**实验报告**

题目：航空公司航线图

姓名：刘凯祥 学号：2019202134 完成日期：2020.12.23

1. **需求分析**

见word文档要求

1. **概要设计**
2. **ADT结构**

ADT Graph {

数据对象V：V是具有相同特性的数据元素的集合，称为顶点集。

数据关系R：

R = {VR}

VR = {<v, w> | v, w∈V且P(v, w), <v, w>表示从v到w的弧，

谓词P(v, w)定义了弧<v, w>的意义或信息}

基本操作P：

CreateGraph(&G);

初始条件：传入空图G

操作结果：构造图G，用邻接表存储

FirstAdjVex(G, v);

初始条件：图G存在，v是G中某个顶点的下标

操作结果：返回v的第一个邻接顶点位置，若顶点在G中没有邻接点，则返回-1

NextAdjVex(G, v, w);

初始条件：图G存在，v是G中某个顶点的下标，w是v的邻接顶点

的下标

操作结果：返回v的（相对于w）的下一个邻接顶点位置，若w是v的最后一个邻接

点，则返回-1

LocateVex(G, vertex);

初始条件：图G存在，vertex为顶点信息

操作结果：若G中存在顶点vertex，则返回其在图中的位置，否则返回-1

}

1. **全局变量**

#define INFINITY 2147483647 //int型的最大值

#define MAX\_VERTEX\_NUM 100 //最大顶点数量，实际上为最大机场数量

#define MAX\_AIRLINE\_NUM 2500//最大现有航线数量

extern AirLine airline[MAX\_AIRLINE\_NUM];//存储航线信息的结构体

extern int LineNum; //航线数量，经统计共有2346条航线，即弧数

1. **数据结构**

**3.1 datadefine.h文件**

typedef struct Date {

int year; //年

int month; //月

int day; //日

int hour; //小时

int minute; //分钟

} Date; //日期

typedef struct AirLine

{

int FlightID;//航班ID

int FlightNO; //航班号

int DepAirport; //起飞机场

int ArrAirport; //抵达机场

int DepTime; //起飞时间,以分钟为单位

int ArrTime; //抵达时间,以分钟为单位

int AirplaneID;//飞机号

int AirplaneModel; //机型

int AirFare; //机票价格

}AirLine; //航线信息

* 1. **algraph.h文件**

//邻接表的定义

typedef struct ArcNode

{

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

}ArcNode;

typedef struct VNode

{

int data; //顶点信息，即机场

ArcNode \*firstarc; //指向第一条依附该顶点的弧

bool IsVisited; //是否被访问

}VNode, AdjList[MAX\_VERTEX\_NUM];

typedef struct

{

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

}ALGraph;//邻接表

* 1. **mgraph.h文件**

//邻接矩阵的定义

typedef struct ArcCell{

AirLine LineInfo; //存储航线信息的结构体：航班ID，起飞日期，国际/国内航班，航班号，起飞时间，抵达时间，飞机号，机型，机票价格等

ArcCell\* next; //指向下一个结点的指针

}ArcCell, \*AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct VexNode

{

int data;//顶点信息

bool IsVisited;//是否被访问过

}VexNode;

