

有向无环图及其应用

有向无环图

有向无环图的应用

公用表达式

AOV网-拓扑排序

AOE网-关键路径

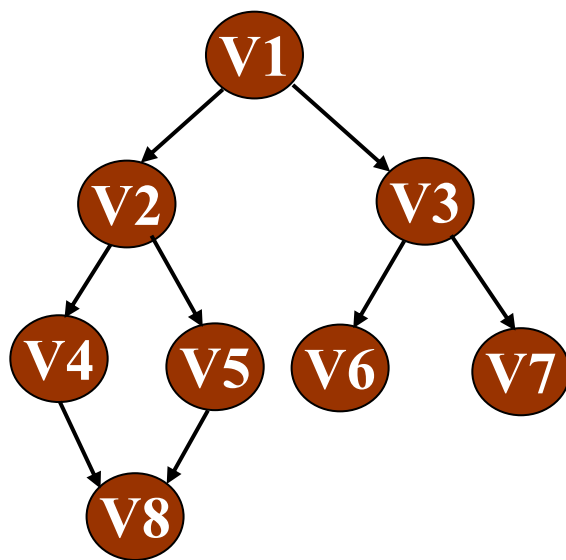
小结和作业

有向无环图

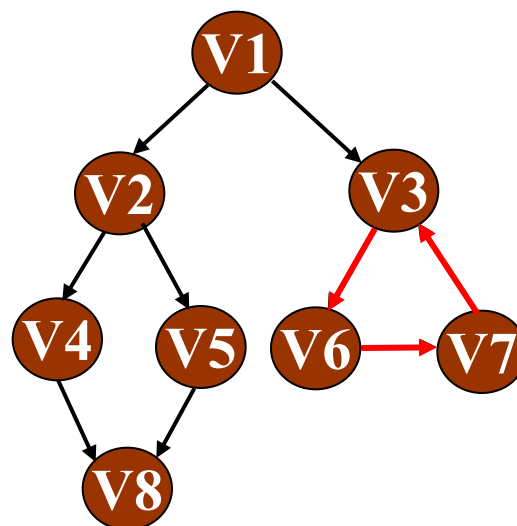
一、定义：

一个无环的有向图，称为有向无环图（DAG图）

DAG = Directed Acyclic Graph



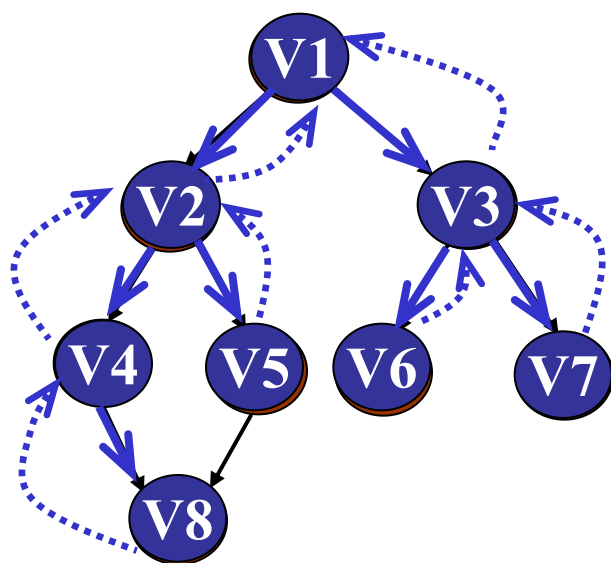
DAG图



有环的有向图

有向无环图

二、如何判断一个图是否是DAG?

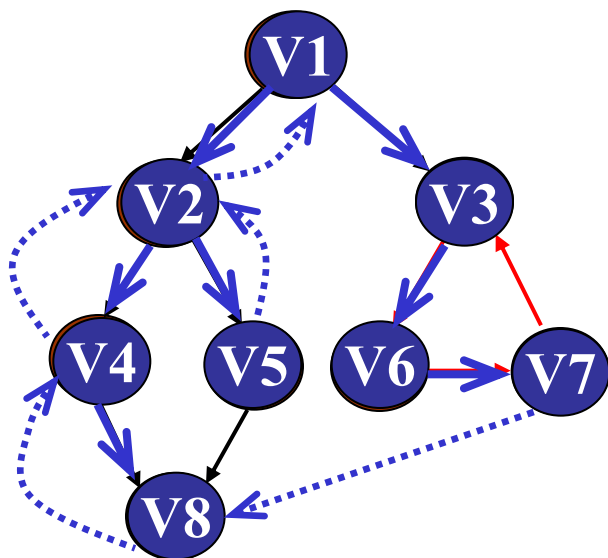


DAG图

深度优先搜索没有出现指向祖先的回边，即：

没有一个顶点有一条边，指向遍历过程中先访问过的顶点（并且这些顶点的DFS函数没有执行完）

有向无环图



V7有一条边指向了V3，
所以有环

V7指向V8的边，没有构成环，因为没有V8到V7的路径

非**DAG**图

有向无环图

```
bool DAG(Graph G) {  
    for (v=0; v<G.vexnum; ++v)  
        visited[v] = FALSE; // 访问标志数组初始化  
    InitStack(S); // 存放顶点  
    for (v=0; v<G.vexnum; ++v) {  
        if (!visited[v])  
            if (!DFS-DAG(G, v)) return(FALSE);  
            // 对尚未访问的顶点调用DFS-DAG  
    }  
    return TRUE;  
}
```

有向无环图

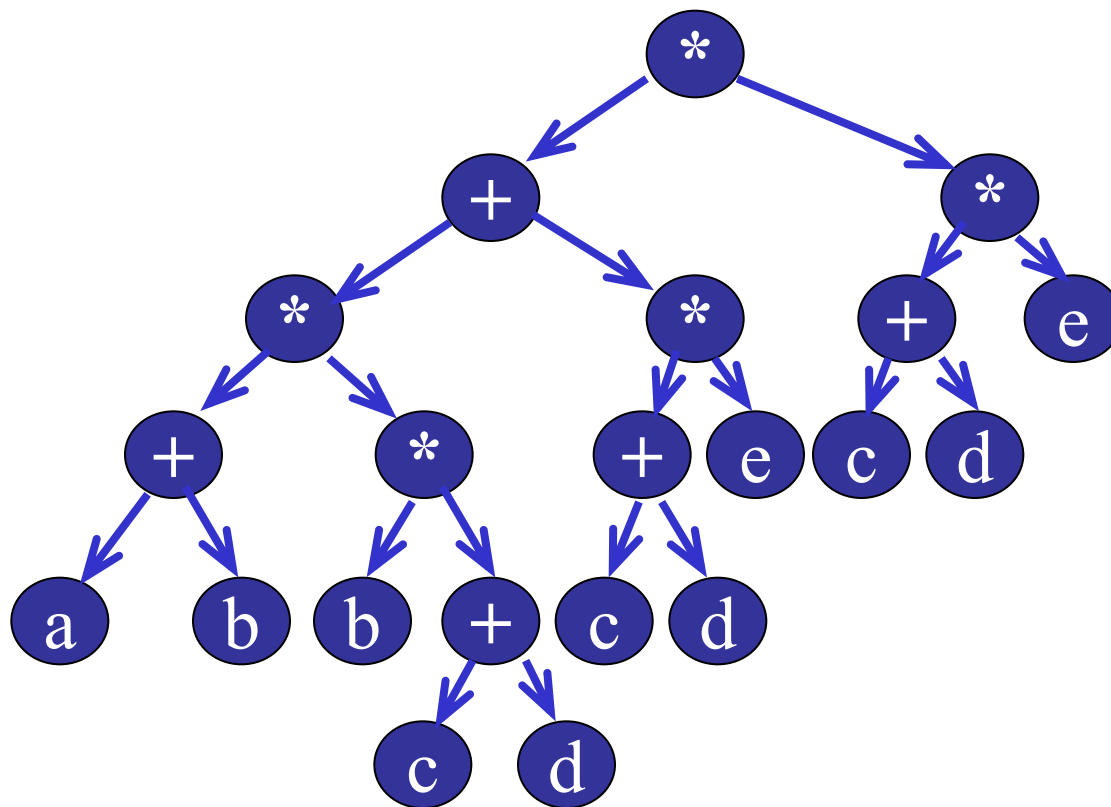
```
Bool DFS-DAG(Graph G, int v) {  
    // 从顶点v出发，深度优先搜索遍历连通图 G  
    visited[v] = TRUE; Push(S,v);  
    for(w=FirstAdjVex(G, v);w>=0; w=NextAdjVex(G,v,w)){  
        {if (w in S) then return(FALSE);//有回边  
        if(!visited[w]){  
            if(!DFS-DAG(G, w)) return(FALSE);  
        }  
        Pop(S, v); //v的所有邻接点已经遍历完  
    }  
    return(TRUE);  
} // DFS-T
```



