7.6 最短路径

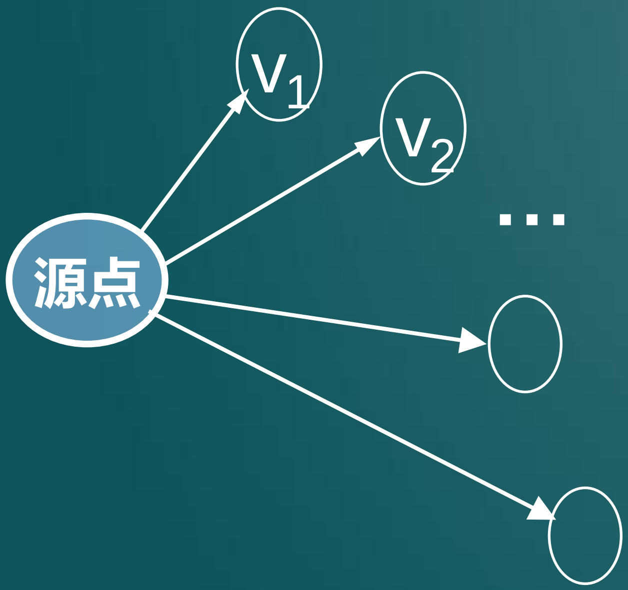
7.6.1 从某个源点到其余各顶点的最短路径

7.6.2 每一对顶点之间的最短路径

7.6.1 从某个源点到其余各顶点的最短路径

求从源点到其余各点的最短路径的算法的基本思想:

迪杰斯特拉（Dijkstra）提出依**路径长度递增的次序**求得各条最短路**径的算法。**



其中，从源点到顶点vj的最短路径是所有最短路径中长度最短者。

路径长度最短的最短路径的特点：

在这条路径上，必定只含一条弧，并且这条弧的权值最小。假设该顶点为V_j。

下一条路径长度次短的最短路径的特点：

它只可能有两种情况：或者是直接从源点到该点（只含一条弧）；或者是从源点经过顶点v_j，再到达该顶点（由两条弧组成）。假设该顶点为V_2。

一般情况下，假设S为已求得最短路径的终点的集合，则可证明：下一条最短路径（设终点为x）或者是弧（v_0, x）；或者是中间只经过S中的顶点而最后到达顶点 x 的路径。

因此，在一般情况下，下一条长度次短的最短路径的长度必是：

$$D[j]=Min\{ D[i] \mid v_i \in V-S \}$$

其中， D[i]或者是弧（v_0，v_i）上的权值，或者是D[k]($v_k \in S$)和弧（v_k，v_i）上的权值之和。

根据以上分析，可以得到如下描述的算法：

设置辅助数组Dist，其中每个分量Dist[k] 表示 当前所求得的从源点到其余各顶点V_k的最短路径。

Dist[k] 初值为：

$$\text{Dist}[k]=\begin{cases} G.arcs[v_0][k] & v_0\text{和}v_k\text{之间存在弧} \\ \infty & v_0\text{和}v_k\text{之间不存在弧} \end{cases}$$

1）在所有从源点出发的弧中选取一条权值最小的弧，即为第一条最短路径。

$$\mathrm{Dist}[k]=\begin{cases} G.arcs[v_0][k] & v_0\text{和}v_k\text{之间存在弧} \\ \infty & v_0\text{和}v_k\text{之间不存在弧} \end{cases}$$

其中的最小值即为第一条最短路径的长度。

2）选择V_j，使得

$D[j]=Min\{\ D[i]\ |v_i \in V-S\ \}$

V_j就是当前求得的一条从V_0出发的最短路径的终点。令$S=S\cup\{j\}$

3）修改从v_0出发到集合V-S上其它各顶点的Dist[k]值。

若 Dist[j]+G.arcs[j][k]<Dist[k]

