实验 07 实验报告

题目：设计一个某航空公司航线图，用户可以提出不同的航班时段、时长、航费额度或者机型要求，程序返回航班线路（航班 ID 顺序表）。

姓名：余宏昊　学号：2015200975　完成日期：2018.12.25

目录

一、 需求分析

模块 I 读取 csv 文件，并进行合适的处理。

1. 定义一个结构体用于存放航班的各项信息（航班 ID、起飞时间、降落时间、航费等），并用结构体的指针数组存放所有的航班，从而构建起航班安排表。
2. 演示程序自行读取指定路径的 csv 文件，并按照设定的格式将航班安排表打印输出。

模块 II 读取航班安排表（数据框图），构建以邻接表为存储结构的图，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。

1. 将机场按序号升序存放在顺序表中。
2. 将每个机场能直飞抵达的航班以链表形式接在对应的机场结点后面。
3. 用户输入遍历起始机场序号，程序遍历各机场时将其序号输出。
4. 利用栈作为辅助实现深度优先遍历。
5. 利用链式队列作为辅助实现广度优先遍历。
6. 程序执行的命令包括：

1）遍历数据框图，逐行读取航班信息；2）为起飞机场构建表头结点，并为其抵达机场与该趟航班信息构建弧结点，插入到以表头结点为头结点的链表中，如果表头结点已存在，则只插入弧结点；3）完成遍历，构建起完整的邻接表；4）读入遍历起始机场序号；5）完成深度优先遍历，顺序输出各机场序号；6）完成广度优先遍历，顺序输出各机场序号；7）结束。

用户需要从所给的机场序号范围中选取一个序号，以此作为起始点。

7. 测试数据

DFSTALGraph(ALGraph, 2)

BFSTALGraph(ALGraph, 2)

BFSTALGraph(ALGraph, 4)

模块 III 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；并求任意两个机场之间的最短飞行时间。

1. 用邻接矩阵作为图的存储结构，并存储任意两个机场之间的连通信息，矩阵大小为 VexNum*VexNum。
2. 矩阵中每个单元格存放两部分信息，一部分为连通信息，用整型表示，-1 表示不连通，1 表示可以直飞，2 表示需要转一次机，3 表示需要转两次机；另一部分存放连通两个机场的航班组合。
3. 用户输入两个机场的序号，程序返回连通信息。

4. 用户输入两个机场的序号，程序返回两个机场之间的最短飞行时间。
5. 程序执行的命令包括：1）读取邻接表，构建邻接矩阵表；2）接受用户输入，确定两个机场的序号；3）返回两个机场之间的连通信息；4）返回两个机场之间的最短飞行时间；5）结束。
6. 测试数据

CheckRoute(mgraph, 3, 49)

ShortestDuration(mgraph, 3, 49)

CheckRoute(mgraph, 6, 49)

ShortestDuration(mgraph, 6, 49)

CheckRoute(mgraph, 50, 30)

ShortestDuration(mgraph, 50, 30)

CheckRoute(mgraph, 2, 16)

ShortestDuration(mgraph, 2, 16)

模块 IV 仅限直飞或一次中转，求任意两个机场的航线。

1. 接受用户输入，确定起飞和降落机场序号；
2. 用单链表存储连接两个机场的所有航线；
3. 接受用户输入，确定转机次数；
4. 根据转机次数输出符合条件的所有航线；
5. 程序执行的命令包括：

1）接受用户输入（机场序号）；2）在航班安排表中进行查找，将符合条件的所有航线以结点形式构建起一个单链表；3）接受用户输入（转机次数）；4）打印输出所有航线；5）结束。

6. 测试用例

```
1.    LimitTransRouteV1(schedule, mgraph, 3, 49, 1);
2.    LimitTransRouteV1(schedule, mgraph, 6, 49, 0);
3.    LimitTransRouteV1(schedule, mgraph, 6, 49, 1);
4.    LimitTransRouteV1(schedule, mgraph, 50, 30, 0);
5.    LimitTransRouteV1(schedule, mgraph, 2, 16, 1);
```

模块 V 给定起飞时段或降落时段或机型要求，或者给定飞行时长或转机时间限制，求任意两个机场的多个备选航线，或者求航费最低的航线。

1. 接受用户输入，获取各种要求；
2. 从连接两个机场的所有航线中过滤出符合条件的所有航线，并将其打印输出；

3. 程序执行的命令包括：

1）接受用户输入，确定起飞和抵达机场；2）用单链表 A 存储所有的航线；3）接受用户输入的各种限定条件；4)；解析限定条件；5）用单链表 B 存储符合条件的所有航线；6）打印输出符合条件的航线；7）结束。

4. 测试用例

a. 3 号机场到 49 号机场之间，05/07 00:00～05/09/23:59 之间出发，任意时段降落，第一段航程机型为 2，任意飞行时长，任意转机时长的所有航线。
b. 3 号机场到 49 号机场之间，05/07 00:00～05/09/23:59 之间出发，任意时段降落，第一段航程任意机型，飞行时长不超过 40 小时，转机时长不超过 10 小时的所有航线。
c. 2 号机场到 16 号机场之间，任意时段出发出发，05/07 00:00～05/08/12:00 之间降落，第一段航程任意机型，飞行时长不超过 20 小时，转机时长不超过 10 小时的所有航线。
d. 2 号机场到 16 号机场之间，任意时段出发出发，任意时段降落，第一段航程任意机型，任意飞行时长，任意转机时长的花费最少的航线。

二、 概要设计

模块 I 读取 csv 文件，并进行合适的处理。

1. 航班安排表的抽象数据类型定义为：

ADT Schedule {

数据对象：$D = \{a_i | a_i \in FlightRecords, i = 1,2,\dots,n, n \geq 0\}$

数据关系：$R_1 = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D, i = 1,2,\dots,n\}$

基本操作：

InitSchedule(&Schedule)

操作结果：构造一个空的航班安排表。

ReadCSV(Csv, &Schedule)

操作结果：读取 csv 文件，构建航班安排表

GetRow(&Schedule)

初始条件：航班安排表已存在。

操作结果：获取航班安排表行数。

GetCol(&Schedule)

初始条件：航班安排表已存在。

操作结果：获取航班安排表列数。

PrintSchedule(Schedule)

初始条件：航班安排表已存在。

操作结果：打印输出航班安排表。

} **ADT** Schedule

2. 本程序包含四个模块：

1) 主程序模块：

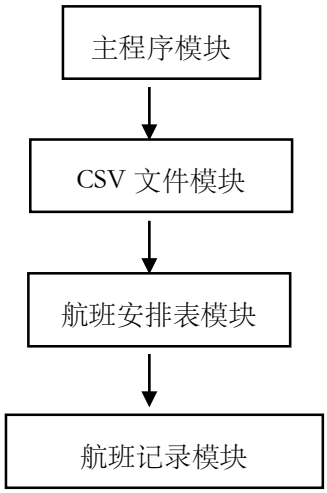
```
void main() {
    初始化;
    读取 CSV 文件;
    构建航空安排表;
    打印输出航空安排表;
    退出程序;
}
```

2) 航班记录单元模块——定义航班记录结构；

3) CSV 文件处理单元模块——实现顺序表的抽象数据类型；

4) 航班安排表单元模块——定义航班安排表结构。

各模块之间的调用关系如下：



模块 II 读取航班安排表（数据框图），构建以邻接表为存储结构的图，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。

需要一个抽象数据类型：图。

完成深度优先遍历需要一个抽象数据类型：栈。

完成广度优先遍历需要一个抽象数据类型：队列。

1. 图的抽象数据类型定义为：

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 = {$< v, w >$ | $v, w \in V$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, V, VR)

初始条件：V 是图的顶点集，VR 是图中弧的集合。

操作结果：按 V 和 VR 的定义构造图 G。

DestoryGraph(&G)

初始条件：图 G 已存在。

操作结果：销毁图 G。

LocateVex(G, u)

初始条件：图 G 存在，u 和 G 中顶点有相同特征。

操作结果：若 G 中存在顶点 u，则返回该顶点在图中位置；否则返回其他信息。

GetVex(G, v)

初始条件：图 G 存在，v 是 G 中某个顶点。

操作结果：返回 v 的值。

PutVex(&G, v, value)

初始条件：图 G 存在，v 是 G 中某个顶点。

操作结果：对 v 赋值 value。

FirstAdjVex(G, v)

初始条件：图 G 存在，v 是 G 中某个顶点。

操作结果：返回 v 的第一个邻接顶点。若顶点在 G 中没有邻接顶点，则返回“空”。

NextAdjVex(G, v, w)

初始条件：图 G 存在，v 是 G 中某个顶点，w 是 v 的邻接顶点。

操作结果：返回 v 的（相对于 w 的）下一个邻接顶点。若 w 是 v 的最后一个邻接点，则返回“空”。

InsertVex(&G, v)

初始条件：图 G 存在，v 和图中顶点有相同特征。

操作结果：在图 G 中增添新顶点 v。

DeleteVex(&G, v)

初始条件：图 G 存在，v 是 G 中某个顶点。

操作结果：删除 G 中顶点 v 及其相关的弧。

InsertArc(&G, v, w)

