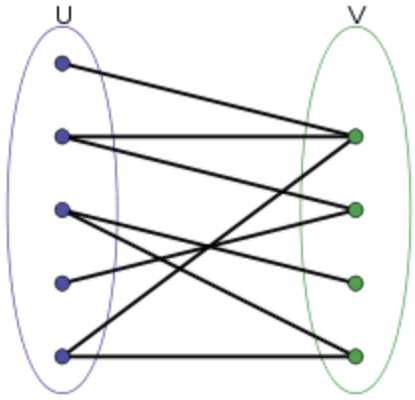
二分图

定义

二分图，又称二部图，英文名叫 Bipartite graph。

二分图是什么？节点由两个集合组成，且两个集合内部没有边的图。

换言之，存在一种方案，将节点划分成满足以上性质的两个集合。



（图源英文维基）

性质

- 如果两个集合中的点分别染成黑色和白色，可以发现二分图中的每一条边都一定是连接一个黑色点和一个白色点。
- "二分图不存在长度为奇数的环"
- 因为每一条边都是从一个集合走到另一个集合，只有走偶数次才可能回到同一个集合。

判定

如何判定一个图是不是二分图呢？

换言之，我们需要知道是否可以将图中的顶点分成两个满足条件的集合。

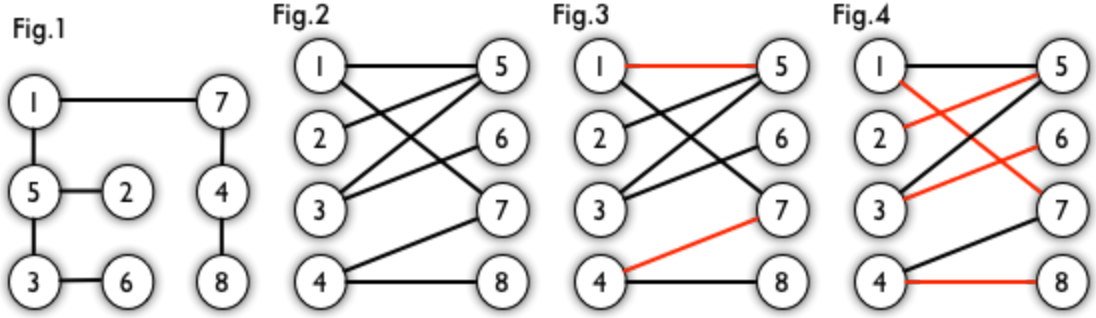
显然，直接枚举答案集合的话实在是太慢了，我们需要更高效的方法。

考虑上文提到的性质，我们可以使用DFS或者BFS来遍历这张图。如果发现了奇环，那么就不是二分图，否则是。

```
bool dfs(int now,int col)
{
    color[now] = col;
    for(int i = head[now];~i;i = edge[i].next){
        int to = edge[i].to;
        if(color[now] == color[to])return false;
        if(!color[to]){
            if(!dfs(to,3-col))return false;
        }
    }
    return true;
}

for(int i = 1;i<=n;i++)
{
    if(!color[i]){
        if(!dfs(i,1)){
            flag = false;
            break;
        }
    }
}
```

- 匹配：在图论中，一个「匹配」（matching）是一个边的集合，其中任意两条边都没有公共顶点。例如，图 3、图 4 中红色的边就是图 2 的匹配。



- **最大匹配**：一个图所有匹配中，所含匹配边数最多的匹配，称为这个图的最大匹配。图 4 是一个最大匹配，它包含 4 条匹配边。

- 完美匹配：如果一个图的某个匹配中，所有的顶点都是匹配点，那么它就是一个完美匹配。图 4 是一个完美匹配。显然，完美匹配一定是最大匹配（完美匹配的任何一个点都已经匹配，添加一条新的匹配边一定会与已有的匹配边冲突）。但并非每个图都存在完美匹配。

