项目说明文档

数据结构课程设计

——关键活动

作 者 姓 名： 张诚睿

学 号： 2150998

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](#_Toc122644835)

[1.1 项目背景分析 1](#_Toc122644836)

[1.2 功能分析 1](#_Toc122644837)

[2 设计 1](#_Toc122644838)

[2.1 数据结构设计 1](#_Toc122644839)

[2.2 类具体成员与操作设计 2](#_Toc122644840)

[2.4 main函数设计 2](#_Toc122644841)

[3 实现 3](#_Toc122644842)

[3.1 求关键活动功能的实现 3](#_Toc122644843)

[3.1.1 功能流程图 3](#_Toc122644844)

[3.1.2 核心代码 3](#_Toc122644845)

[3.1.3 效果演示 4](#_Toc122644846)

[3.2求Ve、V1、Ae、A1功能的实现 4](#_Toc122644847)

[3.2.1 功能概述 4](#_Toc122644848)

[3.2.2 核心代码 4](#_Toc122644849)

[4 测试 6](#_Toc122644850)

[4.1 功能测试 6](#_Toc122644851)

[4.1.1求解一般关键活动功能测试 6](#_Toc122644852)

[4.2 边界测试 7](#_Toc122644853)

[4.2.1 单个起点和单个终点 7](#_Toc122644854)

[4.2.2 不可行的方案 9](#_Toc122644855)

[4.3 出错测试 10](#_Toc122644856)

[4.3.1 交接点不是正整数 10](#_Toc122644857)

[4.3.2 子任务数量不是正整数 10](#_Toc122644858)

# 1 分析

## 1.1 项目背景分析

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”。

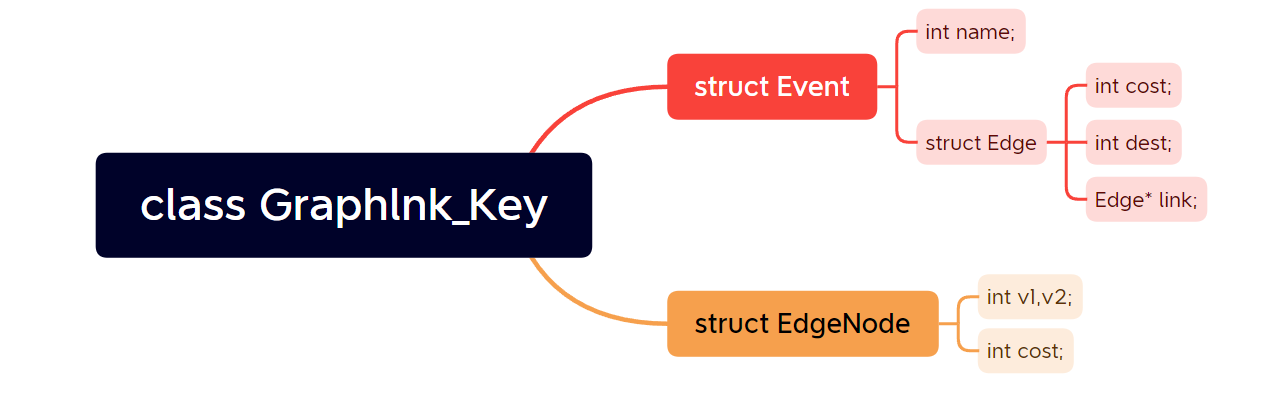
在实践过程中，提前求解“关键活动”来安排工作计划是有必要的，因此设计本项目来完成这一求解功能

## 1.2 功能分析

本项目的功能在于对于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。

# 2 设计

## 2.1 数据结构设计



核心数据结构设计如上，通过类-Graphlnk\_Key实现总体操作。用Event类型的数组和Edge作结点的链表来存储图的邻接表（在本项目中采用了出表）。另有动态申请的EdgeNode类型来存放求解的结果。

## 2.2 类具体成员与操作设计

/\* 关键活动各边 \*/

struct EdgeNode {

int v1, v2;

int cost;

EdgeNode() :v1(0), v2(0), cost(0) {};

};

/\* 邻接表中结点 \*/

struct Edge {

int cost;

int dest;

