

数据结构项目文档

题目：关键活动

指导教师：张颖

姓名：王亮

学号：1653340

数据结构项目文档

一、题目分析

- 项目简介
- 功能需求
- 设计思路

二、设计

- 数据结构设计
- 类的设计
- 函数设计

三、实现

- 求关键路径
 - 算法
 - 相关代码
- 拓扑排序
 - 算法
 - 相关代码
- 按照要求输出
 - 算法
 - 相关代码

四、测试

- 功能测试（包括特殊情况和不可行情况）
 - 简单情况测试
 - 一般情况测试，单个起点和单个终点
 - 不可行的方案测试

一、题目分析

1. 项目简介

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子任务需要的时间，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。

2. 功能需求

1. 输入说明：输入第1行给出两个正整数N ($N \leq 100$) 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。
2. 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。
3. 测试用例：

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

3. 设计思路

首先分析问题，可以确定这是一个AOE网络中求关键路径问题。

具体设计思路：

首先确定项目采用图作为抽象数据结构，确定图的存储表示，定义类的成员变量和成员函数；然后实现计算AOE图的关键活动的函数；最后完成主函数以验证程序的功能并得到运行结果。

二、设计

1. 数据结构设计

这是一个AOE网络（用边表示活动的网络）中的关键活动问题。

因此选择图作为抽象数据结构，并通过链表实现图的邻接表表示。

2. 类的设计

为了实现链接表表示的图，需要Edge边结点、Vertex顶点、Graph图的抽象基类、GraphInk : Graph邻接表表示的图等类。

为了实现完整的图的模板类，还是使用了MSTEdgeNode、MinSpanTree、MinHeap、UFSets、Tree、SeqQueue作为辅助类以实现图中的一些功能。

由于本题目使用的所有数据结构与第8题中相同，且已经在第8题的文档中详细阐述过，故不在这里赘述。

3. 函数设计

为了实现关键路径的计算，首先需要对顶点进行拓扑排序，然后按照算法求出关键路径。

因此主要需要拓扑排序和求关键路径两个主要函数。

三、实现

1. 求关键路径

1.1 算法

计算关键路径的算法：

1. 输入e条带权的有向边，建立邻接表结构。
2. 从源点V0出发，令 $Ve[0]=0$ ，按拓扑有序的顺序计算每个顶点的 $Ve[i]$, $i=1,2,\dots,n-1$ 。若拓扑排序的循环次数小于顶点数n，则说明网络中存在有向环，不能继续求关键路径。
3. 从汇点 V_{n-1} 出发，令 $VI[n-1]=Ve[n-1]$ ，按拓扑有序逆序的顺序求各顶点的 VI 。
4. 根据各顶点的 Ve 和 VI 值，求各有向边的 Ae 和 AI 。
5. 若 $Ae[k]=AI[k]$ 即为关键活动，输出关键活动。

1.2 相关代码

```
/*
 * @brief: 插入各条带权边，然后进行拓扑排序，并把排序结果存在order数组中。
 */
template <typename T, typename E>
bool topologicalSort(GraphInk<T, E>* gl, int numV, int numE, int* order)
{
    int start, end, weight, w, v, cnt;
    int top = -1;
    int n = numV;
    int* count = new int[n];
    for (cnt = 0; cnt < n; cnt++) count[cnt] = 0;
    for (cnt = 0; cnt < numE; cnt++)
    {
        cin >> start >> end >> weight;
        gl->insertEdge(gl->getVertexPos(start), gl->getVertexPos(end),
            weight);
    }
}
```

```

        count[gl->getVertexPos(end)]++;
    }

    for (cnt = 0; cnt < numV; cnt++)
    {
        if (count[cnt] == 0)
        {
            count[cnt] = top;
            top = cnt;
        }
    }

    for (cnt = 0; cnt < numV; cnt++)
    {
        if ( top == -1 )
        {
            return false;
        }
        else
        {
            v = top; top = count[top];
            order[cnt] = v;
            //cout << cnt << " " << order[cnt] << endl;
            w = gl->getFirstNeighbor(v);
            while(w != -1)
            {
                if (--count[w] == 0)
                {
                    count[w] = top;
                    top = w;
                }
                w = gl->getNextNeighbor(v, w);
            }
        }
    }

    return true;
}