初始条件：图 G 存在，v 和 w 是 G 中两个顶点。

操作结果：在图 G 中增添弧<v, w>，若 G 是无向的，则还增添对称弧<w,v>。

DeleteArc(&G, v, w)

初始条件：图 G 存在，v 和 w 是 G 中两个顶点。

操作结果：在 G 中删除弧<v,w>，若 G 是无向的，则还删除对称弧<w,v>。

DFSTraverse(G, Visit())

初始条件：图 G 存在，Visit 是顶点的应用函数。

操作结果：对图进行深度优先遍历。在遍历过程中对每个顶点调用函数 Visit 一次且仅一次。一旦 Visit()失败，则操作失败。

BFSTraverse(G, Visit())

初始条件：图 G 存在，Visit 是顶点的应用函数。

操作结果：对图进行广度优先遍历。在遍历过程中对每个顶点调用函数 Visit 一次且仅一次。一旦 Visit()失败，则操作失败。

} **ADT** Graph

2. 栈和队列的抽象数据类型在之前的大作业中已经出现过，在此不赘述

3. 本程序包含五个模块：

1) 主程序模块：

```
void main() {
    构建以邻接表为存储结构的图；
    接受用户输入，确定遍历的起始结点；
    从起始结点开始进行深度优先遍历，打印所访问的各结点。
    从起始结点开始进行广度优先遍历，打印所访问的各结点。
    退出程序；
}
```

2) 邻接表单元模块——实现邻接表的各项操作；

3) 深度优先遍历单元模块——实现深度优先遍历；

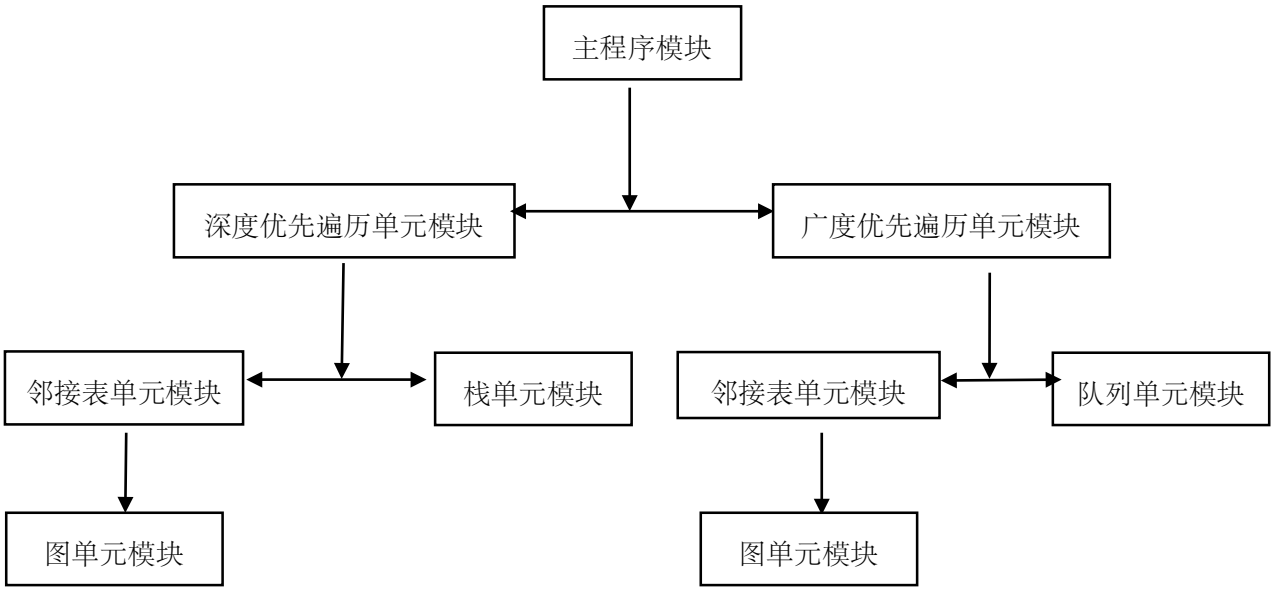
4) 栈单元模块——实现栈的抽象数据类型；

5) 广度优先遍历单元模块——实现广度优先遍历；

6) 队列单元模块——实现队列的抽象数据类型；

7) 图单元模块——实现图的抽象数据类型。

各模块之间的调用关系如下：



模块 III 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；并求任意两个机场之间的最短飞行时间。

1. 图的抽象数据类型与模块 II 一致。
2. 本程序包含五个模块：

1) 主程序模块：

```
void main() {
    根据邻接表构建邻接矩阵；
    获取用户输入，确定起飞和降落机场；
    输出两个机场之间的连通信息；
    输出两个机场之间的最短飞行时间；
    退出程序；
}
```

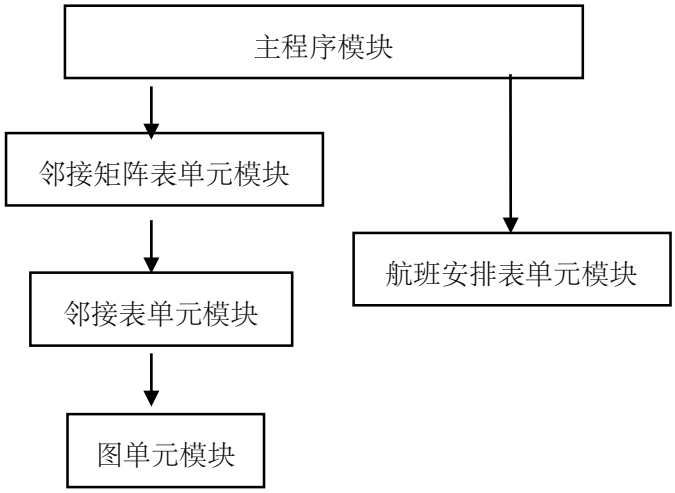
2) 邻接表单元模块——实现邻接表的各种操作；

3) 邻接矩阵表单元模块——实现邻接矩阵表的各种操作；

4) 图单元模块——实现图的抽象数据类型；

5) 航班安排表单元模块——获取航班信息；

各模块之间的调用关系如下：



模块 IV 仅限直飞或一次中转，求任意两个机场的航线。

获取连接两个机场的所有航线，需要一个抽象数据类型：单链表

1. 单链表的抽象数据类型定义在之前的大作业中已经完成，在此不再赘述。
2. 本程序包含六个模块：

1) 主程序模块：

```
void main() {
    键入出发和到达机场序号；
```

键入最大转机次数；

打印输出所有符合条件的航线；

退出程序；

}

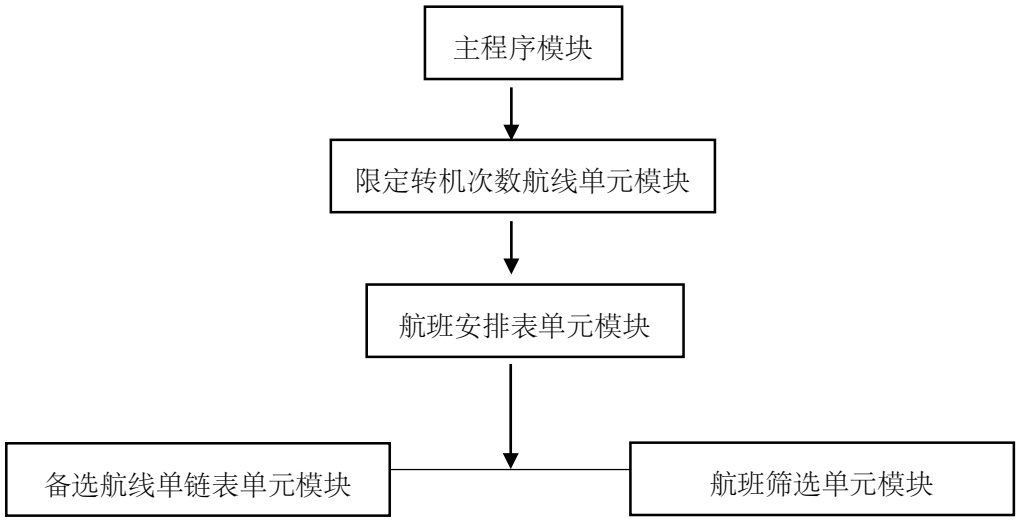
2) 限定转机次数航线单元模块

3) 备选航线单链表单元模块——用单链表存储所有符合条件的航线；

4) 航班安排表单元模块——获取航班信息；

5) 航班筛选单元模块——根据筛选条件筛选航班；

各模块之间的调用关系如下：



模块 V 给定起飞时段或降落时段或机型要求，或者给定飞行时长或转机时间限制，求任意两个机场的多个备选航线，或者求航费最低的航线。

1. 基本思路与模块 IV 一致。
2. 增加一个筛选条件解析和判断模块，用于筛选符合条件的航线。
3. 本程序包含四个模块：

1) 主程序模块：

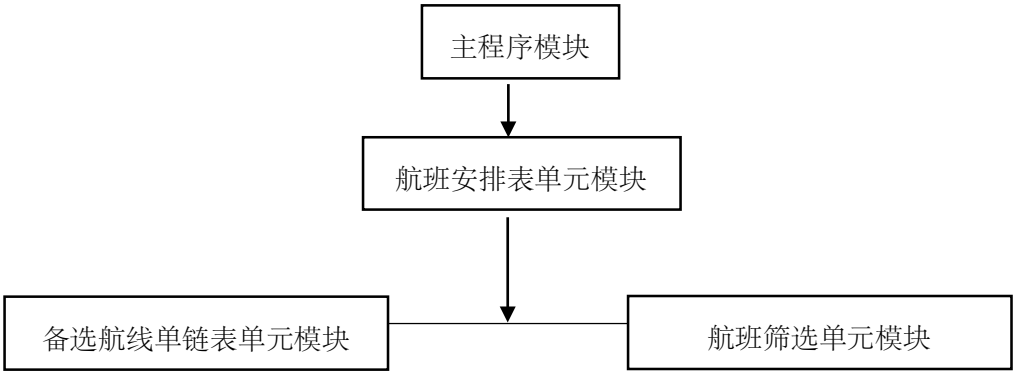
```
void main() {
    键入出发和到达机场序号；
    键入各种筛选条件（包括起飞时段、降落时段、机型要求、飞行时长、转机时长、是否最低价等）；
    打印输出所有符合条件的航线；
    退出程序；
}
```

