作业 HW4 实验报告

姓名：刘博洋 学号：2153538 日期：2022 年 11 月 24 日

1. 涉及数据结构和相关背景

图结构是一种很重要的非线性的数据结构，相对于树而言，图更广泛更普适，树是一个前驱和多个后继，而图可以有多个前驱和后继，图在实际生活中有很多例子，比如交通运输网，地铁网络，社交网络，计算机中的状态执行（自动机）等等都可以抽象成图结构。
图可以分为有向图和无向图，存储结构一般为邻接表、邻接矩阵等等，涉及到的主要算法有图的遍历（深度、广度优先搜索）、最小生成树问题、拓扑排序、关键路径、最短路径等等。

2. 实验内容

2.1 图的遍历

2.1.1 问题描述

本题给定一个无向图，用邻接表作存储结构，用 dfs 和 bfs 找出图的所有连通子集。
所有顶点用 0 到 n-1 表示，搜索时总是从编号最小的顶点出发。使用邻接矩阵存储，或者邻接表（使用邻接表时需要使用尾插法）。

2.1.2 基本要求

输入
第 1 行输入 2 个整数 n m，分别表示顶点数和边数，空格分割
后面 m 行，每行输入边的两个顶点编号，空格分割
输出
第 1 行输出 dfs 的结果
第 2 行输出 bfs 的结果
连通子集输出格式为{v11 v12 ...}{v21 v22 ..}... 连通子集内元素之间用空格分割，子集之间无空格,'{'和子集内第一个数字之间、'}'和子集内最后一个元素之间、子集之间均无空格
对于 20%的数据，有 0<n<=15；
对于 40%的数据，有 0<n<=100；
对于 100%的数据，有 0<n<=10000；
对于所有数据，0.5n<=m<=1.5n，保证输入数据无错。
下载 p107_data.cpp，编译运行以生成随机测试数据

2.1.3 数据结构设计

本题用邻接表作为存储结构，首先根据输入边和顶点的数据建立邻接表，首先用顺序存储顶点信息，然后对于每次输入的边，分别找到其在顺序表中的位置并把另一顶点加入其顶点链表中。
邻接表建立完成后，对图分别进行深度优先搜索和广度优先搜索。
深度优先搜索的思路是，先遍历当前顶点 v，再遍历依次从 v 未被访问的邻接点 u 出发，递归执行 DFS(u)，直至所有从 v 有路径可达的顶点都已被访问过，开始回溯，再次选择未被访问的另一顶点作为起点执行 DFS。这是一种递归算法。
而 BFS 是一种非递归算法，其主要思路是遍历到当前节点后，遍历当前节点所有的邻接点，然后再依次遍历每一个邻接点的邻接点。
本题的难点其实是在括号的输出上，对于 DFS，可以采用先输出空格再输出值的方式，关键在于如何使第一个字符不输出空格。可以用一个计数器，在每次访问

一个联通区域的时候开始计数，然后访问完后置零，当计数器访问联通区域的第一个元素的时候不输出空格，其他情况都输出。
对于 BFS，由于采用的是队列的方式，所以在访问到一个联通域的最后一个元素时不输出空格，具体为判断当访问某个元素时，队列中所有元素的邻接点是否都不存在。

