《数据结构与算法集中实习》

实习报告

学　院:　　遥感信息工程学院

班　级:　　**1802**

实习地点:　教学实验大楼**101**机房

指导教师:　黄玉春

组　员:　　**2018302130155　钟代琪**

2019年　**06** 月　**21** 日

最短路径算法及其实现

钟代琪[1]

1. 武汉大学遥感信息工程学院, 武汉 430070
E-mail: 1045541496@qq.com

摘　要:随着信息的高度增长，图的使用在日常生活中越来越重要。本次实习以最短路径及实现为主要目的，根据迪杰斯特拉算法，算出某个顶点v0到其他顶点v的最短距离，并在htm文件中以百度地图形式可视化的展现出来。同时，本文最后会计算时间复杂度和空间复杂度来评判该算法的可行性。

关键词: 图，最短路径，迪杰斯特拉算法，可视化

Abstract: With the high growth of information, the use of graphs is becoming more and more important in daily life. The main purpose of this internship is to shortest path and realization. According to Dijkstra algorithm, the shortest distance between one vertex V0 and other vertex v is calculated and visualized in the form of Baidu map in HTM file. At the same time, this paper finally calculates the time complexity and space complexity to evaluate the feasibility of the algorithm.

Key Words: Graph, Shortest Path, Dijkstra Algorithms, Visualization

引言

对于日益增长的出行需求，交通系统中一个核心问题为最优路径的选择，包括最短时间，最少费用等等。根据交通网络图中的出发点、换乘点、目的点，以及路线时间、路线费用等权值属性，通过迪杰斯特拉算法，获取最优路径。

1.1 问题的提出

如何获取交通线路中的最优路径。

1.2 本报告研究的意义

根据用户需求，可选择最短时间路径，也可选择最少费用的路径。

1.3 本报告研究的结构安排

本报告根据算法描述、算法分析、算法优劣评判、可能存在的问题以及总结展开，对实际问题中的最优路径做出相应分析。

问题提出

2.1 问题概述

本次实习内容为最短路径的实现及可视化输出。城市信息保存在cpp文件目录的cities.csv文件中，城市之间的交通线路信息保存在cpp文件目录的routes.csv文件中。

cities.csv文件具体格式如下图：

	A	B	C	D
1	Afghanista	Kabul	34.4667	69.1833
2	Albania	Tirane	41.3	19.8167
3	Algeria	Algiers	36.7	3.13333
4	American S	Pago Pago	-14.2667	-170.717
5	Andorra	Andorra la	42.5167	1.53333
6	Angola	Luanda	-8.83333	13.25
7	Antigua ar	W. Indies	17.3333	-61.8
8	Argentina	Buenos Air	-36.5	-60
9	Armenia	Yerevan	40.1667	44.5167
10	Aruba	Oranjestac	12.5333	-70.0333
11	Australia	Canberra (	-35.25	149.133
12	Austria	Vienna	48.2	16.3667

其中，第一列为国家名，第二列为城市名，第三列为城市所在纬度，第四列为城市所在经度。

routes.csv文件具体格式如下图：

	A	B	C	D	E	F
1	Abu Dhabi	Canberra (	plane	24	1339	Qatar Airw
2	Abu Dhabi	Lima	plane	30	1967	KLM Airway
3	Abu Dhabi	London	plane	10	666	Turkish Airl
4	Abu Dhabi	Manama	plane	1.25	130	Direct Fligh
5	Abu Dhabi	Masqat	bus	7	30	Oman Nati
6	Abu Dhabi	Masqat	plane	1	122	Direct flight
7	Abu Dhabi	Pretoria (U	plane	16	714	Egyptair, 1
8	Abu Dhabi	Riyadh	plane	3	172	Gulf Air, 1 s
9	Abu Dhabi	Singapore	plane	11	545	SriLankan A
10	Abu Dhabi	Washingto	plane	14	585	Direct flight
11	Abuja	Cairo	plane	16	971	http://www
12	Abuja	N'Djamena	bus	0.25	0.4	COPY: http

其中，第一列为出发城市，第二列为目的城市，第三列为交通方式，含飞机、火车、巴士三种，第四列为路程时间，第五列为路程费用，第六列为附加信息。

*此项工作得到国家自然科学基金资助，项目批准号：00000000.

