同濟大學
TONGJI UNIVERSITY

项目说明文档

关键活动

指导教师：张颖

1751984 王舸飞

1.分析

1.1 背景分析

在工程任务调度问题中，如果给出了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，但有些任务必须准时完成，否则整个项目的工期就要因此延误，这些任务叫“关键活动”。

本次设计的要求就是给定一个工程的项目完成时间和交界点需求，首先判断其是否可行，可行后再判断完成整个项目所需的最短时间，并且要输出所有的关键活动。

1.2 功能分析

给出的输入分为两部分，第一部分独自占据一行，包含两个数，分别代表任务交接点的个数和任务的个数，首先应先判断它们是否合法。

再进行完合法性判断后将给出一系列任务的开始节点，完成节点和持续时间，在这里值得注意的是给出的任务已经是拓扑有序的，或者说，如果给出的任务不满足拓扑有序要求，即完成节点的值大于开始节点，那么它必然是不可行的任务，此时可以直接输出0。

如果通过了上述一系列判断，将调用生成关键活动的函数输出。输出同样应分为两部分，第一部分为完成该任务所需的最短时间，第二部分为按照格式输出的一系列关键活动的开始节点和结束节点。

2.设计

2.1 数据结构设计

对于关键活动这种多个任务组织的模型，用图的结构进行存储是最好的选择。

2.2 类结构设计

本次给出的任务存在先后顺序之分，同时不同活动之间的路径有权重，故将图定义为带权有向图。由于要添加边和节点，为了插入和删除的高效，应用邻接表存储。

2.3 成员与操作设计

边结构体(Edge)

```
struct Edge{
    int another;//这条边的另一个节点名称
    int cost;//这条边的权值
    Edge *link;//下一条边
    Edge(int num,int weight):another(num),cost(weight),link(NULL){};//构造函数
};//边的初始化定义
```

节点结构体(Vertex)

```
struct Vertex{
     Edge *first_edge;//顶点的第一条边
     Vertex(){first_edge=NULL;}//构造函数
};//结点的初始化定义
```

带权有向邻接表类(Link_Graph)

```
class Link_Graph{
public: Link_Graph(int p_length){length=p_length;}//构造函数
    bool InitEdges(int edges);//初始化节点，即生成邻接表
    void InsertEdge(int v1,int v2,int cost);//给出两条边的位置和权值将其插入邻接表
    int Getlength(){return length;}//返回邻接表的长度
    int GetFirstNeighbor(int v);//获得一个节点的第一个边结点位置
    int GetNextNeighbor(int v,int w);//在v节点的边中，获得位置为w节点右边的节点位置
    int GetWeight(int v1,int v2);//给出两节点位置，返回它们的边权重
    void Clear(){for(int i=0;i<length;i++)NodeTable[i].first_edge=NULL;} //将表的边刷新
    void CriticalPath();//求关键路径
    void Debug();
private:Vertex NodeTable[100];//邻接表数组的指针
    int length;//代表邻接表的长度
};
```

2.4 系统设计

程序运行之后，系统会首先接收用户第一行的输入，即节点的个数和任务的个数，之后判断它们的合法性，只有在合法时才进行下一步。之后，系统会按照之前输入的节点个数创建一个长度为节点个数的邻接表，并调用初始化函数接收用户对于边的输入。

