实验报告：航空路线图

涂奕腾 20201018

一、需求分析

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
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
3. 求任意两个机场之间的最短飞行时间
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)
5. 给定起飞时段或者降落时段或者机型要求，求任意两个机场中转次数不超过 K 的路线(一条即可)
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
7. 给定中转时间不超过 m 且中转次数不超过 k 限制,求两个机场的一条备选航线
8. 给定中转时长限制 m,求两个机场之间的航费(机票价格)最低的一条路径

二、概要设计注：问题顺序与 “一、需求分析”中顺序有所不同

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

遍历部分，通过递归实现 DFS 遍历，用队列实现 BFS 遍历，标记经过的点即可由于点不可重复，故遍历的复杂度为 O(N)。只需考虑如何存图。我们定义了 tim 的结构体记录时间，通过 C++的 STL::vector 实现邻接表，边上记录了时间，编号，费用，机型的信息，空间复杂度为 O(航班数目)。

```
struct tim{
    int date, h, m;
    tim() : date(0), h(0), m(0){}
    tim(int _d, int _h, int _m) : date(_d), h(_h), m(_m) {}
    bool operator < ( const tim &x )const{
        if(x.date != date) return date < x.date;
        else if(x.h != h) return h < x.h;
        else return m < x.m;
    }
    bool operator == ( const tim &x )const{return (date == x.date) && (h == x.h) && (m == x.m);}
    friend int operator - (const tim y, const tim x){
        return 24 * 60 * (y.date - x.date) + 60 * (y.h - x.h) + y.m - x.m;
    }
};
struct EDGE{int to, id, cost, model; tim st,ed;};      vector<EDGE>edge[N + 10];
```

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

定义矩阵类型，通过邻接矩阵的乘法来判断是否可以到达，所乘的幂次为为中转次数 +1。由第一问的结果可以得到，在不考虑时间顺序的前提下所有点都可联通，所以直接输出全 1 矩阵即可。时间复杂度为 O (k * (N ^ 3))

```
while(r--){
    for(int i = 1;i <= N; i++)
        for(int j = 1; j <= N; j++)
            dx[i][j] = ans[i][j], ans[i][j] = 0;
    for(int i = 1;i <= N; i++)
        for(int j = 1;j <= N; j++)
            for(int k = 1; k <= N; k++)
                ans[i][j] |=  Matrix[i][k] & dx[k][j];
}
```

3. 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)

使用问题 1 中的建图方式，直接枚举出发机场的所有航班以及中转点的所有航班是否可到达终点。将答案存在 STL::pair 中，直接排序即可。

4. 求任意两个机场之间的最短飞行时间

从第 4 问起需要考虑时间顺序，重新建图。

我们考虑如何存图，可以根据机场编号、时间、起飞/降落(出点、入点)这三个因素拆点建图。拆点后点的总数目 NUM = 4352。存图的则采用链式前向星。每条边记录的信息包括：时间，花费，是否转机，机型。

```
struct {int next, to, cost, t, transfer, model;}e[M * M];
int tot = 0, head[M<<2];
void addedge(int from, int to, int cost, int T, int transfer, int model = 0){
    e[++tot].next = head[from];
    e[tot].to = to, e[tot].cost = cost, e[tot].t = T,e[tot].transfer = transfer,
e[tot].model = model;
    head[from]=tot;
}

map<tim, int> f1[N + 10], f2[N + 10];//f1:出点编号 f2:入点编号
map<int, int> g[M<<1];//编号对应航班
```

然后我们考虑如何建图。首先我们需要根据航班信息连接每趟航班出发地的出点和到达地的入点(费用，耗时，机型)，转机则是在满足时间要求的前提下连接同一机场的不同入点和出点(费用为 0，转机时间，机型为 0)。总边数略大于 1e5。

```
memset(head,0,sizeof(head)); tot = 0;
for(int u = 1; u <= N; ++u){
    for(auto E : edge[u])
    addedge(f1[u][E.st] , f2[E.to][E.ed], E.cost, E.ed - E.st, 0, E.model);
    for(auto i = f2[u].begin(); i != f2[u].end(); i++)  //转机:同一个机场节点的入点连出点
        for(auto j = f1[u].begin(); j != f1[u].end(); j++)
        if(i->fi < j->fi || i->fi == j->fi) addedge(i->se, j->se, 0, j->fi - i->fi, 1, 0);
}
```