```

2. 拓扑排序

2.1 算法

为了实现拓扑排序，需要增设一个数组Count[]记录各个顶点的入度。为了优化存储空间，直接利用入度为零的顶点的count[]数组元素建立入度为零的顶点栈，同时设立栈顶指针top，指示当前栈顶的位置，即某一个入度为零的顶点的位置。栈初始化时置top=-1，表示空栈。

拓扑排序的步骤：

1. 输入AOV网络。
2. 在AOV网络中选一个没有直接前驱的顶点，并输出之。
3. 从图中删去改点，同时删去所有它发出的有向边。

重复以上2, 3步，直到

- 全部顶点均已输出，拓扑有序序列形成
- 图中还有未输出的顶点，但已跳出循环处理。说明AOV网络中存在有向环

2.2 相关代码

```
/*
 * @brief: 求关键活动，结果保存在二维数组resultMtx中
 */
template <typename T, typename E>
void keyActivities(Graphlnk<T, E>* gl, int numV, int* order, int** rMtx)
{
    int i, j, k;
    E Ae, Al, w;
    E* Ve = new E[numV];
    E* Vl = new E[numV];
    for(i = 0; i < numV; i++) Ve[i] = 0;
    for(i = 0; i < numV; i++)
    {
        j = gl->getFirstNeighbor(order[i]-1);
        while(j!=-1)
        {
            w = gl->getWeight(order[i]-1, j);
            if(Ve[order[i]-1] + w > Ve[j]) Ve[j] = Ve[order[i]-1] + w;
            j = gl->getNextNeighbor(order[i]-1, j);
        }
    }

    cout << Ve[order[numV-1]] << endl;           //输出完成整个项目所需要的时间

    //    for (int x = 0; x < numV; x++)
    //        cout << Ve[x] << " ";
    //    cout << endl;

    for(int x = 0; x < numV; x++)
        Vl[order[x]] = Ve[order[numV-1]];
    for(j = numV-2; j > 0; j--)
    {
        k = gl->getFirstNeighbor(order[j]);
        while(k!=-1)
        {
            w = gl->getWeight(order[j], k);
            if(Vl[k] - w < Vl[order[j]]) Vl[order[j]] = Vl[k] - w;
            k = gl->getNextNeighbor(order[j], k);
        }
    }
```

```

    }

    //    for (int x = 0; x < numV; x++)
    //        cout << V1[x] << " ";
    //    cout << endl;

    for(i = 0; i < numV; i++)
    {
        j = gl->getFirstNeighbor(i);
        while(j != -1)
        {
            Ae = Ve[i]; Al = V1[j] - gl->getWeight(i, j);
            if (Al == Ae)
                rMtx[i][j] = 1;
            j = gl->getNextNeighbor(i, j);
        }
    }
    delete[] Ve; delete[] V1;
}

```

3. 按照要求输出

3.1 算法

由于题目中对输出格式有一定要求：

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。

为了实现这一目的，首先用二维数组记录求出的关键活动。

结合插入边时边链表链入边结点的方式：

```

    p = new Edge<T,E>;           //否则，创建新结点
    p->dest = v2;
    p->cost = weight;
    p->link = NodeTable[v1].adj; //链入v1边链表
    NodeTable[v1].adj = p;

```

可以看出（主要是通过后两行代码），每次插入一个新的边结点，是将边结点插在边链表的头部，所以每个结点直接从前向后遍历边链表即可实现题目要求的编号相同时按输入的逆序输出。

