各模块设计说明

熊宇

工程分为两大部分，头文件（.h）部分和源文件（.cpp）部分。

1. 头文件部分

   该部分由 definition.h 和 functions.h 构成。

① definition.h

```
#pragma once
#include <string>
#include <string.h>
#define NumOfNode 229
#define INF 1000000;

using namespace std;

typedef struct
{
  double time;
  char loc;
}Node;
```

② functions.h

```
#pragma once
#include "definition.h"

void SystemInit(Node * l, double M[][NumOfNode + 2]);
void Examine(Node * l, double M[][NumOfNode + 2]);
double Search(char c);
void GetIns(double cost[][NumOfNode + 2], Node* l, double
M[][NumOfNode + 2], double len[NumOfNode + 2], int visit[NumOfNode
+ 2], int n, int path[][NumOfNode + 2], string OutLoc[][NumOfNode +
2], int timepath[][NumOfNode + 2][NumOfNode + 2]);
```

2. 源文件部分

   该部分由 main.cpp 和 functions.cpp 构成

① main.cpp

```
#define _CRT_SECURE_NOWARNINGS
#include <iostream>
#include "functions.h"
#include "definition.h"
```

```
using namespace std;

double len[NumOfNode + 2];      //表示源点到 i 这个点的距离
int visit[NumOfNode + 2];  //节点是否被访问
int n = NumOfNode + 1; //结点数
Node NodeList[250]; //存储结点的数组
double Mar[NumOfNode + 2][NumOfNode + 2]; //结点的邻接矩阵
double cost[NumOfNode + 2][NumOfNode + 2]; //结点间经过处理后的邻接矩阵
int path[NumOfNode + 2][NumOfNode + 2]; //存储中间结点的数组
string OutLoc[NumOfNode + 2][NumOfNode + 2]; //存储路径的数组
int timepath[NumOfNode + 2][NumOfNode + 2][NumOfNode + 2]; //存储经过的所有结点的数组



int main()
{

  SystemInit(NodeList, Mar);
  //Examine(NodeList, Mar);
  while(1)
      GetIns(cost,NodeList, Mar, len, visit, n, path, OutLoc,timepath);
  system("pause");
  return 0;
}
```

② functions.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cmath>
#include <string.h>
#include <algorithm>
#include <Windows.h>
#include "definition.h"
#include "functions.h"

#define INFINITY 9999

using namespace std;

void SystemInit(Node* l, double M[][NumOfNode + 2])
{
  printf("该旅游系统有十个城市，分别为：A、B、C、D、E、F、G、H、I、J。\n 其中 A、B、C、D 三城风险系数为 0.2，E、F、G 三城风险系数为 0.5，H、I、J 三城风险系数为 0.9。\n");
```

```
   printf("城市间有三种交通方式：公交、火车和飞机，风险系数分别为 2、5、
9。\n");
   printf("公交班次为：每两个城市间通一班公交车，时间为 8 点出发，12 点到
达；14 点开始返程，18 点回归。\n");//4*10=40 个结点
   printf("火车班次为：八班。时刻表见附件。\n");//21*8=168 个结点
   printf("飞机班次为：三班。时刻表见附件。\n");//7*3=21 个结点
   printf("\n\n\n\n\n");

   double CityRisk[10] =
{ 0.2,0.2,0.2,0.2,0.5,0.5,0.5,0.9,0.9,0.9 };

   for (int i = 1;i <= NumOfNode + 1;i++)
   {
       for (int j = 1;j <= NumOfNode + 1;j++)
           M[i][j] = INFINITY;
   }

   for (int i = 1;i <= 10;i++)
   {
       l[4 * i - 3].time = 8;
       l[4 * i - 2].time = 12;
       l[4 * i - 1].time = 14;
       l[4 * i].time = 18;
       for (int j = 4 * (i - 1) + 1;j <= 4 * i;j++)
           l[j].loc = 64 + i;
   }

   for (int i = 1;i <= 40;i++)
   {
       for (int j = 1;j <= 40;j++)
       {
           if ((((l[j].time - l[i].time)) == 4) && (l[i].loc !=
l[j].loc))
               M[i][j] = 2;
       }
   }

   for (int i = 0;i < 8;i++)
   {
       l[40 + 21 * i + 1].time = 6 + i;
       l[40 + 21 * i + 2].time = 6.1 + i;
       l[40 + 21 * i + 3].time = 7 + i;
       l[40 + 21 * i + 4].time = 7.1 + i;
       l[40 + 21 * i + 5].time = 8 + i;
       l[40 + 21 * i + 6].time = 8.1 + i;
       l[40 + 21 * i + 7].time = 9 + i;
       l[40 + 21 * i + 8].time = 9.1 + i;
       l[40 + 21 * i + 9].time = 10 + i;
       l[40 + 21 * i + 10].time = 10.1 + i;
```

