**数据结构课程设计**

**项目说明文档**

**关键活动**

作 者 姓 名： 苏家铭

学 号： 2151299

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 项目分析 1](#_Toc3811)

[1.1 项目背景 1](#_Toc17484)

[1.2项目要求 1](#_Toc22213)

[1.2.1 功能要求 1](#_Toc4972)

[1.2.2 输入要求 1](#_Toc7349)

[1.2.3 输出要求 2](#_Toc12491)

[1.2.4 项目实例 2](#_Toc23462)

[2 项目设计及实现 3](#_Toc239)

[2.1 数据结构设计思路 3](#_Toc19961)

[2.2 类设计 3](#_Toc25820)

[2.2.1 链表 3](#_Toc18468)

[2.2.2 队列 5](#_Toc168)

[2.2.3 图 5](#_Toc24946)

[2.3 项目算法 8](#_Toc19391)

[2.3.1 实现思路 8](#_Toc9161)

[2.3.2 代码实现 10](#_Toc24587)

[3 项目测试 11](#_Toc665)

[3.1 简单情况测试 11](#_Toc26527)

[3.2 一般情况测试，单个起点和单个终点 12](#_Toc6763)

[3.3 不可行的方案测试 13](#_Toc17569)

[3.4 其中一个序列完全属于交集的情况 14](#_Toc1437)

[3.5 输入N或M非法（健壮性） 14](#_Toc11645)

[4 算法性能分析 15](#_Toc14419)

[4.1 正确性 15](#_Toc1955)

[4.2 可使用性 15](#_Toc9385)

[4.3 可读性 15](#_Toc23914)

[4.4 效率 15](#_Toc12703)

[4.5 健壮性 15](#_Toc28615)

# 1 项目分析

## 项目背景

如果在无有向环的有向带权图中用有向边表示一个工程中的各项活动，用有向边上的权值表示活动的持续时间，用顶点表示时间，则这样的有向图叫做用边表示活动的网络(activity on edges,AOE网络)。

AOE网络在某些工程估算方面非常有用，例如，它可以使人们了解：

1. 完成整个工程至少需要多少时间（假设网络中没有环）？
2. 为缩短完成工程所需的世界，应当加快哪些活动？

在AOE网络中，有些活动可以并行地进行。从源点到各个顶点，以至于从原点到回电的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路基的活动所需的世界去接于从源点到汇点的最长路径长度，即在这条路经上所有活动的持续时间之和。这条路径长度最长的路径称之为——关键路径(critical path)。

## 1.2项目要求

### 1.2.1 功能要求

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。

### 1.2.2 输入要求

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。

### 1.2.3 **输出要求**

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。

### 1.2.4 项目实例

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 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 | 17  1 –>2  2 –>4  4 –>6  6 –>7 | 简单情况测试 |
| 2 | 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 | 18  1 –>2  2 –>5  5 –>8  5 –>7  7 –>9  8 –>9 | 一般情况测试，单个起点和单个终点 |
| 3 | 4 5  1 2 4  2 3 5  3 4 6  4 2 3  4 1 2 | 0 | 不可行的方案测试 |

# 2 **项目设计及实现**

## 2.1 **数据结构设计思路**

活动之间的关系是有若干个顶点和有向线段组成的结构，应当采用图作为存储结构，为了计算关键路径时，个各个活动间存在前后关系，需要经过**拓扑排序**（Topo Sort）后才能够求解。

所谓拓扑排序就是将有向无环图(Graph)G中所有顶点排成一个线性序列，该序列满足：若在图G中由顶点vi到顶点 vj有一条路径，则在该线性序列中的顶点vi必定在顶点vj之前。

为了完成拓扑排序，需要借助队列（Queue）来辅助实现。队列中存储当前入度为0的顶点，从队列中出队并且将该元素加入线性序列中。

## 2.2 **类设计**

### 2.2.1 链表

虽然带头结点的双向循环链表结构复杂，但使用代码后会发现许多优势。该链表一般包括两个抽象数据类型（ADT）：链表结点类（DblNode）与链表类（DblList），而两个类之间的耦合关系可以采用嵌套、复合等多种关系。若结点类定义为class那么需要在类内定义友元函数链表类才能访问，为了实现代码的复用性，本人使用结构体来定义链表的结点，此方法虽然使得DblNode类失去了封装性，但链表类可以直接访问结点中的数据成员。