公用表达式

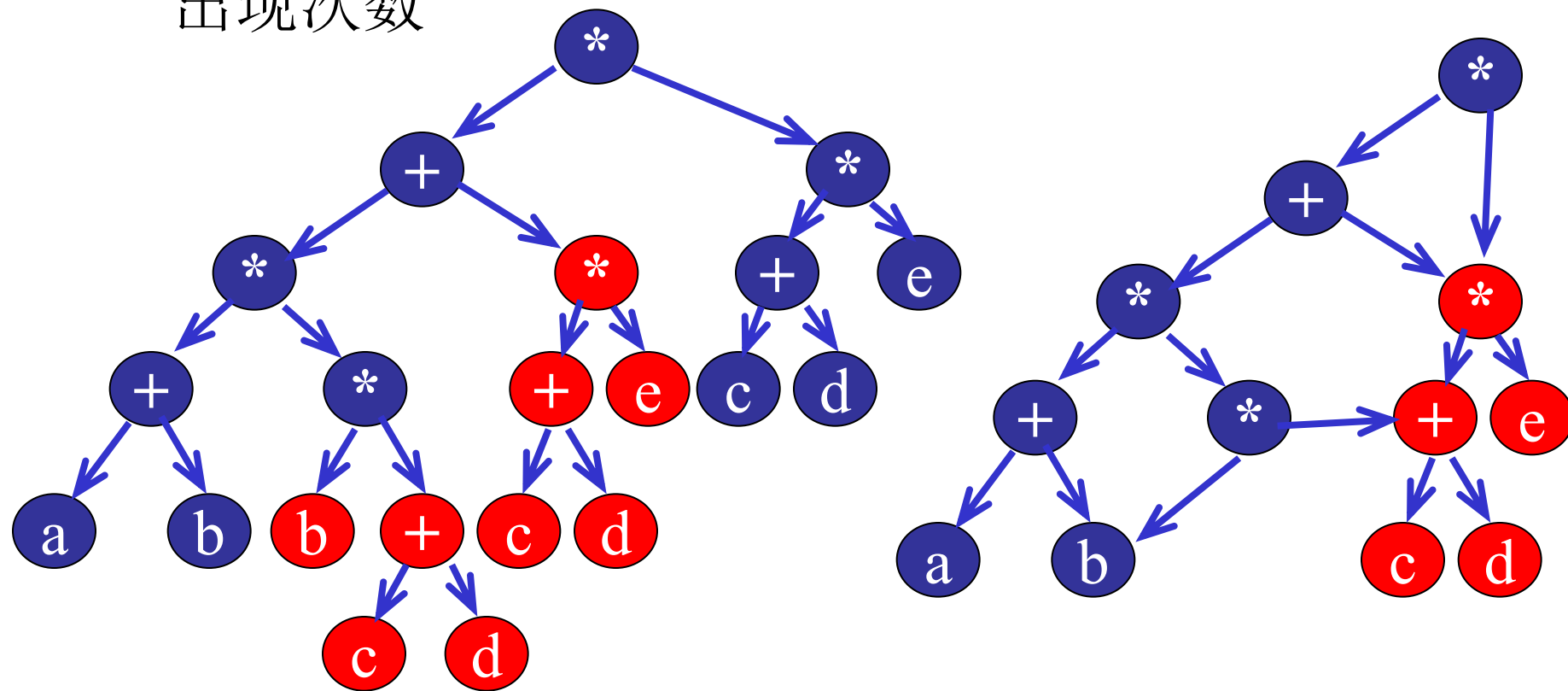
用树表示表达式:

$((a+b)*(b*(c+d))+(c+d)*e)*((c+d)*e)$



公用表达式

多次出现的变量和表达式通过共用，减少出现次数



拓扑排序

一、定义

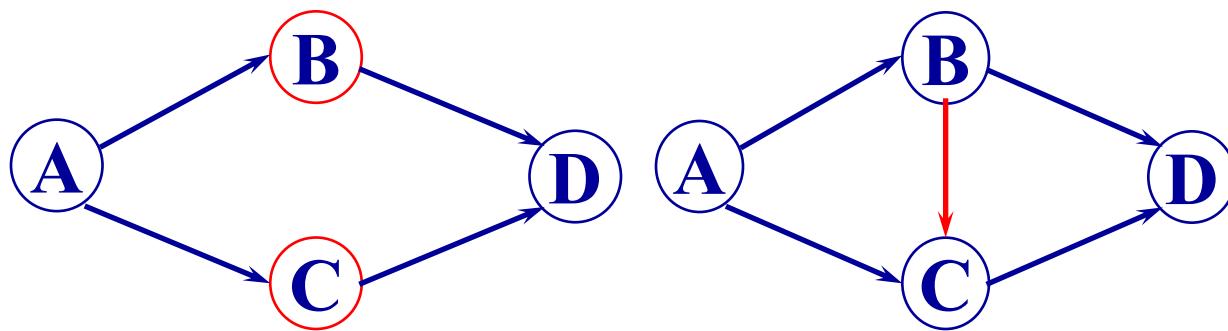
由集合上的一个偏序关系得到集合的全序关系的操作

偏序：自反的、反对称的、传递的

全序：R是集合X上的偏序，对于集合X中的任何元素x, y，如果都有 xRy 或者 yRx ，则称R是全序关系

拓扑排序

- 偏序就是集合中的部分成员可以比较。
- 全序是集合中的任何成员之间都可以比较。



偏序

全序

拓扑排序

按照有向图给出的次序关系，将图中顶点排成一个线性序列，对于有向图中没有限定次序关系的顶点，则可以人为加上任意的次序关系。

由此所得顶点的线性序列称之为拓扑有序序列

拓扑排序

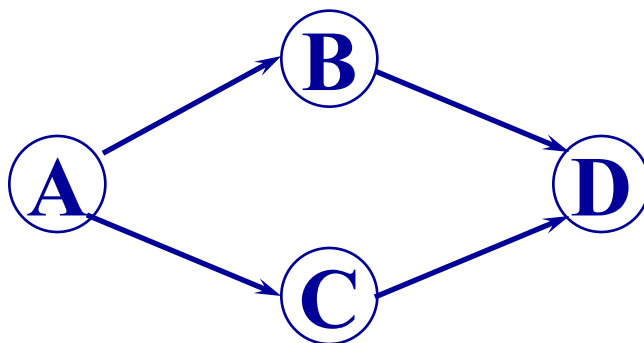
拓扑排序的用途：判断AOV网是否存在环路

AOV网 (Activity On Vertex NetWork)

