一、实验目的

掌握判定该图是否为强连通图的方法

二、实验内容

输入：一个图的邻接矩阵 输出：图的最大强连通分量，以及该图是否为强连通图

三、相关概念

邻接矩阵

邻接矩阵（Adjacency Matrix）是表示顶点之间相邻关系的矩阵。打个比方说，矩阵 A 中的元素 aij 储存的是 i 点到 j 点的路径信息。

强连通图

强连通图是指在有向图 G 中，如果对于每一对 vi、vj，vi≠vj，从 vi 到 vj 和从 vj 到 vi 都存在路径，则称 G 是强连通图。

关联矩阵

可达矩阵，指的是用矩阵形式来描述有向图的各节点之间经过一定长度的通路后可达到的程度

四、问题分析

无向图可以视为有向图处理。如果一个图是强连通图，那么从图中的任意一个顶点出发，都应该能够到达图中的任意其他顶点，也就是说，任意两个顶点之间都应该存在一条路径。邻接矩阵的幂运算实际上就是将这种路径的数量进行累积，从而得到某个顶点到其他所有顶点的可达性。

当我们进行邻接矩阵的幂运算时，例如将邻接矩阵进行平方、立方等运算，结果矩阵中的元素表示了两个顶点之间存在的长度为 2、3 等的路径数量。因此，如果进行了 2 到 n-1 次幂运算，而结果矩阵中某个元素为 0，意味着对应的两个顶点之间不存在长度为 2 到 n-1 的路径，也就是说它们之间不可达。

将结果矩阵的 1 到 n-1 次幂相加，并对角线元素置为 1，得到的可达矩阵，表示了从一个顶点出发，经过长度为 1 到 n-1 的任意路径，能够到达的所有顶点。如果这个可达矩阵中的所有元素都为 1，那么任意两个顶点之间都存在路径，即图是强连通图。

五、测试数据

我们使用以下程序作为数据生成器，以作为测试之用

```cpp
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

int getRandomInt(int min, int max) {
    return min + rand() % (max - min + 1);
}

void printAdjacencyMatrix(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
}

// 生成强联通图
void generateStronglyConnectedGraph(vector<vector<int>>& adjMatrix, int n) {
    // 保证强连通
    for (int i = 0; i < n; ++i) {
        adjMatrix[i][(i + 1) % n] = 1;
    }

    // 添加随机路径
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i != j && getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }
}

// 生成非强连通图
void generateNonStronglyConnectedGraph(vector<vector<int>>& adjMatrix, int n) {
    // Split the nodes into two sets to ensure non-strong connectivity
    int splitPoint = n / 2;

    for (int i = 0; i < splitPoint; ++i) {
        for (int j = 0; j < splitPoint; ++j) {
            if (i != j && getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }
    for (int i = splitPoint; i < n; ++i) {
        for (int j = splitPoint; j < n; ++j) {
            if (i != j && getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
```

```
            }
        }
    }

    // 添加随机边
    for (int i = 0; i < splitPoint; ++i) {
        for (int j = splitPoint; j < n; ++j) {
            if (getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }
}

int main() {
    srand(time(0));

    int n;
    bool isStronglyConnected = true;  // true 生成强连通图，false 生成非强联通图

    cout << "输入点的个数 > ";
    cin >> n;

    vector<vector<int>> adjMatrix(n, vector<int>(n, 0));

    if (isStronglyConnected) {
        generateStronglyConnectedGraph(adjMatrix, n);
    } else {
        generateNonStronglyConnectedGraph(adjMatrix, n);
    }

    printAdjacencyMatrix(adjMatrix);

    return 0;
}
```

六、重要代码分析

求邻接矩阵 2~n-1 次方对应每个点之和

```
int sum, x = 2;
    while (x < dot) {
        cout << "矩阵的" << x << "次方为: " << endl;
        for (int i = 0; i < dot; i++) {
            for (int j = 0; j < dot; j++) {
                sum = 0;
                for (int k = 0; k < dot; k++) {
                    sum +=
                        abs(b[i][k]) *
                        abs(adj[k][j]);  // 有向图中有-1表示反向，所以加上绝对值
```

```
                }
                c[i][j] = sum;
                acc[i][j] += c[i][j];
                cout << c[i][j] << " ";
            }  // j
            cout << endl;
            for (int i = 0; i < dot; i++) {
                for (int j = 0; j < x; j++) {
                    b[i][j] = c[i][j];
                }
            }
        }  // i
        ++x;
    }  // while
```

