

Lab6 (航线规划系统设计与实现) 实验报告

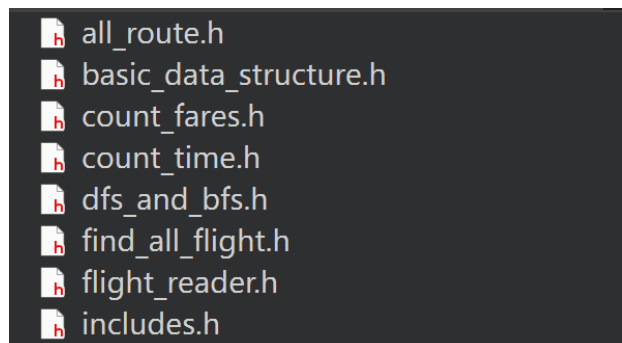
一、实验目的

利用图结构、邻接矩阵和邻接表的基本知识，根据给定的 XM 航空公司的现有航线数据，构造该公司的航线图。在此基础上，设计一个简易的航线规划系统。该系统可根据用户提出的不同需求，返回航班线路的解决方案。

二、功能介绍

1. 完成从任意机场出发的深度优先遍历，输出所有能够到达的机场编号及路径，所有主要路径中所经机场不能重复。
2. 完成从任意机场出发的广度优先遍历，输出所有能够到达的机场编号及路径，所有主要路径中所经机场不能重复。
3. 仅限直飞或 1 次中转，判断任意两个机场的可连通性，并给出 Flight ID 顺序表。
4. 求任意两个机场之间的最短飞行时间（含转机停留时间）。输出 Flight ID，总飞行时间，总中转时间，出发时间，达到时间。
5. 给定起飞时段或者降落时段要求，求经过直飞或者 1 次转机，任意两个机场的多个备选航线。
6. 给定起飞时段或者降落时段要求，求经过直飞或者 1 次转机，任意两个机场之间的最低航费及对应航线。

三、整体思路



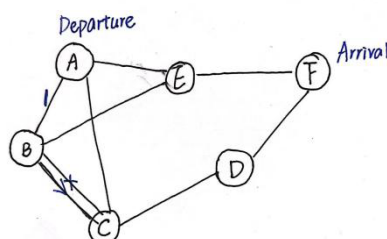
整体程序由 7 个主要的头文件和一个主函数组成：flight_reader 对 csv 文件数据进行读取，

将其存储到对应的结构体中并修改时间格式；dfs_and_bfs 利用 C++ 栈操作实现深度优先搜索，利用 C++ 队列操作实现广度优先搜索；find_all_flight 根据输入的起点和终点，进行所有路径的深度优先搜索，输出直达或一次中转的航班表至 result.txt 文件，并将路径存储到指定数组；count_fares 和 count_time 利用已经存储的 all_route 路径数组信息，计算每一路径所需要的时间和花费，从而选出最优路径。

四、详细设计及关键代码

1. 功能点 1、2 (DFS/BFS)

读取文件后，将每一条航班信息存入结构体内，结构体的内容作为每个信息块的 data 内容。DFS 采用栈操作进行，判断两机场是否相联通时，由于不需选择最短时间或花费的路径，因此判断过程中，在符合 B 机场出发到 C 的时间晚于 A 机场到 B 机场的时间的所有路径中，选择时间最早的作为出发时间，从而将两个机场之间多组航线数据简化为只有一条最优航线数据，从而实现 DFS。例如图中从 B 到 C 的两条路径都符合时间要求，则选取出发时间最早的路径作为最优路径。



由于向下深度搜索时需要满足时间先后的要求，程序用二维数组 time_char[ROUTE_NO][CHAR_LEN]存储在目前搜索路径第 ROUTE_NO 位置的机场，其抵达时间的字符串，从而在栈顶节点搜索为空退栈时，方便获得剩余未退栈节点的时间信息。