typedef struct{

VexNode vexs[MAX\_VERTEX\_NUM]; //顶点向量，存储机场

AdjMatrix arcs; //邻接矩阵,但每一个点上存储的是一个指针，用以指向一个单链表，此链表存储的为以该机场为起飞机场的航线。

int vexnum, arcnum; //图的当前顶点数和弧数

}MGraph;//邻接矩阵

* 1. **function.h文件**

typedef struct Path

{

int TraceLen; //路径长度，指路径上包含几个机场

int TraceID[80];//最多79个机场,存储航班ID

int TraceAirport[80]; //存储路径中的机场信息

int CurTime; //当前时间

int StartTime; //起始时间

int SumTime; //此路径下的总时间

int SumFare; //此路径下的总费用

}Path; //路径结点中的存储结构

typedef struct Req

{

int FirstDepTime;//存储用户希望的最早起飞时间

int LastDepTime;//存储用户希望的最晚起飞时间

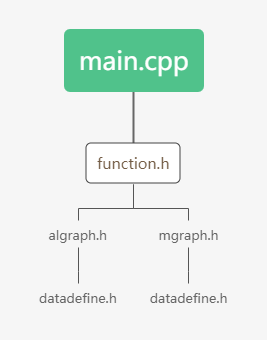
int FirstArrTime;//存储用户希望的最早抵达时间

int LastArrTime;//存储用户希望的最晚抵达时间

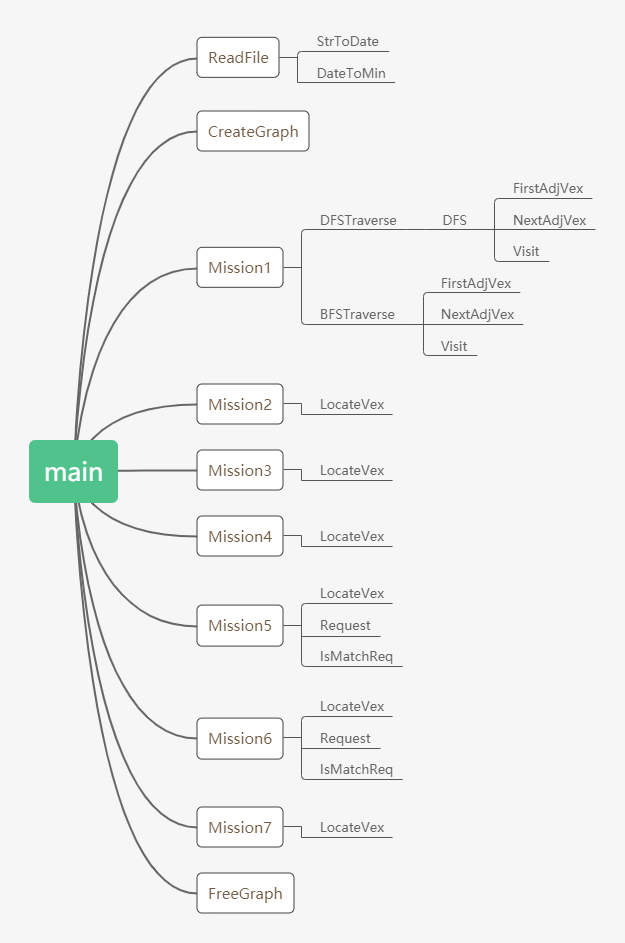
int AirplaneModel[10];//机型,存储用户想要的所有机型

}Req; //用户需求

1. **各模块之间的调用关系**



1. **函数调用关系图**



**三、 详细设计（略）**

**四、 调试分析**

1. 在DFS时传参忘记加&，导致实际上并没有修改IsVisited值，导致死循环；
2. 对于结构体赋值而言，一开始在AirLine结构体中添加了next指针成员，然后直接通过p = airline[i]进行全部赋值，并没有为指针另开辟空间，其结果导致内存泄漏，程序出错。结论：对于结构体中有指针的情况，赋值时不能简单的通过“=”赋值，而应当为其中的指针另开辟空间；
3. **算法的时空复杂度分析**

**CreateGraph:**

邻接表：

时间复杂度：

由于后续功能都是通过邻接矩阵实现，故在创建邻接表时并没有考虑两机场之间不同航线的问题，仅是连接可以直达的机场，故时间复杂度为O(LineNum)（LineNum为Excel表中的航线数量）

空间复杂度：

辅助变量仅是常数级别，故空间复杂度为O(1)

邻接矩阵：

时间复杂度：

邻接矩阵通过遍历Excel表中的所有航线信息进行创建，对于每一个单链表的创建均是通过头插法实现，故时间复杂度为O(LineNum) (LineNum为Excel表中的航线数量)

空间复杂度：

在邻接矩阵的每一个点上需要通过malloc创建新结点从而创建单链表，故空间复杂度为O(LineNum) (LineNum为Excel表中的航线数量)

**DFSTraverse:**

时间复杂度：

遍历n个顶点需要O(n)，而对于每一个结点i又需要扫描第i行的每一个元素，需要O(n)。故总的时间复杂度为O(n^2) （n为顶点数量）

空间复杂度：

在DFS时需要创建一个栈用以存储顶点信息，故空间复杂度为O(n)（n为顶点数量）

**BFSTraverse:**

时间复杂度：

遍历n个顶点需要O(n)，而对于每一个结点i又需要扫描第i行的每一个元素，需要O(n)。故总的时间复杂度为O(n^2) （n为顶点数量）

空间复杂度：

在BFS时需要创建一个队列用以存储顶点信息，故空间复杂度为O(n)（n为顶点数量）

**Mission1（打印输出从任意机场出发的遍历,包括深度优先遍历和广度优先遍历）**

时间复杂度：

内部实际上为对BFSTraverse和DFSTraverse的调用，故时间复杂度为O(n^2) （n为顶

点数量）

空间复杂度：

由函数内部结构可知空间复杂度为O(n) （n为顶点数量）

**Mission2（使用邻接矩阵表来完成任意两个机场的可连通性，包括是否可以直飞、1次中转、2次中转等）**

时间复杂度：

算法通过BFS的思想进行遍历，遍历到目的机场则停止，但由于创建新结点时是将与某一个相关的所有可行航线全部压入队列，所以是一种“发散”的算法，操作次数为每一层操作次数之和，最外层只需操作一次，依据求和后的最大指数项可知时间复杂度为O(n^(k-1))（n为每一个机场平均的可行航线，k为遍历的层数，即转机次数+1）

空间复杂度：

本算法需要创建可行的结点入栈，而最大存储的结点数量为最外层的可能航线数，故空间复杂度为O(n^k)（n为每一个机场平均的可行航线，k为遍历的层数，即转机次数+1）

**Mission3、Mission4、Mission5、Mission6、Mission7算法思想基本相同，均是采用DFS及剪枝优化操作，仅是剪枝的条件互不相同。故在此统一讨论算法的时间复杂度和空间复杂度。（当然，如果高级搜索功能加了很多限制条件，则会剪去很多不必要的分支，由此大大提高算法的效率，以下讨论的是一般情况）**

时间复杂度：

算法总体为DFS加优化剪枝，在遍历到每一个结点时，需要遍历该结点当前机场所在的一行，故此算法的时间复杂度与机场周围的航线数量分布密切相关，存在最优与最差的情况，总体平均的时间复杂度为O(m^n)(m为每个机场平均相连的机场数，n为机场数)为指数级别。

空间复杂度：

每次遍历都需要创造新结点存入栈，但最多仅为顶点数乘以航线数，故空间复杂度为O(n\*m)（n为机场数，m为航线数）

1. **优化方向**
   1. 一些Mission函数过于长，代码不容易维护，可以尝试将一个大函数拆分成几个小函数处理；
   2. 求最短时间理论上需要在遍历完所有情况后才能确定，虽然在算法中加了很多剪枝操作，但算法时间复杂度还是指数级别，需要改进。

