****

**Tongji University**

**数据结构课程设计实验报告**

**009-关键活动**

专 业： 软 件 工 程

指导教师： 张 颖

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

1. **项目概览**
   1. **项目背景**

通常把计划、施工过程、生产流程等都当作一个工程。除了很小的工程外，一般都把工程分成若干个活动（子工程）。而这些活动的完成往往具有先后顺序，即某个（或几个）工程完成后另一个工程才能开始。为了合理描述这种问题，可以创建有向图来表示这种先后关系。而为了进一步求出关键活动、关键路径和完成时间等，则选择使用构造AOE网络的方法来解决问题。

* 1. **项目要求**
     1. **项目整体要求**

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. **项目输入要求**

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，

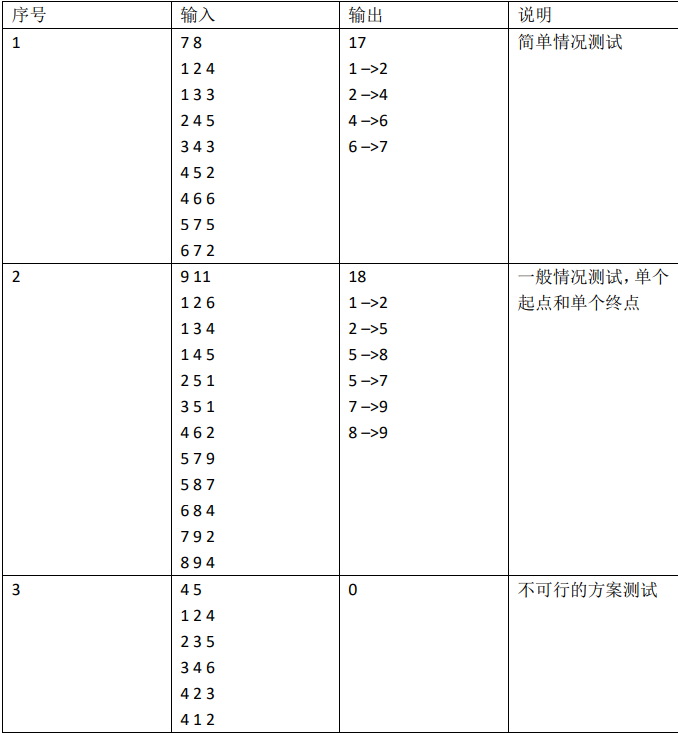
例如：若任务 2 要在任务 1 完成后才开始，则两个任务之间必有一个交接点）的数量，交接点按 1～N 编号，M 是子任务的数量，依次编号为 1～M。随后 M 行，每行给出 3 个正整数，分别是该任务开始和完成设计的交接点编号以及完成该任务所需要的时间，整数间用空格分隔。

* + 1. **项目输出要求**

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，任务先于任务输入，而作为关键活动输 出时则次序相反。

* + 1. **项目示例**



* 1. **项目需求分析**

根据项目要求可以得到，本次项目实质是将任务之间的交接点视作结点，将任务视作边，构建AOE网络。同时将任务完成时间作为边的权值存储在结构体中。

为了求关键活动和活动总时间，则需要利用拓扑排序和计算关键路径算法进行求解，最后遍历Ae数组求得最长关键路径即为完成任务所需时间。

* + 1. **项目功能完善**

工程活动往往需要考虑到多方面因素，而任务衔接也是重点之一。因此需要保证项目的功能足以实现这些目的。需要完成包括数据存储，拓扑排序，求关键活动和关键路径，计算任务所需时间等功能，使项目功能完善。

* + 1. **执行效率优化**

为了提高用户的使用体验，需要优化程序的执行效率。这一点可以从程序的空间开销和时间开销入手。本次项目为了避免递归和过多的循环，选择使用栈等方式进行优化。

* + 1. **代码的健壮性**

程序需要保证代码的健壮性。当用户进行不正确使用时，代码需要进行报错提示，避免崩溃退出。同时也要保证存储结构封装安全，数据存储严谨无遗漏。

* + 1. **关键活动**

为了方便用户查看任务的关键活动，进行进一步的决策规划，故需要将关键活动按要求打印出来，提供可视化窗口。

1. **数据结构设计与算法使用**
   1. **数据结构的使用与设计**

本次项目问题较为复杂，涉及数据结构较多。主要使用了双向链表、向量、成对和图的数据结构。

图作为主题框架支撑起实现程序功能，构造AOE网络以及进行关键运算的作用，而向量作为图的低一层数据结构负责存储各任务衔接点的入度，为拓扑排序提供条件。成对类在程序实现过程中将两个有关系的量联系起来，起到辅助作用。而双向链表则是作为程序的底层数据结构，负责数据的底层存储。

