武汉大学实习报告

院系：遥感信息工程学院　专业：遥感科学与技术　　实习时间：2023/5/2~2023/6/2

实习名称	数据结构与算法课程实习					指导教师	黄玉春
姓名	牛驰原	年级	22级	学号	2022302131042	成绩	

实习目的：
加深理解数据结构与算法关于图的创建和最短路径算法的基本原理；
通过自主学习与资料收集实现地图的可视化展示；
通过解决实际应用题目的实践巩固所学数据结构与算法课程知识；
理论学习和解决问题的实践相结合，提高编程能力，培养解决实际问题全流程的计算机思维。

实习内容及要求：
1）CSV格式数据文件的读写
2）图的创建（邻接矩阵或邻接表）
3）图的遍历（广度优先或深度优先）
4）图的最短路径，并具体给出（A到B）的最短路径及其数值
5）最短路径的地图可视化展示
6） 算法的时间复杂度分析

要求：

1、每个人必须完成1）、2）、4）三 种算法；

2、3）、5）选一个

3、按照“数据结构与算法”课程要求，进行规范的数据结构、算法、以及ADT设计，并进行算法的时间复杂度分析和实际统计，算法、代码注释清晰易读

实习方法或技术路线：
1）算法原理
i.CSV格式数据文件的读写以及邻接表的创建
CSV格式数据文件是以Excel表格为基础的一种数据文件，将表格以文本方式存储。对于CSV 文件，将表格中每行每个单元格之间用逗号隔开，每一行用\n换行，依次为规律进行文件的读写，并将其分别读入定点和边中存储。再利用已读取到的数据进行邻接表的创建。
ii.图的遍历
图的遍历分为广度遍历和深度遍历，广度遍历即对任意顶点的所有邻接点进行访问，然后再由第一个访问的邻接点继续上述操作，然后换第二个邻接点，如此反复，直到所有点都被遍历。因此 广度遍历符合先进先出原则，适宜用队列进行编写。深度遍历则是对一顶点进行访问然后访问其第 一个邻接点，在访问该邻接点的邻接点，直到无邻接点，则返回上一点继续遍历。此种遍历适合使用函数递归方式实现。
iii.图的最短路径

本程序使用的算法的基本思想：基于Dijkstra算法写一个求最短路径的函数，实现每次找到离源点最近的一个顶点，然后以该顶点为中心进行扩展，最终得到源点到终点的最短路径。

2）算法的模块化设计与实现

完成功能的1、2、3、4、5，可以设计为4部分：数据结构设计、文件的读写及创建邻接表保存数据、图的遍历、最短路径问题、地图可视化展示问题。

A.数据结构设计（结构体设计）：

site用于保存城市信息（城市序号，城市名称，国家名称，经纬度）；

cities用于保存所有顶点信息为city的指针类型；

route用于保存城市之间的联通信息（起点城市与终点城市名称，交通方式，时间，花费，以及其他信息）；

routes用于保存所有顶点的联通情况，为route的指针类型；

ArcNode为邻接表的头结点，链式储存邻接表的顶点编号，指向下个结点的指针，和路径信息（route的指针类型作为边信息）；

VNode为邻接表的表结点，储存表结点对应城市的城市信息（顶点信息），和头结点链表的首结点指针；

AdjList为表结点数组，即邻接表；

ALGraph用于储存图的基本信息（邻接表，顶点数，边数，访问辅助数组）。

Dispath用于储存最短路径的基本信息（距离数组和路径数组）。

B.文件的读写及创建邻接表保存数据：

分别对cities和routes文件以逗号和\n为分隔符读取数据并将数据保存在顶点信息和边信息的数组中，同时进行图的创建。

函数void ReadFileCities(char* strFile, cities& cts,ALGraph&G);

函数void ReadFileRoutes(char* strFile, routes& rts,ALGraph&G);