生成可达矩阵

```
for (int i = 0; i < dot; i++)  // 形成此图的可达矩阵
    {
        for (int j = 0; j < x; j++) {
            if (acc[i][j] || i == j)
                acc[i][j] = 1;
        }
    }
```

七、实验结果

输入数据：

```
// test one: true
0 1 1 0 1 0 1 0 1 0
0 0 1 0 1 1 1 0 1 0
0 1 0 1 1 0 0 0 0 1
0 0 1 0 1 1 1 0 1 0
0 1 0 1 0 1 0 1 1 1
1 1 1 1 1 0 1 1 0 1
0 0 0 1 1 0 0 1 0 1
1 1 1 1 1 1 1 0 1 0
0 1 1 1 1 0 0 0 0 1
1 1 0 1 1 1 0 0 1 0

// test two: true
0 1 0 0 0 0 0 1 1 1 1 0 0 1 1
0 0 1 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 1 1 0 1 0 1
0 1 0 0 1 1 1 0 1 1 0 1 1 1 1
0 1 1 0 0 1 1 0 1 0 1 1 0 0 0
1 0 1 0 0 0 1 1 1 1 1 1 0 1 1
```

```
0 1 0 0 1 0 0 1 0 0 0 0 1 1 1
1 1 0 1 0 0 1 0 1 1 1 1 0 0 0
0 0 1 0 1 0 1 0 0 1 0 0 1 1 0
1 0 0 1 0 0 0 0 1 0 1 1 0 0 0
1 0 1 1 1 1 1 0 0 1 0 1 1 0 0
0 0 1 0 1 1 1 1 1 0 0 0 1 0 1
1 0 0 0 0 1 1 1 1 1 0 0 0 1 0
0 0 0 0 1 1 1 1 0 0 1 0 1 0 1
1 1 0 0 0 1 0 1 0 1 1 1 1 0 0

// test three: false
0 1 0 1 1 0 0 0 1 0 1 0
1 0 1 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 1 0 1 0 1 1
1 1 0 0 0 0 0 1 0 0 1 0
1 0 1 0 0 0 0 1 1 0 1 1
1 1 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 1
0 0 0 0 0 0 0 1 0 1 1 0

// test four: false
0 1 1 1 0 0 1 1
1 0 0 1 1 1 1 1
1 1 0 1 1 1 1 0
1 1 1 0 0 1 0 1
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 1
0 0 0 0 1 0 1 0
```

输出结果

```
// test one
此图的可达矩阵为:
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 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
此图为强联通图!

// test two
此图的可达矩阵为:
```

```
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 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 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
此图为强联通图!

// test three
此图的可达矩阵为:
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 0 0 0 1 1 0 1 0 0
0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 1 1 1 1 1
此图是非强连通图!

// test four
此图的可达矩阵为:
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 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
此图是非强连通图!
```

八、应用研究

连通图在图论中是一个非常重要的概念，并且在多个领域有着广泛的应用。以下是一些连通图应用研究的具体实例和领域:

1. 网络通信 在网络通信中，连通图用于描述计算机网络的拓扑结构。确保网络连通性是至关重要的，因为这意味着信息可以在网络中的任何两点之间传输。研究连通图可以帮助优化网络设计，提高网络的容错能力和效率。

2. 社会网络分析 在社会网络分析中，连通图用于研究人际关系和社交网络。通过分析社交网络的连通性，可以发现社交网络中的关键节点（即具有重要连接作用的个人或团体），以及网络中的社团结构和信息传播路径。

3. 交通网络 在交通网络中，连通图用于描述道路和交通系统的结构。研究交通网络的连通性可以帮助优化交通流量，设计更有效的交通路线，以及提高交通系统的整体可靠性和安全性。

4. 电力系统 在电力系统中，连通图用于表示电网的拓扑结构。确保电网的连通性是电力系统稳定运行的关键。研究连通图可以帮助检测和预防电网中的潜在故障，优化电力传输和分配。

5. 生物网络 在生物网络中，连通图用于研究生物系统中的相互作用，例如基因调控网络、蛋白质相互作用网络和生态系统中的物种相互关系。通过分析生物网络的连通性，可以揭示生物系统的功能和动态行为。

6. 机器人导航 在机器人导航中，连通图用于表示机器人的工作环境，例如地图或路径规划。研究连通图可以帮助设计高效的导航算法，使机器人能够在复杂环境中自主移动并完成任务。