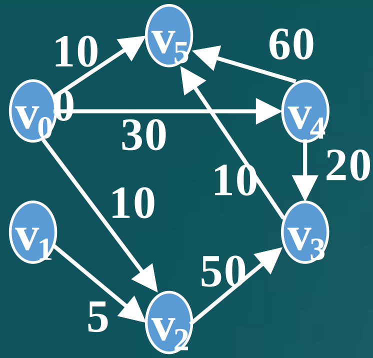
则将 Dist[k] 改为 Dist[j]+G.arcs[j][k]。

4）重复操作（2）、（3）共n-1次。由此求得从v_0到图上其余各顶点的最短路径是依路径长度递增的序列。

```
void ShortestPath_DIJ( MGraph G, int v0, PathMatrix
  &P, ShortPathTable &D) {
  //用Dijkstra算法求有向网G的v0顶点到其余顶点v的最
  短//路径P[v]及其带权长度D[v]。若P[v][w]为TRUE,则
  w是//从v0到v当前求得最短路径上的顶点，final[v]为
  TRUE当//且仅当v∈S,即已经求得从v0到v的最短路径
 for(v=0; v<G.vexnum; ++v) {
     final[v]=FALSE; D[v]=G.arcs[v0][v];
     for(w=0; w<G.vexnum; ++w)  P[v][w]=FALSE;
     //设空路径
     if(D[v]<INFINITY)
     { P[v][v0] = TRUE; P[v][v] = TRUE; }
 }//for
```

算法 7.15

```
D[v0]=0; final[v0]=TRUE;        //初始化,v0顶点属
于S集
//开始主循环，每次求得v0到某个v顶点的最短路径
，并加v到S集
for(i=0; i<G.vexnum; ++i) {             //其余
G.vexnum-1个顶点
    min=INFINITY;                  //当前所知离v0顶点
的最近距离
    for(w=0; w<G.vexnum; ++w)
      if( !final[w] )                                   //w
顶点在V-S中
         if(D[w]<min) {v=w; min=D[w]; }
                                                       //w顶
点离v0顶点更近
    final[v]=TRUE;     //离v0顶点最近的v加入S集
    for ( w=0; w<G.vexnum; ++w ) //更新当前最
短路径及距离
      if( !final[w] && ( min + G.arcs[v][w] < D[w]
)){
```



邻接矩阵为：

$$\begin{pmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

终点	从v_0到各终点的D值和最短路径的求解进程				
	i=1	i=2	i=3	i=4	i=5
v_1	∞	∞	∞	∞	∞ 无
v_2	10 (v_0,v_2)				
v_3	∞	60 (v_0,v_2,v_3)	50 (v_0,v_4,v_3)		
v_4	30 (v_0,v_4)	30 (v_0,v_4)			
v_5	100 (v_0,v_5)	100 (v_0,v_5)	90 (v_0,v_4,v_5)	60 (v_0,v_4,v_3,v_5)	
v_j	v_2	v_4	v_3	v_5	
S	(v_0,v_2)	(v_0,v_2,v_4)	(v_0,v_2,v_3,v_4)	(v_0,v_2,v_3,v_4,v_5)	