这样一来，我们便可以很方便地处理转机问题。然后考虑如何求最小时间。将时间视为距离，则这个问题可以视作最短路径问题。按照上述的建图方式，我们需要跑“出发机场的出点数目”次最短路径算法才能得到最小值。我们不妨设置一个超级源点和一个超级汇点，分别记为 S = NUM + 1,T = NUM + 2。每次建图时只需要从 S 向出发点机场的所有出点建一条费用、耗时均为 0 的边，从到达机场向 T 建一条费用、耗时均为 0 的边，这样以来只需要跑一次最短路径算法即可求出答案：

```
for(auto i = f1[u].begin(); i != f1[u].end(); i++) addedge(S, i->se, 0, 0, 0);
for(auto i = f2[v].begin(); i != f2[v].end(); i++) addedge(i->se, T, 0, 0, 0);
```

而在最短路径算法中，我选择了用线段树优化的 dijkstra 算法(用线段树进行单点修改和区间最小值查询操作)。设节点为数 V，边数为 E，则其时间复杂度为 O(ElogV)(分析见 “四、总结思考”)。核心代码如下：

```
struct SegmentTree{//optimize dijkstra with segment-tree
#define ls p<<1
#define rs p<<1|1
    int Min[M << 3], rk[M << 3], pos[M << 3];
    void pushup(int p){
        if(Min[ls] <= Min[rs]) Min[p] = Min[ls], rk[p] = rk[ls];
        else Min[p] = Min[rs], rk[p] = rk[rs];
    }
    void build(int l, int r, int p) {
        if(l == r){
            Min[p] = inf, pos[rk[p] = l] = p;
            return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, ls); build(mid + 1, r, rs);
        pushup(p);
    }
    void update(int p, int val){
        Min[p] = val;
        while(p>>1) pushup(p >> 1),  p >>= 1;
    }
#undef ls
#undef rs
};
```

```
void dijkstra(int s){
    memset(vis,0,sizeof(vis)), memset(dis,0x7f,sizeof(dis));
    SegmentTree T; dis[s] = 0;
    T.build(1,NUM+2,1);
    T.update(T.pos[s], 0);
    while(T.Min[1] ^ inf){
        int u = T.rk[1]; T.update(T.pos[u], inf);
        if(vis[u]) continue; vis[u]=1;
        for(int i = head[u]; i; i = e[i].next){
            int v = e[i].to;
                if(dis[v] > dis[u] + e[i].t)
                    dis[v] = dis[u] + e[i].t, T.update(T.pos[v], dis[v]);
            }
    }
}
```

5. 给定起飞时段或者降落时段或者机型要求，求任意两个机场中转次数不超过 K 的路线

建图方式类似问题 3。针对起飞时段的限制，我们只需将超级原点连接相应时段内出发机场的出点；针对降落时段的限制，我们只需将相应时段内到达机场的入点连接超级汇点；针对机型限制，我们只需考虑所有与目的机型相吻合的边。将中转次数视作路径长度，用 dijkstra 算法求出最小中转次数，与 K 作比较。

对于输出路径，则需在 dijkstra 中每次进行松弛操作时记录松弛节点的前驱节点。由此我们可以通过迭代出最短路径上终点到起点所经过的节点编号，提前记录节点对应的边的编号(航班 id)，逆序输出即可。

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

类似于于问题 5，将费用视作路径长度，按照问题 5 中的建图方式跑一遍 dijkstra 即可。

7. 给定中转时长限制 m, 求两个机场之间的航费(机票价格)最低的一条路径

第 7 问开始，问题升级为有限制的最短路径问题，使用二维的优先队列优化 dijkstra 算法解决。建图方式同上，但由于数据量较大(点有 4352 个，可能的中转时常限制也有 1e3 的数量级)，可能会炸空间，所以选择牺牲部分时间，使用 unordered_map 记录数据。核心代码如下：

```
#define mp make_pair
unordered_map<int, bool>vis[M<<1];
unordered_map<int, int>dis[M<<1];
unordered_map<int, pii> pre[M<<1];
struct node{
    int pos, ti, cost;
```

```
    bool operator <( const node &x )const{
        if(cost == x.cost) return ti > x.ti;
        return cost > x.cost;
    }
};
priority_queue<node>q;

void dijkstra(int s, int lim){
    for(int i = 1; i <= NUM + 2; ++i) pre[i].clear(), vis[i].clear(), dis[i].clear();
    dis[s][0]=0; q.push((node){s,0,0});
    while(!q.empty()){
        node x=q.top(); q.pop(); int u = x.pos, trans= x.ti;
        if(vis[u][trans]) continue; vis[u][trans] = 1;
        for(int i = head[u]; i; i = e[i].next){
            int v = e[i].to, tr = trans + e[i].t * e[i].transfer, cost =  e[i].cost;
            if(tr > lim) continue;
            if(!dis[v].count(tr) || (dis[v].count(tr) && dis[v][tr] > dis[u][trans] +
cost))
                dis[v][tr] = dis[u][trans] + cost, q.push((node){v, tr, dis[v][tr]}),
pre[v][tr] = mp(u, trans);
        }
    }
}
```

