数据结构

实验报告（六）

AOE 网络的关键路径

学号：3020205015

姓名：石云天

班级：智能机器平台 2 班

日期：2022.12.13

目　录

一、实验内容描述

本次实验主题是 AOE 网络的关键路径，对于给定的 AOE 网络，需要求出该网络的关键路径，所给网络如下图 1 所示：

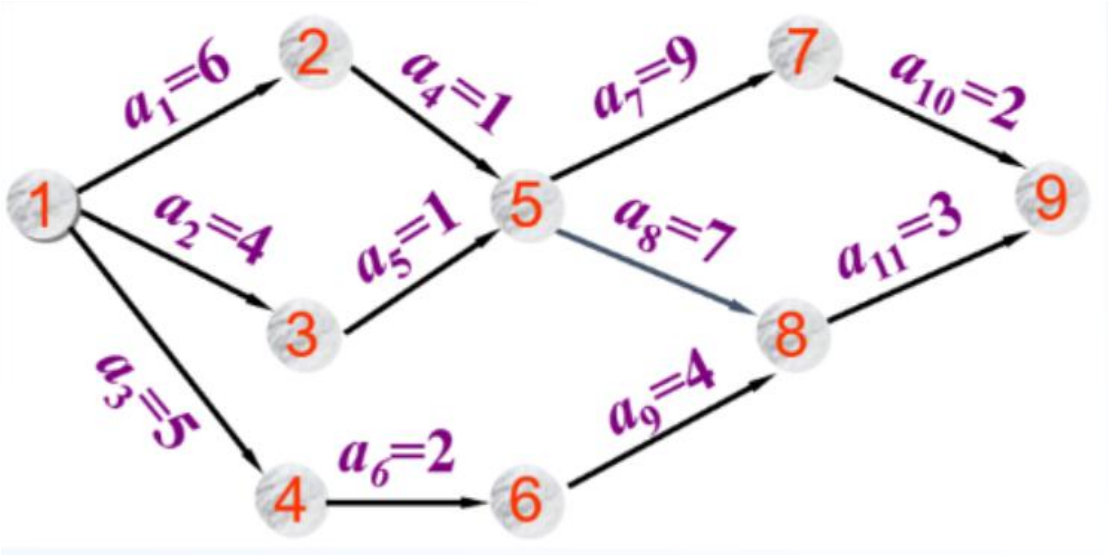

图 1 实验所给的 AOE 活动网络

二、实验步骤

（1）根据上课所讲，回顾 AOE 网络的基本概念，考虑图的存储结构。回顾图的邻接表和邻接矩阵两类存储结构，并根据实验要求选取合适的存储结构。同时认识到 AOE 网络是一个带权的有向无环图，常用于估算工程的完成时间。

（2）仔细阅读实验要求，考虑 AOE 网络关键路径的核心思想，并进一步考虑事件的最早开始时间与最晚开始时间等算法要素。列写伪代码，体会算法思想。

（3）利用 CodeBlocks 编译器，配置环境，基于 C++语言将算法用程序实现。

（4）编译运行程序，使用样例进行程序测试，观察所编程序是否实现要求的功能。

（5）考察算法的时间复杂度和空间复杂度，评价算法的优劣，进一步优化程序。

（6）撰写实验报告，进行实验总结与反思。

三、程序设计

（一）抽象数据类型 ADT

图的抽象数据类型：

ADT Graph{

数据对象：V 是具有相同特性的数据元素的集合，称为顶点集。

数据关系：一条弧连接了两个顶点，这些弧组成的集合称为边集。

基本操作：

CreateGraph(&G)

操作结果：创建图

InitGraph(&G)

操作结果：图的初始化

DeleteGraph(&G)

操作结果：图的销毁

InsertVex (&G,v)

操作结果：在图 G 中增添新的顶点

DeleteVex(&G,v)

操作结果：在图 G 中删除顶点

InsertArc (&G,v,w)