* + 1. **双向链表的设计**

本次项目为了避免占用内存过大，同时考虑到项目存在很多的数据插入和数据删除操作，故使用链表作为底层数据结构。而考虑到程序的执行效率问题，为了避免链表的多次遍历提高时间开销，故选择双向链表，通过牺牲空间复杂度来提高时间复杂度，提高程序运行效率。

本次项目链表类设计与之前项目相同，故此处不再赘述。

* + 1. **向量类的设计**

本次项目涉及到对任务衔接点进行拓扑排序，并且将排序内容运用到求关键路径算法中进行运算。因此选择向量作为数据结构。其以链表作为底层数据结构，利用向量类中的push\_back()和pop\_back()函数实现栈功能。

本次的向量类设计与之前相同，故此处不再赘述。

* + 1. **AOE网络的设计**

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（activities），用有向边上的权值表示活动的持续时间（duration），用顶点表示事件（event）,则这样的有向图叫做用边表示活动的网络（activity on edges , AOE网络）。

完成整个工程所需的时间取决于从源点到汇点的最长路径长度，即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做关键路径。关键路径上的所有活动都是关键活动。

定义顶点类和边类后，将任务及其交接点建立有向图，为了便于求关键路径，同时减少时间开销和空间开销，选择使用邻接表法进行数据存储。

在类中定义斌并实现插入点，插入边，拓扑排序和求关键路径（活动）几个主要函数，并以求邻接边等函数为辅助。

1. 在AOE网络类中定义顶点和边结构体存储数据：
2. int data;
3. edge\* adj;
4. 在vertex结构体中定义整数型变量data和边结构体型指针变量adj。
5. //边另一顶点位置

int dest;

1. //边权值

int cost;

1. //下一条边链指针

edge\* link;

在边结构体中定义整数类型变量dest指向边的另一顶点位置。定义整数类型变量cost存储边的权值，并且递归定义边结构体指针link指向下一条边链。

1. 图类的设计与使用：

数据变量的定义：

1. //顶点表（各边链表的头结点）

vertex\* nodeTable;

1. //顶点总数

int numVertices;

1. //边总数

int numEdges;

1. //顶点最多为

int maxVertices;

1. //入度表

Vector<int> inDegree;

1. //最长关键路径

int sumCost;

1. //辅助拓扑排序

int\* count;

类成员函数的定义：

1. //构造函数

graph(int sz = DefaultVertices);

~graph();

1. //根据位置取其中的顶点名

int getValue(int i)

1. //计算顶点vertex在图中的位置

int getVertexPos(const int vertex) {

1. //取边<v1,v2>权值

int getWeight(int v1, int v2);

1. //插入顶点vertex

bool insertVertex(const int& vertex);

1. //插入边<v1,v2,weight>

bool insertEdge(int v1, int v2, int weight);

1. //取顶点位置为v的第一个邻接顶点位置

int getFirstNeighbor(int v);

1. //取顶点v的邻接顶点w的下一个邻接顶点位置

int getNextNeighbor(int v, int w);

1. //计算关键路径并打印关键活动

void criticalPath();

1. //拓扑排序

void topologicalSort();

* 1. **算法功能的实现**
     1. **拓扑排序算法的实现**

检测有向环是对AOE网络构造拓扑有序序列。即将各个顶点排列成一个线性有序序列，是所有应存在的前驱和后继关系都能得到满足。构造全部定点的拓扑有序序列的运算就叫拓扑排序。