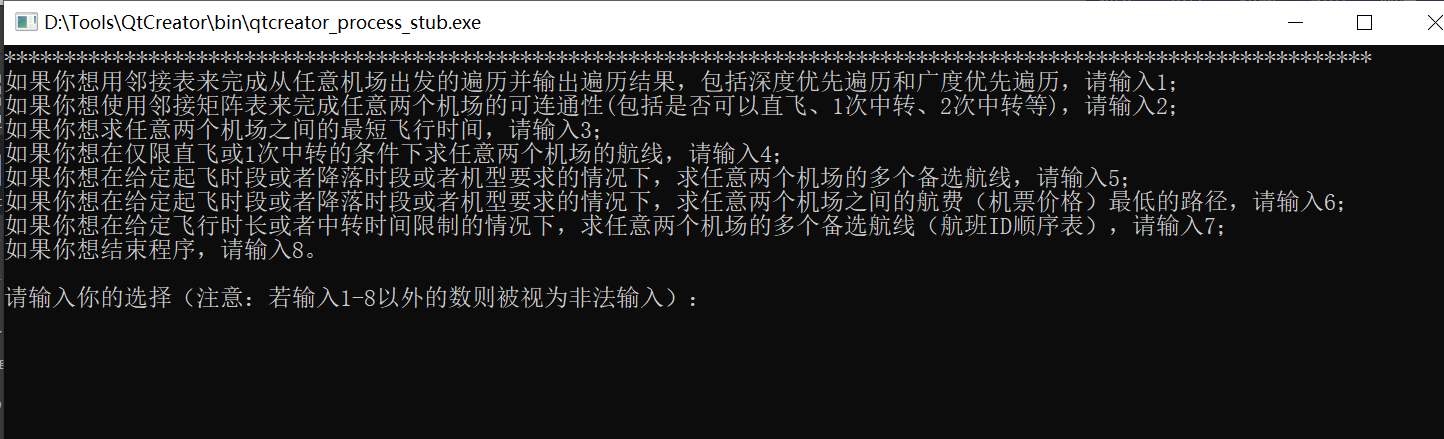
**五、 用户手册**

**1. 运行环境**

此程序需要在Qt中运行，使用的编码为GB2312, 如果使用其他编码，可能在对话窗口出现中文乱码；

**2. 操作步骤**

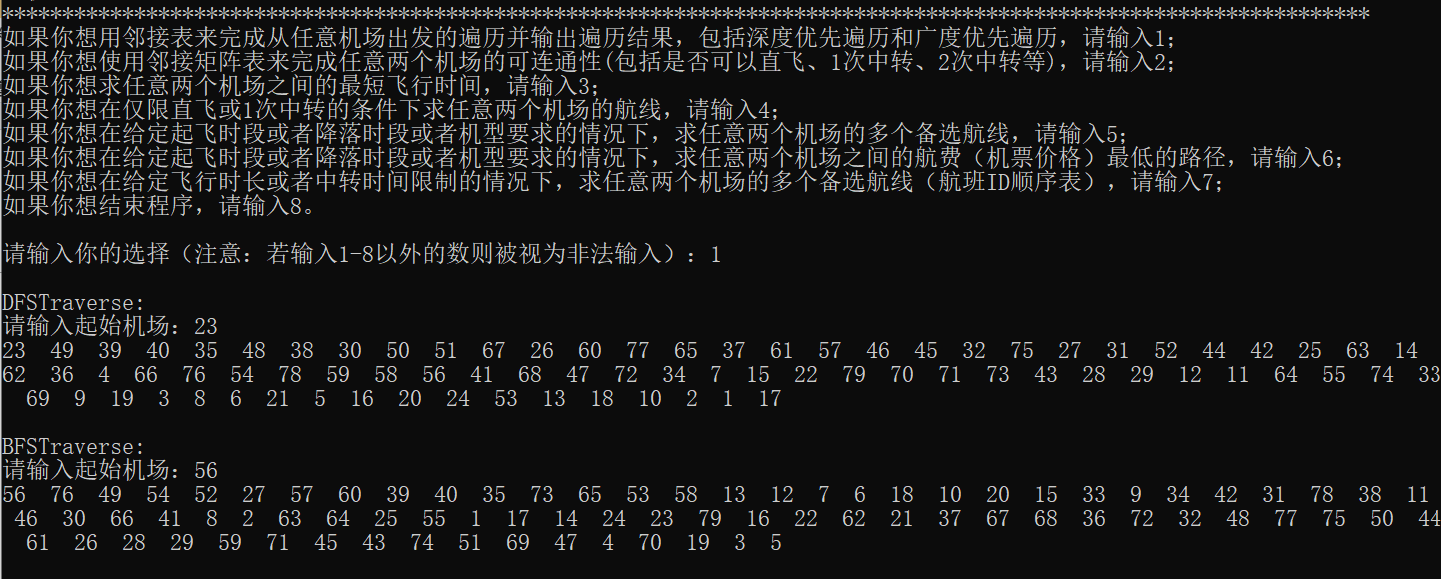
打开文件夹中的pro文件，点击运行按钮，出现如下对话框



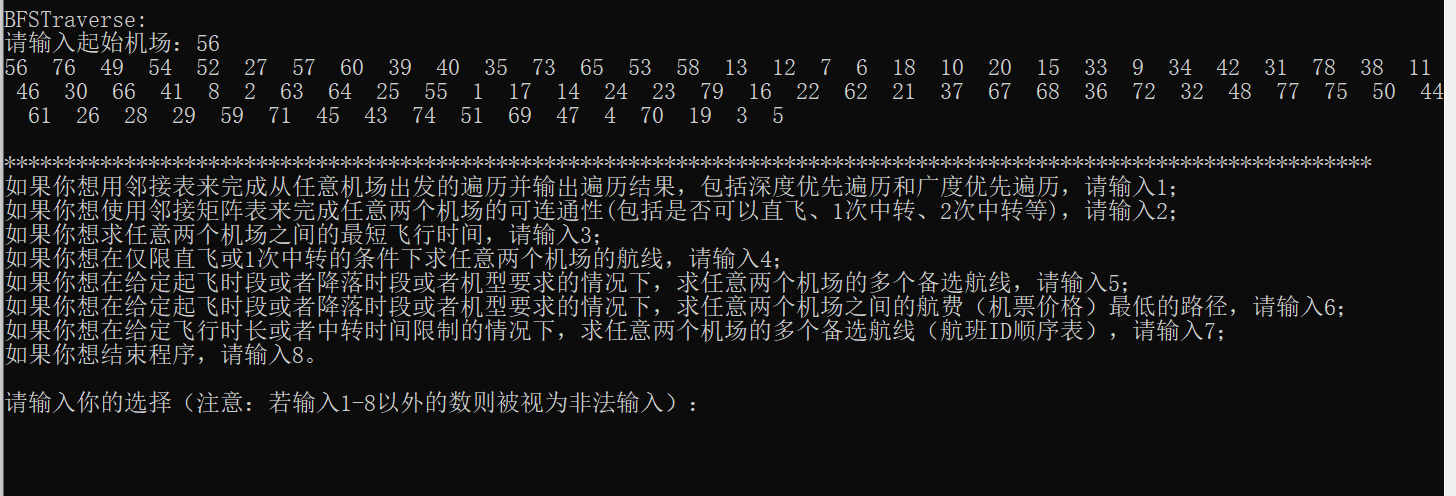
按照文字提示，用户希望执行哪项功能，则相应的直接输入对应0-8的数字，键入回车键表示输入结束

