一、问题分析

- ## 分析并确定要处理的对象（数据）是什么

  N 个部门，M 条通路

- ## 分析并确定要实现的功能是什么

  输入并储存这些部门
  从每个部门查看能连接到哪些部门
  统计连接到的部门是否包括每个部门
  统计能够连接到所有部门的个数
  输出能够连接到所有部门的个数

- ## 分析并确定处理后的结果如何显示

  将能够联通所有的部门的部门个数输出到屏幕上

二、数据结构和算法设计

- ## 抽象数据类型设计

抽象数据类型：图
选择原因：因为输入的结构是一个网状结构，所以选用图来完成

- ## 物理数据对象设计

物理数据类型：基于邻接矩阵实现的单向无权图

```
class map_matirx:public Graph
{
public:
    map_matirx(int num);
    ~map_matirx();
    int n();
    int e();
    int first(int v);
    int next(int v, int w);
    bool setPoint(int v, T val);
    bool setEdge(int v1, int v2, int wght);
    bool delEdge(int v1, int v2);
    bool isEdge(int i, int j);
    int weight(int v1, int v2);
    bool setMark(int v, bool val);
    bool getMark(int v);
    int findPoint(T v);
    T getPoint(int v);
```

```
    void reset_mark();          //将所有标记设为未访问
    vector<T> BFS(int n);       //输出图的广度优先遍历的序列
private:
    int numPoint, numEdge;
    T* point;
    bool* mark;
    int** matirx;
};
```

- ## 算法思想的设计

构建模块

1、将输入的点和边存入图中

2、将边反向存入另一图中

需要的基本操作

setpoint

setMark

setEdge

计算模块

1、以每个顶点为起点，得出它正向遍历的序列

2、得出对应点反向遍历的序列

3、将两序列合并，删除重复节点

4、将序列中节点数与总节点数比较，若相等，则计数器加一

需要的基本操作

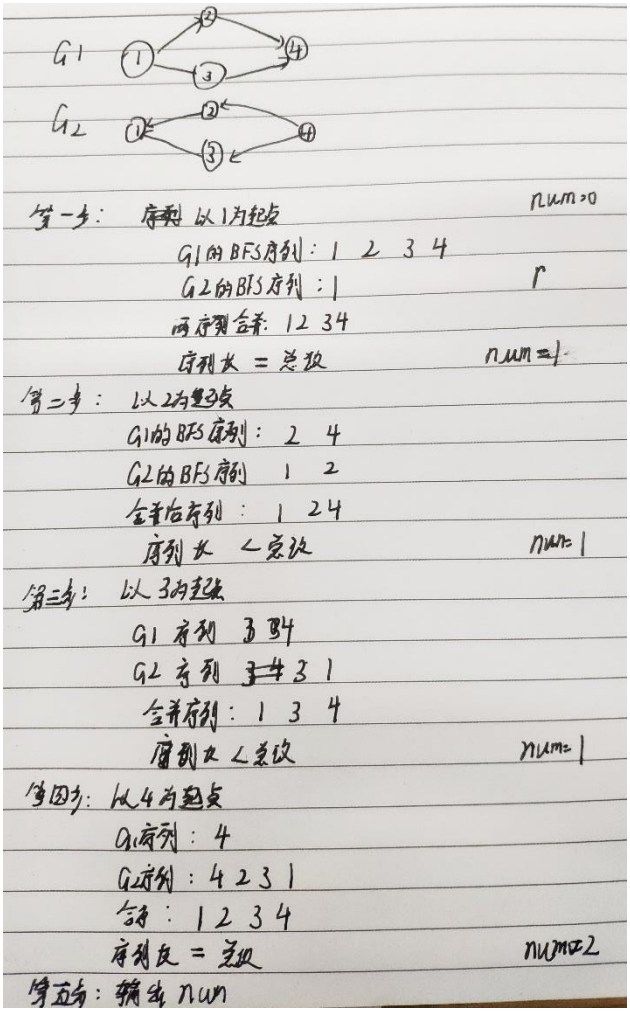
BFS

reset_mark

输出模块

直接将计算模块的计数器输出

- 样例推导



- 关键功能的算法步骤（不能用源码）

构建模块

```
void creat(map_matirx<int>& G1,map_matirx<int>& G2, int n, int e)
{
    for (int i = 0; i < n; i++)
    {
        G1.setPoint(i, i + 1);
        G1.setMark(i, false);
        G2.setPoint(i, i + 1);
        G2.setMark(i, false);
    }
    for (int i = 0; i < e; i++)
    {
        int s1, s2;
        cin >> s1 >> s2;
```

```
            G1.setEdge(s1-1, s2-1, 1);
            G2.setEdge(s2-1, s1-1, 1);
        }
    }
```

计算模块

```
Int sum(G1,G2,int n){
Num=0;
for (int i = 0; i < n; i++)
        {
            vector<int> l1, l2;
            l1 = G1.BFS(i);
            l2 = G2.BFS(i);
            for (vector<int>::iterator it = l2.begin(); it != l2.end(); it++)
            {
                l1.push_back(*it);
            }
            sort(l1.begin(), l1.end());
            l1.erase(unique(l1.begin(), l1.end()), l1.end());      //去除相同的顶点
            if (l1.size() == n)                          //如果能达到所有的顶点，计数器+1
                num++;
            G1.reset_mark();                             //重置 G1 的访问
            G2.reset_mark();                             //重置 G2 的访问
        }
Return num;
}
```

输出模块

```
cout << num;
```

三、算法性能分析

输入模块： 对每个顶点和边进行插入，时间复杂度为 O(D+E);

计算模块： 对每个顶点进行 BFS 遍历，时间复杂度为 O(D(D+E));

输出模块：只有一个输出，时间复杂度为 O(1)

总的时间复杂度为 O(D(D+E))