用顶点表示活动，弧表示活动间的优先关系的有向图。

AOV网中不应该出现有向环：如果存在环，则某项活动以自己为先决条件，

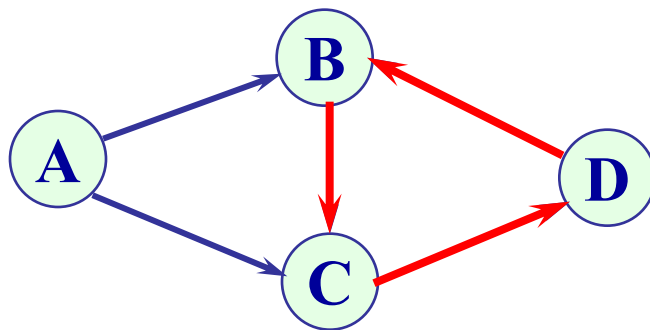
拓扑排序



可求得拓扑有序序列:

A B C D 或 A C B D

拓扑排序



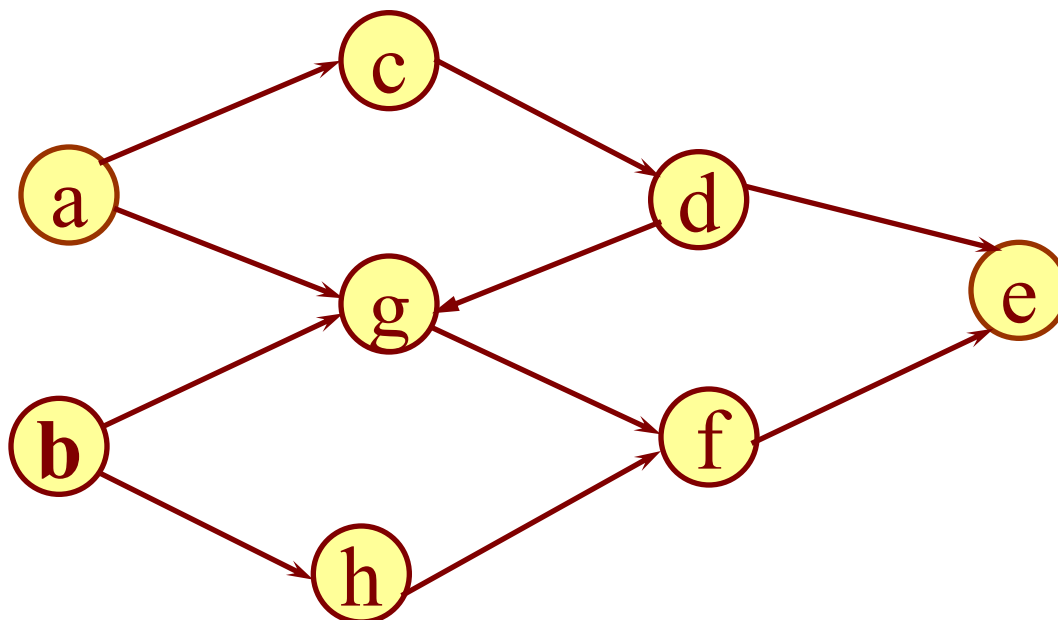
不能求得它的拓扑有序序列。

因为图中存在一个回路 $\{B, C, D\}$

拓扑排序---方法1

- 1、从有向图中选取一个没有前驱的顶点，并输出之；
- 2、从有向图中删去此顶点以及所有以它为尾的弧；
- 3、重复上述两步，直至图空，或者图不空但找不到无前驱的顶点为止。

拓扑排序---方法1



拓扑序列: **b h a c d g f e**

拓扑排序---方法1

在算法中需要用定量的描述替代定性的概念

没有前驱的顶点 \longleftrightarrow 入度为零的顶点

删除顶点及以它为尾的弧 \longleftrightarrow 弧头顶点的入度减1

拓扑排序---算法

```
Status TopologicalSort(ALGraph G){  
    FindInDegree(G, indegree);  
    InitStack(S);  
    for(i=0;i<G.vexnum;i++){if(!indegree[i]) push(S,i);}  
    count=0;           //对输出顶点计数  
    while (!EmptyStack(S)) {  
        .....  
    }//while  
    if (count<G.vexnum) return ERROR;  
    else return OK;  
}
```

拓扑排序---算法

```
Status TopologicalSort(ALGraph G){
```

```
.....
```

```
while (!EmptyStack(S)) {
```

```
    Pop(S, v); ++count; printf(v);
```

```
    for (w=FirstAdj(v); w; w=NextAdj(G,v,w)){
```

```
        --indegree(w); // 弧头顶点的入度减1
```

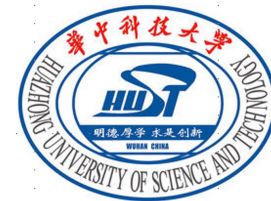
```
        if (!indegree[w]) Push(S, w);
```

```
    }//for
```

```
}//while
```

```
.....
```

```
}
```



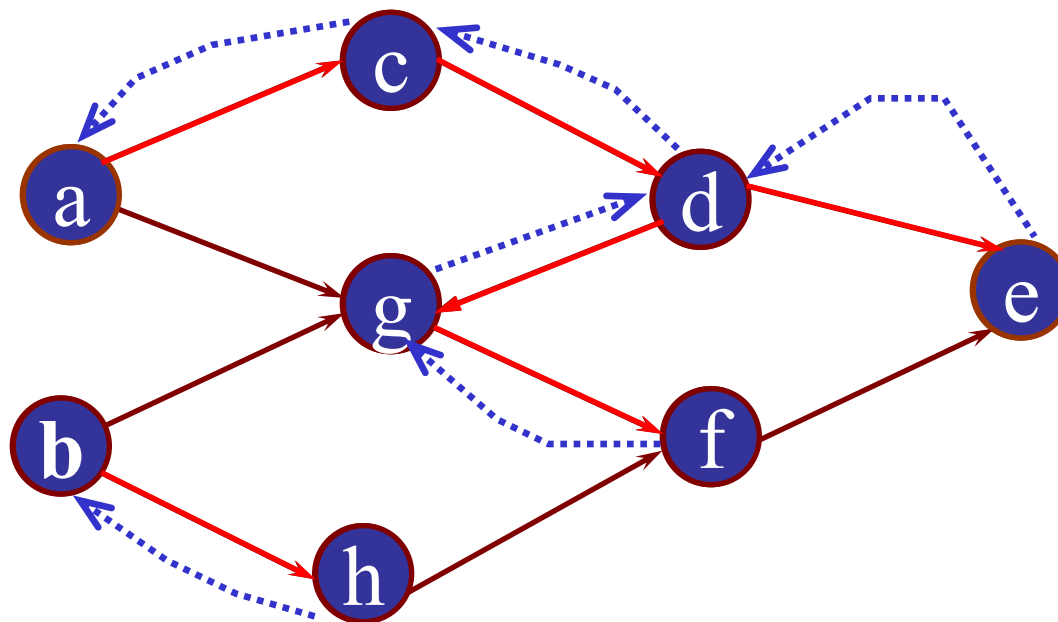
拓扑排序---算法

算法分析:

总的时间复杂度: $O(n+e)$

拓扑排序---方法2

对有向无环图利用深度优先搜索进行拓扑排序。



退出DFS函数顺序: e f g d c a h b

此图的一个拓扑序列为: b h a c d g f e

拓扑排序---方法2

结论：

最先退出DFS函数的顶点是出度为零的顶点，为拓扑排序序列中最后一个顶点。

因此，按退出DFS函数的先后记录下来的顶点序列即为逆向的拓扑排序序列。

拓扑排序---方法2

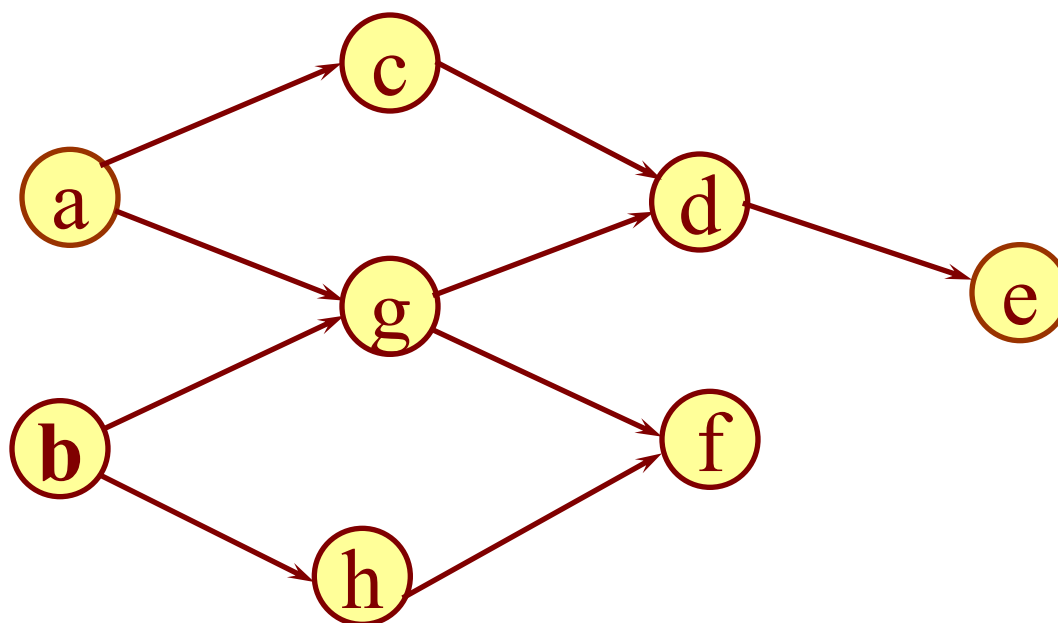
```
void DFS-TopologicalSort (Graph G, int v) {//如何确定v
    for (v=0; v<G.vexnum; ++v)
        visited[v] = FALSE; // 访问标志数组初始化
    InitStack(S);//存放顶点，按照出DFS的次序
    for (v=0; v<G.vexnum; ++v) {
        if (!visited[v]) DFS-T(G, v);
        // 对尚未访问的顶点调用DFS
    }
    while(!Empty(S)) {//输出拓扑排序的结果
        Pop(S, v); printf(“%d”, v)}
}
```

拓扑排序---方法2

```
void DFS-T(Graph G, int v) {  
    // 从顶点v出发，深度优先搜索遍历连通图 G  
    visited[v] = TRUE;  
    for(w=FirstAdjVex(G, v);  
        w>=0; w=NextAdjVex(G,v,w))  
        {if (!visited[w]) DFS-T(G, w);}  
    // 对v的尚未访问的邻接顶点w递归调用DFS-T  
    Push(S, v); // 顶点v的DFS函数执行完毕  
} // DFS-T
```

