数据结构与算法 (七)

张铭 主讲

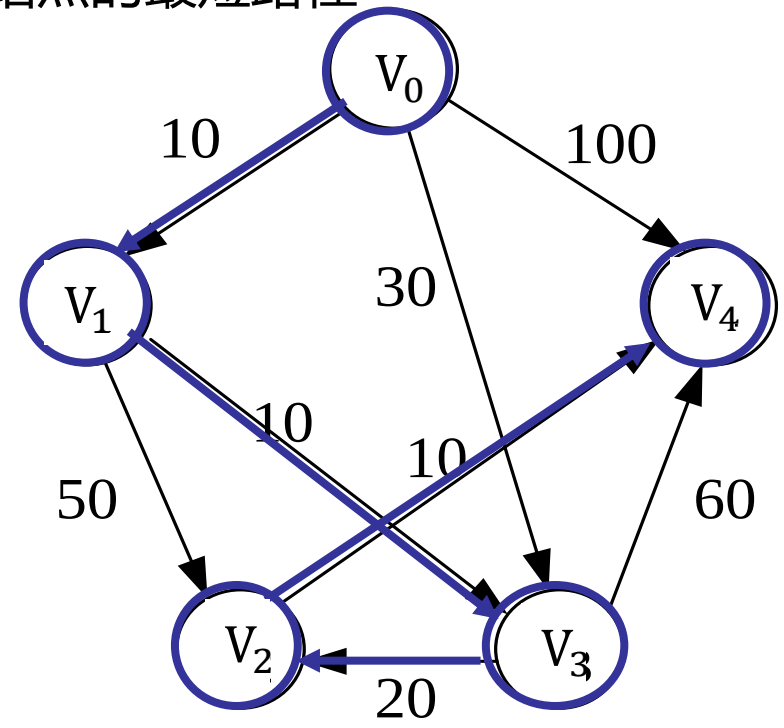
采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6（“十一五”国家级规划教材)