```
        l[40 + 21 * i + 11].time = 11 + i;
        l[40 + 21 * i + 12].time = 11.1 + i;
        l[40 + 21 * i + 13].time = 12 + i;
        l[40 + 21 * i + 14].time = 12.1 + i;
        l[40 + 21 * i + 15].time = 13 + i;
        l[40 + 21 * i + 16].time = 13.1 + i;
        l[40 + 21 * i + 17].time = 14 + i;
        l[40 + 21 * i + 18].time = 14.1 + i;
        l[40 + 21 * i + 19].time = 15 + i;
        l[40 + 21 * i + 20].time = 15.1 + i;
        l[40 + 21 * i + 21].time = 16 + i;

        l[40 + 21 * i + 1].loc = 'A';
        l[40 + 21 * i + 2].loc = 'A';
        l[40 + 21 * i + 3].loc = 'B';
        l[40 + 21 * i + 4].loc = 'B';
        l[40 + 21 * i + 5].loc = 'C';
        l[40 + 21 * i + 6].loc = 'C';
        l[40 + 21 * i + 7].loc = 'D';
        l[40 + 21 * i + 8].loc = 'D';
        l[40 + 21 * i + 9].loc = 'E';
        l[40 + 21 * i + 10].loc = 'E';
        l[40 + 21 * i + 11].loc = 'F';
        l[40 + 21 * i + 12].loc = 'F';
        l[40 + 21 * i + 13].loc = 'G';
        l[40 + 21 * i + 14].loc = 'G';
        l[40 + 21 * i + 15].loc = 'H';
        l[40 + 21 * i + 16].loc = 'H';
        l[40 + 21 * i + 17].loc = 'I';
        l[40 + 21 * i + 18].loc = 'I';
        l[40 + 21 * i + 19].loc = 'J';
        l[40 + 21 * i + 20].loc = 'J';
        l[40 + 21 * i + 21].loc = 'A';
    }

    l[209].time = 7;
    l[209].loc = 'A';
    l[210].time = 7.15;
    l[210].loc = 'A';
    l[211].time = 7.5;
    l[211].loc = 'J';
    l[212].time = 7.65;
    l[212].loc = 'J';
    l[213].time = 8;
    l[213].loc = 'D';
    l[214].time = 8.15;
    l[214].loc = 'D';
    l[215].time = 9;
    l[215].loc = 'A';
```

```
l[216].time = 10;
l[216].loc = 'B';
l[217].time = 10.15;
l[217].loc = 'B';
l[218].time = 10.5;
l[218].loc = 'F';
l[219].time = 10.65;
l[219].loc = 'F';
l[220].time = 11;
l[220].loc = 'H';
l[221].time = 11.15;
l[221].loc = 'H';
l[222].time = 11.5;
l[222].loc = 'B';

l[223].time = 14;
l[223].loc = 'C';
l[224].time = 14.15;
l[224].loc = 'C';
l[225].time = 14.5;
l[225].loc = 'E';
l[226].time = 14.65;
l[226].loc = 'E';
l[227].time = 15;
l[227].loc = 'G';
l[228].time = 15.15;
l[228].loc = 'G';
l[229].time = 15.5;
l[229].loc = 'C';

for (int i = 1;i <= 229;i++)
{
    for (int j = 1;j <= 229;j++)
    {
        if ((l[i].loc == l[j].loc))
        {
            if (((l[j].time - l[i].time) > 0))
                M[i][j] = CityRisk[(l[i].loc) - 65];
        }
        else
        {
            if (((l[j].time - l[i].time)) == 0.9)
                M[i][j] = 5;
            else
            {
                if (((l[j].time - l[i].time)) == 0.35)
                    M[i][j] = 9;
            }
```