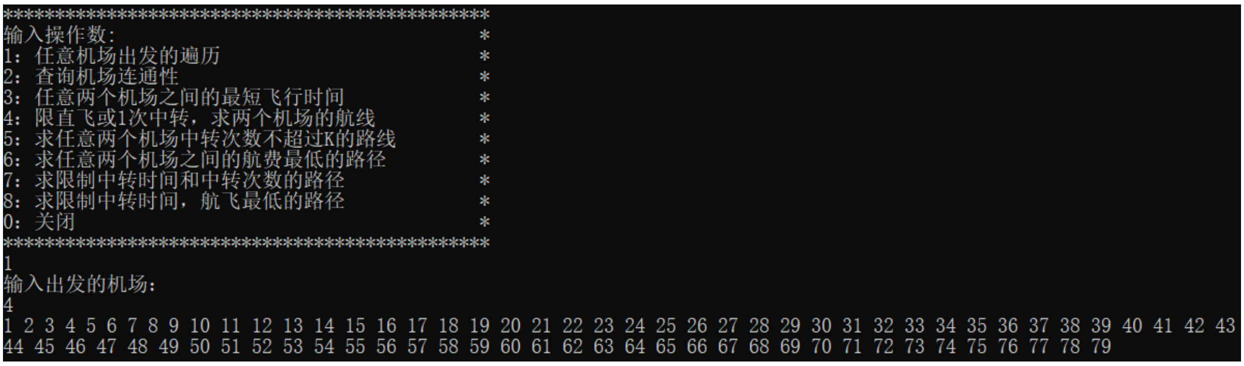
8. 给定中转时间不超过 m 且中转次数不超过 k 限制, 求两个机场的一条备选航线

类似于问题 7，用二维 dijkstra 求解，为了节约空间，可以将转机次数 k 作为 dijkstra 的限制，求最短的转机时间。计算答案时检查 dis 中是否有满足条件的 $<k, m>$ 对即可。

三、测试用例

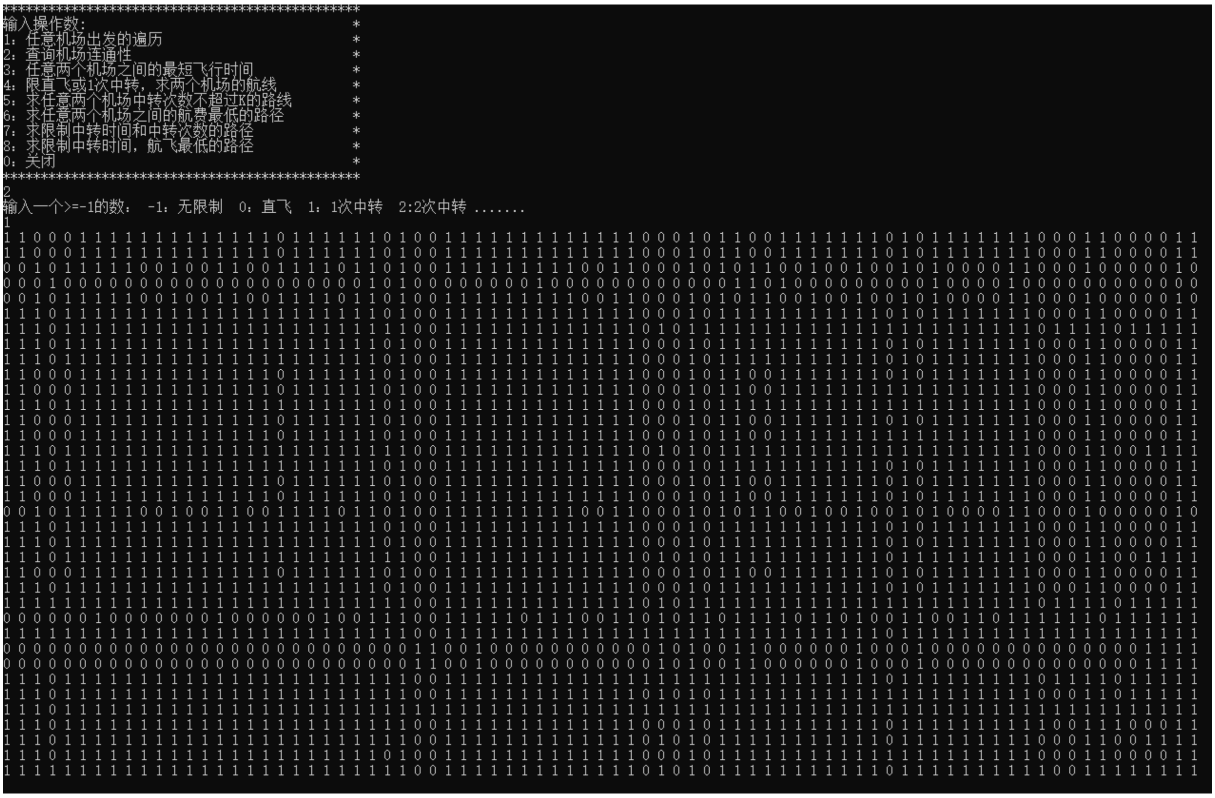
1. 用邻接表来完成从任意机场出发的遍历，包括深度优先遍历和广度优先遍历

```
1

4
```



