迪杰斯特拉算法

[编辑](javascript:;)

本词条由[“科普中国”百科科学词条编写与应用工作项目](http://www.cast.org.cn/n35081/) 审核 。

迪杰斯特拉算法是由荷兰计算机科学家[狄克斯特拉](http://baike.baidu.com/view/156673.htm)于1959 年提出的，因此又叫[狄克斯特拉算法](http://baike.baidu.com/view/2541415.htm)。是从一个顶点到其余各顶点的[最短路径](http://baike.baidu.com/view/349189.htm)算法，解决的是有向图中最短路径问题。迪杰斯特拉算法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。[1]

**中文名**

迪克斯特拉算法

**外文名**

Dijkstra's Algorithm

**分    类**

计算机算法

**用    途**

单源最短路径问题

目录

1. 1 [定义](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#1)
2. 2 [原理](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#2)
3. 3 [问题描述](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#3)
4. 4 [算法思想](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#4)
5. 5 [算法实现](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#5)
6. ▪ [pascal语言](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#5_1)
7. ▪ [java语言](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#5_2)
8. ▪ [C语言](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#5_3)
9. 6 [堆优化](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#6)
10. ▪ [思考](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#6_1)
11. ▪ [实现](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#6_2)
12. ▪ [代码](http://baike.baidu.com/link?url=gm-CV3-GbVAfgdEcG4MFLG0AIRhlt2GlB5gRNX8y4cJBIJ2lY7A9I_SGhCAGZ3XZjZYkdYMMnkXpB46ZRWYYia#6_3)

定义

[编辑](javascript:;)

Dijkstra算法是典型的[算法](http://baike.baidu.com/view/7420.htm)。[Dijkstra算法](http://baike.baidu.com/view/7839.htm)是很有代表性的[算法](http://baike.baidu.com/view/7420.htm)。Dijkstra一般的表述通常有两种方式，一种用永久和临时标号方式，一种是用[OPEN](http://baike.baidu.com/subview/26337/12488714.htm), CLOSE表的方式，这里均采用永久和临时标号的方式。注意该算法要求图中不存在负权边。[2]

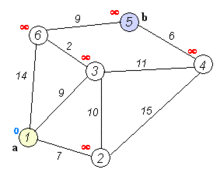
原理

[编辑](javascript:;)

1.首先，引入一个辅助向量D，它的每个分量 D

http://e.hiphotos.baidu.com/baike/s%3D17/sign=480315771e950a7b71354ac30ad18932/7aec54e736d12f2ebd33e0214cc2d562853568b8.jpg

表示当前所找到的

[](http://baike.baidu.com/pic/%E8%BF%AA%E6%9D%B0%E6%96%AF%E7%89%B9%E6%8B%89%E7%AE%97%E6%B3%95/4049057/0/8ad4b31c8701a18b76d304889e2f07082838fe39?fr=lemma&ct=single)Dijkstra算法运行动画过程

从起始点

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

（即源点

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

）到其它每个顶点

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

的长度。

例如，D[3] = 2表示从起始点到顶点3的路径相对最小长度为2。这里强调相对就是说在算法执行过程中D的值是在不断逼近最终结果但在过程中不一定就等于长度。[1]

2.D的初始状态为：若从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

到

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

有弧（即从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

到

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

存在连接边），则D

http://e.hiphotos.baidu.com/baike/s%3D17/sign=480315771e950a7b71354ac30ad18932/7aec54e736d12f2ebd33e0214cc2d562853568b8.jpg

为弧上的权值（即为从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

到

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

的边的权值）；否则置D

http://e.hiphotos.baidu.com/baike/s%3D17/sign=480315771e950a7b71354ac30ad18932/7aec54e736d12f2ebd33e0214cc2d562853568b8.jpg

为∞。

显然，长度为 D

http://d.hiphotos.baidu.com/baike/s%3D18/sign=227e4f8b818ba61edbeecc27403419f1/4bed2e738bd4b31c144addfa84d6277f9e2ff879.jpg