深度优先遍历和广度优先遍历的基本思路和所使用的判断是否联通的布偶函数相同，DFS 使用栈进行辅助操作，BFS 使用队列进行辅助操作。

```

17 //判断两个机场之间路线、时间是否连通,并记录最终取arrival的到达时间
18 bool if_connected (int dep_apt_num,int arr_apt_num,char last_arr_time[100],
19 struct information_of_flight ioof[100][100])
20 {
21     if(ioof[dep_apt_num][arr_apt_num].flight_num != 0)//如果两地之间有飞机
22     {
23         for(int i=1;i<=ioof[dep_apt_num][arr_apt_num].flight_num;i++)//遍历两地之间的航班
24         {
25             if(strcmp(last_arr_time,ioof[dep_apt_num][arr_apt_num].dep_time[i])<=0)
26             //找到两地之间符合时间要求的最早航班
27             {
28                 //记录到达终点机场的最早航班的到达时间进入数组
29                 return true;
30             }//end if strcmp
31         }//end for
32         return false;
33     }//end if !=0
34     else return false;
35 }//end bool

```

图 1 两机场是否连通的判断函数

```

72 //DFS
73 void dfsVisit(struct information_of_flight ioof[100][100],int dep_apt)
74 {
75     stack<int> dfs_stack;//栈
76
77     cout<<"The DFS departure id:"<<endl;
78     visited[dep_apt] = 1;
79     strcpy(time_char[dep_apt],"000000000000");//出发机场的到达时间为0,初始化
80     dfs_stack.push(dep_apt);
81     //whole_dfs
82     //cout<<dep_apt;
83     int pop_ele = 0;
84
85     while(!dfs_stack.empty())
86     {
87         pop_ele = dfs_stack.top();
88         for(int i=1;i<=80;i++)//遍历 pop_ele的邻接,获得 pop_ele的第一个未访问过的邻接
89         {
90             if(if_connected(pop_ele,i,time_char[pop_ele],ioof) && visited[i] == 0 )
91             {
92                 visited[i] = 1;
93                 dfs_stack.push(i);
94                 //whole_dfs
95                 //cout<<"->"<<i;
96                 pop_ele = dfs_stack.top();
97                 break;
98             }//end if
99
100             if(i == 80)//不存在相邻且未访问的节点
101             {
102                 //打印存储全部栈内容,即要到达出栈点的详细全部路径
103                 dfs_print[0] = dfs_stack.size();
104                 stack_route_print(dfs_stack);
105                 //dfs基本操作
106                 cout<<endl;
107                 dfs_stack.pop();
108             }//in if == 80
109         }//end for
110     }//end while
111 }//end dfsVisit

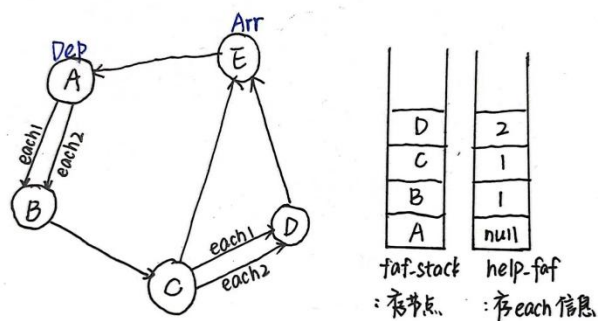
```

图 2 深度优先搜索函数-栈操作

2. 功能点 3（直飞或中转一次的所有路径）

由于下方功能点有对出发、到达时间要求的限制,因此在功能点 3 的深搜函数中,程序提前加入时间限制的参数,并在主函数调用功能 3 时,将函数参数赋值为“000000000000”和“999999999999”,保证所有出发时间均在这一个范围内,以消除时间约束。