http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

第7章　图

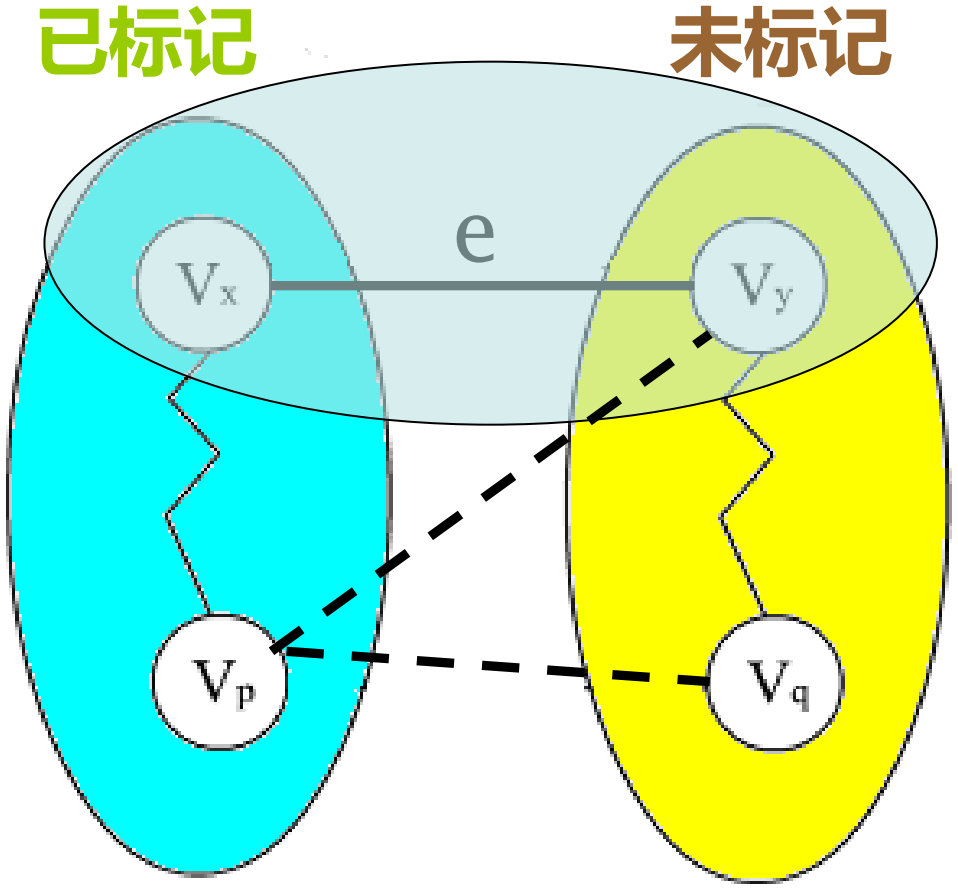
单源最短路径

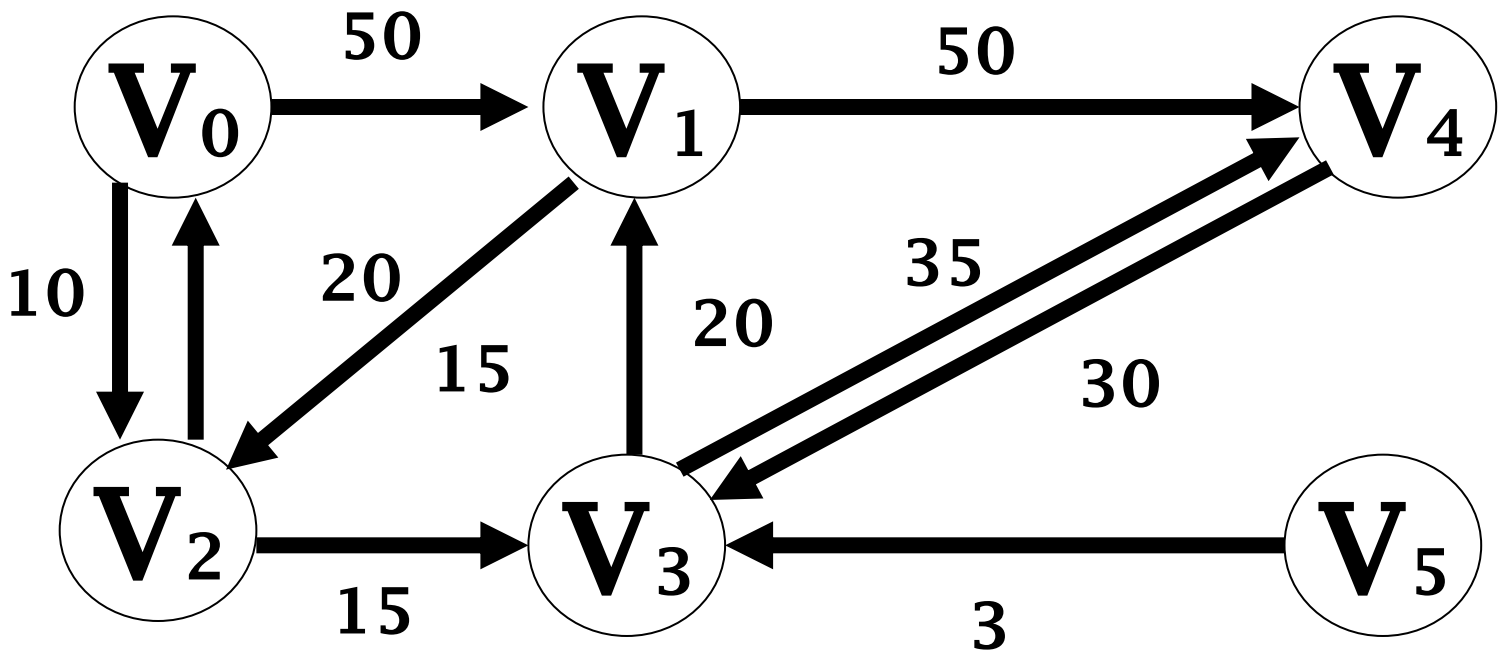
- **单源最短路径(single-source shortest paths)**
  - 给定带权图 G = <V，E>，其中每条边 (v_i，v_j) 上的权 W[v_i，v_j] 是一个 **非负实数** 。计算从任给的一个源点 s 到所有其他各结点的最短路径



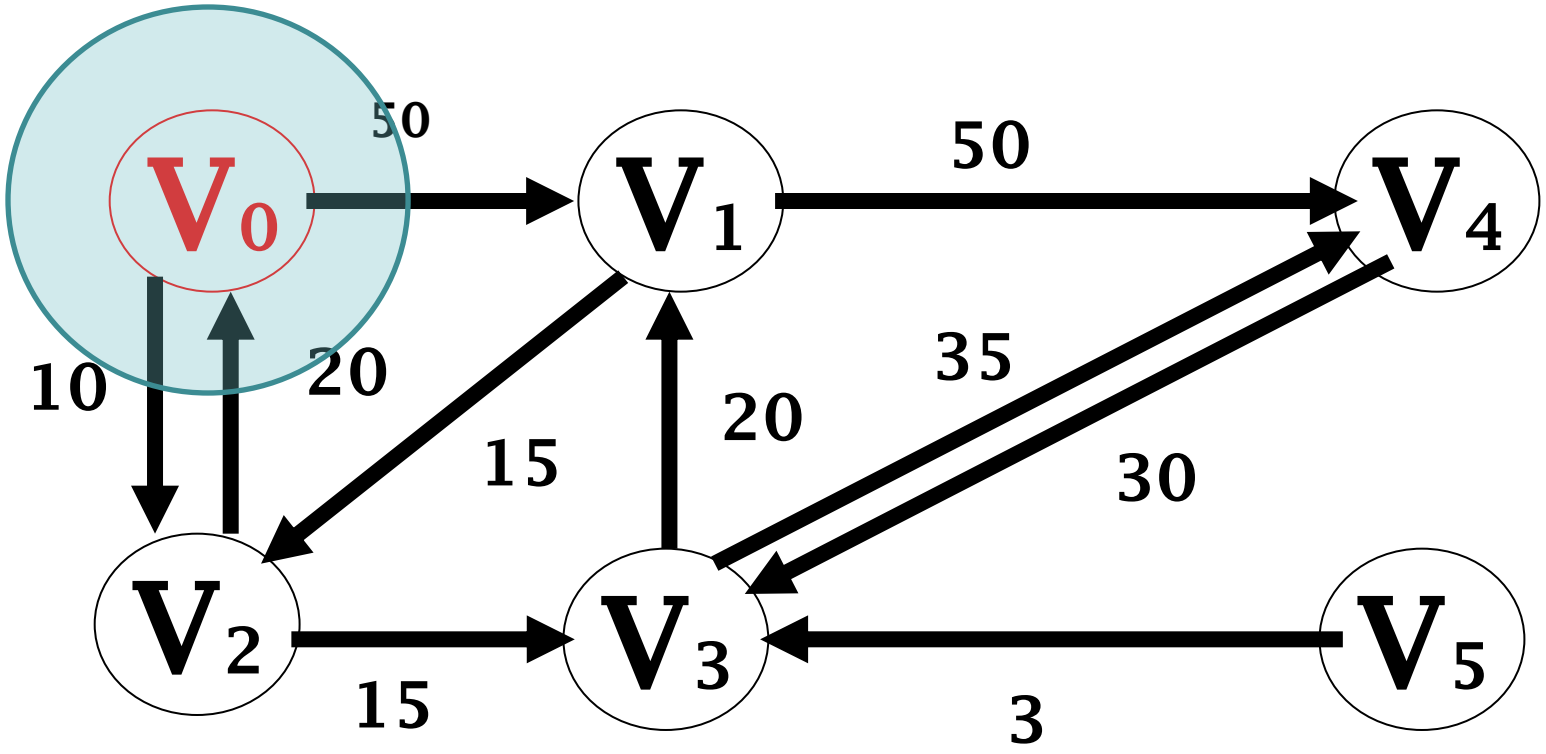
Dijkstra算法基本思想

- 把所有结点分成两组
  - 第一组 U 包括已确定最短路径的结点
  - 第二组 V-U 包括尚未确定最短路径的结点
- 按最短路径长度递增的顺序逐个把第二组的结点加到第一组中
  - 直至从 s 出发可达结点都包括进第一组中

