拓扑排序

一.概念

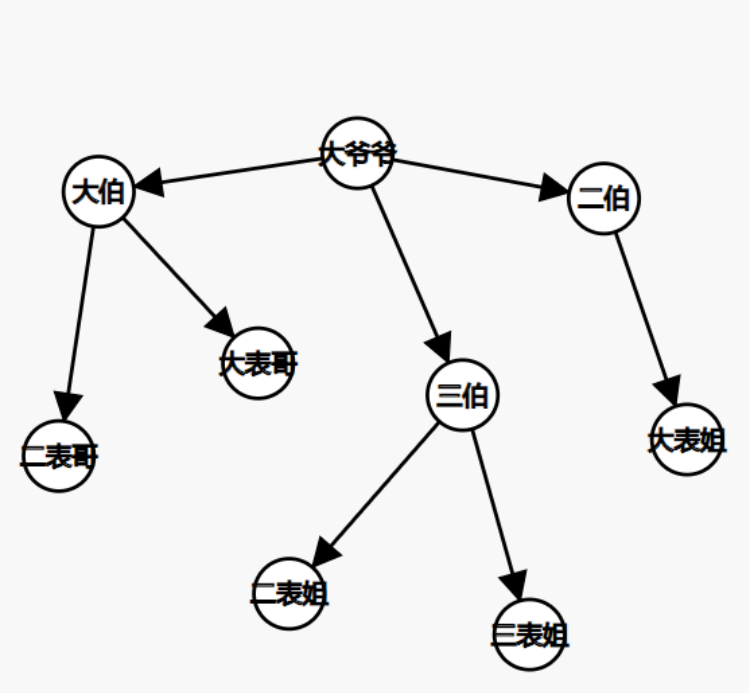
在图论中，拓扑排序是一个常见且重要的考点。
拓扑排序是一个**有向无环图**（DAG）的所有顶点的线性序列，且该序列必须满足下面2个条件：

1. **每个顶点出现且只能出现一次**
2. **若存在一条从顶点A到顶点B的路径，那么在序列中顶点A出现在顶点B的前面**

简而言之：如果出现前置条件，必须先做A才能做B，有一定的先后顺序就得考虑拓扑排序

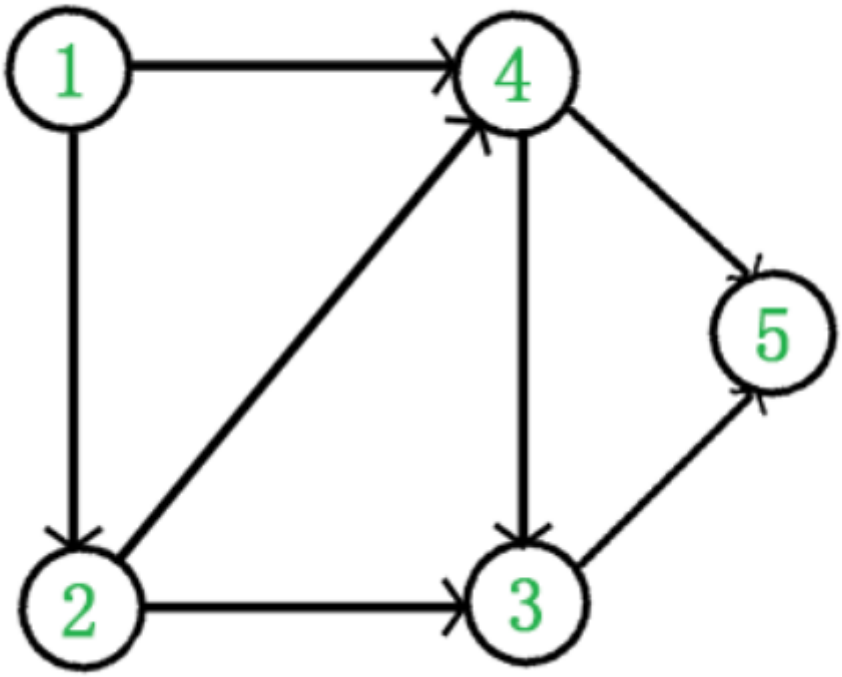
另外我们也可以发现，如果出现了**环**，上述条件2是无法满足的；如果是**无向图**，先后顺序也会混乱。所以有向无环图（DAG）才有拓扑排序，非DAG图并没有拓扑排序一说。

举个拓扑排序生活中的例子，如下：
每个家庭都会有一个家谱，有个人的家族很大，辈分关系很混乱，请你帮整理一下这种关系。
输出一个序列，使得每个人的后辈都比那个人后列出



比如： 大爷爷---> 大伯 ---> 二伯 ---> 三伯 ---> 二表姐 ---> 三表姐 ---> 大表姐 ---> 大表哥 ---> 二表哥
这里的随便一想就发现，**情况可能会有多种**，此处不加赘述，各位可以自行探讨