**功能1的实现：**

在输入1后，则会打印输出“DFSTraverse:”表示利用深度优先搜索遍历，之后对话框打印输出“请输入起始机场：”，此时用户直接输入一个数字，表示想要的起始机场，输入完后键入回车键则会打印相应的结果，打印完DFS结果后，程序打印输出“BFSTraverse：”，“请输入起始机场：”。输入要求同DFS输入。之后打印输出结果（如下图）：

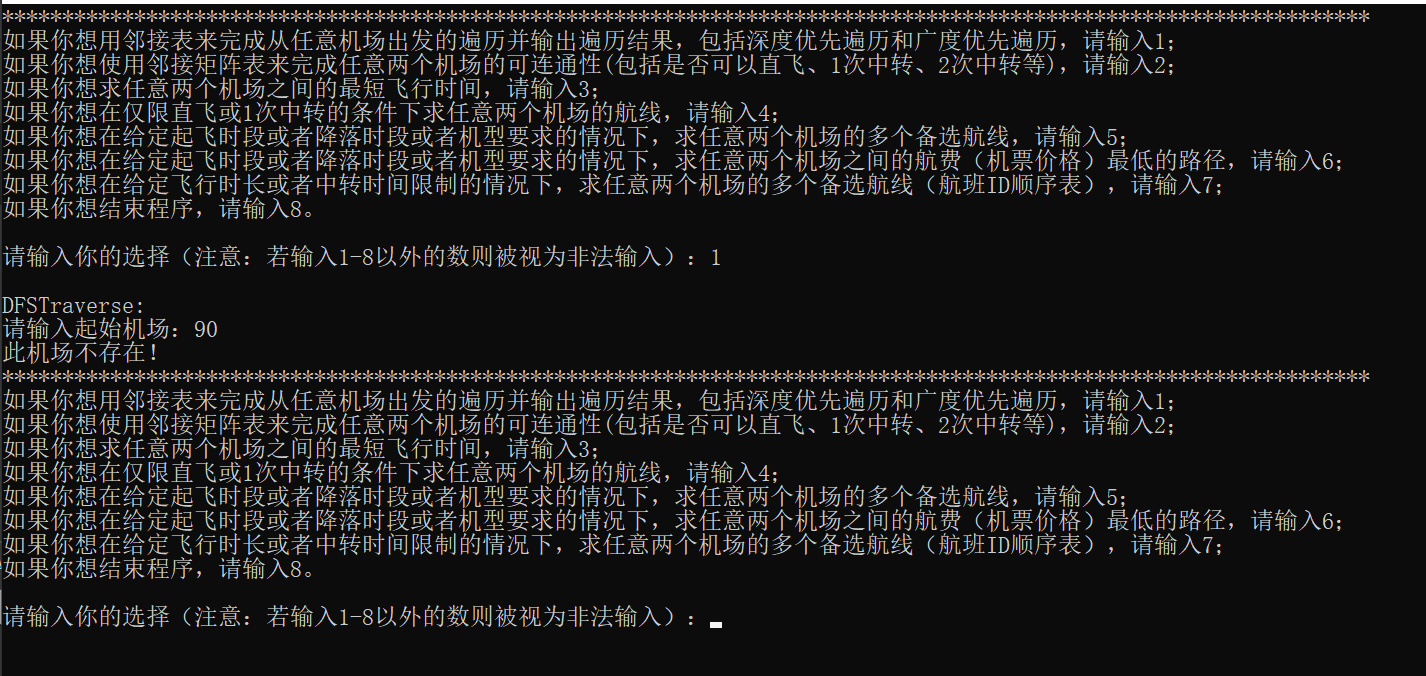


在打印输出BFS结果后，打印输出下一次的功能选择（如下图），可以键入0-8的数字表示想要的功能（如下图）：



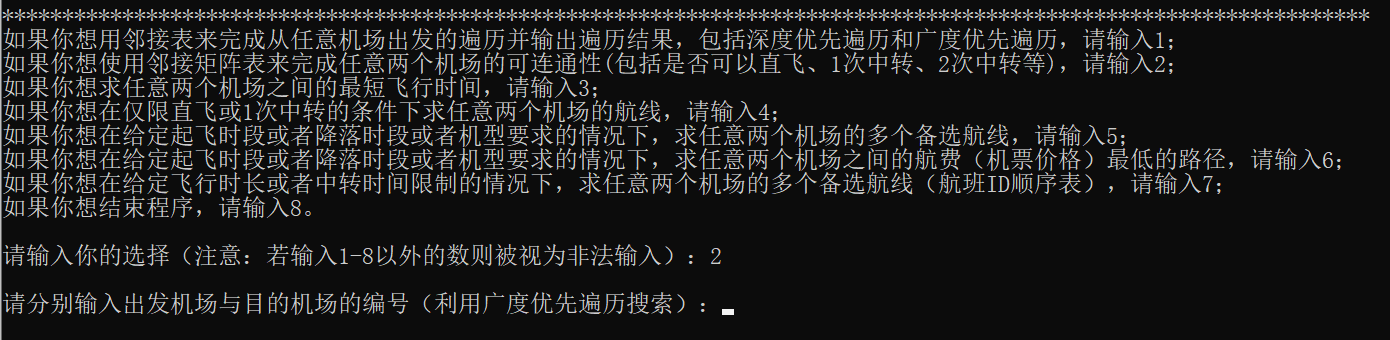
由于此功能每次只能在DFS和BFS方面各输入一个机场，所以如果有多个想要遍历的初始机场时，需要再次键入“1”，选择功能1，按照指示输入。

