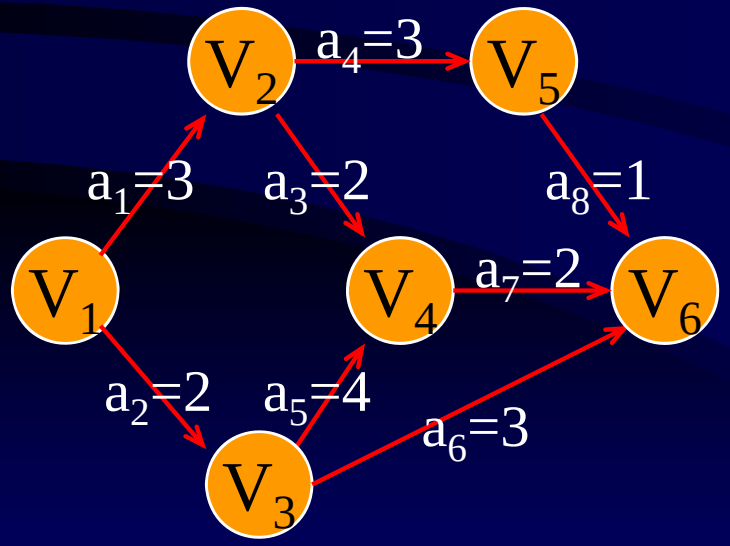
7.4图的应用（续）

有向无环图的关键路径

AOE-网描述工程，有两个问题需要解决：

*完成整项工程至少需要多少时间？

*那些活动是影响工程进度的关键？



8个活动、6个事件的AOE-网，v_1表示整个工程开始，v_6表示整个工程结束。事件v_4表示活动a_3和a_5已经完成，活动a_7可以开始

7.4图的应用（续）

- 源点：唯一的一个入度为零的点
- 汇点：唯一的一个出度为零的点
- 关键路径：路径长度（路径上各活动持续时间之和）最长的路径。这也是完成工程所需要的最短时间。
- 事件v_i最早发生时间ve(i)：从源点到v_i的最长路径长度。
- 活动a_i的最早开始时间e(i)：为活动a_i($< v_j, v_k >$)弧的弧尾事件ve(j)
- 事件v_i最迟发生时间vl(i)：不推迟整个工程完成的前提下事件v_i最迟必须开始的时间。
- 活动a_i的最迟开始时间l(i)：不推迟整个工程完成的前提下活动a_i最迟必须开始的时间。
- 关键活动：l(i) = e(i)的活动称为关键活动。关键路径上的所有活动都是关键活动。要求工程最少需要完成时间、要加快工程进度，只能考察和改进关键活动，它们是影响工程进度的关键。

7.4图的应用（续）

有向无环图的关键路径算法

思路：

1.求出AOE-网中的关键活动，即求出l(i) = e(i) 的活动a_i (< v_j, v_k >) 。

2.对活动a_i (< v_j, v_k >)，其持续时间为dut(<j,k>)，则有：

（1）e(i) = ve(j);

（2）l(i) = vl(k) – dut(<j,k>)