根据用户所选择的起始城市和目的城市，以及其所希望的最优路径进行分析，并最终在htm文件中可视化输出。

2.2 问题分析及数据结构

针对该问题，首先需要建立世界城市网络模型，并且由于城市间往返交通属性不同，且同意线路可能具有多种方式，因此，可以使用带权有向图的模型来解决。其中，顶点即为城市，边（弧）即为两城市之间的交通线路。

由于城市数量较大，采用邻接矩阵的方法存储会消耗过多的内存，故在此采用邻接表的方式进行存储。

代码如下：

```
typedef struct {
  char country[64] = { 0 };
  char city[64] = { 0 };
double longitude, latitude;
}City,VertexType;
```

第一个结构体定义了城市信息，并重命名为City和VertexType，包含国家country，city，longitude，latitude。

```
typedef struct ArcNode{
  int adjvex;
  struct ArcNode *nextarc;
  TransKind kind[6] = { 0 };
  float time;
  int money;
  InfoType info[1024] = { 0 };
}ArcNode;
```

第二个结构体定义了边节点的信息，并重命名为ArcNode，包含邻接的顶点adjvex，下一个邻接边指针nextarc，交通方式kind，所需时间time，所需费用money以及附加信息info。

```
typedef struct VNode {
  VertexType data;
  ArcNode *firstArc;
}VNode,AdjList[MAX_VERTEX_NUM];
```

第三个结构体定义了顶点节点（头节点）的信息，并重命名为VNode以及一个顶点向量AdjList[MAX_VERTEX_NUM]。其中包含了顶点信息data，该顶点指向的第一个边节点firstarc。

```
typedef struct {
  AdjList vertices;
  int vexnum, arcnum;
}Graph;
```

最后，给出图的定义，命名为Graph，其中包含该图的顶点向量vertices，顶点个数vexnum，边个数arcnum。

2.3 创建函数

根据所定义的数据结构创建城市数组vector<City> city和城市网图Graph G.

首先，先建立城市信息数组，由于城市个数不确定，且城市数量较大，采用数组表示法并不方便，而对于STL中的vector模板可较方便的存储未知数目的元素信息，因此，本算法中采取vector存储的方式存储城市信息。

```
city.push_back(city0);
```

同时注意到，csv文件本质上为逗号分隔值得文件，即两列信息中间以逗号进行分隔，故考虑采用strtok()函数进行取值，并根据循环判断（j==0、1、2、3）每次取值应存进哪个变量中去。

```
record = strtok(buf, ",");
record = strtok(NULL, ",");
```

代码如下：

```
Status CreatCities(const char* pathname,
vector<City> &city) {
  //该函数用于制作城市表，从city.csv中读取城市信息
  FILE *fp = NULL;
  if ((fp = fopen(pathname, "rt")) == NULL) {
    printf("Cannot open the city file!");
    return ERROR;
  }
  City city0;//作为城市临时存储变量
  int i = 0, j = 0;
  char* record = NULL;
  char buf[1024];//每一行的信息缓冲区
  char* line = NULL;
  while ((line = fgets(buf, 1024, fp)) != NULL) {
    record = strtok(buf, ",");//以","作为分隔符读取信息
    while (record != NULL) {
      switch (j) {
      case 0://第一个逗号前面为country
        strcpy(city0.country, record);
        j++; break;

      case 1://第二个逗号前面为city
        strcpy(city0.city, record);
        j++; break;
      case 2://第三个逗号前面为纬度
        city0.latitude = atof(record); j++;
break;
      case 3://第四个逗号前为经度
        city0.longitude = atof(record); j = 0;
break;
      }
      record = strtok(NULL, ",");
    }
    city.push_back(city0);//将第i个城市存入city中
  }
  fclose(fp);
  return OK;
  }
```

建立好城市信息表后，将城市之间的交通信息以带权有向图的方式存入G中。由于route.csv文件中给出的是具

体的城市名，不便于在图G中存储信息，故先创建一个函数SearchCity()读取某城市在城市表city中的下标，

```
if (!strcmp(city[pos].city, des))return pos;
```

从而将图中的城市信息与city中的城市信息相对应。

代码如下：

```
int SearchCity(const char* des,vector<City>city) {
  //该函数返回某城市des在城市表city中的下标序号
  int pos;
  for (pos = 0; pos < city.size(); pos++) {
    if (!strcmp(city[pos].city, des))return pos;//找到则返回
  }
  return -1;//若未找到则返回-1
    }
```