7.6.2 每一对顶点之间的最短路径

一个方法是：每次以一个顶点为源点，重复执行迪杰斯特拉算法。这样，便可求得任一对顶点之间的最短路径。

现介绍弗洛伊德（Floyd）算法，从图的带权邻接矩阵cost 出发，其基本思想是：

从 v_i 到 v_j 的所有可能存在的路径中，选出一条长度最短的路径。

若$<v_i,v_j>$存在，则存在路径$\{v_i,v_j\}$

// 路径中不含其它顶点

若$<v_i,v_1>,<v_1,v_j>$存在，则存在路径$\{v_i,v_1,v_j\}$

// 路径中所含顶点序号不大于1

若$\{v_i,\ldots,v_2\}$, $\{v_2,\ldots,v_j\}$存在，

则存在一条路径$\{v_i, \ldots, v_2, \ldots v_j\}$

// 路径中所含顶点序号不大于2

…

依次类推，在经过n次比较后，则 vi 至 vj 的最短路径应是上述这些路径中，路径长度最小者。

现定义一个n阶方阵序列

$D^{(-1)}$，$D^{(0)}$，$D^{(1)}$，…$D^{(k)}$，…$D^{(n-1)}$

其中

$D^{(-1)}[i][j]=G.arcs[i][j]$

$D^{(k)}[i][j]=Min\{D^{(k-1)}[i][j]，D^{(k-1)}[i][k]+D^{(k-1)}[k][j]\}$

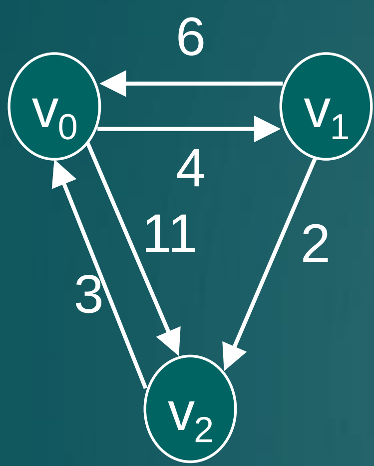
从上述计算公式可见，$D^{(1)}[i][j]$是从vi到vj的中间顶点的序号不大于1的最短路径的长度；$D^{(k)}[i][j]$是从vi到vj的中间顶点的序号不大于k的最短路径的长度；$D^{(n-1)}[i][j]$是从vi到vj的最短路径的长度。

```
void ShortestPath_FLOYD( MGraph G, PathMatrix
   &P[ ], DistancMatrix &D) {
   //用Floyd算法求有向网G中各对顶点v和w之间的最短路
   //径P[v][w]及其带权长度D[v][w]。若P[v][w][u]为
   //TRUE，则u是从v到w当前求得最短路径上的顶点。
   for(v=0; v<G.vexnum; ++v) {
      //各对结点之间初始已知路径及距离
      for(w=0; w<G.vexnum; ++w)  {
         D[v][w]=G.arcs[v][w];
         for(u=0; u<G.vexnum; ++u)
   P[v][w][u]=FALSE;
         if(D[v][w]<INFINITY)     {     //从v到w有直接路
   径
            P[v][w][u] = TRUE; P[v][w][w] = TRUE;
         }//if
      }//for
```

算法 7.16

```
for(u=0; u<G.vexnum; ++u) {
    for(v=0; v<G.vexnum; ++v)
        for(w=0; w<G.vexnum; ++w)
            if( D[v][u]+D[u][w]<D[v][w] ) {
                //从v经u到w的一条路径更短
                D[v][w]=D[v][u]+D[v][w];
                for ( i=0; i<G.vexnum; ++i )
                    P[v][w][i] = P[v][u][i] || P[u][w][i];
        }//if
}// ShortestPath_FLOYD
```

例如：　带权有向图　　　　　　　　邻接矩阵



$$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$$

利用Floyd算法求得的最短距离及最短路径如图7.37 所示。

D	$D^{(-1)}$			$D^{(0)}$			$D^{(1)}$			$D^{(2)}$		
	0	1	2	0	1	2	0	1	2	0	1	2
0	0	4	11	0	4	11	0	4	6	0	4	6
1	6	0	2	6	0	2	6	0	2	5	0	2
2	3	∞	0	3	7	0	3	7	0	3	7	0

P	$P^{(-1)}$			$P^{(0)}$			$P^{(1)}$			$P^{(2)}$		
	0	1	2	0	1	2	0	1	2	0	1	2
0		AB	AC		AB	AC		AB	ABC		AB	ABC
1	BA		BC	BA		BC	BA		BC	BCA		BC
2	CA			CA	CAB		CA	CAB		CA	CAB	

图7.37 有向图的各对顶点间的最短路径及路径长度