图 7.33 的交通图，对从北京到上海的火车，需给出北京至天津、天津至徐州及徐州至上海各段的出发时间、到达时间及票价等信息。

(2) 以邻接表作交通图的存储结构，表示边的结点内除含有邻接点的信息外，还应包括交通工具、路程中消耗的时间和花费以及出发和到达的时间等多项属性。

【选作内容】

增加旅途中转次数最少的最优决策。

实习报告示例：5.5 题　校园导游咨询

实习报告

题目：编制一个为来访客人进行最短路径导游的程序

班级：计算机 95(1)　姓名：丁一　学号：954211　完成日期：1997.12.4

一、需求分析

1. 从清华大学的平面图中选取 19 个有代表性的景点，抽象成一个无向带权图。以图中顶点表示景点，边上的权值表示两地之间的距离。

2. 本程序的目的是为用户提供路径咨询。根据用户指定的始点和终点输出相应路径，或者根据用户指定的景点输出景点的信息。

3. 测试数据(附后)。

二、概要设计

1. 抽象数据类型图的定义如下：

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∈V,(v,w)表示 v 和 w 之间存在路径}

基本操作 P：

CreateGraph (&G,V,VR)

初始条件：V 是图的顶点集，VR 是图中边的集合。

操作结果：按 V 和 VR 的定义构造图 G。

DestroyGraph (&G)

初始条件：图 G 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 的信息。

FirstEdge（G，v）

初始条件：图 G 存在，v 是 G 中某个顶点。

操作结果：返回依附于 v 的第一条边。若该顶点在 G 中没有邻接点，则返回“空”。

NextEdge（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：返回依附于 v 的（相对于 w 的）下一条边。若不存在，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 及其相关的边。

InsertEdge（&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 中增添边（v，w）。

DeleteEdge（&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 中删除边（v，w）。

GetShortestPath（G，st，nd，&Path）

初始条件：图 G 存在，st 和 nd 是 G 中两个顶点。

操作结果：若 st 和 nd 之间存在路径，则以 Path 返回两点之间一条最短路径，否则返回其他信息。

} **ADT Graph**

2. 主程序

```
void mian( )
{
  初始化；
  do {
    接受命令（输入景点信息或输出最短路径）；
    处理命令；
  } while（“命令”!＝“退出”）；
```

}

3. 本程序只有两个模块，调用关系简单。

三、详细设计

1. 顶点、边和图类型

```
#define  MAXVTXNUM   20                    // 图中顶点数的最大值
typedef struct {
        string name;                       //该顶点代表的景点的名称
        string info;                       //景点的信息
    } VertexType;                          // 顶点类型
typedef struct {
        int  lengh;                        // 边的权值，表示路径长度
        int  ivex, jvex;                   // 边的两端顶点号
    } EdgeType;                            // 边的类型
typedef struct EdgeNode {
        EdgeType   elem;
        EdgeNode   *ilink, *jlink;
    } EdgeNode, *EdgePtr;                  // 边的结点类型，指向边的指针
typedef struct {
        VertexType data;
        EdgePtr    firstEdge;              // 指向第一条依附该顶点的边
                                           的指针
    } VNode;                               // 顶点类型
typedef struct {
        VNode Adjmulist[MAXVTXNUM];        // 邻接多重表
        int    vexNum, edgeNum;            // 图中的顶点数和边数
    } GraphType;                           // 图类型
```

图的基本操作：

```
void InitGrah(GraphType &g);
// 初始化邻接多重表，表示一个空的图。g.vexNum=g.edgeNum=0。
status LocateVex(GraphType &g, string uname, int &i );
// 在图中查找其景点名称和 uname 相同的顶点。若存在，则以 i 返回其在邻接多
// 重表中的位置并返回 TRUE(g.Adjmulist[i].data.name=uname)，否则返回
// FALSE。
```

```
void GetVex(GraphType g, int i, VertexType &v);
// 以 v 返回邻接多重表中序号为 i 的顶点 g. Adjmulist[i]. data。
EdgePtr FirstEdge(GraphType g, int vi);
// 返回图 g 中指向依附于顶点 vi 的第一条边的指针 g. adjmulist[vi]. firstEdge。
void NextEdge(GraphType g, int vi, EdgePtr p, int &vj, EdgePtr &q);
// 以 vj 返回图 g 中依附于顶点 vi 的一条边(由指针 p 所指)的另一端点;
// 以 q 返回图 g 中依附于顶点 vi 且相对于指针 p 所指边的下一条边。
void InsertVex(GraphType &g, VertexType v);
// 在图 g 的邻接多重表中添加一个顶点 v。
void InsertEdge(GraphType &g, EdgeType e);
// 在图 g 的邻接多重表中添加一条边 e。
void DeleteVex(GraphType &g, VertexType v);
// 从图 g 中删除顶点 v 及所有依附该顶点的边。
void DeleteEdge( GraphType &g, EgdeType e);
// 从图中删除边 e。
```