2.1.4 功能说明（函数、类）

```
#include<iostream>
#include<iomanip>
#include<queue>
using namespace std;
typedef struct ArcNode {
    int adjvex;
    struct ArcNode* nextarc;//邻接节点
}ArcNode;
typedef struct VNode {
    int data;
    ArcNode* firstarc;
}VNode,Adjlist[10000];//顶点结构
typedef struct {
    Adjlist vertices;
    int vexnum, arcnum;
}ALGraph;//图的类型定义
ALGraph G;
int visit[10000] = { 0 };
queue<int> Q;
int cnt = 0;
int locateVex(ALGraph G, int a)
{
    int i;
    for ( i = 0;i < G.vexnum;i++)
    {
        if (a == G.vertices[i].data)
            return i;
    }
    if (i == G.vexnum)
    {
        cout << "ERROR!";
        exit(1);
    }
    return 0;
}
void input_createadjlist(ALGraph& G)
{
```

```
cin >> G.vexnum >> G.arcnum;
for (int i = 0;i < G.vexnum;i++)
{
    G.vertices[i].data = i;
    G.vertices[i].firstarc = NULL;
}
for (int k = 0;k < G.arcnum;k++)
{
    int v1, v2;
    cin >> v1 >> v2;
    ArcNode* t1, *t2;
    ArcNode* p;
    p = (ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex = v2;
    p->nextarc = NULL;
    t1 = G.vertices[v1].firstarc;
    if (t1 == NULL)
    {
        G.vertices[v1].firstarc = p;
    }
    else
    {
        while (t1->nextarc != NULL)
        {
            t1 = t1->nextarc;
        }
        t1->nextarc = p;
    }
    ArcNode* q;
    q= (ArcNode*)malloc(sizeof(ArcNode));
    q->adjvex = v1;
    q->nextarc = NULL;
    t2 = G.vertices[v2].firstarc;
    if (t2 == NULL)
    {
        G.vertices[v2].firstarc = q;
    }
    else
    {
        while (t2->nextarc != NULL)
        {
            t2 = t2->nextarc;
        }
        t2->nextarc = q;
```

```
        }


    }
    return;
}
void dfs(ALGraph& G,int x)
{
    ArcNode* p;
    if(cnt==0)
    cout << G.vertices[x].data;
    else cout << " "<<G.vertices[x].data;
    cnt++;
    visit[x] = 1;
    p = G.vertices[x].firstarc;
     /*if (p != NULL)
        cout << " ";*/
    while (p)
    {
        if (visit[p->adjvex] == 0)
        {
            dfs(G, p->adjvex);
        }
        p = p->nextarc;
    }
}//dfs 算法
void DFStraverse(ALGraph G)
{
    int num=0;
    cout << "{";
    for (int i = 0;i < G.vexnum;i++)
    {
        visit[i] = 0;
    }
    for (int i = 0;i < G.vexnum;i++)
    {
        if (visit[i] == 0)
        {
            if (num >= 1)
            {
                cout<<"}{";
            }
            num++;
            dfs(G, i);
```

```
        }
        cnt = 0;//计数器置零
    }
    cout  << "}";//遍历整张图
}
void bfs(ALGraph &G,int v)
{
    Q.push(v);
    visit[v] = 1;
    while (Q.empty() != 1)
    {
        v = Q.front();
        Q.pop();
        cout << G.vertices[v].data;
        ArcNode* p = G.vertices[v].firstarc;

        while (p)
        {
            if (visit[p->adjvex] == 0)
            {
                Q.push(p->adjvex);
                visit[p->adjvex] = 1;
            }
            p = p->nextarc;
        }
        queue<int> T = Q;
        int flag = 0;
        while (T.empty() == 0)
        {
            ArcNode* t = G.vertices[T.front()].firstarc;
            if (t != NULL)
                flag = 1;
            T.pop();
        }
        if (flag != 0)
            cout << " ";
    }
}
void  BFSTraverse(ALGraph& G)
{
    cout << "{";
    int num=0;
    for (int i = 0;i < G.vexnum;i++)
    {
```

```
        if (visit[i] == 0)
        {
            if (num >= 1)
                cout << "}{";
            bfs(G, i);
            num++;
        }


    }
    cout << "}";
}
int main()
{
    input_createadjlist(G);
    DFStraverse(G);
    for (int i = 0;i < G.vexnum;i++)
    {
        visit[i] = 0;
    }
    cout << endl;
    BFSTraverse(G);
    cout << endl;
    return 0;
}
```

2.1.5 调试分析（遇到的问题和解决方法）

本题难度不算很大，主要遇到的问题是要理解 BFS 和 DFS 的基本思路，难点就是对于括号和空格的控制，刚开始是采用 cout<<’ ’ <<’\b’<<},即在每一个连通域最后输出一个退格键再输出括号，以此来控制每部分括号与最后一个数据直接没有空格，这种方法在 c++控制台中是有效的，因为\b 的功能就是使光标往前退一格，但是当把输出结果重定向到 txt 时，就发现\b 变成了一个特殊形状的字符，表示这种方法并不可用。于是后来进行分析，决定采用先输出空格再输出数据，并且判断使每个联通域第一个元素不输出空格。

2.1.6 总结和体会

本题是学习图这种数据结构的一个入门基础题，让我比较清楚地了解和巩固了 DFS 和 BFS 的具体思想,并学会如何根据具体情况对输入输出做更加精确的处理。

2.2 小世界现象

2.2.1 问题描述

六度空间理论又称小世界理论。理论通俗地解释为：“你和世界上任何一个陌生人之间所间隔的人不会超过 6 个人，也就是说，最多通过五个人你就能够认识任何一个陌生人。”

假如给你一个社交网络图，请你对每个节点计算符合“六度空间”理论的结点占结点总数的百分比。

说明：由于浮点数精度不同导致结果有误差，请按 float 计算。

2.2.2 基本要求

输入
第 1 行给出两个正整数，分别表示社交网络图的结点数 N（1<N≤2000，表示人数）、边数 M（≤33×N，表示社交关系数）。
随后的 M 行对应 M 条边，每行给出一对正整数，分别是该条边直接连通的两个结点的编号（节点从 1 到 N 编号）。
输出
对每个结点输出与该结点距离不超过 6 的结点数占结点总数的百分比，精确到小数点后 2 位。每个结节点输出一行，格式为“结点编号:（空格）百分比%”。

