

LintCode领扣题解 (/problem) / 朋友圈 · Friend Circles

朋友圈 · Friend Circles

中文

- dfs (/problem/?tags=dfs)
- bfs (/problem/?tags=bfs)
- Two Sigma (/problem/?tags=two-sigma)
- Bloomberg (/problem/?tags=bloomberg)
- union find (/problem/?tags=union-find)

描述

一个班中有N个学生。他们中的一些是朋友，一些不是。他们的关系传递。例如，如果A是B的一个直接朋友，而B是C的一个直接朋友，那么A是C的一个间接朋友。我们定义朋友圈为一组直接和间接朋友。  
给出一个N\*N 矩阵M表示一个班中学生的关系。如果m (i) (j) = 1，那么第i个学生和第j个学生是直接朋友，否则不是。你要输出朋友圈的数量。

1.1≤N≤200。 2.对于所有学生M[i][i] = 1。 3.如果 M[i][j] = 1, 那么 M[j][i] = 1。

样例

样例 1:

输入: [[1,1,0],[1,1,0],[0,0,1]]  
输出: 2  
解释:  
0号和1号学生是直接朋友，所以他们位于一个朋友圈内。  
2号学生自己位于一个朋友圈内。所以返回2。

样例 2:

输入: [[1,1,0],[1,1,1],[0,1,1]]  
输出: 1  
解释:  
0号和1号学生是直接朋友，1号和2号学生处于同一个朋友圈。  
所以0号和2号是间接朋友。所有人都处于一个朋友圈内，所以返回1。

在线评测地址: <https://www.lintcode.com/problem/friend-circles/> (<https://www.lintcode.com/problem/friend-circles/>)

收起题目描述 ^

语言类型


ALL (15)

python (7)

java (5)

cpp (3)