此时根据边信息创建网图G。同样，由于routes.csv文件也是逗号分隔值文件，故采用strtok()获取值，并根据循环判断（j==0、1、2、3）每次取值应存进哪个变量中去。值得注意的是，由于j==5时，即第六列为附加信息列，分隔符应以’\n’来分隔值，

```
record = strtok(NULL, "\n");
```

否则对于多个逗号的附加信息可能会造成信息丢失。

代码如下：

```
Status CreateGraph(Graph &G,
vector<City>city,const char* pathname) {
  //该函数用于建立城市网图G，城市信息来自city，边的信息来自route.csv文件
  G.vexnum = city.size();//顶点数为city个数
  G.arcnum = 0;//边数初始为0
  for (int i = 0; i < G.vexnum; i++) {//初始化定点向量
    G.vertices[i].data = city[i];
    G.vertices[i].firstArc = NULL;
  }
  FILE* fp = NULL;
  if ((fp = fopen(pathname, "rt")) == NULL) {
    printf("Cannot open the route file!");
    return ERROR;
  }
  char* record = NULL;
  char buf[1024];
  char* line = NULL;
  while ((line = fgets(buf, 1024, fp)) != NULL) {
    record = strtok(buf, ",");
    int j = 0,v1,v2;//j用于循环计数，v1为起点，v2为终点
    ArcNode* arc = new ArcNode;//arc临时储存边信息
    arc->nextarc = NULL;
    while (j <= 6) {
      if (j == 0) {//j == 0时找到起点序号
        char temp[1024] = { 0 };//temp用于存储两个逗号之间字符串信息
        strcpy_s(temp, record);
        v1 = SearchCity(temp, city);//v1为起点下标
        record = strtok(NULL, ",");
        j++;
      }
      else if (j == 1) {//j == 1时找到终点序号
        char temp[1024] = { 0 };
        strcpy_s(temp, record);
        v2 = SearchCity(temp, city);//v2为终点下标
        arc->adjvex = v2;//该边的adjvex为v2
        record = strtok(NULL, ",");
        j++;
      }
      else if (j == 2) {//j == 2时输入交通方式
        char temp[1024] = { 0 };
        strcpy_s(temp, record);
        if (!strcmp(temp, "plane")) {
          strcpy(arc->kind, "plane");
        }
        else if (!strcmp(temp, "bus")) {
          strcpy(arc->kind, "bus");
        }
        else if (!strcmp(temp, "train")) {
          strcpy(arc->kind, "train");
        }
        record = strtok(NULL, ",");
        j++;
      }
      else if (j == 3) {//j == 3时输入所需时间
        char temp[1024] = { 0 };
        strcpy_s(temp, record);
        arc->time = atof(temp);
        record = strtok(NULL, ","); j++;
      }
      else if (j == 4) {//j == 4时输入所需费用
        char temp[1024] = { 0 };
        strcpy_s(temp, record);
        arc->money = atoi(temp);
        record = strtok(NULL, "\n"); j++;//由于后续信息为边的附加信息，所以不以逗号为分隔符，而以换行符为分隔符
      }
      else if (j == 5) {//j == 5时输入附加信息
        char temp[1024] = { 0 };
        strcpy_s(temp, record);
        strcpy(arc->info, temp);
        j++;
      }
      else if (j == 6) {//j == 6时将该边连接到v1的邻接表上去
        if (G.vertices[v1].firstArc == NULL)G.vertices[v1].firstArc = arc;//若头节点为空，则连接到头节点
        else {//若头节点不空，则连接到尾结点上去
          ArcNode *p = G.vertices[v1].firstArc;
          while (p->nextarc) {
            p = p->nextarc;
          }
```

```
            p->nextarc = arc;
            G.arcnum++;
          }
          j++;
        }
      }
    }
    fclose(fp);
    return OK;
  }
```