若用户输入了一个不存在的机场，则系统会打印输出出错信息，并进入下一次的功能选择，如下图（输入90时的情况）

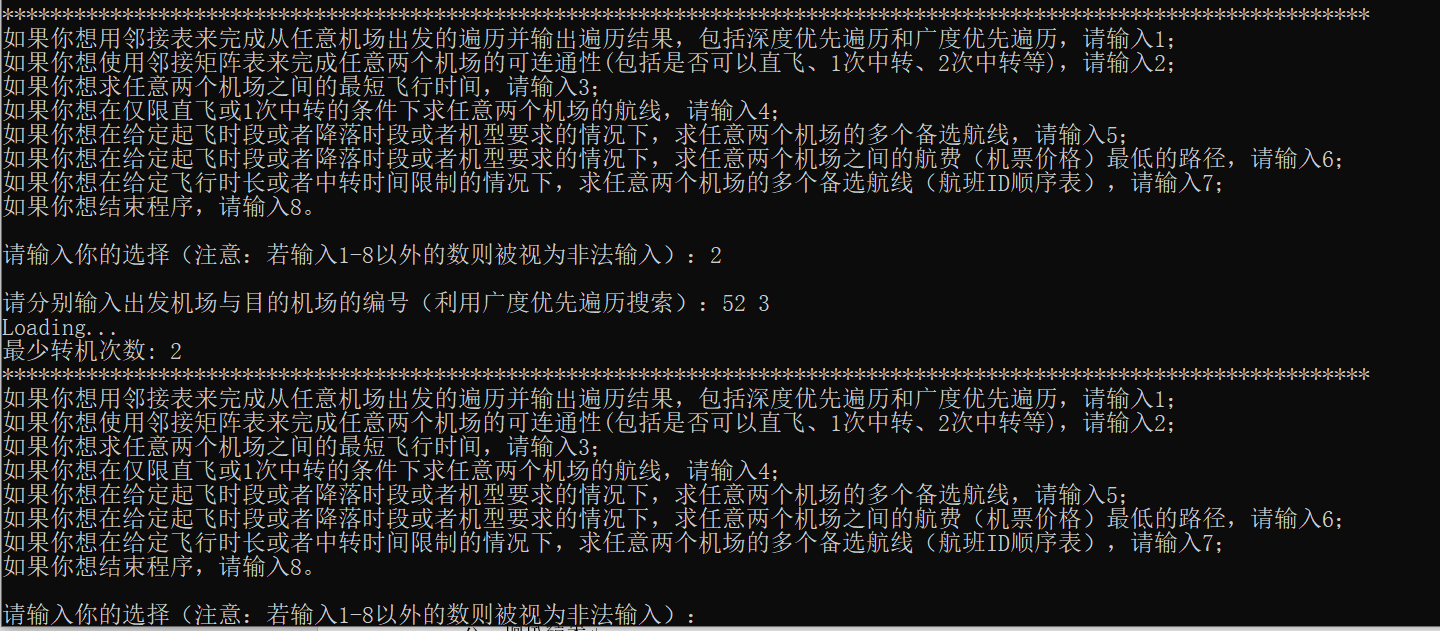


**功能2的实现：**

键入“2”后，出现下图的输出，此时用户可以输入两个数字分别表示出发机场和目的机场的编号，两个数字之间通过空格分隔。

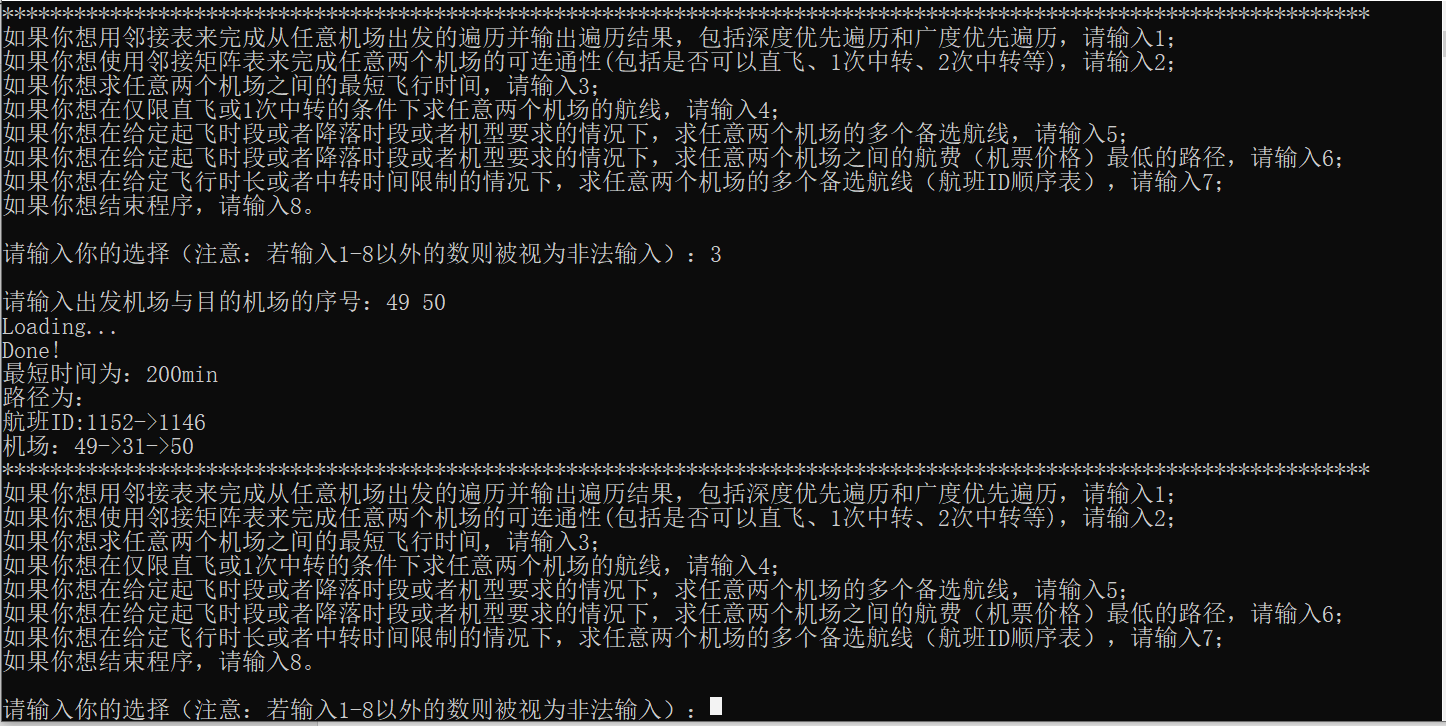


后打印输出“Loading…”、结果及下一次的功能选择（如下图）



**功能3的实现：**

用户键入“3”后按照功能2的输入形式输入两个机场的编号即可，输入完成后则会打印输出结果（如下图）



**功能4的实现：**

用户键入“4”后按照功能2的输入形式输入两个机场的编号即可，输入完成后则会打印输出结果（如下图）