对于边的输入函数返回值为一个布尔变量，如果他为0代表不可行，直接输出0，否则调用关键活动函数输出求得的路径长度和关键活动。

2.5 类定义说明

本次采用的邻接表类同第八题——电网造价问题的邻接表几乎相同，作出的几点微小改动如下：

1、定义节点时将节点的名称删去了，因为这里给出的节点直接为拓扑排序中的序列值，它必为整数，故不需要一个字符型变量进行存储。

与之对应的，根据名称取地址和根据地址取名称的函数也失去了意义，将它们删除。

2、本题中直接给出了节点个数和边个数，同时将节点编号和边权重一起输入，首先输入函数有了终点，其次可以将节点的输入函数和边的输入函数合二为一。

3、由于本题使用的是有向图，故在插入一条边时仅仅将后输入的节点挂到先输入的节点中即可。

值得注意的是，本题给出的拓扑排序序列是从1开始的，故在插入时应将节点编号减一。

同理，在输出的时候也应将节点的编号加一。

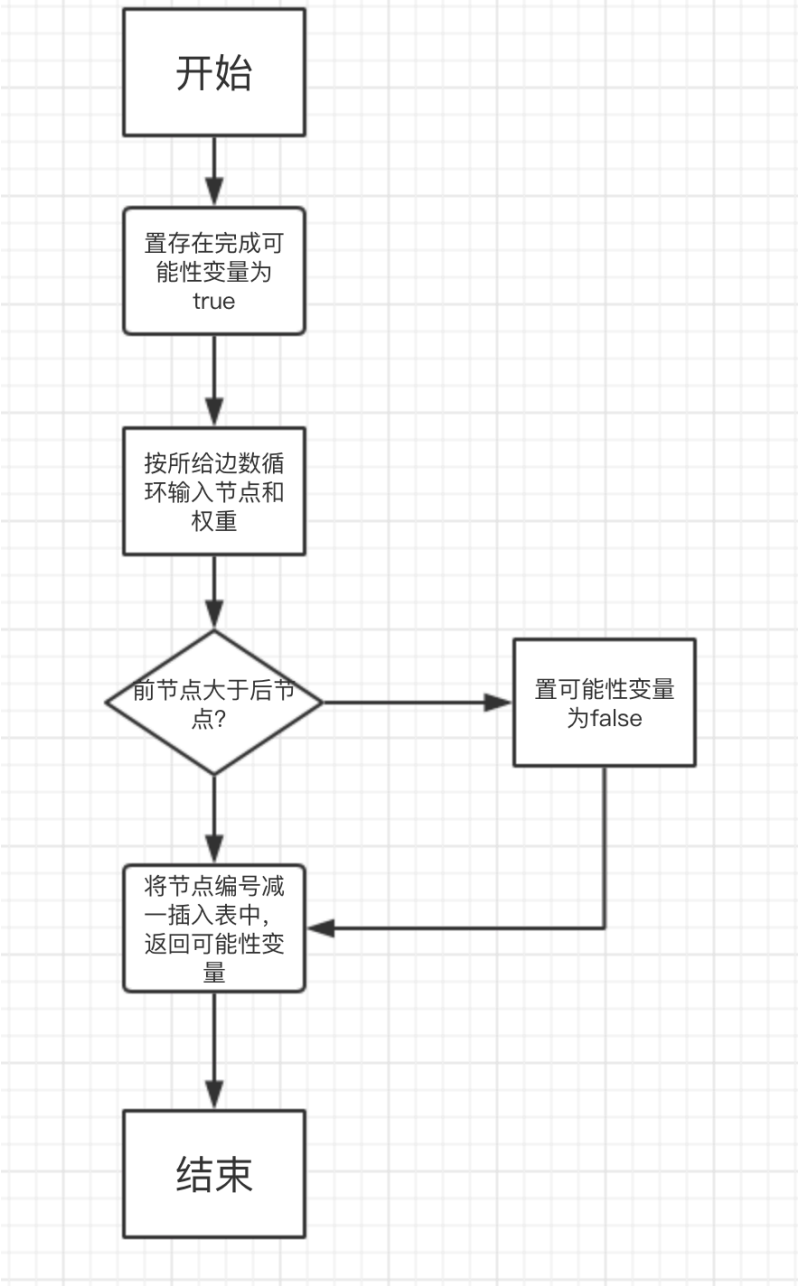
除了上述几点差异，还有一些关键性算法的不同，将它们补充在“实现”部分。

邻接表的插入、寻找、获得权重的函数已经在第八题中有所描述，下面不再赘述了。

3.实现

3.1 初始化表算法的实现

3.1.1 初始化表流程图



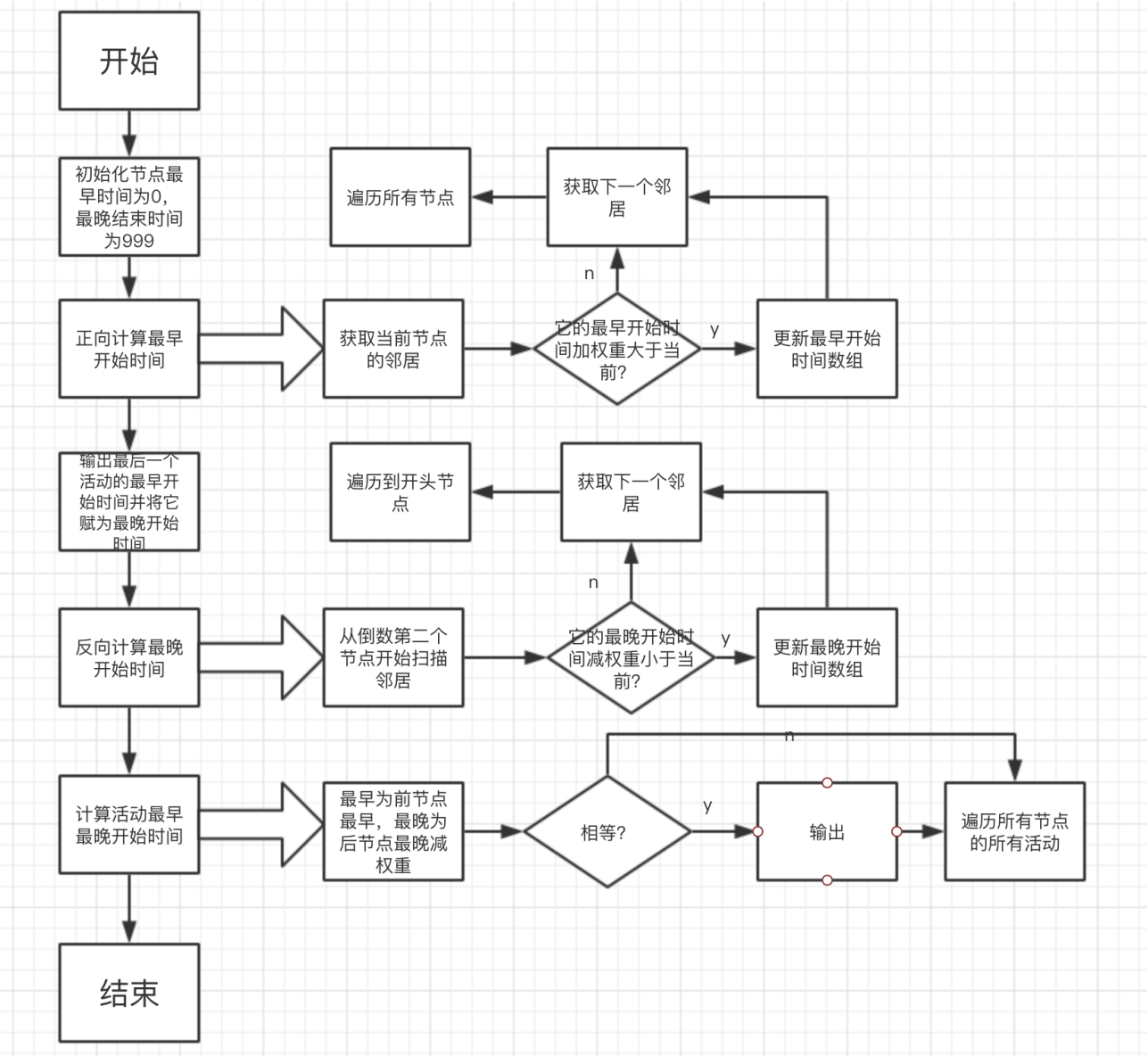
3.1.2 初始化表核心代码

```
bool Link_Graph::InitEdges(int edges){
    bool ok=true;
    int v1,v2,cost;

    for(int i=0;i<edges;i++){
        cin>>v1>>v2>>cost;
        if(v1>v2)ok=false;
        InsertEdge(v1-1,v2-1,cost);
    }
    return ok;
}
```

3.2 求关键活动功能的实现

3.2.1 求关键活动流程图



3.2.2 求关键活动核心代码

```
void Link_Graph::CriticalPath(){
    int Ve[length];
    int Vl[length];
    int i,j,k,weight,Ae,Al;

    for(i=0;i<length;i++)Ve[i]=0;
    for(i=0;i<length;i++)Vl[i]=999;

    for(i=0;i<length;i++){//正向计算最早开始时间

