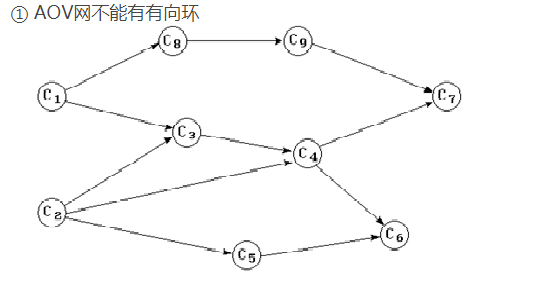
深夜十一点半

首先什么是AOV，通俗点，就是一堆活动，这些活动是有前置条件的，那就是上一个活动要完成。上例子

上面的图就是c8开始前要c1先完成才能开始，这样的图是不能闭环的，很容易理解。

而aov的拓扑排序呢就是要在一堆数据中排好，就像上面的图那样

先讲下思路，待会就要睡了。首先找到入度为0的点。存入队列或栈，找完后输出队列的点并把涉及到的点的入度数量-1。再找一次入度为0的。直到没有点。我的实现就是基于这样的想法。但是没有用栈或队列存储找到的点。速度有些慢。我的实现的时间复杂度应该是最差情况是n的平方。有点慢。接下来怎么判断改图不是合法的AOV即是有闭环的呢？就是你判断到最后就会有点是相互指向的。就是你找不到入度为0的，但是图上还有点。那就是闭环。很简单。个人觉得没什么难度。

上代码（js实现，数据是固定的）

window.onload = function ()

{

// u数组是出发顶点，一共有0、1、2、3、4、5

var u = new Array(0,0,1,2,3,3,4,4);

// v数组是终点顶点，一共有0、1、2、3、4、5

// 这样就可以记录u->v

var v = new Array(2,1,4,3,1,5,3,5);

// 使用邻接表来存储图的信息

// 关于邻接表之前有讲过

var first = new Array(1,2,3,5,7,-1);

var next = new Array(-1,0,-1,-1,-1,4,-1,6);

// con数组是用来记录每个顶点的入度的

var con = new Array(6);

var k;

// point记录的是输出的顶点个数，prePoint记录的是上一次输出的顶点个数

var point = 0,prePoint = 0;

// 初始化操作，con数组都为0

for(let i = 0;i < 6;i ++)

{

con[i] = 0;

}

function TopSort ()

{

// 初始化操作，con数组记录每个顶点的入度

// 8为v数组长度，这个循环的意思是，遍历v数组，

// 又因为con数组的下标对应的是每个顶点，这样

// 每一次循环v[i]出来的都是终点，刚好对应con数组的下标

// 就可以con[v[i]] ++

for(let i = 0;i < 8;i ++)

{

con[v[i]] ++;

}

// 当point不等于顶点数时，进行循环

// 当为有向图时，该循环会一直循环下去

while(point != 6)

{

prePoint = point;

// 每次进来遍历六个顶点

for(let i = 0;i < 6;i ++)

{

// 找到入度为0的点

if(con[i] == 0)

{

// 输出

alert(i);

// 找到被输出的点的的出度

// 终点顶点的入度-1

k = first[i];

while(k != -1)

{

con[v[k]] --;

k = next[k];

}

// 输出的点的个数+1

point ++;

// 被输出的点的入度点置为-1

con[i] = -1;

}

}

// 判断有向环

if (prePoint == point) {

alert("Network has a cycle");

break;

}

}

}

TopSort();

}

接下来是AOE关键路径求法，AOE就是在AOV的路径上加权值。继续上面的图，c1到c8是要时间的，关键路径就是相当于找木桶的最长的长板。关键路径的求法其实就是先正向拓扑排序一遍，再逆向拓扑排序一遍。在正向拓扑排序时，输出入度为0的顶点的时候，就可以计算该输出的顶点的出度的最早开始时间（E[i]）了。比如说顶点0到顶点1要a1 = 6，顶点0到顶点2要啊a2 = 4，顶点1到顶点4要a4 = 1，顶点2到顶点4要a5 = 1.现在要计算顶点4的最早开始时间。那么就是在输出顶点1的时候，顶点1的最早开始时间是6（顶点0到顶点1），就是a1加a4，在输出顶点2的时候，就是a2加a5，这时候比较两者谁大，取较大值。正向拓扑排序算完所有点的最早开始时间后。就要开始逆向拓扑排序，就是输出出度为0的点。构造两个AOE，一个正向排序用，一个逆向排序用。这个是具体的实现，先不管。先讲下逆向拓扑排序要做的事情。在输出出度为0的点的时候，计算最晚开始时间。用输出的点的最晚开始时间减去入度的点到输出的点的权值。一轮下来后，看哪些点是最早开始时间等于最晚开始时间的，那个就是关键路径要走过的点，称为关键活动。输出。就行了