上传题解



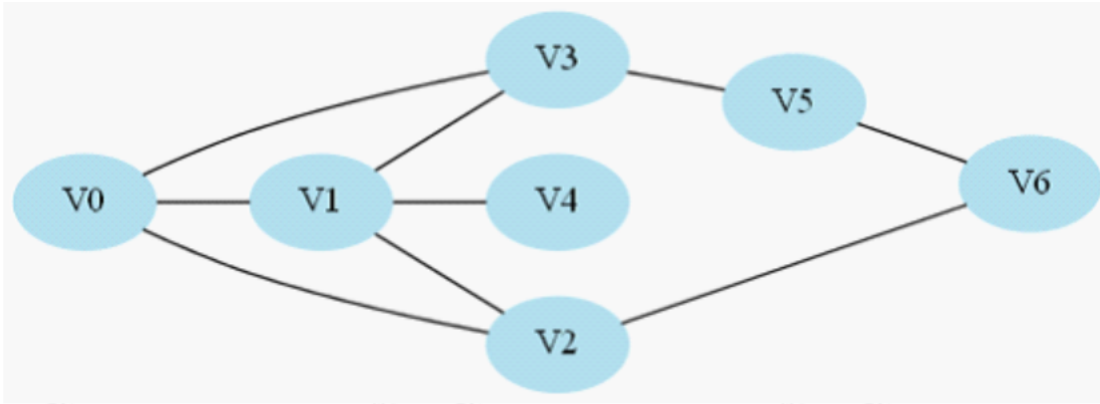
DDBear

更新于 11/30/2020, 2:03:20 AM

预备知识

BFS 宽度优先搜索

广度优先搜索（也称宽度优先搜索，缩写BFS，以下采用广度来描述）是连通图的一种遍历策略。因为它的思想是从一个顶点V0开始，辐射状地优先遍历其周围较广的区域。



$V_0$ 能遍历 $V_3, V_1, V_2$

$V_3$ 能遍历 $V_5$

$V_3$ 还能到 $V_1$  但 $V_1$ 已经被遍历过了，就不用再次遍历了

$V_1$ 遍历 $V_4$

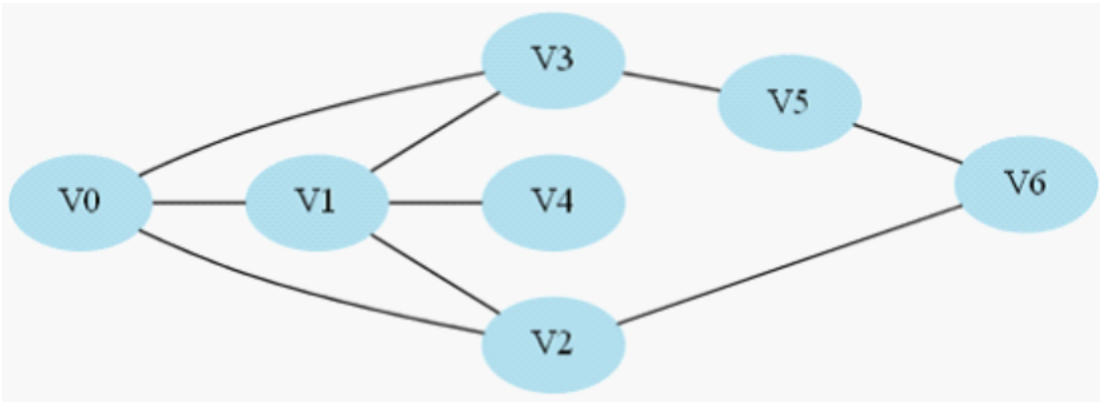
$V_1$ 还能遍历 $V_2, V_3$ 但他们两个也都是遍历过的了，就不用再次遍历了

$V_2$ 遍历到 $V_6$

所以按照BFS 遍历顺序 $V_0 V_3 V_1 V_2 V_5 V_4 V_6$

## DFS 深度优先搜索

深度优先搜索（缩写DFS）有点类似广度优先搜索，也是对一个连通图进行遍历的算法。它的思想是从一个顶点 $V_0$ 开始，沿着一条路一直走到底，如果发现不能到达目标解，那就返回到上一个节点，然后从另一条路开始走到底，这种尽量往深处走的概念即是深度优先的概念。



$V_0 -> V_3 -> V_5 -> V_6$ 到头了

$V_0 -> V_1 -> V_4$

$V_0 -> V_1 -> V_2$

所以按照DFS顺序 $V_0 V_3 V_5 V_6 V_1 V_4 V_2$

BFS和DFS都是存在多种方案的，上面的只是其中一种

## 并查集

并查集(Disjoint-Set)是一种可以动态维护若干个不重叠的集合，并支持合并与 查询两种操作的一种数据结构。

并查集是指用集合里的一个元素来当这个集合的代表元

如果两个元素所在集合的代表元相同, 那么我们就知道这两个元素在一个集合当中。

如果我们想合并两个集合, 只需要把其中一个集合的代表元改为第二个集合的代表元

**初始化:** 每一个点都是一个集合, 因此自己的父节点就是自己  $fa[i] = i$

**查询:** 每一个节点不断寻找自己的父节点, 若此时自己的父节点就是自己, 那么该点为集合的根结点, 返回该点。

**修改:** 合并两个集合只需要合并两个集合的根结点, 即  $fa[X] = Y$ , 其中  $X, Y$  是两个元素的根结点

$x \rightarrow x1 \rightarrow x2 \rightarrow x3 \rightarrow x4 \rightarrow x5 \rightarrow x6 \rightarrow \dots \rightarrow$  代表元 T

我们在第一次寻找x的代表元的回溯的时候, 顺便把这条路径的所有  $x_i$  的父亲改为了代表元 T

这样我们以后再次访问  $x \dots x6 \dots T$  这条链上的内容时候就可以很快的得到答案

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
//伪代码
for i 1:n
    fa[i]=i    //伪代码, 一开始让所有的父亲都是本身
              //我们规定代表元的父亲为本身, 如果一个节点的父亲不是本身, 说是它在一个元素个数大于1的集合中, 而且这个节点并不是代表元
function find(x) //寻找x所在集合的代表元
    if(fa[x]==x)
        return x; //x是代表元, 直接返回
    else
        return fa[x]=find(fa[x]) //x不是代表元, 寻找x的父亲的代表元是谁, 并且直接把代表元赋值给x的父亲
function uniu(x,y) //合并两个集合
    fa[find(x)]=find(y)
```

## 题目算法

### 算法一 BFS

- 遍历每个人, 如果这个人之前没有访问过, 说明有多一个新的朋友圈, 答案记录加一 就从这个点作为起点 做一次BFS, 找出所有的直接朋友与间接朋友, 并把他们标记访问。
- BFS流程
  - 将起点压入队列, 标记访问
  - 取出队首, 从队首向外找朋友, 看都能扩展到哪些还没访问的朋友, 压入队列并标记访问
  - 重复执行上一步, 直到队列为空

### 算法二 DFS

- 遍历每个人, 如果这个人之前没有访问过, 说明有多一个新的朋友圈, 答案记录加一 就从这个点作为起点 做一次DFS, 找出所有的直接朋友与间接朋友, 并把他们标记访问。
- DFS流程
  - 从起点开始DFS递归, 并把起点标记访问
  - 在当前递归层