3. （1）从ve(0) =0开始向前递推： ve(j) = Max(i) {ve(i) + dut(<i,j>)}

<i,j>属于T, j =1,2,…,n-1；T是所有以第j个顶点为头的弧的集合

（2）从vl(n-1) = ve(n-1) 开始向后递推： vl(i) = Min(j) {vl(j) - dut(<i,j>)}

<i,j>属于S, i =n-2,…,0；S是所有以第i个顶点为尾的弧的集合

7.4图的应用（续）

有向无环图的关键路径算法

输入：AOE-网

输出：求出所有关键活动，进而求出所有关键路径

步骤：

step1.输入e条弧<j,k>，建立AOE-网的存储结构；

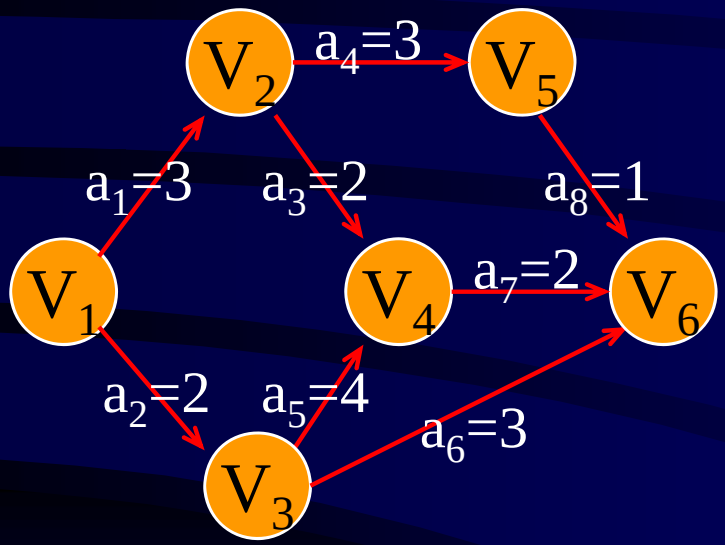
step2.从源点v_0出发，令ve[0]=0,按拓扑有序求其余各顶点的最早发生时间ve[i](1<=i<=n)。如果得到拓扑有序序列中顶点个数小于网中顶点数n，则说明网中存在环，算法终止；否则执行step3；

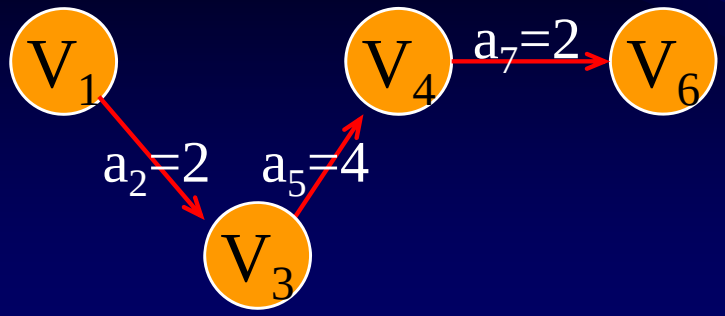
step3.从汇点v_n出发，令vl[n-1]=ve[n-1]，按逆拓扑有序求其余各顶点的最迟发生时间vl[i](n-2>=i>=2)；

step4.根据各顶点的ve和vl值，求每条弧s的最早开始时间e(s)和最迟开始时间l(s)。若某条弧满足e(s)=l(s)，则为关键活动。

7.4图的应用（续）

有向无环图的关键路径示例





顶点	ve	vl
v_1	0	0
v_2	3	4
v_3	2	2
v_4	6	6
v_5	6	7
v_6	8	8

活动	e	l	l-e
a_1	0	1	1
a_2	0	0	0
a_3	3	4	1
a_4	3	4	1
a_5	2	2	0
a_6	2	5	3
a_7	6	6	0
a_8	6	7	1

7.4图的应用（续）

*关键活动的改进可以改进工程进度；

*关键活动速度提高要有限度,前提是不能改变关键路径

*若网中有多条关键路径， 要改进工程进度，必须同时提高几条关键路径上的关键活动的速度。

7.4图的应用（续）

DS/SS

带权有向图的最短路径

- 源点：路径上的第一个顶点
- 终点：路径上的最后一个顶点
- 最短路径：

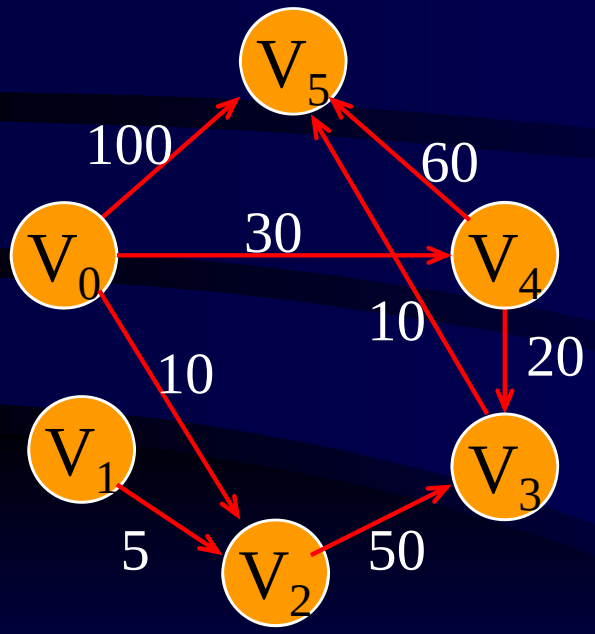
  *从源点到终点所含边的数目最少的路径。（用BFS边历方法可以求得）

  *从源点到终点路径上边的权值之和最小的路径。（用Dijkstra算法和Floyd算法可以求得）

- 从某个源点到其余各顶点的最短路径（设源点为v_0，终点可以是$v_1, v_2, \ldots, v_n$，则可以求得n条最短路径）
- 每一对顶点之间的最短路径（源点可以为$v_1, v_2, \ldots, v_n$，终点也可以是$v_1, v_2, \ldots, v_n$，源点与终点不能相等，可以求得n(n-1)条最短路径）