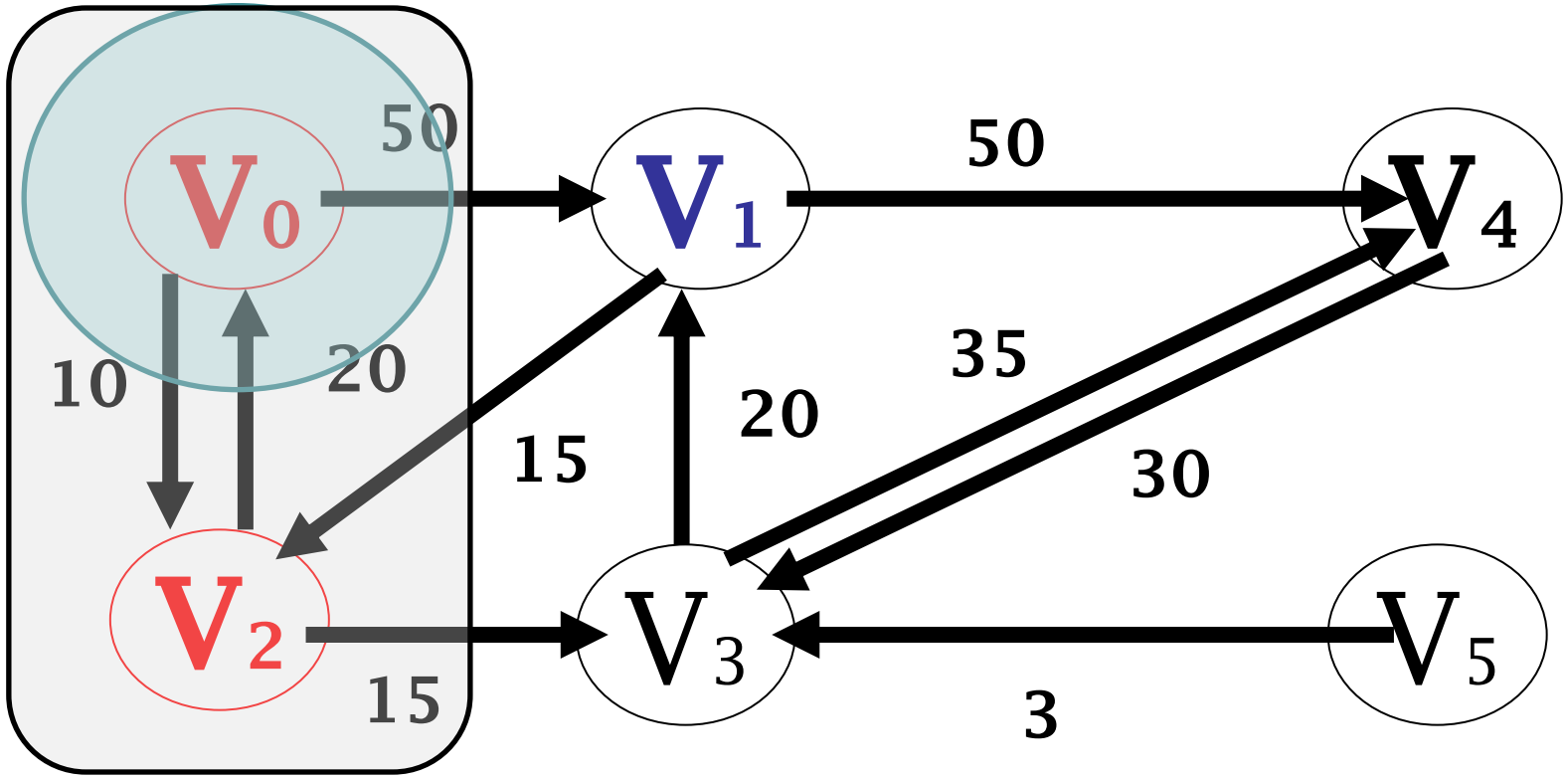




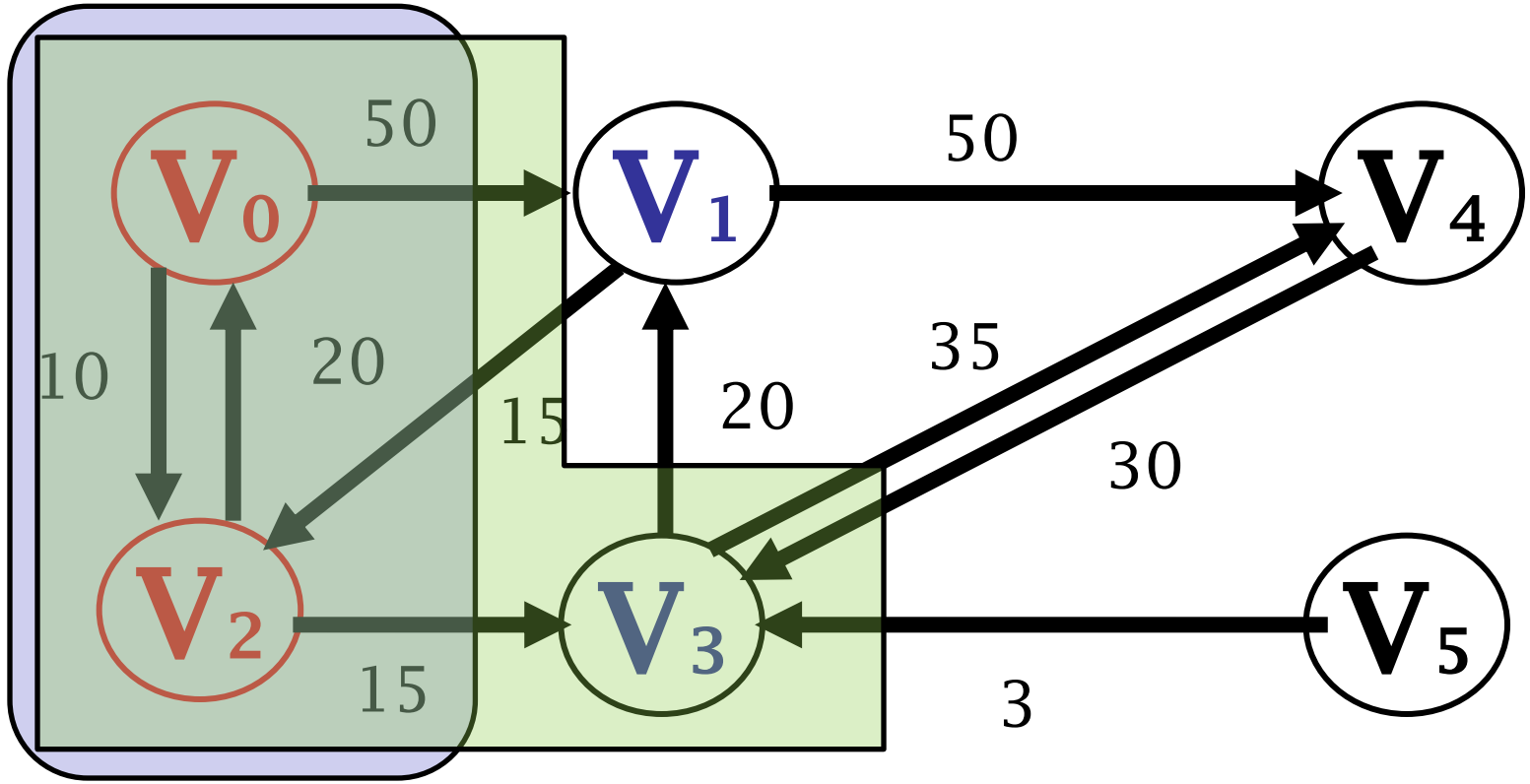
	V_0	V_1	V_2	V_3	V_4	V_5
初始状态	0 Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0