2.4 小结

由上述分析，我们可以得到所建立的城市网图G，城市信息表city。根据G中各点权值，可以由迪杰斯特拉算法求的最优路径。

迪杰斯特拉算法

3.1 算法概述

迪杰斯特拉算法是由荷兰科学家Dijkstra提出的，计算从一个顶点到其余各顶点的最短路径（最优路径），其解决的是网图（有权图）中的最短路径问题。

迪杰斯特拉算法的主要特点是以起始点为中心向外层层扩展，直到到达终点为止。

3.2 算法原理

首先引入辅助变量D，其每个分量D[i]表示当前所找到的由起点v0到其他顶点vi的最短路径长度。

在本例中，其定义为：

```
typedef float ShortPathTable[MAX_VERTEX_NUM];
```

值得注意的是，D的值是在不断更新并且逼近最终结果的，过程中并不一定等于最短路径。

D[i]的初始状态为从v0到vi弧的权值，如果该两顶点间无弧，则D[i] = ∞。

此时，长度为D[j] = Min{D|vi∈v}的路径就是从v0出发到vj的长度最短的一条路径，此路径为(v,vj)。

下一条长度次短的是从v0到下一个顶点的最短路径所对应的顶点，且这条最短路径长度仅次于从v0到vj的最短路径长度。

假设该路径次短路径的终点为vk，则该路径只可能为（v0，vk）或者（v0，vj，vk）。

根据以上算法过程可以证明，假设S为已求得从v0出发的最短路径长度的顶点的集合，则下一条终点为v的次短路径，其路径要么是弧（v0，v），要么是从v0出发只经过S中的若干顶点最后到达v，则该最短路径长度D[j]=Min{D[i]|vi∈v-S}，其中D[i]是弧(v,vi)的权值或D[k](vk∈S)和弧(vk,vi)权值之和的较小值。

由于需要记录最短路径，故建立一个数组，在每次求得最短路径值时，保存上一个顶点的下标值。若无上一个顶点，即为起点或不经过该点，则保存为-1。

```
int Path[MAX_VERTEX_NUM];
Path[v] = -1;
```

3.3 算法设计与算法实现

由于两个城市间的交通线路权值有多样性包括时间time和费用money，且两个城市间交通方式的多样性，例如飞机plane和火车train均能到达，则必然有最优路径的选择，故创建一个函数GetWeight()，根据用户不同的需求，返回不同的权值，如：

if (kind == 'T')，则返回时间权值

else if (kind == 'M')，则返回费用权值

且一定返回该类型的最优权值。

代码如下：

```
float GetWeight(Graph G,int v0,int v,char kind) {
  //该函数用于返回v0到v的边的权重，其中G为所用的图
  //v0为起点，v为终点，kind为'T'时则返回所需时间，为'M'时则返回所需费用
  //若从v0到v有多于一种方式，则返回权值较小的值
  //若不存在该边，则返回INFINITY
  float min = INFINITY*1.0;//最小值初始化为无穷大
  if (kind == 'T') {//求所需时间
    ArcNode *p = G.vertices[v0].firstArc;
    for (; p!=NULL; p = p->nextarc) {
      if (p->adjvex == v) {//若终点为v则判断权值是否小于min
        if (p->time < min)min = p->time;
      }
    }
    return min;
  }
  else if (kind == 'M') {//求所需费用
    ArcNode *p = G.vertices[v0].firstArc;
    for (; p != NULL; p = p->nextarc) {
      if (p->adjvex == v) {
        if (p->money < min)min = p->money;
      }
    }
      return min;
  }
    }
```

在此基础之上，进行迪杰斯特拉算法。

代码如下：

```
int Path[MAX_VERTEX_NUM];//Path[]中储存了每个点在其最短路径中上一个点的下标，若不存在上一个点，即为起点，则其值为-1
void ShortestPath(Graph G, int v0, ShortPathTable &D,char kind) {
  //该函数使用迪杰斯特拉算法求解图G中以v0为起点到达其他点的最短路径，其值存在数组D中
  //kind为所求权值的类型，'T'为时间,'M'为费用
  int v, w, i;
  float min;
  int final[MAX_VERTEX_NUM];
  //final[v]为TRUE当且仅当v0到v的最短路径已求得

    for (v = 0; v < G.vexnum; v++) {//初始化
```

```
final[], D[], Path[]
    final[v] = FALSE; D[v] = GetWeight(G, v0, v, kind);
    Path[v] = -1;
  }
  D[v0] = 0.0; final[v0] = 1;//初始化D[v0] = 0,final[v0] = 1

  //开始主循环，每次求得v0到某个v顶点的最短路径，并将final置为1
  for (i = 1; i < G.vexnum; i++) {//求其余剩下的G.vexnum - 1个顶点
    min = INFINITY*1.0;//当前所知离v0顶点的最近距离
    for (w = 0; w < G.vexnum; w++) {
      if (!final[w] && D[w] < min) { v = w; min = D[w]; }//w顶点离v0顶点更近
    }
    final[v] = TRUE;//离v0顶点最近的v的final[v]置为1
    for (w = 0; w < G.vexnum; w++) {//更新当前最短路径及距离
      if (!final[w] && (min + GetWeight(G, v, w, kind) < D[w])) {
        D[w] = min + GetWeight(G, v, w, kind);
        Path[w] = v;//记录到达w路径的上一个顶点v值
      }
    }
  }
    }
```