功能点 3 之后都需要输出精准的航班 ID，可以说功能点 1、2 都是对机场节点的连通性进行深度搜索，而功能点 3 之后则是对不同航线（包含不同机场节点、不同飞行时间）的连通性进行深度搜索。由于航线包含了机场节点和两机场节点（faf 和 i）之间的所有路径，因此在 DFS 过程中，除了存放机场节点信息的 faf_stack 外，还引入辅助栈 help_stack 存放两个机场节点之间的航线编号。每一次向深一层搜索时，实质上是遍历所有路径，若联通，则机场编号入左栈（如下图），航线编号入右栈，以完成一层深度遍历。



两个航线相连通的条件是 1) 两机场相联通 2) 时间先后顺序符合 3) 该路径未被访问过 4) 该节点机场未出现在现在正在搜索的路径中，程序通过借助三维数组 faf_visited[DEP_ID][ARR_ID][BRANCH_NO] 判断路径是否被访问过，前两维代表从机场编号 DEP 到机场编号 ARR 的路径，BRANCH_NO 代表该两机场间的路径编号，从而精确到具体的路径 ID。判断机场是否出现在正在搜索的路径中时，程序借助 save 数组临时存放栈中信息，并且在打印路径时也采用类似方法。

```

51 //faf新进栈的航班不能出现在已存在的路径中
52 bool judge_repete(int faf_stack<int> faf_stack)
53 {
54     int faf_save[100];
55     int ini_size = faf_stack.size();
56
57     if(ini_size == 1)
58     {
59         //void
60         return true;
61     }
62
63     else
64     {
65         for(int i=1; i <= ini_size; i++)
66         {
67             faf_save[i] = faf_stack.top();
68             faf_stack.pop();
69         }
70
71         for(int j=ini_size; j>=1; j--)
72         {
73             if(j == ini_size)
74             {
75                 faf_stack.push(faf_save[j]);
76             }
77             else
78             {
79                 faf_stack.push(faf_save[j]);
80             }
81         }
82         //不是出发点
83
84         for(int m=1; m<=ini_size; m++)
85         {
86             if(faf == faf_save[m]) return false;
87         }
88
89         return true;
90     }
91 }

```

图 3 判断机场是否重复出现在路径中的函数

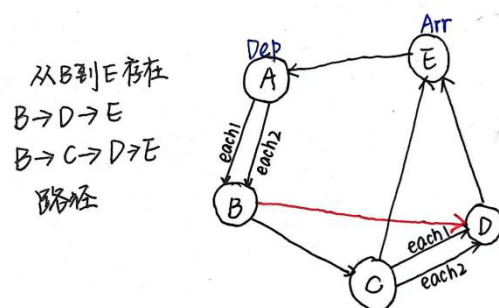


图 4 路径示意图

当栈顶继续搜索结果为空，即没有符合要求的路径可以入栈时，将栈顶退栈，包括主栈（存放机场节点编号）和辅助栈（存放路径编号）。由于航线有向图存在多种路径的情况，例如图 4 中从 B 到 E 存在的 B->D->E 和 B->C->D->E 两种路径，若先搜了 BDE 的路径，则 DE 之间的路径会被标记，如果在 D 退栈时不删除这个标记，则后续搜索时，由于路径已经被标记过，则不会再搜索出 BCDE 的路径，从而出现错误。因此，程序在每一个栈顶节点退栈时，删除退栈节点与所联通的所有其他节点之间的标记。

```

39 void delete_all_flag(int faf, struct information_of_flight iof[100][100])
40 {
41     for(int i=1; i<=79; i++)
42     {
43         for(int each=1; each<=iof[faf][i].flight_num; each++)
44         {
45             faf_visited[faf][i][each] = 0;
46         }
47     }
48 }