2) 备选航线单链表单元模块——用单链表存储所有符合条件的航线;

3) 航班安排表单元模块——获取航班信息;

4) 航班筛选单元模块——根据筛选条件筛选航班;

各模块之间的调用关系如下:



三、 详细设计

模块 I 读取 csv 文件，并进行合适的处理。

1. 航班记录结构

```
typedef struct {
    int           flight_id;
    char     depart_date[15];
    char      flight_type[5];    //Intl/Dome
    int           flight_no;
    int           departure;    //Departure Airport
    int             arrival;    //Arrival Airport
    char  departure_time[20];
    char    arrival_time[20];
    int      departure_time1;    //Departure time in numerical form
    int        arrival_time1;    //Arrival time in numerical form
    int          airplane_id;
    int       airplane_model;
    int            airfares;
} flight_record;
```

2. 航班安排表结构

```
typedef struct {
    flight_record     * records;
    int                row,col;
} Schedule;
```

3. 航班安排表的基本操作设置如下：

```
Status InitRecord(flight_record * record);
    //初始化航线记录
Status InitSchedule(Schedule * schedule);
    //初始化航班安排表
int get_row(char *filename);
    //获取航班安排表行数
int get_col(char *filename);
    //获取航班安排表列数
Status read_csv(char * filename, Schedule * schedule);
```

```
    //将 csv 文件读入到航班安排表中
Status PrintSchedule(Schedule schedule);
    //打印航班安排表
Status PrintFlight(flight_record flight);
    //打印输出航班
Status GetFlightID(Schedule schedule, Airport Departure, Airport Arrival,
int * ID);
    //获取两个机场之间的所有航班 ID，存放到一个整型数组中
flight_record * GetRecord(Schedule schedule, int i);
    //根据航班 ID 获取航班记录
```

其中部分操作的伪码算法如下：

```
Status InitRecord(&record)
{
    record->flight_id=-1;record->flight_no=-1;
    record->departure=-1;record->arrival=-1;
    record->departure_time1=-1;record->arrival_time1=-1;
    record->airplane_id=-1;record->airplane_model=-1;
    record->airfares=-1;
    memset(record->depart_date,0,sizeof(record->depart_date));
    memset(record->flight_type,0,sizeof(record->flight_type));
    memset(record->departure_time,0,sizeof(record->departure_time));
    memset(record->arrival_time,0,sizeof(record->arrival_time));
    return OK;
} //InitRecord

Status InitSchedule(&schedule)
{
    schedule->records=(flight_record
*)malloc(MAX_FLIGHT_NUM*sizeof(flight_record));
    for(i=0;i<MAX_FLIGHT_NUM;++i)
        InitRecord(&schedule->records[i]);
    schedule->col=schedule->row=0;
    return OK;
```

```
} //InitSchedule


Status read_csv(&filename, &schedule)
{
    InitSchedule(schedule);
    stream = fopen(filename, "r");
    if(stream!=NULL)
    {
        line=fgets(buffer, sizeof(buffer), stream);
        while ((line = fgets(buffer, sizeof(buffer), stream))!=NULL)//当没有读
取到文件末尾时循环继续
        {
            record = strtok(line, delims);  //X~XXXX
            schedule->records[j].flight_id=atoi(record);
            record = strtok(NULL, delims);  //mm/dd/yyyy
            strcpy(schedule->records[j].depart_date, record);
            record = strtok(NULL, delims);  //Intl/Dome
            strcpy(schedule->records[j].flight_type, record);
            record = strtok(NULL, delims);  //XX~XXX
            schedule->records[j].flight_no=atoi(record);
            record = strtok(NULL, delims);  //X~XX
            schedule->records[j].departure=atoi(record);
            record = strtok(NULL, delims);  //X~XX
            schedule->records[j].arrival=atoi(record);
            record = strtok(NULL, delims);  //mm/dd/yyyy XX:XX
            strcpy(schedule->records[j].departure_time, record);
            record = strtok(NULL, delims);  //mm/dd/yyyy XX:XX
            strcpy(schedule->records[j].arrival_time, record);
            record = strtok(NULL, delims);  //X~XXX
            schedule->records[j].airplane_id=atoi(record);
            record = strtok(NULL, delims);  //X
            schedule->records[j].airplane_model=atoi(record);
            record = strtok(NULL, delims);  //XXX~XXXX
```

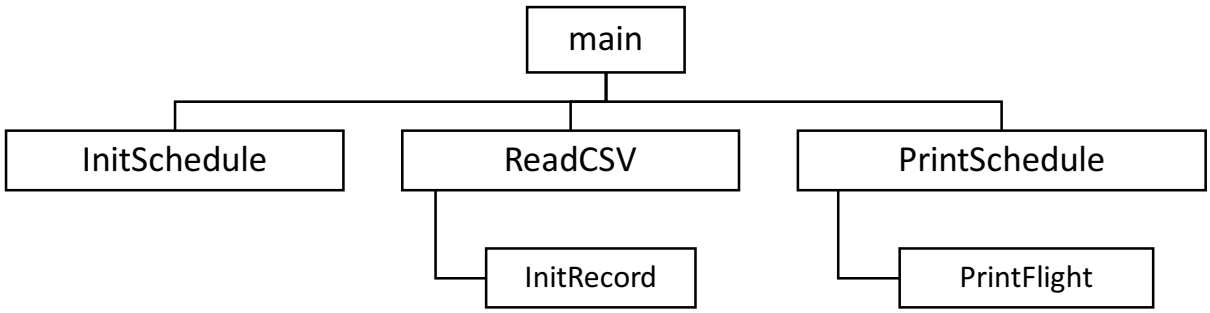
```
            schedule->records[j].airfares=atoi(record);

            strcpy(ts1, schedule->records[j].departure_time);
            ts1b = strtok(ts1,delims2);
            ts1b = strtok(NULL,delims2);
            days1=atoi(ts1b)-1;
            ts1b = strtok(NULL,delims2);
            ts1b = strtok(NULL,delims2);
            hours1=atoi(ts1b);
            ts1b = strtok(NULL,delims2);
            minutes1=atoi(ts1b);
            time1=days1*24*60+hours1*60+minutes1;
            schedule->records[j].departure_time1=time1;

            strcpy(ts2, schedule->records[j].arrival_time);
            ts2b = strtok(ts2,delims2);
            ts2b = strtok(NULL,delims2);
            days2=atoi(ts2b)-1;
            ts2b = strtok(NULL,delims2);
            ts2b = strtok(NULL,delims2);
            hours2=atoi(ts2b);
            ts2b = strtok(NULL,delims2);
            minutes2=atoi(ts2b);
            time2=days2*24*60+hours2*60+minutes2;
            schedule->records[j].arrival_time1=time2;

            ++j;
        } //while
        schedule->row=get_row(filename);
        schedule->col=get_col(filename);
    } //if
    return OK;
}
```

```
Status GetFlightID(schedule, Departure, Arrival, &ID)
{
    for(i=0;i<schedule.row-1;++i)
    {
        if(Departure==schedule.records[i].departure&&Arrival==schedule.record
s[i].arrival)
            ID[++num]=schedule.records[i].flight_id;
    }
    ID[0]=num;
    return OK;
} //GetFlightID
```

4. 主函数的伪码算法：

```
void main()
{  //主函数
    InitSchedule(&schedule);
    read_csv(file, &schedule);
    PrintSchedule(schedule);
} //main
```

5. 函数的调用关系图反映了演示程序的层次结构：



模块 II 读取航班安排表（数据框图），构建以邻接表为存储结构的图，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。

1. 图种类、结点类型、弧结点类型、表头结点类型、邻接表类型

```
typedef enum {DG, DN, UDG, UDN} GraphKind; //{有向图，有向网，无向图，无向网}
typedef int VertexType;
typedef struct ArcNode {
    VertexType        arrival;    //到达机场序号
```

```
    struct ArcNode    *nextarc;    //指向下一条弧的指针
    flight_record *flight_info;    //航班信息
} ArcNode;
typedef struct VNode {
    VertexType     departure;    //出发机场序号
    ArcNode        *firstarc;    //指向第一条依附该顶点的弧的指针
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList         vertices;
    int        vexnum,arcnum;    //图的当前顶点数和弧数
    GraphKind           kind;    //图的种类标志
} ALGraph;
```