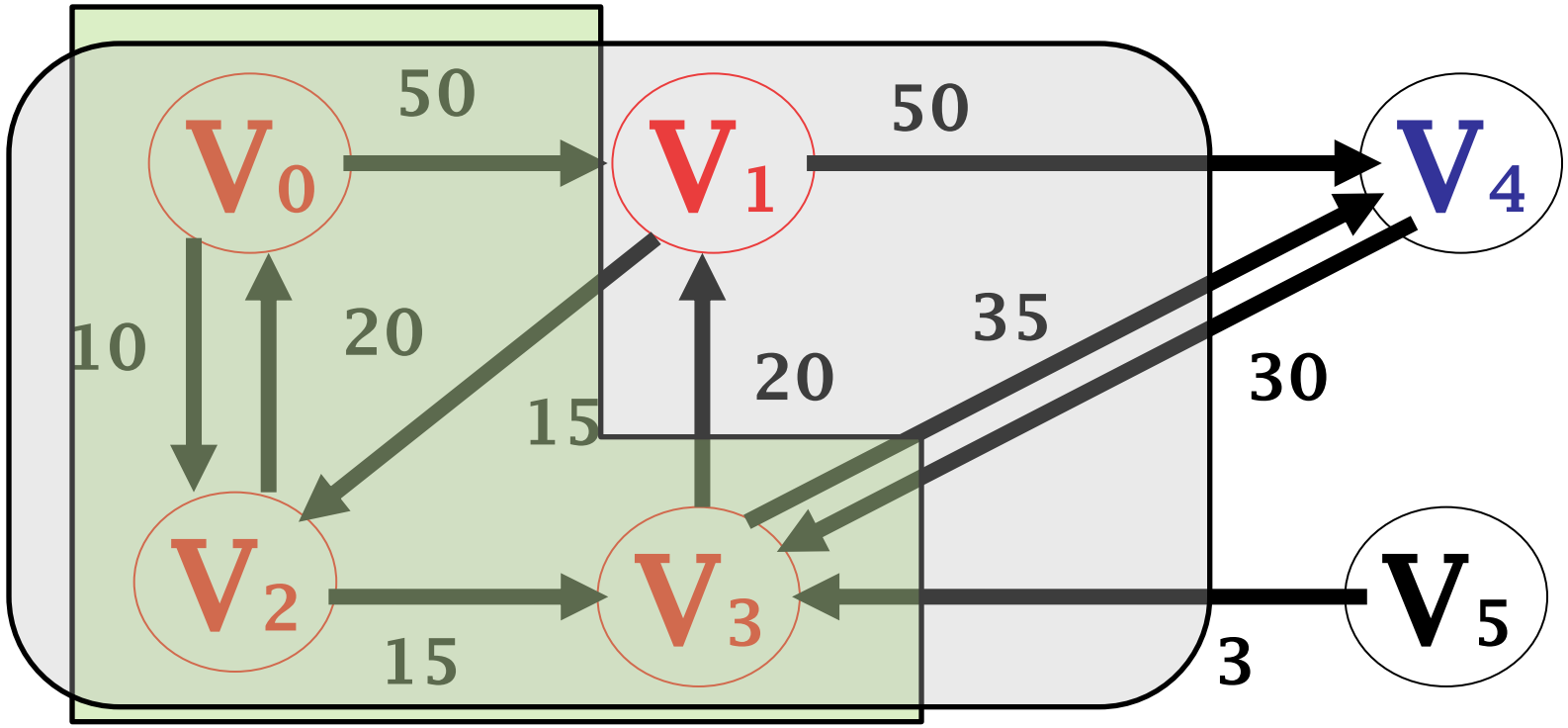
	V_0	V_1	V_2	V_3	V_4	V_5
初始	0 Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0
V_0进入第一组	0 Pre:0	50 Pre:0	10 Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0



