设计简单计算机网络结构，用Dijktra算法求各终端的路由

摘 要 本课程设计主要解决计算机网络中，路由器应该如何选择转发路径使得所经过的 路径最短。运用Dijktra算法求出一源点到其他所有节点的最短路径，主要特点是以起始点 为中心向外层层扩展，直到扩展到终点为止。并计算出最短路径的长度，以及该最短路径 的线路。课程设计及中，系统开发平台为Windows XP，程序设计语言为Visual C++6.0程 序运行平台为Windows 98/2000/XP。在程序设计中采用了邻接矩阵和镀铬数组相结合的方 法实现了对网络中结点和路径的存储和运算。程序通过调试运行，初步实现了设计目标， 经过调试和完善后，将可以运用在实际中解决问题。

关键词 程序设计；计算机网络；Visual C++6.0； Dijkstra算法；最短路径

目录

1引言

本课程设计主要解决计算机网络结构中路由器路径的选择，路由器路径的选择对于现在计算机网络数据传输有着重要的作用。提高网络传输速度的关键是找到最佳路径，Dijkstra算法是找到最短路径的方法之一，运用Dijkstra算法找到最短路径，可以大大提高 网络的传输速度和网络的利用率。

1.1课程设计背景

计算机在我们的生活中扮演着越来越重的角色。我们的生活的因为计算机变得更精 彩。我们知道，21世纪是一个以网络为核心的信息时代。要实现信息化就必须依靠完善的 网络，因为网络可以非常迅速地传递信息。

计算机网络，是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统，网络管理软件及网络通信协议的管理和协议下，实现资源和信息传递的计算机系统。路由器将不同网络连接起来，另外，路由器是信息选择传送的路线。选择通畅快捷的近路，能发发提高通信速度，减轻网络系统通信负荷，节约网络系统资源，提高网络系统畅通率。从而让网络系统挥发出更大的效益出来。路由的主要工作是经过路由器的每个数据帧寻找一条最佳传输路径，并将数据有效地传送到目的站点，由此可见，选择最佳路径的策略既路由算法是路由器的关键所在。

用Dijikstra算法求出各终端路由到各路由的最短路径。模拟一个有向网中源点到其他 节点的最短路径和距离，并求出所经过的路径的顶点。设计的只要目的是找到网中两点的 最短路径，这样在计算机实际运用中可以使得数据在网络中的传输更加快捷，有保证，有 高效，同时也有利于提高数据的传输量。

所以，我们想要提高网络的传送速度和效率，最主要的是找到最佳路由选择实现方法，路径选择算法的好坏直接决定着网络性能的高低以及网络资源的利用率，而各种路由选择算法的目标都是使网络延迟最小，吞吐量最大，经过的节点数最少，这些问题其实归结起来就是最短路径问题。

1. 2课程设计目的

在程序设计中关键是如何将路由器的工作原理用数据结构联系起来，转为用邻接矩 阵，数组，字符串，Dijkstra算法等C++语言知识设计出程序。通过程序的编写，调试和 运行可以进一步掌握数据结构计算的实现的基本方法。更进一步理解网络图的表示方法，

熟悉将Dijkstra算法运用于实际应用中。与此同时很大程度上培养自我完成程序设计和解决难点的能力。

1.3程序设计内容

本课程设计是用邻接矩阵完成图的表示和路径以及长度的求解。对邻接矩阵arc[n][m] 中的每一个元素只能有三种情况：1.当n=m时，p[n][m]=0；　　2.当顶点n到m无边时，G[n][m]=MAX；　3.当顶点n到m有边并且权值为G[n][m]时，p[n][m]=G[n][m]。建立图的 表示模块，在建立图之后从单源点开始求最短路径，并显示出来。

