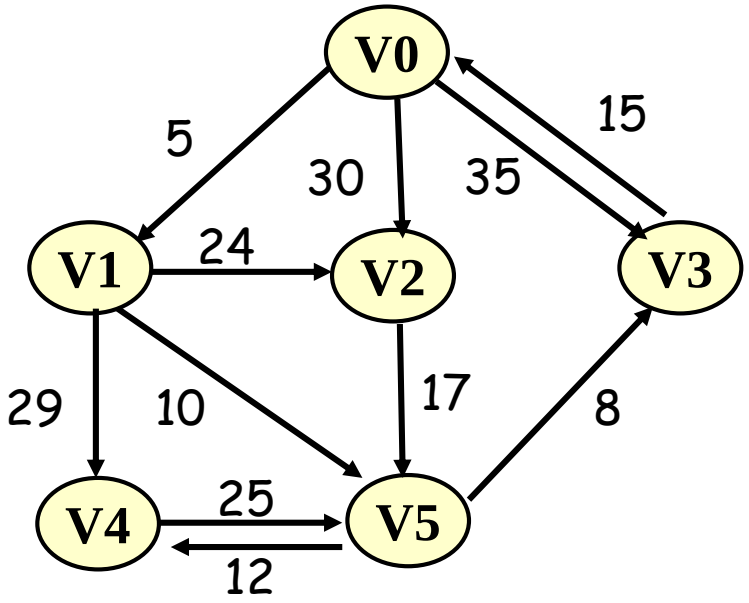
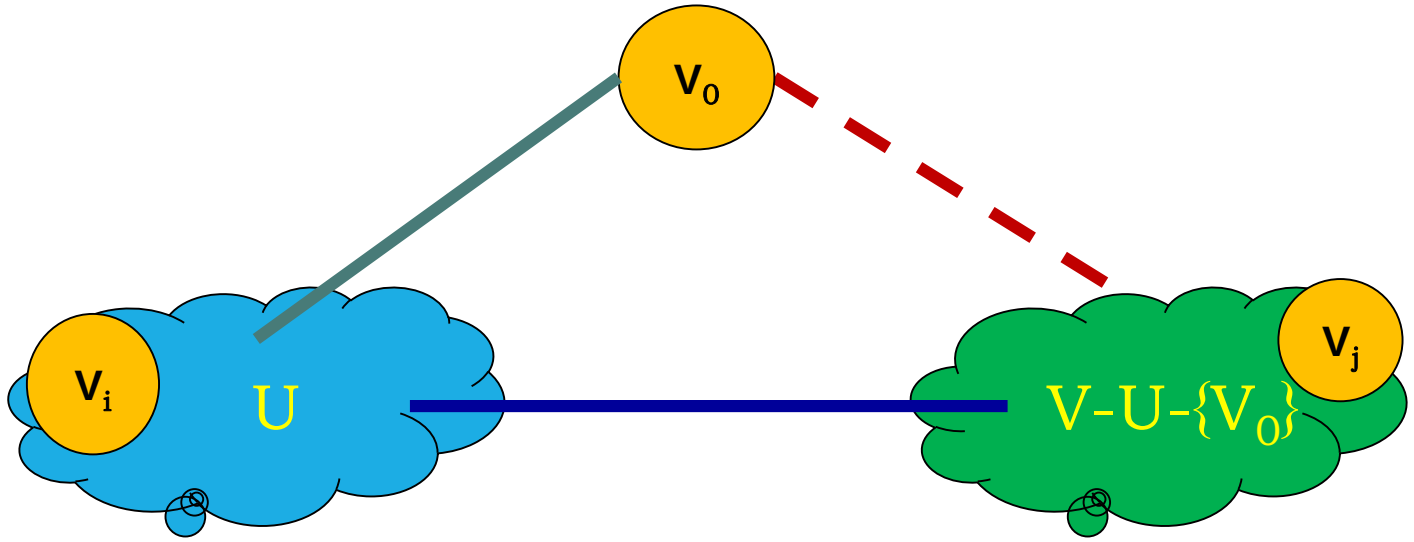
6.6 最短路径