```

/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
dfs(x)
{
    for i 1:n
        if i和x是朋友 并且 i没有访问过
            标记访问
            dfs(i)
}

```

## 算法三 并查集

- 初始化  $fa$  数组,  $ans = n$ , 一开始每个人一个集合, 然后根据朋友关系进行合并, 再减少集合数量
- 遍历每个人  $i$ 
  - 遍历他的每个朋友  $j$
  - 如果  $i, j$  通过并查集查询不在一个集合, 就把这两人所在的集合合并 并让答案减一

## 时间复杂度

### 算法一

一共最多  $N$  个人 每个人 进队一次 出队一次, 所以这一部分是  $O(N)$ , 出队时候会遍历他的所有朋友, 遍历朋友是  $O(N)$  所以总复杂度是  $O(N^2)$

### 算法二

一共最多  $N$  个人 每个人递归一次, 所以这一部分是  $O(N)$ , 递归时候会遍历他的所有朋友, 遍历朋友是  $O(N)$  所以总复杂度是  $O(N^2)$

### 算法三

并查集在有路径压缩的情况下 一次查询, 合并的时间复杂度  $O(\alpha)$  其中  $\alpha$  是反阿克曼函数, 它的增长极其极其极其缓慢, 可以近似的认为是  $O(1)$

所以并查集的时间复杂度是  $O(N^2)$

## 空间复杂度

### 算法一

一共最多  $N$  个人 每个人 进队一次 出队一次, 所以队列大小是  $O(N)$

标记数组 `visited` 也是  $O(N)$

### 算法二

一共最多  $N$  个人 每个人递归一次, 所以递归空间是  $O(N)$ ,

标记数组 `visited` 也是  $O(N)$

### 算法三

并查集  $fa$  数组 空间  $O(N)$

# 代码

## 算法一

c++

java

python

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
import collections

class Solution:
    def beginbfs(self, M):
        # 人数
        n = len(M)
        # 答案
        ans = 0
        # 标记是否访问过
        visisted = {}
        for i in range(n):
            visisted[i] = False
        # 遍历每个人, 如果这个人还没访问过 就从这个人开始做一遍bfs
        for i in range(n):
            if (visisted[i] == False):
                ans += 1
                q = collections.deque()
                # 标记起点并压入队列
                visisted[i] = True
                q.append(i)
                while (len(q) != 0):
                    # 取出队首
                    now = q.popleft()
                    # 从队首找朋友
                    for j in range(n):
                        # 找到新朋友 (之前没访问过的朋友) 就标记访问并压入队列
                        if (M[now][j] == 1 and visisted[j] == False):
                            visisted[j] = True
                            q.append(j)
                return ans
        """
        @param M: a matrix
        @return: the total number of friend circles among all the students
        """
        def findCircleNum(self, M):
            # Write your code here
            ansbfs = self.beginbfs(M)
            return ansbfs
```