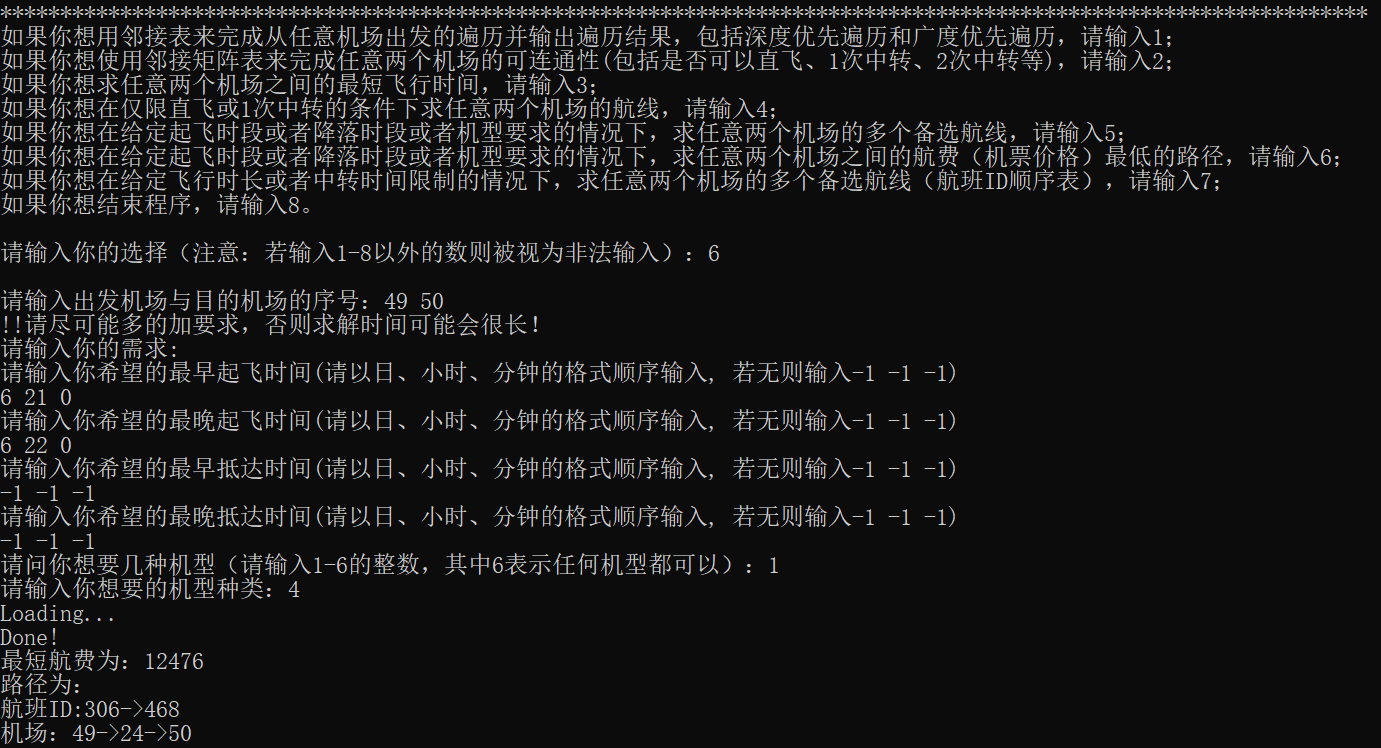
**功能5的实现：**

在功能选择时键入“5”，之后按照指示输入两个机场的编号。在输入完机场编号后对话框会打印出相应的指示，指示用户输入需求，在输入时间限制时，每次输入三个数字，分别表示日、小时、分钟，三个数字之间用空格分隔。以下图输入为例，此处输入的最早起飞时间为6日21点0分（默认为2017年5月）。若用户对起飞时间等没有要求，则输入三个“-1”。在机型要求方面，根据界面指示，需要先输入用户一共想要的机型种类数，其中若键入1-5的数，则相应的在后续的指示下继续输入1-5种机型。若输入6，则表示任何机型都接受，用户无需再输入机型。（具体输入和结果见下图）



