项目说明文档

数据结构课程设计

——关键活动

作 者 姓 名： 王星洲

学 号： 1652977

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 1](#_Toc495668153)

[1.1 背景分析 1](#_Toc495668154)

[1.2 功能分析 1](#_Toc495668155)

[2 设计 2](#_Toc495668156)

[2.1 数据结构设计 2](#_Toc495668157)

[2.2 类结构设计 2](#_Toc495668158)

[2.3 成员与操作设计 2](#_Toc495668159)

[2.4 系统设计 4](#_Toc495668160)

[3 实现 4](#_Toc495668161)

[3.1 插入功能的实现 4](#_Toc495668162)

[3.1.1 插入功能流程图 4](#_Toc495668163)

[3.1.2 插入功能核心代码 5](#_Toc495668164)

[3.1.3 插入功能截屏示例 5](#_Toc495668165)

[3.2 删除功能的实现 7](#_Toc495668166)

[3.2.1 删除功能流程图 7](#_Toc495668167)

[3.2.2 删除功能核心代码 7](#_Toc495668168)

[3.2.3 删除功能截屏示例 8](#_Toc495668169)

[3.3 查找功能的实现 10](#_Toc495668170)

[3.3.1 查找功能流程图 10](#_Toc495668171)

[3.3.2 查找功能核心代码 10](#_Toc495668172)

[3.3.3 查找功能截图示例 11](#_Toc495668173)

[3.4 修改功能的实现 13](#_Toc495668174)

[3.4.1 修改功能流程图 13](#_Toc495668175)

[3.4.2 修改功能核心代码 13](#_Toc495668176)

[3.4.3 修改功能截屏示例 14](#_Toc495668177)

[3.5 统计功能的实现 15](#_Toc495668178)

[3.5.1 统计功能流程图 15](#_Toc495668179)

[3.5.2 统计功能核心代码 15](#_Toc495668180)

[3.5.3 统计功能截屏示例 16](#_Toc495668181)

[3.6 总体系统的实现 16](#_Toc495668182)

[3.6.1 总体系统流程图 16](#_Toc495668183)

[3.6.2 总体系统核心代码 17](#_Toc495668184)

[3.6.3 总体系统截屏示例 17](#_Toc495668185)

[4 测试 19](#_Toc495668186)

[4.1 功能测试 19](#_Toc495668187)

[4.1.1 插入功能测试 19](#_Toc495668188)

[4.1.2 删除功能测试 19](#_Toc495668189)

[4.1.3 查找功能测试 20](#_Toc495668190)

[4.1.4 修改功能测试 20](#_Toc495668191)

[4.1.5 统计功能测试 21](#_Toc495668192)

[4.2 边界测试 21](#_Toc495668193)

[4.2.1 初始化无输入数据 21](#_Toc495668194)

[4.2.2 删除头结点 22](#_Toc495668195)

[4.2.3 删除后链表为空 22](#_Toc495668196)

[4.3 出错测试 23](#_Toc495668197)

[4.3.1 考生人数错误 23](#_Toc495668198)

[4.3.2 操作码错误 23](#_Toc495668199)

[4.3.3 插入位置不存在 23](#_Toc495668200)

[4.3.4 删除考号不存在 24](#_Toc495668201)

[4.3.5 查找考号不存在 24](#_Toc495668202)

[4.3.6 修改考号不存在 25](#_Toc495668203)

# 1 分析

## 1.1 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。

## 1.2 功能分析

1. 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。

2. 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。如下面测试用例2中，任务<5，7>先于任务<5，8>输入，而作为关键活动输出时则次序相反。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，本题考查了AOE网络。整个工程只有一个开始点和一个完成点，可以用作工程估算，主要使用的数据结构就是图。对于图，我采用邻接矩阵的方式来存储该有向有权图，而为了处理关键事件问题，我采用一个struct存储对应的事件结点。为了寻找关键事件，需要查询某事件的最早及最晚开始时间，而其中最早开始时间是事件的前置活动的最长所需时间，也就是用广度搜索该事件结点，获得最大权值的路径。因此，我们需要用到queue来进行广度搜索。

## 2.2 流程设计

首先按照题干要求我们会得到用户要输入的信息，接下来使用邻接矩阵及结点结构生成对应的图，接下来采用类似广度搜索的方式使用队列得到每个事件的最早及最晚开始事件，对于两个相邻关键事件之间的活动，通过公式判断其最早和最迟开始事件是否相等，进而判断其是否为关键活动，并按照题目要求次序输出。

