# 交通查询系统设计

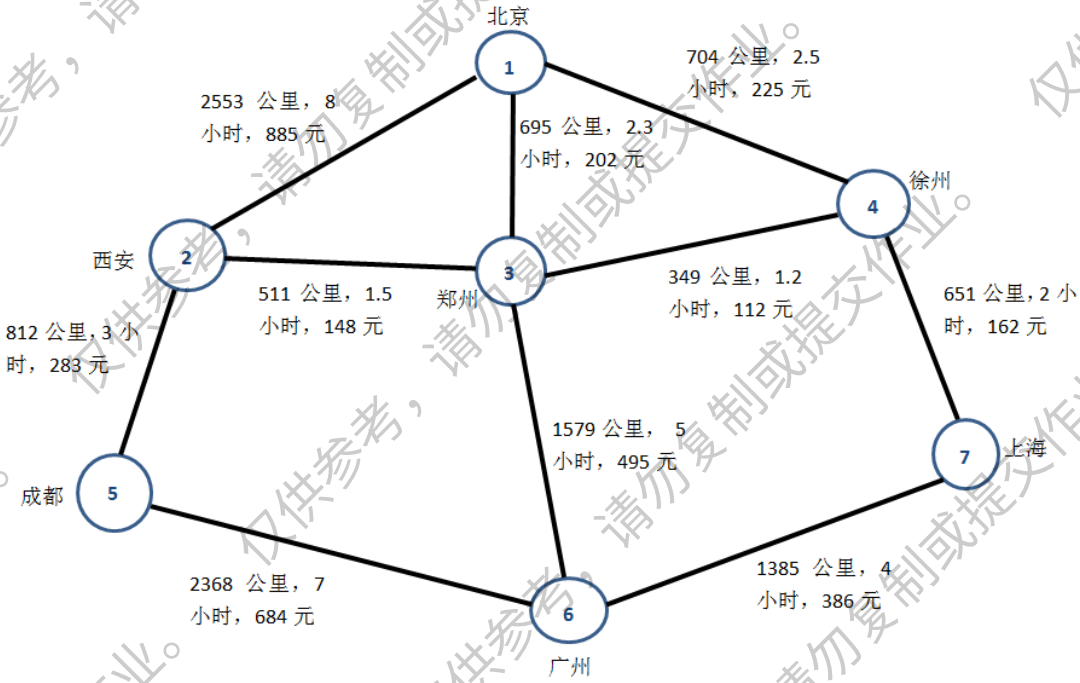
## 一、 题目内容

**[问题描述]**

今天铁路交通网络非常发达，人们在出差、旅游时，不仅关注交通费用，还关注里程和时间。请按照下图设计一个交通查询系统，能够满足旅客查询从任一个城市到另一个城市的最短里程、最低花费、最短时间、最少中转次数等问题。

**[基本要求]**

设计合适的数据结构和算法编写程序完成上述功能，并具有查询界面，能够按照下拉菜单选项进行选择查询。




## 二、解决的思路和算法思想

为了“满足旅客查询从任一个城市到另一个城市的最短里程、最低花费、最短时间、最少中转次数等问题”的要求，因为每个城市之间里程、时间、花费是呈线性的，三个问题可以归结为一个最短路径问题。可以采用 **Floyed** 算法计算多源最短路径，可以达到一次调用即可算出所有点之间的最短路径。虽然 **Floyed**

算法时间复杂度达到$O(n^3)$，但是与 **Dijkstra** 算法对比，其不需要每次计算，在查询系统中显得更加便捷，一次计算后面就无需等待。我的思路是将距离、耗时、花费、边长（固定为 1）分别当做权值调用 **Floyed()**得到图类中的 Dist[][]与 Path[][]。运用 **FindPath()**递归打印路径得到的最优路径，再从 Dist[][]矩阵中得到最优路径下的最小权值。

# 三、 选择或设计的数据结构说明

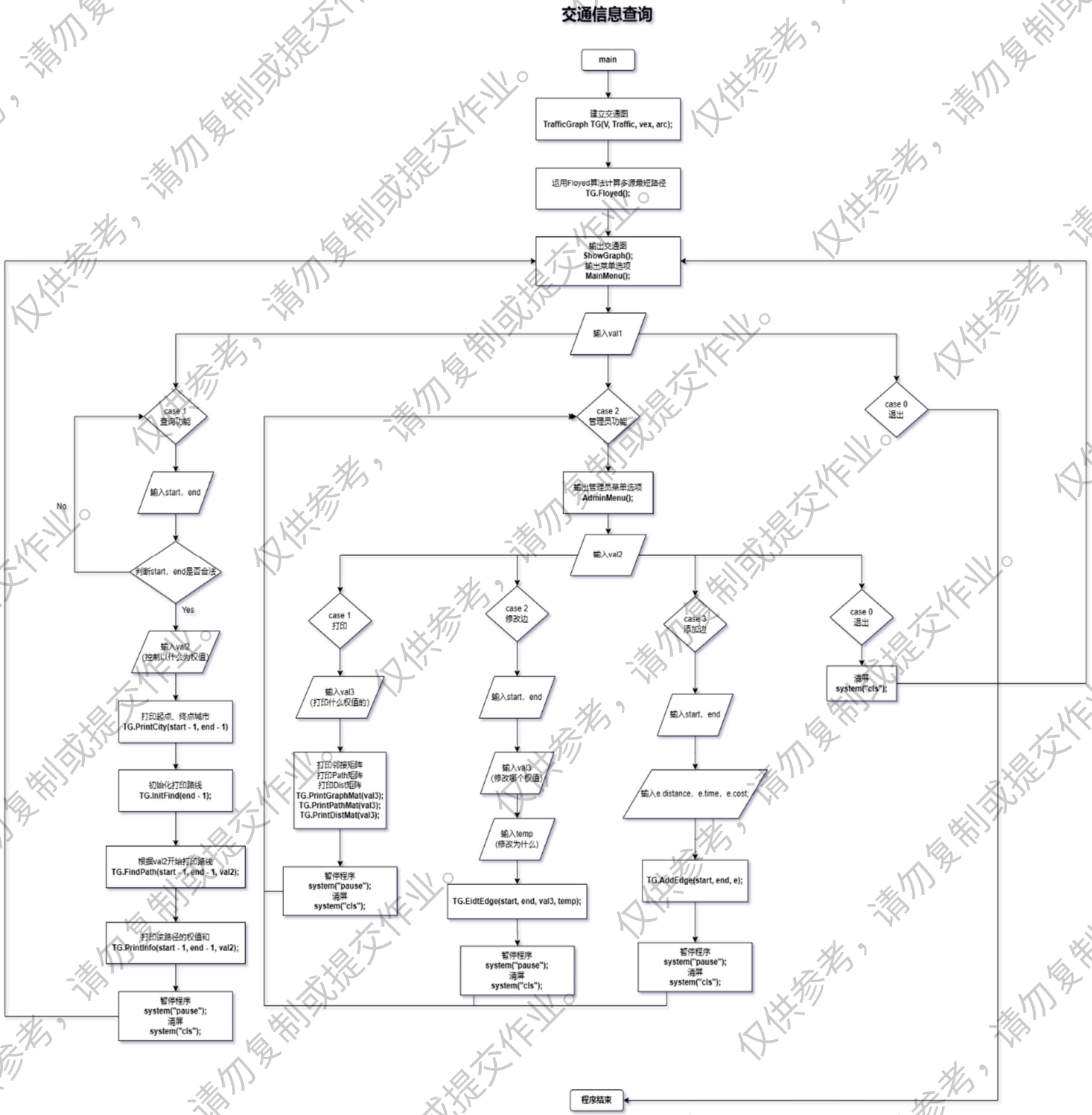
首先分析题目的要求，题干给定的是一幅交通图，图的基本存储结构有邻接矩阵法或者邻接表法。本题选用邻接矩阵法存储交通图，我设计了一个图类 **TrafficGraph**，包含**顶点信息数组 VertexType Vex[]**（存储城市名）、**邻接矩阵 EdgeType Edge[][]**、**顶点数 int VexNum**、**边数 int ArcNum**，以及 **EdgeType Dist[][]矩阵**、**int Path[][]矩阵**。