匈牙利算法求最大匹配

今天不讲，下周讲~

一些绕来绕去的知识点，但很重要

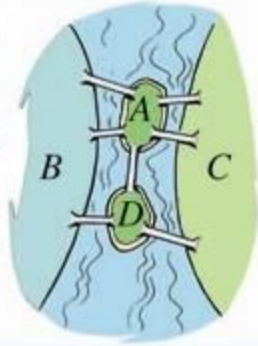
- 最小覆盖：即在所有顶点中选择最少的顶点来覆盖所有的边。
- - 二分图有两个定理：**最小覆盖数=最大匹配数、最大独立集=顶点个数-最小覆盖** 。
- 最大独立集：集合中的任何两个点都不直接相连。
- 最小路径覆盖 ： 在图中找一条最小路径，这些路径覆盖图中所有的顶点，每个顶点都只与一条路径相关联。
- 最大团: 选出一些点， 使两两之间都有边， 包含顶点数最多的集合称之为最大团.
- 图的最大团就是补图的最大独立集

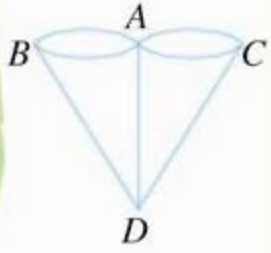
欧拉回路

- 出度
- 入度

七桥问题

18 世纪东普鲁士的哥尼斯堡城，有一条河穿过，河上有两个小岛，有七座桥把两个岛与河岸联系起来（如下图）。有人提出一个问题：一个步行者怎样才能不重复、不遗漏地一次走完七座桥，最后回到出发点。后来大数学家欧拉把它转化成一个几何问题（如右图）——一笔画问题。





定义

通过图中所有边一次且仅一次行遍所有顶点的通路称为欧拉通路。

通过图中所有边一次且仅一次行遍所有顶点的回路称为欧拉回路。

具有欧拉回路的图称为欧拉图。

具有欧拉通路的图称为半欧拉图。

有向图的时候可以类似地定义。

性质

欧拉图中所有顶点的度数都是偶数。

若 G 是欧拉图，则它若干个边不重的圈的并。

判别法

- 无向图欧拉回路判断：所有顶点的度数都为偶数。
- 有向图欧拉回路判断：所有顶点的出度与入读相等。
- 无向图欧拉路径判断： 之多有两个顶点的度数为奇数，其他顶点的度数为偶数。
- 有向图欧拉路径判断： 至多有两个顶点的入度和出度绝对值差1（若有两个这样的顶点，则必须其中一个出度大于入度，另一个入度大于出度）,其他顶点的入度与出度相等。
- 判断有向图和无向图是否存在欧拉回路和欧拉路径非常简单，就是要注意要用并查集统计图的联通分量个数。保证联通分量的个数为1个上述算法才成立。

求欧拉回路

边递归，边删边，同时在子节点全部递归完之后将当前节点放入栈中

栈的出栈顺序就是欧拉路的顺序

```
void dfs(int u)
{
    for(int i=head[u]; ~i; i=edge[i].next)
    {
        int v=e[i].v;
        if(!edge[i].flag)
        {
            edge[i].flag=1;
            edge[i^1].flag=1;
            dfs(v);
        }
    }
    s.push(u);
}
```

拓扑排序

我们都知道大学的课程是可以自己选择的，每一个学期可以自由选择打算学习的课程。唯一限制我们选课是一些课程之间的顺序关系：有的难度很大的课程可能会有一些前置课程的要求。比如课程A是课程B的前置课程，则要求先学习完A课程，才可以选择B课程。

定义

拓扑排序的英文名是 Topological sorting。

拓扑排序要解决的问题是给一个图的所有节点排序。

因此我们可以说 在一个DAG（有向无环图）中，我们间图中的顶点以线性方式进行排序，使得对于任何的顶点 u 到 v 的有向边 (u, v) , 都可以有 u 在 v 的前面。