# 3 实现

## 3.1 初始化

### 3.1.1 初始化核心代码

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

const int MAX = 2147483647;

struct node {

int early; //最早可以开始的时间

int late; //最晚必须开始的时间

int inDegree; //入度

int outDegree; //出度

vector<int> next;//后继结点

vector<int> pre; //前驱结点

node(int e, int l, int p, int n) ://构造函数

early(e), late(l), inDegree(0), outDegree(0) {}

};

int main() {

int N, M;

cin >> N >> M;

vector<vector<int> > edges(N + 1, vector<int>(N + 1, -1));

vector<node> G(N + 1, node(0, MAX, 0, 0));//early=0,late=MAX

int a, b, c;

while (M > 0) {

cin >> a >> b >> c;

edges[a][b] = c;

G[a].next.push\_back(b);

G[b].pre.push\_back(a);

G[b].inDegree++; //后继结点入度加1

G[a].outDegree++; //前驱结点出度加1

M--;

}

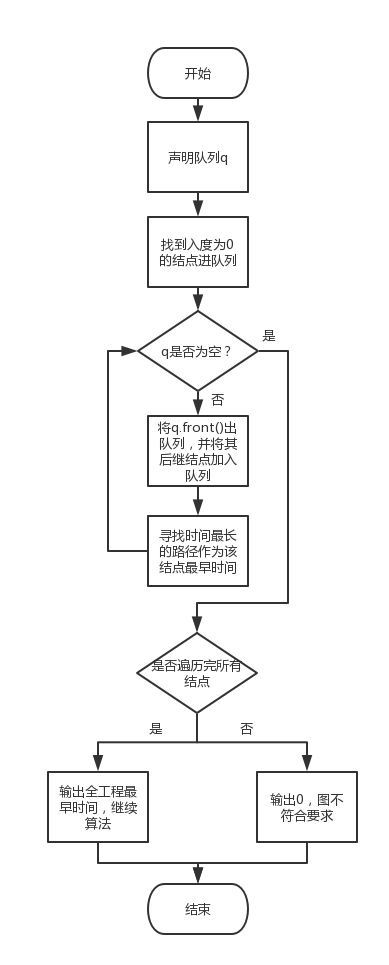
…

### 3.1.2 初始化说明

Node是为了存储事件，也就是图中的结点，每个结点记录它的入度，出度，前继结点数组以及后续结点数组，再声明两个变量储存事件的最早和最迟开始时间。用a，b，c接收用户的输入，并将邻接矩阵的对应位置赋予对应的权值。而其余位置则赋值为-1。

## 3.2 计算事件最早开始时间

### 3.2.1流程图



### 3.2.2核心代码

//从前往后计算每个结点最早可以开始的时间

queue<int> q;

int count = 0; //记录入队的点的个数，若最后不是N个，说明无解

for (int i = 1; i <= N; i++) {

if (G[i].inDegree == 0) {

G[i].early = 0;//开始时间为0

q.push(i); //入度为0的结点进入队列

count++;

}

}

while (!q.empty()) {

int cur = q.front();//当前结点序号

q.pop();

//每个结点(\*it)最早可以开始的时间受制于若干条前驱路径中

//花费时间最长的一条

for (auto it = G[cur].next.begin(); it != G[cur].next.end(); it++) {

if (G[cur].early + edges[cur][\*it] > G[\*it].early)

G[\*it].early = G[cur].early + edges[cur][\*it];//更新

if (--G[\*it].inDegree == 0) {

q.push(\*it);

count++;

}

}

}

if (count != N) {

cout << 0 << endl;

system("pause");

return 0;

}

int completeTime = -1;//完成工程需要花费的总时间

for (int i = 1; i <= N; i++) {

if (G[i].early>completeTime)

completeTime = G[i].early;