其次题干中所给的图中每一条边拥有**距离 distance**、**耗时 time**、**花费 cost**、**边长 len** 三种信息，我设计了一个结构体 **EdgeType** 存储边的信息，并且为了实现最少中转次数的计算添加了一个数据成员**边长 len**。

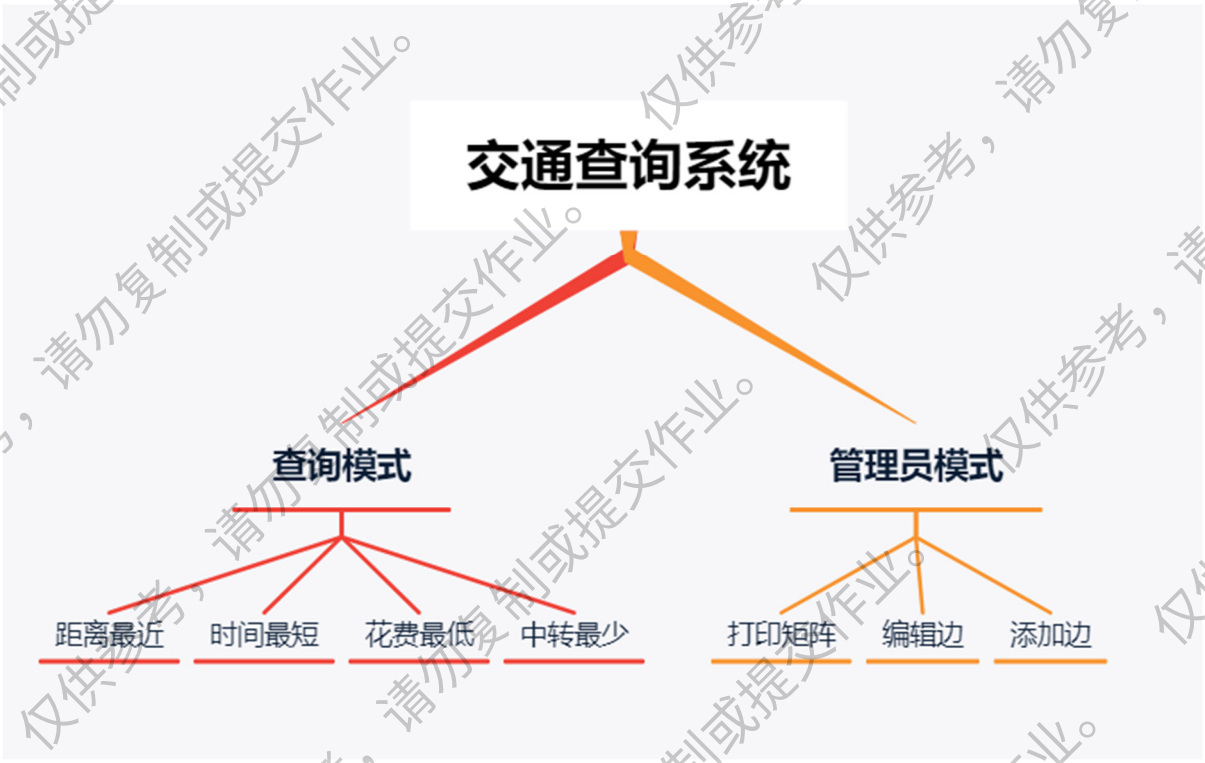
再者，为了方便构造邻接矩阵，我设计一个结构体 **GraphInfo** 来输入图的每一条边，**GraphInfo** 包含了**起点 Vex1**、**终点 Vex2** 以及**边权值 EdgeType Weight**。初始化一个 **GraphInfo** 类型的数组放入每一条边及其信息，然后图类的构造函数会构造出图的邻接矩阵。

