



北京師範大學
BEIJING NORMAL UNIVERSITY

一、实验要求

1. 上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。
2. 独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。
3. 上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

二、实验内容

设计、实现一个全国大城市间的交通咨询程序，为旅客提供三种最优决策方案：（1）

飞行时间最短（2）总用时最短（3）费用最小（4）中转次数最少。

三、实验步骤（写出问题分析或者算法思路）

飞行时间最少和费用最小可以归结为基础的 Dijkstra 算法实现，中转次数最少可以将有向图的权值设为 1，然后再使用 Dijkstra 算法。

但是在总用时最短问题时要对 Dijkstra 算法进行一点修改，

算法的基础图还是飞行时间的有向图：

在找到 dist 数组中最小者(即一条新的单源最短路径)后,我们要对 dist 数组重新进行修改,来判断通过现在的单源最短路径出发到新城市的时间花费是否比原来 dist 数组中存的更小,不过由于考虑的是总时间,所以在计算到新城市的时间时不仅要有飞行时间,还要加上中间由于转机产生的时间间隔,如从北京到西安是 17:00 到达,但是再从西安到广州的飞机是 7:15 起飞,因此就会有等候飞机的时间间隔,在修改 dist 数组时将这个时间间隔考虑进去。这样的话通过原来的 Dijkstra 的算法,通过一步步扩展单源最短路径,得到的就是从某一城市出发,到其他城市总时间最少的飞行路线。

求总时间最少路线的 Dijkstra 算法代码如下图：

```
void ShortestPath(MGraph& G, Type start_city, Weight dist[], int path[])
{
    int source_index = GetVertexPos(G, start_city);
    int n = G.numVertices; int already_find[maxVertices];
    int i, j, k;
    Weight w, min_weight;
    for(i=0; i<n; i++)
    {
        dist[i] = G.Edge[source_index][i];
        already_find[i] = 0;
        if(i != source_index && dist[i] < maxWeight) path[i] = source_index;
        else path[i] = -1;
    }
    already_find[source_index] = 1;
    dist[source_index] = 0;
    for(i=0; i<n-1; i++)
    {
        min_weight = maxWeight;
        int temp_index = source_index;
        for(j=0; j<n; j++)
            if(!already_find[j] && dist[j] < min_weight) {temp_index = j; min_weight = dist[j];}
        already_find[temp_index] = 1;
        for(k=0; k<n; k++)
        {
            w = G.Edge[temp_index][k];
            if(!already_find[k] && w < maxWeight && dist[temp_index] + w < dist[k])
            {
                dist[k] = dist[temp_index] + w;
                path[k] = temp_index;
            }
        }
    }
}
```

同时还要提前考虑时间间隔如何表示的问题，我采取的方法是：对应于原来的有向图邻接矩阵新建一个对应的二维数组“时间矩阵”，二维数组的里面存放的是“时间结点”，通过结构体定义，二维数组就像邻接矩阵一样，行编号和列编号对应的是结点数组里城市的编号，如北京 0，上海 1，西安 2 等。(0, 1) 对应北京到上海的时间结点。时间节点里存放 起始地 到 目的地的飞机起飞时间和到达时间，都化为分钟表示。如 (0, 1) 对应的是北京到上海的起飞时间和到达时间。

通过“时间矩阵”，在 Dijkstra 算法中重新修改 dist 数组时，就可以计算得到从新找到的单源最短路径末端的城市（其下标用 temp_index 代表）到其他城市所需的时间间隔。比如，从北京出发，遍历一遍 dist 数组后找到最短路径对应的城市是上海，然后我们要去修改 dist 数组，从上海到西安的时间间隔可以通过 从北京到上海的到达时间 和 从上海到西安的出发时间 这两个数据算出，因为我们已有上海对应的下标 (temp_index)，我们可以通过 path[temp_index] 得到最短路径中上海的上一个城市（北京）的下标 (last_index)，进而通过“时间矩阵”就可以得到 从北京到上海的时间节点及其存储的数据。