操作结果：在图 G 中<v,w>增添新的边<v,w>

DeleteArc(&G,v,w)

操作结果：在图 G 中删除边<v,w>

}**ADT Graph**;

AOE 网络的抽象数据类型：

ADT AOE_Graph{

数据对象：V 是具有相同特性的数据元素的集合，称为顶点集。一般和实际工程相结合，根据实际工程来确定。

数据关系：AOE 活动网络是特殊的图，为带权的有向无环图。

基本操作：

InitAOEGraph(&G)

操作结果：初始化建立一个 AOE 活动网络

CriticalPath(G)

操作结果：输出 AOE 网络的关键路径

}**ADT AOE_Graph**;

（二）存储结构

图的存储结构主要有邻接矩阵和邻接表两种方式。邻接矩阵利用静态二维数组存储结果，邻接表是一种链式存储结构。大多数图都利用邻接表进行存储，本实验中同样采用邻接表的方式。为了建立邻接表，我们还需要建立一些辅助用的数据结构：边、结点、图，具体如下所示：

```
typedef struct Node{
    int to;
    int from;
    Node *link;
}edge;

typedef struct vex{
    char data;
    edge *adj;
}vex;

typedef struct Gnode{
    int n;//节点个数
    int e;//边的个数
    vex vlist[10000];
}Graph;
```

（三）算法简述

从源点到各个顶点，以及从源点到汇点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同，但只有各条路径上所有活动都完成，整个工程才算完成。因此，完成整个工程所需的时间取决于从源点到汇点的最长路径长度，即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做关键路径。下面对求 AOE 网络关键路径算法的思想进行介绍：

AOE 活动网络中有几个关键的核心要点，分别是事件的最早开始时间 ve[],事件的最迟允许开始时间 vl[],活动的最早开始时间 e[],活动的最迟允许开始时间 l[]。下面对各要点进行详细介绍：

（1）事件的最早开始时间 ve[]

事件的最早开始时间的定义是从源点 v_0 到各个顶点 v_i的 ve[j]。求解思路是

从源点 ve[0] = 0 开始，向前递推，公式如下：

$$\boldsymbol{ve[j]}=\mathbf{max}\ \{\boldsymbol{ve[i]}+\boldsymbol{dur}(\langle \boldsymbol{i},\boldsymbol{j}\rangle)\}\quad \langle \boldsymbol{i},\boldsymbol{j}\rangle \in \boldsymbol{T},\boldsymbol{j}=\mathbf{1},\mathbf{2},\ldots,\boldsymbol{n}-\mathbf{1}$$

其中，T 是所有以第 j 个顶点为弧头的集合。

（2）事件的最迟允许开始时间 ve[]

事件的最迟允许开始时间的定义是从汇点 v_{n-1} 到各个顶点 v_i的 vl[i]。求解思路是从汇点 vl[n-1] = ve[n-1] 开始，向后递推，公式如下：

$$\boldsymbol{vl[i]}=\mathbf{min}\ \{\boldsymbol{vl[j]}-\boldsymbol{dur}(\langle \boldsymbol{i},\boldsymbol{j}\rangle)\}\quad \langle \boldsymbol{i},\boldsymbol{j}\rangle \in \boldsymbol{S},\boldsymbol{i}=\boldsymbol{n}-\mathbf{2},\ldots,\mathbf{0}$$

其中，S 是所有以第 i 个顶点为弧尾的集合。

（3）活动的最早开始时间和最迟允许开始时间

求活动 a_k (k=1,2,…,e) 的 e[k] 和 l[k]，可以参考下面的公式：

e[k] = ve[i]

l[k] = vl[j] - dur(<i, j>)