```

图 5 删除退栈节点与其他节点的路径标记

在搜索时，由于题目限制中转一次或直飞，因此存放路径的栈的大小不得超过 3，即起点、中转站、终点，因此在搜索时，通过限制入栈元素个数来控制路径长度，总体搜索函数如下。

```

160 void find_all_flight(int dep_id, int arr_id, struct information_of_flight info[100][100],
161                     char pre_time_order[100], char aft_time_order[100])
162 {
163     //cout<<"in find all flight"<<endl;
164     strcpy(time_faf[dep_id], "000000000000");
165     //主栈里面存放连接的机场ID
166     stack<int> faf_stack;
167     //辅助栈中存放主栈对应的机场的第N次航班信息
168     stack<int> faf_help;
169     // 没有含义，占位
170     faf_stack.push(dep_id);
171     faf_help.push(0);
172     int faf = 0; // 栈顶元素
173     int help = 0; // help 栈顶元素
174     if(dep_id == arr_id) // 输入限制
175     {
176         cout<<"You are at your target airport!"<<endl;
177     }
178     else
179     {
180         // 栈深搜，打印所有路径
181         while(!faf_stack.empty())
182         {
183             if(faf_stack.size()==1) // 如果回到起始点是因为退栈，新一轮深搜
184             {
185                 strcpy(time_faf[dep_id], "000000000000");
186             }
187             faf = faf_stack.top();
188             help = faf_help.top();
189             if(faf == arr_id) // 如果栈顶元素是要到达的机场，那么打印栈里面的对应路径
190             {
191                 //cout<<"find!"<<endl;
192                 faf_stack.route_print(dep_id, arr_id, faf_stack, faf_help, info);
193                 //dfs基本操作
194                 //cout<<endl;
195                 delete_all_flag(faf, info);
196                 faf_stack.pop(); // 弹出栈顶元素
197                 faf_help.pop(); // 弹出栈顶元素
198             }
199         }
200     }
201 }

```

```

207     else
208     {
209         for(int i=1;i<=80;i++)//遍历每一个邻接点
210         {
211             if( faf == i ) continue;
212             int cir_flag = 0;//用于跳出两层循环
213
214             for(int each=1;each <= info[faf][i].flight_num;each++)//遍历邻接点中的每一个
215             {
216                 if(faf_stack.size()>1 || (faf_stack.size() == 1 &&
217                     strcmp(info[faf][i].dep_time[each].pre_time_order)>=0
218                     && strcmp(info[faf][i].dep_time[each].aft_time_order<=0))
219                 {
220                     //cout<<"i:"<<i<<" each:"<<each<<endl;
221                     if(plus_connected(faf,i,time_faf[faf],each,info) && faf_visited[faf][i][each] == 0
222                         && judge_repete(i,faf_stack) && faf_stack.size() <=2 )
223                     {
224                         //cout<<"in if"<<endl;
225                         //cout<<"i:"<<i<<" each:"<<each<<endl;
226                         //cout<<"size:"<<faf_stack.size()<<endl;
227                         //cout<<"pre_time_order"<<" "<<aft_time_order<<endl;
228
229                         faf_visited[faf][i][each] = 1;//标记被访问过的路径
230
231                         strcpy(time_faf[i],info[faf][i].arr_time[each]);
232                         //cout<<"flag:"<<info[49][57].arr_time[1]<<endl;
233
234                         faf_stack.push(i);
235                         faf_help.push(each);
236
237                         faf = faf_stack.top();
238                         help = faf_help.top();
239
240
241                         //cout<<1<<endl;
242
243                         cir_flag = 1;
244                         break;
245                     }
246                 }
247             }
248             //if
249
250             //for each
251
252             if(cir_flag == 1) break;
253
254             if(i==80)
255             {
256                 //去除即将出栈点到后面的所有路径
257                 faf = faf_stack.top();
258                 help = faf_help.top();
259                 delete_all_flag(faf,info);
260                 //出栈
261                 faf_stack.pop();
262                 faf_help.pop();
263                 //break;
264
265             }
266             //if i==80
267
268             //for i
269
270         }
271     }
272 }
273
274 //cout<<"There's no route !"<<endl;
275
276 }
277 }
278 }