函数void CreateDG(ALGraph& G,cities cts,routes rts)。

a.参数解释：

strFile为需要读取的文件名称，cts和rts分别是顶点信息数组和边信息数组，G为图的类型储存数据。

b.读取方式：

将’,’作为分隔符，读取字符串时用字符一个一个地读直至出现’,’字符,读取int或者double型数据时，直接使用fscanf读取同时再使用一次吃掉‘,’字符。

例如：

```
//Read country
char c;
fscanf(fp, "%c", &c);
while (c != ',')
{
        cts[i].country[j] = c;
        fscanf(fp, "%c", &c);
        j++;
}
cts[i].country[j] = '\0';
```

```
            j = 0;

            //Read latitude and longitude
            fscanf(fp, "%lf", &cts[i].latitude);
            fscanf(fp, "%c", &c);
            fscanf(fp, "%lf\n", &cts[i].longitude);
```

routes.csv文件中的数据类型也为以上两种，读取方式也基本相同；

c.读取数据保存到图中：

在前两个函数中使用临时变量储存遍历得到的顶点数和边数给G中的数据结构赋值。第三个函数则存储顶点信息和边信息，同时初始化访问辅助数组，主要使用循环遍历的方式。

例如：

```
    //Store vertices
    for (i = 0; i < G.vexnum; i++)
    {
            strcpy(G.vertices[i].data.city, cts[i].city);
            strcpy(G.vertices[i].data.country, cts[i].country);
            G.vertices[i].data.latitude = cts[i].latitude;
            G.vertices[i].data.longitude = cts[i].longitude;
            G.vertices[i].firstarc = NULL;
            G.ved[i] = 0;//Initialize the auxiliary array
    }
```

而储存边信息的时候，需要创建各表结点的链表，采用了逆向创建链表的方式。

如下：

```
    //Store arc
    for (i = 0; i < G.arcnum; i++)
    {
            Locate(G,rts[i].depcity,depvex);
            Locate(G,rts[i].descity,desvex);
            //Reverse Creation of the chain tables
            ArcNode* arc = new ArcNode;
            arc->adjvex = desvex;
            arc->rt =(rts+i);
            arc->nextarc = G.vertices[depvex].firstarc;
            G.vertices[depvex].firstarc = arc;
    }
```

C.图的深度遍历：

i.遍历原理：

是对一顶点进行访问然后访问其第一个邻接点，再访问该邻接点的邻接点，直到无邻接点，则返回上一点继续遍历。若该顶点所有能连接到的点均被遍历，但图为完全遍历，则在从另一未被访问的顶点继续进行深度遍历。由于深度遍历特性，宜采用递归方式实现。

ii.实现对一个顶点的深度遍历：

v为遍历起点，visited数组为是否访问过。遍历的过程中使用printf函数将访问过的城市打印出来。

如下：

```
void DFS(ALGraph &G, int v)
{
    printf("%s-->", G.vertices[v].data.city);
    int w;
    G.ved[v] = TRUE;
    //对顶点v的链表依次遍历 同时每次指针后移就进行dfs递归
    ArcNode* p;
    p = G.vertices[v].firstarc;
    while (p)
    {
       w = p->adjvex;
       if (!G.ved[w])
            DFS(G, w);
       p = p->nextarc;
    }
}
```

iii.实现对整个图的深度遍历：

可能为非连通图 所以要对每个顶点进行dfs遍历。

如下：

```
void DFSTrasval(ALGraph &G)
{
    //可能为非连通图 所以要对每个顶点进行dfs遍历
    for (int i = 0; i < G.vexnum; i++)
    {
            if (!G.ved[i])
                    DFS(G, i);
    }
    printf("END\n");
}
```