程序的功能模块：

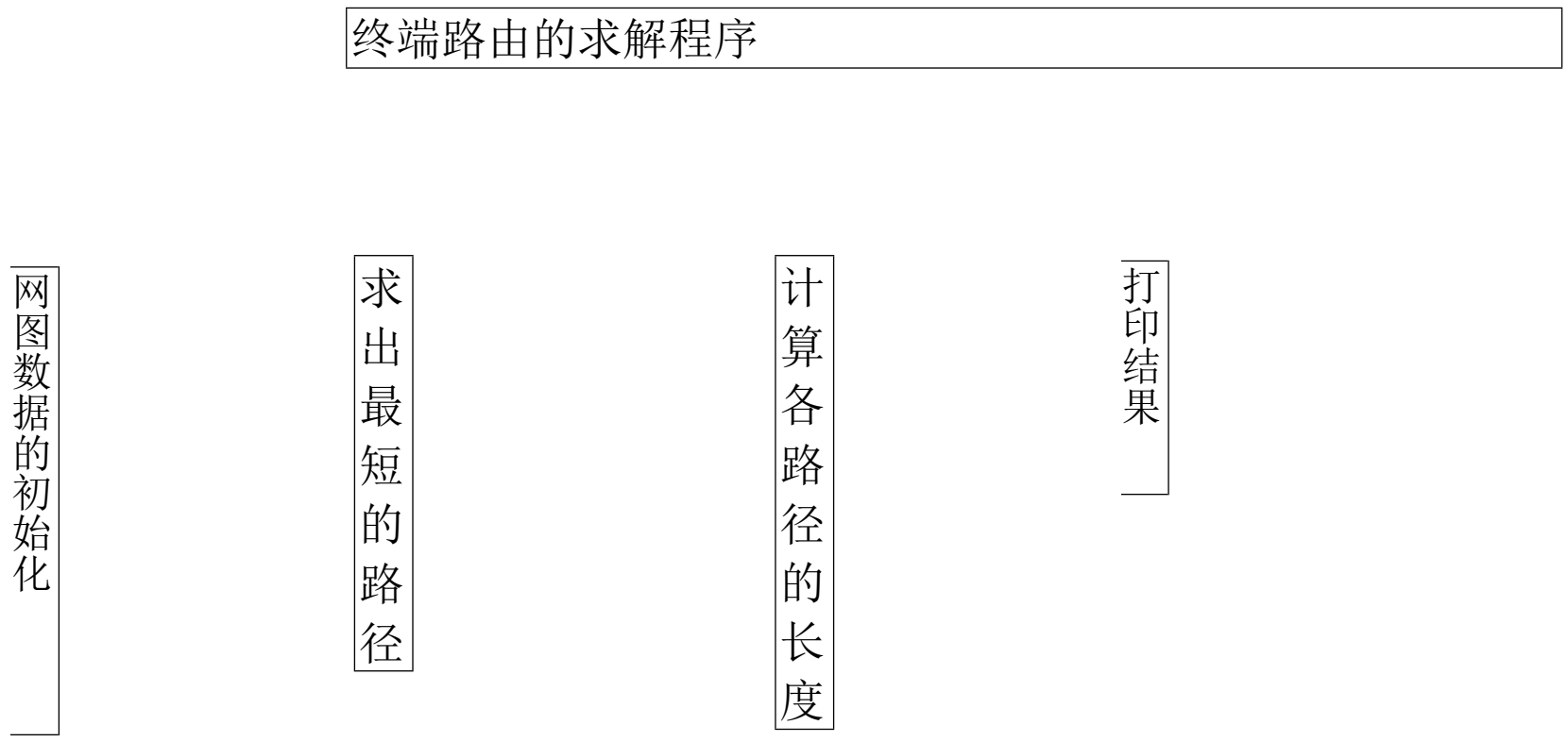

图1. 1程序的功能模块

2设计思路与方案

2. 1设计思路

在网图中，任意两个顶点之间都可能存在边，并且两点之间可达的路径可能不是唯一的，不同的路径所经过的路径值的大小不一样，为了减小到达点之间的距离，我们就需要找到两点之间的最短路径。

在网图中，任意两顶点都可能存在边，但是无法通过顶点的存取位置反映顶点之间的邻接关系，因此图没有顺序存储结构，所以图的存取需要借助邻接矩阵或邻接表来存储。图的邻接矩阵是一个n*n的矩阵，所以其空间代价是O(n*n)。它的存储空间只与它的顶 点数有关，与边数无关。适用与边稠密的图。邻接表的空间代价与图的边数及顶点数有关，每个顶点在顶点表中都要占据一个数据元素的位置(既使该顶点没有邻接表)，且每条边 必须出现在某个顶点的边表中，所以邻接表的空间代价是O(n+e)，适用于稀疏的图中。 因为本课程设计用于计算机网络结构查找路由的最短路径，是稀疏的图的可能性比较小， 所以本课程设计选择邻接矩阵存储图的信息。

