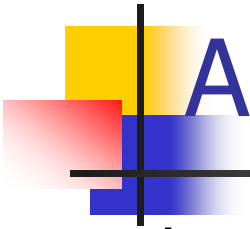


Activity-on-Edge (AOE) Networks

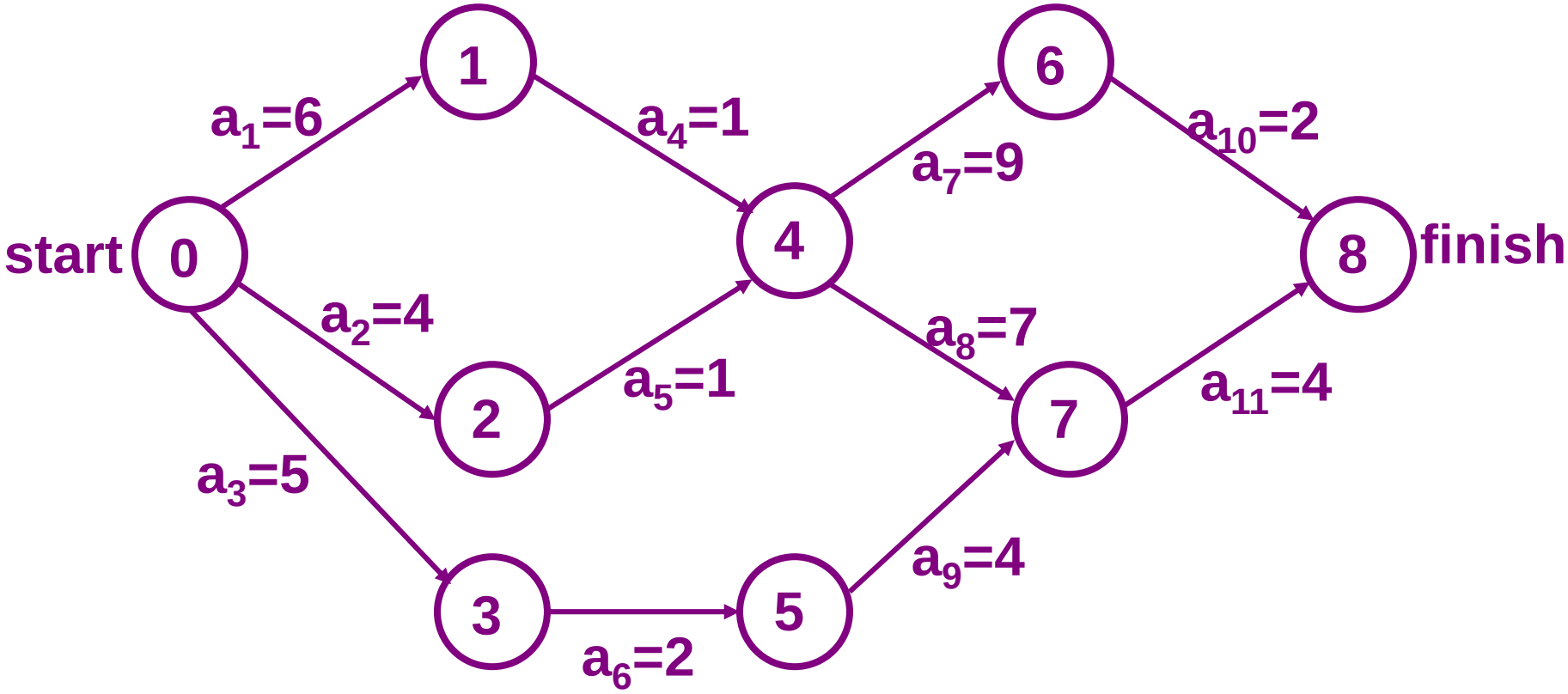
The activity-on-edge (AOE) network:

• directed edges --- tasks to be performed

• vertices --- events, signaling the completion of certain activities.

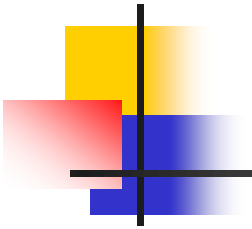
• activities represented by edges leaving a vertex cannot be started until the event at that vertex has occurred.

• an event occurs only when all activities entering it have been completed.