还有给定一个DAG（有向无环图），如果从 i 到 j 有边，则认为 j 依赖于 i 。如果 i 到 j 有路径（ i 可达 j ），则称 j 间接依赖于 i 。

拓扑排序的目标是将所有节点排序，使得排在前面的节点不能依赖于排在后面的节点。

Kahn 算法

将入度为 0 的边组成一个集合 S

每次从 S 里面取出一个顶点 v （可以随便取）放入 L , 然后遍历顶点 v 的所有边 $(u_1, v), (u_2, v), (u_3, v) \cdots$, 并删除，并判断如果该边的另一个顶点，如果在移除这一条边后入度为 0 , 那么就将这个顶点放入集合 L 中。不断地重复取出顶点然后.....

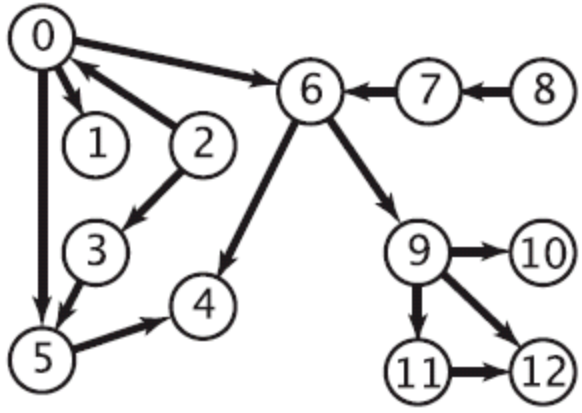
最后当集合为空后，就检查图中是否存在任何边。如果有，那么这个图一定有环路，否者返回 L , L 中顺序就是拓扑排序的结果

首先看来自Wiki的伪代码

```
L← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
    remove a node n from S
    insert n into L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least onecycle)
else
    return L (a topologically sortedorder)
```

代码的核心是，是维持一个入度为 0 的顶点。

可以参考该图



对其排序的结果就是：2 -> 8 -> 0 -> 3 -> 7 -> 1 -> 5 -> 6 -> 9 -> 4 -> 11 -> 10 -> 12

时间复杂度

假设这个图 $G = (V, E)$ 在初始化入度为 0 的集合 S 的时候就需要遍历整个图，并检查每一条边，因而有 $\mathcal{O}(E + V)$ 的复杂度。然后对该集合进行操作，显然也是需要 $\mathcal{O}(E + V)$ 的时间复杂度。

因而总的时间复杂度就有 $\mathcal{O}(E + V)$

实现

我的代码：

```
void topu()
{
    queue<int>Q;
    V.clear();
    for(int i = 1;i<=n;i++){
        if(du[i] == 0)Q.push(i);
    }
    while(!Q.empty())
    {
        int t = Q.front();
        V.push_back(t);
        Q.pop();
        for(int i = head[t];~i;i = edge[i].next)
        {
            int to = edge[i].v;
            du2[to]--;
            if(du2[to] == 0)Q.push(to);
        }
    }
}
```