通过使用邻接矩阵存储图的信息，再通过Dijkstra算法求出最短路径。

2. 2设计方案

第一步，将一个有向带权的网图的基本信息存储到邻接矩阵中，用InitMatrix()函数初始化矩阵，再利用GreateMatrix()输入有向带权值的图的信息。再用PrintMatrix()函数输出 输入的矩阵。

第二步，通过Dijkstra算法，并以数组为辅，求解最短路径，最短路径所经过的每个 顶点。

第三步，打印出Dijkstra算法求出的最短路径的信息。

2. 3设计流程图：

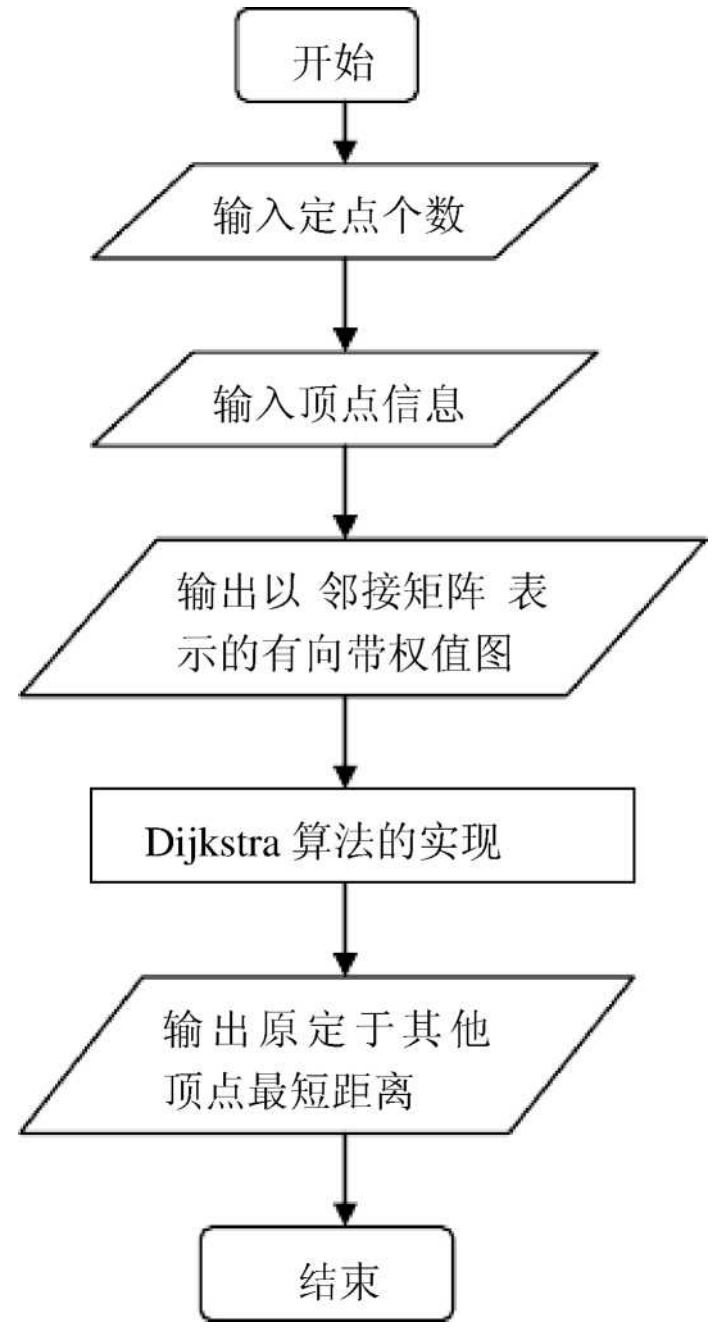
开始
输入定点个数
输入顶点信息
输出以 邻接矩阵 表
示的有向带权值图
Dijkstra 算法的实现
输出原定于其他
顶点最短距离
结束