}

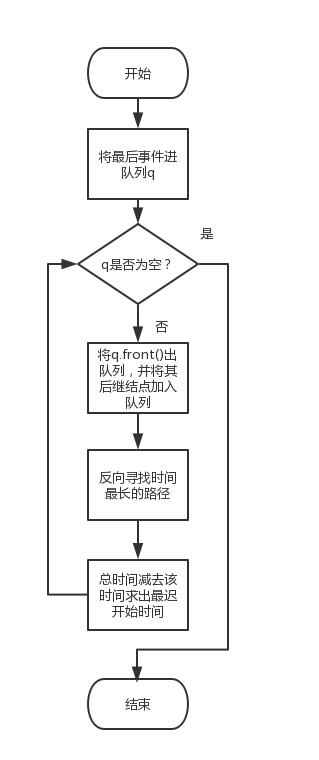
cout << completeTime << endl;

### 3.2.3 说明

首先，该查找最早开始时间的算法与广度优先搜索算法类似，采用一个队列，将入度为0的结点压入队列，再查找它的后继结点，将它们压入队列的同时让当前结点出队列，并且将其每一个后继结点的最早开始时间通过循环比较出来，直到最后一个结点，这时候所有结点应全部遍历完成，每一个节点的early应该已经被求出。如果有结点没有被遍历过，说明该图存在环或其他违法结构，无法成功遍历。输出0，如果成功求出各结点最早开始时间，那么其中最大的就是工程总时间。

## 3.3 计算事件最迟开始时间

### 3.3.1流程图



### 3.3.2核心代码

//从后往前计算每个结点最晚必须开始的时间

for (int i = 1; i <= N; i++) {

if (G[i].outDegree == 0) { //最后的结点

G[i].late = completeTime; //最晚必须开始的时间

q.push(i);

}

}

while (!q.empty()) {

int cur = q.front();

q.pop();

//每个结点(\*it)最晚必须开始的时间受制于若干条后继路径中

//花费时间最长的一条，因为该路径把deadline往前推得越多

for (auto it = G[cur].pre.begin(); it != G[cur].pre.end(); it++) {

if (G[\*it].late > G[cur].late - edges[\*it][cur])

G[\*it].late = G[cur].late - edges[\*it][cur];

if (--G[\*it].outDegree == 0)

q.push(\*it);

}

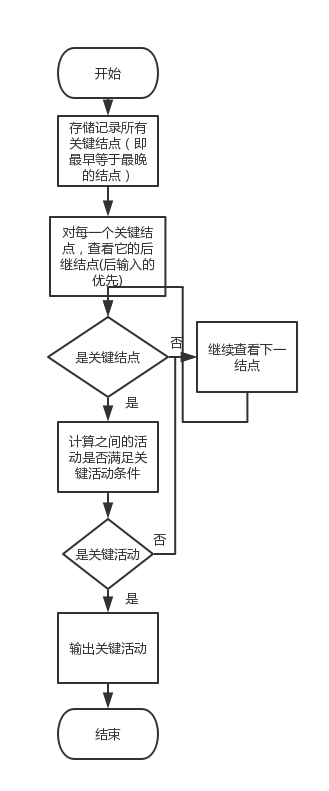
}

### 3.3.3 说明

具体过程就是反向的最早时间，用总时间减去剩余时间的最长路径时间所得就是各个事件的最晚开始时间。这样我们就有了关键事件的判断依据，也便于之后关键活动的查找算法执行。

## 3.4 查找关键活动

### 3.4.1流程图



### 3.4.2核心代码

vector<int> result;//关键事件

vector<bool> inResult(N + 1, false);

for (int i = 1; i <= N; i++) {

if (G[i].early == G[i].late) {

result.push\_back(i);

inResult[i] = true;

}

}

for (auto it = result.begin(); it != result.end(); ++it) {

for (auto itt = G[\*it].next.rbegin(); itt != G[\*it].next.rend(); ++itt) {

if (inResult[\*itt] && (edges[\*it][\*itt] == G[\*itt].early - G[\*it].early))//关键活动

cout << \*it << "->" << \*itt << endl;

}

}

### 3.4.3 说明

由于关键活动的判断依准是，其最早开始时间等于其最迟开始时间，而其最早开始时间可表示为其前相连事件的最早开始时间，最迟开始时间为其后相连事件的最迟开始时间，因此，我们可以写出edges[\*it][\*itt] == G[\*itt].early - G[\*it].early这个表达式，同时，由于关键活动的两边必定为关键事件，所以我们可以通过inResult[\*itt]来提高比较的效率。

