数据结构

第十三讲 拓扑排序和关键路径

孙猛

http://www.math.pku.edu.cn/teachers/sunm

2017年11月16日

Slides on Model Checking

- **https://moves.rwth-aachen.de/teaching/ss-16/ss16introduction-to-model-checking/**

课程内容

- 拓扑排序
- 关键路径

拓扑排序

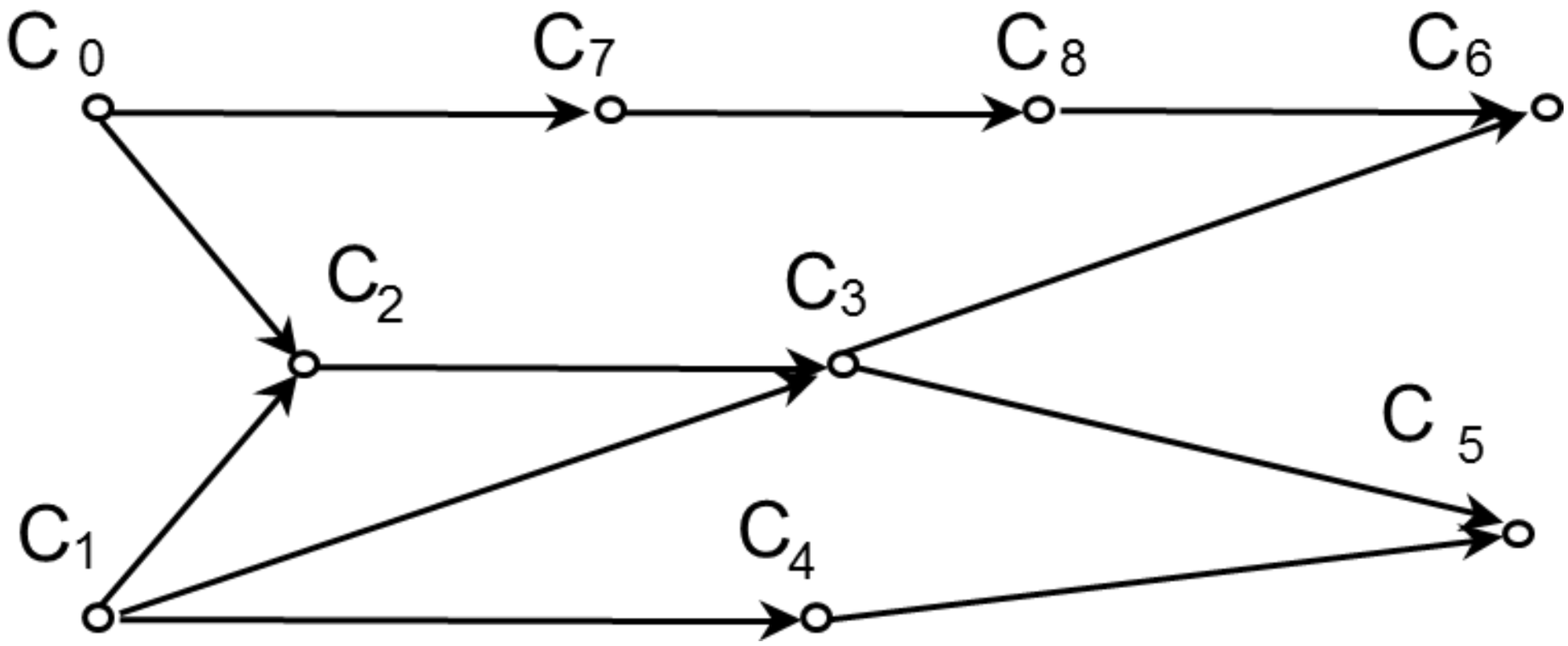
- **AOV网**
  - **若在有向图中，用顶点表示活动，边表示活动间的某种约束关系，这样的有向图称为顶点活动网(Activity On Vertex network，简称AOV网)。**
  - **AOV网被用于各种工程计划等。**
  - **问题：需要根据这种制约关系做出有关活动的安排。**
  - **在一些问题里，AOV网上的顶点或边还可能带有权值。**

计算机专业学生的课程安排

- 主要活动就是完成各门课程教学工作

课程代码	课程名称	先修课程
C_0	数学分析	
C_1	计算概论	
C_2	离散数学	C_0, C_1
C_3	数据结构	C_1, C_2
C_4	算法	C_1
C_5	编译原理	C_3, C_4
C_6	操作系统	C_3, C_8
C_7	普通物理	C_0
C_8	计算机原理	C_7

AOV网举例



拓扑排序

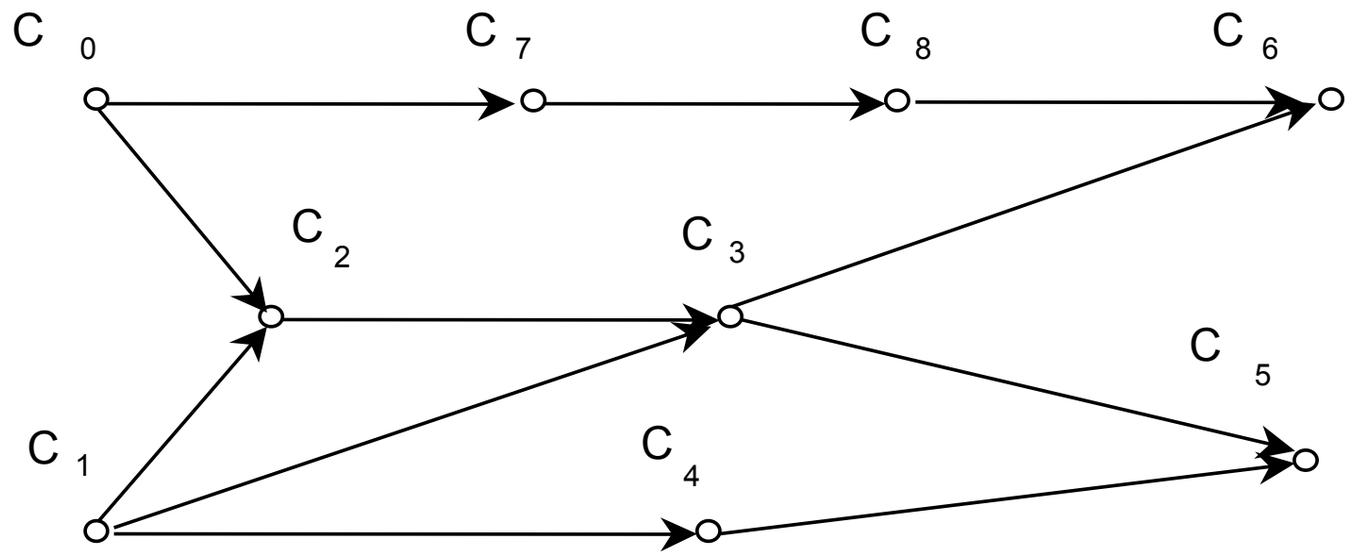
- 对于一个AOV网，如果图中所有顶点可以排成一个线性序列 v_{i1}，v_{i2}，…，v_{in}，并且该线性序列具有以下性质：如果在AOV网中，从顶点v_i到顶点v_j存在一条路径，则在线性序列中，顶点v_i一定排在顶点v_j之前。就把这种序列称为拓扑序列，把构造拓扑序列的操作称为拓扑排序。
- 如果限制各个活动只能串行，那么它的AOV网的拓扑序列就是整个工程得以顺利完成的一种可行方案。
- 一个AOV网的拓扑序列不一定存在。
- 一个AOV网的拓扑序列不是唯一的。