D.最短路径问题：选择模式，找到最低费用路径或者最短时间路径，并进行输出。

函数`void Dijkstra1(ALGraph G, int start, Dispath& dp);`

函数`void printfShortpaths1(ALGraph G, Dispath& dp,int end)`

参数解释：G为图，start为始发城市在图中对应的顶点序号，end为终点城市在图中对应的顶点序号。dp为存储最短路径的二维数组。

基本步骤如下：

a.初始化距离为无穷，辅助最短路径数组为0

如下：

```
for (i = 0; i < G.vexnum; i++)            // 1. 初始化距离为无穷，辅助最短路径数组为0
    {
            dp.dis[i] = INFINE;
            dp.path[i][0] = 0;
    }
```

dp的dis数组表示从始发城市到各地的最短路径的距离（初始化为无穷）INFINE为头文件中的宏定义`#define INFINE DBL_MAX`。而path二维数组非常巧妙，第一维表示初始城市到各地的路径，第二维度表示具体的路径（用顶点数来记录），其中第1个元素path[i][0]代表是否找到最短路径，如果找到则将该值更改为1，便于后续查找最短路径。

b.遍历start链，初始化一些dis和path

由于Dijkstra算法思想是基于每次找到离源点最近的一个顶点，然后以该顶点为中心进行扩展，因此先将离源点最近的那些点存入dp中去。

如下：

```
ArcNode* p;                                    // 2. 遍历start链，初始化一些dis和path
    p = G.vertices[start].firstarc;
    while (p)
    {
            dp.dis[p->adjvex] = p->rt->time;
            dp.path[p->adjvex][1] = start;
            dp.path[p->adjvex][2] = p->adjvex;
            dp.path[p->adjvex][3] = -1;
            p = p->nextarc;
    }

  dp.path[start][0] = 1;//以后不再会用到path[start]
```

将第4个元素赋予-1的原因是将其作为一个判断信息，便于后面查找更新最短路径的使用。

c.找最短路径

大致思路是通过遍历dp的dis数组找到最短的路径并记录最短路径的终点城市对应序号

如下：

```
 double mindis = INFINE;                              // 3. 找最短路径
    int k;
    for (j = 0; j < G.vexnum; j++)
    {
            if (dp.path[j][0] == 0 && dp.dis[j] < mindis) {
                    k = j;
                    mindis = dp.dis[j];
            }
    }

    dp.path[k][0] = 1;//找到了start到k的最短路
```

d.更新路径

根据第三步找到的最短路径对应的序号，以它为中心进行扩展，将离它最近的那些点与该最短路径相连并于dp数组中的dis值进行比较并更新。

如下：

```
    p = G.vertices[k].firstarc;              // 4. 更新dis和path
    while (p)
    {

            if (dp.path[p->adjvex][0] == 0 && dp.dis[p->adjvex] > dp.dis[k] + p->rt->time)
            {
                    dp.dis[p->adjvex] = dp.dis[k] + p->rt->time;//更新长度dis

                    int t = 1;//更新路径path 先把path[k]的拷贝给path[p->adjvex] 再给path[p-
>adjvex]添上其本身和结束标志
                    while (dp.path[k][t] != -1)
                    {
                            dp.path[p->adjvex][t] = dp.path[k][t];
                            t++;
```

```
                    }
                    dp.path[p->adjvex][t] = p->adjvex;
                    dp.path[p->adjvex][t + 1] = -1;
                }

                p = p->nextarc;

            }
```

e.循环第三步和第四步直到在连通的图中已经找齐了start到各个点的最短路径，即

```
 if (mindis == INFINE)return;
```

通过以上五步便实现了以时间作为决定变量的最短路径的算法，同样也可以将花费作为决定变量来求解最短路径。

f.通过遍历dp数组来实现最短路径的输出。

如下：

```
        //若到end结点城市没有路径
        if (dp.dis[end] == INFINE) {
                printf("To City: %s  Can not go to the city!\n", G.vertices[end].data.city);
                return;
        }
        //若到end结点城市已找到最短路径
        else {
                printf("To City: %s  Time: %.2lf hours  Path: %s", G.vertices[end].data.city,
dp.dis[end], G.vertices[dp.path[end][1]].data.city);
                //输出路径 以->连接各城市
                for (int j = 2; dp.path[end][j] != -1; j++)
                {
                        printf("->%s", G.vertices[dp.path[end][j]].data.city);
                }
                printf("\n");
                return;
        }
```