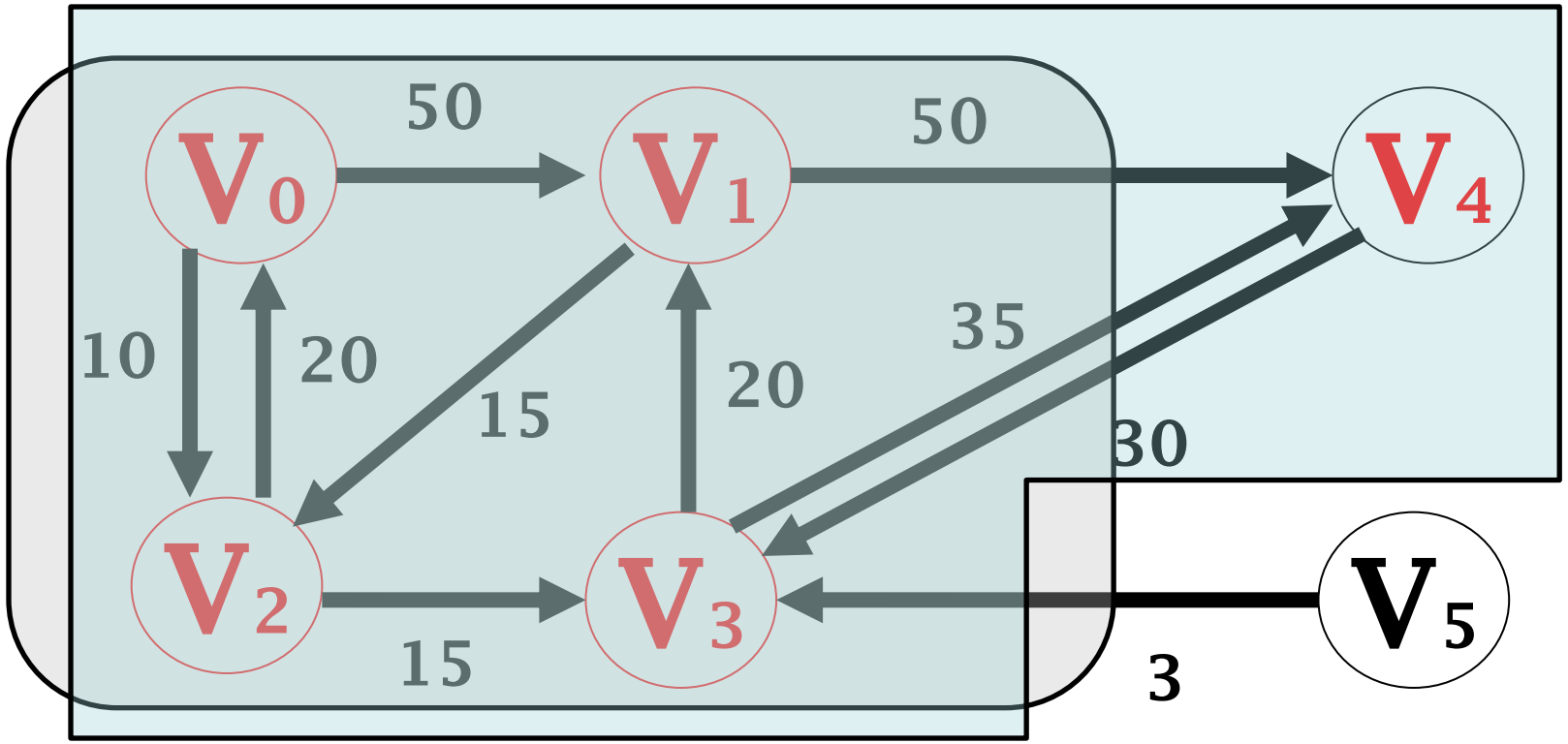
	V_0	V_1	V_2	V_3	V_4	V_5	
V_2进入之前	0 Pre:0	50 Pre:0	10 Pre:0	∞ Pre:0	∞ Pre:0	∞ Pre:0	
V_2进入第一组	0 Pre:0	50 Pre:0	10 Pre:0	25 Pre:2	∞ Pre:0	∞ Pre:0	



	V_0	V_1	V_2	V_3	V_4	V_5
V_3进入之前	0 Pre:0	50 Pre:0	10 Pre:0	25 Pre:2	∞ Pre:0	∞ Pre:0
V_3进入第一组	0 Pre:0	45 Pre:3	10 Pre:0	25 Pre:2	60 Pre:3	∞ Pre:0



	V_0	V_1	V_2	V_3	V_4	V_5
V_1进入之前	0 Pre:0	45 Pre:3	10 Pre:0	25 Pre:2	60 Pre:3	∞ Pre:0
V_1进入第一组	0 Pre:0	45 Pre:3	10 Pre:0	25 Pre:2	60 Pre:3	∞ Pre:0



	V_0	V_1	V_2	V_3	V_4	V_5
V_4进入之前	0 Pre:0	45 Pre:3	10 Pre:0	25 Pre:2	60 Pre:3	∞ Pre:0
V_4进入第一组	0 Pre:0	45 Pre:3	10 Pre:0	25 Pre:2	60 Pre:3	∞ Pre:0

Dijkstra单源最短路径迭代过程

步数	S	v_0	v_1	v_2	v_3	v_4
初始	$\{v_0\}$	Length:0 pre:0	length:50 pre:0	length:10 pre:0	length:∞ pre:0	length:∞ pre:0
1	$\{v_0, v_2\}$	Length:0 pre:0	length:50 pre:0	length:10 pre:0	length:25 pre:2	length:∞ pre:0
2	$\{v_0, v_2, v_3\}$	Length:0 pre:0	length: 45 pre:3	length:10 pre:0	length:25 pre:2	length:60 pre:3
3	$\{v_0, v_2, v_3, v_1\}$	Length:0 pre:0	length: 45 pre:3	length:10 pre:0	length:25 pre:2	length:60 pre:3
4	$\{v_0, v_2, v_3, v_1, v_4\}$	Length:0 pre:0	length: 45 pre:3	length:10 pre:0	length:25 pre:2	length:60 pre:3