在详细了解各核心要点后，对整个算法具体流程进行介绍，其可以分为五步：

（1）通过邻接表将建立一个 AOE 活动网络。

（2）将 ve[]数组初始化为 0，从第一个节点开始，逐步求解每个节点的最早开始时间 ve[]。

（3）将 vl[]数组初始化为 ve[]中的最后一个元素，从倒数第二个节点开始，逐步向前求解每个节点的最迟允许开始时间 vl[]。

（4）根据节点的 ve[]和 vl[]数组，求出每个边（也就是每项活动）的最早开始时间 e[]和最迟允许开始时间 l[]。

（5）上述二者相等的活动就是关键活动，那么自然也就得到了关键路径。

（四）程序代码

为保证实验报告的清晰和可读性，将源代码以**附录形式**附于文末。

四、调试分析

（一）调试过程和主要错误

本次实验编写的求 AOE 网络关键路径算法在整体思路上都较为清晰，在调试过程中也没有出现很多问题。需要特别注意的是要区分节点的序号和数值。我在邻接表中存储的都是各节点的序号，因此初代程序计算出的都是以节点序

号为顺序的关键路径，与样例进行对比后发现并不相符，经过检查与调试，我发现只要改用节点的值为顺序来描述关键路径就可以得到正确的结果，在对程序进行修改后，成功解决了这一问题

（二）时间复杂度

对于求 AOE 网络关键路径算法的时间复杂度，在拓扑排序求 ve[i]和逆拓扑排序求 vl[i]时，时间复杂度为 O(n+e)，求 e[k]和 l[k]时，时间复杂度为 O(e)。因此总的时间复杂度仍为 O(n+e)。如果一个 AOE 网络存在多条关键路径，任一关键活动的加速不能使整个工程提前完成，只有提高各个关键路径上的所有关键活动才可以使整个工程提前完成。

五、程序测试

在完成全部程序编写后，输入测试样例进行测试，由于使用邻接表的存储结构，需要将 AOE 活动网络的各条边的信息输入进去。尤其需要注意的是，节点的序号和值是不同的，在本实验中，序号为 i 的节点对应的值为 i+1，因此在输入节点时，要输入节点序号而不是节点的值。具体输入如下：

```
0 1 6
0 2 4
1 4 1
2 4 1
4 6 9
4 7 7
6 8 2
7 8 3
0 3 5
3 5 2
5 7 4
```

求 AOE 网络关键路径算法测试结果见下图 2：

```
输入所求的AOE网络:
0 1 6
0 2 4
1 4 1
2 4 1
4 6 9
4 7 7
6 8 2
7 8 3
0 3 5
3 5 2
5 7 4
关键路径:
<1,2>
<2,5>
<5,7>
<7,9>
```