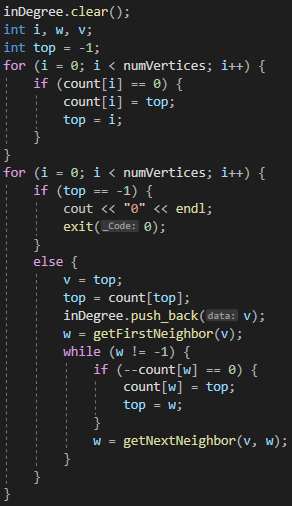
进行拓扑排序的步骤如下：

1. 输入/读取网络，设置n为顶点个数；
2. 在网络中选一个没有直接前驱的顶点并输出它；
3. 从图中删去该顶点，同时删去所有它发出的有向边。
4. 重复（2）（3）步，直到：
5. 全部定点均已输出，拓扑有序序列形成，拓扑排序完成；
6. 图中还有为输出的顶点，但已经跳出处理循环；
7. 后者说明网络中存在有向环。

本次实验不需要输出拓扑有序序列，而是需要将拓扑排序作为计算关键路径算法的辅助算法进行实现，其主要目的是构建AOE网络的入度表，实现计算关键路径时逆拓扑有序计算Al的操作。

当拓扑有序算法检测到图中有有向环时，说明任务不合理，输出0；

具体算法代码如下：



* + 1. **计算关键路径算法的实现**

关键活动是关键路径上的所有活动。因此找到了关键活动当，就可以找到关键路径。

1. 事件最早可能开始时间Ve是源点到顶点的最长路径长度；
2. 事件最迟允许开始时间Vl是保证汇点n-1准时完成的情况下，事件允许的最迟开始时间。其树枝上等于Ve[n-1]减去Vi到Vn-1的最长路径；
3. 活动最早可能开始时间Ae=Ve；
4. 活动最迟允许开始时间Al[k]=Vl[j]-dur（<i,j>）；
5. 计算出以上四个量后，就可以利用判断是否满足Ae=Al来判断活动是否为没有时间余量的关键活动。

通过定义可以得到Ae和Ve分别在拓扑有序和逆拓扑有序条件下的递推公式。通过相同的思路可以建立起计算关键路径的算法：

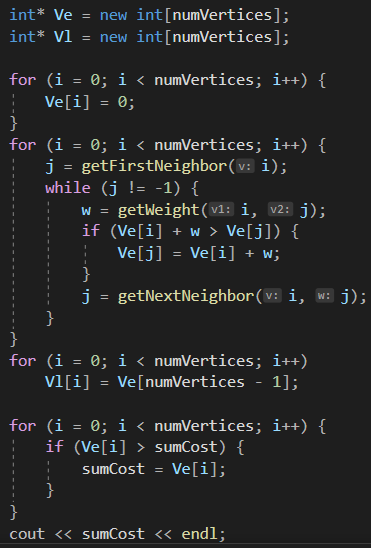
1. 输入/读取AOE网络，建立邻接表结构；
2. 从源点出发，令Ve[0]=0，按拓扑有序的顺序计算每个顶点的Ve。若拓扑排序的循环次数小于顶点数，则说明网络中有有向环，无法求关键路径。
3. 从汇点出发，令Vl=Ve[n-1]，按逆拓扑有序求各顶点的Vl；
4. 根据各顶点的Vl和Ve值，求各有向边的Ae和Al；
5. Ae=Al的活动即为关键活动，进行输出。