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

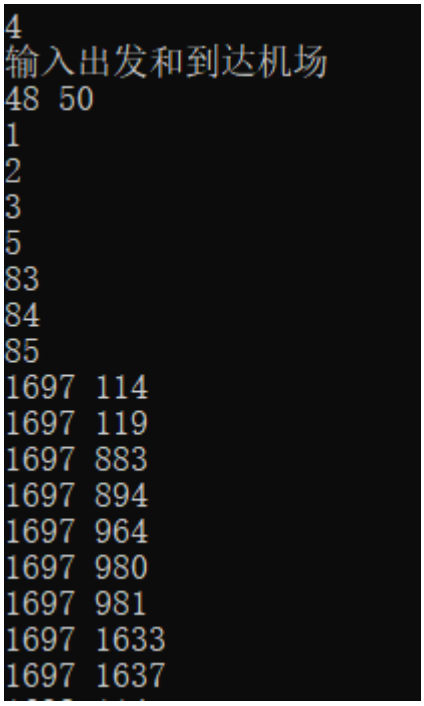
```
2

1
```

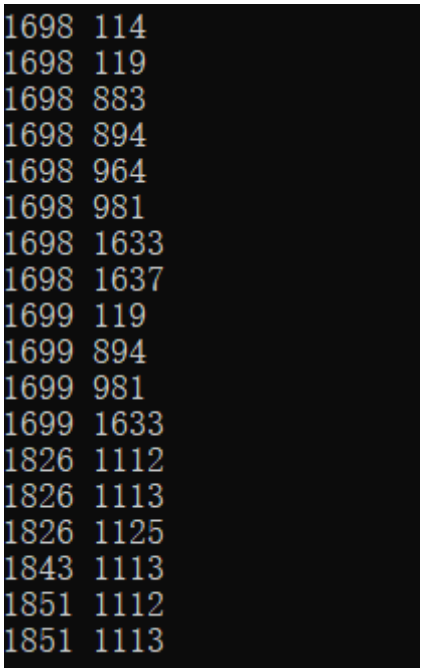


注：默认自环始终存在

3. 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)

