关键活动

1651573 刘客

功能介绍

- **输入说明**：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。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。
- 本实验采用了**拓扑排序**算法以及求解**关键活动**算法

编译说明

- 在windows平台下的.exe文件
- 在Linux平台下的out文件

一 . 设计

1. 数据结构设计

> 由题意知,该题目的实质就是求解一个AOE网络,并将关键活动输出,因此采用了**图**、**邻接表**、**栈**等结构进行实现

2. 类的设计

- Edge 结构体
  - 成员变量

成员名称	属性	类型	描述
vertexIndex	public	int	节点索引(尾节点)
duration	public	int	活动持续的时间
nextEdge	public	Edge*	指向下一条边的指针

- 成员函数

函数名称	返回值类型	描述
Edge	无	构造函数
getIndex	int	获得当前索引
getDuration	int	获得当前活动持续时间
getNextEdge	Edge*	获得下一条边的指针

- LinkNode 类
  - 成员变量

成员名称	属性	类型	描述
vertexIndex	private	int	节点索引(首节点)
link	private	Edge*	所链接的边的指针

  - 成员函数

函数名称	返回值类型	描述
LinkNode	无	构造函数
LinkNode	无	重载的构造函数
setLink	void	设置该节点所链接的边
getFirstEdge	Edge*	获得该节点所链接的第一条边
deleteNode	void	将分配的内存收回

- Graph 类
  - 成员变量

成员名称	属性	类型	描述
vertexV	private	LinkNode	存储LinkNode* 的数组
vertexNum	private	int	顶点的个数

- 成员函数

函数名称	返回值类型	描述
Graph	无	构造函数
~Graph	无	析构函数
deleteFun	void	收回内存
getVertex	LinkNode*	获得第i个节点的指针

- Stack 结构体
  - 成员变量

成员名称	属性	类型	描述
container	public	int*	用来存储索引的数组
currentIndex	public	int	指向栈顶元素的索引

  - 成员函数

函数名称	返回值类型	描述
Stack	无	构造函数
~Stack	无	析构函数
push	void	将元素压入栈内
pop	int	将元素推出栈外
getTop	int	获得栈顶元素
isEmpty	bool	判断栈是否为空

二. 主程序实现

> 首先将输入的数据读取,使用Graph、LinkNode和Edge类将数据之间的关系存储起来。再进行拓扑排序,如果发现排序完成后存在回路,则输出0,若拓扑排序成功,则继续进行关键活动的查找,最后将其输出。

- 函数

函数名称	返回值类型	描述
topiSort	bool	进行拓扑排序
findKeyActivity	void	寻找关键活动
main	int	主函数

- topiSort

> 形参分析
> Graph& graph 存储各个活动之间关系的图
> int* inPoint 入度数组(在初始化数据时已经进行了统计)
> int* sortVector 存储拓扑排序的vector(待装填状态)
> Stack& myStack 辅助的栈
> int size 顶点的个数

> 返回值分析
> bool类型
> 当拓扑排序成功时返回true
> 当拓扑排序失败时返回false

```
bool topiSort(Graph& graph, int *inPoint, int* sortVector, Stack& myStack,int size)
{
    int count = 1;
    //将入度为0的点先推入栈中
    for (int i = 1; i < size; i++) {
        if (inPoint[i] == 0) {
            myStack.push(i);
            //sortVector 存储拓扑排序序列
            sortVector[count++] = i;
        }
    }

    //当栈不为空时,取出栈顶的元素,然后在graph中寻找以栈顶元素为起点的边,将这些边连接的终点的入度减一
    //如果有的终点减一后入度为0，则将其再度推入栈中
    while (!myStack.isEmpty()) {
        int index = myStack.pop();
        LinkNode* node = graph.getVertex(index);
        if (node == NULL) {
            //说明该节点是终点
            break;
        }
        Edge* tempEdge = node->getFirstEdge();
        while (tempEdge != NULL) {
            inPoint[tempEdge->getIndex()]--;
            if (inPoint[tempEdge->getIndex()] == 0) {
                myStack.push(tempEdge->getIndex());
                sortVector[count++] = tempEdge->getIndex();
            }
            tempEdge = tempEdge->getNextEdge();
        }
    }
    //当while循环结束时,判断入度数组中是否所有元素入度全部为0,如果有不为0的元素,说明拓扑排序失败返回false
    for (int i = 1; i < size; i++) {
        if (inPoint[i] != 0) {
            return false;
        }
    }
```

```
    return true;
}
```

