项目说明文档

数据结构课程设计

——关键活动

作 者 姓 名： 杨滕超

学 号： 2151298

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 项目概述](#_Toc495668153) 4

[1.1 项目背景](#_Toc495668154) 4

[1.2 项目功能](#_Toc495668155) 4

[1.3 项目分析](#_Toc495668154) 4

[2 项目设计](#_Toc495668156) 4

[2.1 总体设计](#_Toc495668158) 4

[2.2 类设计](#_Toc495668158) 5

[2.2.1 边结点类](#_Toc495668163) 5

[2.2.2 Vector类](#_Toc495668163) 5

[2.2.3 Vector游标类](#_Toc495668163) 6

[2.2.4 Queue类](#_Toc495668163) 7

[2.2.5 List类](#_Toc495668163) 8

[2.2.6 关键活动类](#_Toc495668163) 8

[3 项目实现](#_Toc495668161) 10

[3.1 总体实现](#_Toc495668162) 10

[3.1.1 总体实现流程图](#_Toc495668167) 10

[3.1.2 总体实现思路](#_Toc495668167) 10

[3.1.3 总体实现主要代码](#_Toc495668167) 11

[3.2 输入活动边实现 1](#_Toc495668166)3

[3.2.1 输入活动边实现流程图 1](#_Toc495668167)3

[3.2.2 输入活动边实现思路](#_Toc495668167) 13

[3.2.3 输入活动边实现主要代码](#_Toc495668168) 13

[3.3 寻找关键活动实现](#_Toc495668162) 15

[3.3.1 寻找关键活动实现流程图](#_Toc495668167) 15

[3.3.2 寻找关键活动实现思路](#_Toc495668167) 15

[3.3.3 寻找关键活动实现主要代码](#_Toc495668167) 16

[4 项目测试](#_Toc495668161) 18

[4.1 简单情况输入](#_Toc495668174) 18

[4.2 作为关键活动输入顺序不同](#_Toc495668174) 19

[4.2.1 任务<5, 7>先于任务<5, 8>输入](#_Toc495668167) 19

[4.2.2 任务<5, 8>先于任务<5, 7>输入](#_Toc495668167) 19

[4.3 图未连通](#_Toc495668174) 20

[4.4 存在环](#_Toc495668174) 20

[4.5 没有活动](#_Toc495668174) 20

[4.6 只有一个活动](#_Toc495668174) 21

[4.7 输入非法字符](#_Toc495668174) 21

[4.8 输入重复边](#_Toc495668174) 21

[4.9 输入边数量不足以构成连通图](#_Toc495668174) 21

[4.10 仅有一个任务交接点，无法添加活动](#_Toc495668174) 22

[4.11 输入任务交接点数量过多](#_Toc495668174) 22

[4.12 输入边时任务交接点相同](#_Toc495668174) 22

[4.13 输入边时不存在的顶点](#_Toc495668174) 22

1 项目概述

* 1. 项目背景

由一系列活动构成一个完整的项目，在现实生活中非常常见，例如施工流程、进程管理等等。其中，关键活动是为了准时完成项目而必须完成的活动，即处于关键路径上的活动。从而我们可以找出哪些活动需要优先完成，哪些活动可以缓一缓。在计算机中，我们可以将其抽象为一个网络，利用边和边的权值代表活动以及活动所花的时间，利用顶点表示活动交接点，如何寻找关键路径成为本项目的关键。

本项目与现实中所遇到的问题紧密联系，源于生活而高于生活，解决此类问题具有重要意义，有助于解决现实中施工流程、进程管理等调度问题，在管理方面提供极大的便利，极大地解放了人类关于该方面的工作量。

* 1. 项目功能

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。

* 1. 项目分析

1. 对于本项目的高效性，我们采用邻接表的存储方式，考虑到现实生活中对于图的建立往往是边数远小于顶点平方的数量，因此对于邻接表操作更具有高效性。并且我们需要注意的是，本项目需要存储的是一张有向图。
2. 对于用户的输入具有充分的提示，提示用户如何使用本程序；对于用户的错误输入，程序能够识别并输入错误提示，请求用户重新输入，而非崩溃不能运行，说明本项目具有较好的健壮性。
3. 功能完善，本项目基本实现题目所要求的所有功能，包括但不限于寻找关键路径、按要求输出等等。
4. 项目设计
   1. 总体设计

