《数据结构》上机报告

2018 年 12 月 1 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分：

实验题目	有向无环图（DAG 图）	
问题描述	述工程或系统的进行过程的有效工具。如果有向图中从顶点 v 到 w 有一条有向路径，则 v 一定排在 w 之前，这样构成的一个顶点序列就称为拓扑序，构造拓扑序的过程就是拓扑排序。若拓扑排序不能输出所有顶点，说明 AOV 网络中存在有向环，此 AOV 网络所代表的工程是不可行的。	
基本要求	1. 给出一组有向图，判断是否含有环； 2. 一个工程项目由一组活动（或称子任务）构成，活动之间有的可以并行执行，有的必须在完成了其它一些活动后才能执行，并且每个活动完成需要一定的时间。对于一个工程，需要研究的问题是： （1）由这样一组活动描述的工程是否可行？ （2）若可行，计算完成整个工程需要的最短时间。 （3）这些活动中，哪些活动是关键活动（也就是必须按时完成的任务，否则整个项目就要延迟）。 现给定一个 AOE 网，有向边（弧）表示活动，弧上权值表示活动完成需要的时间。	
	已完成基本内容（序号）：	1， 2
选做要求		
	已完成选做内容（序号）	

数据结构设计

```
typedef int VertexType;
typedef struct ArcNode
{
    int adjvex;//弧指向的顶点的位置
    ArcNode *nextarc;//指向下一个与该顶点邻接的顶点
    int info;//弧的相关信息
}ArcNode;//边表结点
typedef struct VNode
{
    VertexType data;//用于存储顶点
    int indegree;
    ArcNode *firstarc;//指向第一个与该顶点邻接的顶点
}VNode, AdjList[MAX_VERTEX_NUM];//表头节点，顺序表存储
typedef struct
{
    AdjList vertices;//邻接表
    AdjList _vertices;//逆邻接表
```

<table>
<tr><td></td><td>

```
    int vexnum, arcnum;//边数，顶点数
    int kind;//图的种类
}ALGraph;
```

本实验主要运用的数据结构是图，通过图来存储数据，一种方法是邻接矩阵法，适用于边比较稠密的图，另一种方法是邻接表法，主要适用于边比较稀疏的图。本实验是邻接表法，邻接表法是用一个顶点表来存放顶点数据，其相当于一个顺序表；还有一个边结点表用来存储与顶点相邻的顶点的数据，结构体包括顶点信息info和顶点下标adjvex和指向下一个与该顶点邻接的顶点的指针nextarc其相当于一个链式表。还有一个结构体表示图，用来将前两个表封装起来使用，便于操作图。

</td></tr>
<tr><td>功能(函数)说明</td><td>

```
/***********************************************************
函数功能：定位顶点
函数说明：通过遍历查找与目标定点相同的顶点的编号返回即可。
***********************************************************/
int LocateVertex_L(ALGraph& G, VertexType v)
{
    int i;
    for (i = 1; i <= G.vexnum; i++)
        if (G.vertices[i].data == v)
            return i;
    return -1;
}
/***********************************************************
函数功能：创建图
函数说明：创建图有邻接表法和邻接矩阵法；邻接表法就是用一个结构数
组即顺序表存储顶点，称为顶点表；创建边结点表存储在与顶点相关联的
一条边（弧）的另一个顶点，以链表的方式进行头插法建立。邻接矩阵法
就是用一个方阵来存储两个顶点是否相邻的信息，若i,j顶点相邻，就将
矩阵的i行j列元素置1，（或者为权值，若是无向图，j行i列也做此操作）
在本题目中，用的是邻接表创建的有向图，还要计算某个顶点的入度，所
以同时建立邻接表和逆邻接表（方便计算入度）。
***********************************************************/
void CreateGraph(ALGraph &G)
{
    int i, j, k;
    VertexType v1, v2;
    ArcNode *p;
    int w;
    cin >> n >> m;
    G.vexnum = n;
    G.arcnum = m;
    for (i = 1; i <= G.vexnum; i++)
    {
        G.vertices[i].data = i ;
        G._vertices[i].data = i;
```