本次算法中若能够建立关键路径，则遍历Ve数组寻找其中最大值，这一最大值即为活动总体的所需时间。

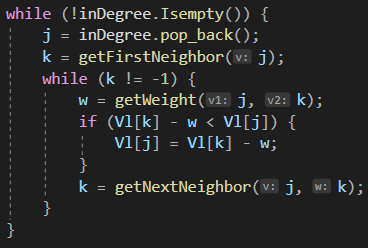
具体代码实现如下：

1. 按拓扑有序求Ve数组，并且遍历Ve求活动需要时间；

将Vl数组赋初值为后续逆拓扑有序求值做铺垫。



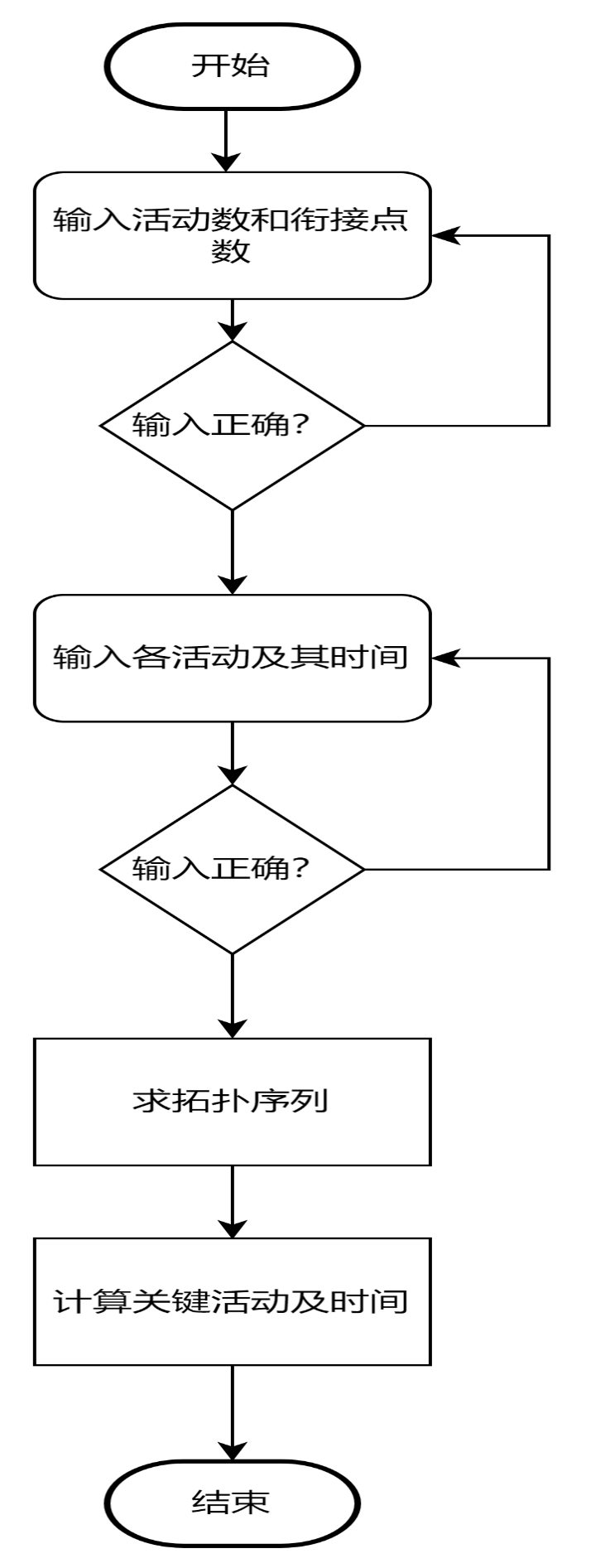
1. 逆拓扑有序求Vl数组：



1. 求关键活动并输出：



1. **程序流程图解**
   1. **程序整体流程图**



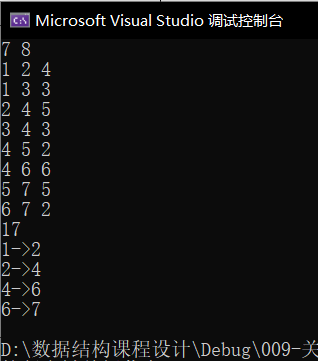
否

是

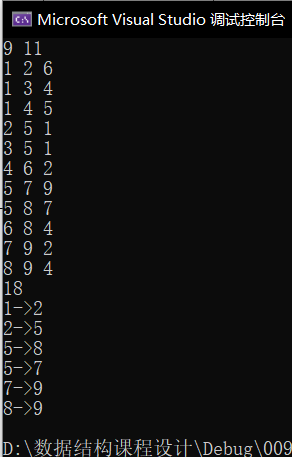
否

是

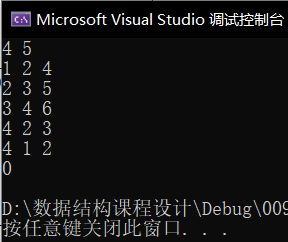
1. **程序测试**
   1. **简单情况测试**

****

* 1. **一般情况测试**

****

* 1. **不可行方案测试**

****