2.2.3 数据结构设计

本题实际上是用到了层次遍历的思想或者 BFS 的思想。
本题还是选择用邻接表作为储存结构，将每个节点和它的邻接点存入顶点顺序表和邻接点链表中，对于每个点执行 BFS，即先访问每个节点，再顺次访问每个节点的所有邻接点，这里需要用到队列，遍历完每一层后队列为空然后下一层的第一个节点，然后对每一层进行记录，对遍历到的点进行计数，最后用计数得到的点除以总点数便得到百分比。

2.2.4 功能说明（函数、类）

```
#include <iostream>
#include <cstring>
#include <queue>
#include <iomanip>

using namespace std;

const int maxn = 2000;
typedef struct ArcNode {
    int adjvex;
    struct ArcNode* nextarc;
}ArcNode;
typedef struct VNode {
    int data;
    ArcNode* firstarc;
}VNode, Adjlist[10000];
typedef struct {
    Adjlist vertices;
    int vexnum, arcnum;
}ALGraph;
ALGraph G;
int visit[10000] = { 0 };
void input_createadjlist(ALGraph& G)
{
    cin >> G.vexnum >> G.arcnum;
    for (int i = 0;i < G.vexnum;i++)
    {
        G.vertices[i].data = i;
        G.vertices[i].firstarc = NULL;
```

```
}
for (int k = 0;k < G.arcnum;k++)
{
    int v1, v2;
    cin >> v1 >> v2;
    ArcNode* t1, * t2;
    ArcNode* p;
    p = (ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex = v2;
    p->nextarc = NULL;
    t1 = G.vertices[v1].firstarc;
    if (t1 == NULL)
    {
        G.vertices[v1].firstarc = p;
    }
    else
    {
        while (t1->nextarc != NULL)
        {
            t1 = t1->nextarc;
        }
        t1->nextarc = p;
    }
    ArcNode* q;
    q = (ArcNode*)malloc(sizeof(ArcNode));
    q->adjvex = v1;
    q->nextarc = NULL;
    t2 = G.vertices[v2].firstarc;
    if (t2 == NULL)
    {
        G.vertices[v2].firstarc = q;
    }
    else
    {
        while (t2->nextarc != NULL)
        {
            t2 = t2->nextarc;
        }
        t2->nextarc = q;
    }


}
return;
```

```
}//用邻接表存储图的信息
int BFS(int i)
{
    queue <int> q;
    visit[i] = 1;
    q.push(i);//用队列存储每一层的元素
    int count1 = 1;//count 对每一个节点遍历到的元素进行计数
    int level = 0;//记录层次
    int last = i;
    int tail;
    while (!q.empty())
    {
        int t = q.front();
        q.pop();
        ArcNode* p = G.vertices[t].firstarc;
        ArcNode* m = p;
        while (p)
        {

            if (visit[p->adjvex] == 0)
            {
                q.push(p->adjvex);
                visit[p->adjvex] = 1;
                count1++;
                tail = p->adjvex;
            }
            p = p->nextarc;
        }
        if (t == last)
        {
            level++;
            last = tail;
        }
        if (level == 6)
            break;//达到 6 层退出

    }
    return count1;
}
int main()
{
    input_createadjlist(G);

    int ans = 1;
```

```
    for (int i = 1;i <= G.vexnum;i++)
    {
        for (int j = 0;j <= G.vexnum;j++)
            visit[j] = 0;
        ans = BFS(i);
        cout << i << ": " << setiosflags(ios::fixed) << setprecision(2) << (ans
* 100.00) /G.vexnum << "%" << endl;
    }

    return 0;
}
```

2.2.5 调试分析

本题调试过程中没有遇到太多问题，主要是对于层数的判断需要用 tail,last,等等几个变量来描述，编程时要弄清楚每个变量的含义即 BFS 的执行过程。

2.2.6 总结和体会

本题是对于 BFS 更深层次的理解，BFS 相当于一种层序遍历的思想，每遍历一层距离增加 1，利用这种性质，我们可以更好的求解本题。在这题中我不但加深了对于 BFS 搜索算法的过程的理解，更对于层数的判断条件理解的更加透彻。

2.3 村村通

2.3.1 问题描述

N 个村庄，从 1 到 N 编号，现在请你修建一些路使得任何两个村庄都彼此连通。我们称两个村庄 A 和 B 是连通的，当且仅当在 A 和 B 之间存在一条路，或者存在一个村庄 C，使得 A 和 C 之间有一条路，并且 C 和 B 是连通的。