= Min{ D |

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

∈V } 的路径就是从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

出发到顶点

http://d.hiphotos.baidu.com/baike/s%3D16/sign=eef798ace41190ef05fb96d9cf1b2064/43a7d933c895d143c689ac6270f082025baf07f7.jpg

的长度最短的一条路径，此路径为(

http://a.hiphotos.baidu.com/baike/s%3D28/sign=8922c04fb23533faf1b69426a9d31b22/8718367adab44aedbe89f7a5b01c8701a08bfba8.jpg

)。

3.那么，下一条长度次短的是哪一条呢？也就是找到从源点

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

到下一个顶点的最短路径长度所对应的顶点，且这条最短路径长度仅次于从源点

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

到顶点

http://d.hiphotos.baidu.com/baike/s%3D16/sign=eef798ace41190ef05fb96d9cf1b2064/43a7d933c895d143c689ac6270f082025baf07f7.jpg

的最短路径长度。

假设该次短路径的终点是

http://a.hiphotos.baidu.com/baike/s%3D14/sign=12efb060718b4710ca2ff9c8c2ce63f2/63d9f2d3572c11df01a2f762602762d0f703c204.jpg

，则可想而知，这条路径要么是(

http://f.hiphotos.baidu.com/baike/s%3D29/sign=174cae78b0de9c82a265fe866c81033f/4b90f603738da977bc06cc6ab351f8198618e342.jpg

)，或者是(

http://f.hiphotos.baidu.com/baike/s%3D49/sign=f00c828bac51f3dec7b2b86d95ee39c1/ac4bd11373f0820237b5db7e48fbfbedab641b4f.jpg

)。它的长度或者是从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

到

http://a.hiphotos.baidu.com/baike/s%3D14/sign=12efb060718b4710ca2ff9c8c2ce63f2/63d9f2d3572c11df01a2f762602762d0f703c204.jpg

的弧上的权值，或者是D

http://d.hiphotos.baidu.com/baike/s%3D18/sign=227e4f8b818ba61edbeecc27403419f1/4bed2e738bd4b31c144addfa84d6277f9e2ff879.jpg

加上从

http://d.hiphotos.baidu.com/baike/s%3D16/sign=5833cf20fb1986184547eb824bedbecd/902397dda144ad34b2aacdc6d3a20cf431ad855e.jpg

到

http://a.hiphotos.baidu.com/baike/s%3D14/sign=12efb060718b4710ca2ff9c8c2ce63f2/63d9f2d3572c11df01a2f762602762d0f703c204.jpg

的弧上的权值。

4.一般情况下，假设S为已求得的从源点

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

出发的最短路径长度的顶点的集合，则可证明：下一条次最短路径（设其终点为

http://f.hiphotos.baidu.com/baike/s%3D9/sign=6605e2a1f403918fd3d131fa51c3e7/4ec2d5628535e5dde233109e75c6a7efce1b6257.jpg

）要么是弧(

http://d.hiphotos.baidu.com/baike/s%3D25/sign=583f7537f11fbe09185ec4116a60e51b/dc54564e9258d109902ae3f3d258ccbf6d814da1.jpg

)，或者是从源点

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

出发的中间只经过S中的顶点而最后到达顶点

http://f.hiphotos.baidu.com/baike/s%3D9/sign=6605e2a1f403918fd3d131fa51c3e7/4ec2d5628535e5dde233109e75c6a7efce1b6257.jpg

的路径。

因此，下一条长度次短的的最短路径长度必是D

http://d.hiphotos.baidu.com/baike/s%3D18/sign=227e4f8b818ba61edbeecc27403419f1/4bed2e738bd4b31c144addfa84d6277f9e2ff879.jpg

= Min{ D

http://e.hiphotos.baidu.com/baike/s%3D17/sign=480315771e950a7b71354ac30ad18932/7aec54e736d12f2ebd33e0214cc2d562853568b8.jpg

|

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

∈V-S }，其中D

http://e.hiphotos.baidu.com/baike/s%3D17/sign=480315771e950a7b71354ac30ad18932/7aec54e736d12f2ebd33e0214cc2d562853568b8.jpg

