



HARBIN INSTITUTE OF TECHNOLOGY

哈尔滨工业大学

集合论与图论实验报告

实验项目名称: Dijkstra 算法求最短路径 成绩评定_____

实验项目类型: 研究型 实验时间: 2017/05/20

学生姓名: 班级: 学号:

学院: 计算机学院 专业: 软件工程 任课教师: 刘峰

(一) 实验背景与意义

最短路径问题是图论研究中的一个经典算法问题，旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。算法具体的形式包括：

确定起点的最短路径问题 – 即已知起始结点，求最短路径的问题。[1]

确定终点的最短路径问题 – 与确定起点的问题相反，该问题是已知终结结点，求最短路径的问题。在无向图中该问题与确定起点的问题完全等同，在有向图中该问题等同于把所有路径方向反转的确定起点的问题。

确定起点终点的最短路径问题 – 即已知起点和终点，求两结点之间的最短路径。
全局最短路径问题 – 求图中所有的最短路径。

在现实生活中，很多地方需要求出两个地区之间的最短路径，例如寄信，运送货物等，但两地之间的路径往往不止一条，可能直达，可能途径中转城市以及需要考虑中转时的过路费等问题，最短路径显得十分重要。本程序应用迪杰克斯拉算法，仅考虑两地之间的距离而不考虑过路费，计算两地之间的最短距离。

实验内容

问题描述：

1.自然语言描述

假设在几个景点之间有一至多条路径，显然这些路径是无向的，但这些路径的长度不一。假设游客在其中一个城市 A，求一条最短路径可以让游客到达 B 城市。

2.数学语言描述

设图为 $G(V, E, g)$ ，其中 g 是 E 上的非负权函数。若求解 $A1, A2 \in V$ 中的最短路径 $(A1, A2)$ ，则将任意联结 $A1$ 、 $A2$ 的路径 $E1$ 、 $E2$ 、..... E_k 进行比较，若 E_i 小于其他所有路径，则 E_i 为 G 中的 $A1$ 到 $A2$ 的最短路径。

算法实现：

1. Dijkstra 算法的基本思想：

Dijkstra 算法采用的是贪心法的算法策略

大概过程：

创建两个表，OPEN，CLOSE。

OPEN 表保存所有已生成而未考察的节点，CLOSED 表中记录已访问过的节点。

1. 访问路网中距离起始点最近且没有被检查过的点，把这个点放入 OPEN 组中等待检查。
2. 从 OPEN 表中找出距起始点最近的点，找出这个点的所有子节点，把这个点放到 CLOSE 表中。
3. 遍历考察这个点的子节点。求出这些子节点距起始点的距离值，放子节点到 OPEN 表中。
4. 重复第 2 和第 3 步，直到 OPEN 表为空，或找到目标点。

(二) 实验环境

操作系统: Win10

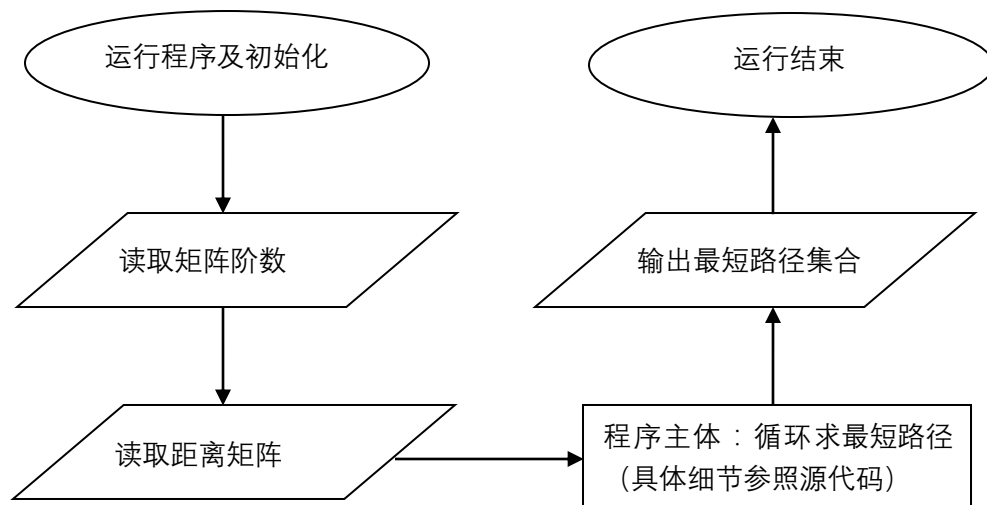
编译器: Visual Studio Community 2015

(三) 程序设计

数据结构设计:

序号	字段名	字段含义	类型	长度	默认值	说明
1	Road[][]	距离矩阵	Int[][]	32 位	0	记录各城市之间距离
2	min_road[]	最短路径	Int[]	32	无	城市 1 到个城市最短距离

程序流程图:



