

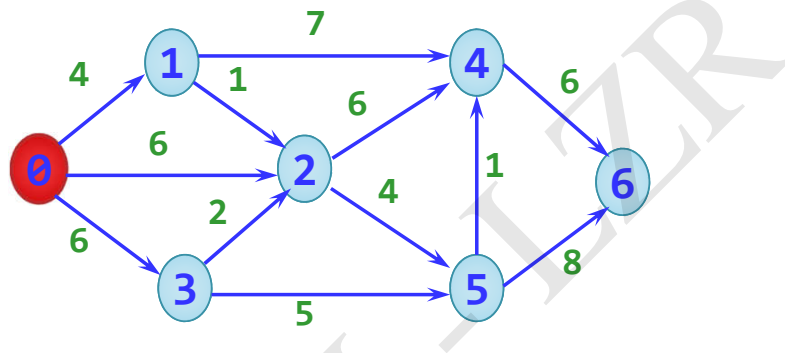
实现迪杰斯特拉算法：

- 设置一个距离数组 $\text{dist}[0..n-1]$ ， $\text{dist}[i]$ 用来保存从源点 v 到顶点 i 的目前最短路径长度。
- $\text{path}[j]$ 保存源点到顶点 j 的最短路径，实际上为最短路径上的前一个顶点 u ，即： $\text{path}[j]=u$ 。
- 当求出最短路径后，由 $\text{path}[j]$ 向前推出源点到顶点 j 的最短路径。

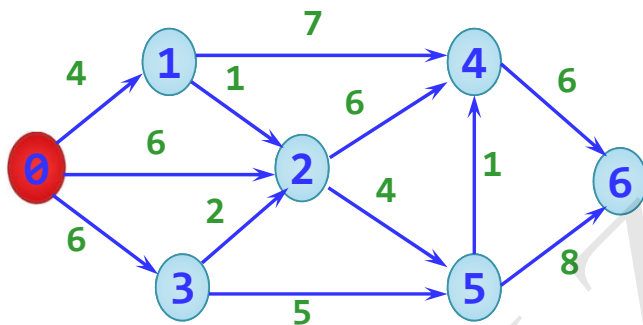


$\text{path}[j]=u$

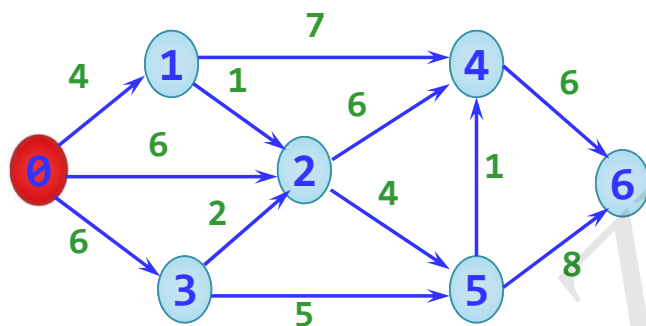
迪杰斯特拉算法演示



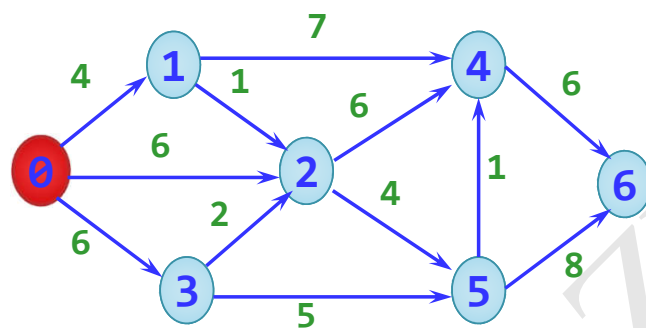
U	dist							path							最小点u
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	
1,2,3,4,5,6	0	4	6	6	∞	∞	∞	0	0	0	0	-1	-1	-1	1



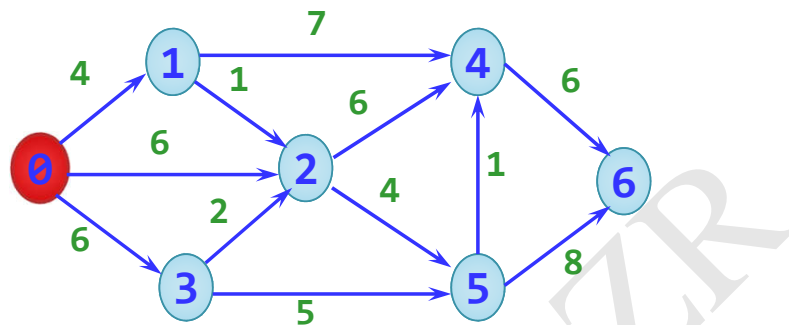
U	dist							path							最小点u
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	
1,2,3, 4,5,6	0	4	6	6	∞	∞	∞	0	0	0	0	-1	-1	-1	1
2,3,4,5,6	0	4	5	6	11	∞	∞	0	0	1	0	1	-1	-1	2