## 四、主程序的流程图和主要功能模块流程图

- 主程序



- 功能模块




## 五、 程序源代码

```
// TrafficGraph.h
#pragma once
#include <string>
using namespace std;
#define INFINITY 888880          // 无穷
typedef string VertexType;       // 顶点的数据类型

//////////////////////////////图结构定义//////////////////////////////
// 带权图边权值的数据类型
struct EdgeType
{
    int distance = INFINITY;     // 路线的距离
    double time = INFINITY;      // 路线所需要的时间
    int cost = INFINITY;         // 路线的花费
    int len = INFINITY;          // 路径长
};

// 为了便捷使用，定义一个结构体来输入图
struct GraphInfo
{
    int Vex1;            // 起点
```

```
    int Vex2;           // 终点
    EdgeType Weight;    // 权值
};

//  邻接矩阵储存交通图
class TrafficGraph
{
public:
    TrafficGraph(VertexType V[], GraphInfo GI[], int n, int e);
    ~TrafficGraph();
    void EidtEdge(int start, int end, int flag, int weight);// 编辑现有边
    void AddEdge(int start, int end, EdgeType e);   // 增加新边
    void PrintGraphMat(int flag);                   // 打印邻接矩阵
    void PrintDistMat(int flag);                    // 打印 Dist 矩阵
    void PrintPathMat(int flag);                    // 打印 Path 矩阵
    void PrintInfo(int start, int end, int flag);   // 打印带权路径和
    void PrintCity(int v1, int v2);                 // 打印起点终点城市
    void Floyed();                                  // Floyed 计算多源最短路径
    void InitFind(int end);                         // 初始化查找路径
    void FindPath(int start, int end, int flag);    // 打印输出最短路径

private:
    VertexType* Vex;        // 顶点表
    EdgeType** Edge;        // 邻接矩阵，边表
    EdgeType** Dist;        // Floyed 算法辅助数组，记录带权路径长度
    EdgeType** Path;        // Floyed 算法路径数组，记录算法产生的路径（中转点）
    int VexNum;             // 当前顶点数
    int ArcNum;             // 当前弧数
};

//////////////////////////显示菜单///////////////////////////
void ShowGraph();      // 显示交通图菜单
void LowcostMenu();    // 显示选择权重菜单
void MainMenu();       // 主菜单
void AdminMenu();      // 管理员菜单
```

```
// TrafficGraph.cpp
#include "TrafficGraph.h"
#include <iostream>
using namespace std;

//////////////////////////图操作///////////////////////////////
// 交通图构造函数
TrafficGraph::TrafficGraph(VertexType V[], GraphInfo GI[], int n, int e)
```

```
9     {
10        // 点数、边数
11        VexNum = n;
12        ArcNum = e;
13        // 动态数组初始化
14        Vex = new VertexType[VexNum];
15        Edge = new EdgeType * [VexNum];
16        Path = new EdgeType * [VexNum];
17        Dist = new EdgeType * [VexNum];
18        for (int i = 0; i < VexNum; i++)
19        {
20            Edge[i] = new EdgeType[VexNum];
21            Path[i] = new EdgeType[VexNum];
22            Dist[i] = new EdgeType[VexNum];
23        }
24        // 复制输入的边信息
25        for (int i = 0; i < ArcNum; i++)
26        {
27            if (i < VexNum)
28                Vex[i] = V[i];
29            GI[i].Weight.len = 1;
30            Edge[GI[i].Vex1 - 1][GI[i].Vex2 - 1] = GI[i].Weight;
31            Edge[GI[i].Vex2 - 1][GI[i].Vex1 - 1] = GI[i].Weight;
32        }
33    }
34
35    // 交通图析构函数
36    TrafficGraph::~TrafficGraph()
37    {
38        VexNum = 0;
39        ArcNum = 0;
40        for (int i = 0; i < VexNum; i++)
41        {
42            delete[]Edge[i];
43            delete[]Dist[i];
44        }
45        delete[]Vex;
46        delete[]Dist;
47        delete[]Edge;
48        delete[]Path;
49    }
50
51    // 编辑现有边
52    void TrafficGraph::EidtEdge(int start, int end, int flag, int weight)
```

```
53      {
54          switch (flag)
55          {
56          case 1:
57              Edge[start][end].distance = weight;
58              return;
59          case 2:
60              Edge[start][end].time = weight;
61              return;
62          case 3:
63              Edge[start][end].cost = weight;
64              return;
65          case 4:
66              Edge[start][end].len = weight;
67              return;
68          }
69      }
70
71      // 增加新边
72      void TrafficGraph::AddEdge(int start, int end, EdgeType e)
73      {
74          Edge[start][end] = e;
75      }
76
77      // 打印起点终点城市
78      void TrafficGraph::PrintCity(int v1, int v2)
79      {
80          cout << Vex[v1] << " -> " << Vex[v2];
81      }
82
83      // 打印邻接矩阵（flag 选择以什么为权值：1 距离，2 时间，3 花费，4 中转）
84      void TrafficGraph::PrintGraphMat(int flag)
85      {
86          cout << "\n\t\t\tGraphMat" << endl;
87          switch (flag)
88          {
89          case 1:
90              cout << "\t\t\tdistance" << endl;
91              for (int i = 0; i < VexNum; i++)
92              {
93                  cout << "\t\t\t";
94                  for (int k = 0; k < VexNum; k++)
95                  {
96                      if (Edge[i][k].distance == INFINITY)
```

```
97                      cout << "∞\t";
98                  else
99                      cout << Edge[i][k].distance << "\t";
100             }
101             cout << endl;
102         }
103         return;
104     case 2:
105         cout << "\t\t\ttime" << endl;
106         for (int i = 0; i < VexNum; i++)
107         {
108             cout << "\t\t\t";
109             for (int k = 0; k < VexNum; k++)
110             {
111                 if (Edge[i][k].time == INFINITY)
112                     cout << "∞\t";
113                 else
114                     cout << Edge[i][k].time << "\t";
115             }
116             cout << endl;
117         }
118         return;
119     case 3:
120         cout << "\t\t\tcost" << endl;
121         for (int i = 0; i < VexNum; i++)
122         {
123             cout << "\t\t\t";
124             for (int k = 0; k < VexNum; k++)
125             {
126                 if (Edge[i][k].cost == INFINITY)
127                     cout << "∞\t";
128                 else
129                     cout << Edge[i][k].cost << "\t";
130             }
131             cout << endl;
132         }
133         return;
134     case 4:
135         cout << "\t\t\t 最少中转" << endl;
136         for (int i = 0; i < VexNum; i++)
137         {
138             cout << "\t\t\t";
139             for (int k = 0; k < VexNum; k++)
140             {
```

```
                    if (Edge[i][k].len == INFINITY)
                        cout << "∞\t";
                    else
                        cout << Edge[i][k].len << "\t";
                }
                cout << endl;
            }
            return;
        }
    }

    // 打印 Dist 矩阵（flag 选择以什么为权值：1 距离，2 时间，3 花费，4 中转）
    void TrafficGraph::PrintDistMat(int flag)
    {
        cout << "\n\t\t\tDistMat" << endl;
        switch (flag)
        {
        case 1:
            cout << "\t\t\tdistance" << endl;
            for (int i = 0; i < VexNum; i++)
            {
                cout << "\t\t\t";
                for (int k = 0; k < VexNum; k++)
                {
                    if (Dist[i][k].distance == INFINITY)
                        cout << "∞\t";
                    else
                        cout << Dist[i][k].distance << "\t";
                }
                cout << endl;
            }
            return;
        case 2:
            cout << "\t\t\ttime" << endl;
            for (int i = 0; i < VexNum; i++)
            {
                cout << "\t\t\t";
                for (int k = 0; k < VexNum; k++)
                {
                    if (Dist[i][k].time == INFINITY)
                        cout << "∞\t";
                    else
                        cout << Dist[i][k].time << "\t";
                }
```

```
185             cout << endl;
186         }
187         return;
188     case 3:
189         cout << "\t\t\tcost" << endl;