分为两种情况：到达终点城市没有路径和有最短路径，进行控制台的输出即可。

E.地图可视化展示:

a.实现方式：

以所给示例为模板提炼出html文件实现可视化的代码模板，然后根据所求最短路径信息写入htm文件实现可视化。

b.写htm文件基本步骤：

1.创建地图，添加基本信息；

2.创建坐标点和路径标记，添加备注；

3.添加末尾语句使文件代码完整。

c.本算法中的函数用于写htm文件的主循环中还实现了最短路径的黑窗口输出，由于html文件写入部分的代码过长且重复，因此报告中只介绍思路不做展示。

```
 void Generatehtml(ALGraph G,Dispath&dp,int start,int end,char *strFile,cities cts)
```

参数解释：FileName:html文件名；G:图；dp:起点到给点的最短路径；start:起点城市序号；end：终点城市序号；cts：顶点信息

主循环：创建该段路径起点、终点的图上标记，以及路径本身的标记；写入htm文件

中。

d.写入htm文件时需要注意的细节：

1.<>之间不能有任何隔断符号包括<>内的形式也必须严格按照要求来写；

2.打印的时候要注意出现的 ’ ″ 等符号与程序的部分语言所引起的冲突，因此手动修改fprintf里涉及到此类符号的内容，并在每次填入字符串的时候使用supple函数进行补\的操作。（例如带有’符号的国家Lao People’s Democratic）

Supple函数如下：

```
char* Supple(char* c)//某些城市或者info等字符串会出现 ’ 的字符 需要在其前面加上’\’
{
	int i = 1;
	int n = strlen(c);
	//读文件的时候就已经将字符串的下一位设置成了’/0’ 因此可以将此作为字符串遍历的终止条件
	while (c[i] != ’\0’)
	{
		//找到 ’ 字符
		if (c[i] == ’\’’) {
			//若它的上一位已经是 \ 字符就跳过本次循环 进行下次循环
			if (c[i - 1] == ’\\’) {
				i++; continue;
			}
			//将字符串增加一位 并从第i位往后移 且将第i位改为 \ 字符
			c[n+1] = ’\0’;
			for (int j = n; j > i; j--){ c[j] = c[j - 1]; }
			c[i] = ’\\’;
			n++;
			i++;
		}
		i++;
	}
	return c;
 }
```

由于设计的Supple函数本身即为字符指针类型的，因此在fprintf将字符串带入模%s的时候对字符串使用Supple函数进行补符号的操作。

例如：

```
fprintf(fp, ″var infoWindow%d = new BMap.InfoWindow(\″<p style = ’font-
size:14px;’>country: %s<br/>city : %s</p>\″); marker%d.addEventListener(\″click\″, function ()
{ this.openInfoWindow(infoWindow%d); });″, i - 1, Supple(cts[dp.path[end][i]].country),
Supple(cts[dp.path[end][i]].city), i - 1, i - 1);
```

表示标记对应的信息窗的申请和应用（点击标记事件），其中对fpritf部分符号前面手动补充\并且对字符串使用Supple函数进行补符号的操作。

3.可能会出现到达终点城市没有路径的情况

此时需要额外打一个if补丁，出现这种情况时为了防止fprintf无效数据，只进行标记和标记信息窗的创建，并不能创建路线和路线信息窗。

F.主函数实现步骤：

1.创建结构体变量存储城市和路径的数据

```
//创建结构体变量存储城市和路径的数据
```

```
cities cts;
cts = new site[MAX_VERTEX_NUM];
routes rts;
rts = new route[MAX_ARC_NUM];
ALGraph G;
```