## 算法二

c++

java

python

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution:
    def dfs(self, x, M, visisted):
        for i in range(len(M)):
            if (M[x][i] == 1 and visisted[i] == False):
                visisted[i] = True
                self.dfs(i, M, visisted)
    def beindfs(self, M):
        # 人数
        n = len(M)
        # 答案
        ans = 0
        # 标记是否访问过
        visisted = {}
        for i in range(n):
            visisted[i] = False
        # 遍历每个人, 如果这个人还没访问过 就从这个人开始做一遍dfs
        for i in range(n):
            if (visisted[i] == False):
                ans += 1
                visisted[i] = True
                self.dfs(i, M, visisted)
        return ans
"""
@param M: a matrix
@return: the total number of friend circles among all the students
"""
    def findCircleNum(self, M):
        # Write your code here
        ansdfs = self.beindfs(M)
        return ansdfs
```

## 算法三

c++

java

python

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution:
    def find(self, x, fa):
        if x == fa[x]:
            return x
        else:
            fa[x] = self.find(fa[x], fa)
            return fa[x]
    def beginset(self, M):
        # 人数
        n = len(M)
        # 答案
        ans = n
        # fa数组
        fa = {}
        for i in range(n):
            fa[i] = i
        # 遍历每个人, 进行并查集合并
        for i in range(n):
            for j in range(n):
                if (i != j and M[i][j] == 1):
                    # 这两个朋友在不同的集合里 就把这两个集合合并
                    # 合并两个集合, 集合数量减少1个
                    if (self.find(i, fa) != self.find(j, fa)):
                        ans -= 1
                        fa[self.find(i, fa)] = self.find(j, fa)
        return ans
    """
    @param M: a matrix
    @return: the total number of friend circles among all the students
    """
    def findCircleNum(self, M):
        # Write your code here
        ansset = self.beginset(M)
        return ansset
```

👍 获赞 3

💬 1 条评论

# 你的口袋题库

## 2000+算法真题、国内外名企题库免费开放



九章算法APP



九章算法助教团队

更新于 8/18/2020, 4:00:52 PM

考点:

- bfs
- dfs
- 并查集

题解:

- 本题在于考察基本的联通块计数问题, 搜索可行, 也可以用并查集。题解采用bfs, 如果当前对角线位置为1, 开始bfs, 如果为2, 表示这个人已经搜索统计过了。
- bfs中的i循环是为了控制当前搜索的点到起点距离相同, j取出队列首个元素, 然后k是遍历朋友关系。
- 如果对角线为1, 说明当前这个人位于一个新的联通块内, count += 1。

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution:
    """
    @param M: a matrix
    @return: the total number of friend circles among all the students
    """
    def BFS(self, student, M):
        queue = []
        queue.append(student)
        while len(queue) :
            size = len(queue)
            for i in range(0, size) :    #控制每轮搜索的点到起点的距离相同
                j = queue[0]
                del queue[0]
                M[j][j] = 2
                for k in range(0, len(M[0])):    #遍历朋友关系
                    if M[j][k] == 1 and M[k][k] == 1:    #如果M[k][k]==1, 说明k没被遍历, 需要继续搜索
                        queue.append(k)
    def findCircleNum(self, M):
        # Write your code here
        count = 0
        for i in range(0, len(M)):
            if M[i][i] == 1 :    #如果当前对角线为1, 说明这个人位于新的联通块内
                count += 1    #计数+1
                self.BFS(i, M) #开始搜索
        return count
```

👍 获赞 1

💬 添加评论



九章算法助教团队

更新于 7/30/2020, 8:04:16 PM

考点:

- bfs
- dfs
- 并查集

题解:

- 本题在于考察基本的联通块计数问题, 搜索可行, 也可以用并查集。题解采用bfs, 如果当前对角线位置为1, 开始bfs, 如果为2, 表示这个人已经搜索统计过了。
- bfs中的i循环是为了控制当前搜索的点到起点距离相同, j取出队列首个元素, 然后k是遍历朋友关系。
- 如果对角线为1, 说明当前这个人位于一个新的联通块内, count++。



```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
public class Solution {
    /**
     * @param M: a matrix
     * @return: the total number of friend circles among all the students
     */
    public int findCircleNum(int[][] M) {
        // Write your code here
        int count = 0;
        for (int i = 0; i < M.length; i++) {
            if (M[i][i] == 1) { //如果当前对角线为1, 说明这个人位于新的联通块内
                count++; //计数+1
                BFS(i, M); //开始搜索
            }
        }
        return count;
    }