7. 互联网和万维网 在互联网和万维网中，连通图用于描述网页之间的链接结构。研究互联网的连通性可以帮助改进搜索引擎算法、提高网页排名的准确性，以及检测和预防网络攻击。

8. 分子化学 在分子化学中，连通图用于表示分子的结构。研究分子的连通性可以帮助理解化学反应的机制，设计新的化合物和材料，以及预测分子的物理和化学性质。

9. 供应链管理 在供应链管理中，连通图用于描述供应链中的各个环节及其相互关系。研究供应链的连通性可以帮助优化供应链管理，提高供应链的效率和响应能力，降低成本和风险。

10. 灾害管理和紧急响应 在灾害管理和紧急响应中，连通图用于表示资源分配和救援网络。研究紧急响应网络的连通性可以帮助设计高效的救援计划，提高灾害应对的效率和效果。

九、实验感受

从代码的角度讲，我的代码实现时间复杂度主要消耗在于矩阵运算。矩阵乘法的时间复杂度为 $O（n^3）$，而我使用的方法是中规中矩的更新可达矩阵的方法，判断的效率其实并不高。我认为判断连通图是可以用 tarjan 缩点的思想来解决的。我们可以将所有的连通点都看作是一个点，然后通过 dfs 来遍历全图，这样只需要判断最后整个图是不是只剩下一个点就行了。这样做时间复杂度为 O(V+E)，大大提升了程序效率。这里我也简单写一个 tarjan 算法的实现吧。

```
void tarjan(int u) {
    int i,j;
    int v;
    DFN[u]=LOW[u] = ++Index;
    instack[u] = true;
    stack[++top] = u;
    for (i = node[u]; i != -1; i = edge[i].next) {
        v = edge[i].v;
```

```
        if (!DFN[v]) {//如果点v没被访问
            tarjan(v);
            if (LOW[v] < LOW[u])
                LOW[u] = LOW[v];
        } else//如果点v已经被访问过
            if (instack[v] && DFN[v]<LOW[u])
                LOW[u] = DFN[v];
    }
    if (DFN[u] == LOW[u]) {
        Bcnt ++;
        do {
            j = stack[top--];
            instack[j] = false;
            Belong[j] = Bcnt;
        }
        while (j != u);
    }
}
void solve() {
    int i;
    top = Bcnt = Index = 0;
    memset(DFN, 0, sizeof(DFN));
    memset(LOW, 0, sizeof(LOW));
    for (i = 1;i <= n;i ++)
        if (!DFN[i])
            tarjan(i);
}
```

这个程序也不是用邻接矩阵的方法存图了，而是使用链式前向星的方式保存图的信息。但是言归正传，这个想法虽然没有问题，但是并不符合现在的课程要求（笑）。使用可达矩阵判断连通图的方式足矣。也就没有必要使用这些课程之外的方法。

想了一下，其实也可以通过并查集来判断连通性。我们用并查集来进行连通分量的统计，若连通分量大于 1，则说明图中存在多个连通分量，图不为连通图

```
int n,m;
int father[N];
int Find(int x) {
    if(father[x] == -1)
        return x;
    return father[x] = Find(father[x]);
}
void Union(int x,int y) {
    x = Find(x);
    y = Find(y);
    if(x != y)
        father[x] = y;
}
int main() {
    memset(father, -1, sizeof(father));
    scanf("%d%d", &n, &m);
```

```
    for(int i = 1; i <= m; i ++){
        int x,y;
        scanf("%d%d", &x, &y);
        Union(x, y);
    }

    int cnt = 0;//记录连通分量
    for(int i = 1; i <= n; i ++)
        if(Find(i) == i)
            cnt++;
    printf("%d\n", cnt);
    return 0;
}
```

【组员感想】 郑华展： 通过参与离散数学课程设计，我成功丰富并巩固了在离散数学应用方面的经验。在课程中，我特别深入研究了图论，通过解决设计中提出的问题，我对图论的概念和原理有了更深刻的理解。这个过程不仅增强了我的理论基础，还培养了我在离散数学应用方面的实际技能。此外，通过编写相关的实验代码，我的编程能力得到了进一步提升，代码逻辑变得更加简洁易懂，程序架构设计能力得到了进一步提升

任翔： 通过参与离散数学课程设计，我显著丰富并深化了在离散数学应用领域的经验。特别是在图论的学习中，我深入探索了其核心概念与原理，通过解决课程设计中的一系列问题，我对图论的理解达到了新的高度。这一过程不仅巩固了我的理论知识基础，还极大地锻炼了我在离散数学应用方面的实践能力。同时，通过编写与图论相关的实验代码，我不仅提高了编程技能，还使得代码逻辑更为清晰简洁，程序架构设计能力得到了显著增强。这些宝贵的经验将对我的未来发展产生深远影响。