```
            }
        }
    }
}

void Examine(Node* l, double M[][NumOfNode + 2])
{
    FILE* fp = fopen("List.txt", "w+");
    FILE* pf = fopen("Matrix.txt", "w+");
    for (int i = 1;i <= NumOfNode;i++)
        fprintf(fp, "%lf %c\n", l[i].time, l[i].loc);
    for (int i = 1;i <= NumOfNode;i++)
    {
        for (int j = 1;j <= NumOfNode;j++)
        {
            fprintf(pf, "%lf ", M[i][j]);
        }
        fprintf(pf, "\n");
    }
}

double Search(char c)
{
    switch (c)
    {
    case 'A':
        return 0.2;
        break;
    case 'B':
        return 0.2;
        break;
    case 'C':
        return 0.2;
        break;
    case 'D':
        return 0.2;
        break;
    case 'E':
        return 0.5;
        break;
    case 'F':
        return 0.5;
        break;
    case 'G':
        return 0.5;
        break;
    case 'H':
        return 0.9;
        break;
```

```
   case 'I':
       return 0.9;
       break;
   case 'J':
       return 0.9;
       break;
   }
 }

 void GetIns(double cost[][NumOfNode + 2], Node* l, double
M[][NumOfNode + 2], double len[NumOfNode + 2], int visit[NumOfNode
+ 2], int n, int path[][NumOfNode + 2], string OutLoc[][NumOfNode +
2], int timepath[][NumOfNode + 2][NumOfNode + 2])
 {
   int choice;

   for (int i = 1;i <= n;i++)
       for (int j = 1;j <= n;j++)
           for (int k = 1;k <= n;k++)
               timepath[i][j][k] = 0;

   FILE* out = fopen("Diary.txt", "a+");

   for (int i = 1;i <= n;i++)
       for (int j = 1;j <= n;j++)
       {
           OutLoc[i][j] = "";
           //cout << OutLoc[i][j] << endl;
       }

   for (int k = 1;k <= n;k++)
       for (int i = 1;i <= n;i++)
           path[i][k] = 1;

   double GoTime, timewant;
   char Ffrom, Fto;
   //FILE* fp = fopen("cost.txt", "w+");
   printf("请选择你想采用的规划策略（1 为最少风险策略，2 为限时最少风险策
略）：\n");
   scanf("%d", &choice);
   if (choice == 1)
   {
       printf("请按以下格式输入您的旅行计划：出发时间 起始城市 终点城市
（如 8.5 A C）：\n");
       scanf("%lf %c %c", &GoTime, &Ffrom, &Fto);
       fprintf(out, "选择最低风险策略，%lf 从%c 出发前往%c。\n",
GoTime, Ffrom, Fto);
       l[NumOfNode + 1].time = GoTime;
       l[NumOfNode + 1].loc = Ffrom;
```

```
      for (int a = 1;a <= NumOfNode + 1;a++)
      {
          if (l[a].loc == Ffrom)
          {
              if (l[a].time >= GoTime)
                  M[NumOfNode + 1][a] = Search(Ffrom);
          }
      }

      for (int i = 1;i <= n;i++)
          for (int j = 1;j <= n;j++)
              cost[i][j] = 9999;

      /*for (int i = 1;i <= n;i++)
      {
          for (int j = 1;j <= n;j++)
              fprintf(fp, "%lf %c %lf %c  %lf %lf\n", l[i].time,
l[i].loc, l[j].time, l[j].loc, M[i][j], cost[i][j]);
      }*/

      for (int i = 1;i <= n;i++)
          for (int j = 1;j <= n;j++)
          {
              if ((M[i][j] != INFINITY))
                  cost[i][j] = M[i][j] * (l[j].time - l[i].time);
              else
              {
                  if (l[i].loc == l[j].loc)
                      cost[i][j] = 0;
              }
          }
      /*for (int i = 1;i <= n;i++)
      {
          for (int j = 1;j <= n;j++)
              fprintf(fp, "%lf %c %lf %c  %lf %lf\n", l[i].time,
l[i].loc, l[j].time, l[j].loc, M[i][j], cost[i][j]);
      }*/

      for (int k = 1;k <= n;k++) { //选中的中间值
          for (int i = 1;i <= n;i++) { //数组横坐标
                  for (int j = 1;j <= n;j++) { //数组纵坐标
                      if ((cost[i][j] > cost[i][k] + cost[k][j]) &&
(l[i].time <= l[k].time) && (l[k].time <= l[j].time))
                          {    //如果以 k 中间点为中间点检测到路径更短
                              cost[i][j] = cost[i][k] + cost[k][j];
//更新路径值
                              OutLoc[i][j] += l[k].loc;
                                  timepath[i][j][path[i][j]] = k;
                                  path[i][j] ++;   //更新要经过的中间点
```