图2.1总设计流程图

3详细设计

3. 1程序函数的作用说明

函数声明	功能声明
void InitMatrix()	初始化邻接矩阵表示的有向带权值图
void GreateMatrix(0	建立邻接矩阵表示的有向带权值图（既通过输入图的每条边建立图的邻接矩阵）
void PrintMatrix()	输出邻接矩阵表示的有向带权值图（既用邻接矩阵体现出输入的图的信息）
void Path()	辅助函数
viod Dijkstra()	用Dijkstra算法求最短路径
viod PrintPath()	输出从源点到每个顶点最短路劲及长度的函数

3. 2建立有向带权值网图

输入数据，以初始化矩阵。

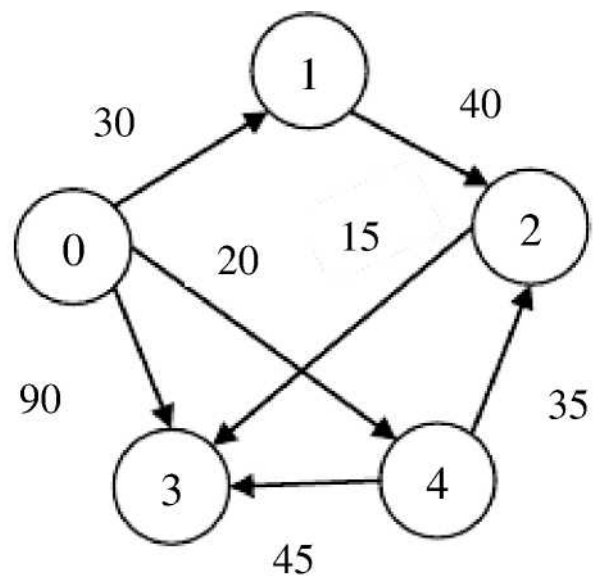

图3. 1有向图

流程图:

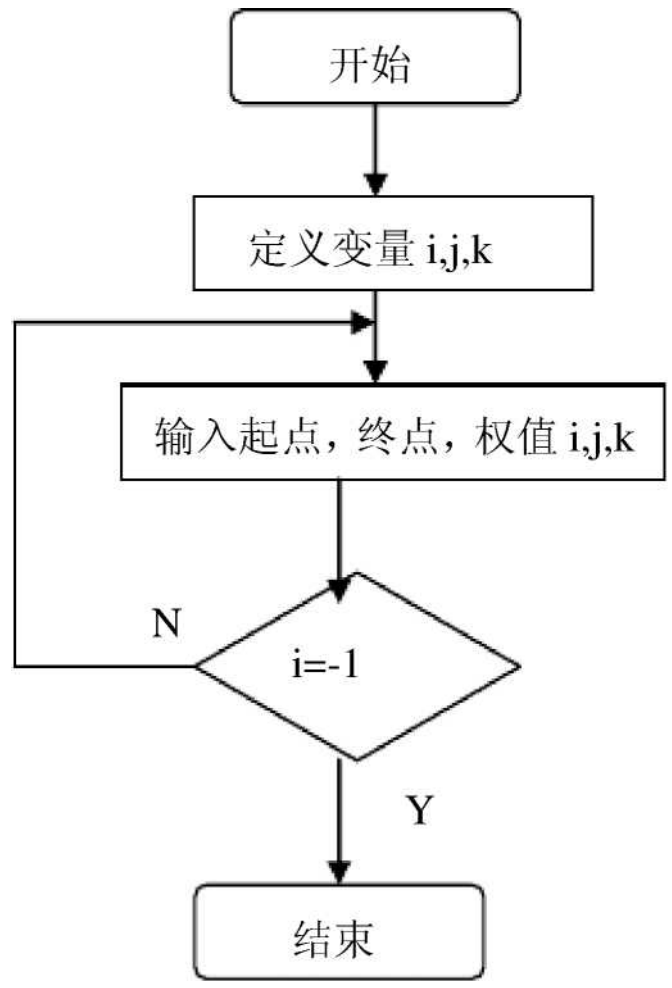

图3. 2输入流程图

输入数据的核心代码：

```
void CreateMatrix(adjmatrix G)
{
int i, j, x;//分别表示起点，终点，权值
cout << "请输入顶点信(顶点是以0为第一个顶点，依次对应下去)\n起点终点权值(输入 -1 结束顶点的输入)"<< endl;

cin >> i >> j >> x;
while(i != -1)//遇到-1，输入完-1和j,x的值循环结束
{
G[i][j] = x;
cin >> i >> j >> x;
}

}
```