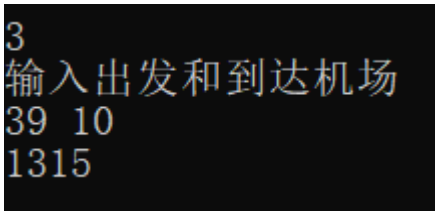
```
4

48 50
```





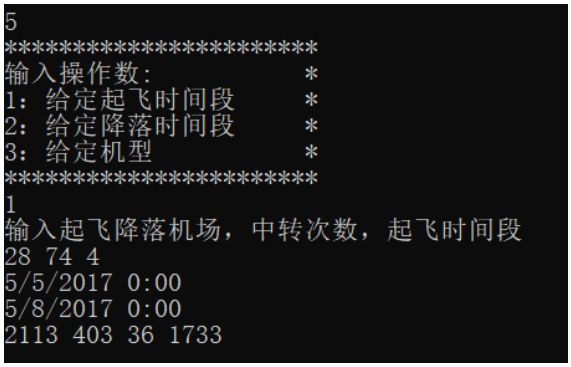
4. 求任意两个机场之间的最短飞行时间

```
3

39 10
```



5. 给定起飞时段或者降落时段或者机型要求，求任意两个机场中转次数不超过 K 的路线

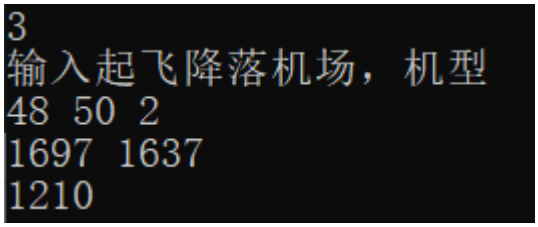
```
5

1

28 74 4

5/5/2017 0:00

5/8/2017 0:00
```



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

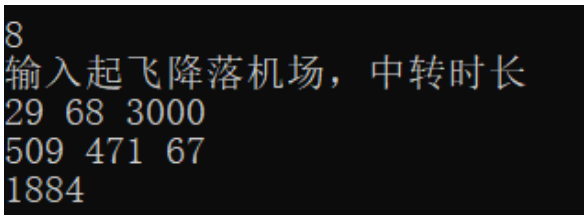
```
6

1

48 50

5/5/2017 0:00

5/6/2017 0:00
```


```
6

3

48 50 2
```



7. 给定中转时长限制 m，求两个机场之间的航费(机票价格)最低的一条路径

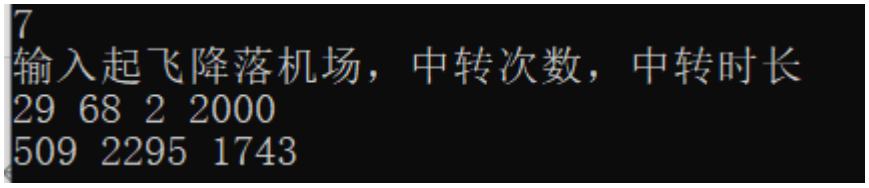
```
8

29 68 3000
```



8. 给定中转时间不超过 m 且中转次数不超过 k 限制，求两个机场的一条备选航线

```
7

39 10 4 610
```


```
7

29 68 2 2000
```



四、总结思考——dijkstra 算法的时间复杂度分析

Dijkstra 算法的核心步骤为 2 步：(记已确定最短路长度的点的集合为 S，而未确定最短路长度的点的集合为 T)

(1)从 T 集合中选取一个最短路长度最小的节点，移至 S 集合中

(2)对刚加入 S 集合的节点的所有出边执行松弛操作

我们对以下几种 dijkstra 算法的实现进行分析：

1.暴力：不使用任何数据结构维护

每次 (1)中直接在 T 集合中暴力枚举寻找最短路长度最小的结点。2 操作总时间复杂度为 O(E), (1)操作的的总时间为 O(V^2)，故总时间复杂度为 O(V^2 + E) = O(V^2)。

2.二叉堆

每松弛一条边(u, v)就将 v 插入二叉堆中，若 v 已在二叉堆中可以直接修改相应的权值，这样(1)操作直接取堆顶元素即可。一共需要进行 E 次插入/修改操作，V 次删除堆顶操作，每次操作的复杂度都是 O(logV)，则总复杂度：O((V+E)logV) = O(ElogV)

3.优先队列

类似于二叉堆，但是由于先前更新时插入的元素不能被删除，也不能被修改，所以优先队列中元素个数是 O(E)的，时间复杂度为 O(ElogE)

4.线段树

类似于二叉堆，将堆的插入操作改为线段树上的单点修改，而(1)则是线段树上查询全局最小值，这两个操作的时间复杂度均为 O(logV)，总时间复杂度为 O(ElogV)

5.斐波那契堆

类似于二叉堆，但斐波那契堆插入的时间复杂度为 O(1)，故总时间复杂度为 O(VlogV+E) = O(VlogV)。但是由于斐波那契堆实现较复杂且常数较大，所以使用较少。