其中部分操作的伪 EdgePtr 码算法如下：

```
void NextEdge(GraphType g, int vi, EdgePtr p, int &vj, EdgePtr &q)
{
//以 vj 返回依附于顶点 vi 的一条边(由指针 p 所指)的另一端点;
//以 q 返回 vi 在图 g 中相对于该边的下一条边。
  if (p->elem. ivex == vi) {q = p->ilink; vj = p->elem. jvex;}
  else {q = p->jlink; vj = p->elem. ivex;}
}

void InsertEdge(GraphType &g, EdgeType e)
{
//在图 g 的邻接多重表中添加一条边 e。
p = (EdgePtr)malloc(sizeof(EdgeNode));
p->elem = e;
p->ilink = FirstEdge(g, e. ivex);
p->jlink = FirstEdge(g, e. jvex);
g. Adjmulist[e. ivex]. firstedge = g. Adjmulist[e. jvex]. firstedge = p;
}
```

2. 路径类型

```
typedef struct {
        int vx, vy;
      } Edge;
```

```
typedef struct {
        Edge   edges[MAXVTXNUM];  // 路径中边的序列
        int    len;               // 路径中边的数目
      } PathType;
typedef struct {
        string  vertices[MAXVTXNUM];  // 路径中景点的序列
        int  num;
      } PType;
```

相关的基本操作有：

```
void InitPath( PathType &pa );
//初始化 pa 为一条空路径(pa.len = 0)。
void copyPath( PathType &p1, PathType p2);
//复制路径 p1=p2。
void InsertPath(PathType &pa, int v, int w );
//在路径 pa 中插入一条边(v,w)。
int PathLength( PathType pa );
//返回路径 pa 的长度。
void OutPath( GraphType g, PathType pa, PType &vtxes );
//将路径转换为景点名称的序列。
```

其中部分操作的伪码算法如下：

```
void copyPath( PathType &p1, PathType p2)
{
//复制路径 p1=p2。
  for ( i = 0; i<p2.len; i++ ) {
    p1.edges[i].vx = p2.edges[i].vx;
    p1.edges[i].vy = p2.edges[i].vy
  }
  p1.len = p2.len;
}

void InsertPath(PathType &pa, int v, int w)
{
  //在路径 pa 中插入一条边(v, w)。
  pa.edges[pa.len].vx = v;
  pa.edges[pa.len].vy = w;
  pa.len++;
}
```

```
void OutPath(GraphType g, PathType pa, PType &vtxes)
{
  //将路径转换为景点名称的序列。
  m = 0;
  for ( i = 0; i<pa.len; i++ ) {
    GetVex(g, pa.edges[i].vx, vtx );
    vtxes[m++] = vtx.name;
  }
  GetVex(g, pa.edges[pa.len].vy, vtx);
  vtxes[m] = vtx.name;
  vtxes.num = m;
}
```

3. 主程序和其他伪码算法

```
void main( )
{
  //主程序
  Initialization;        // 系统初始化
  do {
    显示各景点名称;
    ReadCommand(cmd);    // 读入一个操作命令符
    Interpret(cmd)       // 解释执行操作命令符
  } while ( cmd != 'q' && cmd != 'Q' );
} // main

void Initialization( )
{
  // 系统初始化
  ClrScr;              // 清屏
  显示以串的形式键入含导游图的数据文件名;
  scanf(filename);  //读入文件名
  fin = fopen( filename, 'r');
  CreatGraph(ga, fin);     // 从文件读入数据并建立图的多重邻接表
} //Initialization

void ReadCommand(char &cmd)
{
  // 读入操作命令符
  显示键入操作命令符的提示信息;
  do {cmd = getche( );} while (cmd 不属于['s', 'S', 'v', 'V', 'q', 'Q']);
```