初始化后的邻接矩阵

G[5][5] $\left|\begin{array}{ccccc} Inf & 10 & Inf & 30 & 100 \\ Inf & 0 & 50 & Inf & Inf \\ Inf & Inf & 0 & Inf & 10 \\ Inf & Inf & 20 & 0 & 60 \\ Inf & Inf & Inf & Inf & 0 \end{array}\right|$ }（注：Inf表示Vi不能到达Vj

输出流程图:

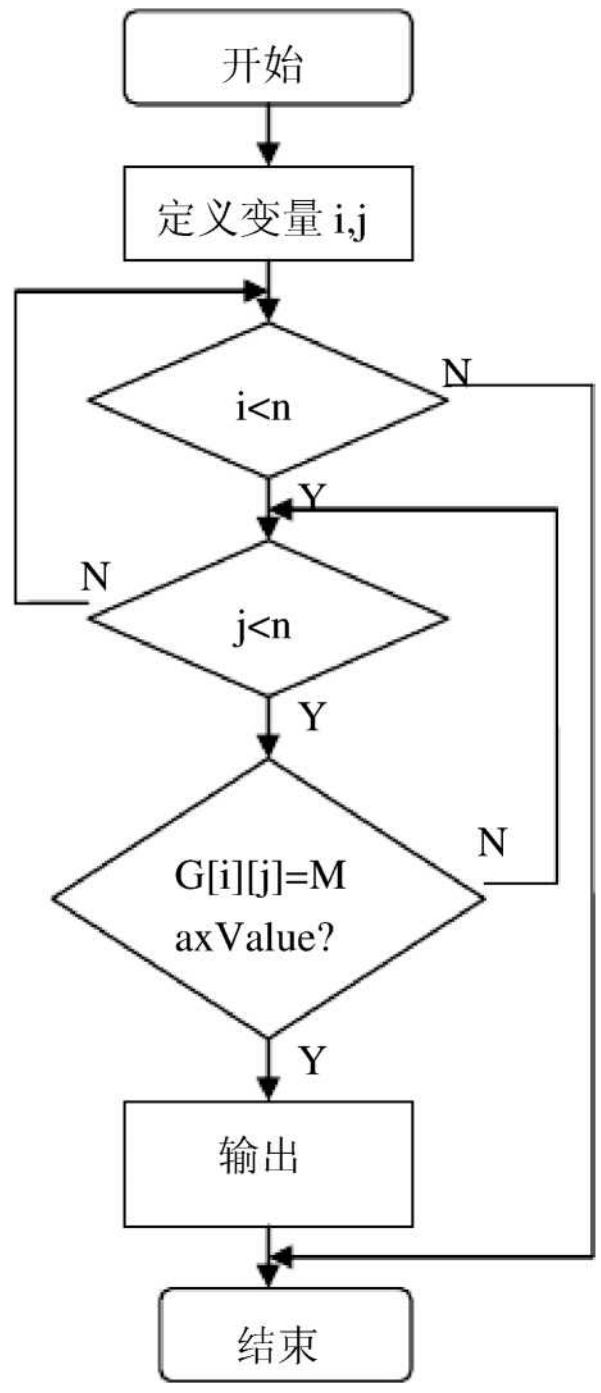

图3. 3输出矩阵流程图

输出邻接矩阵主要代码如下：

```
void PrintMatrix(adjmatrix G, int n)
{
int i, j；
for(i = 0; i < n; i++)
{
for(j = 0; j < n; j++)
```

```
{

if(G[i][j] == MaxValuey/如果权值为最大值输出Inf,否则输出对应的权值大小  cout <<
setiosflags(ios::left) << setw(5) << "Inf";

else cout << setiosflags(ios::left) << setw(5) << G[i][j];

}

cout << endl;

}

}
```