本项目本质上是图论中一个典型的AOE(关键活动)问题，考虑到种种因素影响，本项目决定利用邻接表存储图，利用图的拓扑排序进行关键路径的寻找。

* 1. 类设计
     1. 边结点设计

对于邻接表的边结点，我们需要在其中存储以下信息：一条弧中终点的编号、该边的权值以及指向下一个边结点的指针。

主要代码如下：

struct edge {

//指向的位置，权值大小

int to, val;

//指向下一个结点

edge\* next;

//初始化构造函数

edge(const int& to, const int& val, edge\* next = NULL) :

to(to), val(val), next(next) {}

};

2.2.2 Vector类

向量作为存储同一种类型数据的一维数组，相当于是数组的扩展，在其基础上增加方便程序员工作的操作。其中的优点在于：对于查找某个位置的元素的时间复杂度为O(1)，但是它的缺点也十分明显：向量的删除与插入的时间代价是巨大的，时间复杂度基本是O(n)级别；但特殊的是，在最后插入元素时，时间复杂度为O(1)。

我们设计Vector类的同时，同样设计了它的迭代器。

主要成员函数如下：

·//构造函数

Vector();

Vector(int size);

Vector(const Vector<T>& V);

Vector(int size, const T& val);

·//清空Vector

void clear();

·//最后添加元素

void emplace\_back(const T& x);

·//最后删除元素

void pop\_back();

·//返回最后元素

const T back() const ;

T& back() ;

·//返回开头元素

const T front() const ;

T& front() ;

·//插入

void insert(const Vector<T>::vector\_iterator& it, const T& x);

·//删除

void remove(const Vector<T>::vector\_iterator& it);

·//迭代器begin

inline Vector<T>::vector\_iterator begin() ;

·//迭代器end

inline Vector<T>::vector\_iterator end() ;

·//返回迭代器的查找

typename Vector<T>::vector\_iterator find(const T& x);

·//重载[]

T& operator[](const int index) ;

·//重载=

Vector<T>& operator=(const Vector<T>& V);

·//重新设置大小

void resize(int size);

void resize(int size, const T& val);

2.2.3 Vector游标类

其中数据成员包含了指向Vector中数组元素的指针，并实现的重载\*、->、++、--、<、>等操作符。

主要函数如下：

·//重载\*

T& operator\*();

·//重载->

T\* operator->();

·//重载==

inline bool operator==(const vector\_iterator& it) const ;

·//重载!=

inline bool operator!=(const vector\_iterator& it) const;

·//重载不等号

inline bool operator<(const vector\_iterator& it) const;

inline bool operator>(const vector\_iterator& it) const ;

inline bool operator<=(const vector\_iterator& it) const;

inline bool operator>=(const vector\_iterator& it) const ;

·//重载自加

inline Vector<T>::vector\_iterator& operator++();

·//重载自减

inline Vector<T>::vector\_iterator& operator--();

·//后置自加

Vector<T>::vector\_iterator operator++(int);

·//后置自减

Vector<T>::vector\_iterator operator--(int);

·//与数字相加

friend Vector<T>::vector\_iterator operator+(const vector\_iterator& it,int num);

·//与数字相减

friend Vector<T>::vector\_iterator operator-(const vector\_iterator& it,int num);

·//自加

Vector<T>::vector\_iterator operator+=(int num);

·//自减

Vector<T>::vector\_iterator operator-=(int num);

·//两个相减

friend int operator-(const vector\_iterator& it1, const vector\_iterator& it2);

2.2.4 Queue类

队列和栈一样，同样是一种运算受限的线性表，它允许的操作是在表的前端(front)进行删除，表的后端(rear)进行插入，遵循先进先出的原则。其中的前端称为队头，后端称为队尾。队列的实现同样分为两种方式，顺序队列和链式队列，对于顺序队列，为了实现空间的循环利用，采用循环队列的方式，其中的要点在于队头指针和队尾指针往后移动增加1后对最大空间大小取模。而对于链式队列，其优点在于没有空间的限制，可以灵活地申请空间。

基于上述考虑，我们采用以链表为基础的队列。考虑到链表无论是在头或尾的插入删除结点的时间复杂度均为O(1)，因此我们将链表头作为队列头，链表为作为队列尾。

该类主要成员函数如下：

//判断队列是否为空

inline bool empty() const {

return data.empty();

