项目说明文档

数据结构课程设计

——关键活动

作 者 姓 名： 何慧琳

学 号： 2152343

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

在实际工程项目中往往有多个子任务组成，每个子任务开始的顺序或条件和用时不同，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”。本实验项目就是在给定工程项目的任务调度中找出关键活动，找出关键任务有利于整个项目花最短的时间竣工，因此本项目在工程管理方面具有重要意义。

## 1.2 功能分析

求关键活动首先应当能获取一个项目的任务调度方案，于是要有输入功能，获得输入的方案后要能先将其在内部存储表示出来，接着对其进行一系列操作找出关键活动并输出，此外，若调度方案不可行，也要给出反馈。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要能储存任务调度方案，由于一个项目由各子任务和子任务之间的先后关系，子任务的用时组成，采用带权图来储存较合适，结点即表示各任务间的交接点，边表示一个个子任务，边权表示子任务的用时，于是定义了一个图类（Graph）、图的边表结点类（Edge）、图的结点类（Vertex）。找到关键活动后为了按照一定次序输出采用队列先保存符合条件的子任务，并在其内将子任务排序，在找完所有关键活动后再逐个出队列并输出，于是还需一个队列类（Queue）。

## 2.2 类结构设计

**图类（Graph）**

由于找关键活动前要先对图进行拓扑排序，需要快速找到每个结点相邻的顶点，于是采用邻接表表示。图的基本组成就是顶点和边，于是定义了内部成员结点数组，每个结点后还带了边链表，已包含边的信息，此外为了确定开多大的结点数组以及访问的有效范围，定义了两个变量分别表示当前结点个数和最大结点个数。拓扑排序时需要检查结点的入度，于是开了个入度数组保存每个结点的入度，这在建立图的时候完成初始化，在拓扑排序时被调用。综合前面的功能分析，图应当实现插入结点、插入边、拓扑排序、求关键活动的功能，因此有如下成员及操作设计。

**公有操作：**

Graph(const int vertices = 0) : MaxNumVertex(vertices) //构造函数

{

NodeTable = new Vertex<NameType, DistType>[MaxNumVertex];

count = new int[MaxNumVertex];

};

~Graph();

int GetSeq(NameType name);//返回以name为名的结点在节点数组中的序号

void InsertVertex(NameType vertex);//插入节点

void InsertEdge(NameType v1, NameType v2,DistType weight);//插入边

int TopologicalOrder(int\* TopoQueue);//拓扑排序

void CriticalPath();//求关键活动

**私有成员：**

Vertex<NameType, DistType>\* NodeTable;//结点数组

int\* count;//每个结点的入度数组

int NumVertex=0;//当前结点个数

int MaxNumVertex;//最大结点个数

**图的结点类（Vertex）**

一个结点首先有自己的名称，由于采用邻接表表示图，因此每个结点后还链接了一个边表，故有如下成员及操作设计。

**私有成员：**

friend class Graph<NameType, DistType>;

NameType name;//结点名

Edge<DistType>\* adj\_first,\*adj\_current;//指向边表的第一条边和当前最后一条边

Vertex(Vertex\* v) :name(v.name), adj\_first(v->adj\_first), adj\_current(v->adj\_current) {};//复制构造函数

Vertex():name(NULL),adj\_first(NULL),adj\_current(NULL) {};//默认构造函数

**边表结点类（Edge）**

一个结点的边表存储与该结点关联的边的信息，一个边表结点储存一条边的终点在节点数组中的序号和边的权值，同时还应带一个指针指向下一个边表结点（与该顶点相关联的下一条边）。于是有如下成员及操作设计

**私有成员：**

int dest;//边的终点序号

DistType cost;//边的权值

Edge<DistType>\* link;//指向下一条边

Edge() { }//默认构造函数

Edge(int D, DistType C) :dest(D), cost(C), link(NULL) { }//带参构造函数

int operator != (Edge<DistType>& E) const//运算符重载

{

return dest != E.dest;

}

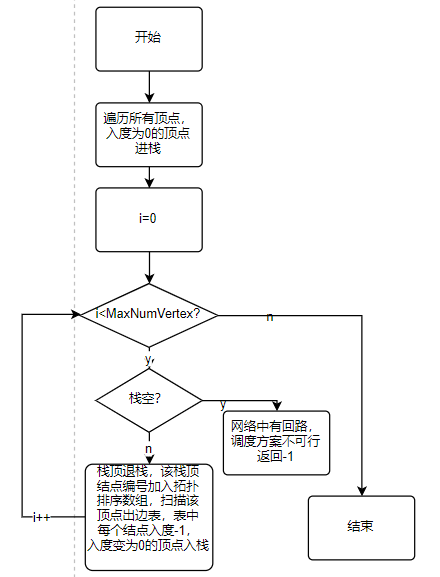
## 2.3 系统设计

系统首先调用main()函数提示用户进行操作，完成交接点编号和任务调度方案的输入工作，调用Graph中的函数完成对图的建立，最后调用Graph中的求关键路径函数完成求解并显示。