节点类数据成员：

template<class T>

T data;//链表结点数据

DblNode<T>\* lLink,\*rLink;//链指针域,前驱（左链）、后继（右链）

链表类数据成员：

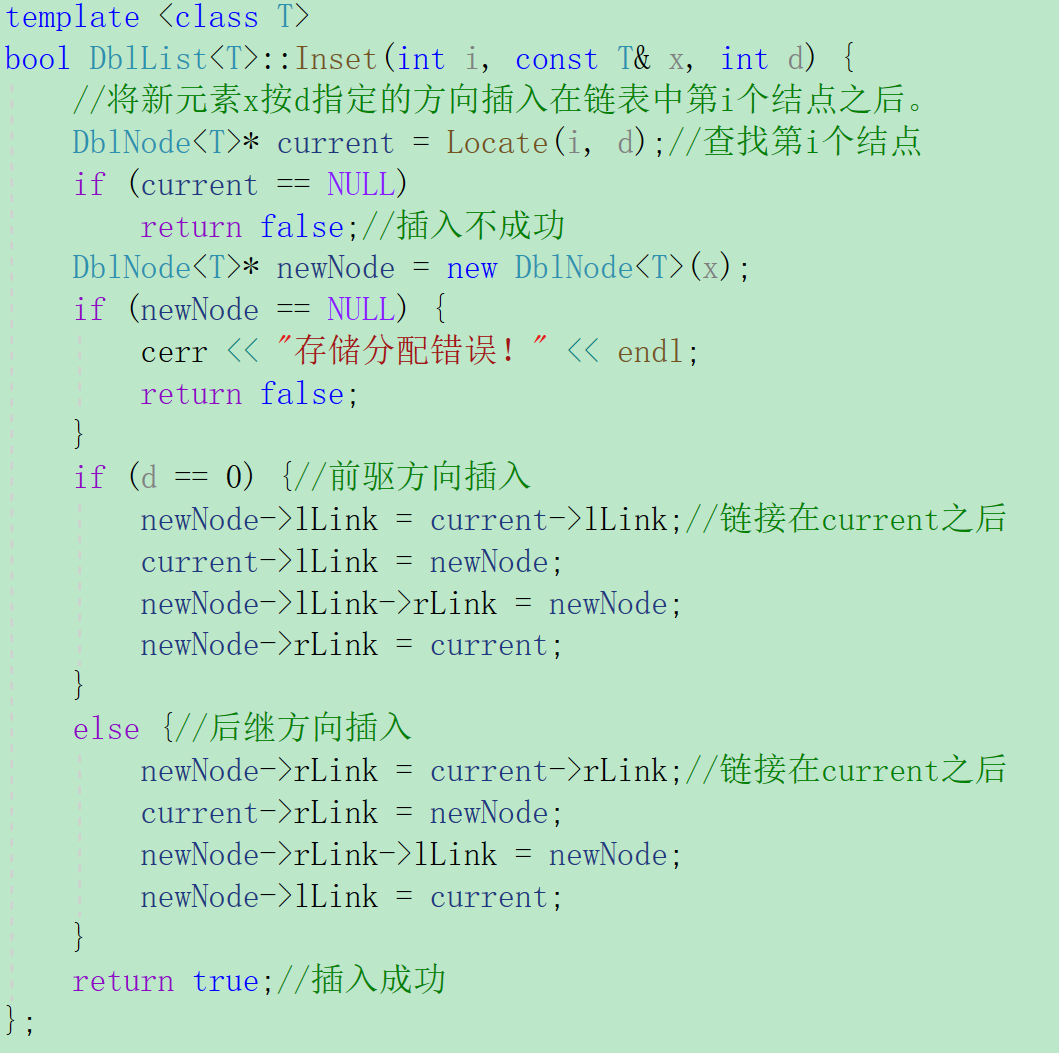
DblNode<T>\* first;//链表头结点

该链表已经实现19种功能：置空、计算长度、返回头结点、设置头结点、搜索、定位、返回元素地址、修改元素的值、插入、删除、置空、判空、判满、尾部插入、头部删除、向后插入i个元素、向后插入若干元素（直到遇到endTag）、输出链表、等号重载。

具体函数为：

* void makeEmpty();//将链表置为空表
* int Length()const;//计算双链表的长度
* void setHead(DblNode<T>\* ptr)//设置附加头结点地址
* DblNode<T>\* getHead()const//返回附加头结点地址
* DblNode<T>\* Search(const T& x);//在链表中沿后继寻找等于数据x的元素
* DblNode<T>\* Locate(int i,int d);//在链表中定位序号为i（>=0）的结点，d=0按前驱方向，d!=0按后继方向
* bool getDate(int i, T& x,int d);//取出按d方向第i个元素的地址
* void setData(int i, T& x,int d);//用x修改按d方向第i个元素的值
* bool Inset(int i, const T& x,int d);//在第i个元素后插入x,d=0按前驱方向，d!=0按后继方向
* bool Remove(int i, T& x,int d);//删除第i个元素,x返回该元素的值,d=0按前驱方向，d!=0按后继方向
* bool IsEmpty()const//判表空否?空则返回true
* bool IsFull()const {return false;}//判表满否？不满则返回false
* void push\_back(T data);//在尾部插入一个数据
* void push\_front(T data);//在前部插入一个数据
* void input\_num(int i);//输入i个元素
* void input\_endTag(T endTag);//输入若干元素，直到遇到endTag
* void output(int d);//输出
* DblList<T>& operator= (const DblList<T>&L);//重载函数：赋值

插入算法的代码实现：



主要算法的时间复杂度：

* 插入：O(n)
* 删除：O(n)
* 搜索：O(n)
* 遍历：O(n)
* 修改：O(n)

### 2.2.2 队列

队列(Queue)是只允许在一端进行插入，而在另一端进行删除的运算受限的线性表，允许插入的一端叫做队尾(back)，允许删除的一端叫做队头(front)。队列也叫做先进先出的线性表，即FIFO(First In/First Out).

而本项目基于已经实现了的双向循环链表，实现了链式队列。每次入队时在链表的末尾插入一个元素，出队时则删除链表的第一个元素。出队入队都可以通过链表的first指针的左右指针找到队首和队尾，则无需再定义last指针，也无需遍历一遍链表找队尾元素，即出队、入队的时间复杂度可以降为O(1).

队列类数据成员：

template<class T>

DblList<T> list;

该链式队列已经实现6种功能：入队、出队、置空、判空、返回队首元素、返回队列长度。

具体函数声明为：

* void push(const T& val);//入队
* void pop();//出队
* void makeEmpty();//置空
* bool empty()const;//判断空否
* int size()const;//返回队列大小
* const T& front();//返回队首元素

主要算法的时间复杂度：

* 入队：O(1)
* 出队：O(1)
* 置空：O(n)
* 判断空否：O(1)
* 返回队首元素：O(1)
* 返回队列大小：O(n)

### 2.2.3 图

图是由顶点的有穷非空集合和顶点之间的边的集合组成，通常表示成：G=(V,E),其中G表示一个图，V是图G中顶点的集合，E是图G中边的集合。

图的存储方式一般为邻接矩阵和邻接表，本项目中选择邻接表作为图的存储结构，为了往后能实现两种存储方式又不用重复定义相同成员，本项目公有继承基类GRAFH，实现了派生类grafh。