190         for (int i = 0; i < VexNum; i++)
191         {
192             cout << "\t\t\t";
193             for (int k = 0; k < VexNum; k++)
194             {
195                 if (Dist[i][k].cost == INFINITY)
196                     cout << "∞\t";
197                 else
198                     cout << Dist[i][k].cost << "\t";
199             }
200             cout << endl;
201         }
202         return;
203     case 4:
204         cout << "\t\t\t最少中转" << endl;
205         for (int i = 0; i < VexNum; i++)
206         {
207             cout << "\t\t\t";
208             for (int k = 0; k < VexNum; k++)
209             {
210                 if (Dist[i][k].len == INFINITY)
211                     cout << "∞\t";
212                 else
213                     cout << Dist[i][k].len << "\t";
214             }
215             cout << endl;
216         }
217         return;
218     }
219 }
220
221 // 打印 Path 矩阵（flag 选择以什么为权值：1 距离，2 时间，3 花费，4 中转）
222 void TrafficGraph::PrintPathMat(int flag)
223 {
224     cout << "\n\t\t\tPathMat" << endl;
225     switch (flag)
226     {
227     case 1:
228         cout << "\t\t\tdistance" << endl;
```

```
229             for (int i = 0; i < VexNum; i++)
230             {
231                 cout << "\t\t\t";
232                 for (int k = 0; k < VexNum; k++)
233                 {
234                     if (Path[i][k].distance == INFINITY)
235                         cout << "∞\t";
236                     else
237                         cout << Path[i][k].distance << "\t";
238                 }
239                 cout << endl;
240             }
241             return;
242         case 2:
243             cout << "\t\t\ttime" << endl;
244             for (int i = 0; i < VexNum; i++)
245             {
246                 cout << "\t\t\t";
247                 for (int k = 0; k < VexNum; k++)
248                 {
249                     if (Path[i][k].time == INFINITY)
250                         cout << "∞\t";
251                     else
252                         cout << Path[i][k].time << "\t";
253                 }
254                 cout << endl;
255             }
256             return;
257         case 3:
258             cout << "\t\t\tcost" << endl;
259             for (int i = 0; i < VexNum; i++)
260             {
261                 cout << "\t\t\t";
262                 for (int k = 0; k < VexNum; k++)
263                 {
264                     if (Path[i][k].cost == INFINITY)
265                         cout << "∞\t";
266                     else
267                         cout << Path[i][k].cost << "\t";
268                 }
269                 cout << endl;
270             }
271             return;
272         case 4:
```

```
273             cout << "\t\t\t 最少中转" << endl;
274             for (int i = 0; i < VexNum; i++)
275             {
276                 cout << "\t\t\t";
277                 for (int k = 0; k < VexNum; k++)
278                 {
279                     if (Path[i][k].len == INFINITY)
280                         cout << "∞\t";
281                     else
282                         cout << Path[i][k].len << "\t";
283                 }
284                 cout << endl;
285             }
286             return;
287         }
288     }
289
290     // 打印带权路径和
291     void TrafficGraph::PrintInfo(int start, int end, int flag)
292     {
293         switch (flag)
294         {
295         case 1:
296             cout << " 距离仅为 " << Dist[start][end].distance << " 公里" << endl;
297             break;
298         case 2:
299             cout << " 花费仅为 " << Dist[start][end].cost << " 元" << endl;
300             break;
301         case 3:
302             cout << " 用时仅为 " << Dist[start][end].time << " 小时" << endl;
303             break;
304         case 4:
305             cout << " 中转次数仅有 " << Dist[start][end].len - 1 << " 次" << endl;
306             break;
307         default:
308             return;
309         }
310     }
311
312     // floyed 算法计算多源最短路径
313     void TrafficGraph::Floyed()
314     {
315         for (int i = 0; i < VexNum; i++)
316         {
```

```
        for (int j = 0; j < VexNum; j++)
        {
            Dist[i][j] = Edge[i][j];
            Path[i][j].cost = -1;
            Path[i][j].distance = -1;
            Path[i][j].time = -1;
            Path[i][j].len = -1;
        }
    }
    // 外层循环表示添加一个顶点 Vi 作为中转
    for (int i = 0; i < VexNum; i++)
    {
        // 内层双循环用来遍历整个矩阵
        for (int j = 0; j < VexNum; j++)
        {
            for (int k = 0; k < VexNum; k++)
            {
                // 当 Dist 里的值已经不是最短路径时
                // 更新为通过中转点 Vi 的路径长
                // ！！！这里体现的是迭代的思想
                // Dist 里的值永远保持最优（这里和最小生成树的算法非常像）
                if (Dist[j][k].distance>Dist[j][i].distance+Dist[i][k].distance)
                {
                    // 更新 Dist[][]~
                    Dist[j][k].distance=Dist[j][i].distance+Dist[i][k].distance;
                    // 在 path 里说明是通过谁中转的
                    Path[j][k].distance = i;
                }
                if (Dist[j][k].cost>Dist[j][i].cost+Dist[i][k].cost)
                {
                    Dist[j][k].cost = Dist[j][i].cost + Dist[i][k].cost;
                    Path[j][k].cost = i;
                }
                if (Dist[j][k].time > Dist[j][i].time + Dist[i][k].time)
                {
                    Dist[j][k].time = Dist[j][i].time + Dist[i][k].time;
                    Path[j][k].time = i;
                }
                if (Dist[j][k].len > Dist[j][i].len + Dist[i][k].len)
                {
                    Dist[j][k].len = Edge[j][i].len + Edge[i][k].len;
                    Path[j][k].len = i;
                }
            }
```

```
361             }
362         }
363     }
364
365     // 记录程序一开始的终点（方便输出）
366     int destination = -1;
367     // 记录上一次输出的节点编号，防止重复输出
368     int out = -1;
369     // 初始化查找路径
370     void TrafficGraph::InitFind(int end)
371     {
372         destination = end;
373         out = -1;
374     }
375
376     // 打印输出最短路径（flag 为考虑什么因素）
377     void TrafficGraph::FindPath(int start, int end, int flag)
378     {
379         // 记录程序一开始的终点（方便输出）
380         static int destination = end;
381         // 记录上一次输出的节点编号，防止重复输出
382         static int out = -1;
383         // 中转点
384         int mid = -1;
385         switch (flag)
386         {
387         case 0:
388             return;
389         case 1:
390             mid = Path[start][end].distance;
391             break;
392         case 2:
393             mid = Path[start][end].cost;
394             break;
395         case 3:
396             mid = Path[start][end].time;
397         case 4:
398             mid = Path[start][end].len;
399             break;
400         default:
401             cout << "\t\t\t 输入错误！" << endl;
402             return;
403         }
404         // 如果起点终点没有中间点了，就输出起点
```

```
405         if (mid == -1)
406         {
407             // 要看看起点是不是已经输出了，是就不用输出
408             if (out != start)
409                 cout << Vex[start] << " -> ";
410         }
411         // 还有中间点就递归
412         else
413         {
414             FindPath(start, mid, flag);
415             FindPath(mid, end, flag);
416         }
417         // 要看看终点是不是已经输出了，是就不用输出
418         if (out != end)
419         {
420             cout << Vex[end];
421             // 当没有到达真正终点时打印->
422             if (destination != end)
423                 cout << " -> ";
424             out = end;
425         }
426     }
427
