项目说明文档

数据结构课程设计

——关键活动

作 者 姓 名： 李佳诺

学 号： 1751188

指 导 教 师： 张 颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 目录

1 分析 2

1.1 背景分析 2

1.2 功能分析 3

2 设计 3

2.1 算法设计 3

2.2 数据结构设计 3

2.3 成员函数设计 4

3 实现 5

3.1 拓扑排序（计算入度） TopSort\_Earliest() 5

拓扑排序（计算入度）流程图 5

3.2 拓扑排序（计算出度）TopSort\_Latest() 8

拓扑排序（计算出度）流程图 8

4 测试 10

4.1 简单情况测试 10

4.2 一般情况测试，单个起点和单个终点 11

4.3 不可行的方案测试 11

# 1 分析

## 背景分析

本实验项目是要求在任务调度问题中，如果还给出了完成每个字任务需要的时间，则可以算出完成整个工程项目需要的最短时间。在这些子任务中，有些任务即使推迟几天完成，也不会影响全局的工期；但是有些任务必须准时完成，否则整个项目的工期就要因此而延误，这些任务叫做“关键活动”。

请编写程序判定一个给定的工程项目的任务调度是否可行；如果该调度方案可行，则计算完成整个项目需要的最短时间，并且输出所有的关键活动。

## 功能分析

输入说明：输入第1行给出两个正整数N（N<=100）和M，其中N是任务交接点（即衔接两个项目依赖的两个子任务的结点，例如：若任务2要在任务1完成后才开始，则两个任务之间必有一个交接点）的数量，交接点按1～N编号，M是字任务的数量，依次编号为1～M。随后M行，每行给出3个正整数，分别是该任务开始和完成设计的交接点编号以及完成该任务所需要的时间，整数间用空格分隔。

输出说明：如果任务调度不可行，则输出0；否则第一行输出完成整个项目所需要的时间，第2行开始输出所有关键活动，每个关键活动占一行，按照格式“v->W”输出，其中V和W为该任务开始和完成涉及的交接点编号。关键活动输出的顺序规则是：任务开始的交接点编号小者优先，起点编号相同时，与输入时任务的顺序相反。

# 设计

## 算法设计

先做一次拓扑排序（计算入度），算出每个活动的最早完成时间，然后再反过来做一次拓扑排序（计算出度），算出每个活动的最晚完成时间，顺便也算出所有活动的机动时间，最后机动时间等于0的就是关键路径了。

## 数据结构设计

本项目所用到的数据结构有队列、AOV网络

## 成员函数设计

int getMax(int arr[])

int TopSort\_Earliest()

void TopSort\_Latest()

void PrintKeyRoute()

# 实现

## 拓扑排序（计算入度） TopSort\_Earliest()

### 拓扑排序（计算入度）流程图

开 始

入度为0的有向边的终点入队，并删除其出边

是否遍历所有终点？

拓扑排序完成

cnt == N ?

AOV网络中存在有向环

结 束

否

是

否

是

int TopSort\_Earliest()

{

int V, cnt = 0, Indegree[MAX] = { 0 };

queue<int> q;

//计算各结点的入度

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

for (int j = 0; j < N; j++)

if (A[i][j] != INFINITY)

Indegree[j]++; //对于有向边<i,j>累计终点j的入度

//入度为0的入队

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

if (Indegree[i] == 0)

q.push(i);

while (!q.empty())

{

V = q.front();

q.pop();

cnt++;

for (int j = 0; j < N; j++)

if (A[V][j] != INFINITY)

{ //<V, j>有有向边

if (EarliestTime[V] + A[V][j] > EarliestTime[j])

{ //如果 V的最早完成时间 + j所需时间 > j的最早完成时间

EarliestTime[j] = EarliestTime[V] + A[V][j];

}

if (--Indegree[j] == 0) //去掉V后，如果j的入度为0

q.push(j);

}

}

ECT = getMax(EarliestTime); //最早完成时间应是所有元素中最大的

if (cnt != N) return 0; //如果没有取出所有元素，说明图中有回路

else return 1;

}

## 拓扑排序（计算出度）TopSort\_Latest()

### 拓扑排序（计算出度）流程图

开 始

出度为0的有向边的终点入队，并删除其出边

是否遍历所有终点？

否

是

结 束

void TopSort\_Latest()

{

int V, Outdegree[MAX] = { 0 };

queue<int> q;

//计算各结点的出度

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

for (int j = 0; j < N; j++)

if (A[i][j] != INFINITY)

Outdegree[i]++; //对于有向边<i,j>累计起点i的出度

//出度为0的入队

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

if (Outdegree[i] == 0)

q.push(i);

//初始化LatestTime

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

LatestTime[i] = INFINITY;

LatestTime[idx] = ECT; //将最后一个活动的最晚完成时间设为它的最早完成时间

while (!q.empty())

{

V = q.front();

q.pop();

//cnt++; //不需要再算cnt了

for (int j = 0; j < N; j++)

if (A[j][V] != INFINITY)

{ //<j, V>有有向边

if (LatestTime[V] - A[j][V] <= LatestTime[j])

{ //必须用<=,只<的话只能算一条关键路径，<=才能算出所有的关键路径(错误原因)