2. ALGraph 的基本操作设置如下：

```
ALGraph * CreateALGFromDF(Schedule schedule);
    //根据 DataFrame 创建以邻接表为存储结构的图
Airport FirstAdjPort(ALGraph algraph, Airport airport1);
    //在邻接表中，找出所给机场的第一个直飞可达机场
Airport NextAdjPort(ALGraph algraph, Airport airport1, Airport airport2);
    //在邻接表中，找出所给机场在 airport1 之后的下一个直飞可达机场
Status DFSTALGraph(ALGraph algraph, int departure);
    //深度优先遍历
Status BFSTALGraph(ALGraph algraph, int departure);
    //广度优先遍历
```

其中部分操作的伪码算法如下：

```
ALGraph * CreateALGFromDF(Schedule schedule)
{
    algraph=(ALGraph *)malloc(sizeof(ALGraph));
    InitALGraph(algraph);
    num_flights=schedule.row-1;
    for(i=0;i<num_flights;++i)
    {
        departure=schedule.records[i].departure;
        if(taken[departure]==0)
```

```
            algraph->vexnum++;
        newarc=CreateArcNode(&schedule.records[i]);
        AddArcNode(&algraph->vertices[departure], newarc);
        taken[departure]=1;
        algraph->arcnum++;
    }
    return algraph;
} //CreateALGFromDF

Airport FirstAdjPort(ALGraph algraph, Airport airport1)
{
    if(algraph.vertices[airport1].firstarc!=NULL)
        airport2=algraph.vertices[airport1].firstarc->arrival;
    return airport2;
} //FirstAdjPort

Airport NextAdjPort(ALGraph algraph, Airport airport1, Airport airport2)
{
    pn = algraph.vertices[airport1].firstarc;
    for(;pn!=NULL;pn=pn->nextarc)
    {
        if(pn->arrival==airport2)
            break;
    }
    for(;pn!=NULL;pn=pn->nextarc)
    {
        if(pn->arrival!=airport2)
        {
            airport3=pn->arrival;
            break;
        }
    }
    return airport3;
```

```
} //NextAdjPort

Status DFSTALGraph(ALGraph algraph, Airport departure)
{   //利用栈从任意结点出发实现深度优先遍历
    for(v=1;v<=algraph.vexnum;++v) visited[v]=0;  //访问标志数组初始化
    InitStack(&S);
    visited[departure]=1;
    Visit(departure);
    Push(&S, departure);
    while(StackEmpty(S)==0)
    {
        GetTop(&S, &port);
        for(w=FirstAdjPort(algraph, port);w>=0;w=NextAdjPort(algraph, port,
        w))
            if(visited[w]==0)
            {visited[w]=1;Push(&S, w);Visit(w);break;}
        if(w<0) Pop(&S, &port);
    }
    return OK;
} //DFTALGraph

Status BFSTALGraph(ALGraph algraph, int departure)
{   //利用队列从任意结点出发实现广度优先遍历
    for(v=1;v<=algraph.vexnum;++v) visited[v]=0;  //访问标志数组初始化
    InitLkQueue(&LQ);
    visited[departure]=1;
    Visit(departure);
    EnQueue(&LQ, departure);
    while(QueueEmpty(LQ)==0&&GetHead(&LQ, &port))
    {
            for(w=FirstAdjPort(algraph, port);w>=0;w=NextAdjPort(algraph,
        port, w))
            if(visited[w]==0)
```

```
            {visited[w]=1;EnQueue(&LQ, w); Visit(w);break;}
        if(w<0) DeQueue(&LQ, &port);
    }
    return OK;
} //BFTALGraph
```

3. 邻接表的基本操作设置如下：

```
Status InitArcNode(ArcNode * arcnode);
    //初始化弧结点
Status InitVNode(VNode * vnode);
    //初始化表头结点
Status InitALGraph(ALGraph * graph);
    //初始化邻接表
ArcNode * CreateArcNode(flight_record * flight);
    //构建弧结点
Status AddArcNode(VNode * vnode, ArcNode * arcnode);
    //加入弧结点
```

其中部分操作的伪码算法如下：

```
ArcNode * CreateArcNode(flight_record * flight)
{
    newarc=(ArcNode *)malloc(sizeof(ArcNode));
    newarc->nextarc=NULL;
    newarc->arrival=flight->arrival;
    newarc->flight_info=flight;
    return newarc;
} //CreateArcNode

Status AddArcNode(VNode * vnode, ArcNode * arcnode)
{
    if(vnode->firstarc==NULL)
    {arcnode->nextarc=NULL;vnode->firstarc=arcnode;}
    else
    {
        if(arcnode->arrival<=vnode->firstarc->arrival)
```

```
        {arcnode->nextarc=vnode->firstarc;vnode->firstarc=arcnode;}
        else
        {
            pn = vnode->firstarc;
            for(;pn->nextarc!=NULL;pn=pn->nextarc)
            {
                if(arcnode->arrival<=pn->nextarc->arrival)
                {arcnode->nextarc=pn->nextarc;pn->nextarc=arcnode;break;}
            }
            if(pn->nextarc==NULL)
            {arcnode->nextarc=NULL;pn->nextarc=arcnode;}
        }
    }
    return OK;
} //AddArcNode
```

4. 栈和队列的基本操作设置如下：

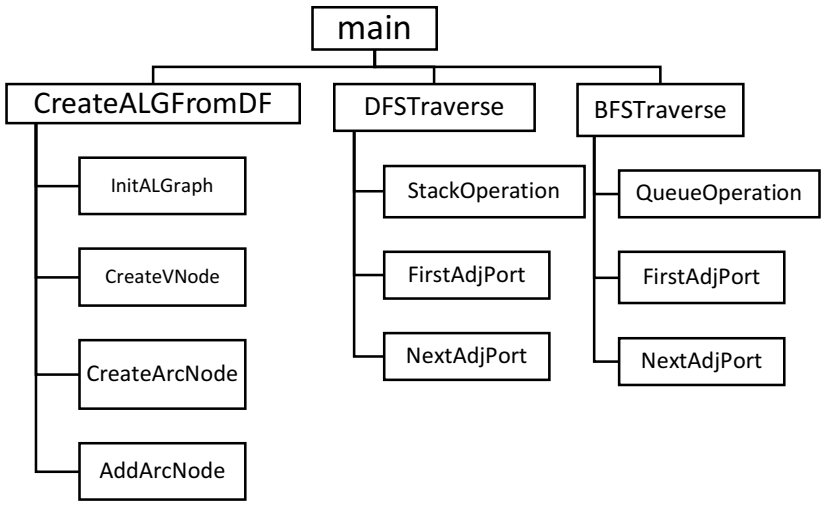
```
Status InitStack(Stack * S);
    //初始化栈
Status DestroyStack(Stack * S);
    //销毁栈
Status Push(Stack * S, Airport airport);
    //压栈
Status Pop(Stack * S, Airport * airport);
    //弹栈
Status GetTop(Stack * S, Airport * airport);
    //获取栈顶元素
int StackEmpty(Stack S);
    //判空
Status Traverse(Stack * S);
    //遍历

Status InitLkQueue(LkQueue * LQ);
    //初始化链式队列
```

```
Status DestroyLkQueue(LkQueue * LQ);
    //销毁链式队列
Status EnQueue(LkQueue * LQ, Airport airport);
    //入列
Status DeQueue(LkQueue * LQ, Airport * airport);
    //出列
Status GetHead(LkQueue * LQ, Airport * airport);
    //获取队列头元素
int QueueEmpty(LkQueue LQ);
    //判空
```

5. 主函数的伪码算法：

```
void main()
{  //主函数
    algraph=CreateALGFromDF(schedule);
    scanf(&StartingPoint);
    DFSTALGraph(algraph, StartingPoint);
    BFSTALGraph(algraph, StartingPoint);
} //main
```

6. 函数的调用关系图反映了演示程序的层次结构：



模块 III 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；并求任意两个机场之间的最短飞行时间。

需要用到一个抽象数据类型：图

1. 邻接矩阵表单元格结构、图结构

```
typedef int VRType;
```

```
typedef char InfoType;
typedef int VertexType;
typedef struct ArcCell {
    VRType  adj;    //-1 表示不连通，1 表示只能直飞，2 表示 1 次中转，3 表示 2 次中转
    int   transfers[MAX_VERTEX_NUM];    //中转组合
} ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
typedef struct {
    VertexType   vexs[MAX_VERTEX_NUM];   //顶点向量
    AdjMatrix    arcs;                   //邻接矩阵
    int          vexnum,arcnum;          //图的当前顶点数和弧数
    GraphKind    kind;                   //图的种类标志
} MGraph;
```

2. 邻接矩阵表的基本操作设置如下：

```
Status InitArcCell(ArcCell * arccell);
    //初始化弧
Status InitMGraph(MGraph * mgraph);
    //初始化邻接矩阵
MGraph * CreateMGFromALG(ALGraph algraph);
    //根据 ALG 创建以邻接矩阵为存储结构的图
Status PrintMGraph(MGraph * M);
    //打印邻接矩阵
Status CheckRoute(MGraph * M, Airport Departure, Airport Arrival);
    //判断两个机场之间的连通性
```