GRAFH类数据成员：

int max\_vertices\_num;//图中最大顶点数

int edges\_num;//当前图中边数

int vertices\_num;//当前图中顶点数

Grafh类数据成员：

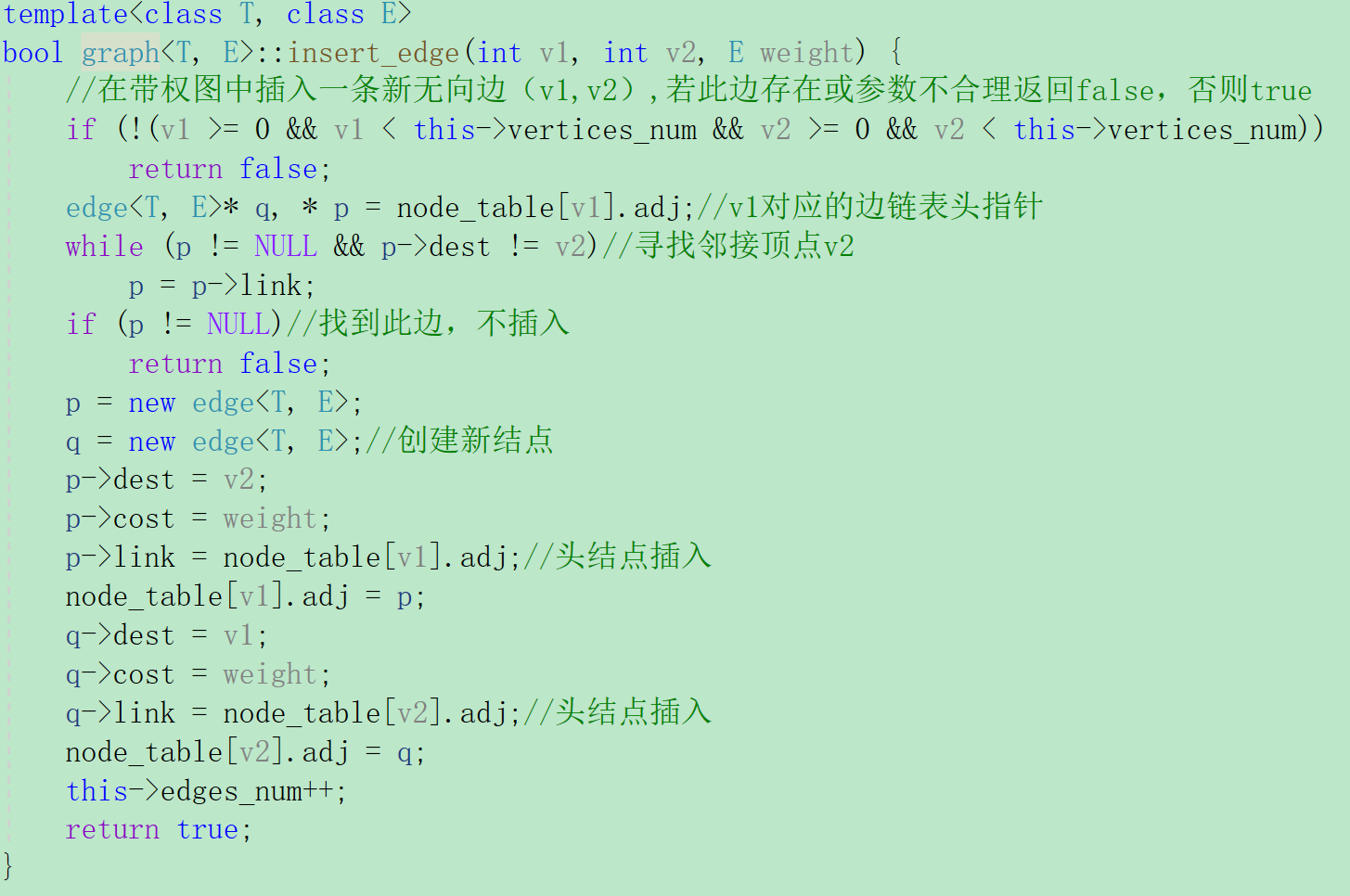
vertex<T, E>\* node\_table;//顶点表（各边链表的头结点）

与为了达到项目的要求，图实现了包括返回结点的值、返回边权、插入顶点、插入边（有向/无向）、删除顶点、删除边、左移/右移运算符重载等17种功能。

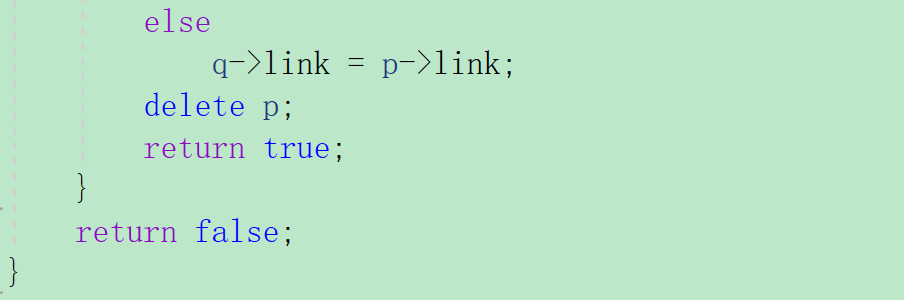
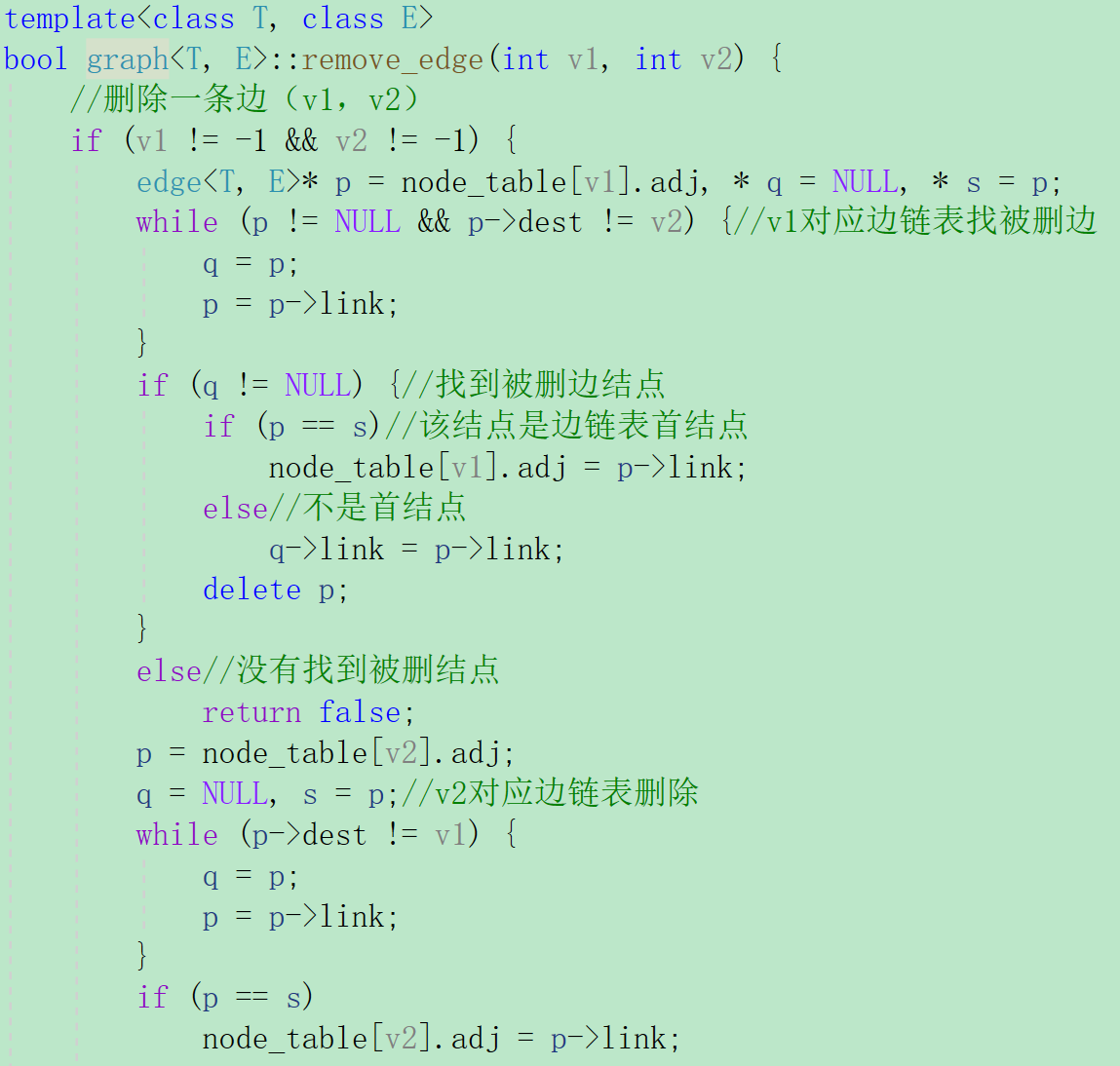
函数声明如下：