    public void BFS(int student, int[][] M) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(student);
        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) { //控制每轮搜索的点与起点的距离相同
                int j = queue.poll();
                M[j][j] = 2;
                for (int k = 0; k < M[0].length; k++) { //遍历朋友关系
                    if (M[j][k] == 1 && M[k][k] == 1) { //如果M[j][k]==1, 说明k没被遍历, 需要继续搜索
                        queue.add(k);
                    }
                }
            }
        }
    }
}
```

👍 获赞 0

💬 1 条评论



九章算法助教团队

更新于 6/9/2020, 7:04:12 AM

考点:

- bfs
- dfs
- 并查集

题解:

- 本题在于考察基本的联通块计数问题, 搜索可行, 也可以用并查集。题解采用bfs, 如果当前对角线位置为1, 开始bfs, 如果为2, 表示这个人已经搜索统计过了。
- bfs中的i循环是为了控制当前搜索的点与起点距离相同, j取出队列首个元素, 然后k是遍历朋友关系。
- 如果对角线为1, 说明当前这个人位于一个新的联通块内, count++

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution {
public:
    /**
     * @param M: a matrix
     * @return: the total number of friend circles among all the students
     */
    void BFS(int student, vector<vector<int>> &M) {
        queue<int> q;
        q.push(student);
        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) { //控制每轮搜索的点到起点的距离相同
                int j = q.front();
                q.pop();
                M[j][j] = 2;
                for (int k = 0; k < M[0].size(); k++) { //遍历朋友关系
                    if (M[j][k] == 1 && M[k][k] == 1) { //如果M[k][k]==1, 说明k没被遍历, 需要继续搜索
                        q.push(k);
                    }
                }
            }
        }
    }

    int findCircleNum(vector<vector<int>> &M) {
        // Write your code here
        int count = 0;
        for (int i = 0; i < M.size(); i++) {
            if (M[i][i] == 1) { //如果当前对角线为1, 说明这个人位于新的联通块内
                count++; //计数+1
                BFS(i, M); //开始搜索
            }
        }
        return count;
    }
};
```

👍 获赞 0

💬 添加评论



卢同学

更新于 12/6/2020, 5:28:19 PM

暴力bfs即可, 和number of island差不多的做法, 唯一的区别是边的定义不太一样, 这里 $M[i][i] == 1$ 就相当于i和i形成了一条边。

```
/**
 * 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括：系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
 * - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
 */
from collections import deque
class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
        if not M:
            return 0
        visited = set()
        res = 0
        for x in range(len(M)):
            if x not in visited:
                visited.add(x)
                self.bfs(x, M, visited)
                res = res + 1
        return res

    def bfs(self, x, M, visited):
        q = deque([x])
        while q:
            x = q.popleft()
            for y in range(len(M)):
                if x == y:
                    continue
                if M[x][y] == 1 and y not in visited:
                    visited.add(y)
                    q.append(y)
```

👍 获赞 2

💬 添加评论

**JeremyXu**

更新于 8/31/2020, 5:15:08 PM

Union Find 时间复杂度 $O(n^2)$  其实就是个联通块问题, 但这里指的图不是矩阵本身, 而是所有的同学, 其实输入矩阵可以看作一个邻接矩阵, 求的是这个邻接矩阵表示的图中, 联通块的个数。

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ /
Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code
 */
from collections import deque

class UnionFind:

    def __init__(self, n):

        self.father = { i : i for i in range(n)}
        self.count = n

    def union(self, a, b):
        root_a = self.find(a)
        root_b = self.find(b)

        if root_a != root_b:
            self.father[root_b] = root_a
            self.count -= 1

    def find(self, point):

        path = []

        while point != self.father[point]:

            path.append(point)

            point = self.father[point]

        for p in path:
            self.father[p] = point

        return point

class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:

        if not M or not M[0]:
            return 0

        n = len(M)

        uf = UnionFind(n)

        for i in range(n):
            for j in range(n):

                if M[i][j] == 1:
                    uf.union(i, j)

        return uf.count
```

👍 获赞 2      💬 添加评论



芋米

更新于 8/1/2020, 8:21:19 PM

时间复杂度 $O(n^2)$  DFS策略, 直线遍历, 扫行, 扫列。凡是扫过的朋友, 都置为visited。

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution {
    public int findCircleNum(int[][] M) {
        int count = 0;
        if (M == null || M.length == 0 || M[0].length == 0) return count;
        boolean[] visited = new boolean[M.length];
        for (int i = 0; i < M.length; i++) {
            if (!visited[i]) {
                dfs(M, visited, i);
                count++;
            }
        }
        return count;
    }

    private void dfs(int[][] M, boolean[] visited, int i) {
        if (visited[i] == false) {
            visited[i] = true;
            for (int j = 0; j < M.length; j++) {
                if (i != j && M[i][j] == 1) {
                    dfs(M, visited, j);
                }
            }
        }
    }
}
```