3.3求最短路径

求最短路径流程图：

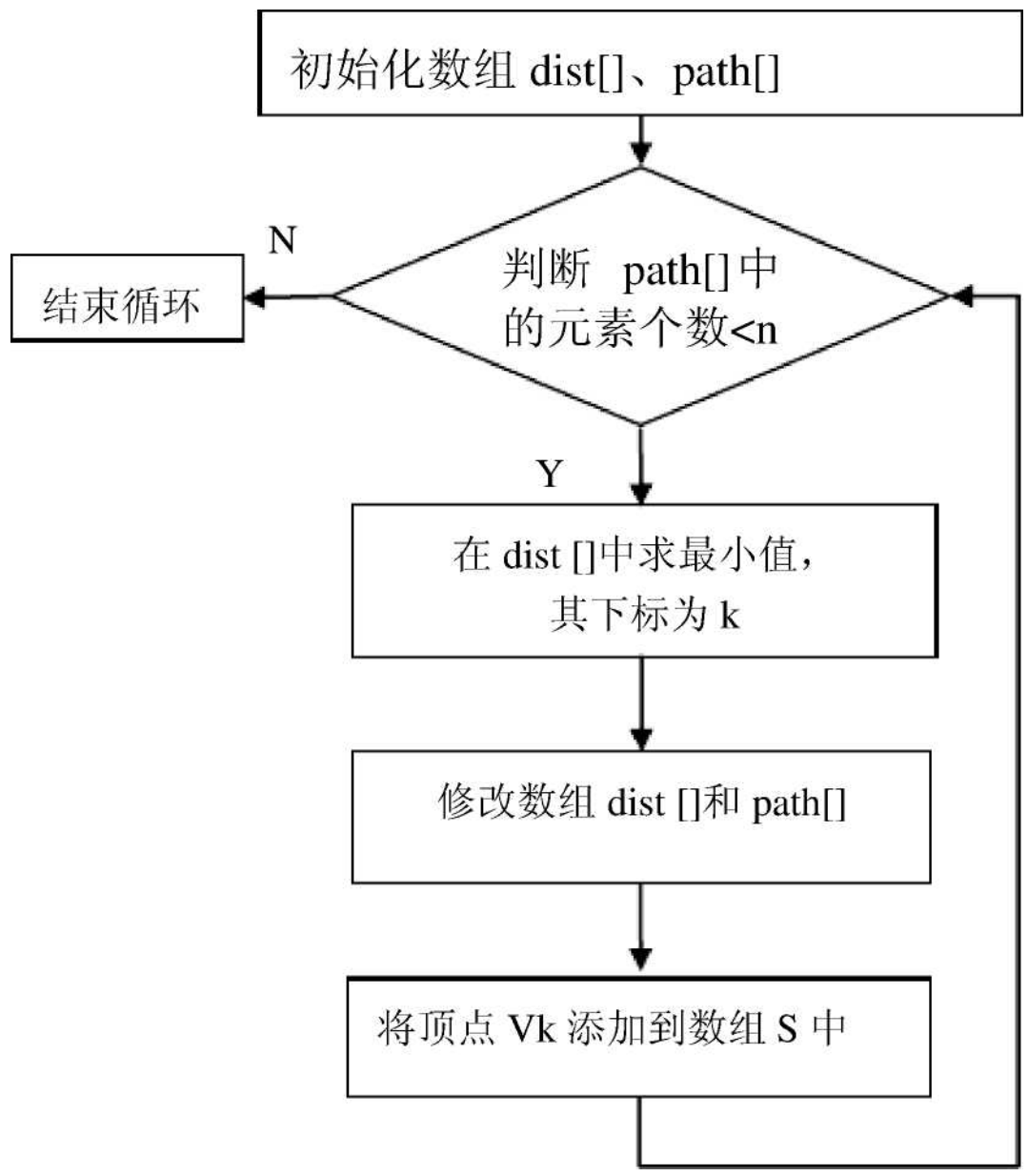

图3.4 Dijkstra算法流程图

用Dijkstra算法求解路径到各终端路由，是本程序的核心部分。核心函数如下:

```
void Dijkstra(adjmatrix GA, int dist[], edgenode *path[], int i, int n)
```