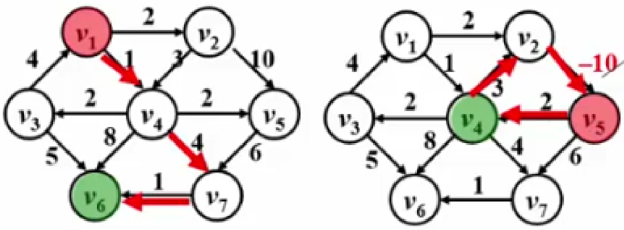
有权图的单源最短路径



负值圈
(negative-cost cycle)

- ☐ 按照递增的顺序找出到各个顶点的最短路

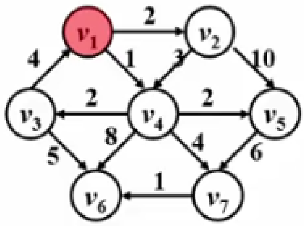
- **Dijkstra 算法**
  - 令S={源点s + 已经确定了最短路径的顶点v_i}
  - 对任一未收录的顶点v，定义dist[v]为s到v的最短路径长度，但该路径仅经过S中的顶点。即路径{s→(v_i∈S)→v}的最小长度
  - 若路径是按照递增（非递减）的顺序生成的，则
    - 真正的最短路必须只经过S中的顶点（为什么？）
    - 每次从未收录的顶点中选一个dist最小的收录（贪心）
    - 增加一个v进入S，可能影响另外一个w的dist值！
      - dist[w] = min{dist[w], dist[v] + <v,w>的权重}

- 方法1：直接扫描所有未收录顶点 – $O(|V|)$
  - $T = O(|V|^2 + |E|)$ 对于稠密图效果好

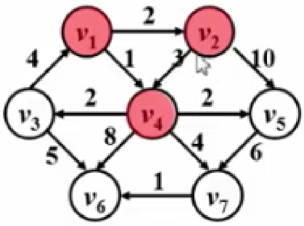
- 方法2：将dist存在最小堆中 – $O(\log|V|)$
  - 更新dist[w]的值 – $O(\log|V|)$
  - $T = O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$

对于稀疏图效果好

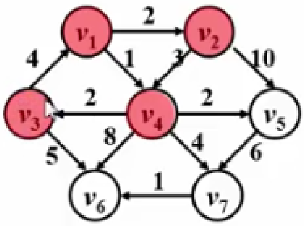
```
void Dijkstra( Vertex s )
{ while (1) {
    V = 未收录顶点中dist最小者;
    if ( 这样的V不存在 )
      break;
    collected[V] = true;
    for ( V 的每个邻接点 W )
      if ( collected[W] == false )
        if ( dist[V]+E<V,W> < dist[W] ) {
          dist[W] = dist[V] + E<V,W> ;
          path[W] = V;
        }
  }
}
```



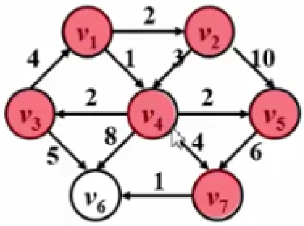
下标	1	2	3	4	5	6	7
dist	0	2	∞	1	∞	∞	∞
path	-1	1	-1	1	-1	-1	-1



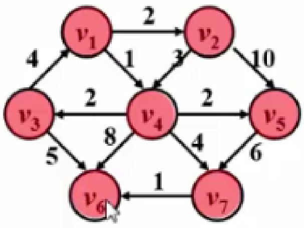
下标	1	2	3	4	5	6	7
dist	0	2	3	1	3	9	5
path	-1	1	4	1	4	4	4



下标	1	2	3	4	5	6	7
dist	0	2	3	1	3	8	5
path	-1	1	4	1	4	3	4



下标	1	2	3	4	5	6	7
dist	0	2	3	1	3	8	5
path	-1	1	4	1	4	3	4



下标	1	2	3	4	5	6	7
dist	0	2	3	1	3	6	5
path	-1	1	4	1	4	7	4

- **Floyd 算法**
  - $D^k[i][j]$ = 路径$\{ i \rightarrow \{ l \le k \} \rightarrow j \}$的最小长度
  - $D^0, D^1, \ldots, D^{|V|-1}[i][j]$即给出了i到j的真正最短距离
  - 最初的D^{-1}是什么？
  - 当D^{k-1}已经完成，递推到D^k时：
    - 或者$k \notin$最短路径$\{ i \rightarrow \{ l \le k \} \rightarrow j \}$，则$D^k = D^{k-1}$
    - 或者$k \in$最短路径$\{ i \rightarrow \{ l \le k \} \rightarrow j \}$，则该路径必定由两段最短路径组成：$D^k[i][j]=D^{k-1}[i][k]+D^{k-1}[k][j]$

```
void Floyd()
{   for ( i = 0; i < N; i++ )
      for( j = 0; j < N; j++ ) {
        D[i][j] = G[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( D[i][k] + D[k][j] < D[i][j] ) {
            D[i][j] = D[i][k] + D[k][j];
            path[i][j] = k;
          }
}
```