# 3 实现

## 3.1 拓扑排序功能的实现

### 3.1.1 拓扑排序功能流程图



### 3.1.2 拓扑排序功能核心代码

int Graph<NameType, DistType> ::TopologicalOrder(int\* TopoQueue)

{//拓扑排序，将排序后的结果存入Topo\_NodeTable

int top = -1; //入度为零的顶点栈初始化

for (int i = 0; i < MaxNumVertex; i++) //入度为零顶点进栈

if (count[i] == 0) {

count[i] = top;

top = i;

}

for (int i = 0; i < MaxNumVertex; i++) //期望输出n个顶点

if (top == -1) { //中途栈空,转出

return -1;//网络中有回路(有向环),返回-1

}

else { //继续拓扑排序

int j = top;

top = count[top]; //退栈

TopoQueue[i]= j; //结点编号加入拓扑排序数组

Edge<DistType>\* l = NodeTable[j].adj\_first;

while (l) { //扫描该顶点的出边表

int k = l->dest; //另一顶点

if (--count[k] == 0) //该顶点入度减一

{

count[k] = top;

top = k;

} //减至零

l = l->link;

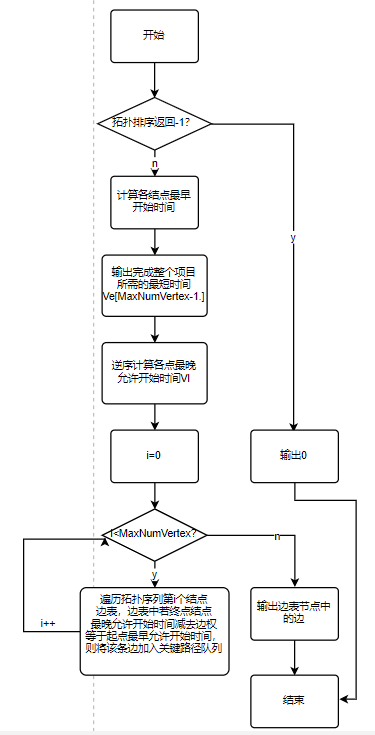
}

}

}

## 3.2 求关键活动功能的实现

### 3.2.1 求关键活动功能流程图



### 3.2.2 求关键活动功能核心代码

Queue<path\_edge> critical\_path;//保存关键路径,用于最终输出

//检查是否有环，获得拓扑结点数组

if (TopologicalOrder(TopoQueue) == -1) {//有环，不可行的方案，输出0

cout << "0" << endl;

return;

}

//无环，已获得拓扑排序结点数组，继续查找关键活动

Edge<DistType>\* p;//边表遍历指针

for (i = 0; i < MaxNumVertex; i++) //初始化Ve数组

Ve[i] = 0;

for (i = 0; i < MaxNumVertex; i++) {//计算各结点最早开始时间(起点到该点最长距离)

int m = TopoQueue[i];

p = NodeTable[m].adj\_first;//指向当前结点边表

while (p != NULL) {

k = p->dest;

if (Ve[m] + p->cost > Ve[k])

Ve[k] = Ve[m] + p->cost;

p = p->link;

}

}

cout << Ve[MaxNumVertex - 1] << endl;//输出完成整个项目所需最短时间

for (i = 0; i < MaxNumVertex; i++)//初始化Vl数组为所有活动最短可能用时

Vl[i] = Ve[MaxNumVertex - 1];

for (i = MaxNumVertex - 2; i; i--) {//从倒数第二个点开始倒序计算各点最晚允许开始时间

int m = TopoQueue[i];

p = NodeTable[m].adj\_first;

while (p != NULL) {

k = p->dest;

if (Vl[k] - p->cost < Vl[m])

Vl[m] = Vl[k] - p->cost;

p = p->link;

}

}

for (i = 0; i < MaxNumVertex; i++) {//输出关键活动

int m = TopoQueue[i];

p = NodeTable[m].adj\_first;

while (p != NULL) {

k = p->dest;

e = Ve[m];

l = Vl[k] - p->cost;

if (l == e)

critical\_path.push(path\_edge(m, k));//符合条件的边加入关键路径

p = p->link;