👍 获赞 2

💬 添加评论



小飞羊

更新于 6/9/2020, 7:03:57 AM

给一个C++版本的造福大家, 然后比较贴切教程里面Union Find的写法, 希望能够好东西和大家分享, 帮助大家!!! 谢谢~

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution {
private:
    vector<int> father;
    int count;

    int find(int num) {
        if (father[num] == num) {
            return num;
        }
        return father[num] = find(father[num]);
    }

    void connect(int a, int b) {
        int rootA = find(a);
        int rootB = find(b);
        if (rootA != rootB) {
            father[rootB] = rootA;
            count--;
        }
    }

public:
    int findCircleNum(vector<vector<int>>& M) {
        if (M.size() == 0) {
            return 0;
        }

        for (int i = 0; i < M[0].size(); ++i) {
            father.push_back(i);
        }
        count = M[0].size();

        for (int i = 0; i < M.size(); ++i) {
            for (int j = 0; j < M[0].size(); ++j) {
                if (M[i][j] == 1 && i != j) {
                    connect(i, j);
                }
            }
        }

        return count;
    }
};
```

👍 获赞 1    💬 添加评论



kuanjung

更新于 6/9/2020, 7:03:52 AM

使用标准并查集模板即可, 在union的cls中定义一个count做优化

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
```

- \* - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
- \* - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
- \* - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
- \* - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
- \* - 更多详情请见官方网站: [http://www.jiuzhang.com/?utm\\_source=code](http://www.jiuzhang.com/?utm_source=code)

\*/

class UnionfindSet {

public int[] parents;

public int[] ranks;

public int count;

public UnionfindSet(int n){

parents = new int[n];

ranks = new int[n];

count = n ;

for(int i = 0; i &lt; parents.length; ++i){

parents[i] = i;

ranks[i] = 1;

}

}

public boolean Union(int u, int v){

int pu = find(u);

int pv = find(v);

if(pu == pv){

return false;

}

if(ranks[pu] &gt; ranks[pv]){

parents[pv] = pu;

}

else if(ranks[pu] &lt; ranks[pv]){

parents[pu] = pv;

}

else{

parents[pu] = pv;

ranks[pv]++;

}

count--;

return true;

}

public int find(int u){

if(parents[u] != u){

parents[u] = find(parents[u]);

}

return parents[u];

}

}

public class Solution {

/\*\*

\* @param M: a matrix

\* @return: the total number of friend circles among all the students

\*/

public int findCircleNum(int[][] M) {

// Write your code here

int n = M.length;

UnionfindSet uf = new UnionfindSet(n);

System.out.println(uf.count);

for(int i = 0; i &lt; n; ++i){

for(int j = i + 1; j &lt; n; ++j){

if(M[i][j] == 1){

uf.Union(i, j);

}

}

}

return uf.count;

(//

课程

(/course/) 旗舰课 (/premium-course/) 1对1私教 (/1on1/)

免费课

Seminar

刷题题解

成功案

消息

更多...

我的课程

(/accounts/profile/)

(/accounts/)

(/accounts/)

邀请有礼

vitation/sha



```
}  
}
```

👍 获赞 1

💬 添加评论



你是疯儿我是傻

更新于 10/29/2020, 6:02:17 PM

java

```
/**  
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。  
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。  
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ /  
Resume / Project 2020版  
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD  
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课  
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code  
 */  
public int findCircleNum(int[][] M) {  
    if(M == null || M[0] == null) return -1;  
  
    final int n = M.length;  
    int numCircle = 0;  
  
    int[] que = new int[1 << 8];  
    for(int c = 0; c < n; c++) {  
        if(M[c][c] != 1) continue;  
  
        numCircle++;  
        int size = 0, cur = 0;  
        que[size++] = c;  
        while(cur < size) {  
            int p = que[cur++];  
            for(int i = 0; i < n; i++) {  
                if(M[p][i] != 1 || M[i][i] != 1) continue;  
                que[size++] = i;  
                M[i][i] |= Integer.MIN_VALUE; //早安, 打工人!  
            }  
        }  
    }  
  
    return numCircle;  
}
```