Edge\* link;

Edge() :cost(0), dest(-1), link(NULL) {};

};

/\* 邻接表 \*/

struct Event {

int name; // 名字都是数字表示的

Edge\* adj;

Event() :name(-1), adj(NULL) {};

};

class Graphlnk\_Key {

private:

Event\* NodeTable; // 动态申请

int size; // 顶点总数

int mission; // 活动总数

int count; // 关键活动数

EdgeNode\* cp; // 动态申请

public:

Graphlnk\_Key() :NodeTable(NULL), size(0), count(0), cp(NULL), mission(0) {};

~Graphlnk\_Key() { delete NodeTable; };

void init();

void Insert(int head, int tail, int cost);

void CriticalPath();

int IndNumber(int pos);

int GetFirstNeighbor(int begin);

int GetWeight(int u, int v);

int GetNextNeighbor(int Tar, int now);

void Show();

};

## 2.4 main函数设计

首先建立Graphlnk\_Key对象AOE并初始化，之后调用求解功能函数Graphlnk\_Key::CriticalPath()，求出结果后调用输出函数输出结果。

# 3 实现

## 3.1 求关键活动功能的实现

### 3.1.1 功能流程图

### 3.1.2 核心代码

求解Ve、V1、Ae、A1的子过程详见3.2

int top = -1, u, lnk = -1, v, w, k;

int\* ind = new int[size]; // 入度数组

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

{

ind[i] = IndNumber(i);

}

// 统计完每个顶点的入度

int\* Ve = new int[size];

int\* V1 = new int[size];

int\* Ae = new int[mission];

int\* A1 = new int[mission];

/\* 求解Ve、V1、Ae、A1的子过程详见3.2 \*/

delete[]ind;

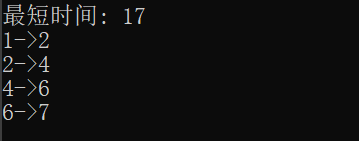
delete[]Ve;

delete[]V1;

delete[]Ae;

delete[]A1;

### 3.1.3 效果演示



## 3.2求Ve、V1、Ae、A1功能的实现

### 3.2.1 功能概述

由于四个过程之间相互耦合，不便于用流程图的方式直观展现逻辑过程，现用文字表述大致实现过程。

首先需要将Ve[]全部元素置零，遍历入度数组ind，对入度为零的点进行入栈，方式为ind[i]=top;top=i;之后只要栈不为空，就退栈一个顶点，用上述同理的方式进行反向拉链用于后续逆拓扑排序，逐个访问退栈顶点的出表，修改相应的入度数组，若有新的顶点入度为0，也进行入栈操作。若Ve[now]+weigh>Ve[visit]，则需要将Ve[visit]修改为Ve[now]+weigh，以此达到求出最长路径长度的效果。

求V1[]前将全部元素置Ve[lnk]，仿照求Ve的过程按逆拓扑排序的顺序求出最迟开始时间。

引入count计数，从头开始遍历每个顶点利用Ae[k]=Ve[i],A1[k]=V1[now]-weigh的方式求解Ae[]和A1[]数组，若Ae[k]==A1[k]，则可以将此边及起点、终点存入结果中。

### 3.2.2 核心代码

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

{

Ve[i] = 0; // 全部置0

}

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

{

if (!ind[i])

{

ind[i] = top;

top = i;

}

}

while (top!=-1)

{

now = top;

top = ind[top]; // 退栈，并保存顶点

ind[now] = lnk;

lnk = now; // 反向拉链

visit = GetFirstNeighbor(now);

while (visit != -1)

{

weigh = GetWeight(now, visit);

if (Ve[now] + weigh > Ve[visit])

{

Ve[visit] = Ve[now] + weigh;

}

if (--ind[visit] == 0)

{

ind[visit] = top;

top = visit;

}

visit = GetNextNeighbor(now, visit);

}

}

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

{

V1[i] = Ve[lnk];

}

while (lnk != -1)

{

visit = ind[lnk];

lnk = visit;

if (lnk == -1)

{

break;

}

ka = GetFirstNeighbor(visit);

while (ka != -1)