3.4 算法复杂度分析

Main函数中部分代码如下：

```
  vector<City>city;
  Graph G; ShortPathTable D;
  char start[32] = { 0 }, des[32] = { 0 };//start为起点城市名称，des为终点城市名称
  CreatCities("cities.csv", city);//创建城市表
  CreateGraph(G, city, "routes.csv");//创建城市网图
  char kind;//权值类型， ‘T’ 为时间， ‘M’ 为距离
  printf("Please enter the starting city:");
  scanf("%s", &start);
  getchar();//滤去换行符
  printf("Please enter the destination:");
  scanf("%s", &des);
  getchar();
  printf("Please enter 'M' for least money or 'T' for shortest time:");
  scanf("%c", &kind);
  int v0 = SearchCity(start, city);//v0为起点下标
  int v = SearchCity(des, city);//v为终点下标
    ShortestPath(G, v0, D, kind);//求最短路径
```

时间复杂度：

理论上，由时间复杂度的计算公式可得

$$f(n) = n + n + n * 3n + n * n + n + n + n * 2n * n = 2n^3 + 4n^2 + 4n$$

故时间复杂度为$T(n) = O(n^3)$。

空间复杂度：

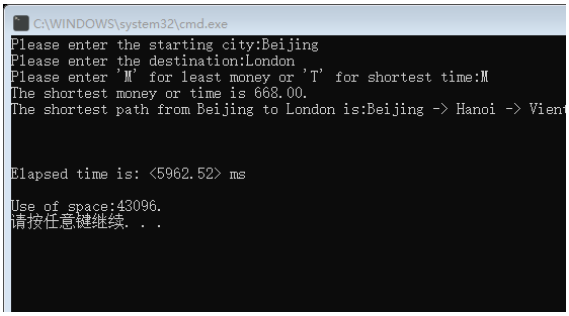
在main函数中创建了城市表city，网图G，辅助数组D，起点名数组start，终点名数组des，路径数组Path，以及其他函数中运行时的temp[1024]和buf[1024]。

```
  int size = sizeof(city) + sizeof(G) + sizeof(D) + sizeof(start) + sizeof(des) + sizeof(Path) + sizeof(char) * 1024 + sizeof(char) * 1024;
```

测试：

当由Beijing前往London费用最少的最优路径调试中，由timer.Start()和timer().Stop计算得到，在该环境下，用时为5962.52ms。

由size得到所用空间字节数为43096。



3.5 可视化输出

根据htm文件格式，可以格式化输出htm文件，以可视化的输出最短路径。

由上述分析可知，Path[]中存储的是每个节点在该路径中上一个结点的下标，即倒序存储。而我们需要顺序的输出其路径，故在此采用栈的存储方式，栈底为终点，栈顶为起点，并依次pop()出来。

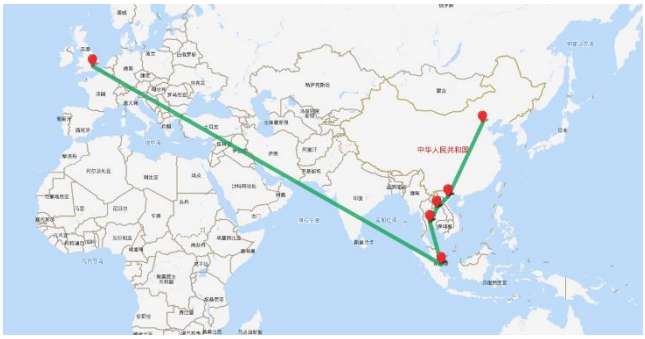
代码如下：

```
FILE* fp = fopen("test.htm", "wt");
  fprintf(fp, "<!DOCTYPE html><html><head><style type='text/css'>body, html{width: 100%%;height: 100%%;margin:0;font-family:'微软雅黑';}#allmap{height:100%%;width:100%%;}#r-result{width:100%%;}</style><script type='text/javascript' src='http://api.map.baidu.com/api?v=2.0&ak=nSxiPohfziUaCuONe4ViUP2N'></script><title>Shortest path from %s to %s</title></head><body><div id='allmap'></div></div></body></html><script type='text/javascript'>var map = new BMap.Map('allmap');var point = new BMap.Point(0,0);map.centerAndZoom(point,2);map.enableScrollWheelZoom(true);",start,des);
  int j = 0;
  stack<int>q;//由于Path[]中每个顶点储存的为上一个顶点的下标，故新建一个栈，栈底为终点，栈顶为起点
```

