

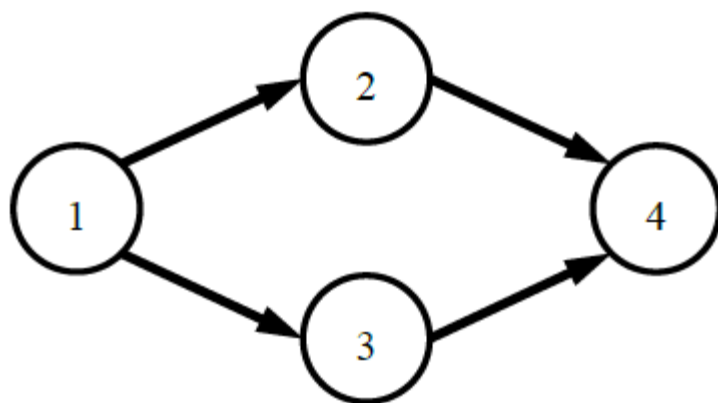
实验 6 图的应用

【问题描述】

应用图的 ADT 的物理实现来解决图的应用问题。

某国的军队由 N 个部门组成，为了提高安全性，部门之间建立了 M 条通路，每条通路只能单向传递信息，即一条从部门 a 到部门 b 的通路只能由 a 向 b 传递信息。信息可以通过中转的方式进行传递，即如果 a 能将信息传递到 b ， b 又能将信息传递到 c ，则 a 能将信息传递到 c 。一条信息可能通过多次中转最终到达目的地。

由于保密工作做得很好，并不是所有部门之间都互相知道彼此的存在。只有当两个部门之间可以直接或间接传递信息时，他们才彼此知道对方的存在。部门之间不会把自己知道哪些部门告诉其他部门。



上图中给了一个 4 个部门的例子，图中的单向边表示通路。部门 1 可以将消息发送给所有部门，部门 4 可以接收所有部门的消息，所以部门 1 和部门 4 知道所有其他部门的存在。部门 2 和部门 3 之间没有任何方式可以发送消息，所以部门 2 和部门 3 互相不知道彼此的存在。

现在请问，有多少个部门知道所有 N 个部门的存在。或者说，有多少个部门所知道的部门数量（包括自己）正好是 N 。

【输入形式】

输入的第一行包含两个整数 N, M ，分别表示部门的数量和单向通路的数量。所有部门从 1 到 N 标号。

接下来 M 行，每行两个整数 a, b ，表示部门 a 到部门 b 有一条单向通路。

【输出形式】

输出一行，包含一个整数，表示答案。

【样例输入】

4 4
1 2
1 3
2 4
3 4

【样例输出】

2

【样例说明】

部门 1 和部门 4 知道所有其他部门的存在。

一、问题分析

要求：

分析并确定要处理的对象（数据）是什么。

分析并确定要实现的功能是什么。

分析并确定处理后的结果如何显示。

本题给定 N 个结点和 M 个二元组，二元组 $\langle a, b \rangle$ 代表 a 到 b 之间有一条单向通路。根据题目所给数据，构建一个有向图，若一个结点可以直接或间接到达其他结点或者其他结点可以直接或间接到达这个结点，那么这个结点就知道所有其他结点的存在，求出在有向图中这样的结点有多少个。也就是题目问的有多少个部门知道所有 N 个部门的存在。最后将符合要求的结点个数打印输出。

二、数据结构和算法设计

抽象数据类型设计

```
template <typename VertexType>
```

```
class Graph {
```

```
private:
```

```
    void operator =(const Graph&) {}          // Protect assignment
```

```
    Graph(const Graph&) {}                    // Protect copy constructor
```

```

public:

    Graph() {}           // Default constructor
    virtual ~Graph() {} // Base destructor

    // Initialize a graph of n vertices
    virtual void Init(int n) =0;

    // Return: the number of vertices and edges
    virtual int n() =0;
    virtual int e() =0;

    // Return v's first neighbor
    virtual int first(int v) =0;

    // Return v's next neighbor
    virtual int next(int v, int w) =0;

    //设置图的类型（有向图或无向图）
    virtual void setType(bool flag)=0;
    //获取图的类型
    virtual bool getType()=0;
    //找到(包含实际信息的)顶点在图中的位置
    virtual int locateVex(VertexType u) =0;
    //返回某个顶点的值(实际信息)
    virtual VertexType getVex(int v)=0;
    //给某个顶点赋值
    virtual void setVex(int v,VertexType value) =0;