病毒传播

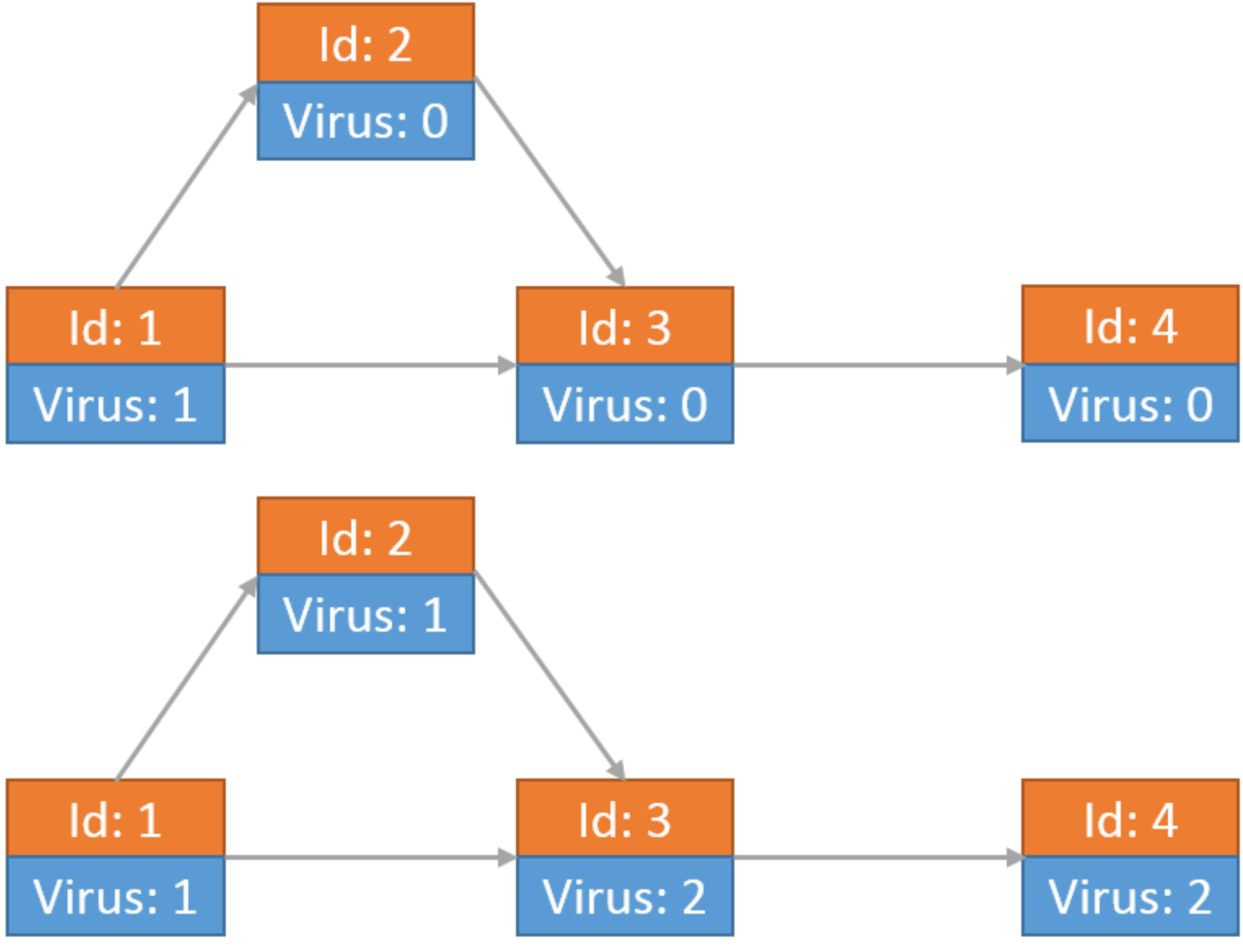
校园网主干是由N个节点(编号1..N)组成，这些节点之间有一些单向的网路连接。若存在一条网路连接(u,v)链接了节点u和节点v，则节点u可以向节点v发送信息，但是节点v不能通过该链接向节点u发送信息。

在刚感染病毒时，校园网立刻切断了一些网络链接，恰好使得剩下网络连接不存在环，避免了节点被反复感染。也就是说从节点i扩散出的病毒，一定不会再回到节点i。

当1个病毒感染了节点后，它并不会检查这个节点是否被感染，而是直接将自身的拷贝向所有邻居节点发送，它自身则会留在当前节点。所以一个节点有可能存在多个病毒。

现在已经知道黑客在一开始在K个节点上分别投放了一个病毒。

小Hi和小Ho根据目前的情况发现一段时间之后，所有的节点病毒数量一定不会再发生变化。那么对于整个网络来说，最后会有多少个病毒呢？


Id: 2
Virus: 0
Id: 1
Virus: 1
Id: 3
Virus: 0
Id: 4
Virus: 0
Id: 2
Virus: 1
Id: 1
Virus: 1
Id: 3
Virus: 2
Id: 4
Virus: 2