428     ////////////////////////////////////显示菜单////////////////////////////////////
429     // 显示交通图菜单
430     void ShowGraph()
431     {
432         cout << "\t\t\t*************************************************************" << endl;
433         cout << "\t\t\t*\t\t  欢迎使用交通查询系统!                    *" << endl;
434         cout << "\t\t\t*                                                           *\n";
435         cout << "\t\t\t*\t\t\t 北京(1)                        *" << endl;
436         cout << "\t\t\t*                                                           *\n";
437         cout << "\t\t\t*                                                           *\n";
438         cout << "\t\t\t*\t 西安(2)\t\t\t\t 徐州(4)          *" << endl;
439         cout << "\t\t\t*                                                           *\n";
440         cout << "\t\t\t*                                                           *\n";
441         cout << "\t\t\t*\t\t\t 郑州(3)                        *" << endl;
442         cout << "\t\t\t*                                                           *\n";
443         cout << "\t\t\t*                                                           *\n";
444         cout << "\t\t\t*\t 成都(5)\t\t\t\t 上海(7)          *" << endl;
445         cout << "\t\t\t*                                                           *\n";
446         cout << "\t\t\t*                                                           *\n";
447         cout << "\t\t\t*\t\t\t 广州(6)                        *" << endl;
```

```
448         cout << "\t\t\t*************************************************************" <<
    endl;
449     }
450     // 显示选择权重菜单
451     void LowcostMenu()
452     {
453         cout << endl;
454         cout << "\t\t\t*--------------------您最关心的是--------------------*\n";
455         cout << "\t\t\t*   1) 最短里程                                        *\n";
456         cout << "\t\t\t*   2) 最低花费                                        *\n";
457         cout << "\t\t\t*   3) 最短时间                                        *\n";
458         cout << "\t\t\t*   4) 最少中转                                        *\n";
459         cout << "\t\t\t*                                                      *\n";
460         cout << "\t\t\t*   0) 返回                                            *\n";
461         cout << "\t\t\t*------------------------------------------------------*\n";
462         cout << "\t\t\t 请输入: ";
463     }
464     // 主菜单
465     void MainMenu()
466     {
467         cout << endl;
468         cout << "\t\t\t*----------------------主菜单----------------------*\n";
469         cout << "\t\t\t*   1) 交通查询                                        *\n";
470         cout << "\t\t\t*   2) 管理员模式                                      *\n";
471         cout << "\t\t\t*   0) 退出                                            *\n";
472         cout << "\t\t\t*------------------------------------------------------*\n";
473         cout << "\t\t\t 请输入: ";
474     }
475     // 管理员菜单
476     void AdminMenu()
477     {
478         cout << endl;
479         cout << "\t\t\t*------------------这是一个奇怪的菜单------------------*\n";
480         cout << "\t\t\t*   1) 打印                                            *\n";
481         cout << "\t\t\t*   2) 修改边                                          *\n";
482         cout << "\t\t\t*   3) 添加边                                          *\n";
483         cout << "\t\t\t*   0) 退出                                            *\n";
484         cout << "\t\t\t*------------------------------------------------------*\n";
485         cout << "\t\t\t 请输入-^_^-: ";
486     }
```

```
1   // main.cpp
2   #include <iostream>
3   #include "TrafficGraph.h"
```

```cpp
using namespace std;

int main()
{
    EdgeType e;
    // val~为菜单选项
    int val1 = 1, val2 = 1, val3 = 1;
    // 起点终点城市编号
    int start = -1, end = -1;
    // 建立交通图
    int arc = 10, vex = 7;
    GraphInfo Traffic[10] =
    {   {1,2,{2553,8,885}},
        {1,4,{704,2.5,225}},
        {1,3,{696,2.3,202}},
        {2,3,{511,1.5,148}},
        {3,4,{349,1.2,112}},
        {2,5,{812,3,283}},
        {4,7,{651,2,162}},
        {5,6,{2368,7,684}},
        {6,7,{1385,4,386}},
        {3,6,{1579,5,495}} };
    VertexType V[7] = { "北京","西安","郑州","徐州","成都","广州","上海" };
    TrafficGraph TG(V, Traffic, vex, arc);
    TG.Floyed();
    while (val1)
    {
        ShowGraph();
        MainMenu();
        cin >> val1;
        switch (val1)
        {
        case 0: // 退出
            cout << "\n\t\t\t\t\t 程序退出，欢迎下次使用！\n";
            break;
        case 1: // 交通查询
            query:
            cout << "\n\t\t\t----------------------\n";
            cout << "\t\t\t 请输入您的始发地编号：";
            cin >> start;
            cout << "\t\t\t 请输入您的目的地编号：";
            cin >> end;
            // 检查输入
            if (start == end || start < 0 || start>7 || end < 0 || end>7)
```

```
            {
                cout << "\t\t\t 输入错误！！！" << endl;
                goto query;
            }
            LowcostMenu();
            cin >> val2;
            // val2==0 返回
            if (!val2)
                goto query;
            cout << "\n\t\t\t-----------------------------\n";
            cout << "\t\t\t 已为您选择";
            TG.PrintCity(start - 1, end - 1);
            cout << "最佳路线\n\n\t\t\t";
             TG.InitFind(end - 1);
            TG.FindPath(start - 1, end - 1, val2);
            TG.PrintInfo(start - 1, end - 1, val2);
            system("pause");
            system("cls");
            break;
        case 2: // 管理员模式
            admin:
            system("cls");
            AdminMenu();
            cin >> val2;
            switch (val2)
            {
            case 1:
                cout << "\n\t\t\t1 距离，2 时间，3 花费，4 中转" << endl;
                cout << "\t\t\t 请选择打印的权值：";
                cin >> val3;
                TG.PrintGraphMat(val3);
                TG.PrintPathMat(val3);
                TG.PrintDistMat(val3);
                system("pause");
                system("cls");
                break;
            // 修改边
            case 2:
                int temp;
                cout << "\n\t\t\t 修改模式：\n" << endl;
                cout << "\t\t\t 请输入起点：";
                cin >> start;
                cout << "\t\t\t 请输入终点：";
                cin >> end;
```

```
92                  cout << "\n\t\t\t1 距离，2 时间，3 花费，4 中转\n";
93                  cout << "\t\t\t 您要修改什么：";
94                  cin >> val3;
95                  cout << "\t\t\t 修改为：";
96                  cin >> temp;
97                  TG.EidtEdge(start, end, val3, temp);
98                  system("pause");
99                  system("cls");
100                 break;
101             case 3:
102                 cout << "\n\t\t\t 添加模式：\n" << endl;
103                 cout << "\t\t\t 请输入起点：";
104                 cin >> start;
105                 cout << "\t\t\t 请输入终点：";
106                 cin >> end;
107                 cout << "\t\t\t 请输入距离，时间，花费：";
108                 cin >> e.distance >> e.time >> e.cost;
109                 TG.AddEdge(start, end, e);
110                 system("pause");
111                 system("cls");
112                 break;
113             case 0:
114                 system("cls");
115                 break;
116             default:
117                 goto admin;
118             }
119             break;
120         default:
121             break;
122         }
123     }
124     return 0;
125 }
126 // 程序结束,共 654 行
```