* bool empty()const;//判空
* bool full()const;//判满
* int vertices\_number(); //返回当前结点数
* int edges\_number();//返回当前边数量
* T get\_value(int i);//取位置为i的顶点中的值
* E get\_weight(int v1, int v2);//返回边（v1,v2）上的权值
* bool insert\_vertex(const T& vertex);//在图中插入一个顶点vertex
* bool remove\_vertex(int v);//在图中删除一个顶点v,此时v是顶点号
* bool insert\_edge(int v1, int v2, E cost);//在图中插入一条新无向边(v1,v2)
* bool insert\_directed\_edge(int v1, int v2, E cost);//在图中插入一条新有向边<v1,v2>
* bool remove\_edge(int v1, int v2);//在图中删除一条边(v1,v2)
* int get\_vertex\_pos(const T vertex);//返回顶点vertex在图中的位置
* int get\_first\_neighbor(int v);//取顶点v的第一个邻接顶点
* int get\_next\_neighbor(int v, int w);//取v的邻接顶点w的下一邻接顶点
* int get\_neighbor\_num(int v);//返回该点的邻接边数量
* friend istream& operator>>(istream& in,const graph& G);
* friend ostream& operator<< (ostream& out,const graph& G);

插入算法代码实现：



删除算法代码实现：



主要算法的时间复杂度：

* 插入顶点：O(n)
* 插入边：O(n+m)
* 删除顶点：O(n+m)
* 删除边：O(n+m)
* 返回顶点值：O(n)
* 返回边权：O(n+m)

其中，n为顶点数量，m为边数量

## 2.3 **项目算法**

### 2.3.1 实现思路

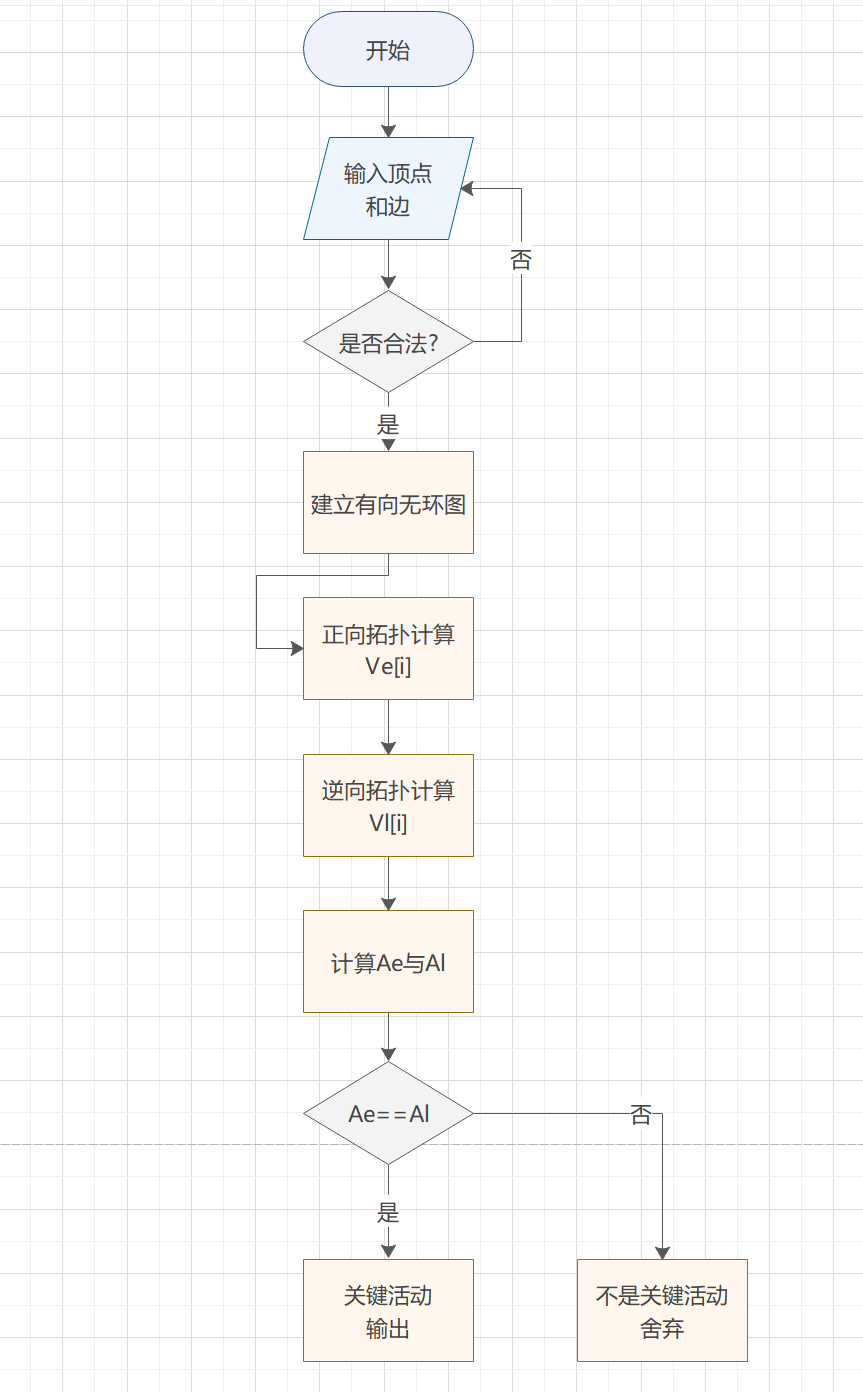
要找出关键路径，就必须找出关键活动，即不按期完成就会影响整个工程完成的活动。关键活动路径上的所有活动都是关键活动。因此，为了找到关键路径，我们首先应找出关键活动。