其中部分操作的伪码算法如下：

```
MGraph * CreateMGFromALG(ALGraph algraph)
{
    mgraph=(MGraph *)malloc(sizeof(MGraph));
    InitMGraph(mgraph);
    mgraph->vexnum=algraph.vexnum;
    for(i=1;i<=algraph.vexnum;i++)
        mgraph->vexs[i]=1;
    for(i=1;i<=algraph.vexnum;i++)   //直飞
```

```
    for(airport1=FirstAdjPort(algraph,
i);airport1>0;airport1=NextAdjPort(algraph, i, airport1))
        {mgraph->arcs[i][airport1].adj=1;
         mgraph->arcs[i][airport1].transfers[0]=0;mgraph->arcnum++;}

    for(i=1;i<=algraph.vexnum;i++)   //转机一次
    {
        for(airport1=FirstAdjPort(algraph,
i);airport1>0;airport1=NextAdjPort(algraph, i, airport1))
            for(airport2=FirstAdjPort(algraph,
airport1);airport2>0;airport2=NextAdjPort(algraph, airport1, airport2))
            {
                if(airport2!=i&&mgraph->arcs[i][airport2].adj==-1)
                {
                    mgraph->arcs[i][airport2].adj=2;
                    mgraph->arcs[i][airport2].transfers[num_lines[i][airport2]
++]=airport1;
                    mgraph->arcnum++;
                }
            }
    }

    for(i=1;i<=algraph.vexnum;i++)   //转机两次
    {
        for(airport1=FirstAdjPort(algraph,
i);airport1>0;airport1=NextAdjPort(algraph, i, airport1))
            for(airport2=FirstAdjPort(algraph,
airport1);airport2>0;airport2=NextAdjPort(algraph, airport1, airport2))
                for(airport3=FirstAdjPort(algraph,
airport2);airport3>0;airport3=NextAdjPort(algraph, airport2, airport3))
                {
                    if(airport3!=i&&mgraph->arcs[i][airport3].adj==-1)
                    {
```

```
                    mgraph->arcs[i][airport3].adj=3;
                    mgraph->arcs[i][airport3].transfers[num_lines[i][airpo
rt3]++]=airport1*100+airport2;
                    mgraph->arcnum++;
                }
            }
    }

    return mgraph;
} //CreateMGFromALG

Status CheckRoute(MGraph * M, Airport Departure, Airport Arrival)
{
    connected=M->arcs[Departure][Arrival].adj;
    if(connected==1)
        printf("Airport No.%d-->Airport No.%d: Direct
Flight\n",Departure,Arrival);
    else if(connected==2)
        printf("Airport No.%d-->Airport No.%d: One
Transfer\n",Departure,Arrival);
    else if(connected==3)
        printf("Airport No.%d-->Airport No.%d: Two
Transfers\n",Departure,Arrival);
    else if(connected==-1)
        printf("Airport No.%d-->Airport No.%d: No
Flight\n",Departure,Arrival);
    return OK;
} //CheckRoute
```

3. 获取最短飞行时间的基本操作设置如下：

```
int GetShortestDuration(Schedule schedule, MGraph * M, Airport Departure,
Airport Arrival);
    //获取两个机场之间的最短飞行时间
```

```
Status QuickestFlight(Schedule schedule, MGraph * M, Airport Departure,
Airport Arrival);
    //将最短飞行时间进行格式化输出
```

其中部分操作的伪码算法如下：

```
int GetShortestDuration(Schedule schedule, MGraph * M, Airport Departure,
Airport Arrival)
{
    connected=M->arcs[Departure][Arrival].adj;
    if(connected==1)  //直飞
    {
        GetFlightID(schedule, Departure, Arrival, ID1);
        flight_record * pn=GetRecord(schedule, ID);
        shortest_duration=pn->arrival_time1-pn->departure_time1;
    }
    else if(connected==2)  //转一次机
    {
        for(int i=0;M->arcs[Departure][Arrival].transfers[i]!=0;i++)    //转机
机场
        {
            Transfer=M->arcs[Departure][Arrival].transfers[i];
            GetFlightID(schedule, Departure, Transfer, ID1);
            GetFlightID(schedule, Transfer, Arrival, ID2);
            for(i=1;i<=ID1[0];i++)
            {
                flight1=GetRecord(schedule, ID1[i]);
                for(j=1;j<=ID2[0];j++)
                {
                    flight2=GetRecord(schedule, ID2[j]);
                    if(flight1->arrival_time1<flight2->departure_time1)
                    {
                        duration=flight2->arrival_time1-
flight1->departure_time1;
                    }
```

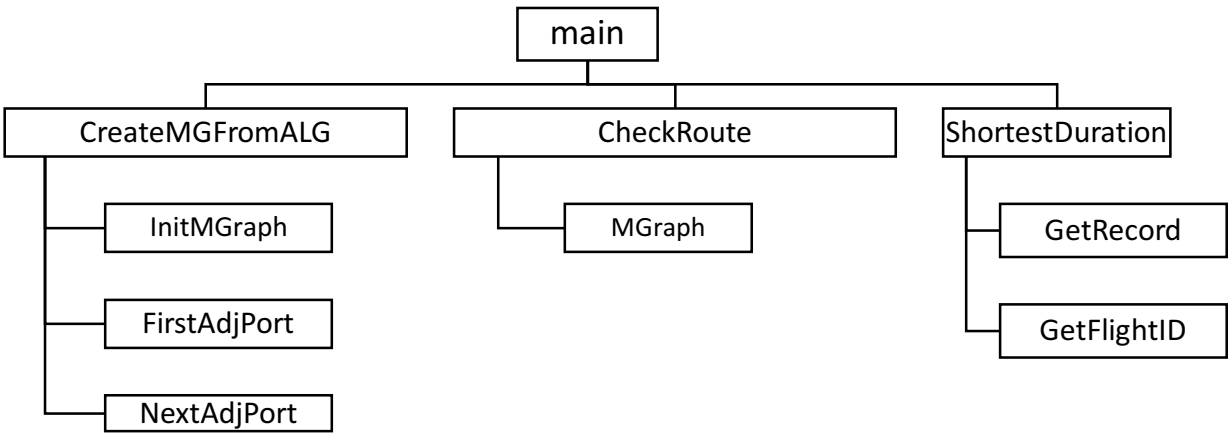
```
                    if(duration<shortest_duration)
                        shortest_duration=duration;
                }
            }
        }
    }
    else if(connected==3)  //转两次机
    {
        for(i=0;M->arcs[Departure][Arrival].transfers[i]!=0;i++)    //转机机场
        {
            Transfers=M->arcs[Departure][Arrival].transfers[i];
            Transfer1=(int)floor(Transfers/100);
            Transfer2=(int)Transfers%100;
            GetFlightID(schedule, Departure, Transfer1, ID1);
            GetFlightID(schedule, Transfer1, Transfer2, ID2);
            GetFlightID(schedule, Transfer2, Arrival,   ID3);
            for(i=1;i<=ID1[0];i++)
            {
                flightA=GetRecord(schedule, ID1[i]);
                for(j=1;j<=ID2[0];j++)
                {
                    flightB=GetRecord(schedule, ID2[j]);
                    for(k=1;k<=ID3[0];k++)
                    {
                        flightC=GetRecord(schedule, ID3[k]);
                        if(flightA->arrival_time1<flightB->departure_time1&&fl
ightB->arrival_time1<flightC->departure_time1)
                        {
                            duration=flightC->arrival_time1-
flightA->departure_time1;
                            if(duration<shortest_duration)
                                shortest_duration=duration;
                        }
```

```
                }
            }
        }
    }
    return shortest_duration;
} //GetShortestDuaration
```

4. 主函数的伪码算法：

```
void main()
{  //主函数
    mgraph=CreateMGFromALG(&algraph);
    scanf(&Departure, &Arrival);
    CheckRoute(mgraph, Departure, Arrival);
    ShortestDuration(Schedule, mgraph, Departure, Arrival)
} //main
```

5. 函数的调用关系图反映了演示程序的层次结构：



模块 IV 仅限直飞或一次中转，求任意两个机场的航线。

1. 航线类型：

```
typedef struct Itis {
    flight_record    * first;   //第一段航程，若没有则为 NULL
    flight_record   * second;   //第二段航程，若没有则为 NULL
    flight_record    * third;   //第三段航程，若没有则为 NULL
    int          num_flights;   //整个航线的航程数
    struct Itis       * next;   //下一个航线
```

```
} FullIti;
typedef struct {
    FullIti          * head;
    int      num_options[4];   //存放所有航线的数量信息，num_options[0]为总航线数，
其他下标代表航程数
} Options;
```

2. 航线单链表的基本操作设置如下：

```
Status InitFullIti(FullIti * fulliti);
    //初始化航线
Status InitOptions(Options * options);
    //初始化存放航线的链表
flight_record * CreateFlight(flight_record * record);
    //复制所给的航班信息，并创建一个新的航班记录
FullIti * CreateFullIti(flight_record * first, flight_record * second,
flight_record * third);
    //根据所给的航程构建航线信息
Status PushHeadOps(Options * options, flight_record * first, flight_record *
second, flight_record * third);
    //头插：插入航线信息
Status PushTailOps(Options * options, flight_record * first, flight_record *
second, flight_record * third);
    //尾插：插入航线信息
Status AddIti(Options * options, FullIti * iti);
    //将航线按第一段航程航班 ID 升序插入到链表中
```