```

```

// Set the weight for an edge
virtual void setEdge(int v1, int v2, int wght) =0;

// Delete an edge
// i, j: The vertices
virtual void delEdge(int v1, int v2) =0;

// Determine if an edge is in the graph
// i, j: The vertices
// Return: true if edge i,j has non-zero weight
virtual bool isEdge(int i, int j) =0;

// Return an edge's weight
// i, j: The vertices
// Return: The weight of edge i,j, or zero
virtual int weight(int v1, int v2) =0;

// Get and Set the mark value for a vertex
// v: The vertex
// val: The value to set
virtual int getMark(int v) =0;
virtual void setMark(int v, int val) =0;
};

```

物理数据对象设计（不用给出基本操作的实现）

物理存储结构采用邻接矩阵来实现。

```

#define MAX_VERTEX_NUM 40
#define UNVISITED 0
#define VISITED 1
using namespace std;

```

```

template <typename VertexType>
class Graphm : public Graph<VertexType> {
private:
    int numVertex, numEdge; //顶点数和边数
    bool undirected; // true if graph is undirected, false if directed
    VertexType vexs[MAX_VERTEX_NUM]; //存储顶点信息

    int **matrix;           // Pointer to adjacency matrix
    int *mark;              // Pointer to mark array
public:
    Graphm(int numVert)     // Constructor
    { Init(numVert); }

    ~Graphm() {             // Destructor
        cout<<"graph delete";
        delete [] mark; // Return dynamically allocated memory
        for (int i=0; i<numVertex; i++)
            delete [] matrix[i];
        delete [] matrix;
    }

    void Init(int n) { // Initialize the graph
        int i;
        numVertex = n;
        numEdge = 0;
        mark = new int[n]; // Initialize mark array
        for (i=0; i<numVertex; i++)
            mark[i] = UNVISITED;
        matrix = (int**) new int*[numVertex]; // Make matrix
    }
};

```

```

    for (i=0; i<numVertex; i++)
        matrix[i] = new int[numVertex];
    for (i=0; i< numVertex; i++) // Initialize to 0 weights
        for (int j=0; j<numVertex; j++)
            matrix[i][j] = 0;
}

int n() { return numVertex; } // Number of vertices
int e() { return numEdge; }   // Number of edges

// Return first neighbor of "v"
int first(int v) {
    for (int i=0; i<numVertex; i++)
        if (matrix[v][i] != 0) return i;
    return numVertex;          // Return n if none
}

// Return v's next neighbor after w
int next(int v, int w) {
    for(int i=w+1; i<numVertex; i++)
        if (matrix[v][i] != 0)
            return i;
    return numVertex;          // Return n if none
}

//设置图的类型（有向图或无向图）
void setType(bool flag){
    undirected=flag;
}

//获取图的类型
bool getType(){

```

```

        return undirected;
    }

    /**返回顶点在图中的位置**/
    int locateVex(VertexType u){
        for(int i=0;i<numVertex;i++){
            if(Comp(u,vexs[i])) //Comp 模板函数写在 book.h 中
                return i;
        }
        return -1;
    }

    /**返回某个顶点的值(实际信息) **/
    VertexType getVex(int v){
        return vexs[v];
    }

    /**给某个顶点赋值**/
    void setVex(int v,VertexType value){
        vexs[v]=value;
    }

    // Set edge (v1, v2) to "wt"
    void setEdge(int v1, int v2, int wt) {
        Assert(wt>=0, "Illegal weight value");
        if (matrix[v1][v2] == 0)
            numEdge++;
        matrix[v1][v2] = wt;
        if(undirected){
            matrix[v2][v1] = wt;
        }
    }
}

```

```

void delEdge(int v1, int v2) { // Delete edge (v1, v2)
    if (matrix[v1][v2] != 0){
        numEdge--;
        matrix[v1][v2] = 0;
        if(undirected){
            matrix[v2][v1] = 0;
        }
    }
}

bool isEdge(int i, int j) // Is (i, j) an edge?
{ return matrix[i][j] != 0; }

int weight(int v1, int v2) { return matrix[v1][v2]; }
int getMark(int v) { return mark[v]; }
void setMark(int v, int val) { mark[v] = val; }
};