}

//获得队列中元素个数

inline int size() const {

return data.Size;

}

//清空队列

void clear() {

data.clear();

}

//在队列为插入元素

void emplace(const T& x) {

data.emplace\_back(x);

}

//在队列头删除元素

void pop() {

data.pop\_front();

}

//查询队列头的元素值

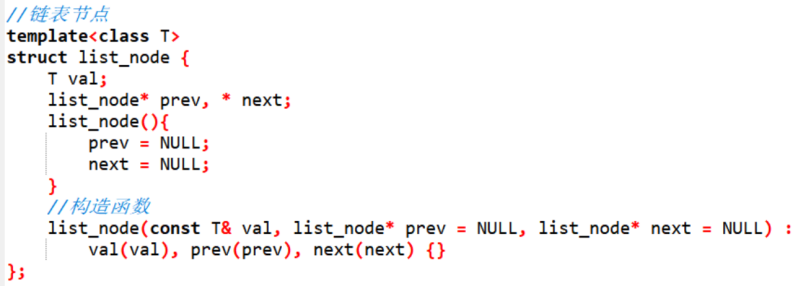
T& front() {

return data.getHead()->next->val;

}

2.2.5 List类

·链表结点类设计：其中包括需要储存的数据，前驱节点和后继节点。如下图所示：



·链表类设计：数据成员包括，头结点，尾结点以及记录节点个数数据。

主要功能(与本题有关)包括：

·//返回链表大小inline int size() const;

·//判断链表是否为空inline bool empty() const;

·//清空链表void clear();

**·//最后添加元素void emplace\_back(const T& x)；**

**·//最后删除元素void pop\_back();**

**·//开头添加元素void emplace\_front(const T& x);**

**·//开头删除元素void pop\_front();**

·//插入void insert(const List<T>::list\_iterator& it, const T& x);

·//删除void remove(const List<T>::list\_iterator& it);

·//返回迭代器的查找typename List<T>::list\_iterator find(const T& x);

·//迭代器begin inline List<T>::list\_iterator begin();

·//迭代器end inline List<T>::list\_iterator end();

2.2.6 关键活动类

该类中存储用户输入的图的信息，以及有关的成员函数。数据成员包括：邻接表（边结点指针数组）、入度数组、出度数组、顶点个数、边个数。

主要设计如下：

class criticalActivity {

private:

//同一起点的边在同一个桶，一个图，一个反图

Vector<edge\*> graph1, graph2;

//记录入度

Vector<int> in;

//记录出度

Vector<int> out;

//N为顶点个数，M为边的个数

int N = 0, M = 0;

public:

//寻找关键活动函数

void AOE();

//得到N, M值

void getNM(const int& N, const int& M);

//初始化

void init();

//记录边

void setEdge(int from, int to, int val);

//检查是否输入的边是否合法

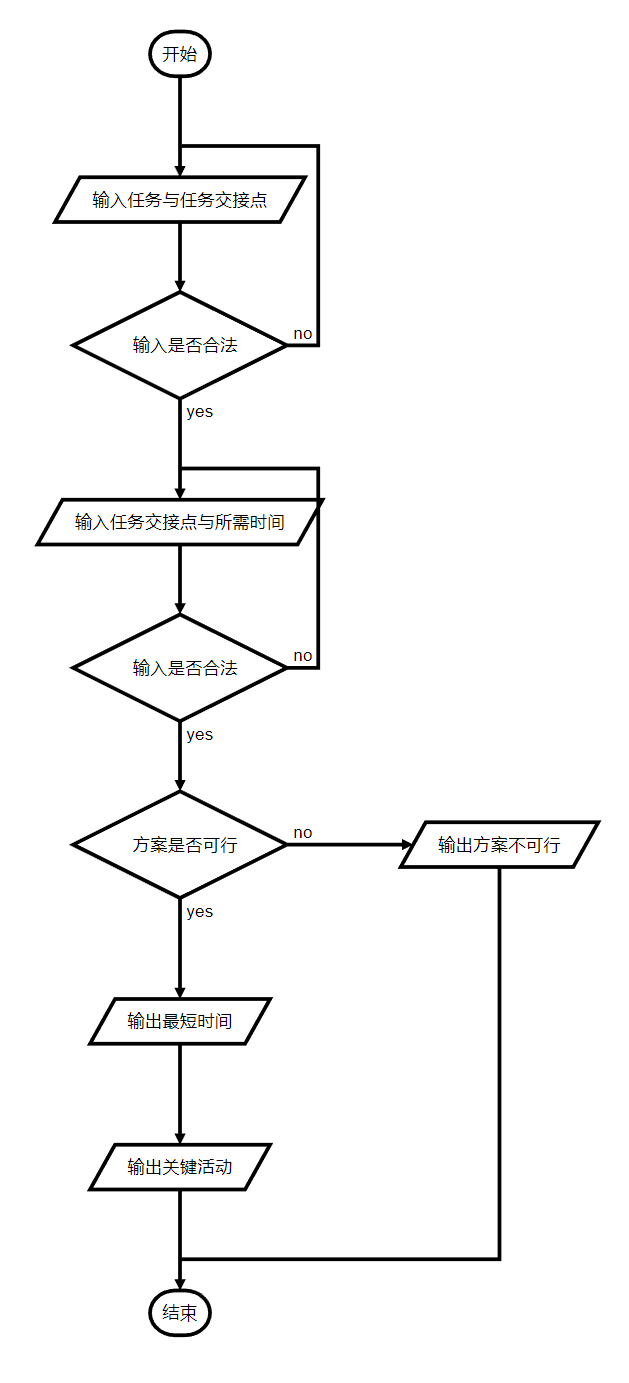
bool checkValid();