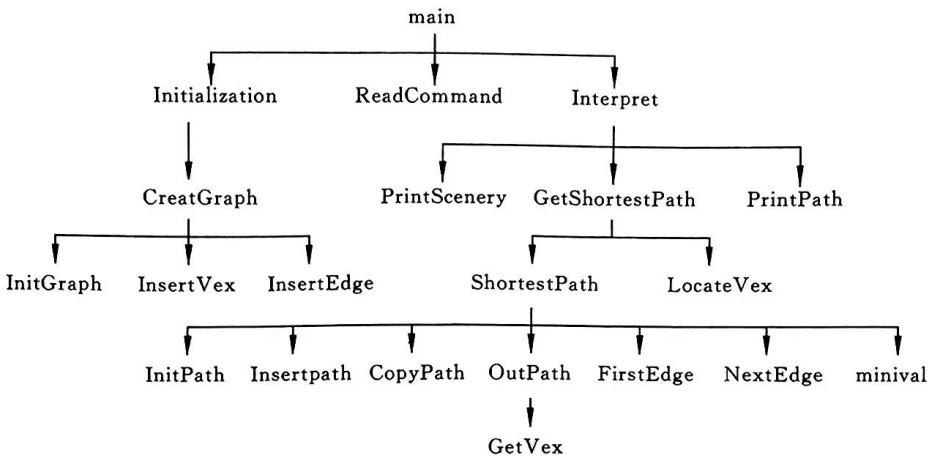
```
}
void Interpret(char cmd)
{
  // 解释执行操作命令 cmd
  switch (cmd) {
    case 's','S': 显示以串的形式键入景点名称的提示信息;
                  scanf(sname);                    //读入景点名称
                      PrintScenery(sname);         //显示景点信息
                      break;
    case 'v','V':显示以串的形式键入始点名称和终点名称的提示信息;
                  scanf(sname, tname);      // 读入始点和终点名称
                      GetShortestPath(ga, sname, tname, pathlen, spath );
                      PrintPath(spath, pathLen );  // 输出最短路径及其长度
                      break;
    case 'q','Q':
  }
} //Interpret
void CreatGraph(graphtype &g, FILE *f)
{
  //从文件 f 中读入顶点和边的数据,建立图的多重邻接表
  InitGraph(g);
  fscanf(f,g.vexNum, g.edgeNum);
  for (i = 0; i<g.vexNum; i++ ) {
    fscanf(f, v.name, v.info );
    InsertVex( g, v);
  }
  for (k = 0; k<g.edgeNum; k++ ) {
    fscanf(f, e.ivex, e.jvex, e.length);
    if (e.length) InsertEdge(g, e);
  }
}
void GetShortestPath(GraphType g, string sname, string tname,
                     int &pathLength, PType &PathInfo )
{
  // 求从景点 sname 到景点 tname 的一条最短路径及其长度
  LocateVex(g, sname, sv);
  LocateVex(g, tname, tv);
```

```
  Shortestpath(g, sv, tv, pathLength, PathInfo);
}

void ShortestPath(GraphType g, int st, int nd,
                  int &pathLength, PType &PathInfo )
{
  // 利用迪杰斯特拉算法的基本思想求图 g 中从顶点 st 到顶点 nd 的一条
  // 最短路径 PathInfo 及其路径长度 pathLength。
  // 设 int dist[MAXVTXNUM]; PathType path[MAXVTXNUM];
    for (i = 0; i<g. vexNum; i++)  //初始化
    { dist[i] = maxint; InitPath(path[i]); }
  p = Firstedge(g, st);
  while (p) {  //初始化 dist 数组,检测依附于起始点的每一条边
    NextEdge(g, p,st,q,adjvex);
    dist[adjvex] = p->length;
    InsertPath(path[adjvex],st,adjvex);
    p = q;
  }
  found = FALSE;
  InitSet(ss); PutInSet(st, ss);     //设 ss 为已求得最短路径的顶点的集合
  while (!found) {
    min = minval(dist);
          //在所有尚未求得最短路径的顶点中求使 dist[i]取最小值的 i 值
    if (min == nd) found = true;
    else {
      v = min; PutInSet(v, ss);  // 将 v 加入集合 ss
      p = Firstedge(g, v);
      while (p) {     //检测依附于 v 的每一条尚未访问过的边
        NextEdge(g, p,v,q,w);
        if (!InSet(w, ss)/ * w 不在 ss 中 * / &&
        (dist[v]+p->length) < dist[w] ) {
          dist[w] = dist[v]+p->length;
          copyPath(path[w], path[v]);
          InsertPath(path[w], v, w);
        }
        p = q;
      } // while (p)
    } // else
  } // while ( !found )
```

```
    pathLength = dist[nd];
    OutPath(g, path[nd], PathVal);
}
```

4. 函数的调用关系图



四、设计和调试分析

1. 原设计在边的属性中加上访问标志域 mark，意图以!(p->mark)代替!InSet(w, ss)来判别是否需要检测该条边，后发现，如此只能求出第一对顶点之间的最短路径。在继续求其他对顶点的最短路径时，必须恢复所有边的访问标志为 FALSE，则需要耗费 $O(e)$的时间，并不比现在的算法优越，故舍去之。

2. 考虑道路网多是稀疏网，故采用邻接多重表作存储结构，其空间复杂度为 $O(e)$，此时的时间复杂度也为 $O(e)$。构建邻接多重表的时间复杂度为 $O(n+e)$，输出路径的时间复杂度为 $O(n)$。由此，本导游程序的时间复杂度为 $O(n+e)$。