</td></tr>
</table>

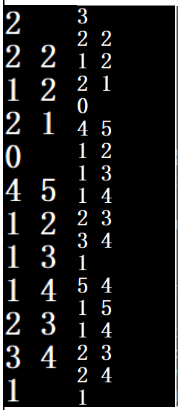
```
        G._vertices[i].firstarc = NULL;
        G.vertices[i].firstarc = NULL;
    }
    for (k = 1; k <= G.arcnum; k++)
    {
        cin >> v1 >> v2>>w;
        i = LocateVertex_L(G, v1);
        j = LocateVertex_L(G, v2);
        /*j为入i为出创建邻接链表*/
        p = new ArcNode;
        p->adjvex = j;
        p->info = w;
        p->nextarc = G.vertices[i].firstarc;
        G.vertices[i].firstarc = p;
        /*i为入j为出创建逆邻接链表*/
        p = new ArcNode;
        p->adjvex = i;
        p->nextarc = G._vertices[j].firstarc;
        G._vertices[j].firstarc = p;
    }
}
/***************************************************************
函数功能：计算顶点的入度
函数说明：从逆邻接表的顶点表开始遍历，对每一个顶点的链表进行遍历，
同时计数，从而得到每个顶点的入度。
***************************************************************/
void FindIndegree(ALGraph &G)
{
    ArcNode *p;
    for (int i = 1; i <= G.vexnum; i++)
    {
        int count = 0;
        p = G._vertices[i].firstarc;
        while (p)
        {
            count++;
            p = p->nextarc;
        }
        G.vertices[i].indegree = count;
    }
}
/***************************************************************
函数功能：拓扑排序
函数说明：主要还是按照删边法进行拓扑排序，建立一个0入度顶点栈，
逐个删除边，将零入度顶点压栈，每次出栈的时候都将该顶点压入拓扑
```

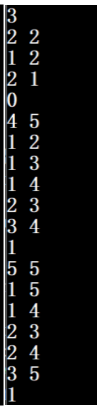
	序列顶点栈(便于后面求关键路径)，还有就是修改事件v[k]的最早发生时间,为各条路径时间和的最大值,详细见下面代码中的注释 ``` ***/ int ToplogicalSort_CP(ALGraph &G) { //CP:Critical Path //求关键路径时候的拓扑排序（与以普通拓扑排序方法不同的地方 //只有一句，再者就是这种方法新建了栈，前者则借助indegree作 //为栈来使用） //T:拓扑序列顶点栈 memset(Ve, 0, sizeof(Ve)); ArcNode *p; int count = 0; FindIndegree(G);//求各顶点的入度 stack<int>S;//0入度顶点栈 int i, k; for (i = 1; i <= G.vexnum; i++) if (G.vertices[i].indegree == 0) S.push(i);//入度为0，进栈; while (!S.empty())//栈不空 { int temp; temp = S.top(); T.push(temp);//拓扑序列元素下标入栈 S.pop();//出栈 count++;//计数 for (p = G.vertices[temp].firstarc; p; p = p->nextarc) { k = p->adjvex; if (--G.vertices[k].indegree == 0) S.push(k);//度为0，入栈 if (Ve[temp] + p->info > Ve[k]) Ve[k] = Ve[temp] + p->info;//修改事件v[k]的最早发生时间,为各条路径时间和的最大值 }//对以G.vertices[S.top()]为顶点的弧的另一个顶点进行操作 } if (count < G.vexnum) return 0;//有环 return 1; } /** ``` 函数功能：求关键路径 函数说明：先初始化时间最迟发生时间为工程完成的最早发生时间，在拓扑排序中已经得到了拓扑序列顶点栈，在此只要栈不空就进行出栈，即为逆拓扑序列，出栈后，对该顶点坐在的邻接表进行遍历，并修改与之相邻