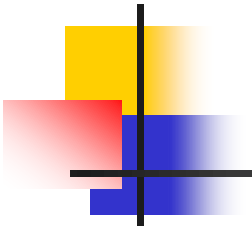

start
0
1
2
3
4
5
6
7
8
finish
a_1=6
a_2=4
a_3=5
a_4=1
a_5=1
a_6=2
a_7=9
a_8=7
a_9=4
a_{10}=2
a_{11}=4

AOE

- 如果在无有向环的带权有向图中,
  - 用有向边表示一个工程中的活动 (Activity)
  - 用边上权值表示活动持续时间 (Duration)
- 　　用顶点表示事件 (Event)
- 这样的有向图叫做用边表示活动的网络, 简称 AOE (Activity On Edges) 网络

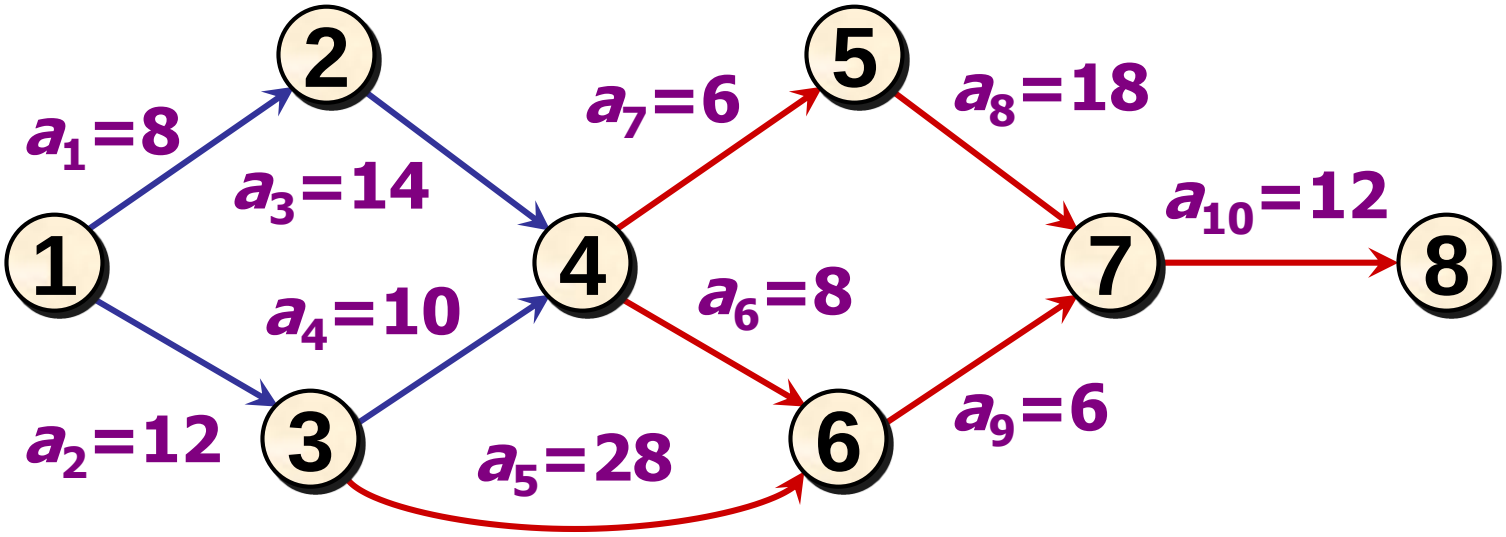


• Since activities in an AOE network can be carried out in parallel, the minimum time to complete the project is the length of the longest path from the start to the finish.



- 可以使人们了解：
  - 完成整个工程至少需要多少时间(假设网络中没有环)?
  - 为缩短完成工程所需的时间, 应当加快哪些活动?
- 从源点到各个顶点, 以至从源点到汇点的有向路径可能不止一条。这些路径的长度也可能不同
- 完成不同路径的活动所需的时间虽然不同, 但只有各条路径上所有活动都完成了, 整个工程才算完成
- 完成整个工程所需的时间取决于从源点到汇点的最长路径长度, 即在这条路径上所有活动的持续时间之和
- 这条路径长度最长的路径叫做关键路径(Critical Path)

- 如果：找出关键路径，必须找出关键活动，
- 即：不按期完成就会影响整个工程完成的活动。
- 关键路径上的所有活动都是关键活动。
- 因此，只要找到了关键活动，就可以找到关键路径。