Dijkstra单源最短路径算法

```
class Dist  {                    // Dist类，用于保存最短路径信息
public:
    int index;                   // 结点的索引值，仅Dijkstra算法用到
    int length;                  // 当前最短路径长度
    int pre;                     // 路径最后经过的结点
};
void Dijkstra(Graph& G, int s, Dist* &D)  {       // s是源点
    D = new Dist[G. VerticesNum()];               // 记录当前最短路径长度
    for (int i = 0; i < G.VerticesNum(); i++) {   // 初始化
        G.Mark[i] = UNVISITED;
        D[i].index = i;  D[i].length = INFINITE;  D[i].pre = s;
    }
    D[s].length = 0;                              // 源点到自身的路径长度置为0
    MinHeap<Dist> H(G. EdgesNum());               // 最小值堆用于找出最短路径
    H.Insert(D[s]);
```

7.5 最短路径

```
for (i = 0; i < G.VerticesNum(); i++) {
    bool FOUND = false;
    Dist d;
    while (!H.isEmpty())  {
        d = H.RemoveMin();                          //获得到s路径长度最小的结点
        if (G.Mark[d.index] == UNVISITED) {   //如果未访问过则跳出循环
            FOUND = true;   break;
    } }
    if  (!FOUND)  break;        // 若没有符合条件的最短路径则跳出本次循环
    int v = d.index;
    G.Mark[v] = VISITED;        // 将标记位设置为 VISITED
    for (Edge e = G.FirstEdge(v); G.IsEdge(e); e = G.NextEdge(e)) // 刷新最短路
        if (D[G.ToVertex(e)].length > (D[v].length+G.Weight(e))) {
            D[G.ToVertex(e)].length = D[v].length + G.Weight(e);
            D[G.ToVertex(e)].pre = v;
            H.Insert(D[G.ToVertex(e)]);
    } }
}
```

Dijkstra算法时间代价分析

- 每次改变D[i].length
  - 不删除，添加一个新值(更小的)，作为堆中新元素。旧值被找到时，该结点一定被标记为VISITED，从而被忽略
- 在最差情况下，它将使堆中元素数目由Θ(|V|)增加到Θ(|E|)，总的时间代价Θ((|V|+|E|) log|E|)

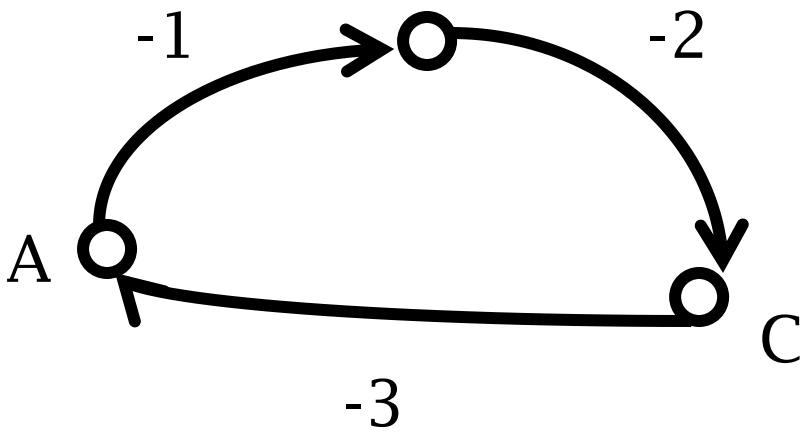
Dijkstra算法

- 是否支持
  - 有向图、无向图
  - 非连通
  - 有回路的图
  - 权值为负
- 如果不支持
  - 则修改方案？

- 针对有向图 (且“有源”)
  - 若输入无向图？
  - 照样能够处理 (边都双向)
- 对非连通图，有不可达
  - 没有必要修改
- 支持回路
- 支持负权值？

Dijkstra算法不支持负权值

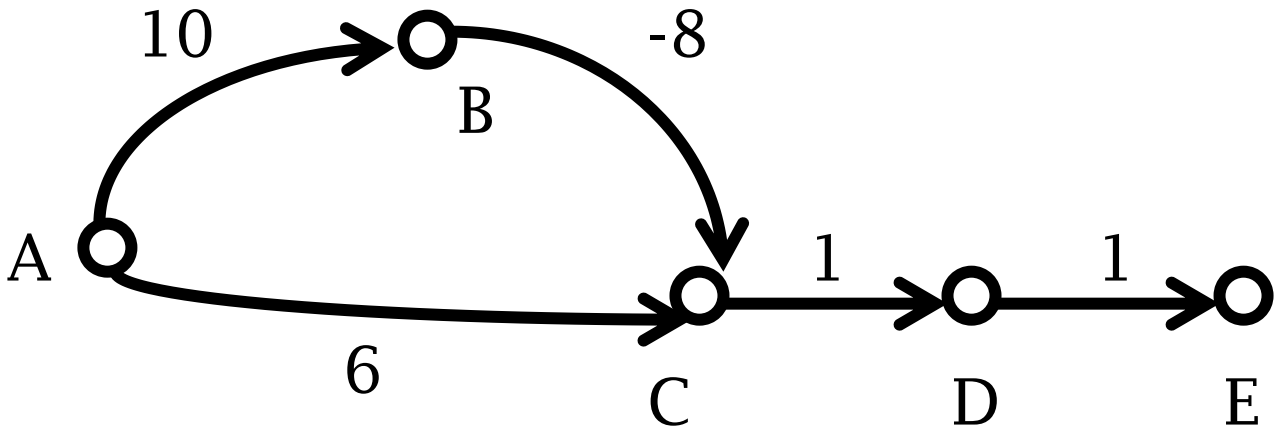
- 如果存在总权值为负的回路，则将出现权值为 -∞ 的情况



7.5 最短路径

如果不存在负权回路呢？Dijkstra算法不受负权边的影响吗？

- 即使不存在负的回路，也可能有在后面出现的负权值，从而导致整体计算错误
- 主要原因是 Dijkstra 算法是贪心法，当作最小取进来后，不会返回去重新计算



7.5 最短路径

- 持负权值的最短路径算法
  - Bellman - Ford 算法
    - 参考书 MIT “Introduction to Algorithms”
  - SPFA 算法