要么是弧(

http://c.hiphotos.baidu.com/baike/s%3D27/sign=992a7b9ca864034f0bcdc501aec35574/d52a2834349b033b8964499b16ce36d3d439bd87.jpg

)上的权值，或者是D

http://d.hiphotos.baidu.com/baike/s%3D17/sign=037e5132ae4bd11300cdb3355baf5d12/eaf81a4c510fd9f91cc1e028262dd42a2934a499.jpg

(

http://a.hiphotos.baidu.com/baike/s%3D14/sign=12efb060718b4710ca2ff9c8c2ce63f2/63d9f2d3572c11df01a2f762602762d0f703c204.jpg

∈S)和弧(

http://a.hiphotos.baidu.com/baike/s%3D14/sign=12efb060718b4710ca2ff9c8c2ce63f2/63d9f2d3572c11df01a2f762602762d0f703c204.jpg

,

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

)上的权值之和。

算法描述如下：

1）令arcs表示弧上的权值。若弧不存在，则置arcs为∞（在本程序中为MAXCOST）。S为已找到的从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

出发的的终点的集合，初始状态为空集。那么，从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

出发到图上其余各顶点

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

可能达到的长度的初值为D=arcs[Locate Vex(G,

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

)]，

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

∈V；

2）选择

http://d.hiphotos.baidu.com/baike/s%3D16/sign=eef798ace41190ef05fb96d9cf1b2064/43a7d933c895d143c689ac6270f082025baf07f7.jpg

，使得D

http://d.hiphotos.baidu.com/baike/s%3D18/sign=227e4f8b818ba61edbeecc27403419f1/4bed2e738bd4b31c144addfa84d6277f9e2ff879.jpg

=Min{ D |

http://h.hiphotos.baidu.com/baike/s%3D12/sign=8da8de7e48fbfbedd859327d78f0572b/242dd42a2834349bd43c21c4caea15ce36d3bebd.jpg

∈V-S } ；

3）修改从

http://h.hiphotos.baidu.com/baike/s%3D9/sign=efa941b69c3df8dca23d83a0cd3f23/29381f30e924b899366736796d061d950a7bf61c.jpg

出发的到集合V-S中任一顶点

http://a.hiphotos.baidu.com/baike/s%3D14/sign=12efb060718b4710ca2ff9c8c2ce63f2/63d9f2d3572c11df01a2f762602762d0f703c204.jpg

的最短路径长度。[1]

问题描述

[编辑](javascript:;)

在[无向图](http://baike.baidu.com/view/93110.htm) G=(V,E) 中，假设每条边 E[i] 的长度为 w[i]，找到由顶点 V0 到其余各点的最短值。[2]

算法思想

[编辑](javascript:;)

按路径长度[递增](http://baike.baidu.com/view/5195137.htm)次序产生算法：

把顶点集合V分成两组：

（1）S：已求出的顶点的集合（初始时只含有源点V0）

（2）V-S=T：尚未确定的顶点集合

将T中顶点按递增的次序加入到S中，保证：

（1）从源点V0到S中其他各顶点的长度都不大于从V0到T中任何顶点的最短路径长度

（2）每个顶点对应一个距离值

S中顶点：从V0到此顶点的长度

T中顶点：从V0到此顶点的只包括S中顶点作中间顶点的最短路径长度

依据：可以证明V0到T中顶点Vk的，或是从V0到Vk的直接路径的权值；或是从V0经S中顶点到Vk的路径权值之和

（[反证法](http://baike.baidu.com/view/276975.htm)可证）

求最短路径步骤

算法步骤如下：

G={V,E}

1. 初始时令 S={V0},T=V-S={其余顶点}，T中顶点对应的距离值

若存在<V0,Vi>，d(V0,Vi)为<V0,Vi>弧上的权值

若不存在<V0,Vi>，d(V0,Vi)为∞

2. 从T中选取一个与S中顶点有关联边且权值最小的顶点W，加入到S中

3. 对其余T中顶点的距离值进行修改：若加进W作中间顶点，从V0到Vi的距离值缩短，则修改此距离值

重复上述步骤2、3，直到S中包含所有顶点，即W=Vi为止

### C语言

下面是该算法的C语言实现[1]