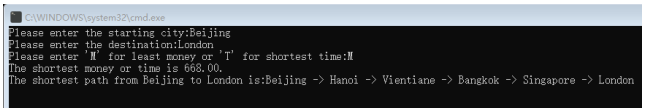
```
  int w = v;
  while (Path[w] != -1) {
    q.push(w);
    w = Path[w];
  }
  q.push(w);
  q.push(v0);
  //根据格式输出
  printf("The shortest path from %s to %s is:",
start, des);
  while (q.size() != 1) {
    int w1 = q.top(); q.pop();
    int w2 = q.top();
    ArcNode arc; GetArc(G, w1, w2, arc, kind);
    fprintf(fp, "var marker%d = new
BMap.Marker(new
BMap.Point(%.4f, %.4f));map.addOverlay(marker%d);\
n", j, G.vertices[w1].data.longitude,
G.vertices[w1].data.latitude, j);
    fprintf(fp, "var infoWindow%d = new
BMap.InfoWindow(\"<p style = 'font-
size:14px;'>country: %s<br/>city : %s</p>\");marke
r%d.addEventListener(\"click\",
function(){this.openInfoWindow(infoWindow%d);});
var marker%d = new BMap.Marker(new
BMap.Point(%.4f, %.4f));map.addOverlay(marker%d);\
n", j, G.vertices[w1].data.country,
G.vertices[w1].data.city, j, j, j+1,
G.vertices[w2].data.longitude,
G.vertices[w2].data.latitude, j+1);
    fprintf(fp, "var infoWindow%d = new
BMap.InfoWindow(\"<p style = 'font-
size:14px;'>country: %s<br/>city : %s</p>\");marke
r%d.addEventListener(\"click\",
function(){this.openInfoWindow(infoWindow%d);});
var contentString%.2d = '%s, %s --> %s, %s (%s
- %.2f hours - $%d - \"%s\")';var path%d = new
BMap.Polyline([new BMap.Point(%.4f, %.4f),new
BMap.Point(%.4f, %.4f)], {strokeColor:'#18a45b',
strokeWeight:8,
strokeOpacity:0.8});map.addOverlay(path%d);path%d.
addEventListener(\"click\",
function(){alert(contentString%.2d);});", j+1, G.ver
tices[w2].data.country,
G.vertices[w2].data.city, j+1, j+1, j+1,
G.vertices[w1].data.city,
G.vertices[w1].data.country,
G.vertices[w2].data.city,
G.vertices[w2].data.country, arc.kind, arc.time, arc.
money, arc.info, j+1, G.vertices[w1].data.longitude,
G.vertices[w1].data.latitude,
G.vertices[w2].data.longitude,
G.vertices[w2].data.latitude, j+1, j+1, j+1);
    j++;
  }
     fprintf(fp, "</script>");
```

实验结果及分析

由北京到伦敦的最少费用路径可视化输出如图所示：



其路线结果在窗口中的显示如下图：



所需费用为$668，经过路线为：Beijing→Hanoi→Vientiane→Bangkok→Singapore→London。

实习心得与体会

本次实习使我深入理解了迪杰斯特拉算法及可视化输出格式。整个代码的逻辑结构不算太复杂，但是中间有许多值得注意的地方，如：

1.城市个数较多，利用邻接矩阵法极大的浪费了空间，且时间效率也很低，故采用邻接表的方式储存；

2.城市个数的不确定，动态分配和静态数组的方式都不太合适，故采用STL模板中的vector来储存。

3.由于字符串长度的不确定，导致缓冲区字符串只能设置较大长度的字符数组来存储，空间复杂度较大；

4.还有，指针的操作需时刻值得注意，否则可能导致野指针或者读写权限有误。

虽然实习在考试周的时间，复习与实习双管齐下，有时可能无法两头兼顾，但是最终还是通过自己的努力，将实习顺利的完成，同时学到了更多的知识，将理论与实际结合解决实际问题。

致谢

感谢黄玉春老师一学期以来的教学，让我掌握了不同的数据结构，并能够根据需求设计一些算法，对编程和算法能力有了极大的提升，也对未来的学习给予了帮助。

参考文献

[1] 严蔚敏，吴伟民.数据结构[M].北京：清华大学出版社，1997.186-190.