每对结点间的最短路径

- 还能用 Dijkstra 算法吗？
- 以每个结点为起点，调用 n 次 Dijkstra 算法

```
void Dijkstra_P2P(Graph& G) {
        Dist **D=new Dist *[G.VerticesNum()];
        for(i=0; i<G.VerticesNum(); i++)
                Dijkstra(Graph& G, i, D[i]);
}
```

Floyd算法求每对结点之间的最短路径

- 用相邻矩阵 adj 来表示带权有向图
- 基本思想：
  - 初始化 $adj^{(0)}$ 为相邻矩阵 adj
  - 在矩阵 $adj^{(0)}$上做 n 次迭代，递归地产生一个矩阵序列$adj^{(1)}$，…，$adj^{(k)}$，…，$adj^{(n)}$
  - 其中经过第k次迭代，$adj^{(k)}[i，j]$ 的值等于从结点 v_i 到结点 v_j 路径上所经过的结点序号不大于 k 的最短路径长度
- 动态规划法

最短路径组合情况分析

由于第 k 次迭代时已求得矩阵 $adj^{(k-1)}$，那么从结点 v_i 到 v_j 中间结点的序号不大于 k 的最短路径有两种情况：、

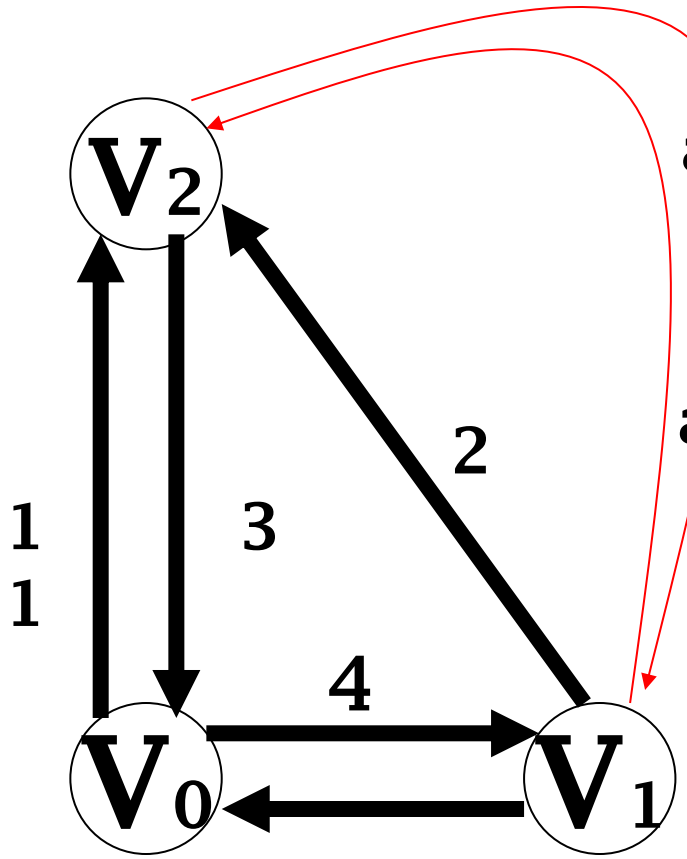
- 一种是中间不经过结点 v_k，那么此时就有

$$adj^{(k)}[i，j] = adj^{(k-1)}[i，j]$$

- 另中间经过结点 v_k，此时 $adj^{(k)}[i，j] < adj^{(k-1)}[i，j]$，那么这条由结点 v_i 经过 v_k 到结点 v_j 的中间结点序号不大于k的最短路径由两段组成

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

7.5 最短路径



$$adj = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & -1 & 2 \end{pmatrix}$$

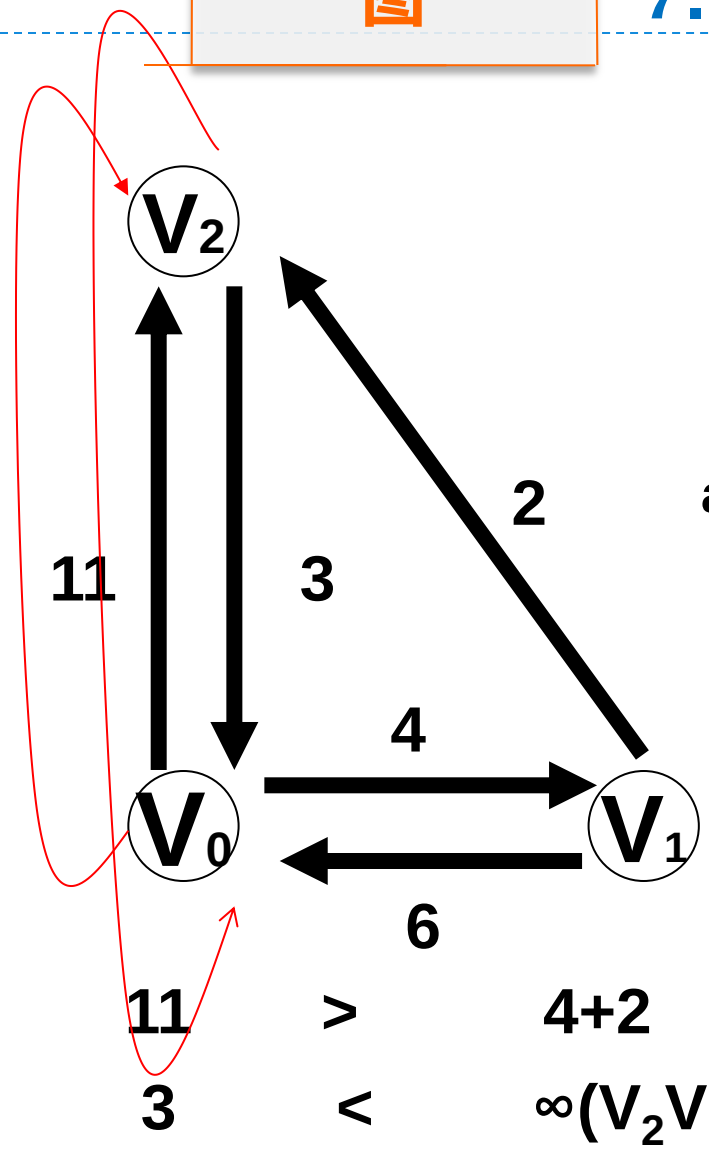
$$adj^{(0)} = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$$