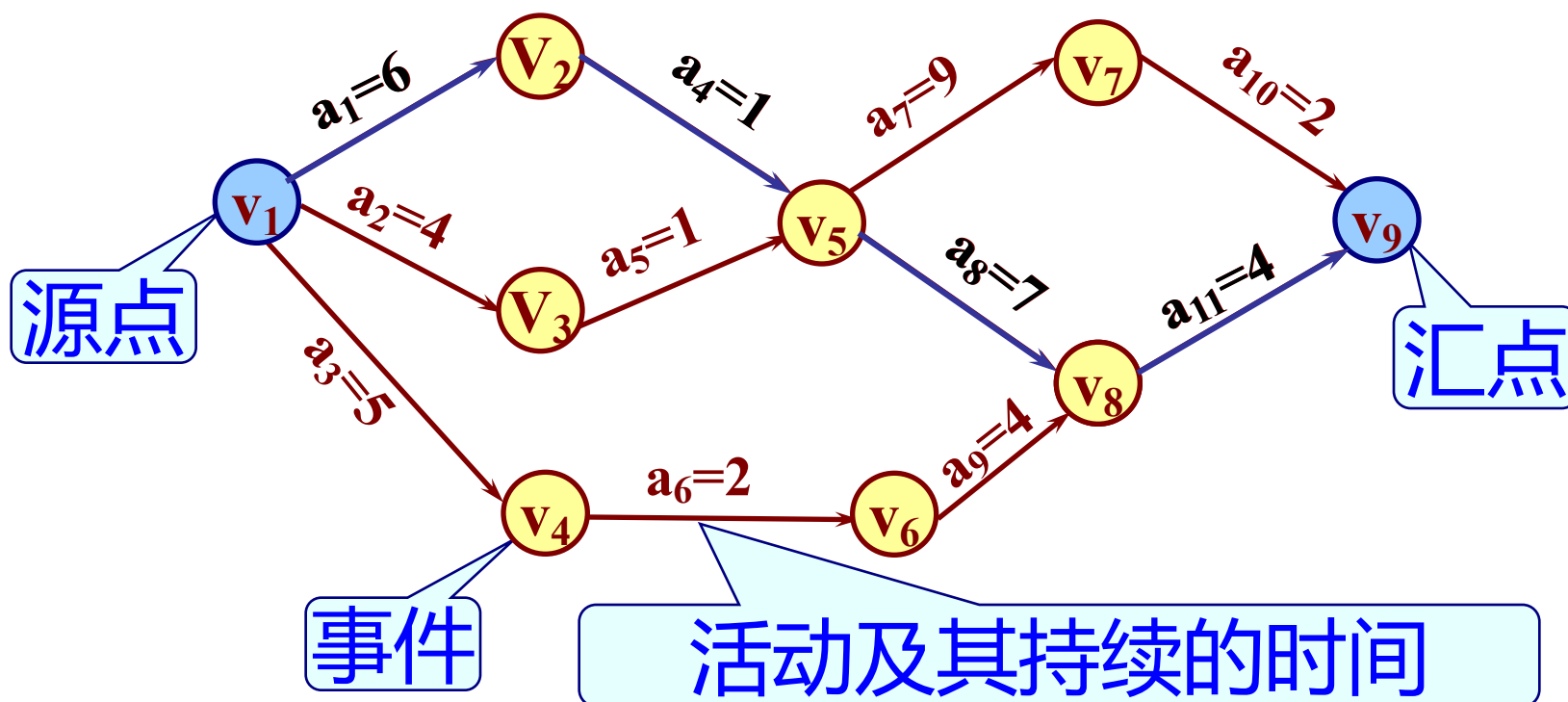

拓扑排序---练习

写出下图的所有的拓扑序列



关键路径

AOE-网(Activity on Edge):边表示活动网。



关键路径

AOE-网是一个带权的有向无环图，可用来估算工程的完成时间。

假设以AOE网表示一个施工流图，弧上的权值表示完成该项子工程所需的时间。

- 1、完成整个工程至少需要多少时间？
- 2、哪些活动是影响工程进度的关键？

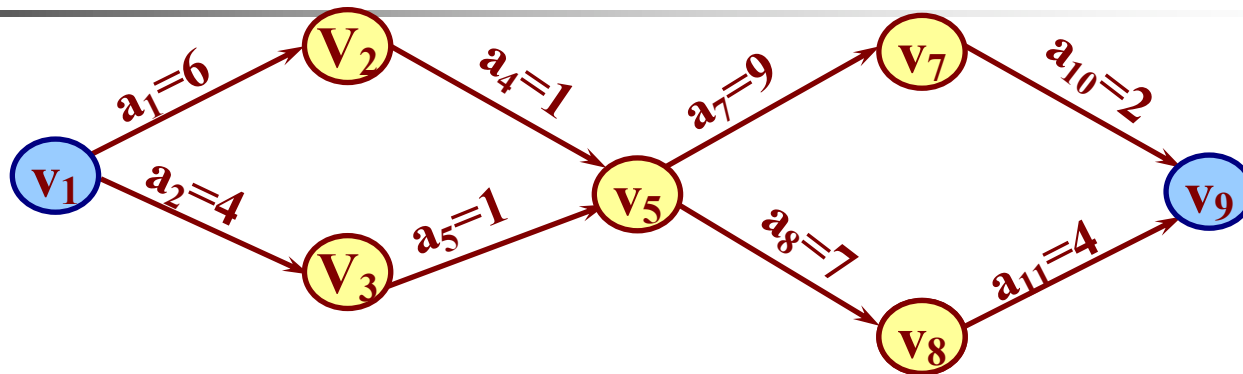
关键路径

- 路径最长的路径叫做关键路径
- 影响工程进度的活动叫关键活动
- 关键路径上的活动一定是关键活动

如何求关键路径

- 用 $e(i)$ 和 $l(i)$ 分别表示活动 a_i 的最早开始时间和最迟开始时间
- $e(i)-l(i)$ 为活动 a_i 的时间余量
- $e(i)=l(i)$ 的活动是关键活动

