拓扑排序

描述

有向无环图（directed acycline graph,简称 DAG 图），是描述工程或系统的进行过程的有效工具。如果有向图中从顶点 v 到 w 有一条有向路径，则 v 一定排在 w 之前，这样构成的一个顶点序列就称为拓扑序，构造拓扑序的过程就是拓扑排序。若拓扑排序不能输出所有顶点，说明 AOV 网络中存在有向环，此 AOV 网络所代表的工程是不可行的。 拓扑排序的算法有 2 种：1 种是删边法，另一种是采用 DFS 深度优先搜索的方法。 本题给出一组有向图，请采用一种拓扑排序方法判断该图是否有环。

输入

第一行输入组数 t

第二行输入两个整数 n、m，表示该图共有 n 个结点和 m 条有向边。（n<=100，m<=100）

接下来 m 行每行包含 2 个整数 ui、vi，表示有一条 ui 指向 vi 的弧。

顶点编号从 1 开始。

本题保证不存在重边和自环。

输出

若有环，输出 0，否则输出 1。

样例输入 1

2

2 2

1 2

2 1

4 5

1 2

1 3

1 4

2 3

3 4

样例输出 1

0

1

```
#include<iostream>
#include<cstring>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 20
int n, m;
typedef struct ArcNode
{
    int adjvex;//弧指向的顶点的位置
    ArcNode *nextarc;//指向下一个与该顶点邻接的顶点
    int info;//弧的相关信息
}ArcNode;//边表结点
typedef struct VNode
{
    VertexType data;//用于存储顶点
    int indegree;
    ArcNode *firstarc;//指向第一个与该顶点邻接的顶点
}VNode, AdjList[MAX_VERTEX_NUM];//表头节点，顺序表存储
typedef struct
{
    AdjList vertices;//邻接表
    AdjList _vertices;//逆邻接表
    int vexnum, arcnum;//边数，顶点数
    int kind;//图的种类
}ALGraph;
int LocateVertex_L(ALGraph& G, VertexType v)
{
    int i;
    for (i = 1; i <= G.vexnum; i++)
        if (G.vertices[i].data == v)
            return i;
    return -1;
}
void CreateGraph(ALGraph &G)
{
    int i, j, k;
```

```
    VertexType v1, v2;
    ArcNode *p;
    cin >> n >> m;
    G.vexnum = n;
    G.arcnum = m;
    for (i = 1; i <= G.vexnum; i++)
    {
        G.vertices[i].data = i;
        G._vertices[i].data = i;
        G.vertices[i].firstarc = NULL;
        G._vertices[i].firstarc = NULL;
    }
    for (k = 1; k <= G.arcnum; k++)
    {
        cin >> v1 >> v2;
        i = LocateVertex_L(G, v1);
        j = LocateVertex_L(G, v2);
        /*j为入i为出创建邻接链表*/
        p = new ArcNode;
        p->adjvex = j;
        p->nextarc = G.vertices[i].firstarc;
        G.vertices[i].firstarc = p;
        /*i为入j为出创建逆邻接链表*/
        p = new ArcNode;
        p->adjvex = i;
        p->nextarc = G._vertices[j].firstarc;
        G._vertices[j].firstarc = p;
    }
}
void FindIndegree(ALGraph &G)
{
    ArcNode *p;
    for (int i = 1; i <= G.vexnum; i++)
    {
        int count = 0;
        p = G._vertices[i].firstarc;
        while (p)
        {
            count++;
            p = p->nextarc;
        }
        G.vertices[i].indegree = count;
    }
}
```

```
int ToplogicalSort(ALGraph &G)
{  //普通拓扑排序
    ArcNode *p;
    FindIndegree(G);
    int top = -1;//stack<int>S;
    int i, k;
    for (i = 1; i <= G.vexnum; i++)
        if (G.vertices[i].indegree == 0)
        {
            G.vertices[i].indegree = top;
            top = i;
        }//入度为0，进栈,S.push(G.vertices[i].data);
    int count = 0;
    while (top != -1)//栈不空
    {
        i = top;
        top = G.vertices[top].indegree;//出栈,S.pop();
        //cout << G.vertices[i].data << " ";//cout<<S.top()<<" ";
        count++;
        for (p = G.vertices[i].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex;
            G.vertices[k].indegree--;
            if (G.vertices[k].indegree == 0)
            {
                G.vertices[k].indegree = top;
                top = k;
            }//度为0，入栈,S.push(G.vertices[k].data);
        }
    }
    if (count < G.vexnum)
        return 0;//有环
    return 1;
}

int main()
{
    int t;
    cin >> t;
    ALGraph G;
    for (int i = 1; i <= t; i++)
    {
        CreateGraph(G);
        cout << ToplogicalSort(G) << endl;
```

```
    }
    return 0;
}
```