关键代码如下图：

```
int last_index=path[temp_index];
for(k=0;k<n;k++)
{
    int internal_time=0;
    if(T.time_pairs[temp_index][k].able_flag==1)
    {
        if(T.time_pairs[temp_index][k].start_time<T.time_pairs[last_index][temp_index].arrive_time)
            internal_time=T.time_pairs[temp_index][k].start_time+24*60-T.time_pairs[last_index][temp_index].arrive_time;
        else
            internal_time=T.time_pairs[temp_index][k].start_time-T.time_pairs[last_index][temp_index].arrive_time;
    }
    w=G.Edge[temp_index][k]+internal_time;
    if(!already_find[k]&&w<maxWeight&&dist[temp_index]+w<dist[k])
    {
        dist[k]=dist[temp_index]+w;
        path[k]=temp_index;
    }
}
```

时间节点和时间矩阵的定义：(able_flag 表示有无路径)

```
typedef struct
{
    int start_time, arrive_time, able_flag;
}time_node;
typedef struct
{
    int numVertices, numEdges;
    Type VerticesList[maxVertices];
    time_node time_pairs[maxVertices][maxVertices];
}Time_lists;
```

创建时间矩阵：

```
void createTime_lists(Time_lists& T, Type v[], int verticenum, Type ed[][2], int time_schedule[][2], int edgenum)
{
    T.numVertices=verticenum;
    T.numEdges=edgenum;
    int i, j, k;
    for(i=0; i<T.numVertices; i++)
    {
        T.VerticesList[i]=v[i];
        for(j=0; j<T.numVertices; j++)
            T.time_pairs[i][j].able_flag=0;
    }
    for(k=0; k<T.numEdges; k++)
    {
        i= GetTimeVertexPos(T, ed[k][0]);
        j= GetTimeVertexPos(T, ed[k][1]);
        T.time_pairs[i][j].start_time=time_schedule[k][0];
        T.time_pairs[i][j].arrive_time=time_schedule[k][1];
        T.time_pairs[i][j].able_flag=1;
    }
}
```

四、程序清单（源程序代码等）

```
#include <iostream>

#include<stdio.h>

#include<string>

#define maxVertices 10

#define maxEdges 30

#define maxWeight 20000

using namespace std;

typedef string Type;

typedef int Weight;

typedef struct

{

    int numVertices,numEdges;

    Type VerticesList[maxVertices];

    Weight Edge[maxVertices][maxVertices];

}MGraph;

typedef struct          //时间节点定义

{

    int start_time,arrive_time,able_flag;

}time_node;

typedef struct          //时间矩阵定义

{

    int numVertices,numEdges;

    Type VerticesList[maxVertices];

    time_node time_pairs[maxVertices][maxVertices];

}Time_lists;

int GetTimeVertexPos(Time_lists&T,Type x)    //通过 x 的值找到时间矩阵
```

中 x 的下标

```
{
    for (int i=0; i<T.numVertices; i++)
    {
        if (T.VerticesList[i]==x) return i;
    }
    return -1;
}

void createTime_lists(Time_lists& T, Type v[], int verticenum, Type
ed[][2], int time_schedule[][2], int edgenum)    //创建时间矩阵
{
    T.numVertices=verticenum;
    T.numEdges=edgenum;
    int i, j, k;
    for (i=0; i<T.numVertices; i++)
    {
        T.VerticesList[i]=v[i];
        for (j=0; j<T.numVertices; j++)
            T.time_pairs[i][j].able_flag=0;
    }
    for (k=0; k<T.numEdges; k++)
    {
        i= GetTimeVertexPos(T, ed[k][0]);
        j= GetTimeVertexPos(T, ed[k][1]);
        T.time_pairs[i][j].start_time=time_schedule[k][0];
        T.time_pairs[i][j].arrive_time=time_schedule[k][1];
        T.time_pairs[i][j].able_flag=1;
    }
}

int GetVertexPos(MGraph& G, Type x)
```

```

    {
        for (int i=0; i<G.numVertices; i++)
            if (G.VerticesList[i]==x) return i;
        return -1;
    }

    int numberOfVertices (MGraph& G)
    {
        return G.numVertices;
    }

    void createMGraph (MGraph& G, Type v[], int verticenum, Type
ed[][2], Weight c[], int edgenum)
    {

        G.numVertices=verticenum;
        G.numEdges=edgenum;
        int i, j, k;
        for (i=0; i<G.numVertices; i++)
        {
            G.VerticesList[i]=v[i];
            for (j=0; j<G.numVertices; j++)
                G.Edge[i][j]=(i==j)?0:maxWeight;
        }
        for (k=0; k<G.numEdges; k++)
        {
            i= GetVertexPos (G, ed[k][0]);
            j= GetVertexPos (G, ed[k][1]);
            G.Edge[i][j]=c[k];
        }
    }

    void Shortest_Total_Time_Path (MGraph& G, Type start_city, Weight

