**拓扑排序、关键路径**

**#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;**

**}**

**#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;**

**}**