如何求关键路径



ve(i): 表示事件i的最早开始时间

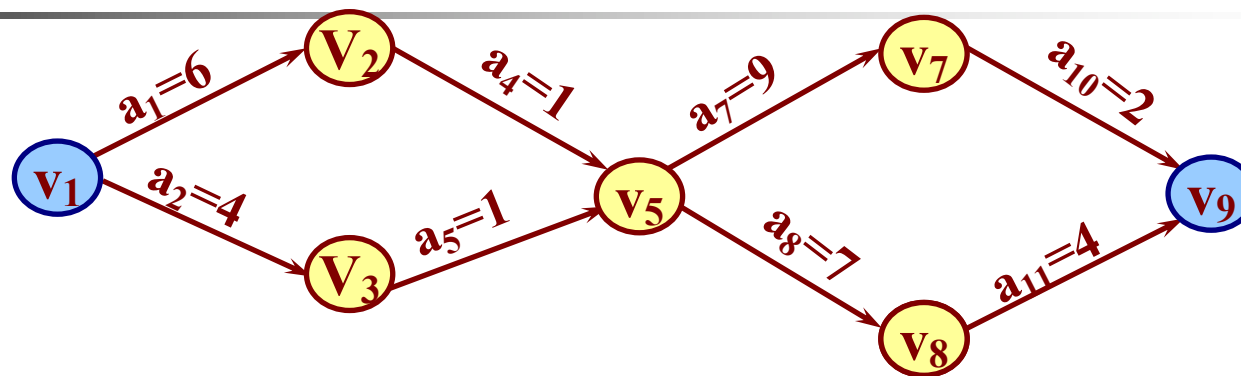
vl(i): 表示事件i的最迟开始时间

已知 $ve(1)=0$ ，计算其余顶点的**ve**值要按照
顶点拓扑排序后的次序进行

$ve(j)=\max(ve(i) + dut(<i,j>))$
 $<i,j>\in T, T$ 是以j为头的弧的集合

V_1 到 V_j 的最长路径

如何求关键路径



$vl(n-1)=ve(n-1)$, 然后按照顶点逆拓扑排序后的次序求其余顶点的 vl

$vl(i)=\min(vl(j) - dut(<i,j>))$

$<i,j>\in S, S$ 是以 i 为尾的弧的集合

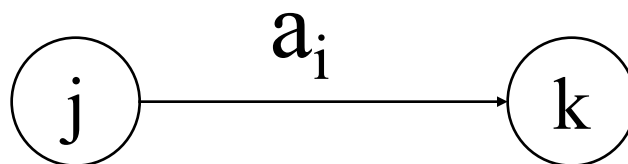
如何求关键路径

用 $e(i)$ 和 $l(i)$ 分别表示活动 a_i 的最早开始间和最迟开始时间

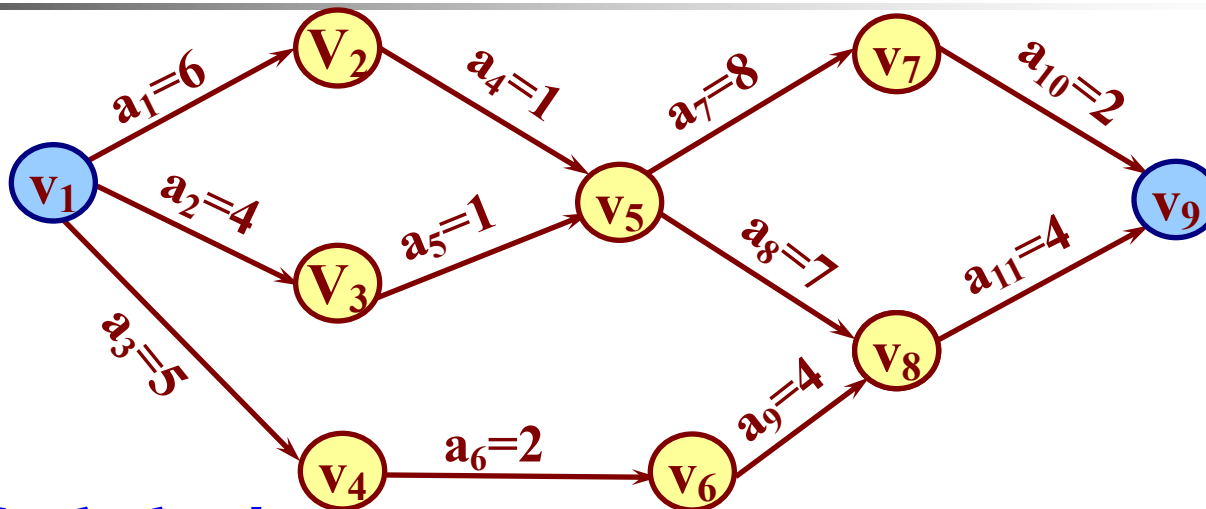
显然有：

$$e(i) = ve(j)$$

$$l(i) = vl(k) - dut(j, k)$$



如何求关键路径

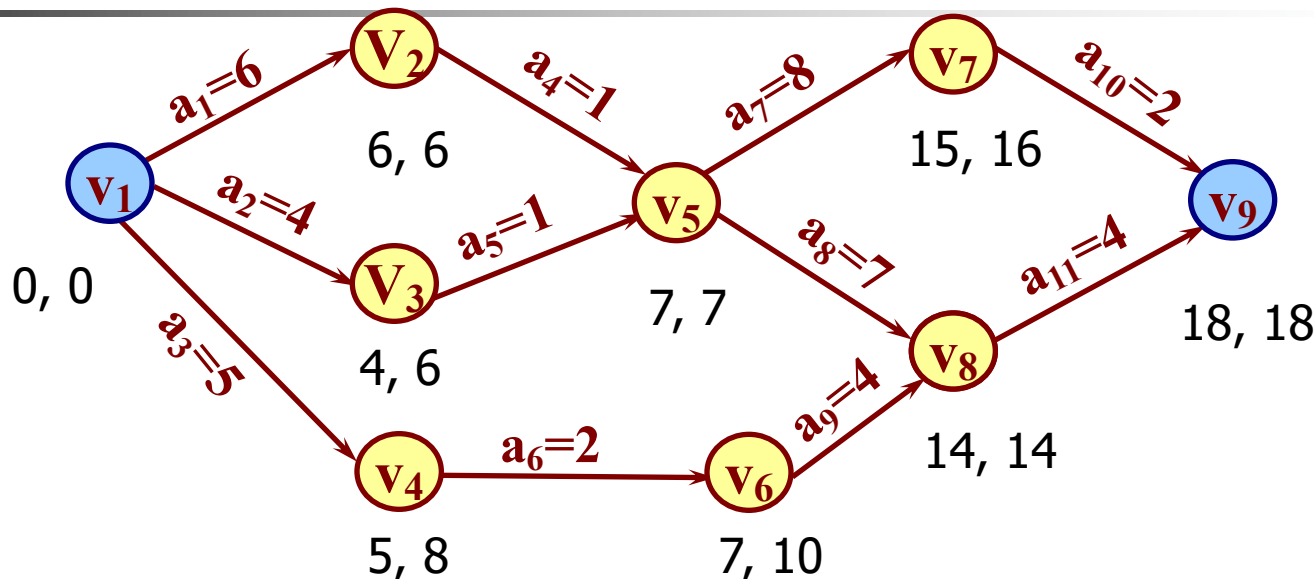


拓扑有序序列: **v1 v2 v3 v4 v6 v5 v8 v7 v9**

逆拓扑有序序列: **v9 v7 v8 v5 v2 v3 v6 v4 v1**

	v ₁	v ₂	v ₃	v ₄	v ₅	v ₆	v ₇	v ₈	v ₉
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

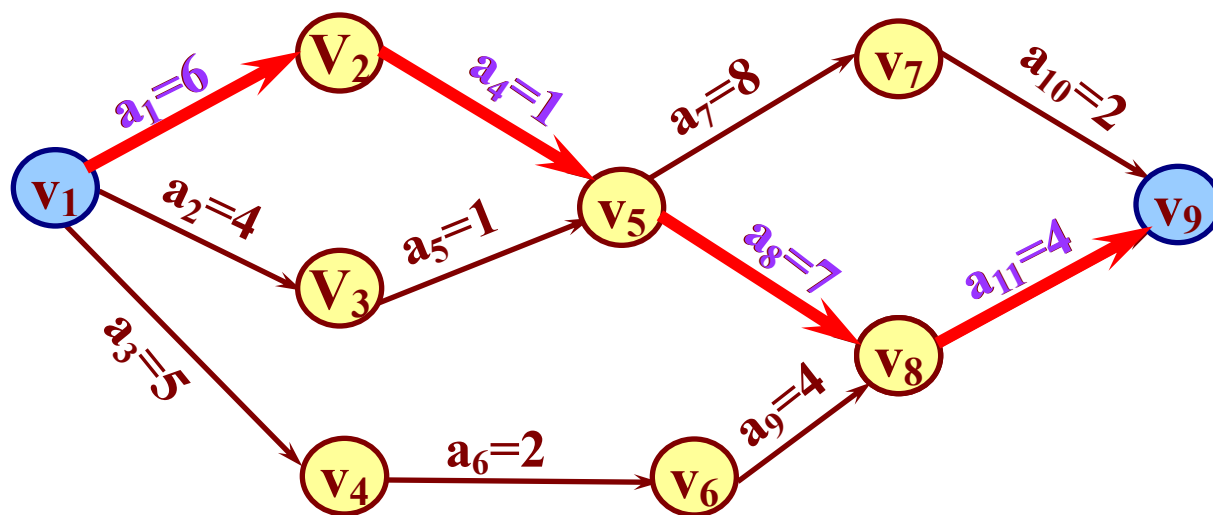
如何求关键路径



	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
	6	4	5	1	1	2	8	7	4	2	4
e	0	0	0	6	4	5	7	7	7	15	14
l	0	2	3	6	6	8	8	7	10	16	14