LatestTime[j] = LatestTime[V] - A[j][V];

D[j][V] = LatestTime[V] - EarliestTime[j] - A[j][V];

}

if (--Outdegree[j] == 0) //去掉V后，如果j的出度为0

q.push(j);

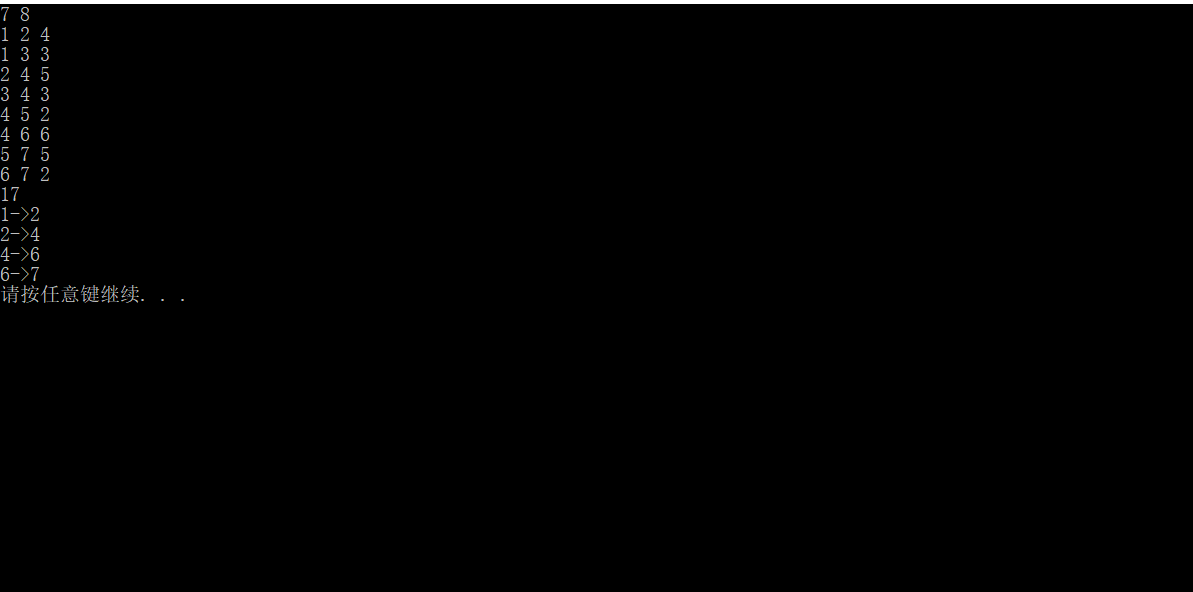
}

}

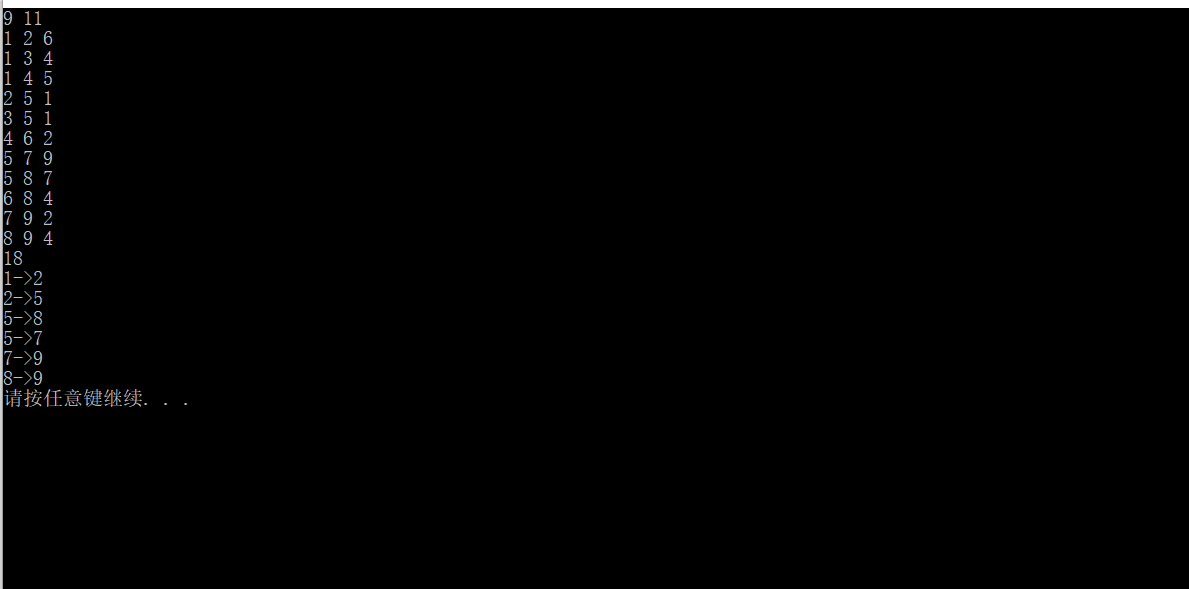
}

# 测试

## 简单情况测试



## 一般情况测试，单个起点和单个终点



## 4.3 不可行的方案测试