```
                        }
                }


            }
        }

        int to[40] = { 0 };
        int i = 1, s = 1;



        for (i = 1;i <= NumOfNode + 1;i++)
        {
            if ((l[i].loc == Fto) && (l[i].time > GoTime))
            {
                to[s] = i;
                s++;
            }
        }
        int res = 1;
        for (int q2 = 2;q2 < s;q2++)
        {
            if (l[q2].time >= l[NumOfNode + 1].time)
            {
                if (cost[NumOfNode + 1][to[q2]] <= cost[NumOfNode +
1][to[res]])
                {
                    res = q2;
                }
                //printf("%lf %c %lf %c %lf %c %s %c\n", l[NumOfNode
+ 1].time, l[NumOfNode + 1].loc, l[to[q2]].time, l[to[q2]].loc,
cost[NumOfNode + 1][to[q2]], l[NumOfNode + 1].loc, OutLoc[NumOfNode
+ 1][to[q2]].c_str(), l[to[q2]].loc);//输出点 1 到点 n 的最短距离;
            }
        }

        //printf("%lf %c 到%lf %c 的路径最低风险系数为：%lf，路径为：\n",
l[NumOfNode + 1].time, l[NumOfNode + 1].loc, l[to[res]].time,
l[to[res]].loc, cost[NumOfNode + 1][to[res]]);//输出点 1 到点 n 的最短
距离;
        //fprintf(out, "到达%c 的时间为%lf，最低风险系数为：%lf。\n",
l[to[res]].loc, l[to[res]].time, cost[NumOfNode + 1][to[res]]);//输
出点 1 到点 n 的最短距离;//_fcloseall()
        for (int ww = 1;ww < path[NumOfNode + 1][to[res]]-1;ww++)
        {
            for (int www = ww + 1;www < path[NumOfNode +
1][to[res]];www++)
```

```
          {
              if (l[timepath[NumOfNode + 1][to[res]][www]].time <
l[timepath[NumOfNode + 1][to[res]][ww]].time)
              {
                  double tt = l[timepath[NumOfNode +
1][to[res]][www]].time;
                  char mm = l[timepath[NumOfNode +
1][to[res]][www]].loc;

                  l[timepath[NumOfNode + 1][to[res]][www]].time =
l[timepath[NumOfNode + 1][to[res]][ww]].time;
                  l[timepath[NumOfNode + 1][to[res]][ww]].time = tt;

                  l[timepath[NumOfNode + 1][to[res]][www]].loc =
l[timepath[NumOfNode + 1][to[res]][ww]].loc;
                  l[timepath[NumOfNode + 1][to[res]][ww]].loc = mm;
              }
          }
      }
      double update = 0;
      char recent = 'X';

      printf("%lf %c 到%lf %c 的路径最低风险系数为：%lf，下面是路径展
示。\n", l[NumOfNode + 1].time, l[NumOfNode + 1].loc,
l[to[res]].time, l[to[res]].loc, cost[NumOfNode + 1][to[res]]);//输
出点 1 到点 n 的最短距离;
      fprintf(out, "到达%c 的时间为%lf，最低风险系数为：%lf。下面是路径
展示。\n", l[to[res]].loc, l[to[res]].time, cost[NumOfNode +
1][to[res]]);//输出点 1 到点 n 的最短距离;
      //_fcloseall()
      while ((-update + l[to[res]].time) > 1e-6)
      {

          //cout << path[NumOfNode + 1][to[res]] << endl;
          Sleep(1000);
          update += 0.1;
          //cout << l[NumOfNode + 1].time << endl;
          if ((-update + l[NumOfNode + 1].time) < 1e-6)
          {

              recent = l[NumOfNode + 1].loc;
              printf("%.2lf %c\n", update, recent);
              fprintf(out, "%.2lf %c\n", update, recent);
          }
          else
          {
              printf("%.2lf %c\n", update, recent);
              fprintf(out, "%.2lf %c\n", update, recent);
          }
```

```
          for (int xx = 1;xx < path[NumOfNode + 1][to[res]];xx++)
          {
              if ((-update + l[timepath[NumOfNode +
1][to[res]][xx]].time) < 1e-6)
              {
                  /*Sleep(1);
                  update += 0.1;*/
                  recent = l[timepath[NumOfNode +
1][to[res]][xx]].loc;
                  //update = l[timepath[NumOfNode +
1][to[res]][xx]].time;
                  printf("%.2lf %c\n", update, recent);
                  fprintf(out, "%.2lf %c\n", update, recent);