```

图 6 功能点 3（题目中的 2）深搜代码

3. 功能点 4（无时间限制的最短飞行时间）

将功能点 3 求得的所有路径存入数组，遍历所有路径，借助 up 和 down 数组，计算其总体飞行时间、中转时间等。

```

107 void count_time(int dep,int arr,struct information_of_flight iof[100][100])
108 {
109     //OK
110     save_time(dep,arr,iof);
111
112     //cout<<"in count_time"<<endl;
113     for(int i=1;i<route_row;i++)
114     {
115         for(int j=1;j<route_length[i];j++)
116         {
117             if(j != route_length[i]-1)
118             {
119                 sum_up[i] += (timechar_to_minute(down[i][j]) - timechar_to_minute(up[i][j]));
120                 sum_down[i] += (timechar_to_minute(up[i][j+1]) - timechar_to_minute(down[i][j]));
121             }
122             else
123             {
124                 sum_up[i] += (timechar_to_minute(down[i][j]) - timechar_to_minute(up[i][j]));
125             }
126         }
127         sum_fly[i] = sum_up[i] + sum_down[i];
128         //cout<<i<<" "<<up[i][1]<<" "<<down[i][route_length[i]-1]<<" "<<sum_fly[i]<<endl;
129     }
130     int min_i = choose_min_time(sum_fly);
131     //打印 ID
132 }
133
134 //cout<<"after for"<<endl;
135
136

```

图 7 时间计算关键代码

4. 功能点 5 (有时间限制的所有路径)

由于限制出发时间，程序将输入的限制（不早于和不晚于）作为参数传入深搜函数，在判断是否入栈的语句中，增加条件：对于第一条路径，如果起飞时间在允许的范围内则入栈，不符合不入栈。

```

for(int each=1;each <= info[faf][i].flight_num;each++)//遍历邻接点中的每一个
{
    if(faf_stack.size()>1 || (faf_stack.size() == 1 &&
        strcmp(info[faf][i].dep_time[each],pre_time_order)>=0
        && strcmp(info[faf][i].dep_time[each],aft_time_order)<=0))
    {
        //cout<<"i:"<<i<<" each:"<<each<<endl;
        if(plus_connected(faf,i,time_faf[faf],each,info) && faf_visited[faf][i][each] == 0
            && judge_repete(i,faf_stack) && faf_stack.size() <=2 )
        {

```

图 8 起飞时间条件限制

5. 功能点 6 (有时间限制的最小花费)

在功能点 5 的基础上，将路径信息存入路径数组中，遍历所有路径，求出最小花费。

```

48 void count_fares(int dep,int arr,struct information_of_flight iof[100][100])
49 {
50
51     for(int i=1;i<route_row;i++)
52     {
53         for(int j=1;j<route_length[i];j++)
54         {
55             fare[i] += id_to_fare(iof,all_route[i][j]);
56         }
57     }
58
59     int min_fare_i = choose_min(fare);
60     //打印最少花费

```

图 9 花费计算关键代码

6. 主函数（面向使用者进行功能选择）

```
10 int main()
11 {
12     flight_reader();
13     write_map();
14     int dep_id;
15     int arr_id;
16     char pre_ord[100];
17     char aft_ord[100];
18
19     //选择功能
20     int choice;
21     cout<<"-----Welcome to Flight Tracker!-----"<<endl<<endl;
22     cout<<"(DIRECTIONS)"<<endl;
23     cout<<"1 for only dfs, inputting your departure id, and I offer all airports you can reach;"<<endl;
24     cout<<"2 for only bfs, inputting your departure id, and I give you another algorithm;"<<endl;
25     cout<<"3 for fetching all route (direct flight or one stopover), "<<
26     endl<<"inputting your departure and arrival id, "
27     "and I offer you all the route you can choose;"<<endl;
28     cout<<"4 for minimum flight time,"<<endl<<"inputting your departure and arrival id, "
29     "I count for the minimum time you need to spend;"<<endl;
30     cout<<"5 for specific time intervals, "<<endl<<"inputting two IDs and departure time interval"
31     "and I offer you all the route you can choose;"<<endl;
32     cout<<"6 for minimum cost, inputting your departure and arrival id,"
33     "I count for the minimum fares you need to cost;"<<endl;
34     cout<<"-----"<<endl<<endl;
35
36     //输入并调用相应功能
37     cout<<"NOW INPUT YOUR CHOICE OF FUNCTION : "<<endl<<endl<<"number:";
38     cin>>choice;
39     cout<<endl<<endl;
```

```
41     if(choice == 1)//1 for only dfs {...}
42
43
44
45
46
47
48     else if(choice == 2)//2 for only bfs {...}
49
50
51
52
53
54
55     else if(choice == 3)//3 for fetching all route (direct flight or one stopover) {...}
56
57
58
59
60
61
62     else if(choice == 4)//4 for minimum flight time {...}
63
64
65
66
67
68
69     else if(choice == 5)//5 for specific time intervals {...}
70
71
72
73
74
75
76     else if(choice == 6)//6 for minimum cost {...}
77
78
79
80
81
82
83     else {...}
84
85
86     cout<<"PROGRAM OVER!"<<endl;
87     return 0;
88 }
```