3.2 相关代码

```

/*
 * @brief: 按照要求输出
 */

```

```

template <typename T, typename E>
void output(Graphlnk<T, E>* gl, int** rMtx, int numV)
{
    int tempVertex;
    int i;
    for(i = 0; i < numV; i++)
    {
        tempVertex = gl->getFirstNeighbor(i);           //结合插入边时的边链表链入结
        点的方式，每个节点直接从前向后遍历边链表即可。
        while (tempVertex != -1)
        {
            if (rMtx[i][tempVertex] == 1)
            {
                cout << gl->getValue(i) << "->" << gl->getValue(tempVertex)
<< endl;
            }
            tempVertex = gl->getNextNeighbor(i, tempVertex);
        }
    }
}

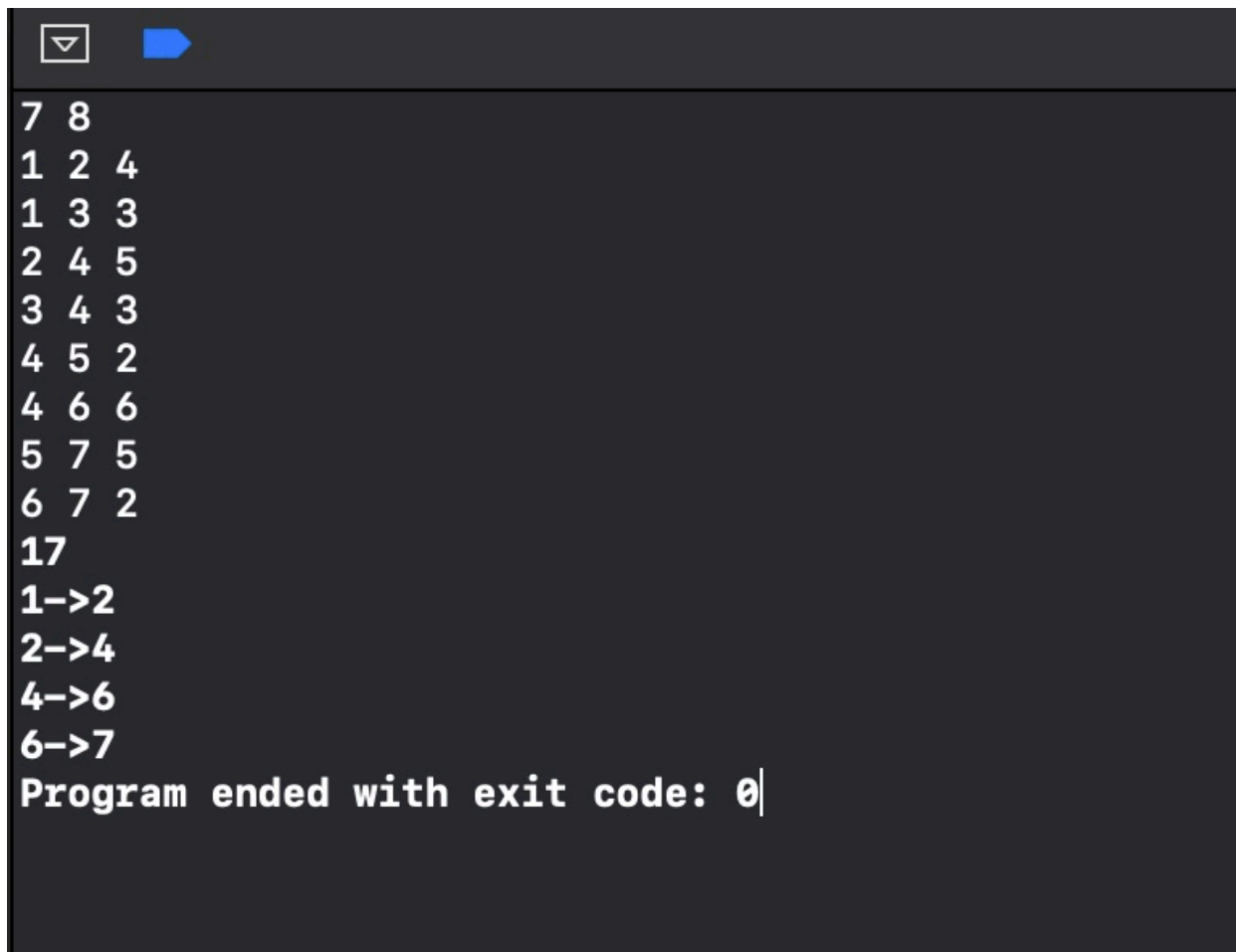
```

四、测试

1. 功能测试（包括特殊情况和不可行情况）

1.1 简单情况测试

测试结果：

A terminal window with a dark background and light gray text. The output consists of several lines of numbers and a final status message. The numbers are arranged in a pattern that suggests a sequence or a path. The final line indicates the program has ended successfully.

```
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
Program ended with exit code: 0
```

测试结果正确。

1.2 一般情况测试，单个起点和单个终点

测试结果：

```
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
Program ended with exit code: 0
```

实现了按照题目要求的方式输出，即任务<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。

测试结果正确。

1.3 不可行的方案测试

```
4 5
1 2 4
2 3 5
3 4 6
4 2 3
4 1 2
0
Program ended with exit code: 0
```

由于存在有向环，因此没有关键活动。

测试结果正确。