{

weigh = GetWeight(visit, ka);

if (V1[ka] - weigh < V1[visit])

{

V1[visit] = V1[ka] - weigh;

}

ka = GetNextNeighbor(visit, ka);

}

}

ka = 0;

count = 0;

cp = new EdgeNode[mission];

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

{

now = GetFirstNeighbor(i);

while (now != -1)

{

weigh = GetWeight(i, now);

Ae[ka] = Ve[i];

A1[ka] = V1[now] - weigh;

if (Ae[ka] == A1[ka])

{

cp[count].cost = weigh;

cp[count].v1 = i;

cp[count].v2 = now;

count++;

}

ka++;

now = GetNextNeighbor(i, now);

}

}

# 4 测试

## 4.1 功能测试

### 4.1.1求解一般关键活动功能测试

**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

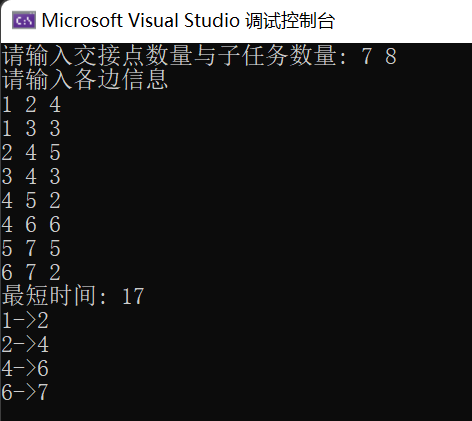
1->2

2->4

4->6

6->7

**实验结果:**



## 4.2 边界测试

### 4.2.1 单个起点和单个终点

**测试用例：**

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：**

29

1 –>2

2 –>5

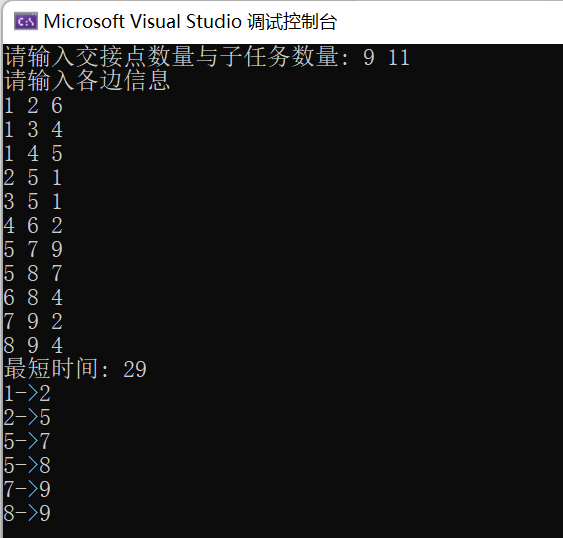
5 –>8

5 –>7

7 –>9

8 –>9

**实验结果：**



### 4.2.2 不可行的方案

**测试用例：**

4 5

1 2 4

2 3 5

3 4 6

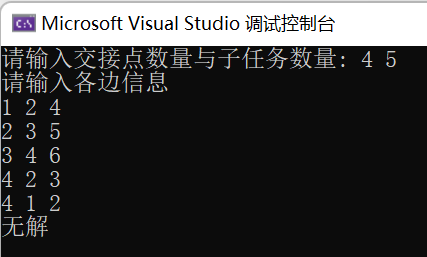
4 2 3

4 1 2

**预期结果：**

提示无解

**实验结果：**



## 4.3 出错测试

### 4.3.1 交接点不是正整数

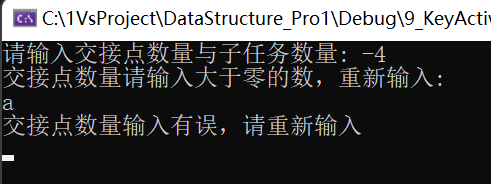
**测试用例：**

-4 a

**预期结果：**

输出提示信息，要求重新输入直至正确，程序不崩溃

**实验结果：**



### 4.3.2 子任务数量不是正整数

**测试用例：**

a b -4

**预期结果：**

输出提示信息，要求重新输入直至正确，程序不崩溃

**实验结果：**