1. 首先从源点V0开始，令Ve[0]=0，在实现拓扑有序的过程中计算每个顶点的最早可能开始时间Ve[i],i=1,2,...,n-1.（如果拓扑排序循环次数小于n，则说明网络中存在环，无法得到线性序列）
2. 从汇点Nn-1开始，令Vl[n-1]=Ve[n-1]，逆向计算各顶点的最迟允许开始时间Vl[i],i=n-2,n-3,...,0.
3. 根据所得Ve数组和Vl数组，遍历计算每条边的最早可能开始时间Ae[k]和Al[k],k=1,2,...m.
4. 若Ae[k]==Al[k]，则该活动为关键活动。

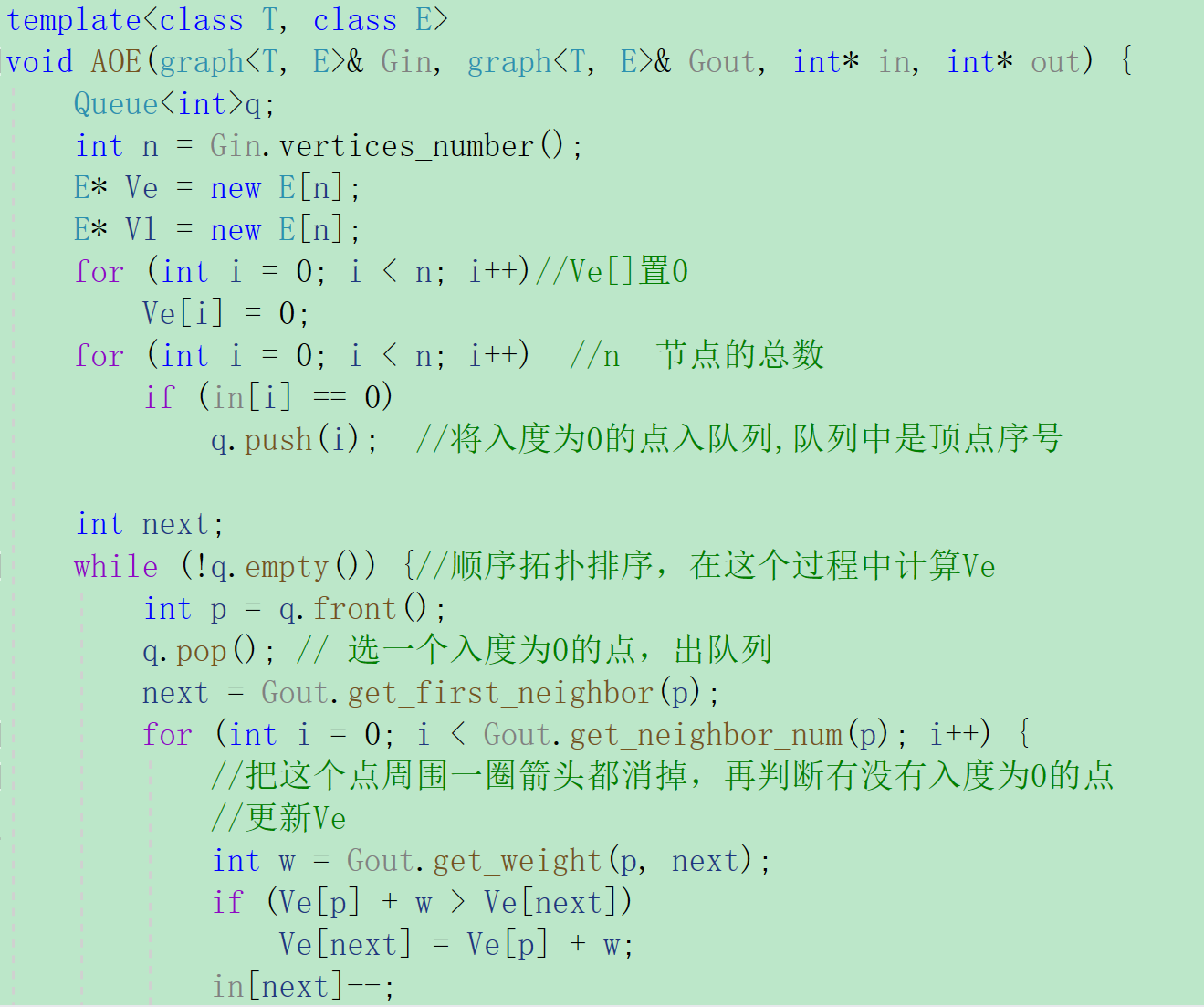
流程中四个参量的计算方法如下：

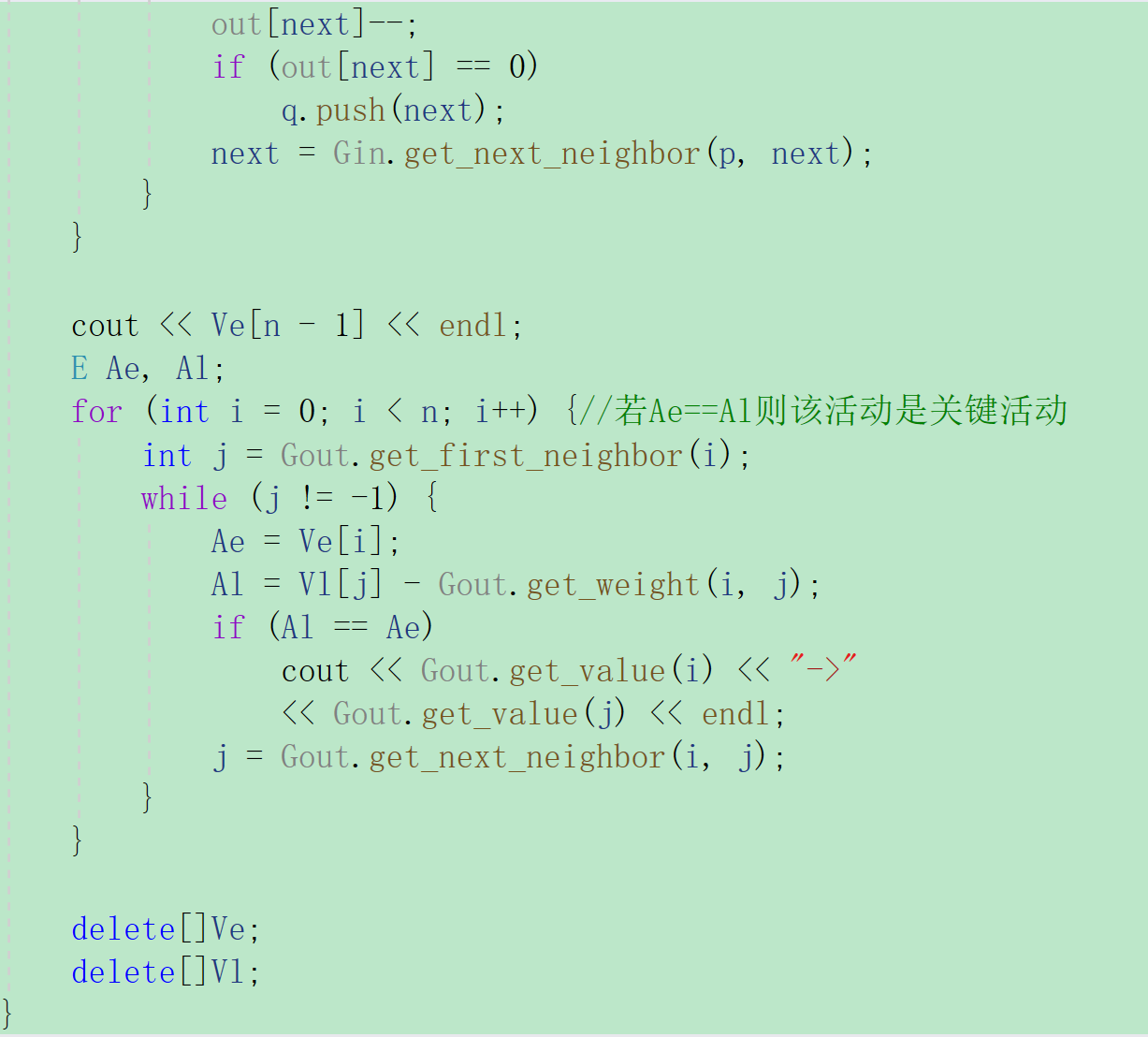
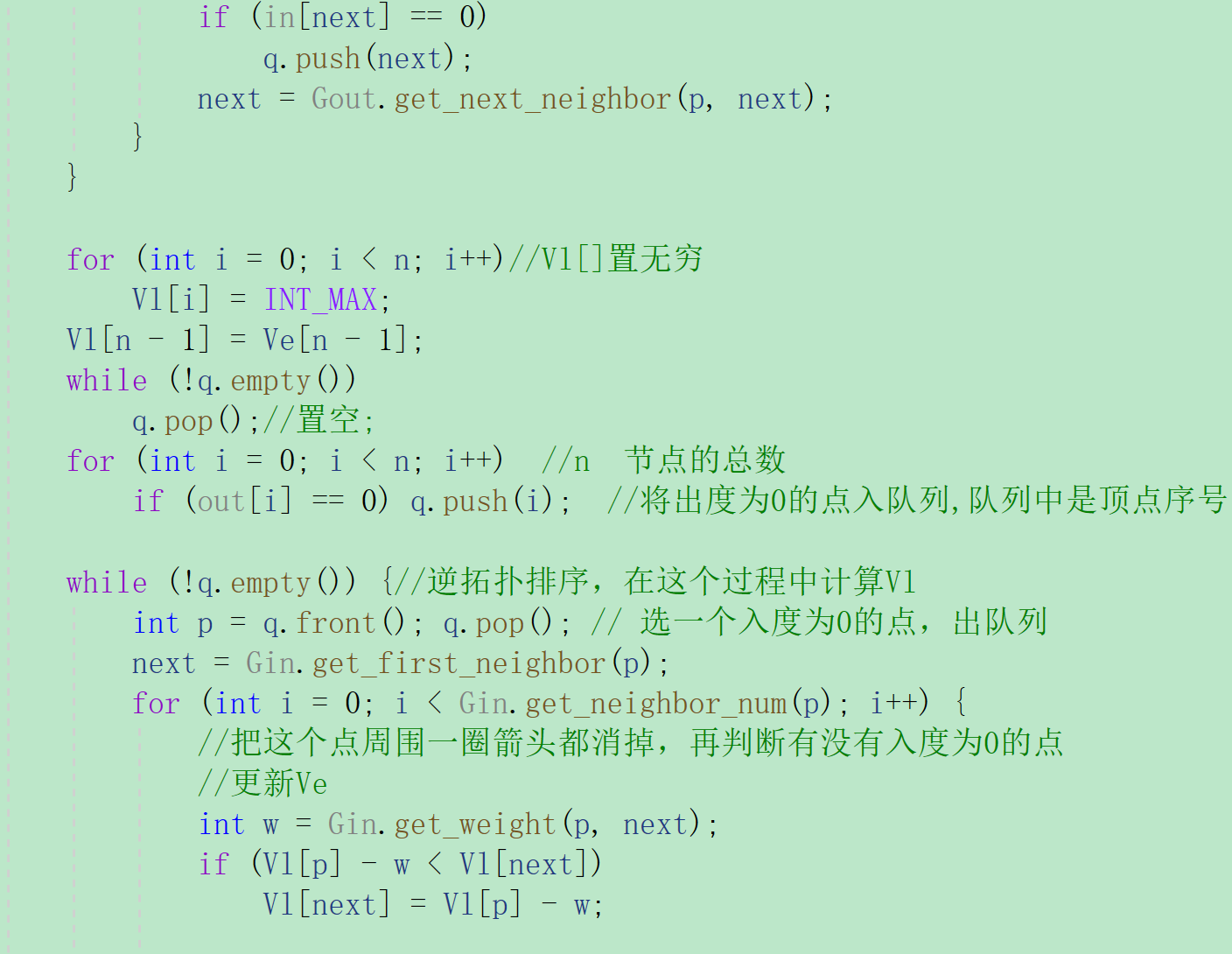
1. Ve[j]=max{Ve[i]+dur(<Vi,Vj>)},j=1,2,..,n-1;(正向计算)
2. Vl[i]=min{Vl[i]-dur(<Vi,Vj>)},i=n-2,n-3,...,0;（逆向计算）
3. Ae[k]=Ve[i];
4. Al[k]=Vl[k]-dur(<Vi,Vj>)