//析构函数释放结点空间

~criticalActivity();

};

1. 项目实现
   1. 总体实现
      1. 总体实现流程图



* + 1. 总体实现思路

首先请用户输入任务交接点个数以及任务个数，此时需要判断是否输入正确，若不正确则需要输出输入错误提示，并请求用户重新输入。

接着请用户输入任务交接点对以及任务所需要的时间，这时候需要判断交接点是否合法，即是否存在，并且两个交接点不能相同，若输入错误，程序给出提示，并请求用户重新输入。

然后检查所输入的边是否将所有任务交接点连通，若没有连通则说明输入方案不可行，直接输出相关提示，若可行则调用求关键活动函数。

* + 1. 总体实现主要代码

cout << "请输入任务交接点个数N(N <= 100), 任务个数M: ";

while (1){

cin >> tempN >> tempM;

if (cin.fail() || tempN <= 0 || tempM <= 0){

cout << "输入有误，请输入正整数" << endl;

cin.clear();

cin.ignore(65536, '\n');

}

else if (tempN > 100){

cout << "任务交接点个数过多，注意N<=100" << endl;

cin.clear();

cin.ignore(65536, '\n');

}

else if (tempN == 1)

{

cout << "仅有一个任务交接点无法存在活动" << endl;

cin.clear();

cin.ignore(65536, '\n');

}

else if (tempM < tempN)

{

cout << "活动数过少，无法连通" << endl;

cin.clear();

cin.ignore(65536, '\n');

}

else{

ca.getNM(tempN, tempM);//将顶点个数，边个数传给对象

break;

}

}

//存图初始化

ca.init();

cout << "请依次输入任务交接点以及任务所需时间: " << endl;

for (int i = 0; i < tempM;){

int from, to, val;

cin >> from >> to >> val;

if (cin.fail() || from <= 0 || to <= 0 || val <= 0){

cin.clear();

cin.ignore(65536, '\n');

cout << "输入有误，请输入正整数" << endl;

}

else if (from > tempN || to > tempN){

cin.clear();

cin.ignore(65536, '\n');

cout << "无此任务交接点，请重新输入" << endl;

}

else if (from == to){

cin.clear();

cin.ignore(65536, '\n');

cout << "输入有误，两个任务交接点不应相同" << endl;

}

else{

ca.setEdge(from - 1, to - 1, val);

++i;//正确输入才下一个

}

}

//检查是否合法

if (ca.checkValid())

ca.AOE();