应用

拓扑排序可以用来判断图中是否有环，

还可以用来判断图是否是一条链。

参考

1. 离散数学及其应用。ISBN:9787111555391
2. https://blog.csdn.net/dm_vincent/article/details/7714519
3. Topological sorting,https://en.wikipedia.org/w/index.php?title=Topological_sorting&oldid=854351542

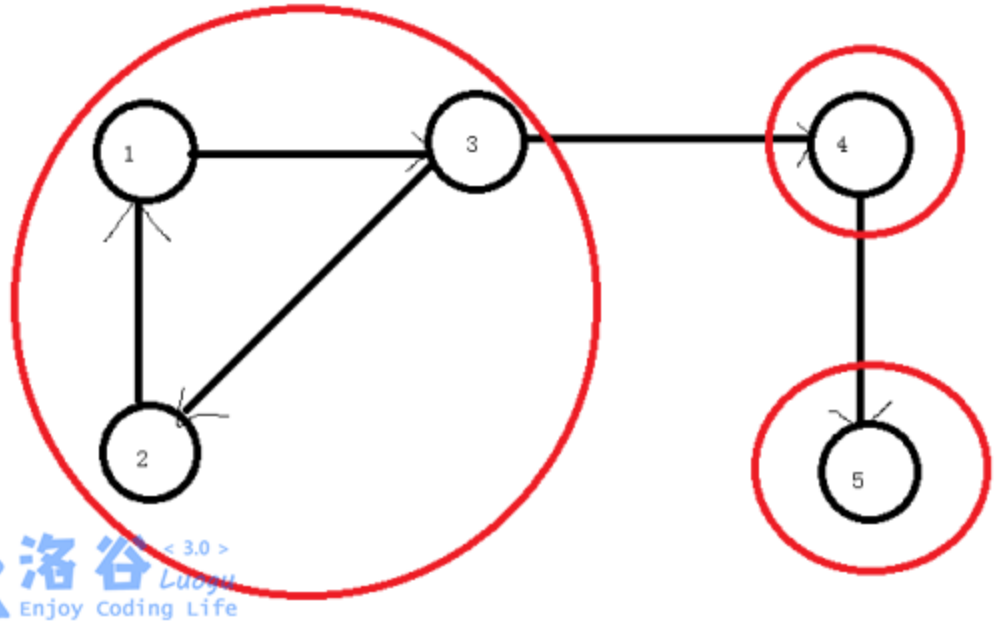
强联通分量

简介

强连通的定义是：有向图 G 强连通是指，G 中任意两个结点连通。

强连通分量（Strongly Connected Components，SCC）的定义是：极大的强连通子图。

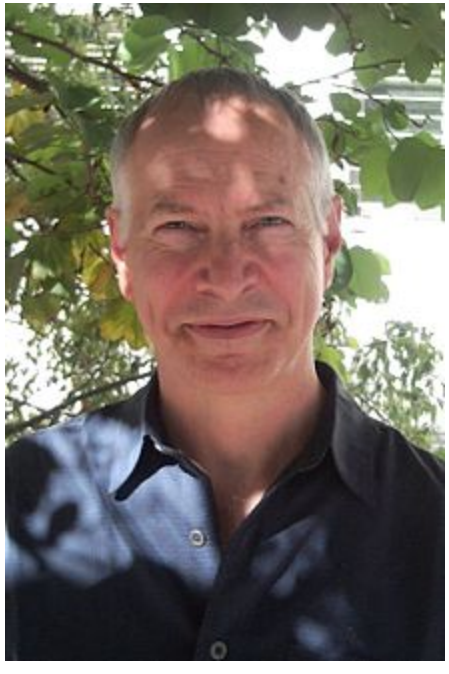
反正就是在图中找到一个最大的图，使这个图中每个两点都能够互相到达。这个最大的图称为强连通分量，同时**一个点也属于强连通分量**。