```

```

dist[], int path[], Time_lists& T)    //总时间最短的 Dijkstra 算法
{
    int source_index=GetVertexPos(G, start_city);

    int n=G.numVertices;int already_find[maxVertices];

    int i, j, k;

    Weight w, min_weight;

    for (i=0; i<n; i++)
    {
        dist[i]=G.Edge[source_index][i];

        already_find[i]=0;

        if(i!=source_index&&dist[i]<maxWeight) path[i]=source_index;

        else path[i]=-1;
    }

    already_find[source_index]=1;

    dist[source_index]=0;

    for (i=0; i<n-1; i++)
    {
        min_weight=maxWeight;

        int temp_index=source_index;

        for (j=0; j<n; j++)

            if(!already_find[j]&&dist[j]<min_weight)
{temp_index=j;min_weight=dist[j];}

        already_find[temp_index]=1;

        int last_index=path[temp_index];

        for (k=0; k<n; k++)
        {

            int internal_time=0;

            if(T.time_pairs[temp_index][k].able_flag==1)

{    if(T.time_pairs[temp_index][k].start_time<T.time_pairs[last_index][t

```

```

emp_index].arrive_time)

internal_time=T.time_pairs[temp_index][k].start_time+24*60-T.time_pairs[
last_index][temp_index].arrive_time;

        else

internal_time=T.time_pairs[temp_index][k].start_time-T.time_pairs[last_i
ndex][temp_index].arrive_time;

        }

w=G.Edge[temp_index][k]+internal_time;

if(!already_find[k]&&w<maxWeight&&dist[temp_index]+w<dist[k])
{
    dist[k]=dist[temp_index]+w;
    path[k]=temp_index;
}
}

}

void ShortestPath(MGraph& G,Type start_city,Weight dist[],int path[])
//普通的Dijkstra 算法
{
    int source_index=GetVertexPos(G,start_city);
    int n=G.numVertices;int already_find[maxVertices];
    int i,j,k;
    Weight w,min_weight;
    for(i=0;i<n;i++)
    {
        dist[i]=G.Edge[source_index][i];

```

```

        already_find[i]=0;

        if(i!=source_index&&dist[i]<maxWeight) path[i]=source_index;
        else path[i]=-1;
    }

    already_find[source_index]=1;
    dist[source_index]=0;
    for (i=0; i<n-1; i++)
    {
        min_weight=maxWeight;
        int temp_index=source_index;
        for (j=0; j<n; j++)
            if(!already_find[j]&&dist[j]<min_weight)
{temp_index=j;min_weight=dist[j];}

        already_find[temp_index]=1;
        for (k=0; k<n; k++)
        {
            w=G. Edge[temp_index][k];

            if(!already_find[k]&&w<maxWeight&&dist[temp_index]+w<dist[k])
            {
                dist[k]=dist[temp_index]+w;
                path[k]=temp_index;
            }
        }
    }

}

void print_least_flight_time_Path(MGraph& G, Type start_city, Weight
dist[], int path[])    //输出飞行时间最少的路径
{

```

```

        int v=GetVertexPos(G, start_city);

        cout<<"从"<<G.VerticesList[v]<<"到其他城市飞行时间最短的最优航线
为: "<<endl;

        int i, j, k, n=numberOfVertices(G);
        int d[maxVertices];
        for (i=0; i<n; i++)
        {
            if(i!=v)
            {
                j=i; k=0;
                while(j!=v) {d[k++]=j; j=path[j];}
                d[k++]=v;

                cout<<"到"<<G.VerticesList[i]<<"的最优航线为:";
                while(k>0) cout<<G.VerticesList[d[--k]]<<" ";
                cout<<"    最短飞行时间为:"<<dist[i]<<"分钟"<<endl;
            }
        }
    }
}

```

```

void print_least_flight_cost_Path(MGraph& G, Type start_city, Weight
dist[], int path[])    //输出飞行费用最少的路径
{
    int v=GetVertexPos(G, start_city);

    cout<<"从"<<G.VerticesList[v]<<"到其他城市飞行费用最少的最优航线
为:"<<endl;

    int temp_index, k;

    int n=numberOfVertices(G);

    int output_path[maxVertices];

    for (int i=0; i<n; i++)