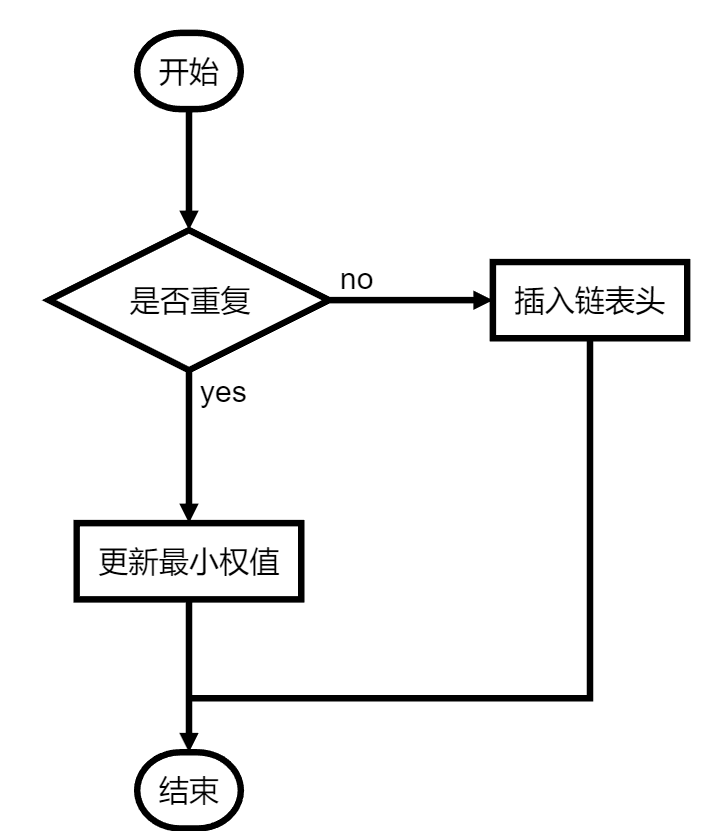
else{

cout << "此方案不可行" << endl;

cout << '0' << endl;

}

* 1. 输入活动边实现
     1. 输入活动边实现流程图



* + 1. 输入活动边实现思路

对于边的加入我们需要考虑几个问题：其一是判断当前输入的边是否与之前所输入的边重复，应该如何检查，若重复了应该如何取舍；其二是若没有重复，根据要求应该如何插入。

本项目的实现中，首先由输入的始点找到对应的链表，在链表中寻找是否存在与当前输入相同的终点。若没有找到则在该链表的头插入，否则更新相应边的权值，取最小值。

* + 1. 输入活动边实现主要代码

edge\* cur1 = graph1[from], \* cur2 = graph2[to];

while (cur1){//如果有东西

if (cur1->to == to)//发现是相同的边就跳出

break;

cur1 = cur1->next;//没发现相同的边就继续往后找

}

if (cur1){//找到相同的边，比较权值大小

if (cur1->val > val);//更新更小{

//此时需要更新，再去更新反图

cur1->val = val;

//反向图的搜索，此时一定存在

while (cur2){

if (cur2->to == from)

break;

cur2 = cur2->next;

}

if(cur2)

cur2->val = val;

//更新完反图直接退出

return;

}

//不用更新也直接退出

return;

}

//没找到相同的边，插入到第一个

edge\* newNode = new edge(to, val);

//插入

newNode->next = graph1[from];

graph1[from] = newNode

//没找到相同的边，插入到第一个

newNode = new edge(from, val);

//插入

newNode->next = graph2[to];