关键路径

描述

一个工程项目由一组活动（或称子任务）构成，活动之间有的可以并行执行，有的必须在完成了其它一些活动后才能执行，并且每个活动完成需要一定的时间。对于一个工程，需要研究的问题是：（1）由这样一组活动描述的工程是否可行？（2）若可行，计算完成整个工程需要的最短时间。（3）这些活动中，哪些活动是关键活动（也就是必须按时完成的任务，否则整个项目就要延迟）。 现给定一个 AOE 网，有向边（弧）表示活动，弧上权值表示活动完成需要的时间。请你编写程序，回答上述三个问题。

注意：求关键路径 ve 要按照拓扑序列的顺序计算，vl 要按照拓扑序列逆序计算。本题不保证顶点已按拓扑排序从小到大编号。可参照课本增加一个栈存放拓扑序列。

输入

第一行包含两个整数 n、m，其中 n 表示顶点数，m 表示活动数。顶点按 1~n 编号。（n<=100）

接下来 m 行表示 m 个活动，每行包含三个正整数 ui、vi、wi，分别表示该活动开始和完成的顶点序号，及活动完成所需时间。

整数之间用空格分割。

输出

如果该有向图有环，则工程不可行，输出 0；否则

第 1 行输出完成整个工程项目需要的时间，

从第 2 行开始输出所有关键活动，每个关键活动占一行，按格式“vi->vj”输出，其中 vi 和 vj 为弧尾和弧头。

注：关键活动输出的顺序规则是：弧尾编号小者优先，弧尾编号相同时，与边输入的顺序相反。

样例输入 1

7 8

1 2 4

1 3 3

2 4 5

3 4 3

4 5 1

4 6 6

5 7 5

6 7 2

样例输出 1

17

1->2

2->4

4->6

6->7

```
#include<iostream>
#include<stack>
#include<cstring>
using namespace std;
typedef int VertexType;
#define MAX_VERTEX_NUM 20
int n, m;
stack<int>T;
int Vl[MAX_VERTEX_NUM];
int Ve[MAX_VERTEX_NUM];
typedef struct ArcNode
{
    int adjvex;//弧指向的顶点的位置
    ArcNode *nextarc;//指向下一个与该顶点邻接的顶点
    int info;//弧的相关信息
}ArcNode;//边表结点
typedef struct VNode
```

```
{
    VertexType data;//用于存储顶点
    int indegree;
    ArcNode *firstarc;//指向第一个与该顶点邻接的顶点
}VNode, AdjList[MAX_VERTEX_NUM];//表头节点，顺序表存储
typedef struct
{
    AdjList vertices;//邻接表
    AdjList _vertices;//逆邻接表
    int vexnum, arcnum;//边数，顶点数
    int kind;//图的种类
}ALGraph;
int LocateVertex_L(ALGraph& G, VertexType v)
{
    int i;
    for (i = 1; i <= G.vexnum; i++)
        if (G.vertices[i].data == v)
            return i;
    return -1;
}
void CreateGraph(ALGraph &G)
{
    int i, j, k;
    VertexType v1, v2;
    ArcNode *p;
    int w;
    cin >> n >> m;
    G.vexnum = n;
    G.arcnum = m;
    for (i = 1; i <= G.vexnum; i++)
    {
        G.vertices[i].data = i;
        G._vertices[i].data = i;
        G._vertices[i].firstarc = NULL;
        G.vertices[i].firstarc = NULL;
    }
    for (k = 1; k <= G.arcnum; k++)
    {
        cin >> v1 >> v2 >> w;
        i = LocateVertex_L(G, v1);
        j = LocateVertex_L(G, v2);
        /*j为入i为出创建邻接链表*/
        p = new ArcNode;
        p->adjvex = j;
```

```
        p->info = w;
        p->nextarc = G.vertices[i].firstarc;
        G.vertices[i].firstarc = p;
        /*i为入j为出创建逆邻接链表*/
        p = new ArcNode;
        p->adjvex = i;
        p->nextarc = G._vertices[j].firstarc;
        G._vertices[j].firstarc = p;
