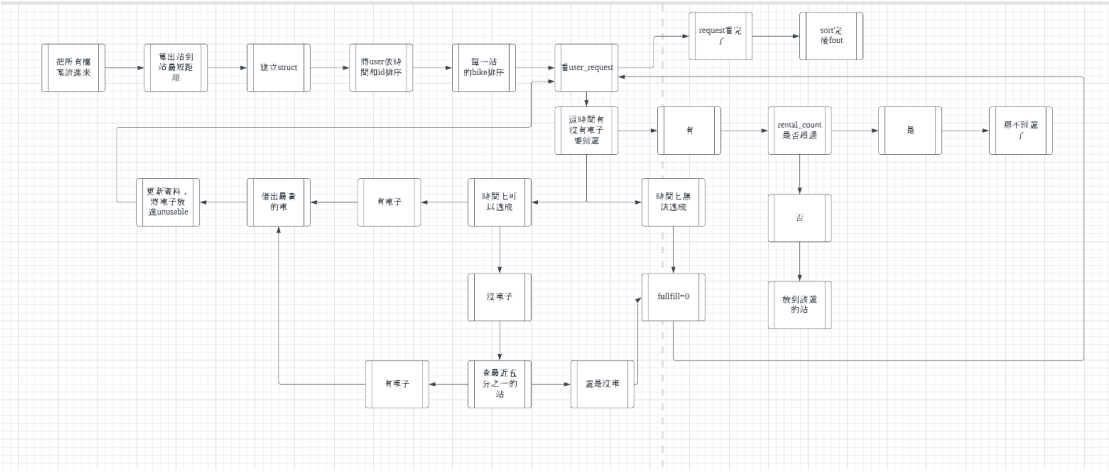
有的話我就挑最貴的車給他

因為我的 bike 是 sort 過的 所以只須取出該種類最後一台車即是最貴的

然後將車子借出後放入 unusable(借出的車的 array)

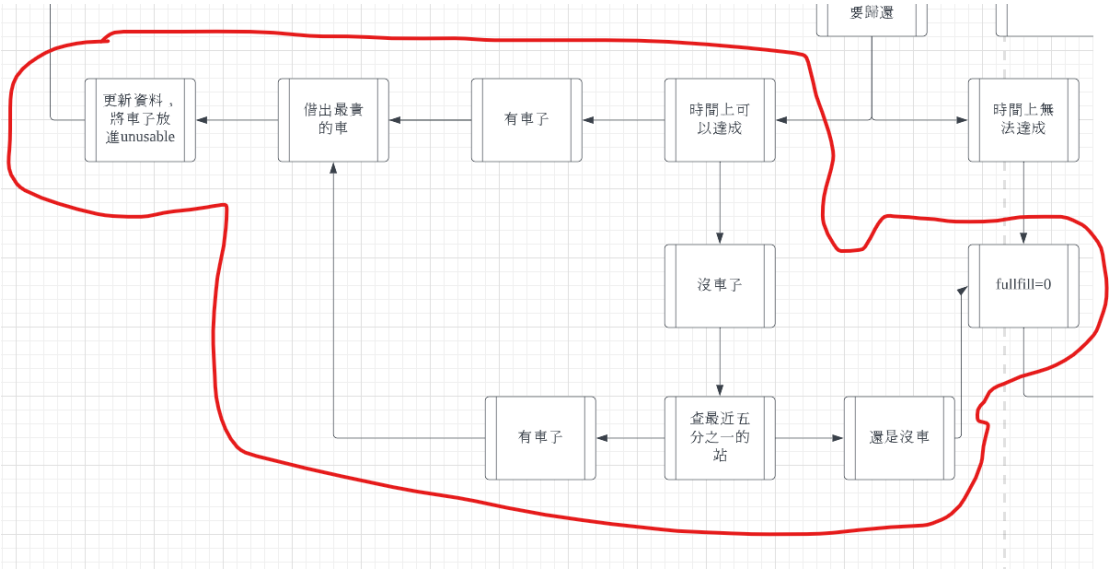
至於 ofstream 的部分就是將先前儲存好的資料表現出來，沒什麼特別的

Advance_flowchart



Advance 的部分基本上大同小異

原則上就是多了 free bike transfer



借車時

如果原本車站有車那就用原本車站的就好了

但如果沒有

我會看最近的 1/5 的站

(例如總站數 100，我就看離他最近的 20 個站有沒有車)

有車的話我就挑最貴的

從 free bike transer 到 user 使用完我都視為借出，不同時借給其他人

但 free bike transfer 的部分我是不收錢的

這樣會增加 fullfill request 的機會

減少車子不夠的可能

關於這次 project 我想說的話

首先辛苦助教們以及教授了，這 project 很複雜要注意的點也很多，想必出題以及 demo 會是十分麻煩的，但我是覺得 spec 有點改的太頻繁了，對於提早作的同學會造成蠻大的困擾，畢竟除了 spec 沒有其他關於這次作業的提示，大家都是仰賴 spec 在寫的，初始版本裡 spec 的錯誤讓我們一開始寫起來會有些困難，然後要內容真的有點太複雜了，而且還有時間限制，感覺有些地方做一點刪減會比較剛好，雖然寫起來蠻過癮的(+很累)，除此之外沒什麼不 ok 的地方了，感謝，謝謝帶給我這麼充實的一學期!!!