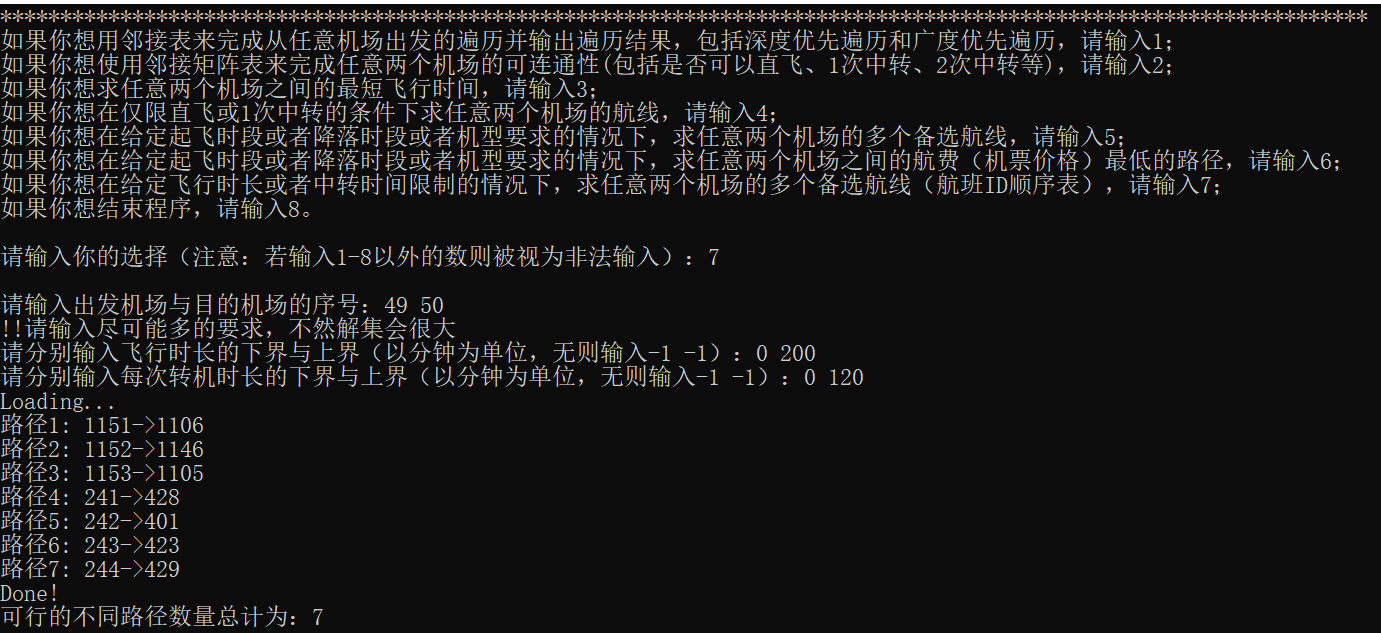
**功能6的实现：**

在功能选择时键入“6”，后续输入操作要求完全同功能5，具体输入要去和输出结果见下图：



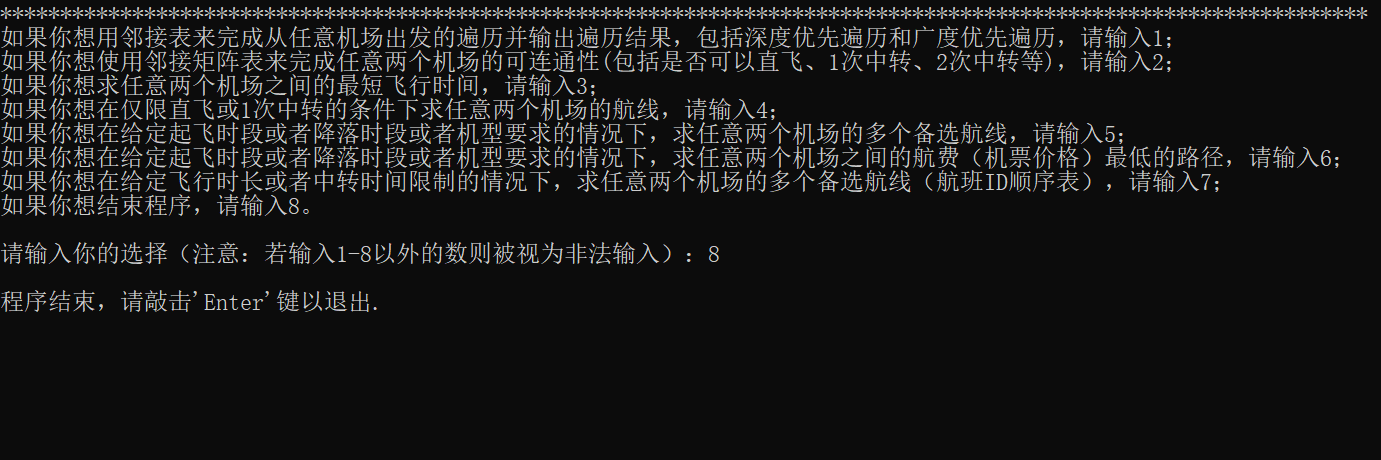
**功能7的实现：**

用户在选择功能时键入“7”，之后根据提示输入起始机场和目的机场的编号，两个整数之间用空格分隔。在对话框界面中根据提示先输入两个整数（用空格分隔），分别表示飞行时长的下界与上界（单位为分钟），之后再根据提示输入两个整数（用空格分隔），分别表示每次转机时长的下界与上界（单位为分钟）。之后打印输出所有可能的结果（航班ID顺序表）以及总的路径数量（见下图）



**结束程序**

在功能选择时键入“8”，之后打印输出“程序结束，请敲击'Enter'键以退出.”，之后敲击'Enter'键则对话框关闭，程序结束。



**3. 注意事项**

3.1用户需要严格按照要求格式输入需求及各方面的要求，不可以修改输入形式；

3.2 对于没有数字要求范围时，默认可以输入任意整形数字；

3.3 在求所有备选路线时，用户需要尽量多的加上一些飞行时间以及机型方面的限制吗，不然可

能输出解集会很大。在不加任何约束条件的情况下，有些机场之间可能多则30多万条可行

路线，而在此种情况下仅输出前2000条航线，并每统计出10万条航线时则打印输出一次

“Loading”