拓扑排序思想

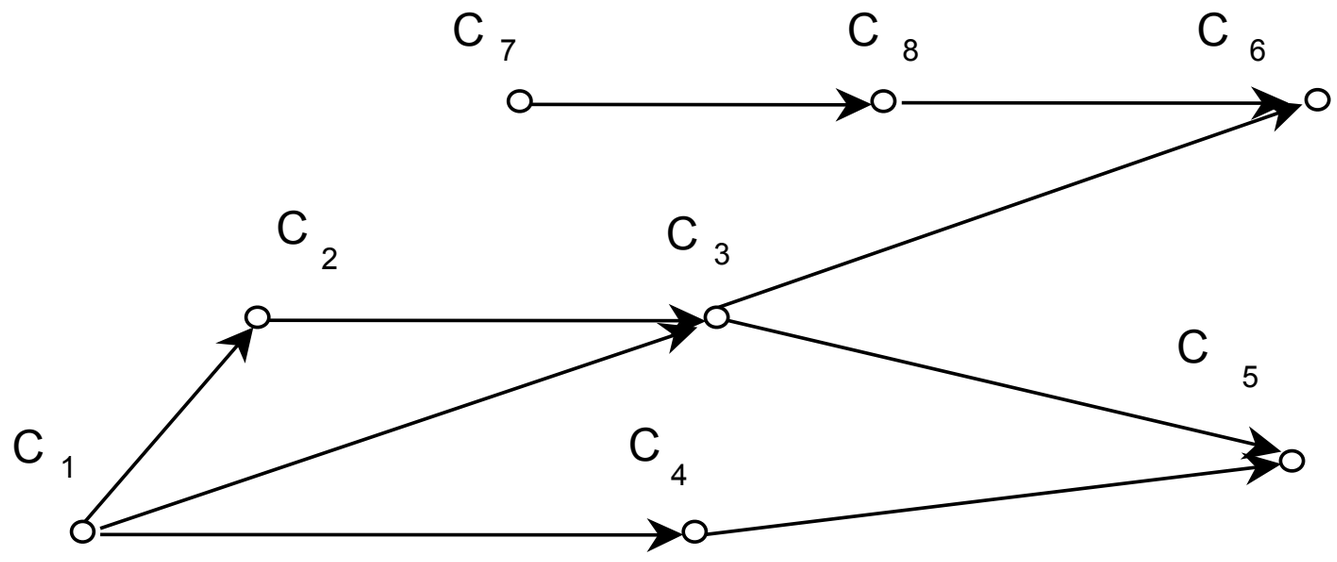
（1）从AOV网中选择一个入度为0的顶点将其输出。

（2）在AOV网中删除此顶点及其所有的出边。

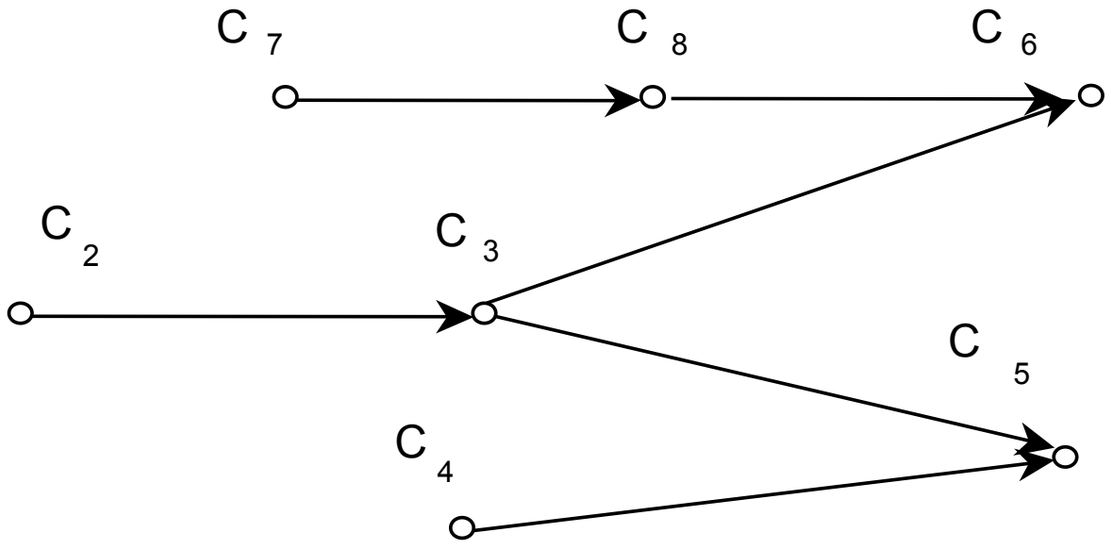
- **反复执行以上两步，直到所有顶点都已经输出为止，此时整个拓扑排序完成；或者直到剩下的顶点的入度都不为0为止，此时说明AOV网中存在回路，拓扑排序无法再进行。一个AOV网的拓扑序列不一定存在。**