```

算法思想

1. 首先根据输入数据构建一个有向图。
2. 再对有向图的邻接矩阵运用 **warshall** 算法求其传递闭包，也就是经过数次传递后，每个结点可以直接或间接到达哪些结点。
3. 将传递闭包矩阵和它的转置矩阵相加，所得到的矩阵代表结点可以直接或间接到达哪些结点或者哪些结点可以直接或间接到达该结点。
4. 将上述矩阵按行遍历，记录元素值全不为 0 的行数，即为所求结点数。将结点个数打印输出。

关于 **warshall** 算法

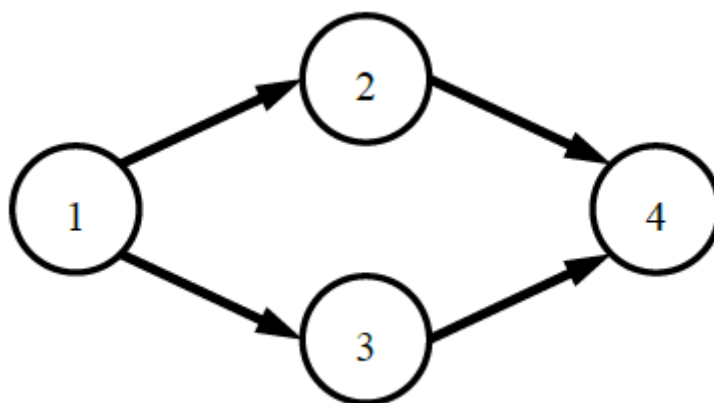
Warshall 在 1962 年提出了一个求关系的传递闭包的有效算法。其具体过程如下，设在 n 个元素的有限集上关系 R 的关系矩阵为 M ：

- (1) 置新矩阵 $A=M$;
- (2) 置 $i=1$;
- (3) 对所有 j 如果 $A[j,i]=1$ ，则对 $k=1\dots n$ 执行：
 $A[j,k]=A[j,k]\vee A[i,k]$;
- (4) i 加 1;
- (5) 如果 $i\leq n$ ，则转到步骤 (3)，否则停止。

所得的矩阵 A 即为关系 R 的传递闭包 t_R 的关系矩阵。

请用题目中样例，基于所设计的算法，详细给出样例求解过程。

1. 根据题目数据，构建有向图如下。



2. 有向图的邻接矩阵为

	1	2	3	4
1	1	1	1	0
2	0	1	0	1
3	0	0	1	1
4	0	0	0	1

运用 warshall 算法得到的传递闭包矩阵为

	1	2	3	4
1	1	1	1	1
2	0	1	0	1

3	0	0	1	1
4	0	0	0	1

3. 将传递闭包矩阵和它的转置矩阵相加，得到如下矩阵。

	1	2	3	4
1	1	1	1	1
2	1	1	0	1
3	1	0	1	1
4	1	1	1	1

4. 按行遍历该矩阵，发现第一行和第四行元素值全不为 0，所以所求结点数 2，输出 2.

关键功能的算法步骤

//数据输入

```
for(int i=0;i<M;i++)
```

```
{
```

```
    cin>>a>>b;
```

在 a 结点和 b 结点之间建立一条边;

```
}
```

//数据处理

```
for(int j=0;j<N;j++) //warshall 算法求出传递闭包
```

```
{
```

```
    for(int i=0;i<N;i++)
```

```
    {
```

```
        if(第 i 个结点和第 j 个结点存在边)
```

```
        {
```

```
            for(int k=0;k<N;k++)
```

```
            {
```

```
                第 i 行=第 i 行+第 j 行;
```

```
            }
```

```

        }
    }
}
for(int i=0;i<N;i++) //将传递闭包矩阵和它的转置矩阵相加
{
    for(int j=0;j<N;j++)
    {
        if(G->isEdge(i,j)) G->setEdge(j,i,1);
    }
}
int flag=1,count=0;
for(int i=0;i<N;i++) //将矩阵按行遍历，记录元素值全不为 0 的行数
{
    for(int j=0;j<N;j++)
    {
        if(!G->isEdge(i,j))
        {
            flag=0;
            break;
        }
    }
    if(flag) count++;
    else flag=1;
}
//数据输出
cout<<count<<endl;