已知在一些村庄之间已经有了一些路，您的工作是再兴建一些路，使得所有的村庄都是连通的，并且新建的路的长度是最小的。

2.3.2 基本要求

输入

第一行包含一个整数 n（3<=n<=100），表示村庄数目。

接下来 n 行,每行 n 个非负整数，表示村庄 i 和村庄 j 之间的距离。距离值在[1,1000]之间。

接着是一个整数 m，后面给出 m 行，每行包含两个整数 a,b,(1<=a<b),表示在村庄 a 和 b 之间已经修建了路。

下载编译并运行 p134.cpp 生成随机测试数据

输出

输出一行，仅有一个整数，表示为使所有的村庄连通，要新建公路的长度的最小值。

2.3.3 数据结构设计

本题求使所有村庄联通的新建公路最小值，实际上是一种最小生成树的问题。而本题每两个村庄直接都是相互关联的，所以是一种稠密图，而对于稠密图而言，Prim 算法要比 Kruskal 算法更加高效，所以本题选用 prim 算法。

Prim 算法的基本过程是从一个集合 V 中的元素出发，每次选取 U-V 中元素到 V 中元素的最小权值，把这条边加入生成树中，然后把节点标记为已访问。所以我们建立一个辅助数组 closedge，有两个数据域，一个数据域是 is_visit，代表判断各个节点是否在被选中的集合 V 内；另一个数据域是 lowcost，存储并更新每次 U-V 中每个元素到 V 中元素的最小权值。输入的时候，开始输入的便是代表每两个点之间距离的距离矩阵，然后对于已经存在的边，

令二者之间距离为0，初始化closedge时，先令所有lowcost等于其他点到第一个点的距离,除了第一个点的is_visit为1外其他都为0,遍历到一个权值最小的节点后,便用sum+=权值记录总最小长度。

2.3.4 功能说明（函数、类）

```
#include<iostream>
#include<iomanip>
using namespace std;
int sum = 0;
#define inf 99999//用很大的数表示无穷
typedef int AdjMatrix[100][100];//距离矩阵
typedef struct  {
    int vex[100];
    AdjMatrix arcs;
    int vexnum, arcnum;
}MGraph;
MGraph G;
 struct {
    int is_visit;//记录是否被访问
    int lowcost;
}closedge[100];
void input()
{
    cin >> G.vexnum;
    for (int i = 0;i < G.vexnum;i++)
        G.vex[i] = i+1;
    for (int i=0;i < G.vexnum;i++)
    {
        for (int j = 0;j < G.vexnum;j++)
            cin >> G.arcs[i][j];
    }
    cin >> G.arcnum;
    for (int i = 0;i < G.arcnum;i++)
    {
        int t1, t2;
        cin >> t1 >> t2;
        G.arcs[t1 - 1][t2 - 1] = 0;//已经有的道路令二者之间距离为0
        G.arcs[t2 - 1][t1 - 1] = 0;
    }
}
void MST_PRIM(MGraph& G)
{
    int u;
    for (int i = 0;i < G.vexnum;i++)
    {
```

```
        closedge[i].is_visit = 0;
        closedge[i].lowcost = G.arcs[0][i];
    }//初始化

    closedge[0].is_visit = 1;//从第一个点开始
    for (int i = 0;i < G.vexnum-1;i++)//遍历 n-1 次
    {
        int min = inf;

        for (int j = 0;j < G.vexnum;j++)
        {
            if (closedge[j].is_visit == 0 && min > closedge[j].lowcost)
            {
                min = closedge[j].lowcost;
                u = j;
            }
        }
        closedge[u].is_visit = 1;
        sum += closedge[u].lowcost;
        for (int m = 0;m < G.vexnum;m++)
        {
            if (closedge[m].is_visit == 0 && closedge[m].lowcost > G.arcs[u][m])
            {
                closedge[m].lowcost = G.arcs[u][m];
            }
        }
    }

}
int main()
{
    input();
    MST_PRIM(G);
    cout << sum;
    return 0;
}
```

2.3.5 调试分析

本题一开始经历了一些曲折，首先我开始的想法是用已经修建好道路的 m 个点构成集合 V，从 V 中任选一个点开始遍历，但是这样代码实现好像相对复杂，调试了一段时间也没有找出问题所在，可能是每次更新的时候没有把所有点的最短权值进行更新的原因。然后便比较巧妙的使已经修建好道路的点直接距离为 0，但是还是从头开始遍历每个村庄，距离为 0 的点也不会加入 sum 中，并且每次更新 lowcost 只要更新访问的元素相关的元素，仍然可以求得正确结果。

2.3.6 总结与收获

本题也可以用 kruskal 算法解决，但是其更适合稀疏矩阵，本题让我对 prim 算法和 kruskal 算法都有了不错的复习和巩固，对二者的差异性、适用条件也有了更深的理解。更重要的是我掌握了一种等效替代的方法，将已经存在道路的村庄之间距离置零，转变思路来解决问题，