与计算关键活动有关的量

1. $Ve(i)$：事件V_i的最早可能开始时间

   是从源点V_0到顶点V_i的最长路径长度。

2. $Vl[i]$：事件V_i的最迟允许开始时间

   是在保证汇点V_{n-1}在$Ve[n-1]$时刻完成的前提下，事件V_i的允许的最迟开始时间。

3. $e[k]$：活动a_k的最早可能开始时间

   设活动a_k在边$<V_i, V_j>$上，则$e[k]$是从源点V_0到顶点V_i的最长路径长度。

   $e[k] = Ve[i]$

4. $l[k]$：活动a_k的最迟允许开始时间

$l[k]$是在不会引起时间延误的前提下，该活动允许的最迟开始时间。

$$l[k] = Vl[j] - dur(<i, j>)$$

其中，$dur(<i, j>)$是完成 a_k 所需的时间。

5. $l[k]-e[k]$：时间余量

表示活动a_k的最早可能开始时间和最迟允许开始时间的时间余量。

$l[k] == e[k]$，表示活动 a_k 是没有时间余量的关键活动

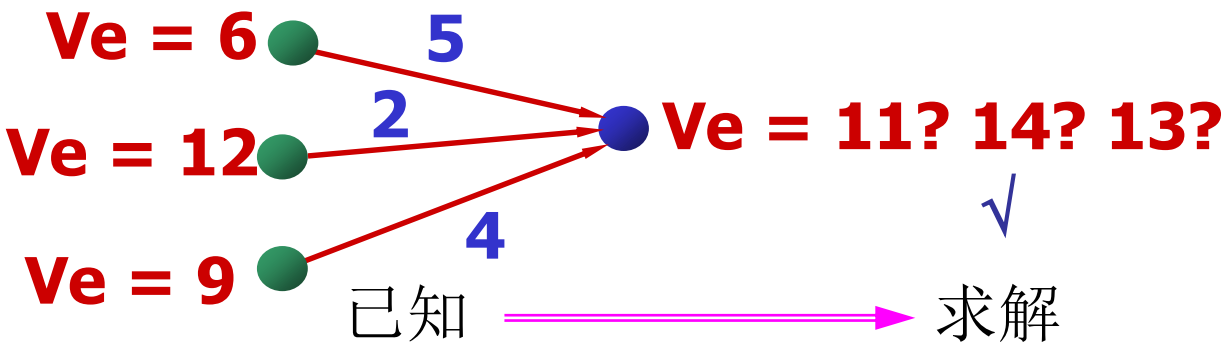
为了找出关键活动，则*求各个活动的 e[k] 与 l[k]，以判别是否 l[k] == e[k]。*

- 为求得：$e[k]$与$l[k]$,
- 需要先求得：从源点V_0到各个顶点V_i的$Ve[i]$和$Vl[i]$。
- 求$Ve[i]$的递推公式
  - 从 $Ve[0] = 0$ 开始，向前递推