3. 获取连接两个机场符合条件的所有航线的基本操作设置如下：

```
Status AllRoutesV1(Schedule schedule, MGraph * M, Airport Departure, Airport
Arrival, Options * Choices);
    //获取连同两个机场的所有航线
Status LimitTransRouteV1(Schedule schedule, MGraph * M, Airport Departure,
Airport Arrival, int Transfers);
    //限定转机次数，输出航线
```

一些伪码算法如下：

```
Status AllRoutesV1(Schedule schedule, MGraph * M, Airport Departure, Airport
Arrival, Options * Choices)
{
    if(M->arcs[Departure][Arrival].adj==1)    //可以直飞
    {
        GetFlightID(schedule, Departure, Arrival, A_ID1);
        for(i=1;i<=A_ID1[0];i++)           //直飞路径
        {
            flight1 = GetRecord(schedule, A_ID1[i]);
            flightI = CreateFlight(flight1);
            fulliti = CreateFullIti(flightI, flightII, flightIII);
            AddIti(Choices, fulliti);
            flightI=flightII=flightIII=NULL;
        }

        for(i=1;i<=M->vexnum;i++)        //转一次机的路径
            if(Departure!=i&&i!=Arrival&&Departure!=Arrival&&M->arcs[Departur
e][i].adj==1&&M->arcs[i][Arrival].adj==1)
            {
                GetFlightID(schedule, Departure, i, A_IDa);
                GetFlightID(schedule, i, Arrival, A_IDb);
                for(j=1;j<=A_IDa[0];j++)
                {
                    flight1=GetRecord(schedule, A_IDa[j]);
                    flightI = CreateFlight(flight1);
                    for(k=1;k<=A_IDb[0];k++)
                    {
                        flight2=GetRecord(schedule, A_IDb[k]);
                        flightII = CreateFlight(flight2);
                        if(flightI->arrival_time1<flightII->departure_time1)
                        {
                            fulliti = CreateFullIti(flightI, flightII,
flightIII);
```

```
                        AddIti(Choices, fulliti);
                    }
                    flightII=NULL;
                }
                flightI=NULL;
            }
        }

    for(i=1;i<=M->vexnum;i++)        //转两次机的路径
        for(j=1;j<=M->vexnum;j++)
            if(Departure!=i&&i!=j&&j!=Arrival&&i!=Arrival&&Departure!=j&&
Departure!=Arrival&&M->arcs[Departure][i].adj==1&&M->arcs[i][j].adj==1&&M->a
rcs[j][Arrival].adj==1)
            {
                GetFlightID(schedule, Departure, i, A_IDI);
                GetFlightID(schedule, i, j, A_IDII);
                GetFlightID(schedule, j, Arrival, A_IDIII);
                for(i=1;i<=A_IDI[0];i++)
                {
                    flight1=GetRecord(schedule, A_IDI[i]);
                    flightI = CreateFlight(flight1);
                    for(j=1;j<=A_IDII[0];j++)
                    {
                        flight2=GetRecord(schedule, A_IDII[j]);
                        flightII = CreateFlight(flight2);
                        for(k=1;k<=A_IDIII[0];k++)
                        {
                            flight3=GetRecord(schedule, A_IDIII[k]);
                            flightIII = CreateFlight(flight3);
                            if(flightI->arrival_time1<flightII->departure_t
ime1&&flightII->arrival_time1<flightIII->departure_time1)
                            {
```

```
                                FullIti * fulliti = CreateFullIti(flightI,
flightII, flightIII);
                                AddIti(Choices, fulliti);
                            }
                            flightIII=NULL;
                        }
                        flightII=NULL;
                    }
                    flightI=NULL;
                }
            }
    }

    else if(M->arcs[Departure][Arrival].adj==2)    //需要转一次机
    {
        for(i=1;i<=M->vexnum;i++)        //转一次机的路径
            if(Departure!=i&&i!=Arrival&&Departure!=Arrival&&M->arcs[Departur
e][i].adj==1&&M->arcs[i][Arrival].adj==1)
            {
                GetFlightID(schedule, Departure, i, B_IDa);
                GetFlightID(schedule, i, Arrival, B_IDb);
                for(j=1;j<=B_IDa[0];j++)
                {
                    flight1=GetRecord(schedule, B_IDa[j]);
                    flightI = CreateFlight(flight1);
                    for(k=1;k<=B_IDb[0];k++)
                    {
                        flight2=GetRecord(schedule, B_IDb[k]);
                        flightII = CreateFlight(flight2);
                        if(flightI->arrival_time1<flightII->departure_time1)
                        {
                            fulliti = CreateFullIti(flightI, flightII,
flightIII);
```

```
                        AddIti(Choices, fulliti);
                    }
                    flightII=NULL;
                }
                flightI=NULL;
            }
        }

    for(i=1;i<=M->vexnum;i++)        //转两次机的路径
        for(j=1;j<=M->vexnum;j++)
            if(Departure!=i&&i!=j&&j!=Arrival&&i!=Arrival&&Departure!=j&&
Departure!=Arrival&&M->arcs[Departure][i].adj==1&&M->arcs[i][j].adj==1&&M->a
rcs[j][Arrival].adj==1)
            {
                GetFlightID(schedule, Departure, i, B_IDI);
                GetFlightID(schedule, i, j, B_IDII);
                GetFlightID(schedule, j, Arrival, B_IDIII);
                for(i=1;i<=B_IDI[0];i++)
                {
                    flight1=GetRecord(schedule, B_IDI[i]);
                    flightI = CreateFlight(flight1);
                    for(j=1;j<=B_IDII[0];j++)
                    {
                        flight2=GetRecord(schedule, B_IDII[j]);
                        flightII = CreateFlight(flight2);
                        for(k=1;k<=B_IDIII[0];k++)
                        {
                            flight3=GetRecord(schedule, B_IDIII[k]);
                            flightIII = CreateFlight(flight3);
                            if(flightI->arrival_time1<flightII->departure_t
ime1&&flightII->arrival_time1<flightIII->departure_time1)
                            {
```

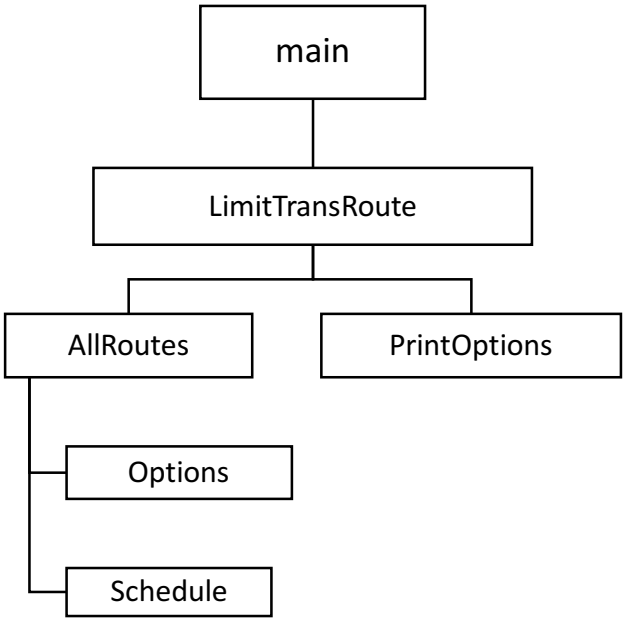
```
                                fulliti = CreateFullIti(flightI, flightII,
flightIII);
                                AddIti(Choices, fulliti);
                            }
                            flightIII=NULL;
                        }
                        flightII=NULL;
                    }
                    flightI=NULL;
                }
            }
    }

    else if(M->arcs[Departure][Arrival].adj==3)    //需要转两次机
    {
        for(i=1;i<=M->vexnum;i++)          //转两次机的路径
            for(j=1;j<=M->vexnum;j++)
                if(Departure!=i&&i!=j&&j!=Arrival&&i!=Arrival&&Departure!=j&&
Departure!=Arrival&&M->arcs[Departure][i].adj==1&&M->arcs[i][j].adj==1&&M->a
rcs[j][Arrival].adj==1)
                {
                    GetFlightID(schedule, Departure, i, C_IDI);
                    GetFlightID(schedule, i, j, C_IDII);
                    GetFlightID(schedule, j, Arrival, C_IDIII);
                    for(i=1;i<=C_IDI[0];i++)
                    {
                        flight1=GetRecord(schedule, C_IDI[i]);
                        flightI = CreateFlight(flight1);
                        for(j=1;j<=C_IDII[0];j++)
                        {
                            flight2=GetRecord(schedule, C_IDII[j]);
                            flightII = CreateFlight(flight2);
                            for(k=1;k<=C_IDIII[0];k++)
```

```
                        {
                            flight3=GetRecord(schedule, C_IDIII[k]);
                            flightIII = CreateFlight(flight3);
                            if(flightI->arrival_time1<flightII->departure_t
ime1&&flightII->arrival_time1<flightIII->departure_time1)
                            {
                                fulliti = CreateFullIti(flightI, flightII,
flightIII);
                                AddIti(Choices, fulliti);
                            }
                            flightIII=NULL;
                        }
                        flightII=NULL;
                    }
                    flightI=NULL;
                }
            }
    }

    else if(M->arcs[Departure][Arrival].adj==-1)
        return ERROR;
    return OK;
} //AllRoutesV1
```