2.4 给定条件构造矩阵

2.4.1 题目描述

给你一个 正 整数 k ，同时给你：

一个大小为 n 的二维整数数组 rowConditions ，其中 rowConditions[i] = [abovei, belowi] 和一个大小为 m 的二维整数数组 colConditions ，其中 colConditions[i] = [lefti, righti] 。

两个数组里的整数都是 1 到 k 之间的数字。

你需要构造一个 k x k 的矩阵，1 到 k 每个数字需要 恰好出现一次 。剩余的数字都是 0 。

矩阵还需要满足以下条件：

对于所有 0 到 n - 1 之间的下标 i ，数字 abovei 所在的 行 必须在数字 belowi 所在行的上面。

对于所有 0 到 m - 1 之间的下标 i ，数字 lefti 所在的 列 必须在数字 righti 所在列的左边。

返回满足上述要求的矩阵，题目保证若矩阵存在则一定唯一；如果不存在答案，返回一个空的矩阵。

2.4.2 基本要求

输入：

第一行包含 3 个整数 k、n 和 m

接下来 n 行，每行两个整数 abovei、belowi，描述 rowConditions 数组

接下来 m 行，每行两个整数 lefti、righti，描述 colConditions 数组

输出：

如果可以构造矩阵，打印矩阵；否则输出-1

矩阵中每行元素使用空格分隔

2.4.3 数据结构设计

本题的关键在于分辨问题的类型，乍一看本题好像没有明示应该用什么算法和数据结构去完成，但是我们可以发现本题条件包含的信息是整个矩阵中只有部分元素之间存在相互位置约束关系，但是我们需要根据这些条件构造出一个确定的矩阵。由拓扑排序的定义，由某个集合上的一个偏序得到该集合上的一个全序，简单来说，偏序是指集合中仅有部分元素之间可比较（有关系），而全序是指集合中全体成员之间均可比较。本题恰好是要求我们实现从偏序到全序的过程，下面的问题在于对于二维的矩阵，如何根据位置关系进行拓扑排序。

本题我采用的方法是分别对行和列进行拓扑排序，把位置关系看成一张图，如果元素 a 在元素 b 上方，那么认为 a 到 b 有一条有向边，然后对入度为 0 的点进行访问，实现列的拓扑排序，先把所有元素都放在矩阵的最后一列中，按次序排列，再对行进行类似的操作，修改矩阵中元素位置，得到结果矩阵。