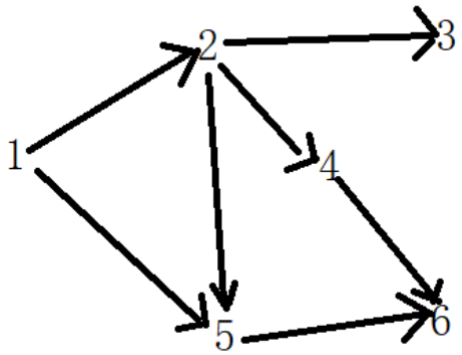
那我们把上述情况转化为图，如下



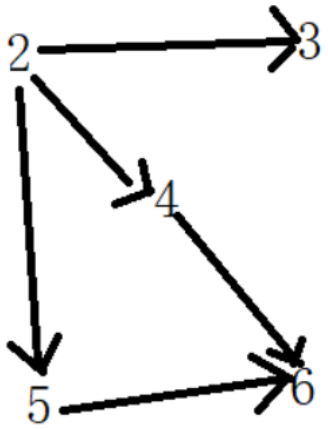
大家也可以思考一下本图的拓扑序有哪些?

二.拓扑排序规则

我们需要结合入度和出度的概念，如果箭头指向本顶点，则本顶点的入度就+1，箭头指出，则出度+1。
拓扑排序就是将入度为0的结点一个一个找出来，请看以下例子:



上图入度为0的是点1，则点1排在前面，此时拓扑排序为{1}，接下来将点1和从1出发的边全部删除
继续排序，找入度为0的顶点



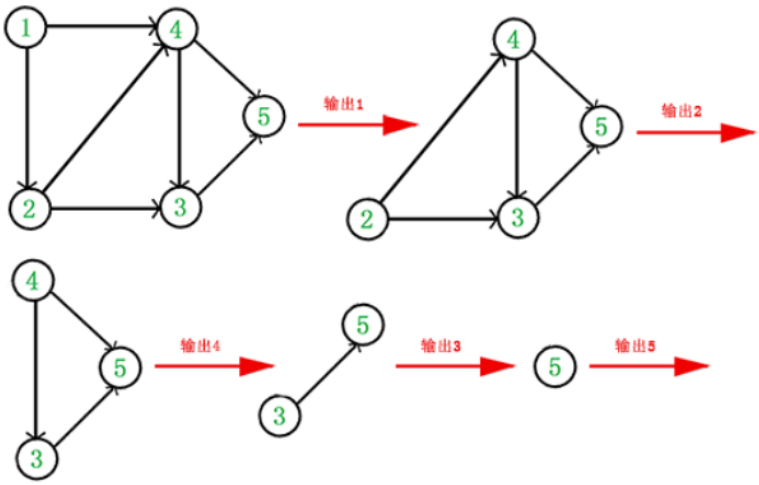
上图中点2的入队为0，所以2进入拓扑排序中，此时拓扑排序为{1,2}，接下来将顶点2和2的所有出边删除，
继续找入度为0的顶点

如上图，此时入度为0的点有3,4,5 三个顶点，所以这三个顶点谁在前面或者后面都行【这也说明了，拓扑排序结果不具备唯一性】。我们此时按照3,4,5去存储，那么拓扑排序为{1,2,3,4,5}，
最后处理还剩一个节点6， 6入序列。
最终拓扑排序为{1,2,3,4,5,6}

三. Kahn(卡恩)算法

它是一个 DAG 图，那么如何写出它的拓扑排序呢？这里说一种比较常用的方法：

1. 从 DAG 图中选择一个 没有前驱（即入度为0）的顶点并输出。
2. 从图中删除该顶点和所有以它为起点的有向边。
3. 重复 1 和 2 直到当前的 DAG 图为空或**当前图中不存在无前驱的顶点为止**。后一种情况说明有向图中必然存在环。



于是，得到拓扑排序后的结果是 { 1, 2, 4, 3, 5 }。

通常，一个有向无环图可以有**一个或多个**拓扑排序序列。

算法核心是用队列去维护一个入度为0的节点的集合

1. **初始化，队列q压入所有入度为0的节点的集合**
2. **每次从q中取出一个点x放入数组储存答案**
3. **然后将x的所有出边删除。若将边 x--->y 删除后，y的入度-1 ， 当y的入度等于0的时候，将y入队**
4. **不断重复2和3过程，知道队列为空**
5. **若数组中元素个数等于n，则说明有拓扑序，否则，则说明有环**

代码如下：

```
#include<bits/stdc++.h>
using namespace std;
int du[105] , n , cnt , first[105] , t , m , u , v;
queue<int> que;
struct node{
    int to,next;
}edge[10005];
inline void add(int u,int v){
    cnt++;
    edge[cnt].to = v;
    edge[cnt].next = first[u];
    first[u] = cnt;
}
```

```
signed main() {

    cin >> n >> m;
    for(int i=1; i<=m; i++) {
        cin >> u >> v;
        add(u,v);
        du[v]++;
    }
    for(int i=1; i<=n; i++) {
        if( du[i]==0 ) que.push(i);
    }
    while( !que.empty() ) {
        u = que.front();
        cout << u << " ";
        que.pop();
        for(int i=first[u] ; i!=0 ; i=edge[i].next){
            v = edge[i].to;
            du[v]--;
            if( du[v]==0 ) que.push(v);
        }
    }

    return 0;
}
```