五、运行结果展示

1. 功能点 1

输入：3

输出：从机场 3 开始的所有深度搜索路径

```
CA\QtTools\QtCreator\bin\qtcreator_process_stub.exe
-----Welcome to Flight Tracker!-----

(DIRECTIONS)
1 for only dfs, inputting your departure id, and I offer all airports you can reach;
2 for only bfs, inputting your departure id, and I give you another algorithm;
3 for fetching all route (direct flight or one stopover),
inputting your departure and arrival id, and I offer you all the route you can choose;
4 for minimum flight time,
inputting your departure and arrival id, I count for the minimum time you need to spend;
5 for specific time intervals,
inputting two IDs and departure time intervaland I offer you all the route you can choose;
6 for minimum cost, inputting your departure and arrival id,I count for the minimum fares you need to cost;
-----

NOW INPUT YOUR CHOICE OF FUNCTION :
number:1

The DFS departure id : 3
```


C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

The dfs routes if the airport are:

The DFS departure id:

[illegible][illegible]

```
3->50->6
3->50->19
3->50
```

PROGRAM OVER!

2. 功能点 2

输入：3

输出：BFS 遍历顺序

```
NOW INPUT YOUR CHOICE OF FUNCTION :
number:2

The BFS departure id : 3

The bfs routes if the airport are:

The BFS departure id:
3->50->5->6->7->8->9->12->15->16->19->20->21->22->24->25->27->30->31->32->33->34->35->36->37->38->41->42->46->48->
51->54->57->60->62->67->68->72->78->49->52->61->63->66->75->76->77->26->44->47->70->71->4->43->45->73->28->29->79->
69->74->58->59->1->2->10->11->13->14->17->18->23->39->40->53->55->56->64->65->PROGRAM OVER!
```

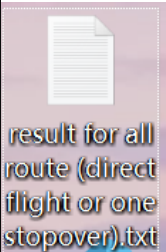
3. 功能点 3

输入：48 49

输出：48 到 49 的所有路径（直飞或一次转机）写入文件（作为附件上传）

```
NOW INPUT YOUR CHOICE OF FUNCTION :
number:3

DEPARTURE ID:
48
ARRIVAL ID:
49
```



```
result for all route (direct flight or one stopover).txt - 记事本

文件 编辑 查看

ID:1826->ID:2198
ID:1826->ID:2199
ID:1826->ID:2200
ID:1843->ID:2200
ID:1851->ID:2199
ID:1851->ID:2200
ID:1697->ID:915
ID:1697->ID:916
ID:1697->ID:917
ID:1697->ID:965
ID:1697->ID:982
ID:1697->ID:983
ID:1697->ID:1048
ID:1697->ID:1049
ID:1697->ID:2319
ID:1697->ID:2320
ID:1697->ID:2321
ID:1698->ID:916
ID:1698->ID:917
ID:1698->ID:965
ID:1698->ID:983
ID:1698->ID:1049
ID:1698->ID:2320
ID:1698->ID:2321
ID:1699->ID:917
ID:1699->ID:965
ID:1699->ID:1049
ID:1699->ID:2321
ID:13
ID:14
ID:1208
ID:1209
ID:1217
ID:1218
ID:1219
ID:1220
```