}

}

# 4 测试

## 4.1 功能测试

**测试用例**：

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**：

17

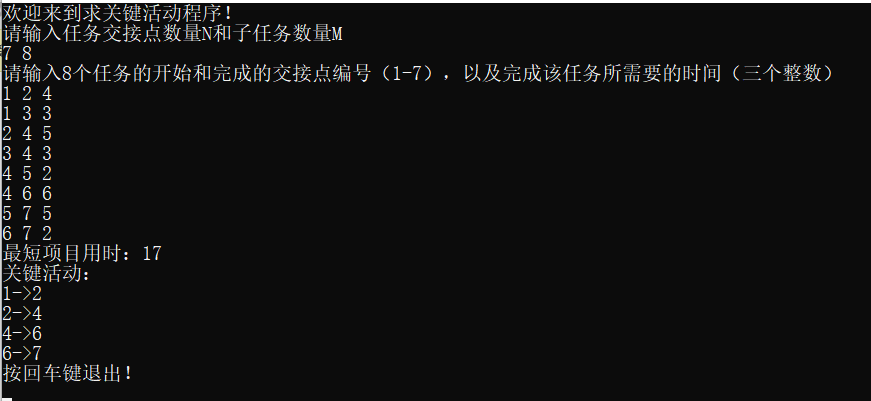
1 –>2

2 –>4

4 –>6

6 –>7

**实验结果**



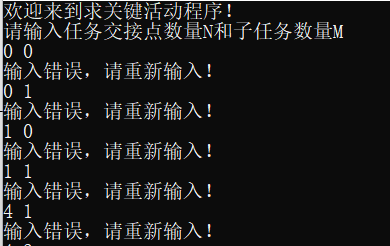
## 4.2 边界测试

### 4.2.1 初始化无输入数据

**测试用例：**初始交接点数量或子任务数量过少或不可能构成通路

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**



### 4.2.2 关键活动单个起点对应多个终点，边的输入顺序无序

**测试用例：**

9 11

2 5 1

3 5 1

5 8 7

6 8 4

7 9 2

4 6 2

5 7 9

8 9 4

1 2 6

1 3 4

1 4 5

**预期结果：**

18

1 –>2

2 –>5

5 –>8

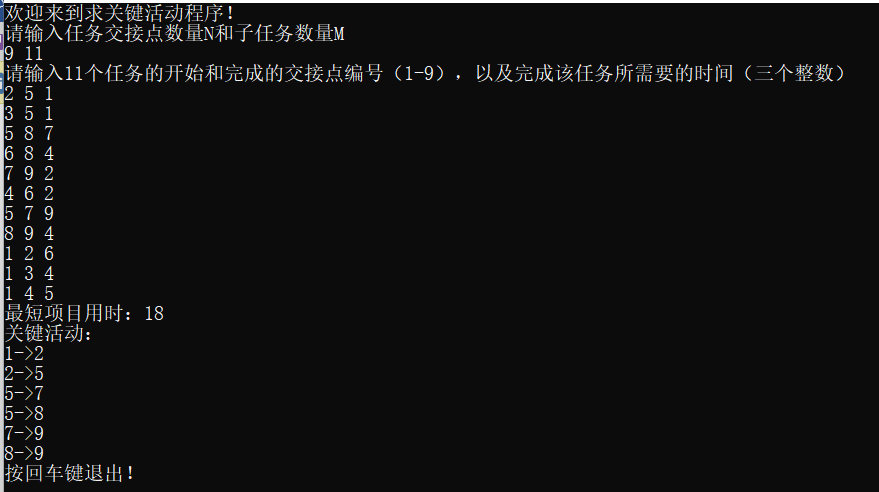
5 –>7

7 –>9

8 –>9

关键活动按起点编号递增顺序输出，起点相同时先输入的边先输出。

**实验结果：**



### 4.2.3 方案不可行

**测试用例：**

4 5

1 2 4

2 3 5

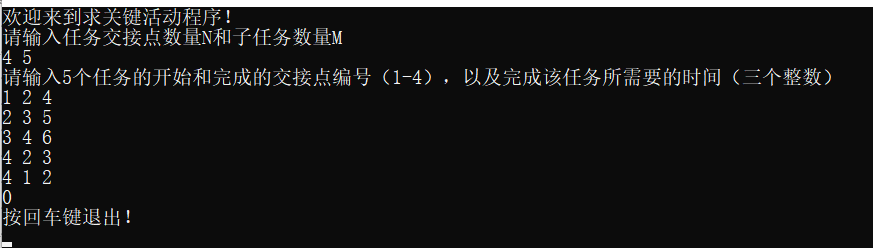
3 4 6

4 2 3

4 1 2

**预期结果：**0

**实验结果：**



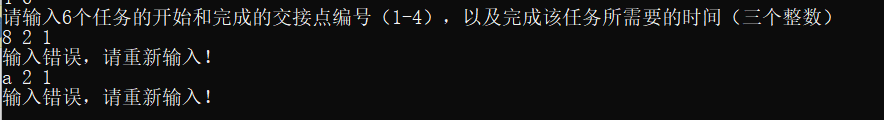
## 4.3 出错测试

### 4.3.1 结点编号输入错误

**测试用例：**输入编号大于结点数或者非数

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

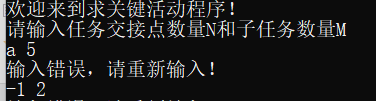
****

### 4.3.2 交接点数量或子任务数量错误

**测试用例：**输入交接点数量或子任务数量错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

****