如图中强连通分量有三个：1-2-3,4,5

Tarjan 算法

Tarjan牛逼！！！



Robert E. Tarjan (1948~) 美国人。
Tarjan 发明了很多很有用的东西，下到 NOIP 上到 CTSC 难度的都有。
(举例子：Tarjan 算法，并查集，Splay 树，Tarjan 离线求 lca（Lowest Common Ancestor，最近公共祖先）等等)
我们这里要介绍的是图论中的 Tarjan 算法，用来处理各种连通性相关的问题。

定义

方便起见，我们先定义一些东西。

`dfn[x]` ：结点 x 第一次被访问的时间戳 (dfs number)

`low[x]` ：结点 x 所能访问到的点的 dfn 值的最小值

这里的树指的是 DFS 树

所有结点按 dfn 排序即可得 dfs 序列

DFS 树的性质

一个结点的子树内结点的 dfn 都大于该结点的 dfn。

从根开始的一条路径上的 dfn 严格递增。

一棵 DFS 树被构造出来后，考虑图中的非树边。

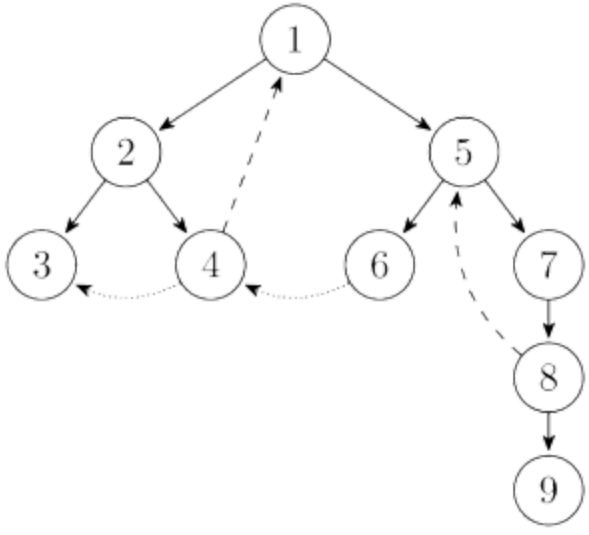
前向边 (forward edge)：祖先→儿子

后向边 (backward edge)：儿子→祖先

横叉边 (cross edge)：没有祖先—儿子关系的

注意：横叉边只会往 dfn 减小的方向连接

注意：在无向图中，没有横叉边（为什么？）


1
2
5
3
4
6
7
8
9

我们需要知道某个节点u下面有没有节点最后又指向u，如果有，那么这个环就形成了连通分量，最深的那个点到u点就是强连通分量，用栈去维护。

我们因此需要知道任何一个点能到达的最小的dfs序是多少

栈一直回溯到子树的根，即 $low[i] == dfn[i]$

实现

```
void dfs(int now)
{
        dfn[now] = low[now] = ++indx;
        S.push(now);
        instack[now] = true;
        for(int i = head[now];~i;i = edge[i].next)
        {
                int to = edge[i].v;
                if(!dfn[to]){
                        dfs(to);
                        low[now] = min(low[now] , low[to]);
                }else if(instack[to]){
                        low[now] = min(low[now] , dfn[to]);
                }
        }
        if(dfn[now] == low[now]){;
                while(S.top()!=now){
                        int t = S.top();S.pop();
                        instack[t] = false;
                }
                instack[now] = false;S.pop();
        }
}
```

应用

我们可以将一张图的每个强连通分量都缩成一个点。

然后这张图会变成一个 DAG。

怎么缩点？

给同一批弹出栈的点赋予一个新的点的id

```
if(dfn[now] == low[now]){
    id[now] = ++tot;
    while(S.top()!=now){
        int t = S.top();
        S.pop();
        instack[t] = false;
        id[t]= tot;
    }
    instack[now] = false;
    S.pop();
}
```

DAG 好啊，能拓扑排序了就能做很多事情了。

举个简单的例子，求一条路径，可以经过重复结点，要求经过的不同结点数量最多。

End