        j=GetFirstNeighbor(i);


        while(j!=-1){
            weight=GetWeight(i,j);
            if(Ve[i]+weight>Ve[j])Ve[j]=Ve[i]+weight;
            j=GetNextNeighbor(i,j);
        }

    }

    Vl[length-1]=Ve[length-1];

    cout<<Vl[length-1]<<endl;//输出最后一个活动的最早开始时间即最短时间

    for(j=length-2;j>0;j--){//反向计算最晚开始时间
        k=GetFirstNeighbor(j);
        while(k!=-1){
            weight=GetWeight(j,k);
            if(Vl[k]-weight<Vl[j])Vl[j]=Vl[k]-weight;
            k=GetNextNeighbor(j,k);


        }


    }



    for(i=0;i<length;i++){//计算活动的时间并判断输出
        j=GetFirstNeighbor(i);
        while(j!=-1){

            Ae=Ve[i];Al=Vl[j]-GetWeight(i,j);
            if(Ae==Al){
                cout<<i+1<<"->"<<j+1<<endl;
            }
            j=GetNextNeighbor(i,j);

        }
    }
```

4.测试

4.1 常规测试

4.1.1 简单情况测试

实验结果：

```
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
```

4.1.2 一般情况测试，单个起点和单个终点

实验结果：

```
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
```

4.1.3 不可行的方案测试

实验结果：

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