十、完整代码

generater.cpp

```
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

int getRandomInt(int min, int max) {
    return min + rand() % (max - min + 1);
}

void printAdjacencyMatrix(const vector<vector<int>>& matrix) {
    for (const auto& row : matrix) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
}
```

```cpp
// 生成强联通图
void generateStronglyConnectedGraph(vector<vector<int>>& adjMatrix, int n) {
    // 保证强连通
    for (int i = 0; i < n; ++i) {
        adjMatrix[i][(i + 1) % n] = 1;
    }

    // 添加随机路径
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i != j && getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }
}

// 生成非强连通图
void generateNonStronglyConnectedGraph(vector<vector<int>>& adjMatrix, int n) {
    int splitPoint = n / 2;

    for (int i = 0; i < splitPoint; ++i) {
        for (int j = 0; j < splitPoint; ++j) {
            if (i != j && getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }
    for (int i = splitPoint; i < n; ++i) {
        for (int j = splitPoint; j < n; ++j) {
            if (i != j && getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }

    // 添加随机边
    for (int i = 0; i < splitPoint; ++i) {
        for (int j = splitPoint; j < n; ++j) {
            if (getRandomInt(0, 1) == 1) {
                adjMatrix[i][j] = 1;
            }
        }
    }
}

int main() {
    srand(time(0));

    int n;
    bool isStronglyConnected = true;  // true 生成强连通图, false 生成非强联通图

    cout << "输入点的个数 > ";
    cin >> n;
```

```
    vector<vector<int>> adjMatrix(n, vector<int>(n, 0));

    if (isStronglyConnected) {
        generateStronglyConnectedGraph(adjMatrix, n);
    } else {
        generateNonStronglyConnectedGraph(adjMatrix, n);
    }

    printAdjacencyMatrix(adjMatrix);

    return 0;
}
```

resolver.cpp

```
#include <math.h>
#include <iostream>
using namespace std;

int main() {
    cout << "请输入顶点数: ";
    int dot;
    cin >> dot;
    int adj[dot][dot], b[dot][dot], c[dot][dot],
        acc[dot][dot];  // 邻接矩阵与可达矩阵
    cout << "请按次序输入每行数据，形成此图的邻接矩阵 (" << dot << "*" << dot
         << "矩阵" << "): " << endl;
    for (int i = 0; i < dot; i++)  // 邻接矩阵赋值
    {
        for (int j = 0; j < dot; j++) {
            cin >> adj[i][j];
            acc[i][j] = adj[i][j];
            b[i][j] = acc[i][j];
            c[i][j] = acc[i][j];
        }
        getchar();
    }

    int sum, x = 2;  // 求邻接矩阵2~n-1次方对应每个点之和
    while (x < dot) {
        cout << "矩阵的" << x << "次方为: " << endl;
        for (int i = 0; i < dot; i++) {
            for (int j = 0; j < dot; j++) {
                sum = 0;
                for (int k = 0; k < dot; k++) {
                    sum +=
                        abs(b[i][k]) *
                        abs(adj[k][j]);  // 有向图中有-1表示反向，所以加上绝对值
                }
                c[i][j] = sum;
                acc[i][j] += c[i][j];
```

```
                cout << c[i][j] << " ";
            }  // j
            cout << endl;
            for (int i = 0; i < dot; i++) {
                for (int j = 0; j < x; j++) {
                    b[i][j] = c[i][j];
                }
            }
        }  // i
        ++x;
    }  // while

    for (int i = 0; i < dot; i++)  // 形成此图的可达矩阵
    {
        for (int j = 0; j < x; j++) {
            if (acc[i][j] || i == j)
                acc[i][j] = 1;
        }
    }

    cout << "此图的可达矩阵为: " << endl;
    for (int i = 0; i < dot; i++) {
        for (int j = 0; j < dot; j++) {
            cout << acc[i][j] << " ";
        }
        cout << endl;
    }

    for (int i = 0; i < dot; i++)  // 判断是否为强连通图
    {
        for (int j = 0; j < dot; j++) {
            if (!acc[i][j]) {
                cout << "此图是非强连通图! " << endl;
                return 0;
            }
        }
    }
    cout << "此图为强联通图!" << endl;
}
```