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include<stdio.h>  #include<stdlib.h>  #define max 11000000000  inta[1000][1000];  intd[1000];//d表示某特定边距离  intp[1000];//p表示永久边距离  inti,j,k;  intm;//m代表边数  intn;//n代表点数  intmain()  {  scanf("%d%d",&n,&m);  intmin1;  intx,y,z;  for(i=1;i<=m;i++)  {  scanf("%d%d%d",&x,&y,&z);  a[x][y]=z;  a[y][x]=z;  }  for(i=1;i<=n;i++)  d[i]=max1;  d[1]=0;  for(i=1;i<=n;i++)  {  min1=max1;  for(j=1;j<=n;j++)  if(!p[j]&&d[j]<min1)  {  min1=d[j];  k=j;  }  p[k]=j;  for(j=1;j<=n;j++)  if(a[k][j]!=0&&!p[j]&&d[j]>d[k]+a[k][j])  d[j]=d[k]+a[k][j];  }  for(i=1;i<n;i++)  printf("%d->",p[i]);  printf("%d\n",p[n]);  return0;  } |

**大学经典教材<<数据结构>>(C语言版 严蔚敏 吴为民 编著) 中该算法的实现**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | /\*  测试数据 教科书 P189 G6 的邻接矩阵 其中 数字 1000000 代表无穷大  6  1000000 1000000 10 100000 30 100  1000000 1000000 5 1000000 1000000 1000000  1000000 1000000 1000000 50 1000000 1000000  1000000 1000000 1000000 1000000 1000000 10  1000000 1000000 1000000 20 1000000 60  1000000 1000000 1000000 1000000 1000000 1000000  结果：  D[0]   D[1]   D[2]   D[3]   D[4]   D[5]   0   1000000   10     50     30     60  \*/  #include <iostream>  #include <cstdio>  #define MAX 1000000  using namespace std;  int arcs[10][10];//邻接矩阵  int D[10];//保存最短路径长度  int p[10][10];//路径  int final[10];//若final[i] = 1则说明 顶点vi已在集合S中  int n = 0;//顶点个数  int v0 = 0;//源点  int v,w;  void ShortestPath\_DIJ()  {       for (v = 0; v < n; v++) //循环 初始化       {            final[v] = 0; D[v] = arcs[v0][v];            for (w = 0; w < n; w++) p[v][w] = 0;//设空路径            if (D[v] < MAX) {p[v][v0] = 1; p[v][v] = 1;}       }       D[v0] = 0; final[v0]=0; //初始化 v0顶点属于集合S       //开始主循环 每次求得v0到某个顶点v的最短路径 并加v到集合S中       for (int i = 1; i < n; i++)       {            int min = MAX;            for (w = 0; w < n; w++)            {                 //我认为的核心过程--选点                 if (!final[w]) //如果w顶点在V-S中                 {                      //这个过程最终选出的点 应该是选出当前V-S中与S有关联边                      //且权值最小的顶点 书上描述为 当前离V0最近的点                      if (D[w] < min) {v = w; min = D[w];}                 }            }            final[v] = 1; //选出该点后加入到合集S中            for (w = 0; w < n; w++)//更新当前最短路径和距离            {                 /\*在此循环中 v为当前刚选入集合S中的点                 则以点V为中间点 考察 d0v+dvw 是否小于 D[w] 如果小于 则更新                 比如加进点 3 则若要考察 D[5] 是否要更新 就 判断 d(v0-v3) + d(v3-v5) 的和是否小于D[5]                 \*/                 if (!final[w] && (min+arcs[v][w]<D[w]))                 {                      D[w] = min + arcs[v][w];                     // p[w] = p[v];                      p[w][w] = 1; //p[w] = p[v] +　[w]                 }            }       }  }      int main()  {      cin >> n;      for (int i = 0; i < n; i++)      {           for (int j = 0; j < n; j++)           {                cin >> arcs[i][j];           }      }      ShortestPath\_DIJ();      for (int i = 0; i < n; i++) printf("D[%d] = %d\n",i,D[i]);      return 0;  } |