2.4.4 功能说明（函数、类）

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

```
typedef struct ArcNode {
    int adjvex;
    struct ArcNode* nextarc;
}ArcNode;
typedef struct VNode {
    int data;
    int indegree;
    ArcNode* firstarc;
}VNode, Adjlist[500];
typedef struct {
    Adjlist vertices;
    int vexnum, arcnum;
}ALGraph;
ALGraph Row,Col;//建立两个图，分别进行拓扑排序
int k;
int rowconditions[10000][2]={0};
int colconditions[10000][2]={0};
int result[500][500]={0};
stack<int> L;
void input()
{
    cin >> k >> Row.arcnum >> Col.arcnum;
    Row.vexnum = Col.vexnum = k;
    for (int i = 1;i <= Row.vexnum;i++)
    {
        Row.vertices[i].data = i;
        Row.vertices[i].firstarc = NULL;
        Row.vertices[i].indegree = 0;
    }//初始化
    for (int i = 1;i <= Col.vexnum;i++)
    {
        Col.vertices[i].data = i;
        Col.vertices[i].firstarc = NULL;
        Col.vertices[i].indegree = 0;
    }
    for (int i = 0;i < Row.arcnum;i++)
    {
        cin >> rowconditions[i][0];
        cin >> rowconditions[i][1];
    }
    for (int j = 0;j < Col.arcnum;j++)
    {
        cin >> colconditions[j][0];
        cin >> colconditions[j][1];
```

```
}
for (int i = 0;i < Row.arcnum;i++)
{
    ArcNode* p,*t;
    p = (ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex = rowconditions[i][1];
    p->nextarc = NULL;
    t = Row.vertices[rowconditions[i][0]].firstarc;
    if (t == NULL)
    {
        Row.vertices[rowconditions[i][0]].firstarc = p;

    }
    else
    {
        while (t->nextarc != NULL)
        {
            t = t->nextarc;
        }
        t->nextarc = p;

    }
    Row.vertices[rowconditions[i][1]].indegree++;
}
for (int i = 0;i < Col.arcnum;i++)
{
    ArcNode* p, * t;
    p = (ArcNode*)malloc(sizeof(ArcNode));
    if (!p)
        return ;
    else
    p->adjvex = colconditions[i][1];
    p->nextarc = NULL;
    t = Col.vertices[colconditions[i][0]].firstarc;
    if (t == NULL)
    {
        Col.vertices[colconditions[i][0]].firstarc = p;
    }
    else
    {
        while (t->nextarc != NULL)
        {
            t = t->nextarc;
        }
```

```
            t->nextarc = p;
        }
        Col.vertices[colconditions[i][1]].indegree++;
    }//分别存入邻接表
}
int topologicalsort(ALGraph& G,bool rl)//rl用来判断对哪张图进行拓扑排序
{
    for (int i = 0;i < G.vexnum;i++)
    {
        if (G.vertices[i].indegree == 0)
        {
            L.push(i);
        }
    }
    int count = 0;
    while (L.empty() == 0)
    {
        int x = L.top();
        L.pop();
        if(rl==0)
        *(result[count]+G.vexnum-1) = x;
        else if (rl == 1)
        {
            int n;
            for ( n = 0;n < G.vexnum;n++)
            {
                if (result[n][G.vexnum - 1] == x)
                {
                    result[n][G.vexnum - 1] = 0;
                    break;
                }

            }
            result[n][count] = x;//修改矩阵元素
        }
        count++;
        ArcNode* p;
        for (p = G.vertices[x].firstarc;p != NULL;p=p->nextarc)
        {
            int k = p->adjvex;
            G.vertices[k].indegree--;
            if (G.vertices[k].indegree == 0)
                L.push(k);
        }//一直找入度为0的点
```

```
    }
    if (count < G.vexnum)
        return -1;//说明有环
    else return 0;
}
int main()
{

    input();
    if (topologicalsort(Row, 0) == -1)
    {
        cout << -1;
        return 0;
    }
    if (topologicalsort(Col, 1) == -1)
    {
        cout << -1;
        return 0;
    }
    for (int i = 0;i < Row.vexnum;i++)
    {
        for (int j = 0;j < Row.vexnum;j++)
            cout << result[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

2.4.5 调试与分析

本题是一个条件和方法比较隐蔽的题目，首先需要通过观察和分析得到本题的基本算法思想。而且本题实际上没有图，如何根据元素之间的关系构造一张有向图，然后再进行拓扑排序，这是对我思维的一种考验和提升。同时，这题对于拓扑排序的知识点巩固的很到位，让我能够自己真正理解拓扑排序的全过程。

调试中遇到的主要问题开始是虽然知道要对于行和列分别拓扑排序，但是却没有想到先把一次排好的顺序存在矩阵中，第二次再做修改，导致我矩阵输出存在一些问题。

2.4.6 总结与收获

快速掌握题目的考点并且能合理构造模型进行解决是一种很重要的能力，本题通篇没有图和拓扑排序，但是却需要利用关系来构造图，这其实也是离散数学中关系的思想运用，所以这启示我在做题时需要对题目进行严密的分析，迅速确定类型然后寻找思路。

2.5 必修课

2.5.1 问题描述

某校的计算机系有 n 门必修课程。学生需要修完所有必修课程才能毕业。

每门课程都需要一定的学时去完成。有些课程有前置课程，需要先修完它们才能修这些课程；而其他课程没有。 不同于大多数学校，学生可以在任何时候进行选课，且同时选课的数量

没有限制。
现在校方想要知道：
从入学开始，每门课程最早可能完成的时间（单位：学时）；
对每一门课程，若将该课程的学时增加 1，是否会延长入学到毕业的最短时间。

2.5.2 基本要求

输入
第一行，一个正整数 n，代表课程的数量。
接下来 n 行，每行若干个整数：
第一个整数为 ti ，表示修完该课程所需的学时。
第二个整数为 ci，表示该课程的前置课程数量。
接下来 ci 个互不相同的整数，表示该课程的前置课程的编号。
该校保证，每名入学的学生，一定能够在有限的时间内毕业。
输出
输出共 n 行，第 i 行包含两个整数：
第一个整数表示编号为 i 的课程最早可能完成的时间。
第二个整数表示，如果将该课程的学时增加 1，入学到毕业的最短时间是否会增加。如果会增加则输出 11，否则输出 00。
每行的两个整数以一个空格隔开。
样例输入