```

```

        {
            if(i!=v)
            {
                temp_index=i;k=0;
                while(temp_index!=v) {output_path[k++]=temp_index;
temp_index=path[temp_index];}
                output_path[k++]=v;
                cout<<"到"<<G.VerticesList[i]<<"的最优航线为:";
                while(k>0) cout<<G.VerticesList[output_path[--k]]<<" ";
                cout<<"    最少费用为: "<<dist[i]<<endl;
            }
        }
    }

    void print_least_transfer_flights_Path(MGraph& G, Type
start_city,Weight dist[], int path[])    //输出转机次数最少的路径
    {
        int v=GetVertexPos(G, start_city);
        int n=numberOfVertices(G);
        int output_path[maxVertices];
        cout<<"从"<<G.VerticesList[v]<<"到其他城市转机次数最少的最优航线
为: "<<endl;
        for(int i=0;i<n;i++)
        {
            if(i!=v)
            {
                int temp_index=i; int k=0;
                while(temp_index!=v) {output_path[k++]=temp_index;
temp_index=path[temp_index];}
                output_path[k++]=v;

```

```

        cout<<"到"<<G.VerticesList[i]<<"的最优航线为: ";
        while(k>0) cout<<G.VerticesList[output_path[--k]]<<" ";
        cout<<"    最少转机次数为: "<<dist[i]-1<<endl;
    }
}
}

void print_least_total_flight_time_Path(MGraph& G, Type start_city,
Weight dist[], int path[]) //输出总时间最少的路径
{
    int v=GetVertexPos(G, start_city);
    cout<<"从"<<G.VerticesList[v]<<"到其他城市总时间最短的最优航线为:
"<<endl;

    int i, j, k, n=numberOfVertices(G);
    int d[maxVertices];
    for(i=0; i<n; i++)
    {
        if(i!=v)
        {
            j=i; k=0;
            while(j!=v) {d[k++]=j; j=path[j];}
            d[k++]=v;
            cout<<"到"<<G.VerticesList[i]<<"的最优航线为: ";
            while(k>0) cout<<G.VerticesList[d[--k]]<<" ";

            int hour=dist[i]/60;
            int minute=dist[i]%60;

            cout<<"    最短时间为: "<<hour<<"小时"<<minute<<"分钟
="<<dist[i]<<"分钟"<<endl;
        }
    }
}

```

```
}

int main()
{

    //initialize basic information arrays 基本信息数组的初始化

    Type cities[8]={"北京","上海","乌鲁木齐","西安","广州","昆明","拉萨","武汉"};

    Type flight_routes[16][2]={{{"北京","上海"}, {"上海","北京"}, {"北京","乌鲁木齐"}, {"乌鲁木齐","北京"}, {"北京","西安"}, {"西安","北京"}, {"西安","广州"}, {"广州","西安"}, {"拉萨","昆明"}, {"昆明","拉萨"}, {"拉萨","武汉"}, {"武汉","拉萨"}, {"乌鲁木齐","昆明"}, {"昆明","乌鲁木齐"}, {"武汉","广州"}, {"广州","武汉"}}};

    Weight
flight_time[16]={65, 65, 115, 55, 95, 100, 140, 80, 85, 85, 90, 90, 165, 165, 100, 100}
;

    Weight
flight_cost[16]={680, 680, 1150, 1150, 930, 930, 1320, 1320, 830, 830, 890, 890, 1480, 1480, 810, 810}
;

    Weight transfer_flight[16];

    Weight
time_schedules[16][2]={{980, 1015}, {1080, 1145}, {480, 595}, {645, 700}, {925, 1020}, {755, 855}, {435, 575}, {615, 695}, {620, 705}, {755, 840}, {855, 945}, {985, 1075}, {570, 735}, {785, 950}, {425, 525}, {685, 785}};

    for (int i=0; i<16; i++)

        transfer_flight[i]=1;

    //different dist arrays and path arrays 不同的 dist 和 path 数组

    Weight flight_time_dist[8];

    Weight flight_cost_dist[8];

    Weight transfer_flight_dist[8];
```

```
Weight total_time_dist[8];

int flight_time_path[8];
int flight_cost_path[8];
int transfer_flight_path[8];
int total_time_path[8];
//different Graphs 不同的图
MGraph flight_time_graph;
MGraph flight_cost_graph;
MGraph transfer_flight_graph;
Time_lists internal_times;

createTime_lists(internal_times, cities, 8, flight_routes, time_schedules, 16
);

createMGraph(flight_time_graph, cities, 8, flight_routes, flight_time, 16);

createMGraph(flight_cost_graph, cities, 8, flight_routes, flight_cost, 16);

createMGraph(transfer_flight_graph, cities, 8, flight_routes, transfer_fligh
t, 16);

char whether_to_leave='N';

while(whether_to_leave=='N')
{
    string start_city;
    int choice;
    cout<<"请输入起始城市: ";
```

```
        cin>>start_city;

        cout<<"请选择功能:1. 最短飞行时间 2. 最少飞行费用 3. 最少转机次
数 4. 总时间最少 ";

        cin>>choice;

        switch(choice)
        {

            case 1:

ShortestPath(flight_time_graph, start_city, flight_time_dist, flight_time_p
ath);

        print_least_flight_time_Path(flight_time_graph, start_city, flight_time_di
st, flight_time_path);

                break;

            case 2:

ShortestPath(flight_cost_graph, start_city, flight_cost_dist, flight_cost_p
ath);

        print_least_flight_cost_Path(flight_cost_graph, start_city, flight_cost_di
st, flight_cost_path);

                break;

            case 3:

ShortestPath(transfer_flight_graph, start_city, transfer_flight_dist, trans
fer_flight_path);

        print_least_transfer_flights_Path(transfer_flight_graph, start_city, trans
fer_flight_dist, transfer_flight_path);

                break;
```