- findKeyActivity

> 形参分析
> Graph& graph 存储各个活动之间关系的图
> int* sortVector 存储拓扑排序的vector(已经有序)
> int size 顶点的个数

```
void findKeyActivity(Graph& graph, int* sortVector, int size) {
    int *Ve = new int[size];//最早开始时间Vector
    int *Vl = new int[size];//最迟开始时间Vector
    //初始化
    for (int i = 0; i < size; i++) {
        Ve[i] = 0;
        //最迟开始时间初始化为最大值
        Vl[i] = INT_MAX;
    }
    //先通过循环填充最早开始时间Vector
    for (int i = 1; i < size - 1; i++) {
        int curIndex = sortVector[i];
        LinkNode* tempNode = graph.getVertex(curIndex);
        Edge* edge = tempNode->getFirstEdge();
        while (edge != NULL) {
            //index是边的终点
            int index = edge->getIndex();
            //如果Ve[某一起点] + edge[该起点到某一终点] > Ve[某一终点],则将Ve[某一终点]更新
            if (Ve[curIndex] + edge->getDuration() > Ve[index]) {
                Ve[index] = Ve[curIndex] + edge->getDuration();
            }
            edge = edge->getNextEdge();
        }
    }
    //再通过循环算出最迟开始时间
    //终点的最早开始时间和最迟开始时间相同
    Vl[sortVector[size - 1]] = Ve[sortVector[size - 1]];
    cout <<"\n最短时间为: " <<Ve[sortVector[size - 1]] << endl<<endl;
    for (int i = size - 2; i >= 1; i--) {
        int curIndex = sortVector[i];
        LinkNode* tempNode = graph.getVertex(curIndex);
        Edge* edge = tempNode->getFirstEdge();
        while (edge != NULL) {
            int index = edge->getIndex();
            //如果Vl[某一起点] + edge[该起点到某一终点] > Vl[某一终点],则将Vl[某一终点]更新
            if (Vl[curIndex] > Vl[index] - edge->getDuration()) {
                Vl[curIndex] = Vl[index] - edge->getDuration();
            }
            edge = edge->getNextEdge();
        }
    }
    cout << "关键活动为:\n";
    //最后根据算出的Ve和Vl数据来求关键活动
```

```
        for (int i = 1; i < size - 1; i++) {
            int curIndex = sortVector[i];
            int Ae = Ve[curIndex];
            LinkNode* tempNode = graph.getVertex(curIndex);
            Edge* edge = tempNode->getFirstEdge();
            int count = 0;
            while (edge != NULL) {
                count++;
                edge = edge->getNextEdge();
            }
            while (count != 0) {
                Edge* tempEdge = tempNode->getFirstEdge();
                for (int i = 1; i < count; i++) {
                    tempEdge = tempEdge->getNextEdge();
                }
                int Al = Vl[tempEdge->getIndex()] - tempEdge->getDuration();
                //如果Al == Ae,则说明为关键活动
                if (Al == Ae) {
                    cout << curIndex << " -> " << tempEdge->getIndex()<<endl;
                }
                count--;
            }
        }
    }
```

三. 程序运行效果

- 简单情况测试

- 一般情况测试

```
请输入N(交接点数量)和M(任务数量)
===格式(N M)===
请输入: 9 11
请依次输入M行,每行给出3个正整数(任务开始交接点A,任务完成交接点B,以及任务时间C)
===格式(A B C)===
请输入:
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

最短时间为: 18

关键活动为:
1 -> 2
2 -> 5
5 -> 8
5 -> 7
7 -> 9
8 -> 9
```

- 不可行的方案测试

```
D:\数据结构课程设计\课程项目9\Project1\Debug\Project1.exe
请输入N(交接点数量)和M(任务数量)
===格式(N M)===
请输入: 4 5
请依次输入M行,每行给出3个正整数(任务开始交接点A,任务完成交接点B,以及任务时间C)
===格式(A B C)===
请输入:
1 2 4
2 3 5
3 4 6
4 2 3
4 1 2
0
```