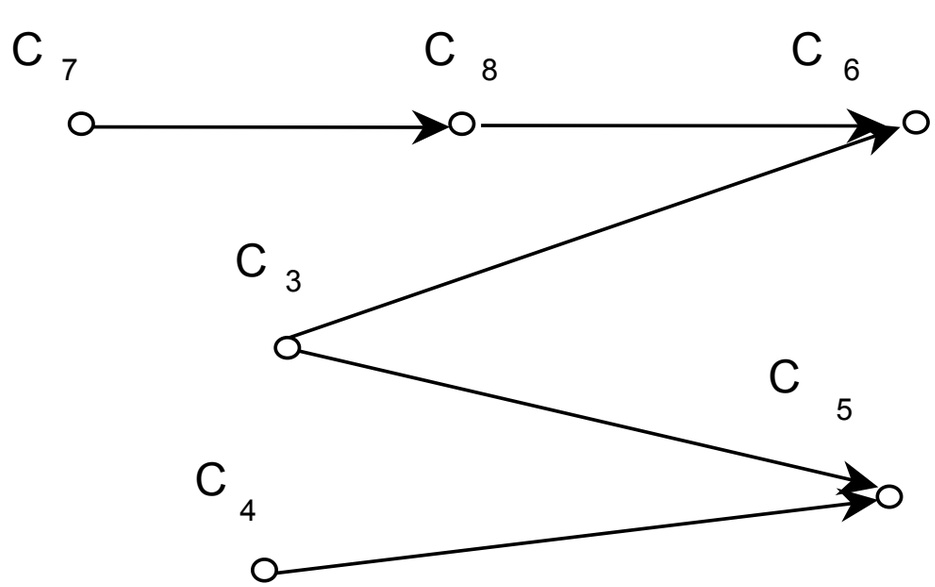
C_0



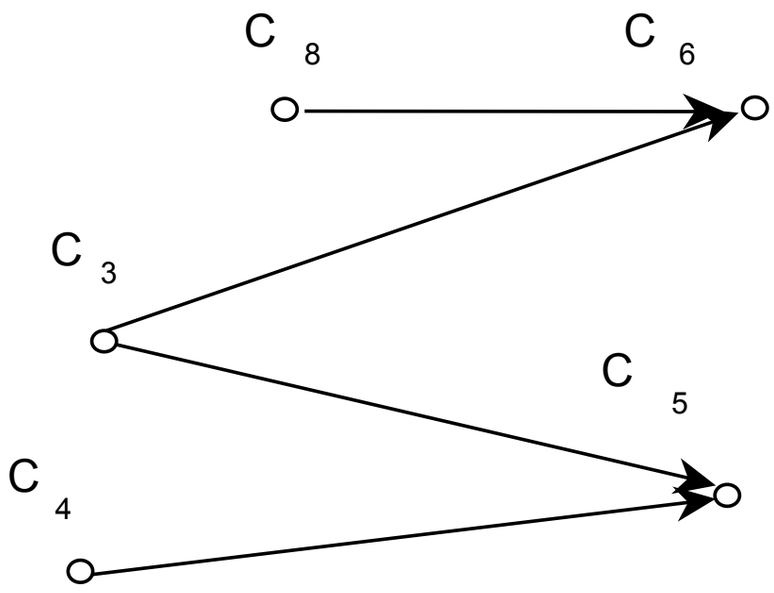
C_0, C_1,



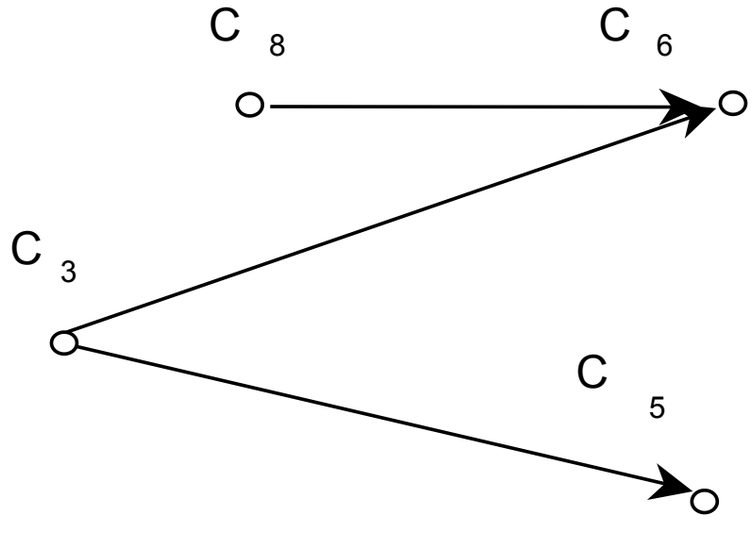
C_0, C_1, C_2,



$C_0, C_1, C_2, C_7,$



$C_0, C_1, C_2, C_7, C_4,$



出现多个入度为0的顶点时，
就出现了一次任意选择

$C_0, C_1, C_2, C_7, C_4, C_3,$

拓扑序列：

$C_0, C_1, C_2, C_7, C_4, C_3, C_5, C_8, C_6$

数据结构

- **AOV网采用邻接出边表表示**

- **辅助空间**

int * proposed

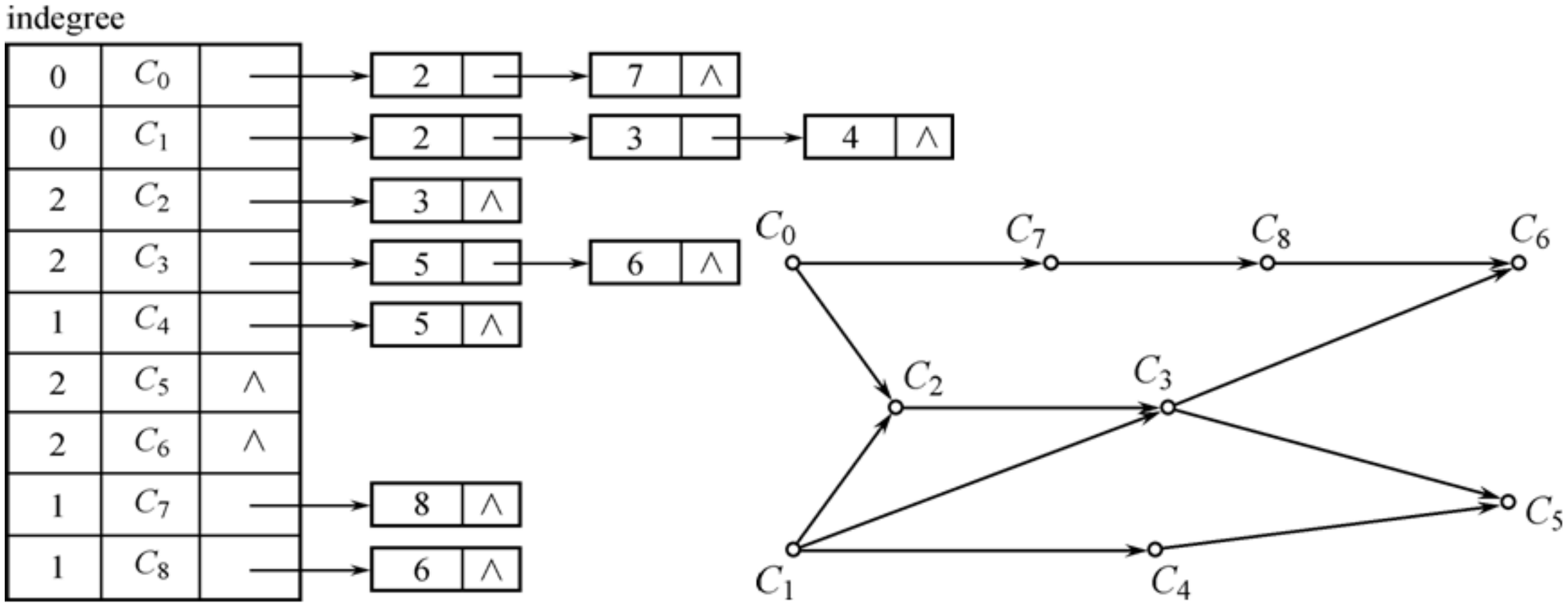
记录拓扑排序序列顶点在顶点表中的下标值

int * indegree

存放各顶点的入度，兼作入度为0顶点栈的空间

采用邻接出边表表示AOV网

- **增加一个表indegree，存放各顶点的入度**



拓扑排序算法的精化

- 拓扑排序前，先调用findInDegree得到所有结点的入度，然后将所有入度为0的顶点压栈（栈采用链接表示，空间利用数组inDegree，栈底元素存－1，栈顶位置用变量top保存）。
- 从栈顶取出一个顶点将其输出，由它的出边表可以得到以该顶点为起点的出边，将这些边终点的入度减1，即删除这些边。
- 如果某条边终点的入度为0，则将该顶点入栈。
- 反复进行上述操作，直到栈为空。
- 如果这时输出的顶点个数小于n，则说明该AOV网中存在回路，否则，拓扑排序正常结束。

统计顶点入度

整数数组 inDegree 记录各顶点入度

还在其中实现一个栈，保存入度为 0 的顶点，简化后续操作。栈底元素存 –1，栈顶位置用变量 top 保存。

findInDegree 统计顶点入度：

```
void findInDegree(GraphList* g, int *inDegree){
    int i;
    EdgeList p;
    for (i = 0; i < VN; i++) inDegree[i] = 0;
    for (i = 0; i < VN; i++)
        for (p = g->vexs[i].edgelist; p != NULL; p = p->nextedge)
            ++inDegree[p->endvex]; /* 邻接终点的入度加一 */
}
```

拓扑排序算法

```
int topoSort(GraphList * paov, Topo * ptopo) {
    int i, nd = 0, top = -1;
    int indegree[VN];
    findInDegree(paov, indegree); /* 求图所有顶点的入度 */
    for (i = 0; i < VN; ++ i)  /* 入度0的顶点入栈 */
        if (indegree[i] == 0) {
            indegree[i] = top; top = i;
        }
    /* 拓扑排序，将顶点序列记入ptopo，返回记入的顶点数 */
    nd = topoList (paov, ptopo, indegree, top);
    if (nd < VN) { /* AOV网中存在回路 */
        printf ("The AOV has no topo-sort sequence.\n");
        return 0;
    }
    return 1;
}
```