如何求关键路径

最终求得的关键路径如下所示：



关键路径算法

- 1) 输入 n 个顶点和 e 条弧 $\langle j, k \rangle$, 建立AOE网存储结构
- 2) 求出网 G 的拓扑排序

若得到拓扑排序中顶点个数 $< n$, 说明 G 中存在环, 算法终止。

令 $ve[0]=0$,

求其余结点的最早发生时间 $ve(j)=\max(ve(i) + dut(\langle i, j \rangle))$

$\langle i, j \rangle \in T$, T 是以 j 为头的弧的集合

关键路径算法

3) 求出G的逆向拓扑序列

从汇点出发，令 $vl(n-1)=ve(n-1)$

求其余各顶点的最迟发生时间：

$$vl(i)=\min(vl(j) - dut(<i,j>))$$

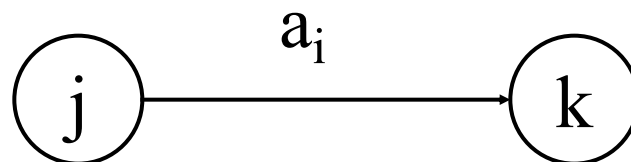
$<i,j>\in S$, S 是以 i 为尾的弧的集合

关键路径算法

4) 根据2) 和3) 中所得的 ve 和 vl , 求每条弧上的活动 a_i 的最早发生时间 $e(s)$ 和 $l(s)$

$$e(i) = ve(j)$$

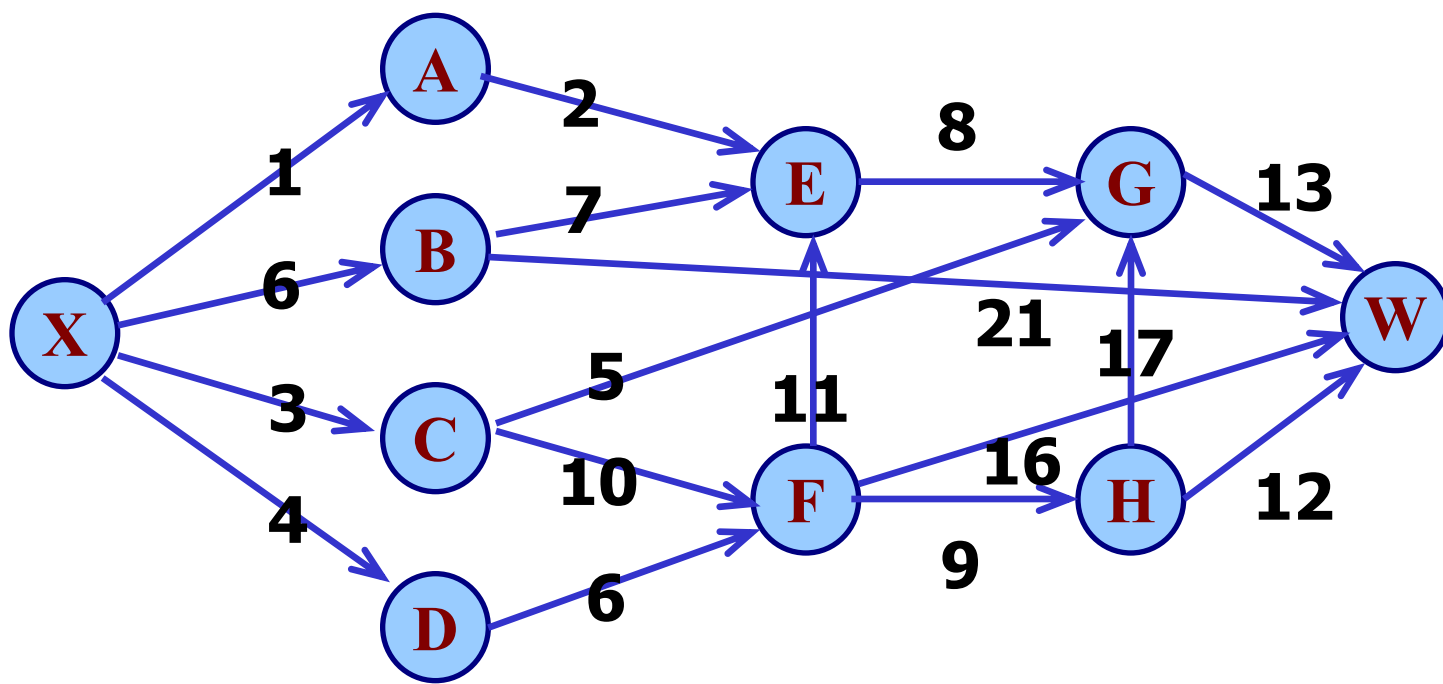
$$l(i) = vl(k) - dut(j, k)$$



5) 如果某条弧上的活动 a_i 满足 $e(i) = l(i)$, 则 a_i 为关键活动。

如何求关键路径

练习：求下图各活动弧 a_i 的 $e(a_i)$ 和 $l(a_i)$ ，个事件 v_j 的 $ve(v_j)$ 和 $vl(v_j)$ ，列出各关键路径。



关键路径算法

算法描述:

- 1) 有向网AOE采用邻接表作为存储结构**
- 2) 栈T: 拓扑序列顶点栈**
- 3) 栈S: 0入度顶点栈**
- 4) 返回值:**
ERROR: 图G中有回路
OK: 栈T返回图G的一个拓扑有序序列

关键路径算法

```
Status TopologicalOrder( ALGraph G, Stack &T){  
    FindInDegree(G, indegree);  
    InitStack(S);  
    for(i=0;i<G.vexnum;i++){if(!indegree[i]) push(S,i);}  
    Initstack(T); count=0; ve[0..G.vexnum-1]=0;  
    while (!EmptyStack(S)) {  
        .....  
    }//while  
    if (count<G.vexnum) return ERROR;  
    else return OK;  
}
```

关键路径算法

■ Status TopologicalOrder(ALGraph G, Stack &T){

.....

```
while (!EmptyStack(S)) {  
    Pop(S, v); Push(T,j);++count; //j号顶点入栈T  
    for (p=G.vertices[j].firstarc; p; p=p->nextarc){  
        k=p->adjvex;  
        if(--indegree(k)==) Push(S,k); //入度-1为0 , 则入栈  
        if((ve[j]+*(p->info))>ve[k])  
            ve[k]= ve[j]+*(p->info)  
        }//for  
    }//while  
    if (count<G.vexnum) return ERROR;  
    else return OK;  
}
```

关键路径算法

```
Status CriticalPath( ALGraph G){//输出G的关键活动
    if(! TopologicalOrder(G,T) return ERROR;
    vl[0.. G.vexnum-1] =ve[0..G.vexnum-1];//用ve初始化vl
    while(!stackEmpty(T)){
        pop(T,j);
        for(p=G.vertices[j].firstarc;p;p=p->nextarc){
            k=p->adjvex; dut=*(p->info);
            if(vl[k]-dut<vl[j])
                vl[j]=vl[k]-dut;
        }//end of for
    }//end of while
    .....}//end of CriticalPath
```

关键路径算法

Status CriticalPath(ALGragh G){**//输出G的关键活动**

.....

for(j=0;j<G.vexnum;++j)

for (p=G.vertices[j].firstarc; p; p=p->nextarc){

k=p->adjvex; dut=*(p->info);

ee=ve[j];el=vl[k]-dut;

tag=(ee=el)?'*':";

printf(j,k,dut,ee,el,tag);

}//end of for(p)

}//end of status

关键路径算法

Status CriticalPath(ALGragh G){//输出G的关键活动

.....

```
for(j=0;j<G.vexnum;++j)
```

```
for (p=G.vertices[j].firstarc; p; p=p->nextarc){
```

```
    k=p->adjvex; dut=*(p->info);
```

```
    ee=ve[j];el=vl[k]-dut;
```

```
    tag=(ee==el)?'*':";
```

```
    printf(j,k,dut,ee,el,tag);
```

```
    }//end of for(p)
```

```
}//end of status
```

关键路径算法分析

1.求关键路径的总的时间复杂度： $O(n+e)$

2.AOE-网求出的路径可能大于一条。

这种情况下只有同时提高所有关键路径上的活动的速度，才能使整个工期缩短。



小结

本节主要内容：

1. 拓扑排序

1. 什么是拓扑排序

2. 如何进行拓扑排序

3. 拓扑排序算法

2. 关键活动

1. 什么是关键活动

2. 如何求关键活动

3. 关键活动算法

P0J2367 【基础】

题目大意：

火星人的亲缘关系系统十分混乱。每个火星人可以有一个或以上的父母，也可以有很多个子女。现在要安排 N ($1 \leq N \leq 100$) 个依次编号为 1 到 N 的火星人的发言顺序，要求辈份高的先发言。请给出一个发言顺序。

输入：

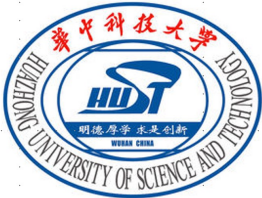
第一行是一个整数 N 。接下来有 N 行，第 i 行有若干个整数，表示第 i 位火星人的子女的编号，当编号为 0 的时候结束。一个火星人的子女可能没有子女。

输出：

输出一行，包含 N 个整数，为火星人的发言顺序。

题目描述

展开



小明要去一个国家旅游。这个国家有 N 个城市，编号为 1 至 N ，并且有 M 条道路连接着，小明准备从其中一个城市出发，并只往东走到城市 i 停止。

所以他就需要选择最先到达的城市，并制定一条路线以城市 i 为终点，使得线路上除了第一个城市，每个城市都在路线前一个城市东面，并且满足这个前提下还希望游览的城市尽量多。

现在，你只知道每一条道路所连接的两个城市的相对位置关系，但并不知道所有城市具体的位置。现在对于所有的 i ，都需要你为小明制定一条路线，并求出以城市 i 为终点最多能够游览多少个城市。

输入格式

第 1 行为两个正整数 N, M 。

接下来 M 行，每行两个正整数 x, y ，表示了有一条连接城市 x 与城市 y 的道路，保证了城市 x 在城市 y 西面。

输出格式

N 行，第 i 行包含一个正整数，表示以第 i 个城市为终点最多能游览多少个城市。

输入输出样例

输入 #1

复制

```
5 6
1 2
1 3
2 3
2 4
3 4
2 5
```

输出 #1

复制

```
1
2
3
4
3
```

说明/提示

均选择从城市 1 出发可以得到以上答案。

对于20%的数据， $N \leq 100$ ；

对于60%的数据， $N \leq 1000$ ；

对于100%的数据， $N \leq 100000, M \leq 200000$ 。



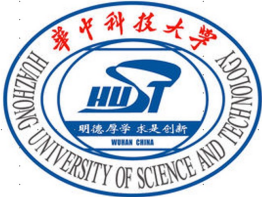
```
#include<bits/stdc++.h>
using namespace std;

const int maxn=100000+15;
int n,m,sum,tot;
int head[maxn],ru[maxn],ts[maxn],dp[maxn];
struct EDGE
{
    int to;int next;
}edge[maxn<<2];
void add(int x,int y)
{
    edge[++sum].next=head[x];
    edge[sum].to=y;
    head[x]=sum;
}
void topsort()
{
    queue<int> q;
    for (int i=1;i<=n;i++)
        if (ru[i]==0) {
            q.push(i);
            ts[++tot]=i;
        }
    while (!q.empty())
    {
        int u=q.front();q.pop();
        for (int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            ru[v]--;
            if (ru[v]==0) {
                q.push(v);ts[++tot]=v;
            }
        }
    }
}
```

```
int main()
{
    scanf("%d%d",&n,&m);
    for (int i=1;i<=m;i++)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        add(u,v);
        ru[v]++;
    }
    topsort();
    for (int i=1;i<=n;i++) dp[i]=1;
    for (int i=1;i<=n;i++)
    {
        int u=ts[i];
        for (int j=head[u];j;j=edge[j].next)
        {
            int v=edge[j].to;
            dp[v]=max(dp[v],dp[u]+1);
        }
    }
    for (int i=1;i<=n;i++)
        printf("%d\n",dp[i]);
    return 0;
}
```