```

三、算法性能分析

1. 数据输入部分

for 循环共执行 n 次，所以时间复杂度为 $O(n)$ 。

2. 数据处理部分

warshall 算法，两层循环嵌套，函数体的时间复杂度为 $O(n)$ ，所以最终的时间复杂度为 $O(n^3)$ 。

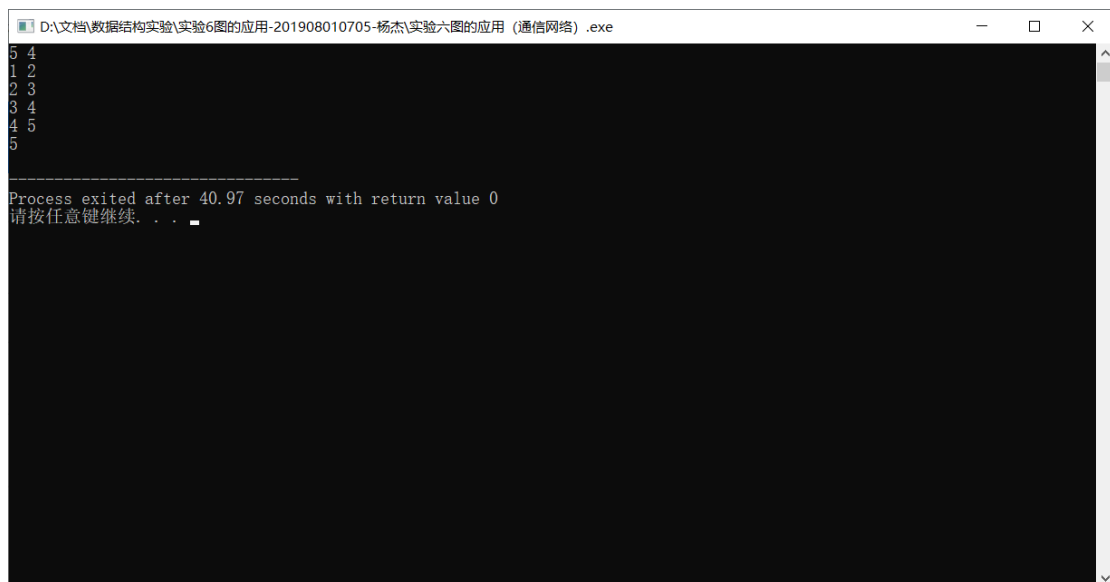
将传递闭包矩阵和它的转置矩阵相加，时间复杂度为 $O(n^2)$ 。

将上述矩阵按行遍历，记录元素值全不为 0 的行数，时间复杂度为 $O(n^2)$ 。

3. 数据输出部分

时间复杂度为 $O(1)$ 。

四、测试用例



```
D:\文档\数据结构实验\实验6图的应用-201908010705-杨杰\实验六图的应用 (通信网络) .exe
5 4
1 2
2 3
3 4
4 5
5

-----
Process exited after 40.97 seconds with return value 0
请按任意键继续. . .
```

实验六测试用例

```
D:\文档\数据结构实验\实验6图的应用-201908010705-杨杰\实验六图的应用 (通信网络).exe
4 4
1 2
1 3
2 4
3 4
2

-----
Process exited after 2.196 seconds with return value 0
请按任意键继续. . .
```

实验六测试用例

```
D:\文档\数据结构实验\实验6图的应用-201908010705-杨杰\实验六图的应用 (通信网络).exe
10 15
1 4
1 10
2 10
3 4
3 10
4 2
4 4
4 4
4 5
5 10
10 3
10 6
10 7
10 8
10 9
10 10
6

-----
Process exited after 107.2 seconds with return value 0
请按任意键继续. . .
```

实验六测试用例

```
D:\文档\数据结构实验\实验6图的应用-201908010705-杨杰\实验六图的应用 (通信网络) .exe
10 15
1 4
1 10
2 10
3 4
3 10
4 2
4 4
4 5
5 10
10 3
10 6
10 7
10 8
10 9
10 10
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 1
1 1 1 1 1 0 1 0 0 1
1 1 1 1 1 0 0 1 0 1
1 1 1 1 1 0 0 0 1 1
1 1 1 1 1 2 2 2 2 1
6
-----
Process exited after 61.99 seconds with return value 0
```

实验六测试用例