拓扑排序算法

```
int topoList(GraphList *paov, int *ptopo, int indegree[], int top) {
    EdgeList p;
    int i, k, nd = 0;
    while ( top != -1) {   /* 有入度0的顶点 */
        i = top;
        top = indegree[top];  /* top 顶点退栈 */
        ptopo[nd++] = i;  /* 将 i 记入拓扑序列 */









    }
    return nd;
}
```

拓扑排序算法

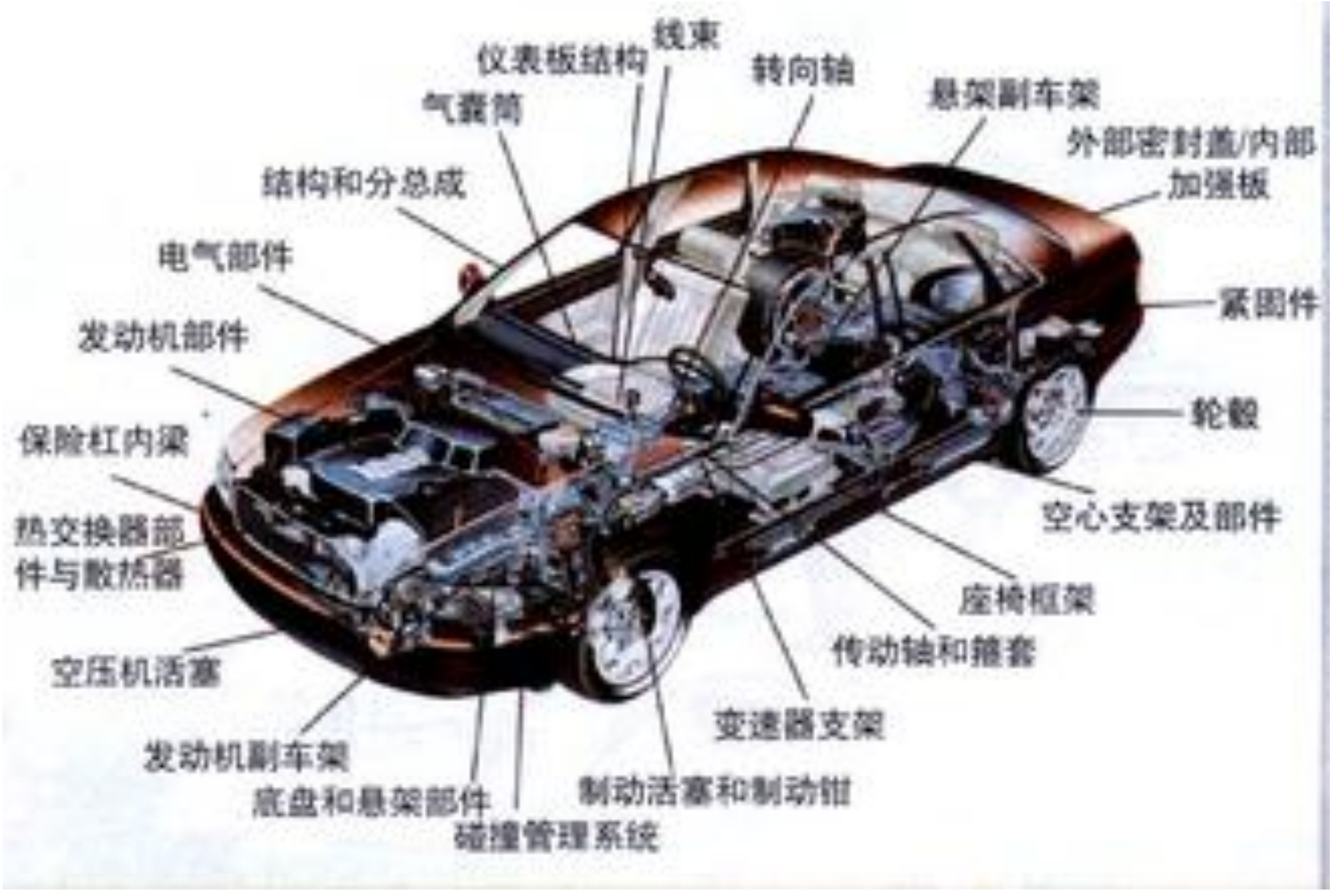
```
int topoList(GraphList *paov, int *ptopo, int indegree[], int top) {
    EdgeList p;
    int i, k, nd = 0;
    while ( top != -1) {   /* 有入度0的顶点 */
        i = top;
        top = indegree[top];  /* top 顶点退栈 */
        ptopo[nd++] = i;  /* 将 i 记入拓扑序列 */
        for (p = paov->vexs[i].edgelist; p != NULL; p = p->nextedge) {
            k = p->endvex;
            if (--indegree[k] == 0) {
                indegree[k] = top; top = k; /* 入度0的顶点入栈 */
            }
        }
    }
    return nd;
}
```

算法分析

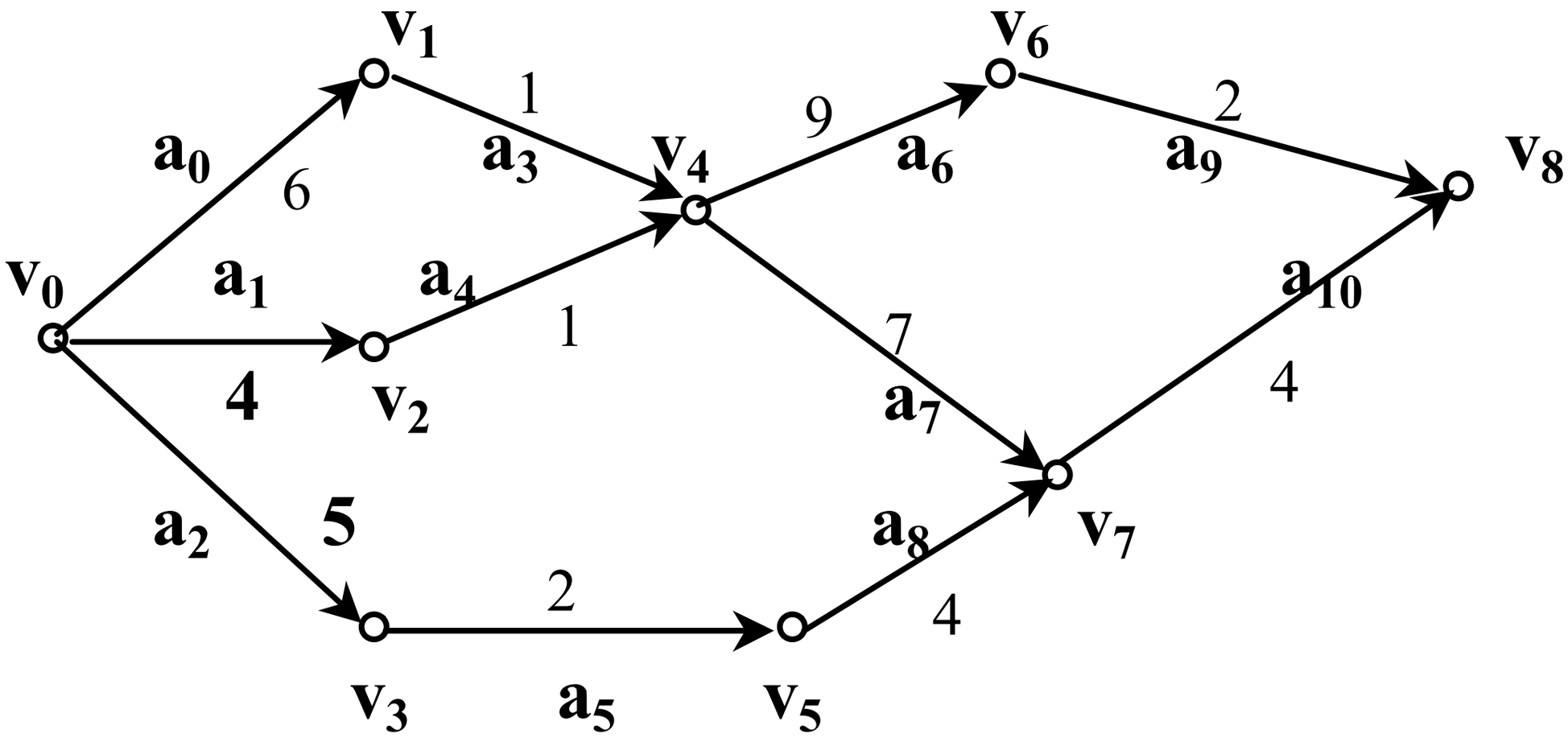
- 设AOV网有n个顶点，e条边。
- 算法初置每个顶点的入度，其中需要检查每条边一次，而后检查入度为零的顶点。总花费的时间为 $O(n+e)$。
- 在拓扑排序的主要工作部分，每个顶点至多连接到 top 链里一次，而且每个边结点都被检查一次（且至多检查一次），时间代价也为$O(n+e)$。
- 因此，拓扑排序算法的时间复杂度为$O(n+e)$。
- 也可以用邻接矩阵实现这一算法，这时：
- 设置入度需要遍历矩阵，初始准备的时间为$O(n^2)$。拓扑排序中的工作时间也是 $O(n^2)$ 。