比较的具体方法为，首先我们把所有关键事件存入数组，并且在一个布尔数组中用1标注出所有关键事件，这样我们依次对每一个数组中的关键事件进行运算，看它后继结点在布尔数组中是否为真，这样就可以找出两个相邻关键事件，由于题目中要求，我们优先对后输入的后继结点进行分析。找出可能为关键活动的活动后，我们使用关键活动的判定方法判断其是否确为关键活动，这样就按题目要求顺序输出了所有的关键活动。

# 4 测试

## 4.1 功能测试

### 4.1.1 简单功能测试

**测试用例**：

输入：

7 8

1 2 4

1 3 3

2 4 5

3 4 3

4 5 2

4 6 6

5 7 5

6 7 2

**预期结果**：

输出：

17

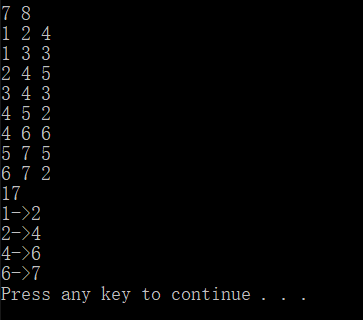
1 –>2

2 –>4

4 –>6

6 –>7

**实验结果**



### 4.1.2 一般情况功能测试

**测试用例：**

输入：

9 11

1 2 6

1 3 4

1 4 5

2 5 1

3 5 1

4 6 2

5 7 9

5 8 7

6 8 4

7 9 2

8 9 4

**预期结果：**

输出：

18

1 –>2

2 –>5

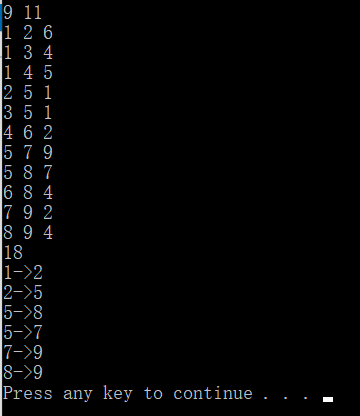
5 –>8

5 –>7

7 –>9

8 –>9

**实验结果：**



## 4.2 边界测试

### 4.2.1不可行的方案测试

**测试用例：**

输入：

4 5

1 2 4

2 3 5

3 4 6

4 2 3

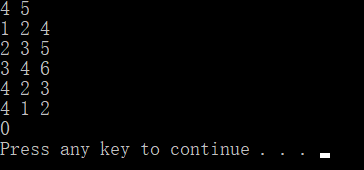
4 1 2

**预期结果：**

输出：

0

**实验结果：**



## 4.3 出错测试

### 4.3.1 输入的数据并非正整数

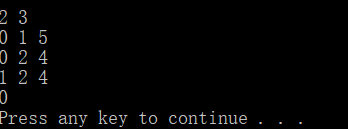
**测试用例：**

输入的结点从0开始

**预期结果：**

输出0表示没有可行方案

**实验结果：**



### 4.3.2 输入的数据出现越界结点

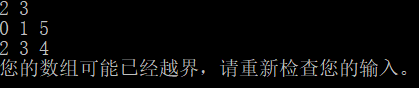
**测试用例：**

输入的结点中越界

**预期结果：**

输出提示语句，结束程序

**实验结果：**



### 4.3.3 输入的数据出现明显错误

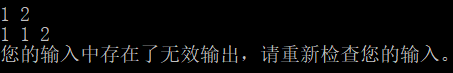
**测试用例：**

输入的数据中有如 0 0 2 这种明显错误

**预期结果：**

输出提示语句，结束程序

**实验结果：**



# 5. 亮点

## 5.1 算法为了满足题目要求做的改进

在算法之中，我们采用了广度优先遍历的类似方法找到了结点的最早及最迟开始时间，并以此推断出了关键事件，由此可以计算出关键活动。而题目中要求在首结点相同时按照输入时的次序倒序输出，所以我在算法中使用rbegin() rend()，将整个后继结点的数组倒序遍历以满足要求，详细请看3.4

## 5.2 出错测试

if (a > N || b > N)

{

cerr << "您的数组可能已经越界，请重新检查您的输入。";

system("pause");

return 0;

}

if (a == b)

{

cerr << "您的输入中存在了无效输出，请重新检查您的输入。";

system("pause");

return 0;

}

我在输入语句之间加入这两段代码，避免了程序直接触发断点或者出现问题的情况，使程序的运行更加安全，稳定，详情请见4.3