                  //printf("%.2lf %c\n", l[timepath[NumOfNode +
1][to[res]][xx]].time, l[timepath[NumOfNode +
1][to[res]][xx]].loc);
              }
              else
              {
                  /*Sleep(1);
                  update += 0.1;*/
              }

          }

          if ((-update + l[to[res]].time) < 1e-6)
          {
              /*Sleep(1);
              update += 0.1;*/
              recent = l[to[res]].loc;
              /*printf("%.2lf %c\n", l[to[res]].time,
l[to[res]].loc);
              fprintf(out, "%.2lf %c\n", l[to[res]].time,
l[to[res]].loc);*/
          }
          else
          {
              /*Sleep(1);
              update += 0.1*/;
          }
      }


      //LowestRisk(GoTime, Ffrom, Fto, l, M, len, visit, n);
  }
  else
  {
      if (choice == 2)
```

```
      {
          printf("请按以下格式输入您的旅行计划：出发时间 起始城市 终点城
市 预计用时（如 8.5 A C 4）：\n");
          scanf("%lf %c %c %lf", &GoTime, &Ffrom, &Fto, &timewant);
          fprintf(out, "选择限时最低风险策略，%lf 从%c 出发前往%c，预计
用时为%lf。\n", GoTime, Ffrom, Fto, timewant);
          fprintf(out, "选择最低风险策略，%lf 从%c 出发前往%c。\n",
GoTime, Ffrom, Fto);
          l[NumOfNode + 1].time = GoTime;
          l[NumOfNode + 1].loc = Ffrom;
          for (int a = 1;a <= NumOfNode + 1;a++)
          {
              if (l[a].loc == Ffrom)
              {
                  if (l[a].time >= GoTime)
                      M[NumOfNode + 1][a] = Search(Ffrom);
              }
          }

          for (int i = 1;i <= n;i++)
              for (int j = 1;j <= n;j++)
                  cost[i][j] = 9999;

          /*for (int i = 1;i <= n;i++)
          {
              for (int j = 1;j <= n;j++)
                  fprintf(fp, "%lf %c %lf %c  %lf %lf\n", l[i].time,
l[i].loc, l[j].time, l[j].loc, M[i][j], cost[i][j]);
          }*/

          for (int i = 1;i <= n;i++)
              for (int j = 1;j <= n;j++)
              {
                  if ((M[i][j] != INFINITY))
                      cost[i][j] = M[i][j] * (l[j].time -
l[i].time);
                  else
                  {
                      if (l[i].loc == l[j].loc)
                          cost[i][j] = 0;
                  }
              }
          /*for (int i = 1;i <= n;i++)
          {
              for (int j = 1;j <= n;j++)
                  fprintf(fp, "%lf %c %lf %c  %lf %lf\n", l[i].time,
l[i].loc, l[j].time, l[j].loc, M[i][j], cost[i][j]);
          }*/
```

```
          for (int k = 1;k <= n;k++) { //选中的中间值
              for (int i = 1;i <= n;i++) { //数组横坐标
                     for (int j = 1;j <= n;j++) { //数组纵坐标

                              if ((cost[i][j] > cost[i][k] +
cost[k][j]) && (l[i].time <= l[k].time) && (l[k].time <=
l[j].time))
                              {    //如果以 k 中间点为中间点检测到路径更
短
                                  cost[i][j] = cost[i][k] +
cost[k][j];    //更新路径值

                                  OutLoc[i][j] += l[k].loc;
                                  timepath[i][j][path[i][j]] = k;
                                  path[i][j] ++;   //更新要经过的中间点
                              }

                     }

              }
          }

          int to[40] = { 0 };
          int i = 1, s = 1;



          for (i = 1;i <= NumOfNode + 1;i++)
          {
              if ((l[i].loc == Fto) && (l[i].time > GoTime) &&
(l[i].time <= GoTime + timewant))
              {
                  to[s] = i;
                  s++;
              }
          }
          if (s > 1)
          {
              int res = 1;
              for (int q2 = 2;q2 < s;q2++)
              {
                  if (l[q2].time >= l[NumOfNode + 1].time)
                  {
                      if (cost[NumOfNode + 1][to[q2]] <=
cost[NumOfNode + 1][to[res]])
                      {
                          res = q2;
                      }
```