关键路径

- 如果在带权的有向图中，用顶点表示状态 （也称为事件），用有向边表示活动，边上的权值表示活动的开销，则此带权的有向图称为边活动网 (Activity On Edge network)，简称AOE网。
- 顶点所表示的状态，实际上就是它的入边所表示的活动都已完成，它的出边所表示的活动可以开始这样一种状态。

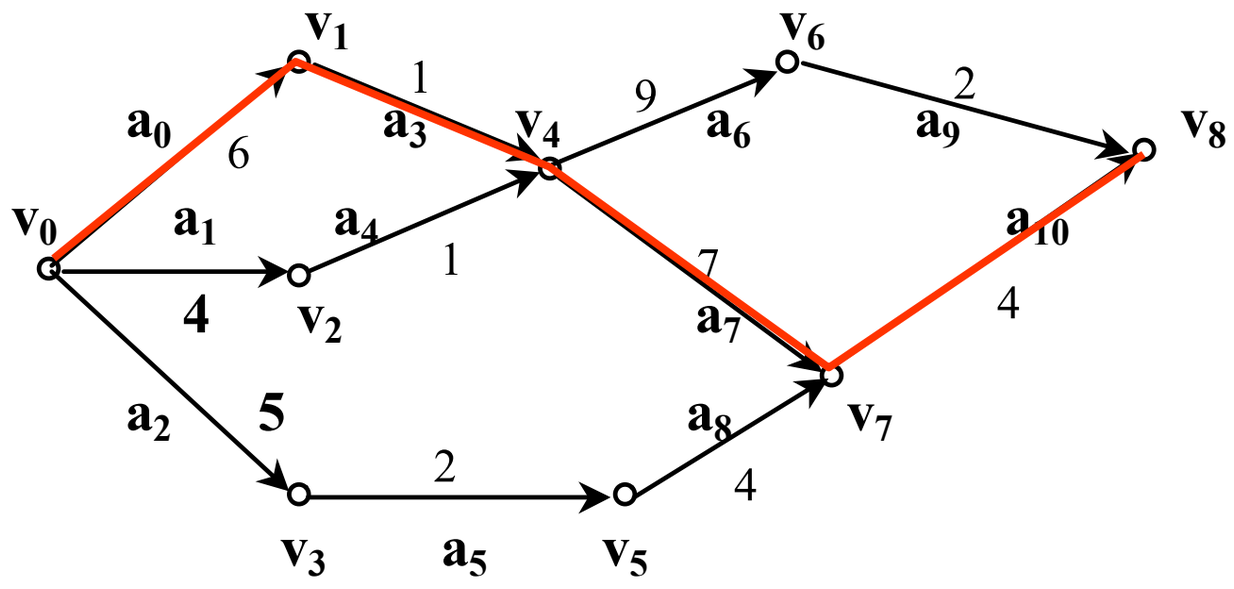

线束
仪表板结构
转向轴
悬架副车架
气囊筒
外部密封盖/内部
加强板
结构和分总成
电气部件
紧固件
发动机部件
轮毂
保险杠内梁
空心支架及部件
热交换器部
件与散热器
座椅框架
传动轴和箍套
空压机活塞
变速器支架
发动机副车架
制动活塞和制动钳
底盘和悬架部件
碰撞管理系统

例子


v1
v6
a0
6
1
a3
v4
9
a6
2
a9
v8
v0
a1
a4
1
7
a10
4
4
v2
a7
a2
5
a8
v7
2
4
v3
a5
v5

关键路径

AOE网中的活动可以并行进行，只要活动的前提事件发生，活动就可以开始。所以完成整个工程的最短时间就是从开始顶点到完成顶点的最长路径长度（路径上各边的权值之和）。
从开始顶点到完成顶点的最长路径称为关键路径。



v_0, v_1, v_4, v_7, v_8 是一条关键路径，长度为18，也就是说，完成整个工程至少需要 18 天。

关键活动的计算

- 关键活动
  - 状态v_j可能的最早发生时间$ee(j)$
  - 状态v_i允许的最迟发生时间$le(i)$
  - 活动a_k=< v_i ,v_j>的最早开始时间$e(k)$
  - 活动a_k=< v_i ,v_j>的最晚开始时间$l(k)$