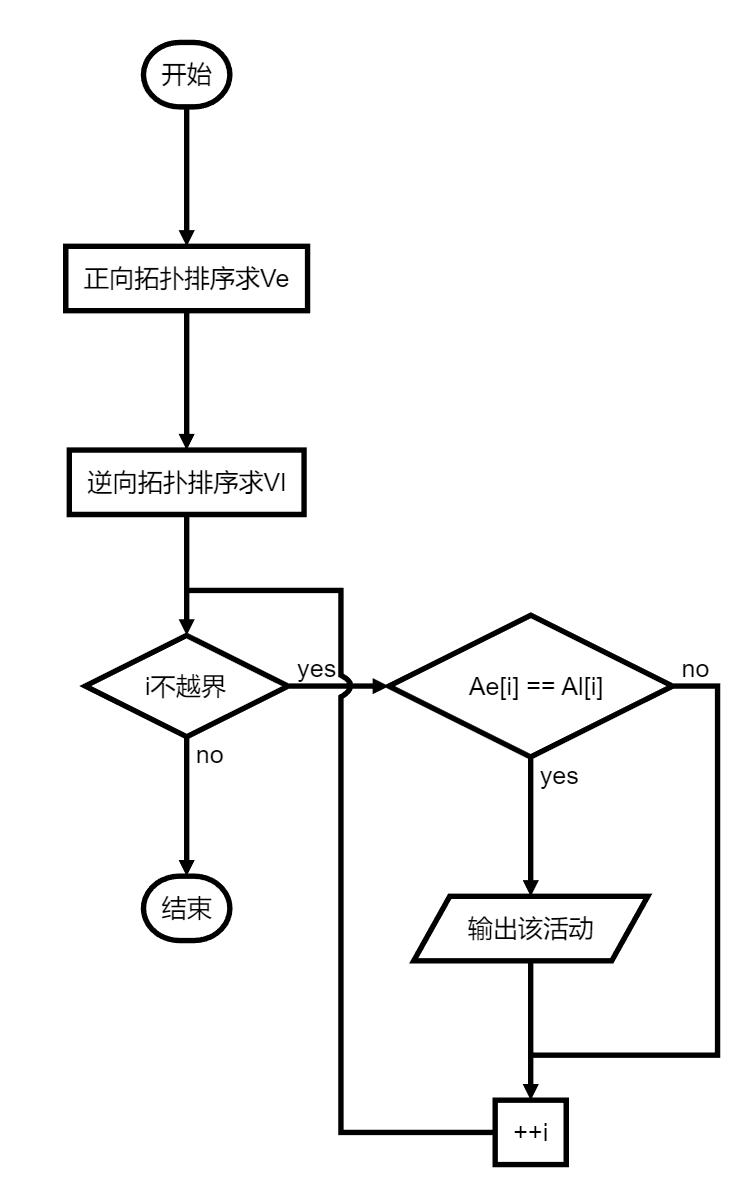
graph2[to] = newNode;

//新添加边，没有重复边，才更新出度和入度

++out[from];//记录出度

++in[to];//记录入度

* 1. 寻找关键活动实现
     1. 寻找关键活动实现流程图



* + 1. 寻找关键活动实现思路

所谓关键活动的求取，就是对所得有向图进行两次拓扑排序，一次正向拓扑排序，一次逆向拓扑排序。通过向此所求得得交接点最早开始得时间和最晚允许开始得时间，以此求得任务最在开始的时间与最晚允许开始的时间。对于任务来说，若两个时间相同，则说明该任务为关键活动。

需要注意的一点是，拓扑排序的同时我们也可以判断该图是否存在环，在第一次拓扑排序完成的时候，存在没有遍历到的任务交接点，则说明图中存在环，说明该输入方案不可行。

进行拓扑排序，我们记录每个顶点的入度或出度，但对于两次正向与逆向的拓扑排序，我们不仅仅需要记录每个顶点的入度与出度，在有向图中，我们还需要邻接表与逆邻接表，由此才能叫快速地查询到以某个任务交接点为始点的所有终点，以及以某个任务交接点为终点的所有始点。

* + 1. 寻找关键活动实现主要代码

Queue<int> q;

//各个顶点最早开始和最晚开始时间

Vector<int> Ve(N, 0), Vl(N, INT\_MAX);

//记录已经能遍历的，来判断有没有环

Vector<bool> visited(N, false);

//入度为零入队，最早开始时间记录为零

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

if (in[i] == 0)

q.emplace(i);

if (q.empty())//甚至没有一个顶点有为零，方案不可行

{

cout << "此方案不可行" << endl;

cout << '0' << endl;

return;

}

int temp = 0;

edge\* tempEdge = NULL;

while (!q.empty())

{

temp = q.front();

q.pop();

visited[temp] = true;//标记遍历过了

tempEdge = graph1[temp];

while (tempEdge)//遍历能从temp指向的顶点

{

//graph[temp]记录了以temp为起点的边

int to = tempEdge->to;

int val = tempEdge->val;

//更新最大值

if (Ve[temp] + val > Ve[to])

Ve[to] = Ve[temp] + val;

//入度为零就入队

if (--in[to] == 0)

q.emplace(to);

//移动到下一个指向的顶点

tempEdge = tempEdge->next;

}

}

//跳出循环队列为空

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

{

if (out[i] == 0)

{

q.emplace(i);

Vl[i] = Ve[i];

}

if (visited[i] == false)//之前居然没有遍历，说明存在环

{

cout << "此方案不可行" << endl;

cout << '0' << endl;

return;