图 2 求 AOE 网络关键路径算法测试结果

通过测试结果可以看出，程序运行符合预期。将其在 AOE 网络中表示出来，所得结果如图 3：

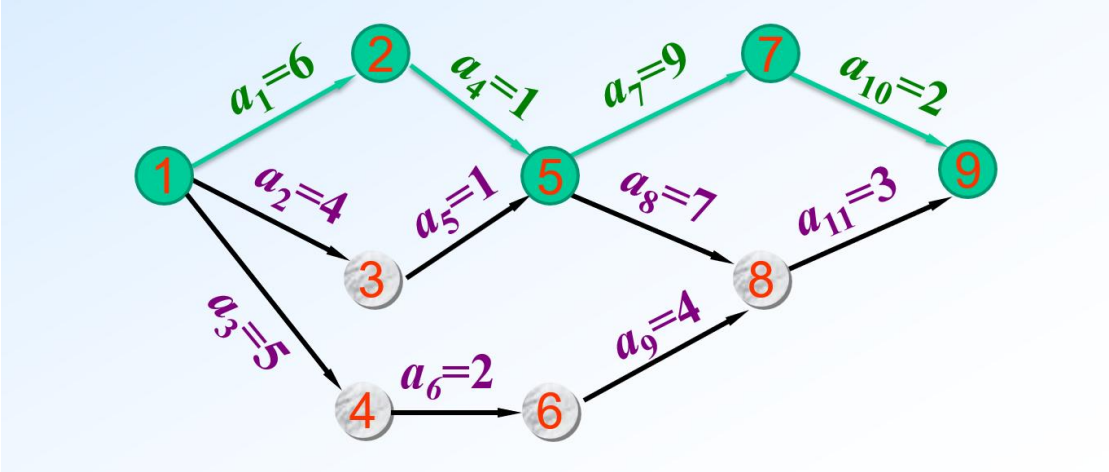

图 3 实际关键路径

六、实验总结

通过这次试验，我发现我对求 AOE 网络关键路径算法的理解不够深入全面，需要不断巩固学习，加深理解。图为非线性结构，在存储过程中，有邻接表与邻接矩阵两种选择，由于使用邻接表进行存储，遍历时更加容易也更为灵活，故本次实验选择邻接表这一存储结构。在实验过程中，我对关键路径中 ve[]与 vl[]两个数组掌握得更加清楚，对于最早开始时间与最迟允许开始时间有了更加实际的理解，两者求解过程上十分类似，只存在方向上的差异，两者都是动态更新的算法，利用 for 循环就可以解决。除此之外，在本次实验过程中，编写、调试程序花费了很长时间，使我对空指针的理解更加深刻，我最初以为 Segmentation Fault 类型报错是因为数组越界，经过不断查阅资料发现是因为调用了空指针，空指针不会指向任何实体，因此在程序编写过程中需要格外注意各指针变量指向的变化，在 delete 操作完成后，最好在后面加一行将指针置为 NULL 的代码，这可以有效避免调用空指针的错误。在本次实验后，我还需要精益求精，不断改进程序，优化函数性能，实现预期目标与功能所需。

附录：程序源代码：求 AOE 网络关键路径算法

```
#include<iostream>
using namespace std;

//邻接表数据结构
typedef struct Node{
    int to;
    int from;
    struct Node * link;
}edge;

typedef struct vex{
    int data;
    edge* adj;
}vex;

typedef struct Gnode{
    int n;//节点个数
    int e;//边的个数
    vex vlist[10000];
}Graph;

//建立邻接表
void InitGraph(Graph &G){
    G.n = 9;
    G.e = 11;
    cout << "输入所求的 AOE 网络: " << endl;
    for(int i = 0;i < G.n;i++){
        G.vlist[i].data = i + 1;
        G.vlist[i].adj = NULL;
    }
    for(int i = 0;i < G.e;i++){
        int tail,head,weight;
        cin >> tail >> head >> weight;
        edge* p = new edge;
        p->from = weight;
        p->to = head;
        p->link = G.vlist[tail].adj;
        G.vlist[tail].adj = p;
    }
}

void CritialPath(Graph G){
```

```
    int ve[G.n];
    int vl[G.n];
    //正向计算事件的最早开始时间
    cout << "关键路径: " << endl;
    for(int i = 0;i < G.n;i++) ve[i] = 0;
    for(int i = 0;i < G.n;i++){
        edge* p = G.vlist[i].adj;
        while(p != NULL){
            int k = p->to;
            if(ve[i] + p->from > ve[k]) ve[k] = ve[i] + p->from;
            p = p->link;
        }
    }
    //逆向计算事件的最晚开始时间
    for(int i = 0;i < G.n;i++) vl[i] = ve[G.n-1];
    for(int i = G.n-2;i > 0;i--){
        edge* p = G.vlist[i].adj;
        while(p != NULL){
            int k = p->to;
            if(vl[k] - p->from < vl[i]) vl[i] = vl[k] - p->from;
            p = p->link;
        }
    }
    //判断并打印关键路径
    for(int i = 0;i < G.n;i++){
        edge* p = G.vlist[i].adj;
        while(p != NULL){
            int k = p->to;
            if(ve[i] == vl[k] - p->from) cout << "<" << i + 1 << "," <<
k + 1 << ">" << endl;
            p = p->link;
        }
    }
}

int main(){
    Graph G;
    InitGraph(G);
    CritialPath(G);
}
```