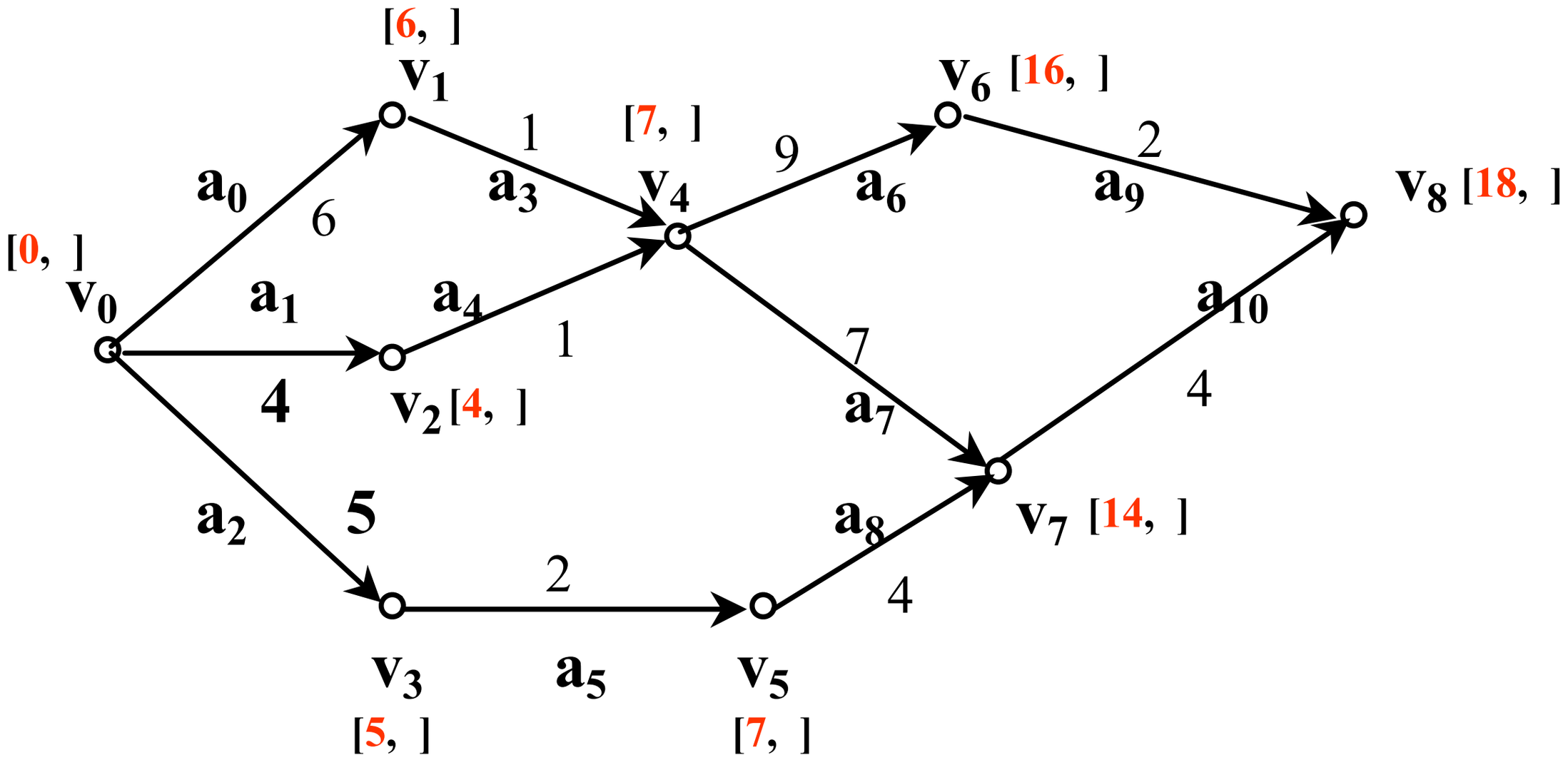
关键活动就是$e(k) = l(k)$的活动。

- $l(k)-e(k)$表示完成活动a_k的时间余量，是在不延误工期的前提下，活动a_k可以延迟的时间。

状态v_j可能的最早发生时间$ee(j)$

- 是从开始顶点到顶点v_j的最长路径长度。这个时间也是所有从v_j出发的有向边所代表的活动能够开工的最早时间。
- $ee(0)=0$
- $ee(j)=\max\{ee(i)+\text{weight}(< v_i, v_j >)\}$
- $< v_i, v_j > \in T,\ 1 \le j \le n-1$
- 其中T是所有以v_j为终点的入边的集合，$\text{weight}(< v_i, v_j >)$为边$< v_i, v_j >$的权。

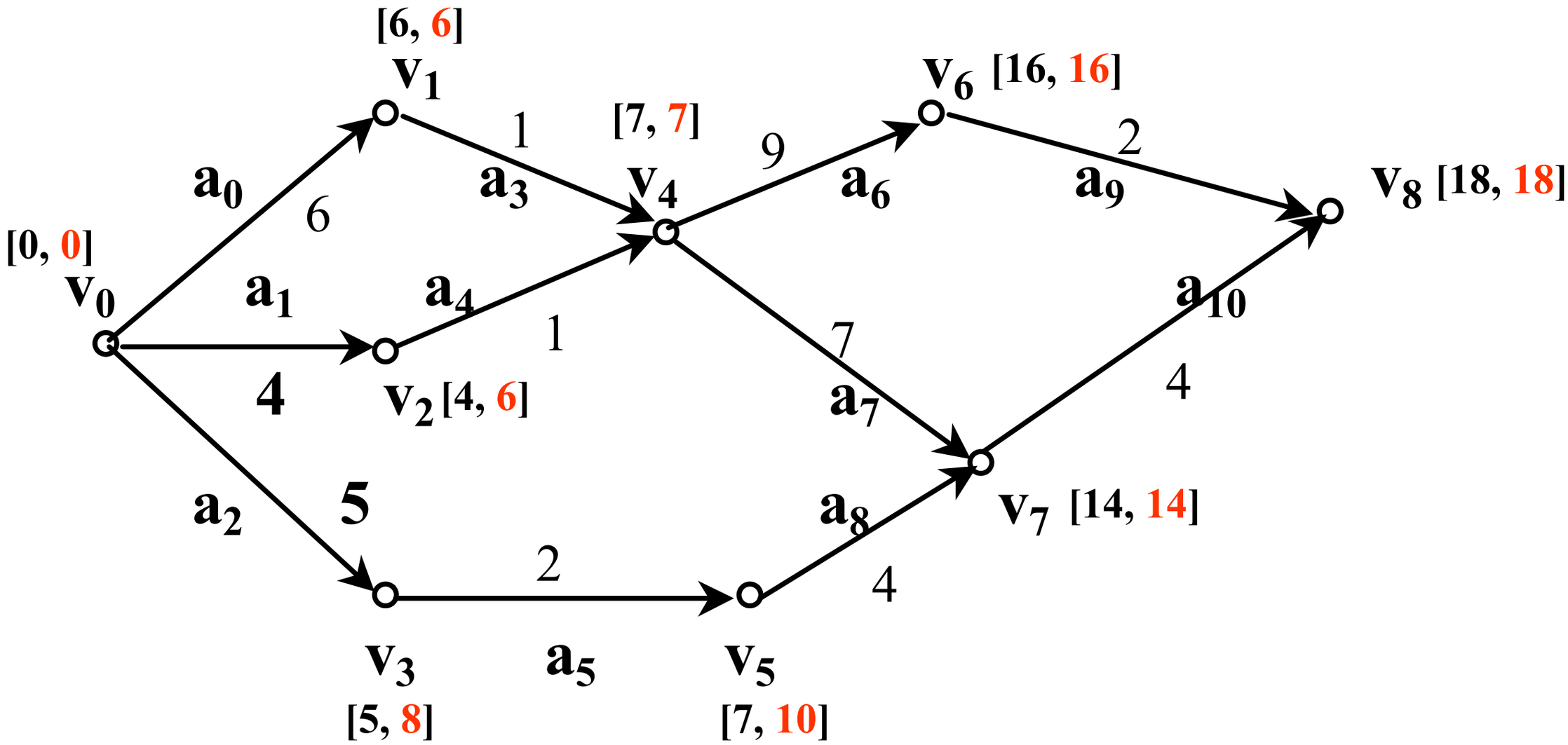
计算状态可能的最早发生时间



状态v_i允许的最迟发生时间$le(i)$

- 是指在不推迟整个工期的前提下，状态v_i允许的最晚发生时间。为了不拖延整个工期，v_i发生的最晚时间不得迟于其后继状态v_j的最晚发生时间减去活动$<v_i,v_j>$的持续时间。显然，$le(n-1)=ee(n-1)$，$le(i)$的计算是从后向前逐个状态进行推算。
- $le(n-1)=ee(n-1)$
- $le(i)=\min\{ le(j)-\text{weight}(< v_i, v_j >) \}$
- $< v_i, v_j > \in S,\ 0 \le i \le n-2$
- S是所有以v_i为开始顶点的出边的集合。
- $\text{weight}(< v_i, v_j >)$为边$< v_i, v_j >$的权。