```
5
64 3 2 3 5
48 0
32 1 2
48 0
48 1 4
```

样例输出

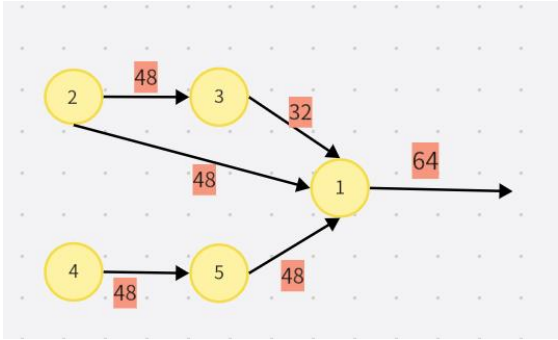
```
160 1
48 0
80 0
48 1
96 1
```

样例解释
一种可能的解释：
1 号课程为“数据结构”，64 学时，前置课程为“程序设计基础” “面向对象程序设计”和“离散数学(2)”；
2 号课程为“程序设计基础”，48 学时，无前置课程；
3 号课程为“面向对象程序设计”，32 学时，前置课程为“程序设计基础”；
4 号课程为“离散数学(1)”，48 学时，无前置课程；
5 号课程为“离散数学(2)”，48 学时，前置课程为“离散数学(1)”。
可以证明，数据结构一定是最后学，因此它对毕业最短时间有影响；而在离散数学(1)(2)和两门程序设计课程中，离散数学需要的学时更多，因此如果增加它们的学时就会延长毕业所需时间。
数据范围与约定

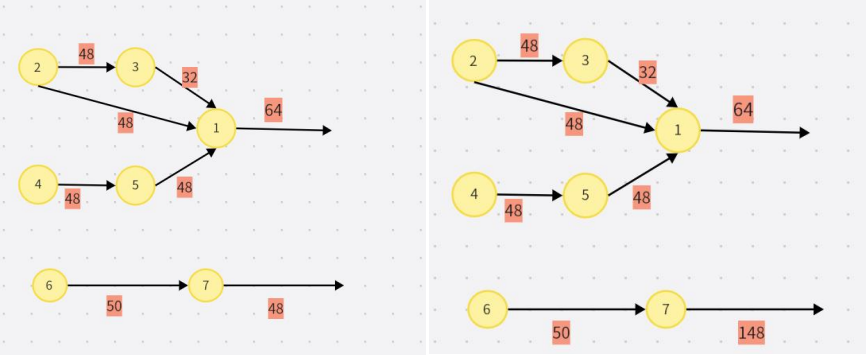
对于所有数据，满足：
1 ≤ n ≤ 100
1 ≤ ti ≤ 100
0 ≤ ci < n

2.5.3 数据结构设计

本题比较明显是求关键活动和关键路径的题目，但是由于没有事先建好的 AOE 网，我们需要把每一门课程当成一个事件，上课的过程当成一个活动，上课时间为边的权值来构造图。
以样例中的数据为例，画出的图如下：



对于这种情况，先采用邻接表进行存储，如果 i 是 j 的前置课程，那么就有一条从 i 到 j 的边，将所有的边和点存入邻接链表。
由于输入的时候顺序是混乱的，无法确定先后顺序，建立入度为 0 顶点栈和拓扑有序顶点栈，进行拓扑排序，排序的同时把各活动的 ve[i]求出来，而 ve[i]初始化时需要加上自身的时间，因为每个节点和它发出的一条边为一个整体。拓扑排序完成后，将 vl[i]全部赋值为 ve 的最大值，然后从每一个出度为 0 的点减去它消耗的时间，求出所有邻接点的最小值即为 vl[i];
在判断关键活动时，只要一个点的最早开始和最迟开始时间相同时，它就为关键活动，这时候要注意一个关键点，由于图中对于最后的节点发出的边没有指向，所以当 p 指向最后节点时，会跳过判断过程，我们需要对齐单独判断，当 p 为空且 p 的最晚开始时间为最大值时，那么 p 所指向的活动为关键活动。这是考虑到了有多个联通分量的情况，6+7 的总时间是 98 小于上方的最长时间 160，但是如果把 7 活动的时间变成 148，那么只有 6 和 7 才是关键活动。



2.5.4 功能说明（函数、类）

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

```
using namespace std;
typedef struct ArcNode {
    int adjvex;
    int time;
    struct ArcNode* nextarc;
}ArcNode;
typedef struct VNode {
    int data;
    int indegree;
    int time;
    ArcNode* firstarc;
}VNode, Adjlist[101];
typedef struct {
    Adjlist vertices;
    int vexnum, arcnum;
}ALGraph;
ALGraph G;
int ve[100], vl[100];//定义事件最早开始时间，最迟开始时间
int e[100], l[100];//定义活动最早开始时间，最迟开始时间
stack<int> L;
stack<int> T;
void input(ALGraph& G)
{
    cin >> G.vexnum;
    for (int i = 1;i <= G.vexnum;i++)
    {
        G.vertices[i].data= i;
        G.vertices[i].indegree = 0;
        G.vertices[i].firstarc = NULL;
    }//初始化
    for (int i = 1;i <=G.vexnum;i++)
    {
        cin >> G.vertices[i].time;
        int num;
        cin >> num;
        for (int j = 1;j <=num;j++)
        {
            int start;
            cin >> start;
            ArcNode* p, * t;
            p = (ArcNode*)malloc(sizeof(ArcNode));
            p->adjvex = i;
            p->nextarc = NULL;
            p->time = 0;
```