2. 从文件中读取城市和路径的数据并存入结构体中

```
char strFile1[256] = "D:\\DataStructure\\Program_1\\data\\cities.csv";
char strFile2[256] = "D:\\DataStructure\\Program_1\\data\\routes.csv";
char strFile3[256] = "D:\\DataStructure\\Program_1\\data\\Graph.htm";
ReadFileCities(strFile1, cts,G);
ReadFileRoutes(strFile2, rts,G);
```

3. 创建有向图

```
//创建有向图
CreateDG(G, cts, rts);
```

4. 深度优先遍历图

```
//深度优先遍历图
DFSTrasval(G);
```

5. 选择搜索方式，出发地与目的地

```
//初始化出发地与目的地的变量
char stcity[64]; char tocity[64]; int nstart; int nend; int model=0;
memset(stcity,0,sizeof(stcity));
memset(tocity, 0, sizeof(tocity));
//读取搜索方式 同时用getchar吃掉换行符号
printf("请选择您想要的搜索方式：输入1为最短时间，输入2为最少花费\n");
scanf("%d", &model); getchar();
//读取始发城市（因可能出现带空格字符的城市 所以采用这种方式代替%s）
//同时用getchar吃掉换行符号
printf("请输入您的出发地:");
scanf("%[^\n]", stcity);getchar();
//同上 读取终达城市
printf("请输入您的目的地:");
scanf("%[^\n]", tocity);getchar();
//找到城市所对应的结点
Locate(G, stcity, nstart);
Locate(G, tocity, nend);
```

6. 根据对应模式找最短路径

```
//根据对应模式找最短路径
Dispath dp;
if (model == 1) {
    Dijkstra1(G, nstart, dp);
    printfShortpaths1(G, dp, nend);
}
if (model == 2) {
    Dijkstra2(G, nstart, dp);
    printfShortpaths1(G, dp, nend);
}
```

7. 创建htm文件 使路径可视化

```
//创建htm文件 使路径可视化
Generatehtml(G, dp, nstart, nend, strFile3,cts);
```

3）算法复杂度分析与实测结果

A.算法复杂度分析——时间复杂度：

i.void ReadFileCities(char* strFile, cities& cts,ALGraph&G)单层for循环

void ReadFileRoutes(char* strFile, routes& rts,ALGraph&G)单层for循环

void CreateDG(ALGraph& G,cities cts,routes rts)两个单层for循环

则T总(n) = O(n);

ii.void Locate(ALGraph G,char* city, int& vex);

route* Locatert(ALGraph G,int start, int end);

两个个函数以遍历的方式根据城市名称获取城市相关信息，均为单层循环

则T总(n) = O(n);

iii.void DFSTrasval(ALGraph &G)

void DFS(ALGraph &G, int v)

深度遍历函数为一层for循环加递归

则T总(n) = ∑Ti(n) = O（n^2）;

ivvoid Dijkstra1(ALGraph G, int start, Dispath& dp)

初始化标识变量的值，一层for循环 主循环：一层for循环中存在两个for循环

void printfShortpaths1(ALGraph G, Dispath& dp,int end)一个while循环 则T(n) = O(n);

则T总(n) = ∑Ti(n) = O（n^2）;

v. void Generatehtml(ALGraph G,Dispath&dp,int start,int end,char *strFile,cities cts)

本身遍历为一层循环，但由于字符串需要进行supple函数进行遍历，故存在两个for循环

则T总(n) = ∑Ti(n) = O（n^2）;

vi. 综上，主函数：

ReadFileCities()调用1次：T1(n) = O(n);

ReadFileRoutes()调用1次：T2(n) = O(n);

CreateDG()调用1次：T3(n) = O(n);

DFSTraverse()调用1次：T4(n) = O(n^2);

Dijkstra1（）或 Dijkstra2（）调用1次：T5(n) = O(n^2);

printfShortpaths1()或printfShortpaths2()调用1次：T6(n) = O(n);