}

}

//开始逆向拓扑排序

while (!q.empty())

{

temp = q.front();

q.pop();

tempEdge = graph2[temp];

while (tempEdge)

{

//graph[temp]记录了以temp为起点的边

int to = tempEdge->to;

int val = tempEdge->val;

//更新最小

if (Vl[temp] - val < Vl[to])

Vl[to] = Vl[temp] - val;

//出度为零就入队

if (--out[to] == 0)

q.emplace(to);

tempEdge = tempEdge->next;

}

}

cout << "\n完成项目最少时间: ";

cout << Ve[N - 1] << endl;

cout << "关键活动为: \n";

//遍历图中的边，符合要求就记录下来

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

{

if (graph1[i])

{

edge\* cur = graph1[i];

while (cur)

{

int to = cur->to;

//最早开始和最晚开始的时间相同即为关键活动

if (Ve[i] == Vl[i] && Ve[to] == Vl[to])

cout << i + 1 << "->" << to + 1 << endl;

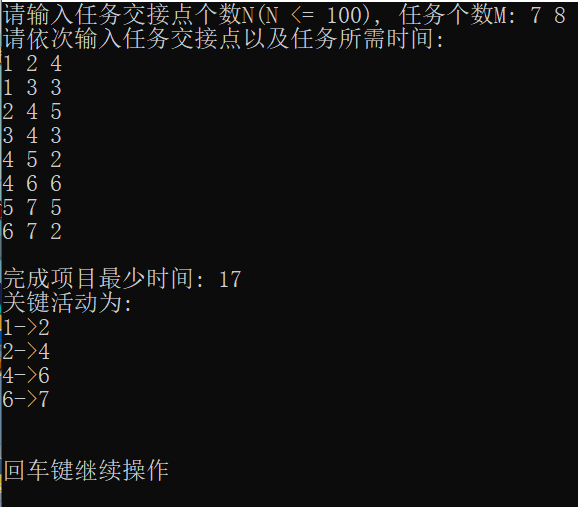
cur = cur->next;