计算状态允许的最迟发生时间



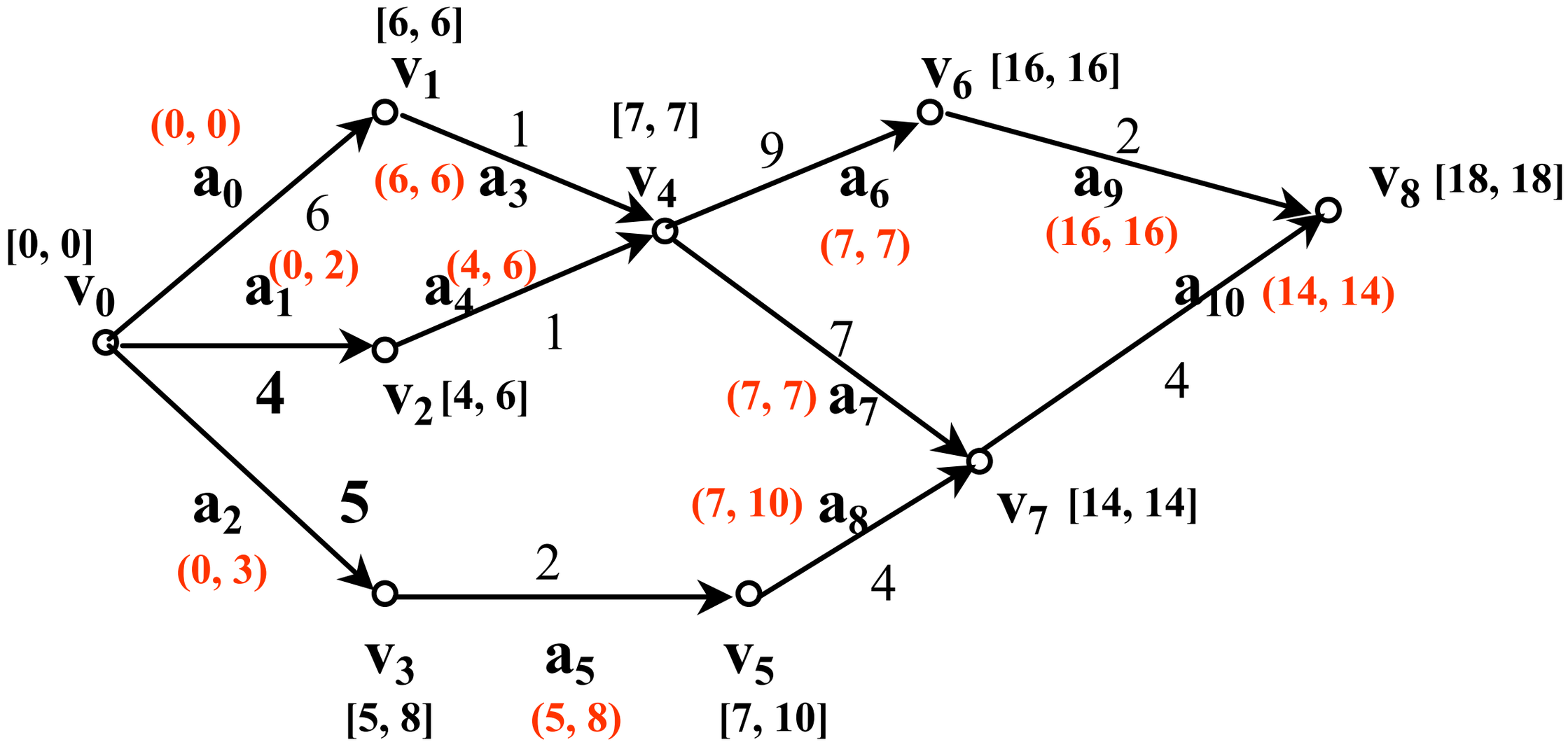
活动$a_k=<v_i, v_j>$的最早开始时间$e(k)$

- 只有状态v_i发生了，活动a_k才能开始。因此，活动a_k的最早开始时间等于状态v_i的最早发生时间。
- $e(k)=ee(i)$

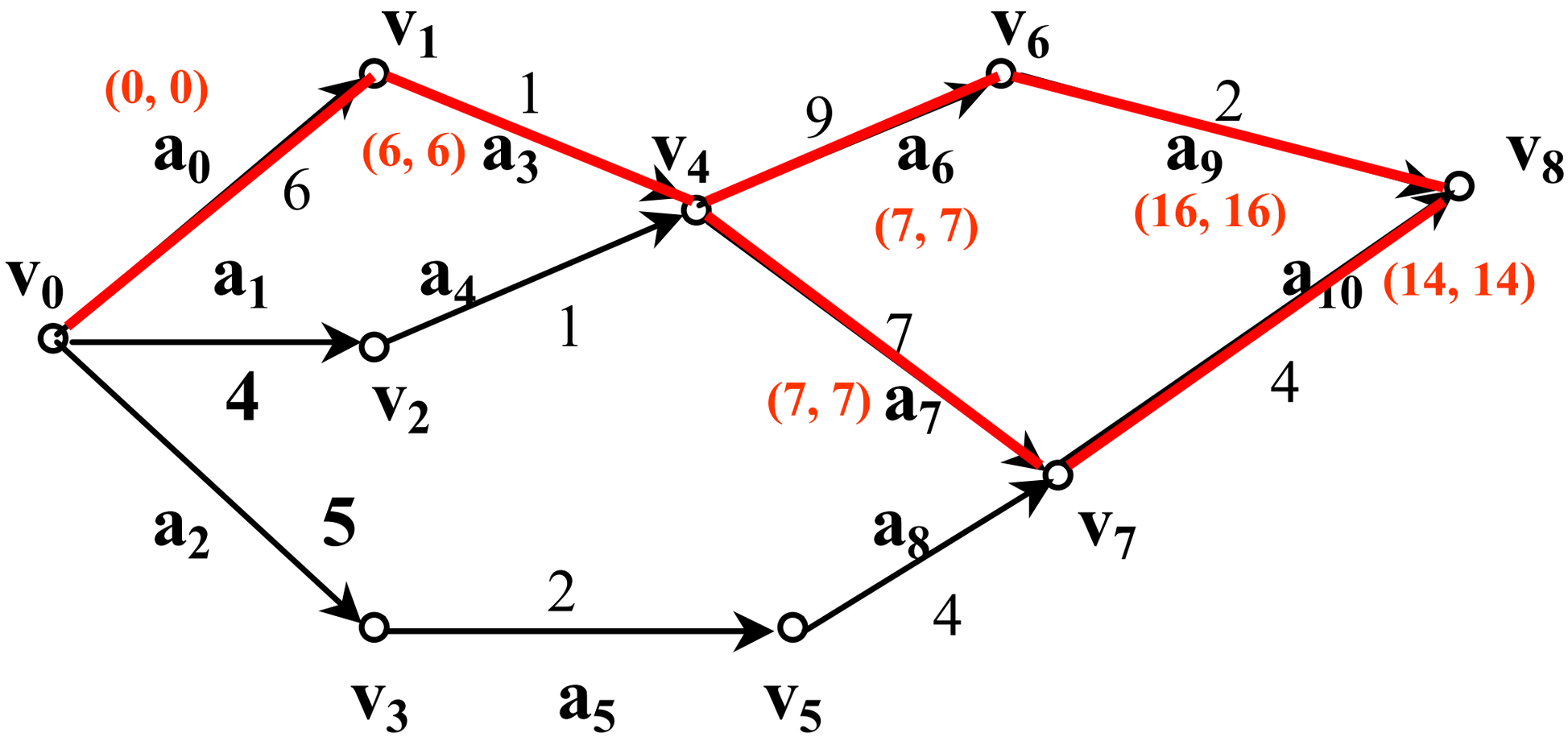
活动$a_k=<v_i, v_j>$的最晚开始时间$l(k)$

- 活动a_k的最晚开始时间$l(k)$等于a_k的最迟完成时间减去a_k的持续时间。
- $l(k)=le(j)-\text{weight}(<v_i, v_j>)$
- $\text{weight}(<v_i, v_j>)$为边$<v_i, v_j>$的权值。