# 六、运行结果截图及分析说明

## 1) 程序主界面与主菜单

采用与题干相似的城市位置排布用文字可视化交通图，并且给出主菜单并且为了美观，文字均使用\t 制表符推进至窗口中央

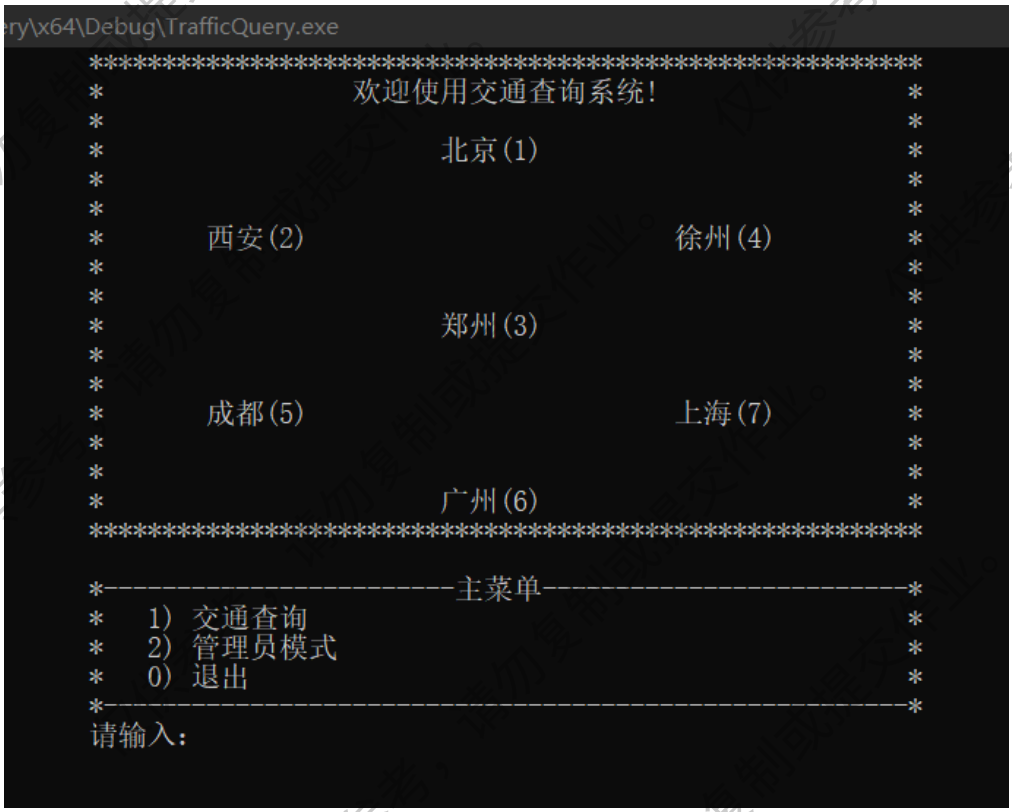



图 1　　主界面

## 2) 查询功能：最短里程

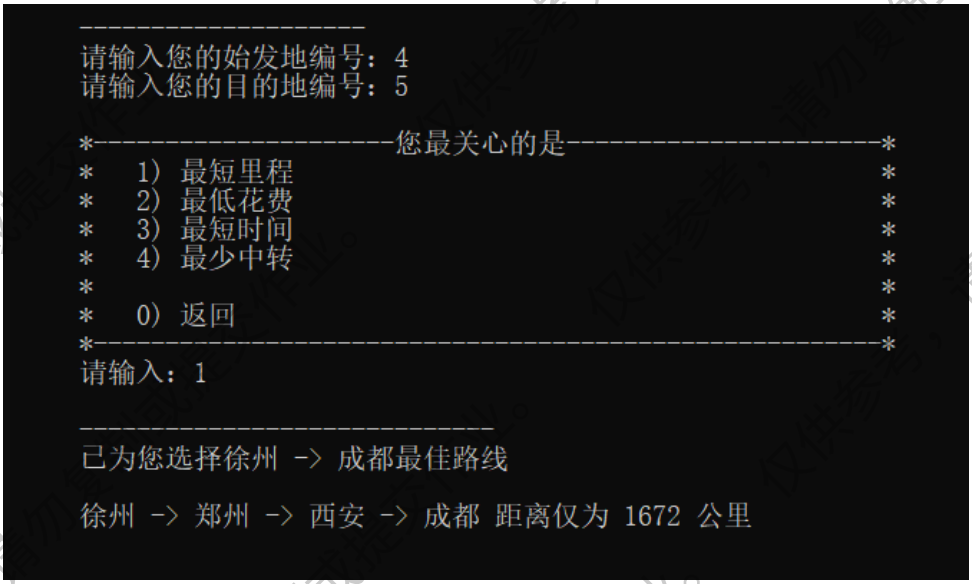



图 2　　查询功能演示 1

### 3) 查询功能：最低花费

程序输出最佳路线后会给出相应的花费

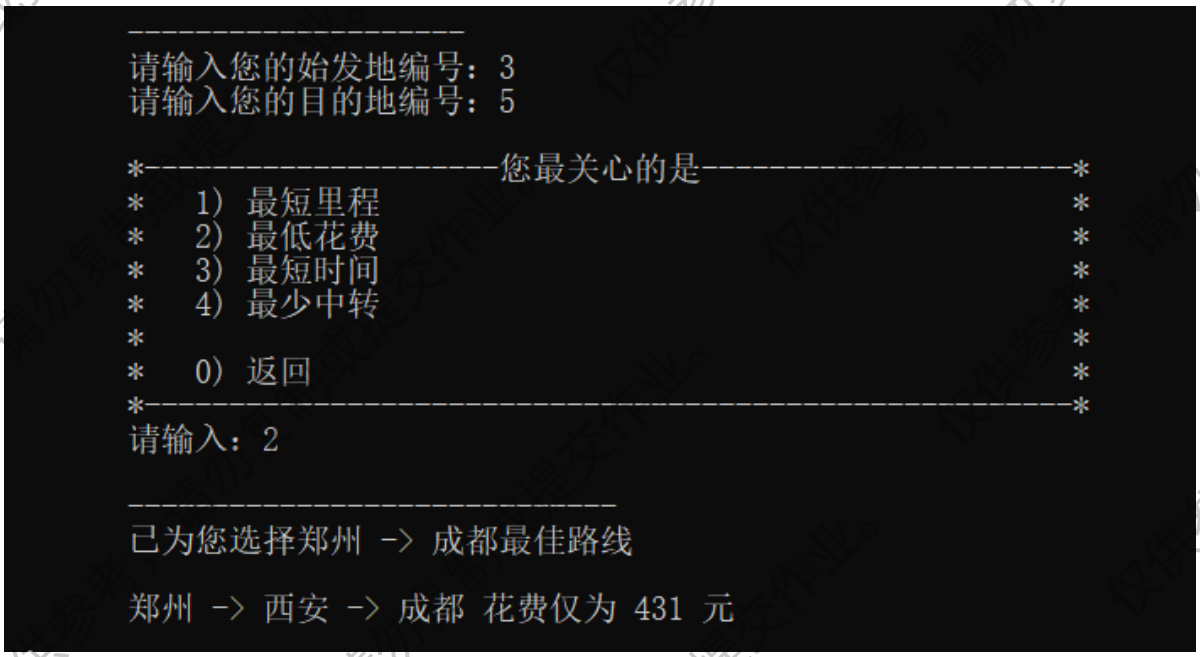



图 3　　查询功能演示 2

### 4) 查询功能：最短时间

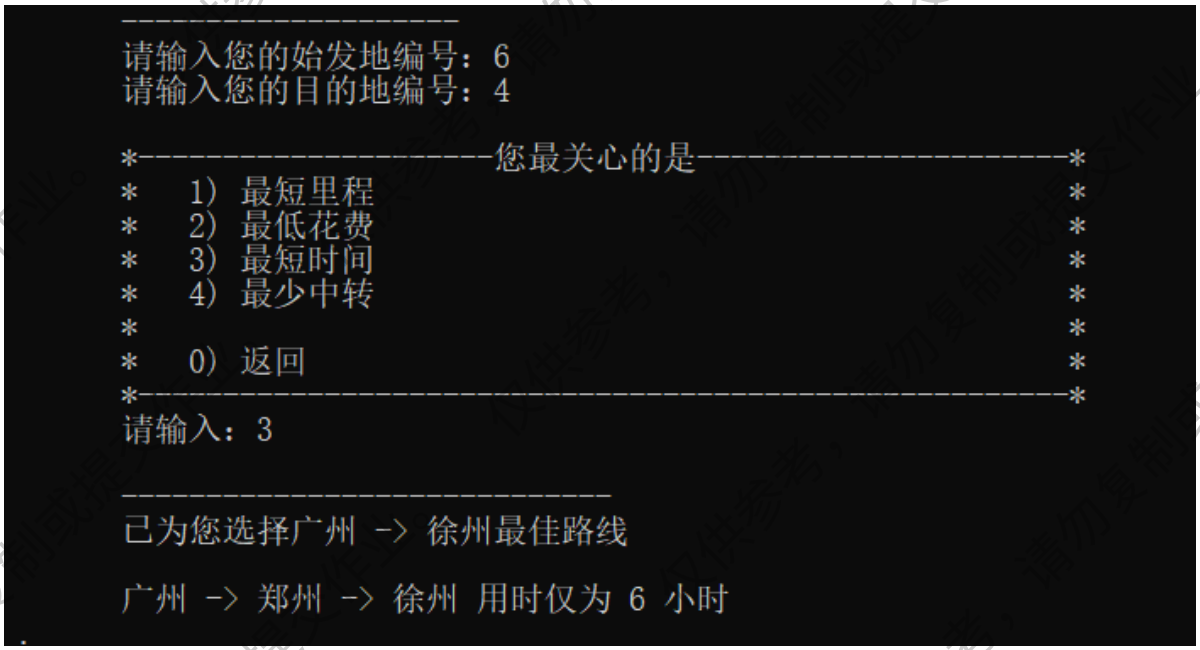



图 4　　查询功能演示 3

**5) 查询功能：最少中转**

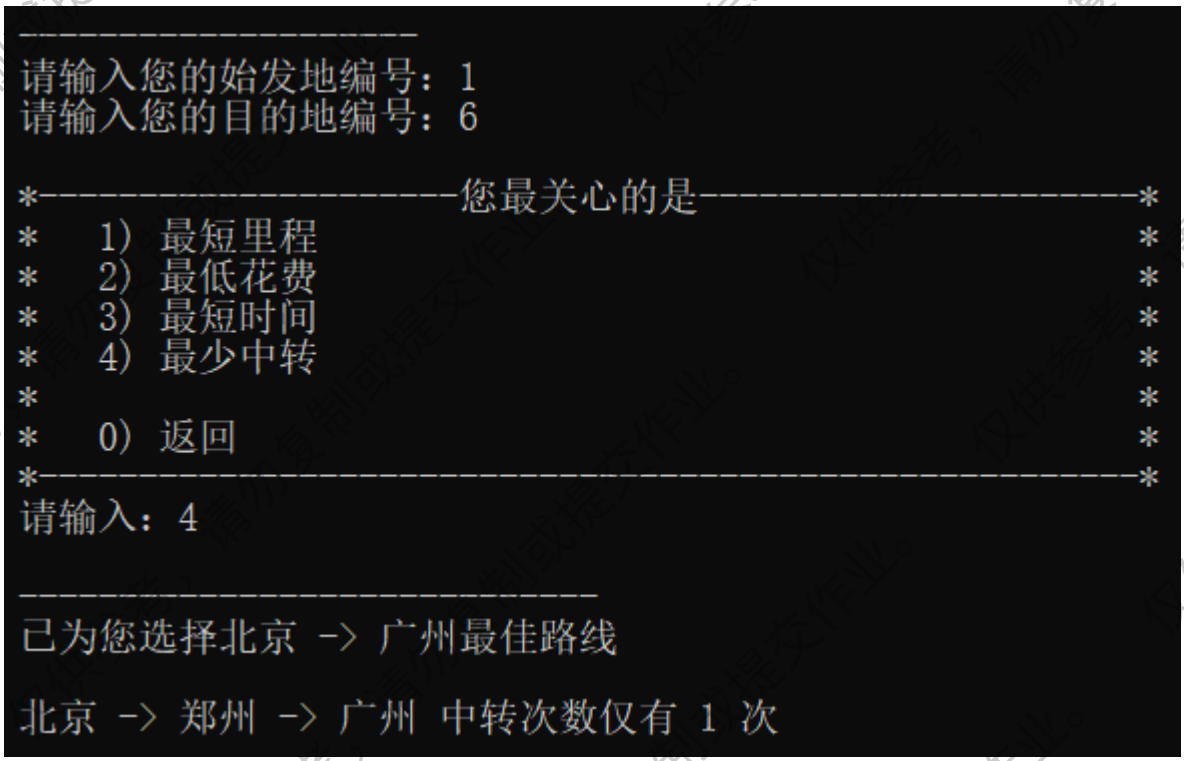



图 5　　查询功能演示 4

**6) 管理员模式菜单**

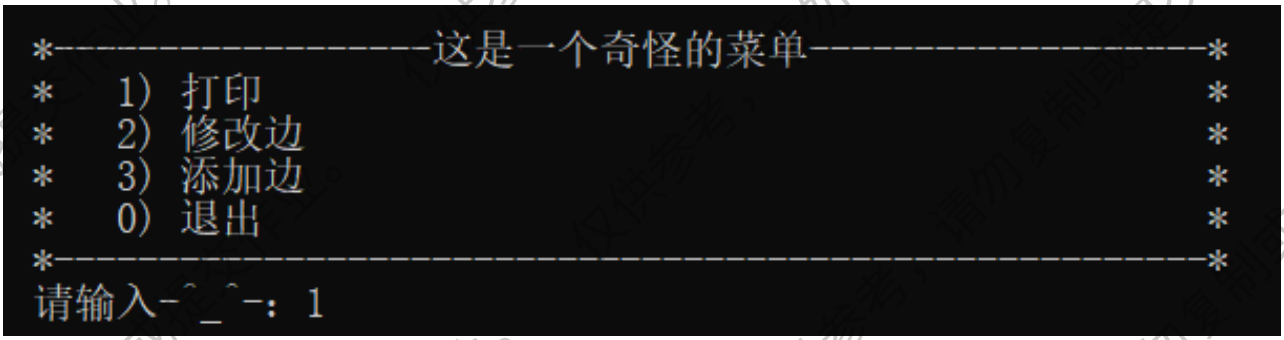



图 6　　管理员模式菜单

### 7) 管理员模式：打印

```
1距离，2时间，3花费，4中转
请选择打印的权值：1

GraphMat
distance
∞       2553    696     704     ∞       ∞       ∞
2553    ∞       511     ∞       812     ∞       ∞
696     511     ∞       349     ∞       1579    ∞
704     ∞       349     ∞       ∞       ∞       651