## 堆优化

[编辑](javascript:;)

### 思考

该[算法复杂度](http://baike.baidu.com/view/7527.htm)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](http://baike.baidu.com/view/249120.htm)这种[数据结构](http://baike.baidu.com/subview/9900/5066576.htm)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（elogn）；e为该点的边数，所以复杂度降为*O*((*m*+*n*)log*n*)。

### 实现

1. 将与源点相连的点加入[堆](http://baike.baidu.com/view/249120.htm)，并调整堆。

2. 选出堆顶元素u（即代价最小的元素），从堆中删除，并对堆进行调整。

3. 处理与u相邻的，未被访问过的，满足三角不等式的顶点

1):若该点在堆里，更新距离，并调整该元素在堆中的位置。

2):若该点不在堆里，加入堆，更新堆。

4. 若取到的u为终点，结束算法；否则重复步骤2、3。

### 代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | procedureDijkstra;  var  u,v,e,i:longint;  begin  fillchar(dis,sizeof(dis),$7e);//距离  fillchar(Inh,sizeof(Inh),false);//是否在堆中  fillchar(visit,sizeof(visit),false);//是否访问过  size:=0;  e:=last[s];  whilee<>0do//步骤1  begin  u:=other[e];  ifnot(Inh[u])then//不在堆里  begin  inc(size);  heap[size]:=u;  dis[u]:=cost[e];  Loc[u]:=size;//Loc数组记录元素在堆中的位置  Inh[u]:=true;  Shift\_up(Loc[u]);//上浮  end  else  ifcost[e]<dis[u]then//在堆里  begin  dis[u]:=cost[e];  Shift\_up(Loc[u]);  Shift\_down(Loc[u]);  end;  e:=pre[e];  end;  visit[s]:=true;  whiletruedo  begin  u:=heap[1];//步骤2  ifu=tthenbreak;//步骤4  visit[u]:=true;  heap[1]:=heap[size];  dec(size);  Shift\_down(1);  e:=last[u];  whilee<>0do//步骤3  begin  v:=other[e];  ifNot(visit[v])and(dis[u]+cost[e]<dis[v])then//与u相邻的，未被访问过的，满足三角不等式的顶点  ifInh[v]then//在堆中  begin  dis[v]:=dis[u]+cost[e];  Shift\_up(Loc[v]);  Shift\_Down(Loc[v]);  end  else//不再堆中  begin  inc(size);  heap[size]:=v;  dis[v]:=dis[u]+cost[e];  Loc[v]:=size;  Inh[v]:=true;  Shift\_up(Loc[v]);  end;  e:=pre[e];  end;  end;  writeln(dis[t]);  end; |

**Dijkstra算法**

1.定义概览

Dijkstra(迪杰斯特拉)算法是典型的单源最短路径算法，用于计算一个节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。Dijkstra算法是很有代表性的最短路径算法，在很多专业课程中都作为基本内容有详细的介绍，如数据结构，图论，运筹学等等。注意该算法要求图中不存在负权边。

问题描述：在无向图 G=(V,E) 中，假设每条边 E[i] 的长度为 w[i]，找到由顶点 V0 到其余各点的最短路径。（单源最短路径）

2.算法描述

1)算法思想：设G=(V,E)是一个带权有向图，把图中顶点集合V分成两组，第一组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点，以后每求得一条最短路径 , 就将加入到集合S中，直到全部顶点都加入到S中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用U表示），按最短路径长度的递增次序依次把第二组的顶点加入S中。在加入的过程中，总保持从源点v到S中各顶点的最短路径长度不大于从源点v到U中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S中的顶点的距离就是从v到此顶点的最短路径长度，U中的顶点的距离，是从v到此顶点只包括S中的顶点为中间顶点的当前最短路径长度。