题目描述

展开



John 的农场在给奶牛挤奶前有很多杂务要完成，每一项杂务都需要一定的时间来完成它。比如：他们要将奶牛集合起来，将他们赶进牛棚，为奶牛清洗乳房以及一些其它工作。尽早将所有杂务完成是必要的，因为这样才有更多时间挤出更多的牛奶。当然，有些杂务必须在另一些杂务完成的情况下才能进行。比如：只有将奶牛赶进牛棚才能开始为它清洗乳房，还有在未给奶牛清洗乳房之前不能挤奶。我们把这些工作称为完成本项工作的准备工作。至少有一项杂务不要求有准备工作，这个可以最早着手完成的工作，标记为杂务1。John 有需要完成的 n 个杂务的清单，并且这份清单是有一定顺序的，杂务 $k(k > 1)$ 的准备工作只可能在杂务1至 $k - 1$ 中。

写一个程序从1到 n 读入每个杂务的工作说明。计算出所有杂务都被完成的最短时间。当然互相没有关系的杂务可以同时工作，并且，你可以假定 John 的农场有足够多的工人来同时完成任意多项任务。

输入格式

- 第1行：一个整数 n ，必须完成的杂务的数目($3 \leq n \leq 10,000$);
- 第2至 $(n + 1)$ 行：共有 n 行，每行有一些用1个空格隔开的整数，分别表示：
 - * 工作序号(1至 n ,在输入文件中是有序的);
 - * 完成工作所需要的时间 $len(1 \leq len \leq 100)$;
 - * 一些必须完成的准备工作，总数不超过100个，由一个数字0结束。有些杂务没有需要准备的工作只描述一个单独的0，整个输入文件中不会出现多余的空格。

输出格式

一个整数，表示完成所有杂务所需的最短时间。

输入输出样例

输入 #1

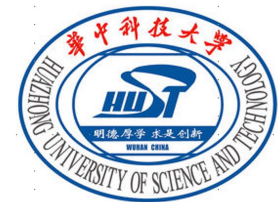
复制

7
1 5 0
2 2 1 0
3 3 2 0
4 6 1 0
5 1 2 4 0
6 8 2 4 0
7 4 3 5 6 0

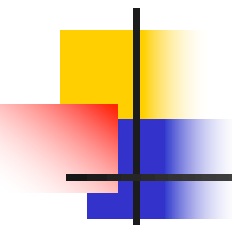
输出 #1

复制

23



```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
int n,l,t,ans[10005],maxans;
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%d",&i);
        scanf("%d",&l);
        int tmp=0;
        while(scanf("%d",&t)&&t)
            tmp=max(ans[t],tmp);
        ans[i]=tmp+1;
        maxans=max(ans[i],maxans);
    }
    printf("%d\n",maxans);
    return 0;
}
```

A decorative graphic consisting of overlapping yellow, red, and blue squares with a black crosshair is positioned in the top left corner.

Dog先生被他的公司解雇了。为了养家糊口，他必须尽快找到一份新工作。如今，很难找到工作，因为失业人数不断增加。因此，一些公司经常使用严格的测试来招聘。

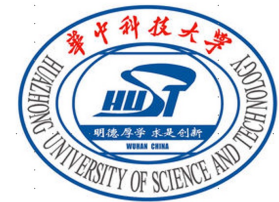
测试是这样的：从一个源城市开始，你可能会通过一些定向道路到达另一个城市。每次到达一个城市，你都可以赚取一些利润或支付一些费用，让这个过程继续下去，直到你到达一个目标城市。老板会计算你为旅行花费的费用和你刚刚获得的利润。最后，他将决定您是否可以被雇用。

为了得到这份工作，**Dog**先生设法获得了他可能达到的所有城市的净利润 V_i （负 V_i 表示花钱而不是获得）以及城市之间的联系。没有通往它的道路的城市是源城市，没有通往其他城市的道路的城市是目标城市。**Mr.Dog**的任务是从源城市开始，选择一条通往目标城市的路线，通过这条路线可以获得最大的利润。

每一组测试数据的第一行有两个整数 N 和 M ($1 \leq N \leq 100000$, $1 \leq M \leq 1000000$), 表示一共有 N 个点和 M 条边。接下来有 N 行, 每一行有一个整数, 依次给出第 1 到第 N 个点的点权。然后有 M 行, 每一行有两个整数 u 、 v , 表示从编号为 u 的点有一条边到编号为 v 的点。
两组测试数据中间没有空行分割。

输出:

对于每一组测试数据, 输出一行, 包含一个整数, 即最大的权值和。



```
16 const int MOD = 1e9 + 7;
17 const int INF = 0x3f3f3f3f;
18 const int MAXN = 1e5 + 5;
19 int n, m, num;
20 int cost[MAXN], in[MAXN], out[MAXN], head[MAXN], dp[MAXN];
21 bool vis[MAXN];
22 struct node
23 {
24     /* data */
25     int fr, to, nxt;
26 }e[MAXN * 10];
27 void add(int x, int y)
28 {
29     e[num].fr = x;
30     e[num].to = y;
31     e[num].nxt = head[x];
32     head[x] = num++;
33 }
34 void toposort()
35 {
36     int cnt = 1;
37     while(cnt < n) {
38         for(int i = 1; i <= n; ++i)
39             if(in[i] == 0 && !vis[i]) {
40                 vis[i] = true;
41                 cnt++;
42                 for(int j = head[i]; j != -1; j = e[j].nxt) {
43                     int x = e[j].to;
44                     in[x]--;
45                     if(dp[i] + cost[x] > dp[x]) dp[x] = dp[i] + cost[x];
46                 }
47             }
48     }
49 }
```

```
50 int main(int argc, char const *argv[])
51 {
52     while(scanf("%d%d", &n, &m) != EOF) {
53         memset(in, 0, sizeof(in));
54         memset(out, 0, sizeof(out));
55         memset(head, -1, sizeof(head));
56         memset(vis, false, sizeof(vis));
57         num = 1;
58         for(int i = 1; i <= n; ++i)
59             scanf("%d", &cost[i]);
60         for(int i = 1; i <= m; ++i) {
61             int x, y;
62             scanf("%d%d", &x, &y);
63             add(x, y);
64             in[y]++;
65             out[x]++;
66         }
67         for(int i = 1; i <= n; ++i)
68             if(in[i] == 0) dp[i] = cost[i];
69             else dp[i] = -INF;
70         toposort();
71         int ans = -INF;
72         for(int i = 1; i <= n; ++i)
73             if(out[i] == 0 && dp[i] > ans) ans = dp[i];
74         printf("%d\n", ans);
75     }
76     return 0;
77 }
```