case 4:

```
Shortest_Total_Time_Path(flight_time_graph, start_city, total_time_dist, total_time_path, internal_times);

print_least_total_flight_time_Path(flight_time_graph, start_city, total_time_dist, total_time_path);

break;

}

cout<<"是否离开程序? (Y/N) ";

cin>>whether_to_leave;

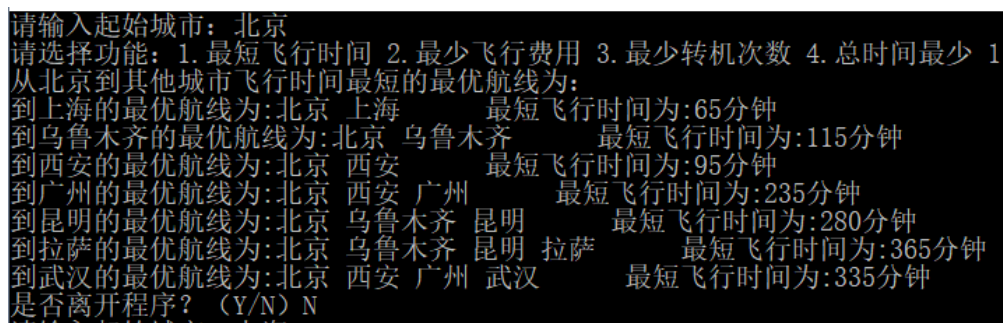
}

return 0;

}
```