2)算法步骤：

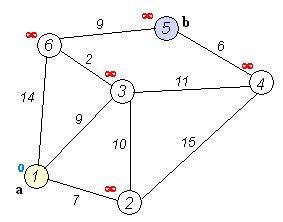
a.初始时，S只包含源点，即S＝{v}，v的距离为0。U包含除v外的其他顶点，即:U={其余顶点}，若v与U中顶点u有边，则<u,v>正常有权值，若u不是v的出边邻接点，则<u,v>权值为∞。

b.从U中选取一个距离v最小的顶点k，把k，加入S中（该选定的距离就是v到k的最短路径长度）。

c.以k为新考虑的中间点，修改U中各顶点的距离；若从源点v到顶点u的距离（经过顶点k）比原来距离（不经过顶点k）短，则修改顶点u的距离值，修改后的距离值的顶点k的距离加上边上的权。

d.重复步骤b和c直到所有顶点都包含在S中。

执行动画过程如下图



3.算法代码实现：

[复制代码](javascript:void(0);)

const int MAXINT = 32767;

const int MAXNUM = 10;

int dist[MAXNUM];

int prev[MAXNUM];

int A[MAXUNM][MAXNUM];

void Dijkstra(int v0)

{

　　bool S[MAXNUM]; // 判断是否已存入该点到S集合中

int n=MAXNUM;

　　for(int i=1; i<=n; ++i)

　　 {

　　dist[i] = A[v0][i];

　　S[i] = false; // 初始都未用过该点

　　if(dist[i] == MAXINT)

　　prev[i] = -1;

　　 else

　　prev[i] = v0;

　　}

　 dist[v0] = 0;

　 S[v0] = true;

　　 for(int i=2; i<=n; i++)

　　 {

　　int mindist = MAXINT;

　　int u = v0; 　　 // 找出当前未使用的点j的dist[j]最小值

　　 for(int j=1; j<=n; ++j)

　　 if((!S[j]) && dist[j]<mindist)

　　 {

　　 u = j; // u保存当前邻接点中距离最小的点的号码

　 　 mindist = dist[j];

　　 }

　　S[u] = true;

　　for(int j=1; j<=n; j++)

　　 if((!S[j]) && A[u][j]<MAXINT)

　　 {

　 　if(dist[u] + A[u][j] < dist[j]) //在通过新加入的u点路径找到离v0点更短的路径

　 　{

　　dist[j] = dist[u] + A[u][j]; //更新dist

　　prev[j] = u; //记录前驱顶点

　　 }

　 　}

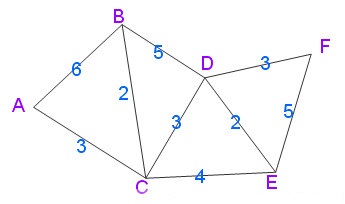
　　}

}

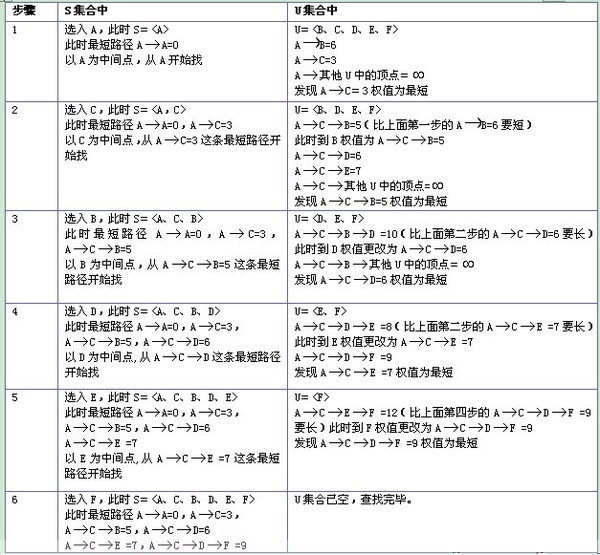
[复制代码](javascript:void(0);)

4.算法实例

先给出一个无向图



用Dijkstra算法找出以A为起点的单源最短路径步骤如下



**Floyd算法**

1.定义概览

**Floyd-Warshall算法**（Floyd-Warshall algorithm）是解决任意两点间的最短路径的一种算法，可以正确处理有向图或负权的最短路径问题，同时也被用于计算有向图的传递闭包。Floyd-Warshall算法的时间复杂度为O(N3)，空间复杂度为O(N2)。