- 从某个源点到其余各顶点的最短路径 **Dijkstra**
- 每一对顶点之间的最短路径 **Floyd**





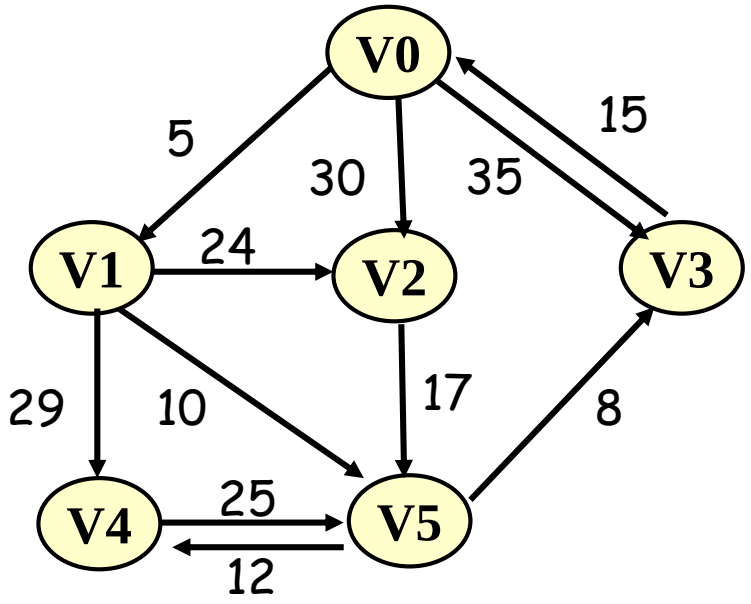
已经计算出最短路径的结点集合

待计算最短路径的结点集合

假设V0---> VX---> V1小于V0--->V1，
则V0---> VX必然小于V0---> V1
这与已知的V0---> V1最小矛盾。

6.6 最短路径

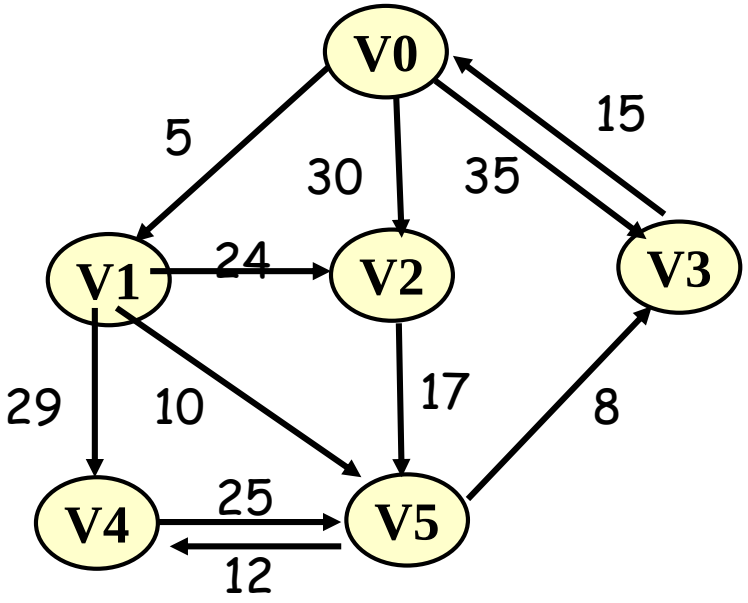
- 从某个源点到其余各顶点的最短路径 **Dijkstra**
- 每一对顶点之间的最短路径 **Floyd**



以V0为源点，可以依次得到V0到其他各个顶点的最短路径如下：

- V0->V1：最短路径为（V0,V1），最短路径长度为5
- V0->V2：最短路径（V0,V1,V2），最多路径长度为29
- V0->V3：最短路径（V0,V1,V5,V3），最短路径长度为23
- V0->V4：最短路径（V0,V1,V5,V4），最短路径长度为27
- V0->V5：最短路径（V0,V1,V5），最短路径长度为15