∞       812     ∞       ∞       ∞       2368    ∞
∞       ∞       1579    ∞       2368    ∞       1385
∞       ∞       ∞       651     ∞       1385    ∞

PathMat
distance
2       2       -1      -1      2       2       3
2       2       -1      2       -1      2       3
-1      -1      3       -1      1       -1      3
-1      2       -1      2       2       2       -1
2       -1      1       2       1       -1      3
2       2       -1      2       -1      6       -1
3       3       3       -1      3       -1      3

DistMat
distance
1392    1207    696     704     2019    2275    1355
1207    1022    511     860     812     2090    1511
696     511     698     349     1323    1579    1000
704     860     349     698     1672    1928    651
2019    812     1323    1672    1624    2368    2323
2275    2090    1579    1928    2368    2770    1385
1355    1511    1000    651     2323    1385    1302
```

图 7　　管理员模式打印演示

### 8) 管理员模式：编辑

```
*------------------这是一个奇怪的菜单-------------------*
*    1) 打印                                           *
*    2) 修改边                                         *
*    3) 添加边                                         *
*    0) 退出                                           *
*------------------------------------------------------*
请输入-^_^-：2

修改模式：

请输入起点：1
请输入终点：2

1距离，2时间，3花费，4中转
您要修改什么：1
```

图 8　　管理员模式编辑演示

### 9) 管理员模式：添加

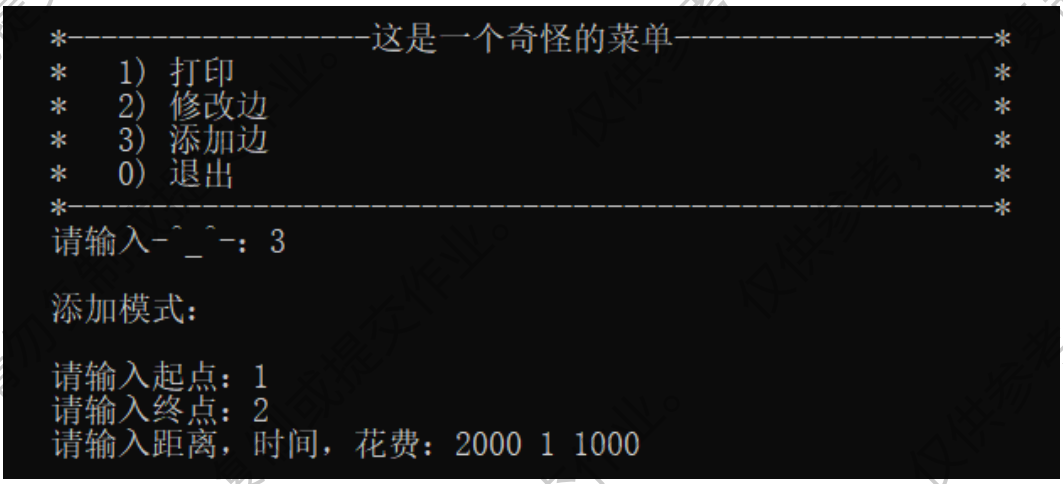



图 9　　管理员模式添加演示

### 10) 程序暂停、清屏功能演示（动图）


图 10　　程序暂停、清屏功能演示（**GIF 动图**）

**11) 程序应对异常输入**

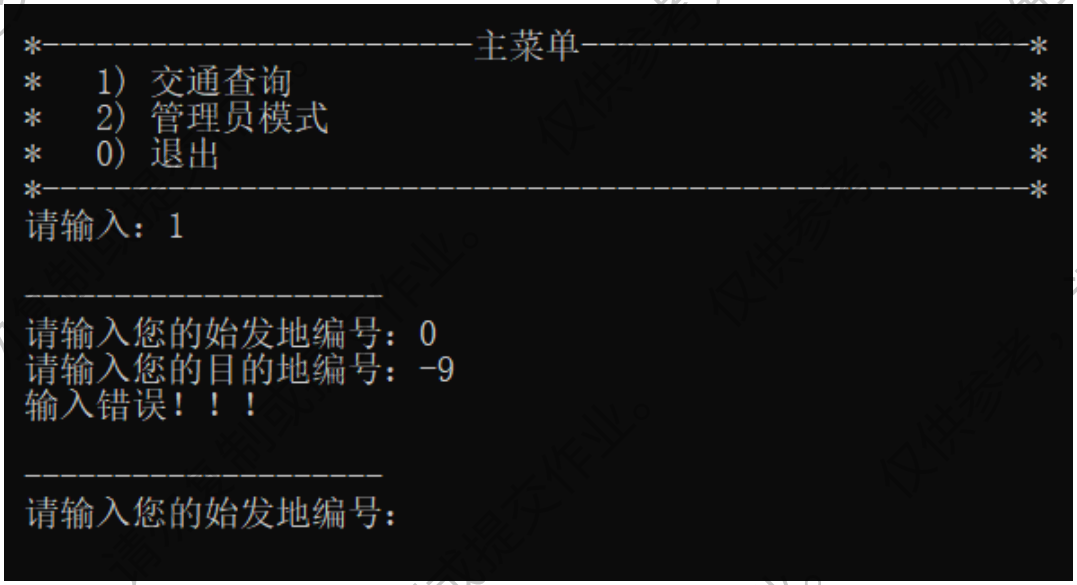



图 11　　程序应对异常输入演示