源程序的全部代码:

文件: 源.c

```

001 #include <stdio.h>
002 #include <stdlib.h>

003 void Get_Min(int a[], int n, int *point);
004 void Dijkstra(int a[][50], int n, int b[]);
005 int main()
006 {
  
```

```
007 int road[50][50] = {0}, min_road[50];
008 int n, i, j;

009 printf("input n:\n>>> "); scanf("%d", &n);
010 for (i = 0; i<n; i++)
011 {
012     for (j = 0; j<n; j++)
013     {
014         road[i][j] = 10000;
015     }
016     road[i][i] = 0;
017 }
018 printf("input the distance between two cities, and end of -1:\n>>> ");

019 while (scanf("%d", &i)&&i>0) {
020     scanf("%d %d", &j, &road[i-1][j-1]);
021     road[j-1][i-1] = road[i-1][j-1];
022 }

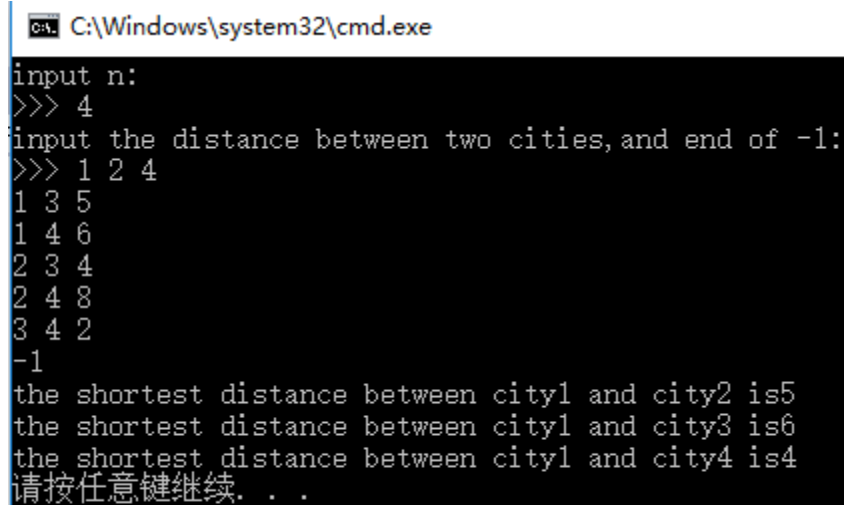
023 for (i = 0; i<n; i++)
024 {
025     min_road[i] = road[0][i];
026 }
027 Dijkstra(road, n, min_road);
028 for (j = 1; j<n; j++)
029     printf("the shortest distance between city1 and city%d is %d\n", j+1, min_road[j]);
030 return 0;
031 }

032 void Get_Min(int a[], int n, int *point)
033 {
034     int i, j, min = 10000;
035     for (i = 0; i<n; i++)
036     {
037         for (j = 0; j<n; j++)
038         {
039             if (a[i]>0 && a[i]<min)
040             {
041                 min = a[i];
042                 *point = i;
043             }
044 }
```

```
045 }
046 }

047 void Dijkstra(int a[][50], int n, int b[])
048 {
049     int i, j = 0, k, point, count;
050     while (1)
051     {
052         count = 0;
053         for (i = 0; i < n; i++)
054         {
055             b[i] = b[i] < a[j][i] + b[j] ? b[i] : a[j][i] + b[j];
056             a[i][j] = 10000;
057         }
058         Get_Min(a[j], n, &point);
059         j = point;
060         for (k = 0; k < n; k++)
061         {
062             if (a[j][k] != 10000)
063                 break;
064             else
065                 count++;
066         }
067         if (count == n)
068             break;
069     }
070 }
```

（五）运行结果及评价



```
C:\Windows\system32\cmd.exe
input n:
>>> 4
input the distance between two cities, and end of -1:
>>> 1 2 4
1 3 5
1 4 6
2 3 4
2 4 8
3 4 2
-1
the shortest distance between city1 and city2 is5
the shortest distance between city1 and city3 is6
the shortest distance between city1 and city4 is4
请按任意键继续. . .
```

Dijkstra 算法:

时间复杂度: 由于在一个外层循环内有两个独立的 n 次循环, 所以时间复杂度为 $O(n^2)$

空间复杂度: 由于不存在递归, 所以空间复杂度为 $O(1)$

1. 结论

Dijkstra 算法通过先把初始点与其他各点分开, 在通过一步一步的比较逐步将另外一个集合中的点放入初始点集合 S 中, 并且更新每个点到最短路径点的距离从而求得最短路径。但是, 该算法复杂度为 n^2 , 我们可以发现, 如果边数远小于 n^2 , 对此可以考虑用堆这种数据结构进行优化, 取出最短路径的复杂度降为 $O(1)$; 每次调整的复杂度降为 $O(e \log n)$; e 为该点的边数, 所以复杂度降为 $O((m+n) \log n)$ 。