Generatehtml()调用1次：T7(n) = O(n^2)

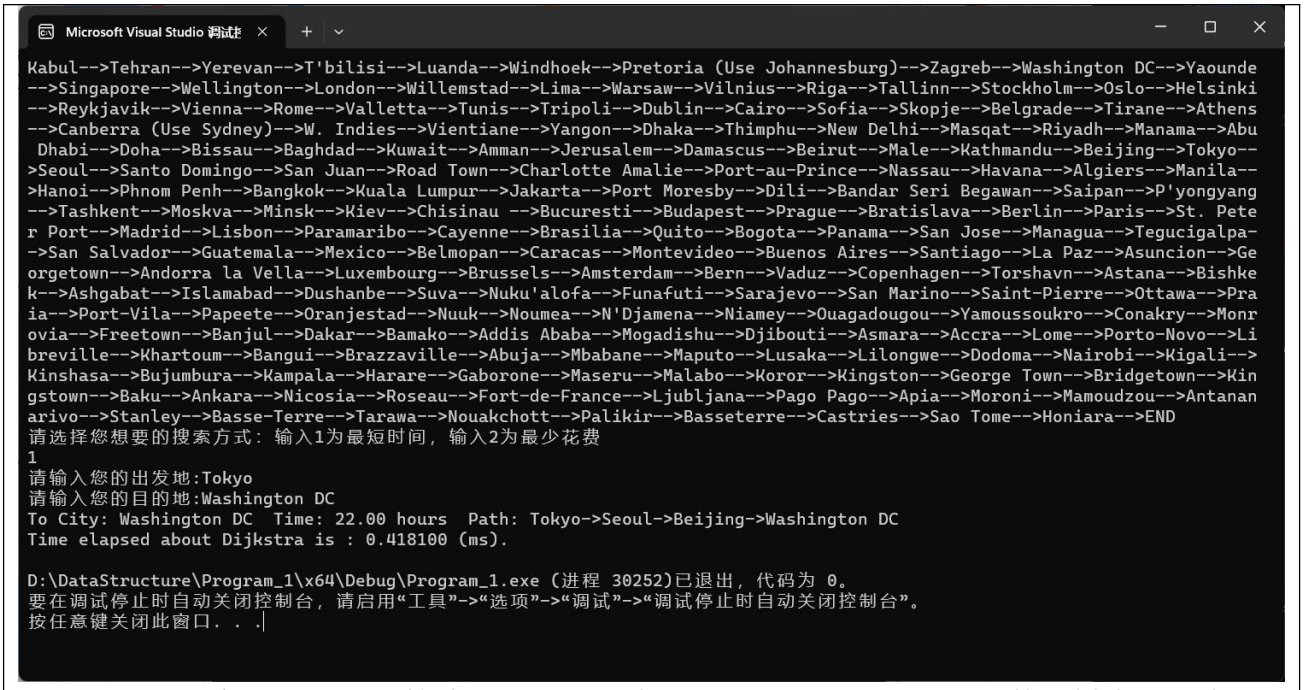
则T总(n) = ∑Ti(n) = O(n^2)