3.4 由于算法效率还存在很大改进的空间，故可能有些机场之间的最短时间需要很长时间才能求出，比如52->3。

**六、 调试结果**

**功能1（用邻接表来完成从任意机场出发的遍历，包括深度优先遍历和广度优先遍历）**

执行命令‘1’：

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

输入‘56’，输出“56 76 49 54 52 27 57 60 39 40 35 73 65 53 58 13 12 7 6 18 10 20 15 33 9 34 42 31 78 38 11 46 30 66 41 8 2 63 64 25 55 1 17 14 24 23 79 16 22 62 21 37 67 68 36 72 32 48 77 75 50 44 61 26 28 29 59 71 45 43 74 51 69 47 4 70 19 3 5” 及下次功能选择文字

执行命令‘1’：

输入‘90’，输出“此机场不存在！”

**功能2（使用邻接矩阵表来完成任意两个机场的可连通性，包括是否可以直飞、1次中转、2次中转等）**

执行命令‘2’：

输入“52 3”（注：其中包含一个空格），输出“Loading...”，“最少转机次数: 2”；

执行命令‘2’：

输入“1 57” （注：其中包含一个空格），输出“Loading...”，“最少转机次数: 1”；

执行命令‘2’：

输入“3 39” （注：其中包含一个空格），输出“Loading...”，“最少转机次数: 3”；

**功能3（求任意两个机场之间的最短飞行时间（含转机停留时间））**

执行命令‘3’：

输入“49 50”（注：其中包含一个空格），输出“Loading...”，“Done!”，“最短时间为：200min”，“路径为：”，“航班ID: 1151->1106”，“机场：49->31->50”；

执行命令‘3’：

输入“4 40”（注：其中包含一个空格），输出“Loading...”，“Done!”，“最短时间为：2405min”，“路径为：”，“航班ID: 69->631->1884->1001->2235”，“机场：4->36->25->42->49->40”；

**功能4（仅限直飞或1次中转，求任意两个机场的航线（航班ID顺序表））**

执行命令‘4’：

输入“48 49”（注：其中包含一个空格），输出“可以直飞！”， “航班ID为：13或1220或1219或1218或1217或1209或1208或14”；

执行命令‘4’：

输入“20 58”（注：其中包含一个空格），输出“不能直飞抵达！”， “可以经过一次转机抵达！

航班ID为：1304->1559

航班ID为：1304->1548

航班ID为：1353->1559

航班ID为：1352->1559

航班ID为：1352->1558

航班ID为：1352->1548”；

执行命令‘4’：

输入“52 3”（注：其中包含一个空格），输出“不能直飞抵达！仅限直飞或1次中转时，无法抵达！”

执行命令‘4’：

输入“4 100”（注：其中包含一个空格），输出“错误输入！无此机场编号”

执行命令‘4’：

输入“35 35”（注：其中包含一个空格），输出“错误输入！起始位置与终止位置相同”

**功能5（给定起飞时段或者降落时段或者机型要求，求任意两个机场的多个备选航线（航班ID顺序表））**

执行命令‘5’：

输入“49 50”（注：其中包含一个空格），输入“6 21 0”，输入“6 22 0”，输入“-1 -1 -1”，输入“-1 -1 -1”输入‘1’，输入‘4’。输出“路径1: 306->468”“Done! 可行的不同路径数量总计为：1”

执行命令‘5’：

输入“48 50”（注：其中包含一个空格），输入“5 10 20”，输入“5 15 20”，输入“5 10 20”，输入“5 23 20”输入‘6’。输出“路径1: 1”“Done! 可行的不同路径数量总计为：1”

**功能6（给定起飞时段或者降落时段或者机型要求，求任意两个机场之间的航费（机票价格）最低的路径）**

执行命令‘6’：

输入“49 50”（注：其中包含一个空格），输入“6 21 0”，输入“6 22 0”，输入“-1 -1 -1”，输入“-1 -1 -1”输入‘1’，输入‘4’。输出“Loading...”，“Done! 最短航费为：12476

路径为：

航班ID:306->468

机场：49->24->50

执行命令‘6’：

输入“48 49”（注：其中包含一个空格），输入“-1 -1 -1”，输入“-1 -1 -1”，输入“-1 -1 -1”，输入“-1 -1 -1”输入‘6’。输出“Loading...”，“Done! 最短航费为：594

路径为：

航班ID:13

机场：48->49

**功能7（给定飞行时长或者中转时间限制，求任意两个机场的多个备选航线（航班ID顺序表））**

执行命令‘7’：

输入“49 50”（注：其中包含一个空格），输入“0 200”，输入“0 120”。输出“Loading…路径1: 1151->1106

路径2: 1152->1146

路径3: 1153->1105

路径4: 241->428

路径5: 242->401

路径6: 243->423

路径7: 244->429

Done!

可行的不同路径数量总计为：7

执行命令‘7’：

输入“3 37”（注：其中包含一个空格），输入“0 2000”，输入“0 1000”。输出“Loading…路径1: 1454->2029

路径2: 1454->1428->1413->767

路径3: 1454->1512->1857->767

路径4: 1454->995->990->767

路径5: 1454->910->793->767

路径6: 1454->227->1793->2156

路径7: 1454->227->209->767

路径8: 1454->227->248->767

路径9: 1454->82->2321->767

Done!

可行的不同路径数量总计为：9

执行命令‘7’：

输入“3 37”（注：其中包含一个空格），输入“0 4000”，输入“0 200”。输出“Loading…

Done! 无法抵达！

**七、 附录**

源程序文件名清单

Datadefine.h, Datadefine.cpp //基本数据和数据结构的定义和实现

Algraph.h, Algraph.cpp //邻接表的定义及基本操作的申明和实现

Mgraph.h, Mgraph.cpp //邻接矩阵的定义及基本操作的申明和实现

Function.h, Function.cpp //所有功能函数的申明和实现

Main.cpp //主程序