    }
}
void FindIndegree(ALGraph &G)
{
    ArcNode *p;
    for (int i = 1; i <= G.vexnum; i++)
    {
        int count = 0;
        p = G._vertices[i].firstarc;
        while (p)
        {
            count++;
            p = p->nextarc;
        }
        G.vertices[i].indegree = count;
    }
}
int ToplogicalSort_CP(ALGraph &G)
{   //CP:Critical Path
    //求关键路径时候的拓扑排序（与以上方法不同的地方只有一句，
    //再者就是这种方法新建了栈，前者则借助indegree作为栈来使用）
    //T:拓扑序列顶点栈
    memset(Ve, 0, sizeof(Ve));
    ArcNode *p;
    int count = 0;
    FindIndegree(G);//求各顶点的入度
    stack<int>S;//0入度顶点栈
    int i, k;
    for (i = 1; i <= G.vexnum; i++)
        if (G.vertices[i].indegree == 0)
            S.push(i);//入度为0，进栈;
    while (!S.empty())//栈不空
    {
        int temp;
        temp = S.top();
        T.push(temp);//拓扑序列元素下标入栈
```

```
        S.pop();//出栈
        count++;//计数
        for (p = G.vertices[temp].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex;
            if (--G.vertices[k].indegree == 0)
                S.push(k);//度为0，入栈
            if (Ve[temp] + p->info > Ve[k])
                Ve[k] = Ve[temp] + p->info;//修改事件v[k]的最早发生时
间,为各条路径时间和的最大值
        }//对以G.vertices[S.top()]为顶点的弧的另一个顶点进行操作
    }
    if (count < G.vexnum)
        return 0;//有环
    return 1;
}
void CriticalPath(ALGraph &G)
{
    ArcNode *p;
    int dut, k;
    int Ee, El;//活动（即边E）最早发生和最迟开始的时间
    for (k = 1; k <= G.vexnum; k++)
        Vl[k] = Ve[G.vexnum];
    //初始化"事件最迟发生时间"数组为Ve数组最大者
    while (!T.empty())
    {
        int temp;
        temp = T.top();
        T.pop();
        for (p = G.vertices[temp].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex;
            dut = p->info;
            if (Vl[k] - dut < Vl[temp])
                Vl[temp] = Vl[k] - dut;//修改事件v[k]的最迟发生时间，
为最小值
        }
    }
    cout << Ve[G.vexnum] << endl;
    for (int j = 1; j <= G.vexnum; j++)
        for (p = G.vertices[j].firstarc; p; p = p->nextarc)//求解Ee，
El和关键活动
        {
            k = p->adjvex;
```

```
            dut = p->info;
            Ee = Ve[j];
            El = Vl[k] - dut;
            if (Ee == El)
                cout << j << "->" << k << endl;
        }
}
int main()
{
    ALGraph G;
    CreateGraph(G);
    if (!ToplogicalSort_CP(G))
        cout << 0 << endl;
    else
        CriticalPath(G);
    return 0;
}
```