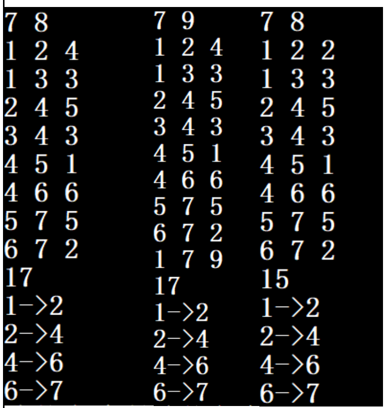
```
的顶点v[k]的最迟发生时间，为多个值中的最小值；最后，对整个的邻接
表进行遍历，同时对活动最早开始时间赋值（等于活动的头顶点最早
发生时间），和最迟开始时间进行赋值（等于活动的尾顶点最迟发生时间-
活动持续时间dut），至此，若是Ee= El，则证明该活动j->k为关键活动
*****************************************************************/
void CriticalPath(ALGraph &G)
{
    ArcNode *p;
    int dut, k;
    int Ee, El;//活动（即边E）最早发生和最迟开始的时间
    for (k = 1; k <= G.vexnum; k++)
        Vl[k] = Ve[G.vexnum];
    //初始化"事件最迟发生时间"数组为Ve数组最大者
    while (!T.empty())
    {
        int temp;
        temp = T.top();
        T.pop();
        for (p = G.vertices[temp].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex;
            dut = p->info;
            if (Vl[k] - dut < Vl[temp])
                Vl[temp] = Vl[k] - dut;//修改事件v[k]的最迟发生时间，
为最小值
        }
    }
    cout << Ve[G.vexnum] << endl;
    for (int j = 1; j <= G.vexnum; j++)
        for (p = G.vertices[j].firstarc; p; p = p->nextarc)//求解Ee，El
和关键活动
        {
            k = p->adjvex;
            dut = p->info;
            Ee = Ve[j];
            El = Vl[k] - dut;
            if (Ee == El)
                cout << j << "->"  << k << endl;
        }
}
/*****************************************************************
函数功能：拓扑排序
函数说明：首先调用求顶点入度函数进行计算顶点入度为邻接表的Indegree
赋值，然后通过对邻接表的遍历找出入度为0的顶点，入栈（此处的栈是
```

```
Indegree数组，其实他就是按照静态链表的方式进行存储，在访问完顶点
之后就将与该顶点相邻的顶点入度减一，若减完之后顶点入度为0，就再入
栈，直到栈为空就停止，这样就排好了拓扑序。若是计数之后的顶点数少于
实际顶点数，说明该图不是有向无环图，否则，则是有向无环图。
*******************************************************/
int ToplogicalSort(ALGraph &G)
{  //普通拓扑排序
    ArcNode *p;
    FindIndegree(G);
    int top = -1;//stack<int>S;
    int i, k;
    for (i = 1; i <=G.vexnum; i++)
        if (G.vertices[i].indegree == 0)
        {
            G.vertices[i].indegree = top;
            top = i;
        }//入度为0，进栈,S.push(G.vertices[i].data);
    int count = 0;
    while (top != -1)//栈不空
    {
        i = top;
        top = G.vertices[top].indegree;//出栈,S.pop();
        //cout << G.vertices[i].data << " ";//cout<<S.top()<<" ";
        count++;
        for (p = G.vertices[i].firstarc; p; p = p->nextarc)
        {
            k = p->adjvex;
            G.vertices[k].indegree--;
            if (G.vertices[k].indegree == 0)
            {
                G.vertices[k].indegree = top;
                top = k;
            }//度为0，入栈,S.push(G.vertices[k].data);
        }
    }
    if (count < G.vexnum)
        return 0;//有环
    return 1;
}
```

开发环境	Win10,vs2017,C++高级程序语言设计

<table>
<tr><td>调试分析</td><td>Problem1：




Problem2：

</td></tr>
<tr><td>心得体会</td><td>本实验主要是对图的邻接表法建立的具体运用，主要方法就是拓扑排序法，以及利用它进行关键路径的求解。

当然其中的主要算法就是两种拓扑排序（一种是普通排序，一种是求关键路径时候的排序）两种排序的核心思想依旧是相同的——删边法，通过建立逆邻接表对入度的求解，找到入度为 0 的顶点，然后压栈，之后进入循环，只要栈不空就进行循环，在循环过程中，对 0 入度顶点的边结点表进行遍历，每次遍历都将它的邻接顶点的入度减一，若入度减为零继续入栈，直到栈空为止。这其中若是要求解关键路径，则还要同时对某个顶点事件发生的最早时间进行判断，（它应当是多个路径时间和里面的最大值），将每个拓扑序列的顶点压栈，便于求解关键路径。

在求解关键路径的时候，还要对顶点事件最迟发生的时间进行求解，这个比较容易，因为最早时间已经求解出，对拓扑序列出栈直到栈空，只要求得（每个顶点事件最早发生时间+活动持续时间）中的最小值即可得最迟发生时间；再者就是边活动最早和最迟发生时间，它也是建立在顶点事件最迟到和最早发生时间上的，所以易得，若两者相等，则该活动为关键活动。

总的来说，求解关键路径和拓扑排序密切相关，所以掌握好两者间的联系是解答该类题目的核心要领。</td></tr>
</table>