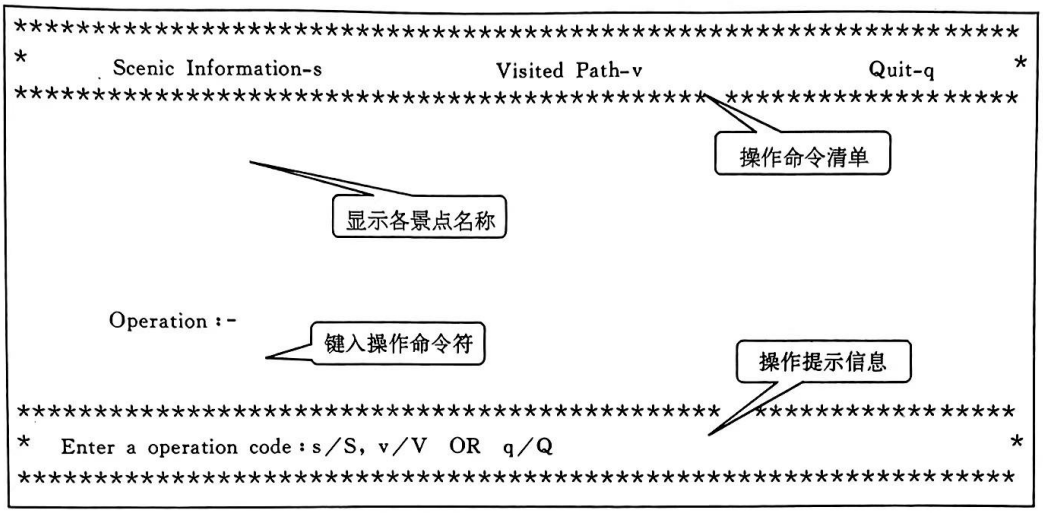
3. 由于导游程序在实际执行时，需要根据用户的临时输入求最短路径。因此，虽然迪杰斯特拉算法的时间复杂度比弗洛伊德算法低，但每求一条最短路径都必须重新搜索一遍，在频繁查询时会导致查询效率降低，而弗洛伊德算法只要计算一次，即可求得每一对顶点之间的最短路径，虽然时间复杂度为 $O(n^3)$，但以后每次查询都只要查表即可，极大地提高了查询效率，而且，弗洛伊德算法还支持带负权的图的最短路径的计算。由此可见，在选用算法时，不能单纯地只考虑算法的渐近时间复杂度，有时还必须综合考虑各种因素。

五、用户手册

1. 本程序的运行环境为 DOS 操作系统，执行文件为：GraphTest.exe；

2. 进入演示程序后首先提示以串的形式输入数据文件名，随即从文件读入道路网的数据并建立图的多重邻接表；

3. 之后即显示文本方式的用户界面：



4. 进入“查询景点信息(s/S)”的命令后，即提示以串的形式输入景点的名称，接受输入之后即显示该景点的信息；

5. 进入“查询路径信息(v/V)”的命令后，即提示以串的形式输入起始点和终点的名称，在接受用户的输入之后即显示两点之间的一条最短路径。

六、测试结果

1. 操作命令符为 v，

Source (始点):NorthGate(北门)；Destination (终点):SouthGate(南门)；

Shortest Path(最短路径)：NorthGate(北门)——28＃Building(二十八号楼)——10thRestaurant(第十食堂)——3rdTeachBuilding(三教)——TrafficTower(交通岗)——SouthGate(南门)

Shortest Distance(最短路径长度)：250

2. 操作命令符为 v，

Source (始点)：JinChuanYuan (近春园)；Destination (终点):30＃Building(三十号楼)；

Shortest Path(最短路径)：JinChuanYuan (近春园)——WestPlayground(西大操场)——Library (图书馆)——10 thRestaurant(第十食堂)——30＃Building(三十号楼)

Shortest Distance(最短路径长度)：95

3. 操作命令符为 v，

Source (始点)：Hospital(校医院)；Destination (终点):MainBuilding(主楼)；

Shortest Path(最短路径)：Hospital(校医院)——JinChuanYuan (近春园)——WestPlayground(西大操场)——Library (图书馆)——10 thRestaurant(第十食堂)——30＃Building(三十号楼)——EestPlayground(东大操场)——MainBuilding(主楼)

Shortest Distance(最短路径长度)：195

七、附录

源程序文件名清单：

```
GraphUnit.H      // 图类型
GraphTest.C      // 主程序
Graph.dat        //清华园数据文件
```

注解：

校园数据文件的格式：

```
图中顶点的数目,图中边的数目
景点名称,景点信息
        ⋮
// 以下是边的信息
始点号,终点号,路径长度
        ⋮
// 顶点号由景点的输入顺序自然形成
```