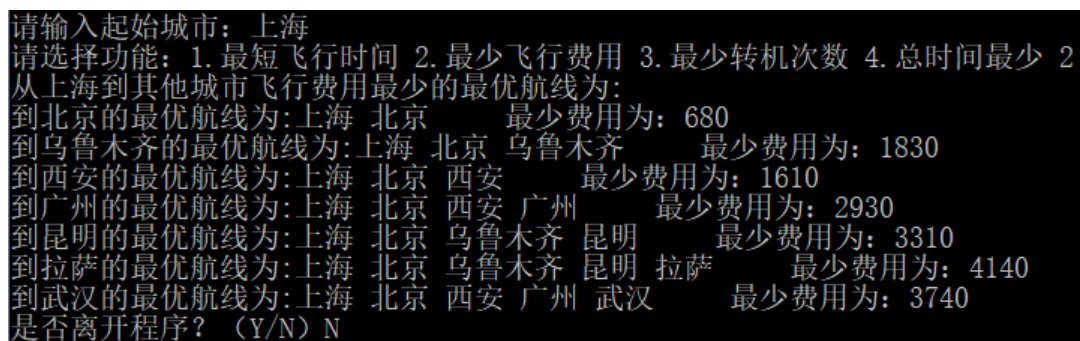
五、运行结果（程序运行时的结果说明或运行截图等）

最短飞行时间：



请输入起始城市：北京
请选择功能：1. 最短飞行时间 2. 最少飞行费用 3. 最少转机次数 4. 总时间最少 1
从北京到其他城市飞行时间最短的最优航线为：
到上海的最优航线为：北京 上海 最短飞行时间为：65分钟
到乌鲁木齐的最优航线为：北京 乌鲁木齐 最短飞行时间为：115分钟
到西安的最优航线为：北京 西安 最短飞行时间为：95分钟
到广州的最优航线为：北京 西安 广州 最短飞行时间为：235分钟
到昆明的最优航线为：北京 乌鲁木齐 昆明 最短飞行时间为：280分钟
到拉萨的最优航线为：北京 乌鲁木齐 昆明 拉萨 最短飞行时间为：365分钟
到武汉的最优航线为：北京 西安 广州 武汉 最短飞行时间为：335分钟
是否离开程序? (Y/N) N

最少飞行费用：



请输入起始城市：上海
请选择功能：1. 最短飞行时间 2. 最少飞行费用 3. 最少转机次数 4. 总时间最少 2
从上海到其他城市飞行费用最少的最优航线为：
到北京的最优航线为：上海 北京 最少费用为：680
到乌鲁木齐的最优航线为：上海 北京 乌鲁木齐 最少费用为：1830
到西安的最优航线为：上海 北京 西安 最少费用为：1610
到广州的最优航线为：上海 北京 西安 广州 最少费用为：2930
到昆明的最优航线为：上海 北京 乌鲁木齐 昆明 最少费用为：3310
到拉萨的最优航线为：上海 北京 乌鲁木齐 昆明 拉萨 最少费用为：4140
到武汉的最优航线为：上海 北京 西安 广州 武汉 最少费用为：3740
是否离开程序? (Y/N) N

最少转机次数:

总时间最短:

```
请输入起始城市: 广州
请选择功能: 1. 最短飞行时间 2. 最少飞行费用 3. 最少转机次数 4. 总时间最少 4
从广州到其他城市总时间最短的最优航线为:
到北京的最优航线为: 广州 西安 北京 最短时间为: 4小时0分钟=240分钟
到上海的最优航线为: 广州 西安 北京 上海 最短时间为: 7小时10分钟=430分钟
到乌鲁木齐的最优航线为: 广州 西安 北京 乌鲁木齐 最短时间为: 23小时40分钟=1420分钟
到西安的最优航线为: 广州 西安 最短时间为: 1小时20分钟=80分钟
到昆明的最优航线为: 广州 武汉 拉萨 昆明 最短时间为: 24小时20分钟=1460分钟
到拉萨的最优航线为: 广州 武汉 拉萨 最短时间为: 6小时30分钟=390分钟
到武汉的最优航线为: 广州 武汉 最短时间为: 1小时40分钟=100分钟
是否离开程序? (Y/N) Y

从西安到其他城市转机次数最少的最优航线为:
到北京的最优航线为: 西安 北京 最少转机次数为: 0
到上海的最优航线为: 西安 北京 上海 最少转机次数为: 1
到乌鲁木齐的最优航线为: 西安 北京 乌鲁木齐 最少转机次数为: 1
到广州的最优航线为: 西安 广州 最少转机次数为: 0
到昆明的最优航线为: 西安 北京 乌鲁木齐 昆明 最少转机次数为: 2
到拉萨的最优航线为: 西安 广州 武汉 拉萨 最少转机次数为: 2
到武汉的最优航线为: 西安 广州 武汉 最少转机次数为: 1
是否离开程序? (Y/N) N
```

六、总结（实验中遇到的问题、取得的经验、感想等）

比较费时间的是总时间最短的算法，另外，对于此题，采用邻接矩阵存储空间利用效率较低，但是可能在总时间最短问题的处理时邻接矩阵较为方便。

提交说明:

提交一个 rar 或 zip 压缩文件，其中包括：实验报告、源程序等，rar 或 zip 文件名为学生学号和姓名。

例如，0801012345 李明.rar 或 0801012345 李明.zip，其文件结构为：

...\0801012345 李明\test1\实验报告 1（0801012345 李明）.DOC

...\0801012345 李明\test1\Software\（所有源程序、工程文件等）

发送到: ds_bnu_homework@163.com

邮件主题内容: 学号_姓名_实验 n, 例如, 0801012345_李明_实验 1