}

}

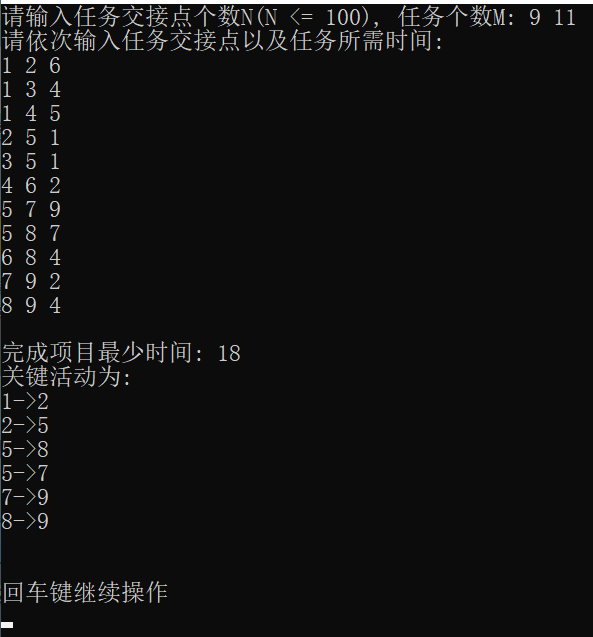
}

1. 项目测试
   1. 简单情况输入

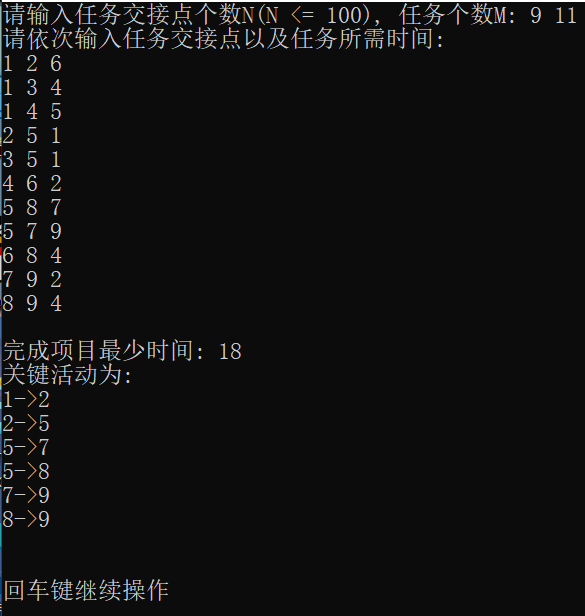


* 1. 作为关键活动输入顺序不同

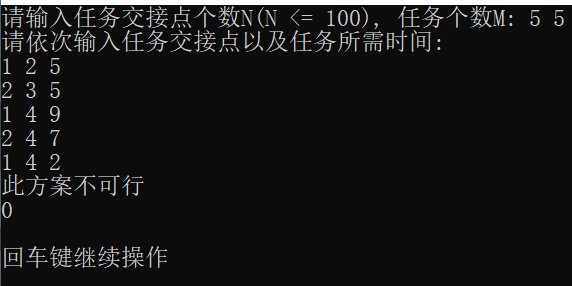
4.2.1 任务<5，7>先于任务<5，8>输入



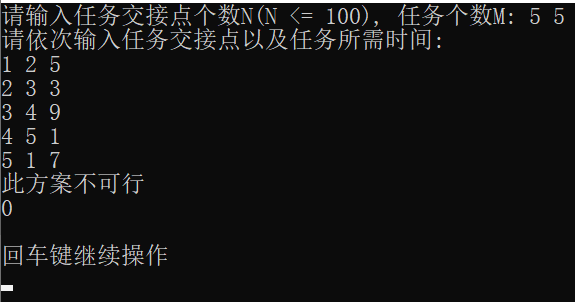
4.2.2 任务<5，8>先于任务<5，7>输入



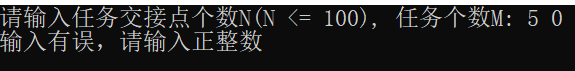
* 1. 图未连通



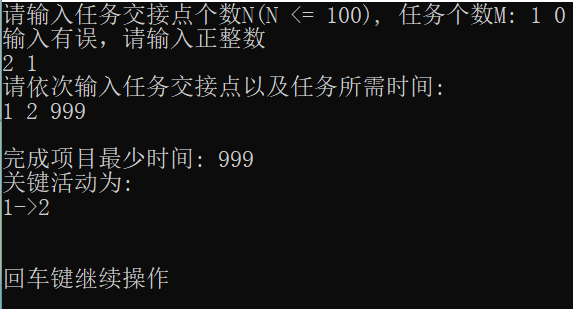
* 1. 存在环



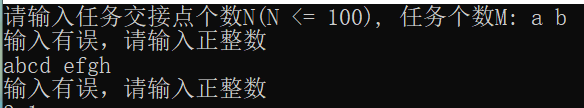
* 1. 没有活动



* 1. 只有一个活动

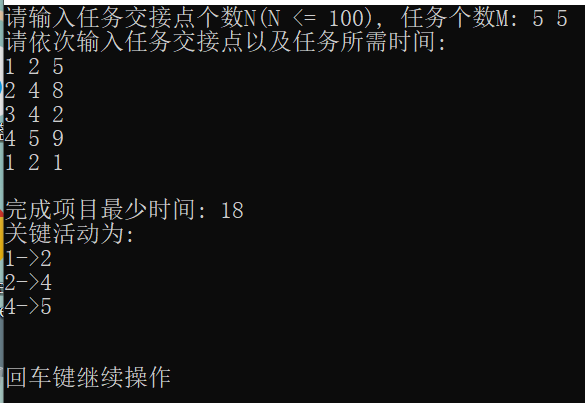


* 1. 输入非法字符



* 1. 输入重复边

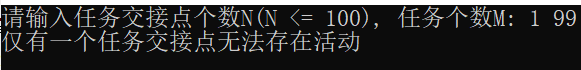
希望取最小值，符合预期



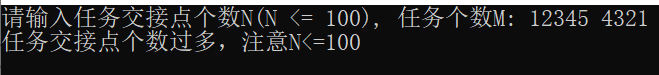
* 1. 输入边数量不足以构成连通图



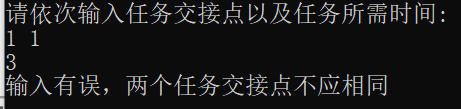
* 1. 仅有一个任务交接点，无法添加活动



* 1. 输入任务交接点数量过多



* 1. 输入边时任务交接点相同



* 1. 输入边时输入不存在的顶点