流程图示意如下：



### 2.3.2 代**码实**现





# 3 项目测试

## 3.1 简单情况测试

测试输入：

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

预期输出：

17

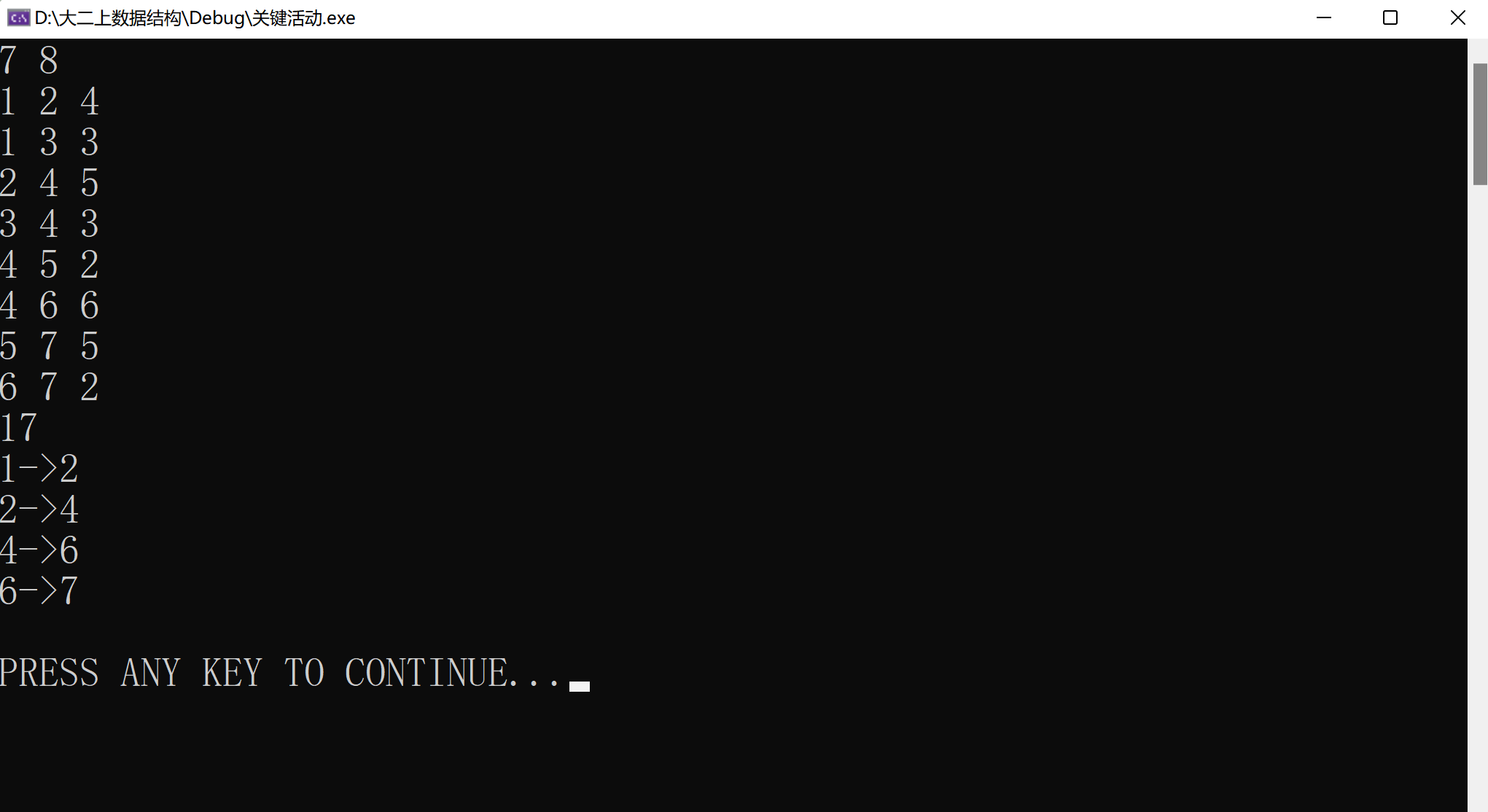
1 –>2

2 –>4

4 –>6

6 –>7

实验结果：



## 3.2 一般情况测试，单个起点和单个终点

测试输入：

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

预期输出：

18

1 –>2

2 –>5

5 –>8

5 –>7

7 –>9

8 –>9

实验结果：



## 3.3 不可行的方案测试

测试输入：

4 5

1 2 4

2 3 5

3 4 6

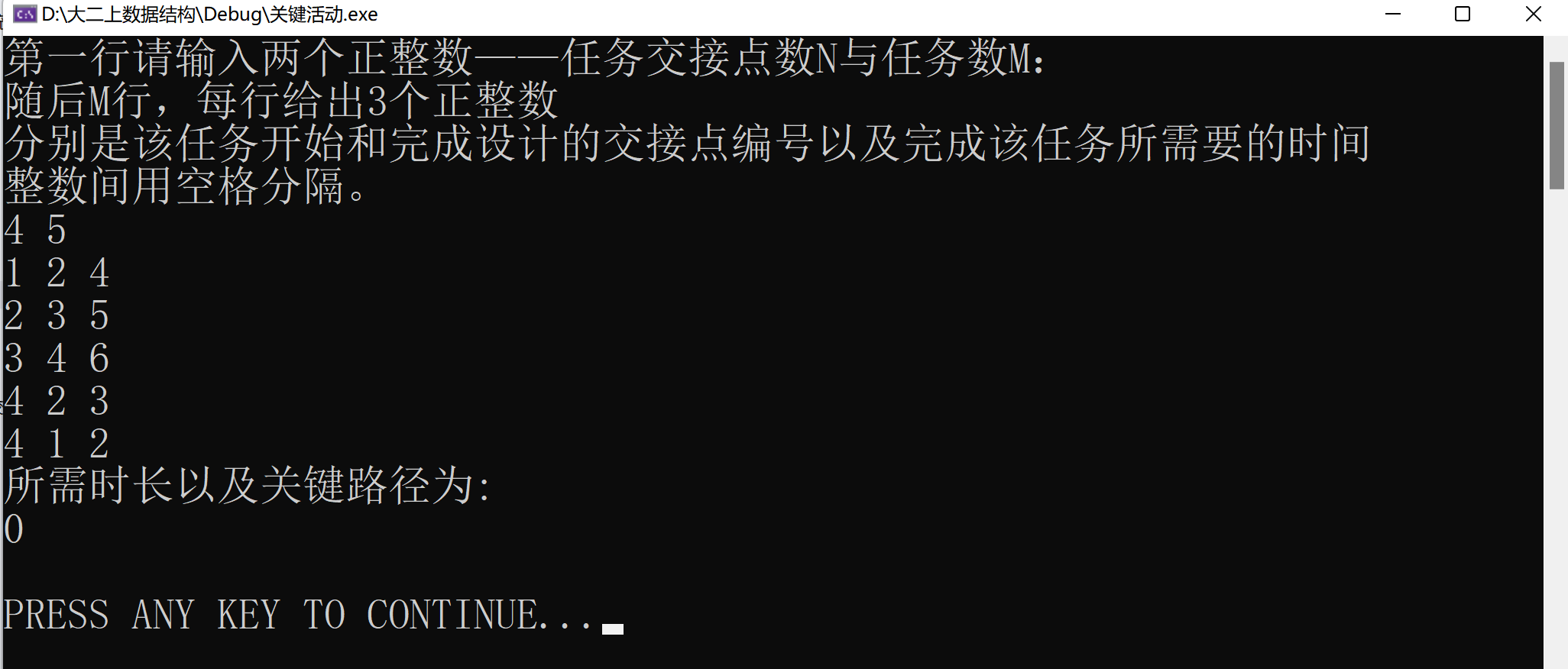
4 2 3

4 1 2

预期输出：

0

实验结果：