4. 功能点 4

输入：49 50

输出：航线 ID、出发时间、飞行时间、到达时间、累计时间

```
NOW INPUT YOUR CHOICE OF FUNCTION :  
number:4  
  
DEPARTURE ID:  
49  
ARRIVAL ID:  
50  
Flight ID :244->429  
FROM AIRPORT(DEPARTURE) 49 AT 201705081500  
MINIMUM FLYING TIME(IN AEROPLANE): 190 mins  
TRANSIT WAITING TIME: 10 mins  
ARRIVE AIRPORT 50 AT 201705081820  
TOTAL FLIGHT TIME: 200 mins  
PROGRAM OVER!
```

注：已验算该结果成绩，总共 200min，与案例相同；调试过程中还发现如下答案，总共 200 分钟，换乘 5 分钟，飞行 195 分钟。

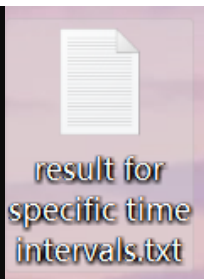
```
Flight ID :1151->1106  
FROM AIRPORT(DEPARTURE) 49 AT 201705051550  
MINIMUM FLYING TIME(IN AEROPLANE): 195 mins  
TRANSIT WAITING TIME: 5 mins  
ARRIVE AIRPORT 50 AT 201705051910  
TOTAL FLIGHT TIME: 200 mins  
PROGRAM OVER!
```

5. 功能点 5

输入：49 50 201705070000 201705091000

输出：该时间范围内起飞的所有 49 到 50 的所有路径（直飞或中转一次），并写入文档（文档附在作业文件夹中）。

```
NOW INPUT YOUR CHOICE OF FUNCTION :  
number:5  
  
DEPARTURE ID:  
49  
ARRIVAL ID:  
50  
Departure time no earlier than: 201705070000  
Departure time no later than: 201705091000
```



```
result for specific time intervals.txt - 记事本

文件  编辑  查看

ID:137->ID:134
ID:137->ID:142
ID:137->ID:1250
ID:138->ID:52
ID:138->ID:142
ID:138->ID:1250
ID:1651->ID:52
ID:1651->ID:142
ID:1651->ID:1250
ID:1652->ID:1250
ID:2269->ID:51
ID:2269->ID:52
ID:2269->ID:134
ID:2269->ID:142
ID:2269->ID:1250
ID:67->ID:103
ID:67->ID:1768
ID:68->ID:1768
ID:1229->ID:103
ID:1229->ID:1768
ID:1229->ID:1832
ID:1230->ID:1768
ID:1276->ID:102
ID:1276->ID:103
ID:1276->ID:1768
ID:1276->ID:1832
ID:1277->ID:103
ID:1277->ID:1768
ID:60->ID:33
ID:60->ID:41
ID:60->ID:58
ID:61->ID:41
ID:156->ID:41
ID:156->ID:58
ID:1194->ID:33
ID:1194->ID:41
ID:1194->ID:54
ID:1194->ID:58
ID:1195->ID:41
ID:1195->ID:58
ID:992->ID:142

行 451, 列 15 100%
```

6. 功能点 6

输入：50 25 201705051200 201705051400

输出：该时间范围内起飞的所有 50 到 25 所有路径中的最小花费

```
NOW INPUT YOUR CHOICE OF FUNCTION :
number:6

DEPARTURE ID:
50
ARRIVAL ID:
25

Departure time no earlier than: 201705051200
Departure time no later than: 201705051400

MINIMUM FARES : 1252
Flight ID :515
PROGRAM OVER!
```