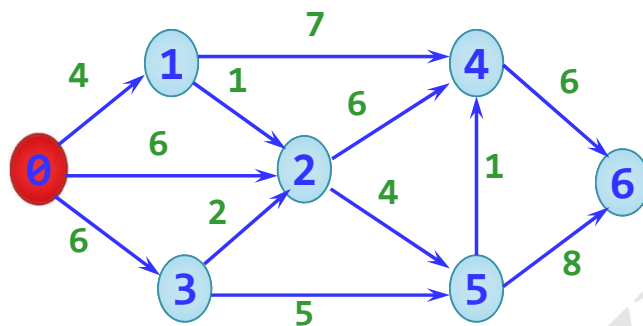
U	dist							path							最小点u
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	
1,2,3, 4,5,6	0	4	6	6	∞	∞	∞	0	0	0	0	-1	-1	-1	1
2,3,4,5,6	0	4	5	6	11	∞	∞	0	0	1	0	1	-1	-1	2
3,4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	3



U	dist							path							最小点u
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	
1,2,3,4,5,6	0	4	6	6	∞	∞	∞	0	0	0	0	-1	-1	-1	1
2,3,4,5,6	0	4	5	6	11	∞	∞	0	0	1	0	1	-1	-1	2
3,4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	3
4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	5



U	dist							path							最小点u
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	
1,2,3,4,5,6	0	4	6	6	∞	∞	∞	0	0	0	0	-1	-1	-1	1
2,3,4,5,6	0	4	5	6	11	∞	∞	0	0	1	0	1	-1	-1	2
3,4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	3
4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	5
4,6	0	4	5	6	10	9	17	0	0	1	0	5	2	5	4



U	dist							path							最小点u
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	
1,2,3,4,5,6	0	4	6	6	∞	∞	∞	0	0	0	0	-1	-1	-1	1
2,3,4,5,6	0	4	5	6	11	∞	∞	0	0	1	0	1	-1	-1	2
3,4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	3
4,5,6	0	4	5	6	11	9	∞	0	0	1	0	1	2	-1	5
4,6	0	4	5	6	10	9	17	0	0	1	0	5	2	5	4
6	0	4	5	6	10	9	16	0	0	1	0	5	2	4	6
	0	4	5	6	10	9	16	0	0	1	0	5	2	4	

- 最后求出顶点0到1~6各顶点的最短距离分别为4、5、6、10、9和16。
- path值为{0,0,1,0,5,2,4}。
- 以求顶点0到顶点4的最短路径为例说明通过path求最短路径的过程：path[4]=5，path[5]=2，path[2]=1，path[1]=0（源点），则顶点0到顶点4的最短路径逆为4、5、2、1、0，则正向最短路径为0→1→2→5→4。

对应的迪杰斯特拉算法如下 (v为源点编号)

```
void Dijkstra(MGraph g, int v)
//求从v到其他顶点的最短路径
{
    int dist[MAXVEX];           //建立dist数组
    int path[MAXVEX];           //建立path数组
    int S[MAXVEX];              //建立S数组
    int mindis,i,j,u=0;

    for (i=0;i<g.n;i++)
    {
        dist[i]=g.edges[v][i]; //距离初始化
        S[i]=0;                //S[]置空
        if (g.edges[v][i]<INF) //路径初始化
            path[i]=v;         //v→i有边时, 置i前一顶点为v
        else                   //v→i没边时, 置i前一顶点为-1
            path[i]=-1;
    }
}
```

```

S[v]=1; //源点编号v放入S中
for (i=0;i<g.n-1;i++) //循环向S中添加n-1个顶点
{ mindis=INF; //mindis置最小长度初值
  for (j=0;j<g.n;j++) //选取不在S中且有最小距离顶点u
    if (S[j]==0 && dist[j]<mindis)
    { u=j;
      mindis=dist[j];
    }
  S[u]=1; //顶点u加入S中
  for (j=0;j<g.n;j++) //修改不在s中的顶点的距离
    if (S[j]==0)
      if (g.edges[u][j]<INF
          && dist[u]+g.edges[u][j]<dist[j])
      { dist[j]=dist[u]+g.edges[u][j];
        path[j]=u;
      }
}
}

```

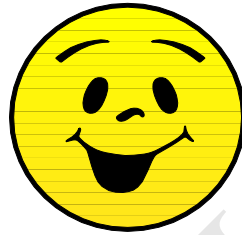
迪杰斯特拉算法Dijkstra(g,v)的时间复杂度为 $O(n^2)$ 。

■ 针对示例的图G，运行该代码，结果如下：

```
"G:\Dev-C++_Code\数据结构 严蔚敏 code\dijkstra.exe"
图G的邻接矩阵:
 0  4  6  6  ∞  ∞  ∞
 ∞  0  1  ∞  7  ∞  ∞
 ∞  ∞  0  ∞  6  4  ∞
 ∞  ∞  2  0  ∞  5  ∞
 ∞  ∞  ∞  ∞  0  ∞  6
 ∞  ∞  ∞  ∞  1  0  8
 ∞  ∞  ∞  ∞  ∞  ∞  0

从0顶点出发的最短路径如下:
从顶点0到顶点1的路径长度为:4  路径为:0, 1
从顶点0到顶点2的路径长度为:5  路径为:0, 1, 2
从顶点0到顶点3的路径长度为:6  路径为:0, 3
从顶点0到顶点4的路径长度为:10  路径为:0, 1, 2, 5, 4
从顶点0到顶点5的路径长度为:9  路径为:0, 1, 2, 5
从顶点0到顶点6的路径长度为:16  路径为:0, 1, 2, 5, 4, 6

Process returned 1 (0x1)   execution time : 0.359 s
Press any key to continue.
```



— END —