## 3.4 其中一个序列完全属于交集的情况

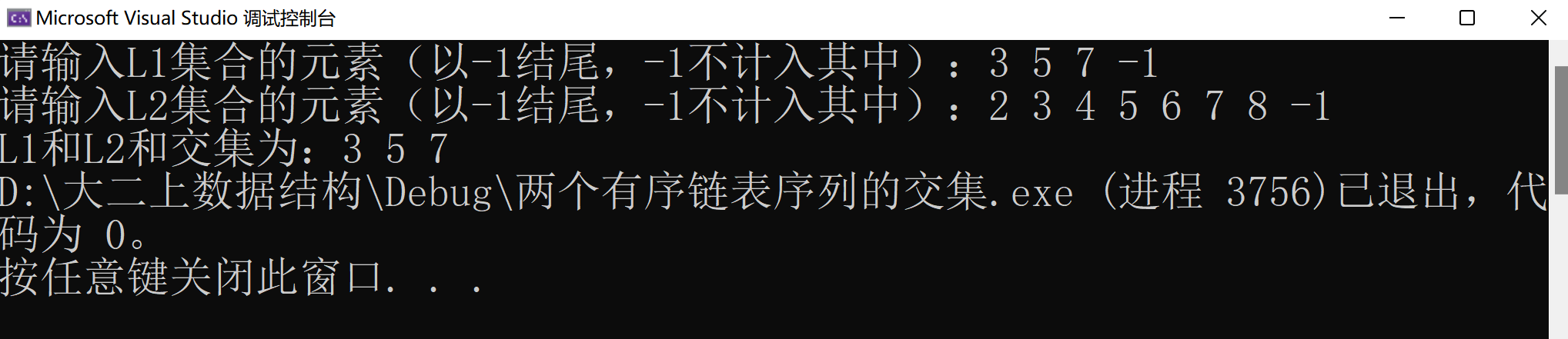
测试输入：

3 5 7 -1

2 3 4 5 6 7 8 -1

预期输出：3 5 7

实验结果：



## 3.5 输入N或M非法（健壮性）

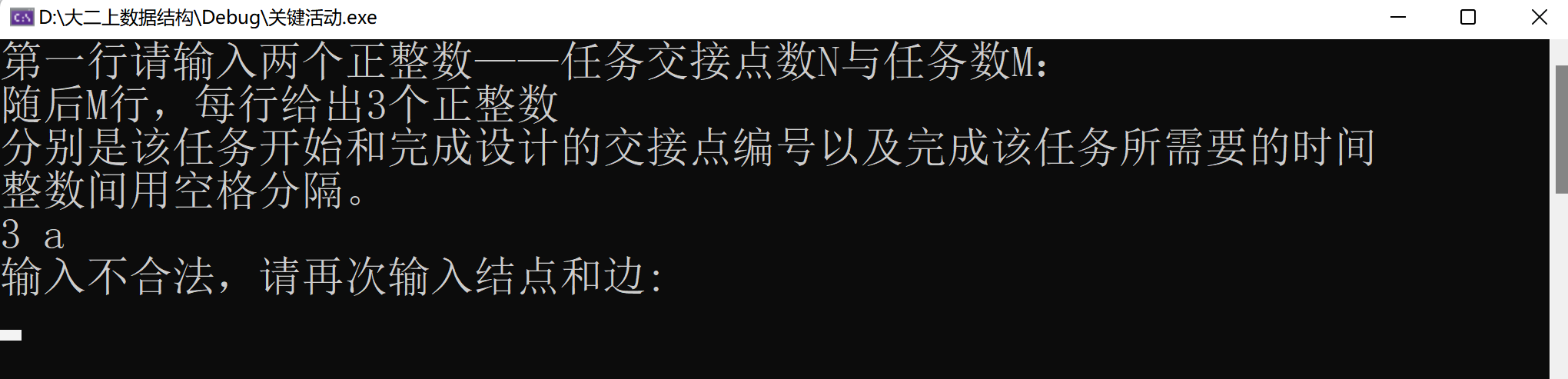
测试输入：

3 a

预期输出：

错误提示

实验结果：



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，由实验结果表明该项目与期望相吻合，满足正确性。

## 4.2 可使用性

本算法可以很方便地使用，求关键路径的功能已封装在一个函数中。并且该算法有良好的界面和完备的用户文档。没有使用公用变量或全局变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

拓扑排序的时间复杂度为O(n+e),而且邻接表对于边稀疏的有向图性能比邻接矩阵更好。

## 4.5 健壮性

本算法对于边界条件，诸如不可行的方案有相应的判断，并且对于新结点的申请失败、输入操作非法的情况也有相应的错误提示。