

《数据结构与算法分析》

实验报告

实验名称 基于 Dijkstra 算法的最短路径求解

姓名 曾庆文

学号 23060218

院系 自动化学院

专业 人工智能

班级 23061011

实验时间 2024.5

一、实验目的

1. 掌握图的邻接矩阵表示法，掌握采用邻接矩阵表示法创建图的算法。
2. 掌握求解最短路径的 Dijkstra 算法。

二、实验设备与环境

编辑器：Visual Studio Code

编译器：gcc

三、实验内容与结果

基本操作的输入输出结果如下：

1. 测试样例：

```
6 8
A B C D E F
A F 100
A E 30
A C 10
B C 5
C D 50
D E 20
E F 60
D F 10
A F

70
A C D F

0 0

请按任意键继续 . . . |
```

输入

输出

终止程序

实验全部代码如下：

```
1. #include<stdio.h>
2. #include<stdbool.h>
3. #define MaxInt 32767 //表示两个顶点相距无穷远
4. #define MVNum 100
5.
6. //图的邻接矩阵存储表示
7. typedef struct
8. {
9.     char vexs[MVNum];
10.    int arcs[MVNum][MVNum];
11.    int vexnum, arcnum;
12. }AMGraph;
13.
14. //迪杰斯特拉算法
15. void ShortestPath_DIJ (AMGraph G, char start, char end)
16. {
17.     int v0 = start - 'A' + 0;
18.     int n = G.vexnum;
19.         //图的顶点个数
20.     int v;
21.     bool S[n];
22.         //源点到终点是否确定最短路径长度
23.     int Path[n];
24.         //最短路径直接前驱顶点的序号
25.     int D[n];
26.         //源点到终点的最短路径长度
27.     for (v = 0; v < n; v++)
28.     {
29.         S[v] = false;
30.         //初始为空集
31.         D[v] = G.arcs[v0][v];
32.         //初始化为弧的权值
33.         if (D[v] < MaxInt)
34.             //两点间有弧
35.             Path[v] = v0;
36.         else
37.             //两点间无弧
38.             Path[v] = -1;
39.     }
40.     S[v0] = true;
41.     D[v0] = 0;
42.         //源点到源点距离为0
43.     /*-----初始化结束，开始主循环-----*/
```

```

35.     for (int i = 1; i < n; i++)
36.     {
37.         int min = MaxInt;
38.         for (int w = 0; w < n; w++)
39.             if (!S[w] && D[w] < min)
40.                 // 若当前有最短路径
41.                 {
42.                     v = w;
43.                     // 更新中间点
44.                     min = D[w];
45.                     // 更新最短路径
46.                 }
47.         S[v] = true;
48.         // 中间该店有最短路径
49.         for (int w = 0; w < n; w++)
50.             // 更新源点到某地的最短路径
51.             if (!S[w] && D[v] + G.arcs[v][w] < D[w])
52.                 {
53.                     D[w] = D[v] + G.arcs[v][w]; // 更新最短路径
54.                     Path[w] = v; // 更改直接前驱
55.                 }
56.     }
57.
58. /*-----最短路径寻找完毕，开始输出-----
59. */
60.
61.     int endToStart = end - 'A' + 0;
62.     // 从终点开始回溯到起点
63.
64.     int city = 1;
65.     // 最短路径经过的城市数
66.
67.     char pathCity[MVNum];
68.     // 最短路径经过的城市
69.
70.     pathCity[0] = G.vexs[endToStart];
71.     // 终点城市
72.
73.     printf ("%d\n", D[endToStart]);
74.     // 输出最短路径值
75.
76.     for (int i = 1; i < MVNum; i++)
77.     {
78.         if (endToStart == v0)
79.             // 如果到达起点，则退出循环
80.             break;
81.
82.         city++;
83.
84.         endToStart = Path[endToStart];
85.         // 从终点开始往前回溯
86.
87.         pathCity[i] = G.vexs[endToStart];
88.     }

```

```

66.     for (int i = city - 1; i >= 0; i-
        -)                                //倒序输出
67.     {
68.         printf("%c", pathCity[i]);
69.         if(i > 0)
70.             printf(" ");
71.         else
72.             printf("\n");
73.     }
74. }
75.
76. // 程序主函数
77. int main ()
78. {
79.     AMGraph G;
80.     char ch1, ch2;
81.         // 两个字符变量在循环中将暂存某些值
82.     char start, end;
83.         // 起点城市和终点城市
84.     while (1)
85.     {
86.         scanf ("%d%d", &G.vexnum, &G.arcnum);
87.         // 城市数和路径数
88.         getchar();
89.         if (G.vexnum == 0 && G.arcnum == 0)
90.             // 都为0 则退出循环
91.             break;
92.         for (int i = 0; i < G.vexnum; i++)
93.         {
94.             scanf ("%c", &G.vexs[i]);
95.             // 每个城市的名称
96.             getchar();
97.         }
98.         for (int i = 0; i < G.vexnum; i++)
99.             for (int j = 0; j < G.vexnum; j++)
100.                 G.arcs[i][j] = MaxInt;
101.         // 先将所有路径初始化为最大值
102.         for (int i = 0; i < G.arcnum; i++)
103.         {
104.             scanf ("%c", &ch1);
105.             // 城市1
106.             getchar();
107.             scanf ("%c", &ch2);
108.             // 城市2

```

```

101.         getchar();
102.         scanf("%d", &G.arcs[ch1 - 'A' + 0][ch2 - 'A'
+ 0]); //城市1、2 间的路径的权值
103.         getchar();
104.     }
105.     scanf ("%c %c", &start, &end);
//起点和终点
106. /*-----输入结束，开始求解最短路径-----
-----*/
107.     ShortestPath_DIJ (G, start, end);
108. }
109. return 0;
110. }

```

四、实验总结及体会

本次实验对我个人来说有以下几个难点：

1. 数据的输入：由于本次实验要输入比较多的数据，因此正确处理缓冲区就很重要。为确保缓冲区的空格符和换行符能够正确处理，每输入一次字符，程序就使用一次 `getchar()` 函数，确保不会输入错误。
2. 迪杰斯特拉算法的书写：迪杰斯特拉算法是本次实验的关键。程序先定义了三个辅助数组 `S[n]`，`Path[n]`，`D[n]`，为了保证算法能够正确完成，先将三个辅助数组初始化，再进行循环求解最短路径。每次循环后及时更新最短路径和直接前缀也是求解的关键。
3. 最短路径结果的输出：先定义一个足够大的字符数组，用于存储最短路径所经过的城市。利用 `Path` 数组不断往前回溯，找到最短路径的直接前驱，并将其储存在字符数组中。需要注意的是回溯是从后往前回溯，而实验的要求的从起点城市按照最短路径输出到终点，因此还要将字符数组的结果倒序输出，才能得到正确输出。
4. 代码的分块书写与 `debug`：因为本次实验的函数比较多，所以我在每写完一个函数，就单独检测其是否能正确运行并得到正确结果。这个习惯大大减少了我 `debug` 的时间。