```
            t = G.vertices[start].firstarc;
            if (t == NULL)
            {
                G.vertices[start].firstarc = p;
            }
            else
            {
                while (t->nextarc != NULL)
                {
                    t = t->nextarc;
                }
                t->nextarc = p;
            }
            G.vertices[i].indegree++;
        }
    }
    for (int i = 1;i <= G.vexnum;i++)
    {
        ArcNode* p = G.vertices[i].firstarc;
        while (p)
        {
            p->time = G.vertices[p->adjvex].time;
            p = p->nextarc;
        }
    }//所有边和点存入邻接表
}
int TopologicalOrder(ALGraph& G)//拓扑排序同时求事件最早开始时间
{
    for (int i = 1;i <= G.vexnum;i++)
    {
        ve[i] = G.vertices[i].time;
    }
    for (int i = 1;i <= G.vexnum;i++)
    {
        if (G.vertices[i].indegree == 0)
            L.push(i);
    }
    int count = 0;
    while (L.empty() == 0)
    {
        int x = L.top();
        L.pop();
        T.push(x);
        count++;
```

```
        for (ArcNode* p = G.vertices[x].firstarc;p;p = p->nextarc)
        {
            int k;
            k = p->adjvex;
            G.vertices[k].indegree--;
            if (G.vertices[k].indegree == 0)
                L.push(k);
            if (ve[x] + p->time > ve[k])
                ve[k] = ve[x]+p->time;//取所有 ve[x]+p->time 的最大值
        }
    }
    if (count < G.vexnum) return 0;//有环
    return 1;
}
int criticalevent(ALGraph& G)
{
    if (!TopologicalOrder(G)) return 0;
    int max = 0;
    for (int i = 1;i <= G.vexnum;i++)
    {
        if(max<ve[i])
            max=ve[i];

    }
    for (int i = 1;i <= G.vexnum;i++)
    {
        vl[i] = max;
    }//将所有 vl 赋值为最大
    while (T.empty() == 0)
    {
        int j = T.top();
        T.pop();
        ArcNode* p = G.vertices[j].firstarc;
        if (p == NULL)
        {
            vl[j] = max;
        }
        else
        for ( p = G.vertices[j].firstarc;p;p = p->nextarc)
        {
            int k = p->adjvex;
            int dut = p->time;
            if (vl[k] - dut < vl[j])
                vl[j] = vl[k] - dut;//取所有 vl[k] - dut 的最小值
```

```
        }

    }
    for (int j = 1;j <= G.vexnum;++j)
    {
        cout << ve[j] << " ";
        ArcNode* p=G.vertices[j].firstarc;
        if (p == NULL)//单独判断
        {
            if (ve[j] == max)
                cout << "1" << endl;
            else cout << "0" << endl;
        }
        else
        {
            int flag = 0;
            for (p = G.vertices[j].firstarc;p;p = p->nextarc)
            {
                int k = p->adjvex;
                int dut = p->time;
                int ee = ve[j];
                int el = vl[k] - dut;
                if (ee == el)//活动的最早开始时间和最晚开始时间相同
                {
                    cout << 1 <</*" "<<ee <<" " << el <<*/ endl;
                    break;
                }

            }
            if (p == NULL)
                cout << 0 << endl;
        }
    }
    return 0;
}
int main()
{
    input(G);
    criticalevent(G);
    return 0;
}
```

2.5.5 调试分析

本题是求关键路径的算法来解决排课问题，本题在调试上还是遇到了一些问题，首先是一开

始没有在 ve 初始化的时候加上自身的时间，导致最后最早完成时间输出时无法再次找到自身时间（因为是拓扑排序后的）而产生答案错误，其次就是由于本题不是 AOE 网，可能有多个起点甚至说有多个联通分量，而活动也是通过边的权值来描述，所以要对最右侧出度为 0 的点进行单独判断。

后来大部分测试数据正确，却只有小部分数据出错，是因为我在赋值最大值时，没有把最终的最大值赋给 v1，导致出现问题。

2.5.6 总结与收获

本题是对图的存储结构、拓扑排序、遍历、判断关键路径关键活动的一个重要题型和算法结构，本来在课上没有理解的非常透彻的知识点在我的代码自己实现中得到了不小的提升。而且在调试代码上，我寻找自己错误的能力也在提高，要把代码的每一部分关联起来，多构造测试数据来进行测试。

3. 实验总结

本次实验题目针对性较强，基本上每个题目都有对应的知识范围和算法，对于我们来说相对友好一些，但是个人认为进入图以后各种算法比前面的要复杂的多，过程也更为繁多，需要我们耐心的去看书、复习 ppt 以及自己耐心调试实现，不能急于求成。本次实验的代码量也比较大，对于我个人的代码能力肯定有了一定的提升，最重要的是在实现某一算法的过程中实际上也是对其各种知识点和细节的巩固，而查找 bug 的过程也正是修补自己知识体系漏洞的过程，十分重要。