4. 主函数的伪码算法：

```
void main()
{  //主函数
    scanf(&Departure, &Arrival);
    scanf(&NumTransfers);
    LimitTransRouteV1(schedule, mgraph, Departure, Arrival, NumTransfers);
} //main
```

5. 函数的调用关系图反映了演示程序的层次结构：



模块 V 给定起飞时段或降落时段或机型要求，或者给定飞行时长或转机时间限制，求任意两个机场的多个备选航线，或者求航费最低的航线。

1. 在模块 IV 的基础上增加了筛选条件解析和判断算法：

具体的伪码算法如下：

```
int GetCondition(int DepartureTime1, int DepartureTime2, int ArrivalTime1,
int ArrivalTime2, int AirplaneModel, int FlightDuration, int
TransferDuration)
{
    Condition=0;
    if(DepartureTime1>0&&DepartureTime2>0)
    {
        if(ArrivalTime1>0&&ArrivalTime2>0)
        {if(AirplaneModel>0) Condition=123;else Condition=12;}
        else
        {if(AirplaneModel>0) Condition=13;else Condition=1;}
    }
    else
    {
        if(ArrivalTime1>0&&ArrivalTime2>0)
        {if(AirplaneModel>0) Condition=23;else Condition=2;}
        else
```

```
        {if(AirplaneModel>0) Condition=3;else Condition=0;}
    }

    if(FlightDuration>0)
    {
        if(TransferDuration>0)
            Condition=45;
        else
            Condition=4;
    }
    else
        if(TransferDuration>0)
            Condition=5;
    return Condition;
} //GetCondition

int ConditionSatisfiedV1(FullIti * iti, int DepartureTime1, int
DepartureTime2, int ArrivalTime1, int ArrivalTime2, int AirplaneModel, int
FlightDuration, int TransferDuration, int Condition)
{
    satisfied=0;
    if(iti!=NULL)
    {
        departuretime1=iti->first->departure_time1;
        arrivaltime1=iti->first->arrival_time1;
        flightduration=iti->first->arrival_time1-iti->first->departure_time1;
        transferduration=0;
        if(iti->second!=NULL)
        {arrivaltime1=iti->second->arrival_time1;flightduration+=iti->second-
>arrival_time1-
iti->second->departure_time1;transferduration+=iti->second->departure_time1-
iti->first->arrival_time1;}
        if(iti->third!=NULL)
```

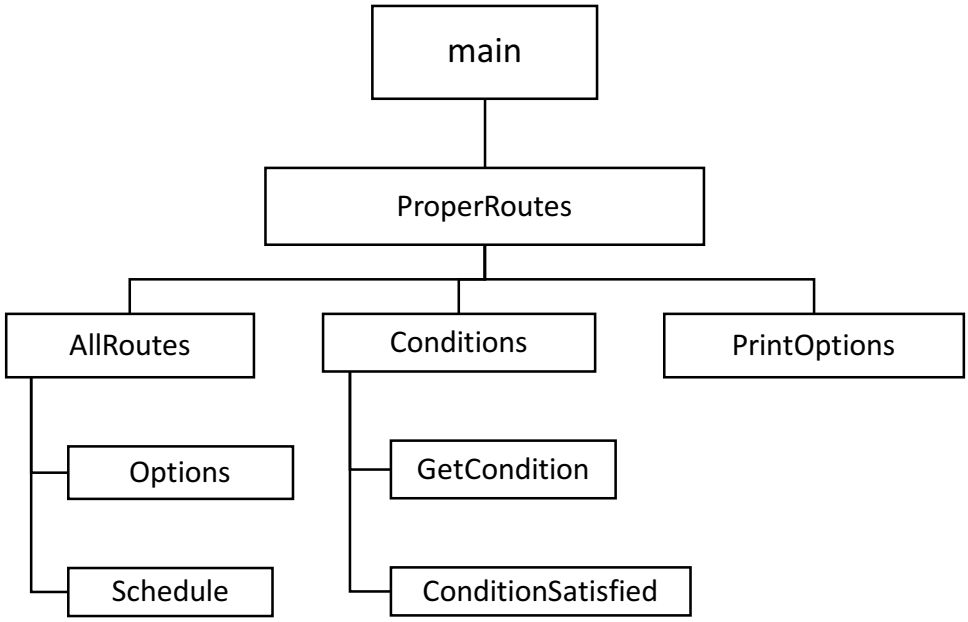
```
        {arrivaltime1=iti->third->arrival_time1;flightduration+=iti->third->a
rrival_time1-
iti->third->departure_time1;transferduration+=iti->third->departure_time1-
iti->second->arrival_time1;}


        switch (Condition) {
            case 0:
                satisfied=1;
                break;
            case 1:
                if(DepartureTime1<=departuretime1&&DepartureTime2>=departuret
ime1)
                    satisfied=1;
                break;
            case 12:
                if(DepartureTime1<=departuretime1&&DepartureTime2>=departuret
ime1&&ArrivalTime1<=arrivaltime1&&ArrivalTime2>=arrivaltime1) satisfied=1;
                break;
            case 13:
                if(DepartureTime1<=departuretime1&&DepartureTime2>=departuret
ime1&&AirplaneModel==iti->first->airplane_model) satisfied=1;
                break;
            case 123:
                if(DepartureTime1<=departuretime1&&DepartureTime2>=departuret
ime1&&ArrivalTime1<=arrivaltime1&&ArrivalTime2>=arrivaltime1&&AirplaneModel=
=iti->first->airplane_model) satisfied=1;
                break;
            case 2:
                if(ArrivalTime1<=arrivaltime1&&ArrivalTime2>=arrivaltime1)
                    satisfied=1;
                break;
            case 23:
```

```
                if(ArrivalTime1<=arrivaltime1&&ArrivalTime2>=arrivaltime1&&Ai
rplaneModel==iti->first->airplane_model) satisfied=1;
                break;
            case 3:
                if(AirplaneModel==iti->first->airplane_model) satisfied=1;
                break;
            case 4:
                if(flightduration<=FlightDuration) satisfied=1;
                break;
            case 45:
                if(flightduration<=FlightDuration&&transferduration<=Transfer
Duration) satisfied=1;
                break;
            case 5:
                if(transferduration<=TransferDuration) satisfied=1;
                break;
            default:
                break;
        }
    }
    return satisfied;
} //ConditionSatisfiedV1
```

2. 主函数的伪码算法：

```
void main()
{  //主函数
    scanf(&Departure, &Arrival);
    scanf(&Conditions);
    ProperRoute(Schedule, Departure, Arrival, Conditions);
} //main
```

3. 函数的调用关系图反映了演示程序的层次结构：


main
ProperRoutes
AllRoutes
Conditions
PrintOptions
Options
Schedule
GetCondition
ConditionSatisfied

四、 调试分析

模块 I 读取 csv 文件，并进行合适的处理。

1. 自行定义结构体用于存放单条航班信息，并利用指针数组存储所有的航班结构体，方便高效。
2. 算法的时空分析：线性结构，时间复杂度和空间复杂度均为 O(n)。

模块 II 读取航班安排表（数据框图），构建以邻接表为存储结构的图，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。

1. 邻接表的空间复杂度为 O(n+e)。
2. 借助栈和队列作为辅助进行遍历，可以有效避免访问已经访问过的结点，时间复杂度均为 O(n+e)。
3. 由于借助栈和队列作为辅助需要额外的存储空间，空间复杂度为 O(n)。

模块 III 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；并求任意两个机场之间的最短飞行时间。

1. 从邻接表的表头结点出发进行类广度优先遍历，将机场连通信息存入到邻接矩阵中，时间复杂度为 O(n)。
2. 查找两个机场之间的连通性时，可直接根据下标进行查找，时间复杂度为 O(1)。
3. 采用邻接矩阵作为存储结构，空间复杂度为 O(n^2)。
4. 计算最短飞行时长时，需要根据输入的起飞、降落机场序号在航班安排表中找出两个机场之间所有的航线，然后分别计算飞行时长，并进行比较，找出最短时长。

模块 IV 仅限直飞或一次中转，求任意两个机场的航线。

1. 考虑到算法的通用性，先把两个机场之间所有的航线用单链表存储起来，用户提出各种要求，在遍历单链表的过程中将只输出符合要求的航线，这样可以避免重复获取航班信息。
2. 获取所有航线的过程比较复杂，需要考虑直飞、一次中转、两次中转等不同情况，时间复杂度约为 O(n^4)。
3. 由于借助单链表结构，算法的空间复杂度为 O(n)。

模块 V 给定起飞时段或降落时段或机型要求，或者给定飞行时长或转机时间限制，求任意两个机场的多个备选航线，或者求航费最低的航线。

1） 基本操作与模块 IV 一致， 增加了一个模块用于判断筛选条件。

五、 用户手册

模块 I 读取 csv 文件，并进行合适的处理。

1. 合理定义文件路径，推荐将 csv 文件存放在 C 文件同一级目录下，免去定义路径的麻烦。
2. 程序将自行完成读取、存放以及打印输出。

模块 II 读取航班安排表（数据框图），构建以邻接表为存储结构的图，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。