6.6 最短路径

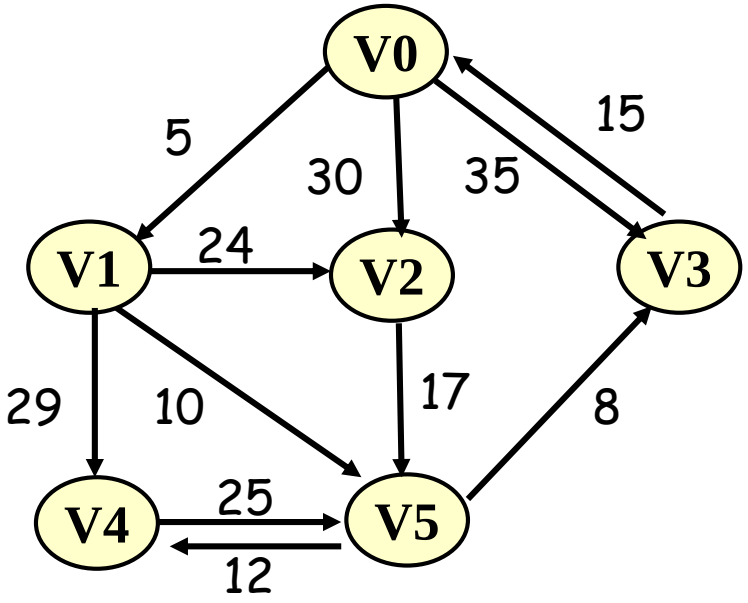


蓝色表示已确定v0到该顶点的最短路径
红色表示数组有更新

循环	S	min	源点V0到各个终点的距离distance[ ]和path[ ]数组元素变化					
			V0	V1	V2	V3	V4	V5
初始	---	---	0	5,0	30,0	35,0	∞,0	∞,0
1	{0}	1	0	5,0	29,1	35,0	34,1	15,1
2	{0,1}	5	0	5,0	29,1	23,5	27,5	15,1
3	{0,1,5}	3	0	5,0	29,1	23,5	27,5	15,1
4	{0,1,5,3}	4	0	5,0	29,1	23,5	27,5	15,1
5	{0,1,5,3,4}	2	0	5,0	29,1	23,5	27,5	15,1
6	{0,1,5,3,4,2}		0	5,0	29,1	23,5	27,5	15,1

如何确定V0到一个顶点的路径？
通过path数组得到

6.6 最短路径



循环	S	min	源点V0到各个终点的距离distance[ ]和path[ ]数组元素变化					
			V0	V1	V2	V3	V4	V5
初始	---	---	0	5,0	30,0	35,0	∞,0	∞,0
1	{0}	1	0	5,0	29,1	35,0	34,1	15,1
2	{0,1}	5	0	5,0	29,1	23,5	27,5	15,1
3	{0,1,5}	3	0	5,0	29,1	23,5	27,5	15,1
4	{0,1,5,3}	4	0	5,0	29,1	23,5	27,5	15,1
5	{0,1,5,3,4}	2	0	5,0	29,1	23,5	27,5	15,1
6	{0,1,5,3,4,2}		0	5,0	29,1	23,5	27,5	15,1

Dijkstra（迪杰斯特拉）算法：　具体实现见算法6-11

设带权图G=(V,E)，顶点集合S用来存放已经求得最短路径的所有顶点，V-S是没有确定最短路径的所有顶点集合。逐个将集合V-S中的顶点加入到集合S中，直到S中包含图中所有顶点，V-S为空集合为止。

算法中设置两个辅助数组：

- Distance[w]数组表示从顶点V0出发，且只经过S中的顶点，最终达到w的最短路径长度。Distance[w]的初值设置方式为：Distance[0]=0，，如果图中有弧<V0,Vw>，则Distance[w]为弧的权值，否则为∞。
- Found[i]数组表示集合S，如果顶点i在S中，Found[i]=TRUE，否则Found[i]=FALSE。
- 在集合V-S中选择距离最小的顶点Vmin加入到集合S中，设置Found[min]=TRUE;
- 对集合V-S中的所有顶点的距离进行更新，如果将Vmin作为中间顶点，如果使得V0到Vi的距离比原来的距离小，则更新原来的距离。
- 重复上述过程，直到S=V为止，即对所有顶点i，Found[i]=TRUE。

Dijkstra（迪杰斯特拉）算法分析：

Dijkstra算法时间复杂度：

算法中的初始化部分的时间复杂度为O(n)，

求最短路径部分由一个大循环组成，其中外循环运行n-1次，
内循环为两个，均运行n-1次。

算法的时间复杂度为$O(n^2)$

Dijkstra算法空间复杂度：

空间开销需要distance[]和path[]数组，大小为O(n)