B.实测结果

示例：

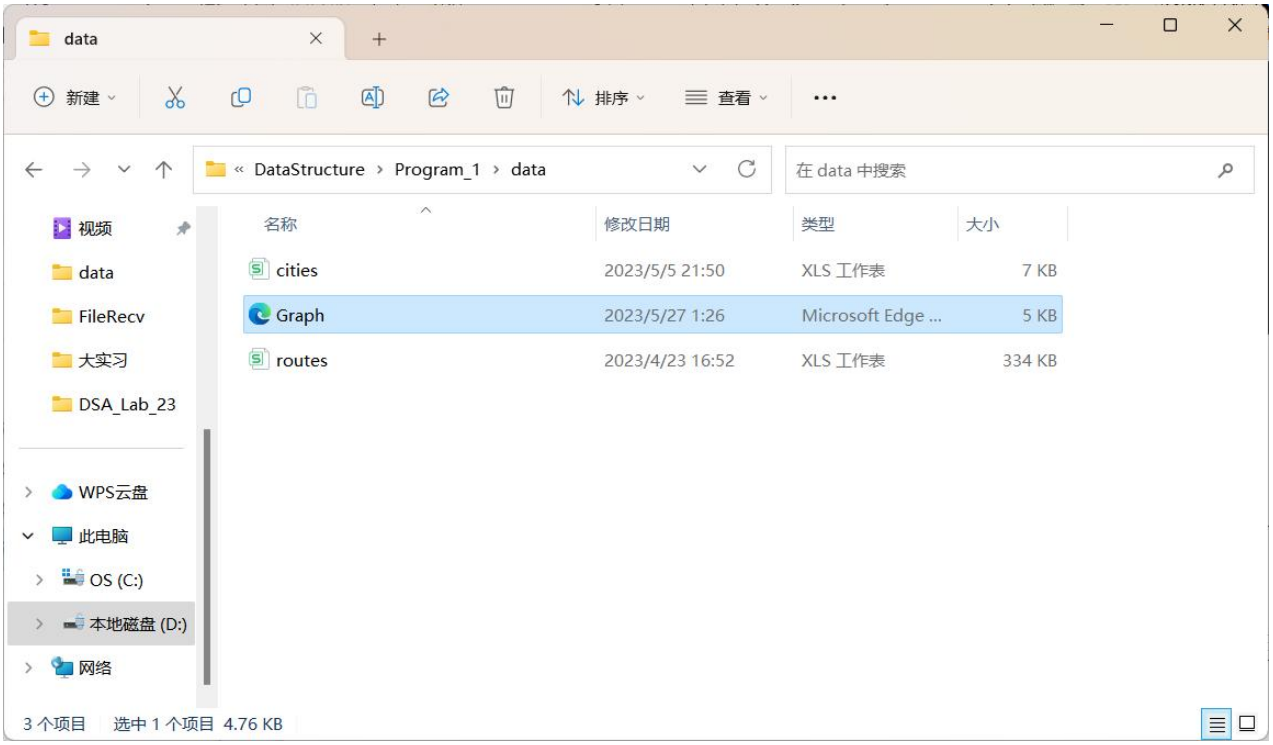
查找（Tokyo）到（Washington DC）（时间最少）路径

1.控制台输出结果



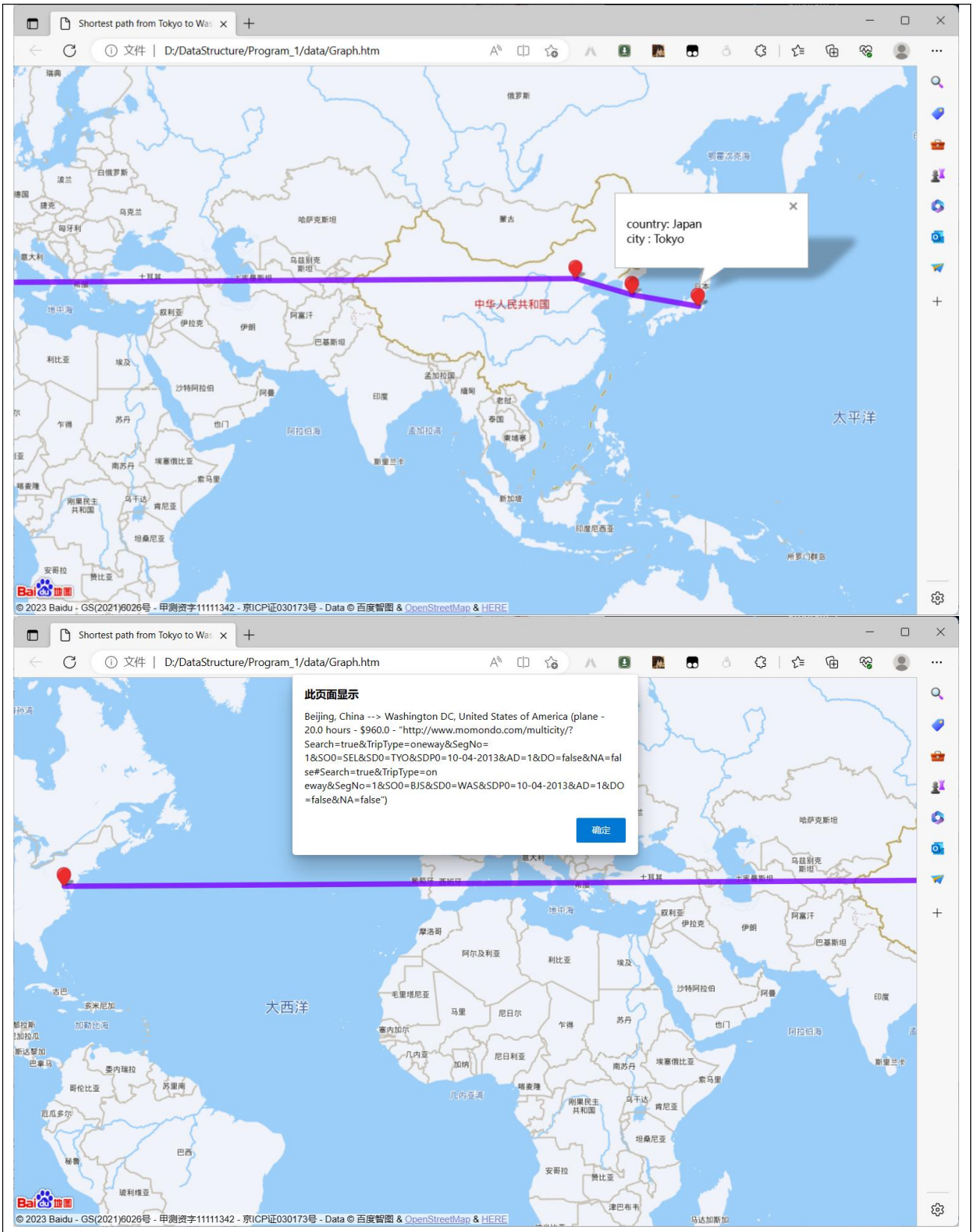
经多次统计，实现最短路径算法的时间平均为0.45ms，由于城市和路径数据样本不够大的原因，导致影响时间的主要因素是文件读写而不是最短路径算法。

2.文件路径展示：



D:/DataStructure/Program_1/data/Graph.htm

3.可视化成果展示：



除了实现示例中标记点和路线的创建，还进行了样式和颜色的更改。

实习结论： 经过这半学期的数据结构理论知识的学习以及实习，我收获了很多。首先是更清晰地了解了和图有关的数据结构和各种基础算法，比如邻接表、Dijkstra算法、Floyd算法…等等，让我对编程和算法有了更深层次的认识。另外，在本次实习中我通过尝试、发现问题、解决问题，比较典型的是可视化中输出文件需要在某些地方添加\转义符号，还有怎样实现对带有空格的字符串进行输入。发现了很多掌握不够熟练的知识点，也总结出一些课本上可能学不到的实战技巧，例如遇到怎么样的问题应该怎样应对，在不停的打断点调试，监视变量值的变化，慢慢的摸索。然后，我在本次实习中接触到了地图可视化的知识，虽然初次尝试遇到很多不会解决的问题，但在老师和同学的帮助下，遇到的问题都顺利解决。通过实习，我也明白了其实知识的掌握总是循序渐进的过程，每一种知识都不例外，只有多思考、练习、扩展知识面，才能慢慢地发现其中更深的奥妙，提高学习知识的兴趣。
教师评语 指导教师_____ 2020年_____月_____日