$$Ve[j] = \max_i \{Ve[i] + dur(<V_i, V_j>)\},$$

$$<V_i, V_j> \in S2, \quad j = 1, 2, \ldots, n\text{-}1$$

$S2$ 是所有指向V_j的有向边$<V_i,V_j>$的集合。

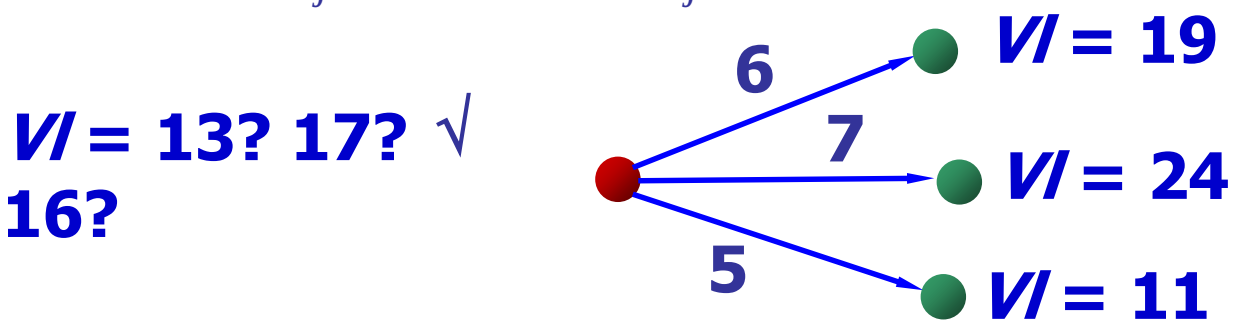


- 从$Vl[n\text{-}1] = Ve[n\text{-}1]$开始，反向递推

$$Vl[j] = \min_{k}\{Vl[k] - dur(<V_j, V_k>)\},$$

$< V_j, V_k > \in S1, j = n\text{-}2, n\text{-}3, \ldots, 0$

$S1$是所有源自V_j的有向边$< V_j , V_k>$的集合。



求解 ⟸ 已知

- 这两个递推公式的计算必须分别在拓扑有序及逆拓扑有序的前提下进行。

设：

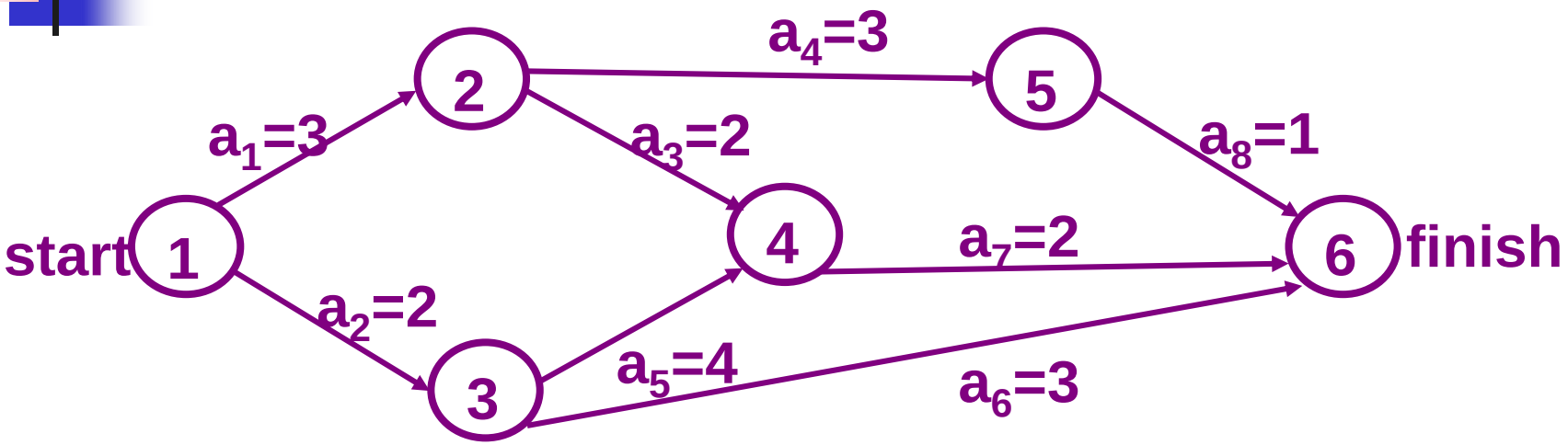
- 活动a_k (k=1, 2, …, e)在带权有向边< V_i, V_j >上,
- 其持续时间用dur (<V_i,V_j>)表示,
- 则有

  $e[k] = Ve[i]$；

  $l[k] = Vl[j] - dur(<V_i,V_j>)$； $k = 1, 2, \ldots, e$。

  这样，就得到计算关键路径的算法。

- 为了简化算法,
- 假定：在求关键路径之前已经对各顶点实现了拓扑排序,并按拓扑有序的顺序对各顶点重新进行了编号。



	1	2	3	4	5	6
Ve(i)	0	3	2	6	6	8
Vl(i)	0	4	2	6	7	8

	a1	a2	a3	a4	a5	a6	a7	a8
e(i)	0	0	3	3	2	2	6	6
l(i)	1	0	4	4	2	5	6	7
l(i)-e(i)	1	0	1	1	0	3	0	1

利用关键路径法求AOE网的各关键活动

```
template <class T, class E>
void CriticalPath(graph<T, E>& G)
{
    int i, j, k;   E Ae, Al, dur;
    int n = G.NumberOfVertices();
    E *Ve = new E[n];  E *Vl = new E[n];
    for (i = 0; i < n; i++) Ve[i] = 0;
    for (i = 0; i < n; i++) {        //正向计算Ve[]
        j = G.getFirstNeighbor(i);
        while (j != -1) {
            dur = G.getWeight (i, j);
```

```
        if (Ve[i]+dur > Ve[j]) Ve[j] = Ve[i]+dur;
        j = G.getNextNeighbor(i, j);
    }
}
Vl[n−1] = Ve[n−1];
for (j = n−2; j > 0; j−−) {        //逆向计算Vl[]
    k = G.getFirstNrighbor (j);
    while (k != −1) {
        dur = G.getWeight (j, k);
        if (Vl[k]−dur < Vl[j]) Vl[j] = Vl[k]−dur;
        k = G.getNextNeighbor (j, k);
    }
}
```

```
    for (i = 0; i < n; i++) {                    //求各活动的e, l
        j = G.getFirstNeighbor (i);
        while (j != -1) {
            Ae = Ve[i];  Al = Vl[k]-G.getWeight(i, j);
            if (Al == Ae)
                cout << "<" << G.getValue(i) << ","
                        << G.getValue(j) << ">"
                       << "是关键活动" << endl;
             j = G.getNextNeighbor (i, j);
        }
    }
    delete [] Ve;  delete [] Vl;
}
```

注意！！

- 所有顶点按拓扑有序的次序编号
- 仅计算 $Ve[i]$ 和 $Vl[i]$ 不够，还须计算 $e[k]$ 和 $l[k]$
- 不是任一关键活动加速一定能使整个工程提前
- 想使整个工程提前，要考虑各个关键路径上所有关键活动
- 如果任一关键活动延迟，整个工程就要延迟

The END