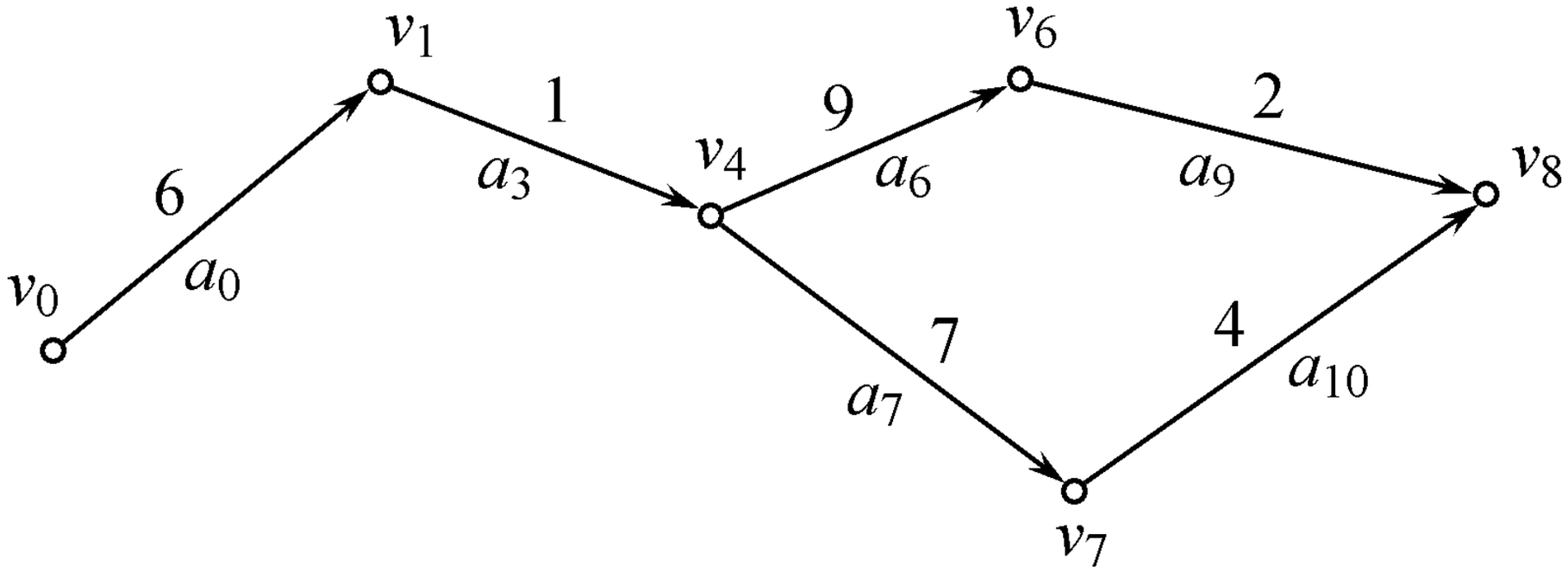
计算活动的最早开始时间和最晚开始时间



关键活动和关键路径


v1
(0, 0)
a0
6
(6, 6)
a3
1
v4
9
a6
(7, 7)
v6
2
a9
(16, 16)
v8
v0
a1
4
v2
a4
1
7
(7, 7)
a7
a10
(14, 14)
4
a2
5
v3
2
a5
v5
a8
4
v7

关键活动和关键路径



关键路径存在性

- 计算$ee(j)$必须在顶点v_j所有前驱顶点的最早发生时间都已经求出的前提下进行，而计算$le(i)$必须在顶点v_i所有后继顶点的最迟发生时间都已经求出的前提下进行，因此，顶点序列必须是一个拓扑序列。
- 下面的程序，首先检查图中是否有环，在没有环存在的情况下，按照上面的思路，逐步计算，并找出$e(k)=l(\mathrm{k})$的关键活动。
- 后面的算法中图用邻接表表示（同拓扑排序），边结点增加weight域记录权值。

数据结构

```
typedef struct EdgeNode {      /* 边表中的结点结构 */
    int endvex; AdjType weight;  /* 相邻顶点和权 */
    EdgeNode* next;
} *EdgeList;
typedef struct {               /* 顶点表中的结点 */
    EdgeList elist;            /* 边表头指针 */
} VexNode;
typedef struct{                /* 图顶点表 */
    int n;                     /* 图的顶点个数 */
    VexNode vexs[MAXVEX];
} GraphList;
typedef struct {               /* 记录拓扑排序序列 */
    int n;
    int vexsno[MAXVEX];  /* 顶点在顶点表中的下标值 */
} Topo;
```

关键路径算法

```
int CriticalPath(GraphList * paoe) {
    AdjType ee[VN], le[VN];
    int i, j = 0; EdgeList p; int ptopo[VN];
    if (topoSort(paoe, ptopo) == 0)      /* 求一个拓扑序列 */
      return 0;
    comp_ee(paoe, ptopo, ee);            /*计算数组ee*/
    comp_le(paoe, ptopo, ee, le);        /*计算数组le*/
    for(i = 0; i < VN; ++i){             /*求关键活动*/
      p = paoe -> vexs[i].edgelist;
      while (p){
        j = p -> endvex;
        if (ee[i] == le[j] – p -> weight)
          printf ("<v%2d, v%2d>\n", i, j);
        p = p -> nextedge;
      }
    }
return 1;
}
```

关键路径算法

把所有事件的最早发生时间初始化为0，而后逐个顶点去更新其后继顶点（事件）的最早发生时间

```
void comp_ee(GraphList* paoe, Topo* ptopo, AdjType ee[]) {
 int i, j, k;  EdgeList p;
 for (i = 0; i < VN; ++ i) ee[i] = 0;
 for (k = 0; k < VN; ++ k) {
   /* 按拓扑序，根据 v_i 的 ee 值调整其相邻顶点的 ee 值 */
   i = ptopo [k];
   for ( p = paoe->vexs[i].edgelist; p != NULL; p = p->nextedge) {
      j = p->endvex;
      if (ee[i] + p->weight > ee[j]) ee[j] = ee[i] + p->weight;
   }
  }
}
```

关键路径算法

把所有事件的最迟发生事件初始化为最后顶点的最迟时间，而后根据后继事件的最迟时间更新各事件的最迟时间。

```
void comp_le(GraphList * paoe, Topo* ptopo, AdjType ee[], AdjType le[]){
    int i, j, k;  EdgeList p;
    for (i = 0; i < VN; ++ i) le[i] = ee[VN - 1];
    for (k = VN - 2; k >= 0; -- k) {
        /* 按拓扑序的逆序，根据相邻顶点的le值调整 v_i 的le值 */
        i = ptopo[k];
        for (p = paoe->vexs[i].edgelist; p; p = p->nextedge) {
            j = p->endvex;
            if( le[j] - p->weight < le[i] ) le[i] = le[j] - p->weight;
        }
    }
}
```

算法分析

- 设AOE网有n个顶点，e条边，在求拓扑序列和状态可能的最早发生时间及允许的最迟发生时间，以及判断关键活动的最早开始时间和最晚开始时间时，都要对图中所有顶点及每个顶点边表中所有的边结点进行检查，时间花费为$O(n+e)$，因此，求关键路径算法的时间复杂度为$O(n+e)$。

本讲重点

- **AOV网和AOE网的概念**
- **拓扑排序算法**
- **关键路径算法**

图的小结

- 图的概念
  - 有向图
  - 无向图
  - 带权图
- 图的周游
  - 深度优先
  - 广度优先
- 图的存储
  - 邻接矩阵表示
  - 邻接表表示
- 最小生成树算法
  - Prim算法
  - Kruskal算法
- 最短路径算法
  - Dijkstra算法
  - Floyd算法
- AOV和AOE网
  - 拓扑排序
  - 关键路径