1. 运行程序，对话框显示“输入起始机场序号”。
2. 输入一个正整数，敲击回车键完成输入。
3. 从所输入的起始机场开始进行深度优先遍历。
4. 从所输入的起始机场开始进行广度优先遍历。
5. 对话框显示遍历结果。

模块 III 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；并求任意两个机场之间的最短飞行时间。

1. 运行程序，对话框显示“输入起飞和降落机场序号”。
2. 输入一对正整数，敲击回车键完成输入。
3. 对话框将显示两个机场之间的连通信息。
4. 对话框将显示两个机场之间的最短飞行时长。

模块 IV 仅限直飞或一次中转，求任意两个机场的航线。

1. 运行程序，对话框显示“请输入出发和抵达机场”。
2. 用户键入两个序号。
3. 对话框显示“请输入最大转机次数”。
4. 用户键入最大转机次数。
5. 对话框将显示符合条件的所有航线。
6. 程序结束。

模块 V 给定起飞时段或降落时段或机型要求，或者给定飞行时长或转机时间限制，求任意两个机场的多个备选航线，或者求航费最低的航线。

1. 运行程序，对话框显示“请输入出发和抵达机场”。
2. 用户键入两个序号。
3. 对话框显示“请输入筛选条件”。
4. 用户键入各种筛选条件（如果不作限定，根据提示进行相应的输入即可）。
5. 对话框将显示符合条件的所有航线。
6. 程序结束。

六、 测试结果

模块 I 读取 csv 文件，并进行合适的处理。

1. 分两部分打印输出了航班安排表。
2. 打印输出了航班的行数和列数：2346 Rows * 11 Columns

模块 II 读取航班安排表（数据框图），构建以邻接表为存储结构的图，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。

1. 获取起始机场序号: 键入数字“2”，确定起始序号为 2。
2. 深度优先遍历结果为:

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

3. 广度优先遍历结果为:

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

4. 获取起始机场序号: 键入数字“4”，确定起始序号为 2。
5. 广度优先遍历结果为:

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

模块 III 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；并求任意两个机场之间的最短飞行时间。

1. **Airport No.3-->Airport No.49: Two Transfers**

   Airport No.3-->Airport No.49: 43 h 30 min (Shortest Duration)

2. **Airport No.6-->Airport No.49: Direct Flight**

   Airport No.6-->Airport No.49: 1 h 40 min (Shortest Duration)

3. **Airport No.50-->Airport No.30: Direct Flight**

   Airport No.50-->Airport No.30: 2 h 35 min (Shortest Duration)

4. **Airport No.2-->Airport No.16: One Transfer**

   Airport No.2-->Airport No.16: 18 h 10 min (Shortest Duration)

模块 IV 仅限直飞或一次中转，求任意两个机场的航线。

1. `LimitTransRouteV1(schedule, mgraph, 3, 49, 1);`

FROM: Airport No.3 TO: Airport No.49 1 Transfer

Flight NO. Departure Departure Time Arrival Arrival Time Dome/Intl Airplane Model Fares

2. LimitTransRouteV1(schedule, mgraph, 6, 49, 0);

FROM: Airport No.6 TO: Airport No.49 0 Transfer

Flight NO. Departure Departure Time Arrival Arrival Time Dome/Intl Airplane Model Fares

515 6 5/6/2017 19:45 49 5/6/2017 21:25 Intl 2 1008

515 6 5/7/2017 19:45 49 5/7/2017 21:25 Intl 2 987

515 6 5/8/2017 19:45 49 5/8/2017 21:25 Intl 2 1212

515 6 5/5/2017 19:45 49 5/5/2017 21:25 Intl 2 1073

3. LimitTransRouteV1(schedule, mgraph, 6, 49, 1);

FROM: Airport No.6 TO: Airport No.49 1 Transfer

Flight NO. Departure Departure Time Arrival Arrival Time Dome/Intl Airplane Model Fares

4. LimitTransRouteV1(schedule, mgraph, 50, 30, 0);

FROM: Airport No.50 TO: Airport No.30 0 Transfer

Flight NO. Departure Departure Time Arrival Arrival Time Dome/Intl Airplane Model Fares

142 50 5/5/2017 8:25 30 5/5/2017 11:00 Dome 2 1134

142 50 5/6/2017 8:25 30 5/6/2017 11:00 Dome 2 802

142 50 5/7/2017 8:25 30 5/7/2017 11:00 Dome 2 541

142 50 5/8/2017 8:25 30 5/8/2017 11:00 Dome 2 799

5. LimitTransRouteV1(schedule, mgraph, 2, 16, 1);

FROM: Airport No.2 TO: Airport No.16 1 Transfer

Flight NO. Departure Departure Time Arrival Arrival Time Dome/Intl Airplane Model Fares

```
108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
257     49  5/8/2017 13:15  16  5/8/2017 16:40  Intl    2       2089

108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
257     49  5/7/2017 12:30  16  5/7/2017 15:45  Intl    2       2221

108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
434     49  5/8/2017 9:30   16  5/8/2017 12:45  Intl    2       1906

108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
434     49  5/7/2017 9:30   16  5/7/2017 12:45  Intl    2       1878
```

模块 V 给定起飞时段或降落时段或机型要求，或者给定飞行时长或转机时间限制，求任意两个机场的多个备选航线，或者求航费最低的航线。

1. 3 号机场到 49 号机场之间，05/07 00:00～05/09/23:59 之间出发，任意时段降落，第一段航程机型为 2，任意飞行时长，任意转机时长的所有航线。

```
FROM: Airport No.3      TO: Airport No.49
No Flights Available
```

2. 3 号机场到 49 号机场之间，05/07 00:00～05/09/23:59 之间出发，任意时段降落，第一段航程任意机型，飞行时长不超过 40 小时，转机时长不超过 10 小时的所有航线。

```
FROM: Airport No.3      TO: Airport No.49
Flight NO.  Departure   Departure Time  Arrival Arrival Time    Dome/Intl
    Airplane Model  Fares
412     3   5/7/2017 1:55   50  5/7/2017 17:25  Intl    5       10281
176     50  5/7/2017 17:55  20  5/7/2017 23:30  Intl    2       3043
495     20  5/8/2017 9:00   49  5/8/2017 14:00  Intl    2       2978

412     3   5/7/2017 1:55   50  5/7/2017 17:25  Intl    5       10281
252     50  5/7/2017 18:05  31  5/7/2017 19:40  Dome    1       876
233     31  5/7/2017 22:00  49  5/7/2017 23:20  Dome    2       426

412     3   5/7/2017 1:55   50  5/7/2017 17:25  Intl    5       10281
279     50  5/7/2017 19:20  33  5/7/2017 21:10  Dome    2       791
```

```
488     33  5/7/2017 21:20  49  5/7/2017 22:55  Dome    2       965

412     3   5/7/2017 1:55   50  5/7/2017 17:25  Intl    5       10281
399     50  5/7/2017 18:40  38  5/7/2017 20:15  Dome    2       334
202     38  5/7/2017 22:10  49  5/7/2017 23:40  Dome    2       372

412     3   5/7/2017 1:55   50  5/7/2017 17:25  Intl    5       10281
455     50  5/7/2017 18:30  62  5/7/2017 19:50  Dome    2       641
418     62  5/7/2017 20:20  49  5/7/2017 22:15  Dome    2       850

412     3   5/7/2017 1:55   50  5/7/2017 17:25  Intl    5       10281
455     50  5/7/2017 18:30  62  5/7/2017 19:50  Dome    2       641
448     62  5/7/2017 21:45  49  5/7/2017 23:35  Dome    2       930
```

3. 2 号机场到 16 号机场之间，任意时段出发出发，05/07 00:00～05/08/12:00 之间降落，第一段航程任意机型，飞行时长不超过 20 小时，转机时长不超过 10 小时的所有航线。

```
FROM: Airport No.2      TO: Airport No.16
Flight NO.  Departure   Departure Time  Arrival Arrival Time    Dome/Intl
    Airplane Model  Fares
108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
434     49  5/7/2017 9:30   16  5/7/2017 12:45  Intl    2       1878

108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
257     49  5/7/2017 12:30  16  5/7/2017 15:45  Intl    2       2221
```

4. 2 号机场到 16 号机场之间，任意时段出发出发，任意时段降落，第一段航程任意机型，任意飞行时长，任意转机时长的花费最少的航线

```
FROM: Airport No.2      TO: Airport No.16
Flight Schedule with the Lowest Fares
Flight NO.  Departure   Departure Time  Arrival Arrival Time    Dome/Intl
    Airplane Model  Fares
108     2   5/6/2017 18:35  49  5/7/2017 6:10   Intl    4       6759
434     49  5/7/2017 9:30   16  5/7/2017 12:45  Intl    2       1878
The Total Cost is $8637
```

七、 附录

源程序文件名清单：

main.c	//主程序
Itinerary_LkList.h	//基于链表的航线实现单元
Graph_Operations.h	//图的各项操作实现单元
AdjList.h	//邻接表实现单元
AdjMatrix.h	//邻接矩阵实现单元
Stack_Queue.h	//栈和队列实现单元
Schedule_DF.h	//航班安排表实现单元