#include <iostream>

#include<list>

#include<stack>

using namespace std;

#define numVertexes 9

typedef struct EdgeNode

{

int adjvex;

int weight;

}EdgeNode;

typedef struct VertextNode

{

int in;

int data;

list<EdgeNode> \*a;

}VertextNode;

VertextNode adjList[numVertexes];

list<EdgeNode> j[numVertexes];

int etv[numVertexes] = { 0 };

int ltv[numVertexes] = { 0 };

stack<int> s1;//用于存储拓扑序列的栈

//拓扑算法

bool TopologicalSort()

{

int i, k, gettop;

int num = 0;

stack<int> s;//用于存储入度为0的顶点

for (i = 0; i<numVertexes; ++i)

{

if (0 == adjList[i].in)

{

s.push(i);

}

}

while (!s.empty())

{

gettop = s.top();

s.pop();

cout << adjList[gettop].data << " ";

s1.push(gettop);

num++;

for (list<EdgeNode> a = \*adjList[gettop].a; !a.empty(); a.pop\_front())

{

EdgeNode e = a.front();

k = e.adjvex;

adjList[k].in--;

if (!adjList[k].in)

{

s.push(k);

}

if ((etv[gettop] + e.weight)>etv[k])

{

etv[k] = etv[gettop] + e.weight;

}

}

}

cout << endl;

if (num<numVertexes)

return 0;

else

return 1;

}

//求关键路径

void CriticalPath()

{

int i, gettop, k, j;

int ete, lte;

//调用改进后的拓扑排序，求出etv和s1的值

TopologicalSort();

//初始化ltv都为汇点的时间

for (i = 0; i < numVertexes; ++i)

{

ltv[i] = etv[numVertexes - 1];

}

//从汇点倒过来逐个计算ltv

while (!s1.empty())

{

gettop = s1.top();

s1.pop();

for (list<EdgeNode> a = \*adjList[gettop].a; !a.empty(); a.pop\_front())

{

EdgeNode e = a.front();

k = e.adjvex;

if ((ltv[k] - e.weight) < ltv[gettop])

{

ltv[gettop] = ltv[k] - e.weight;

}

}

}

//通过etv和ltv求ete和lte

for (j = 0; j < numVertexes; j++)

{

for (list<EdgeNode> a = \*adjList[j].a; !a.empty(); a.pop\_front())

{

EdgeNode e = a.front();

k = e.adjvex;

ete = etv[j];

lte = ltv[k] - e.weight;

if (ete == lte)

{

cout << adjList[j].data << "->" << adjList[k].data << ":" << e.weight << endl;

}

}

}

}

int main()

{

int num, weight;

for (int i = 0; i < numVertexes; ++i)

{

adjList[i].a = j + i;

adjList[i].data = i;

cin >> adjList[i].in;

cin >> num >> weight;

while (num!=0&&weight!=0)

{

EdgeNode e;

e.adjvex = num;

e.weight = weight;

j[i].push\_back(e);

cin >> num >> weight;

}

}

CriticalPath();

return 0;

}

输入：

0 1 6 2 4 3 5 0 0

1 4 1 0 0

1 4 1 0 0

1 5 2 0 0

2 6 7 7 5 0 0

1 7 4 0 0

1 8 2 0 0

2 8 4 0 0

2 0 0

输出：

0 3 5 2 1 4 7 6 8//拓扑排列

/\*以下为所经过的关键点及各个关键路线的权值\*/

0->1:6

1->4:1

4->6:7

4->7:5

6->8:2

7->8:4

所用的图与输入的数据有减一的差别