$2 < 6+11$

$\infty > 3+4$

Floyd算法演示动画

7.5 最短路径



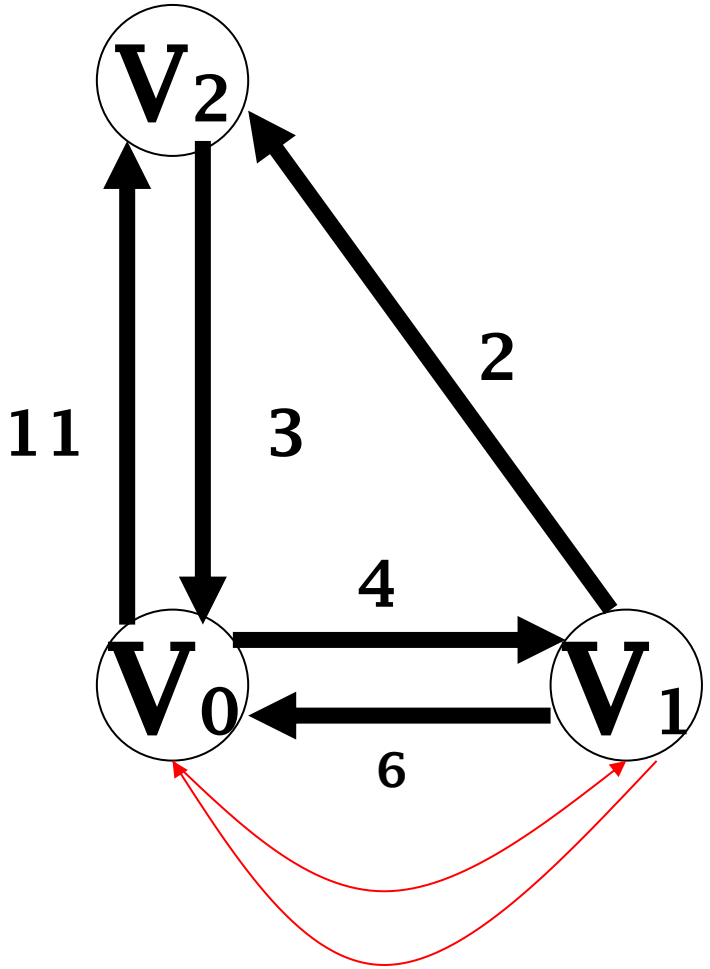
$$adj = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & -1 & 2 \end{pmatrix}$$

$$adj^{(0)} = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$$

$$adj^{(1)} = \begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$$

11 > 4+2

3 < $\infty(V_2V_1, V_1V_0)$



$4 < \infty(V_0V_2, V_2V_1)$

$6 > 2+3$

$$adj = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 2 & -1 & 2 \end{pmatrix}$$

$$adj^{(0)} = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$$

$$adj^{(1)} = \begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$$

$$adj^{(2)} = \begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \quad Path = \begin{pmatrix} 0 & 0 & 1 \\ 2 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$$

每对结点之间最短路径的Floyd算法

```
void Floyd(Graph& G, Dist** &D)  {
    int i,j,v;
    D = new Dist*[G.VerticesNum()];              // 申请空间
    for (i = 0; i < G.VerticesNum(); i++)
        D[i] = new Dist[G.VerticesNum()];
    for (i = 0; i < G.VerticesNum(); i++)        // 初始化数组D
        for (j = 0; j < G.VerticesNum(); j++) {
            if (i == j) {
                D[i][j].length = 0;
                D[i][j].pre = i;
            } else {
                D[i][j].length = INFINITE;
                D[i][j].pre = -1;
            }
        }
```

7.5 最短路径

```
    for (v = 0; v < G.VerticesNum(); v++)
        for (Edge e = G.FirstEdge(v); G.IsEdge(e); e = G.NextEdge(e)) {
            D[v][G.ToVertex(e)].length = G.Weight(e);
            D[v][G.ToVertex(e)].pre = v;
        }
    // 加入新结点后，更新那些变短的路径长度
    for (v = 0; v < G.VerticesNum(); v++)
        for (i = 0; i < G.VerticesNum(); i++)
            for (j = 0; j < G.VerticesNum(); j++)
                if (D[i][j].length > (D[i][v].length+D[v][j].length)) {
                    D[i][j].length = D[i][v].length+D[v][j].length;
                    D[i][j].pre = D[v][j].pre;
                }
}
```

Floyd算法的时间复杂度

- 三重for循环
  - 复杂度是$\Theta(n^3)$

讨论：Dijkstra 找最小 Dist 值

- 如果不采用最小堆，而采用每次遍历的方式寻找最小值，与用最小堆实现的 Dijkstra 相比，时间效率如何？

讨论：Floyd算法保持 pre 的方式

- 将“D[i][j].pre= D[v][j].pre” 改为
  “ D[i][j].pre = v” 是否可以？
  - 上述两种方案不影响 D[i][j].length 的求解
  - 对于恢复最短路径，策略有何不同？
    那种更优？

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材