```
                    //printf("%lf %c %lf %c %lf %c %s %c\n",
l[NumOfNode + 1].time, l[NumOfNode + 1].loc, l[to[q2]].time,
l[to[q2]].loc, cost[NumOfNode + 1][to[q2]], l[NumOfNode + 1].loc,
OutLoc[NumOfNode + 1][to[q2]].c_str(), l[to[q2]].loc);//输出点 1 到点
n 的最短距离;
                }
            }

            for (int ww = 1;ww < path[NumOfNode + 1][to[res]] -
1;ww++)
            {
                for (int www = ww + 1;www < path[NumOfNode +
1][to[res]];www++)
                {
                    if (l[timepath[NumOfNode +
1][to[res]][www]].time < l[timepath[NumOfNode +
1][to[res]][ww]].time)
                    {
                        double tt = l[timepath[NumOfNode +
1][to[res]][www]].time;
                        char mm = l[timepath[NumOfNode +
1][to[res]][www]].loc;

                        l[timepath[NumOfNode +
1][to[res]][www]].time = l[timepath[NumOfNode +
1][to[res]][ww]].time;
                        l[timepath[NumOfNode +
1][to[res]][ww]].time = tt;

                        l[timepath[NumOfNode +
1][to[res]][www]].loc = l[timepath[NumOfNode +
1][to[res]][ww]].loc;
                        l[timepath[NumOfNode +
1][to[res]][ww]].loc = mm;
                    }
                }
            }

            double update = 0;
            char recent = 'X';

            printf("%lf %c 到%lf %c 的路径最低风险系数为：%lf，下面是
路径展示。\n", l[NumOfNode + 1].time, l[NumOfNode + 1].loc,
l[to[res]].time, l[to[res]].loc, cost[NumOfNode + 1][to[res]]);//输
出点 1 到点 n 的最短距离;
            fprintf(out, "到达%c 的时间为%lf，最低风险系数为：%lf。下
面是路径展示。\n", l[to[res]].loc, l[to[res]].time, cost[NumOfNode +
1][to[res]]);//输出点 1 到点 n 的最短距离;
            //_fcloseall()
```

```
            while ((- update + l[to[res]].time) > 1e-6)
            {

                //cout << path[NumOfNode + 1][to[res]] << endl;
                Sleep(1000);
                update += 0.1;
                //cout << l[NumOfNode + 1].time << endl;
                if ((- update + l[NumOfNode + 1].time) < 1e-6)
                {

                    recent = l[NumOfNode + 1].loc;
                    printf("%.2lf %c\n", update, recent);
                    fprintf(out, "%.2lf %c\n", update, recent);
                }
                else
                {
                    printf("%.2lf %c\n", update, recent);
                    fprintf(out, "%.2lf %c\n", update, recent);;
                }
                for (int xx = 1;xx < path[NumOfNode +
1][to[res]];xx++)
                {
                    if ((- update + l[timepath[NumOfNode +
1][to[res]][xx]].time) < 1e-6)
                    {
                        /*Sleep(1);
                        update += 0.1;*/
                        recent = l[timepath[NumOfNode +
1][to[res]][xx]].loc;
                        //update = l[timepath[NumOfNode +
1][to[res]][xx]].time;
                        printf("%.2lf %c\n", update, recent);
                        fprintf(out, "%.2lf %c\n", update,
recent);

                        //printf("%.2lf %c\n",
l[timepath[NumOfNode + 1][to[res]][xx]].time, l[timepath[NumOfNode
+ 1][to[res]][xx]].loc);
                    }
                    else
                    {
                        /*Sleep(1);
                        update += 0.1;*/
                    }

                }

                if ((- update + l[to[res]].time) < 1e-6)
                {
```

```
                    /*Sleep(1);
                    update += 0.1;*/
                    recent = l[to[res]].loc;
                    printf("%.2lf %c\n", l[to[res]].time,
l[to[res]].loc);
                    fprintf(out, "%.2lf %c\n", l[to[res]].time,
l[to[res]].loc);
                }
                else
                {
                    /*Sleep(1);
                    update += 0.1*/;
                }
            }
        }
        else
        {
            printf("无法在您需要的时刻到达\n\n");
            fprintf(out, "无法在您需要的时刻到达\n\n");
        }

    }
    else
    {
        printf("您输入的策略序号不正确，请重新输入！\n\n");
        fprintf(out, "您输入的策略序号不正确，请重新输入！\n\n");
        getchar();
    }
  }

  fclose(out);
 }
```