👍 获赞 0

💬 添加评论



Y同学

更新于 6/9/2020, 7:04:27 AM

用union and find的解法 两层for循环遍历所有的朋友关系, 将一个朋友圈的union到一起 最后数一共有几个朋友圈即可

time complexity:  $O(n^2)$  unionfind中的compressed\_find函数的平均复杂度是 $O(1)$



```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code
 */
class UnionFind:
    def __init__(self, n):
        self.father = {}

        for i in xrange(n):
            self.father[i] = i

    def compressed_find(self, x):
        ancestor = self.father[x]

        while ancestor != self.father[ancestor]:
            ancestor = self.father[ancestor]

        while x != self.father[x]:
            next = self.father[x]
            self.father[x] = ancestor
            x = next

        return ancestor

    def union(self, x, y):
        fa_x = self.compressed_find(x)
        fa_y = self.compressed_find(y)

        if fa_x != fa_y:
            self.father[fa_y] = fa_x

class Solution(object):
    def findCircleNum(self, M):
        """
        :type M: List[List[int]]
        :rtype: int
        """
        if not M or not M[0]:
            return 0

        n = len(M)

        uf = UnionFind(n)

        for i in xrange(n):
            for j in xrange(n):
                if M[i][j] == 1:
                    uf.union(i, j)

        res = set()
        for i in xrange(n):
            for j in xrange(n):
                if M[i][j] == 1:
                    uf.compressed_find(j)
                    res.add(uf.father[j])
        return len(res)
```

👍 获赞 0    💬 添加评论

**Victor**

更新于 6/9/2020, 7:04:19 AM

BFS,

1. build adjacency list
2. for each node, do bfs
3. count the connected part

Time:  $O(n^2)$  Space:  $O(n)$ 

```
/**
 * 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ /
Resume / Project 2020版
 * - Design类课程包括：系统设计 System Design，面向对象设计 OOD
 * - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
 * - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
 */
# version 1, bfs
    # check the corner cases
    from collections import deque

    if not M or not M[0]:
        return 0

    n = len(M)

    # create and fill adjacency list
    adj_list = [set() for i in range(n)]
    for i in range(n):
        for j in range(n):
            if M[i][j]:
                adj_list[i].add(j)
                adj_list[j].add(i)

    count = 0
    # use bfs to find the number of connect parts
    seen = set()
    for i in range(n):
        if i in seen:
            continue

        count += 1
        # bfs
        q = deque([i])
        seen.add(i)
        while q:
            cur = q.popleft()
            for next in adj_list[cur]:
                if next in seen:
                    continue
                q.append(next)
                seen.add(next)

    return count
```

👍 获赞 0

💬 添加评论

加载更多题解

## 进阶课程

视频+互动

直播+互动

直播+互动

互动课

### 九章算法班 2021 版

8周时间精通 57 个核心高频考点, 9 招击破 FLAG、BATJ 算法面试。22....

### 系统架构设计 System Design 2021 版

成为百万架构师必上。30 课时带你快速掌握 18 大系统架构设计知识点与面...

### 九章算法面试高频题冲刺班

每期更新 15% 题目, 考前押题, 一举拿下 FLAG & BATJ Offer

### 面向对象设计 OOD

应届生及亚马逊面试必考, IT 求职必备基础

[首页 \(/?skip\\_redirect=true\)](#) | [联系我们 \(mailto:info@jiuzhang.com\)](mailto:info@jiuzhang.com) | [加入我们 \(/joinus\)](#)

Copyright © 2013-2020 九章算法 浙ICP备19045946号-1  
(<http://www.miibeian.gov.cn/>)

商务合作: [fukesu@jiuzhang.com \(mailto:fukesu@jiuzhang.com\)](mailto:fukesu@jiuzhang.com)



(<http://weibo.com/ninechapter>)



<https://www.zhihu.com/people/crackinterview/>)

(/)