7.4图的应用（续）

某个源点到其余各顶点的最短路径



源点	终点	最短路径	路径长度
v_0	v_1	无	
v_0	v_2	(v_0, v_2)	10
v_0	v_3	(v_0, v_4, v_3)	50
v_0	v_4	(v_0, v_4)	30
v_0	v_5	(v_0, v_4, v_3, v_5)	60

7.4图的应用（续）

最短路径算法

思路：

1.Dijkstra: 按路径长度递增的次序产生最短路径。

2.用向量D[i]表示当前所找到的从源点v到每个终点v_i的最短路径的长度。它的初态为：若从v到v_i有弧，则D[i]为弧上权值；否则为无穷大。这时，长度为D[j] = Min(i){D[i] | v_i属于V}的路径就是从v出发的长度最短的一条最短路径。设为(v, v_j)。

3. 设S为已求得最短路径的终点的集合，则下一条最短路径（设其终点为x）或者是弧(v, 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。即下一条长度次短的最短路径的长度为D[j] = Min(i){D[i] | v_i属于V-S}，其中，D[i]或者是弧(v, v_i)上的权值，或者是D[k] (v_k属于S)和弧(v_k, v_i)上的权值之和。

7.4图的应用（续）

最短路径算法

输入：带权有向图

输出：n条最短路径

步骤：

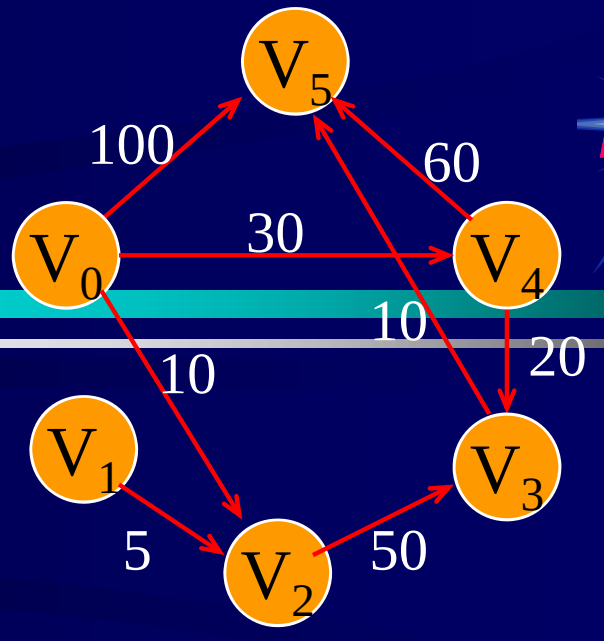
step1.从源点v出发到图上其余各顶点（终点）v_i可能达到的最短路径长度的初值为：若从v到v_i有弧，D[i]为弧上权值；否则为无穷大；

step2.选择v_j，使得D[j] = Min(i){D[i] | v_i属于V-S}，v_j就是当前求得的一条从v出发的最短路径的终点。令S = S U {j}；

step3.修改从v出发到集合V-S上任一顶点v_k可达的最短路径长度。如果D[j] +arcs[j][k]<D[k]，则修改D[k] =D[j] +arcs[j][k]；

step4.重复操作step2、step3共n-1次。由此求得从v到图上其余各顶点的最短路径是依路径长度递增的序列。

7.4图的应用（续）



终点	从v_0到各终点的D值和最短路径的求解过程				
	i=1	i=2	i=3	i=4	i=5
v_1	无穷大	无穷大	无穷大	无穷大	无穷大(无)
v_2	10 (v_0, v_2)				
v_3	无穷大	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	$\{v_0, v_2\}$	$\{v_0, v_2, v_4\}$	$\{v_0, v_2, v_3, v_4\}$	$\{v_0, v_2, v_3, v_4, v_5\}$	

7.4图的应用（续）

每一对顶点之间的最短路径

两种方法：

*每次以一个顶点为源点，重复执行Dijkstra算法n次，便可求得每一对顶点之间最短路径。$T(n) = O(n^3)$

*Floyd算法。 $T(n) = O(n^3)$