2.算法描述

1)算法思想原理：

     Floyd算法是一个经典的动态规划算法。用通俗的语言来描述的话，首先我们的目标是寻找从点i到点j的最短路径。从动态规划的角度看问题，我们需要为这个目标重新做一个诠释（这个诠释正是动态规划最富创造力的精华所在）

      从任意节点i到任意节点j的最短路径不外乎2种可能，1是直接从i到j，2是从i经过若干个节点k到j。所以，我们假设Dis(i,j)为节点u到节点v的最短路径的距离，对于每一个节点k，我们检查Dis(i,k) + Dis(k,j) < Dis(i,j)是否成立，如果成立，证明从i到k再到j的路径比i直接到j的路径短，我们便设置Dis(i,j) = Dis(i,k) + Dis(k,j)，这样一来，当我们遍历完所有节点k，Dis(i,j)中记录的便是i到j的最短路径的距离。

2).算法描述：

a.从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

b.对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比己知的路径更短。如果是更新它。

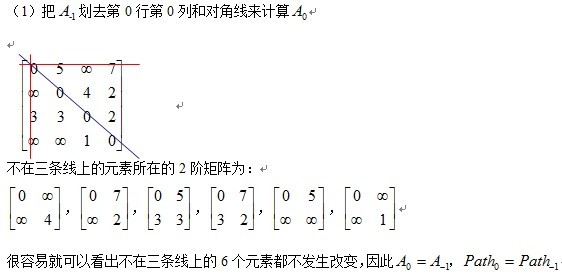
3).Floyd算法过程矩阵的计算----十字交叉法

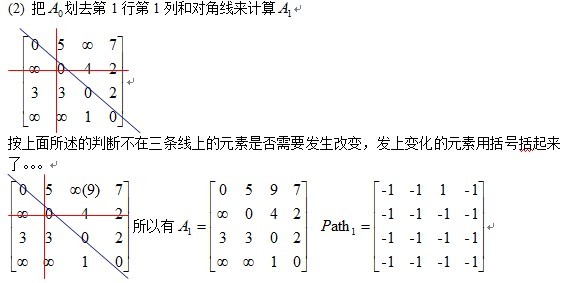
方法：两条线，从左上角开始计算一直到右下角 如下所示

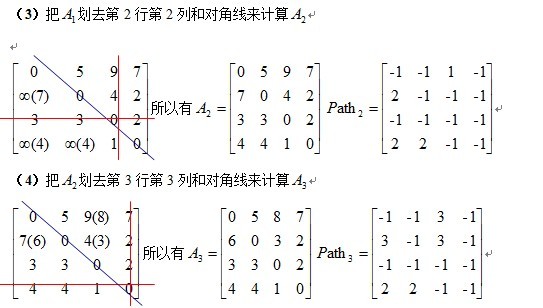
给出矩阵，其中矩阵A是邻接矩阵，而矩阵Path记录u,v两点之间最短路径所必须经过的点



相应计算方法如下：







最后A3即为所求结果

3.算法代码实现

[复制代码](javascript:void(0);)

typedef struct

{

char vertex[VertexNum]; //顶点表

int edges[VertexNum][VertexNum]; //邻接矩阵,可看做边表

int n,e; //图中当前的顶点数和边数

}MGraph;

void Floyd(MGraph g)

{

　　int A[MAXV][MAXV];

　　int path[MAXV][MAXV];

　　int i,j,k,n=g.n;

　　for(i=0;i<n;i++)

　　for(j=0;j<n;j++)

　　{

A[i][j]=g.edges[i][j];

　　 path[i][j]=-1;

　 }

　　for(k=0;k<n;k++)

　　{

　　for(i=0;i<n;i++)

　　for(j=0;j<n;j++)

　　if(A[i][j]>(A[i][k]+A[k][j]))

　　{  
 　　A[i][j]=A[i][k]+A[k][j];

　　path[i][j]=k;

　 }

　}   
}

[复制代码](javascript:void(0);)

算法时间复杂度:O(n3)