题目描述

Linux操作系统有多个发行版，distrowatch.com提供了各个发行版的资料。这些发行版互相存在关联，例如Ubuntu基于 Debian 开发，而Mint又基于Ubuntu开发，那么我们认为Mint同Debian也存在关联。

发行版集是一个或多个相关存在关联的操作系统发行版，集合内不包含没有关联的发行版。

给你一个 n * n 的矩阵 isConnected，其中 isConnected[i][j] = 1 表示第 i 个发行版和第 j 个发行版直接关联，而 isConnected[i][j] = 0 表示二者不直接相连。

返回最大的发行版集中发行版的数量。

输入描述

第一行输入发行版的总数量N,

之后每行表示各发行版间是否直接相关

输出描述

输出最大的发行版集中发行版的数量

备注

1 ≤ N ≤ 200

用例

输入	4 1 1 0 0 1 1 1 0 0 1 1 0 0 0 0 1
输出	3
说明	Debian(1)和 Unbuntu (2)相关 Mint(3)和Ubuntu(2)相关， EeulerOS(4)和另外三个都不相关， 所以存在两个发行版集，发行版集中发行版的数量分别是3和1，所以输出3。

题目解析

本题可以利用 并查集 求解，本题要求的就是各个连通分量的节点数，并输出最大的连通分量的节点数。

如果大家对并查集还不了解，可以看下这个入门视频：

《算法训练营》进阶篇 01 并查集_哔哩哔哩_bilibili

学会并查集数据结构后，本题的解题难度就很小了，本题题解可以参考

华为OD机试 - 发广播_伏城之外的博客-CSDN博客_信道分配 华为od

解决完本题，可以继续尝试2022.Q4题库的其他并查集算法题：

华为OD机试 - 计算快递主站点_伏城之外的博客-CSDN博客

华为OD机试 - 开心消消乐_伏城之外的博客-CSDN博客

华为OD机试 - 机器人_伏城之外的博客-CSDN博客

华为OD机试 - 快递业务站_伏城之外的博客-CSDN博客

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    n = lines[0] - 0;
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const matrix = lines.map((line) => line.split(" ").map(Number));
    console.log(getResult(matrix, n));
    lines.length = 0;
  }
});

function getResult(matrix, n) {
  const ufs = new UnionFindSet(n);

  for (let i = 0; i < n; i++) {
    for (let j = i + 1; j < n; j++) { // 这里j从i+1开始，是因为矩阵是对称的
      if (matrix[i][j] === 1) {
        ufs.union(i, j);
      }
    }
  }

  // connected对象的属性代表某个连通分量的顶级父节点，属性值代表该连通分量下的节点个数
  const connected = {};

  for (let i = 0; i < n; i++) {
    const fa = ufs.find(ufs.fa[i]);
    connected[fa] ? connected[fa]++ : (connected[fa] = 1);
  }

  // 返回最大节点数
  return Math.max.apply(null, Object.values(connected));
}

// 并查集实现
class UnionFindSet {
  constructor(n) {
    this.fa = new Array(n).fill(0).map((_, i) => i);
    this.count = n;
  }

  find(x) {
    if (x !== this.fa[x]) {
      return (this.fa[x] = this.find(this.fa[x]));
    }
    return x;
  }

  union(x, y) {
    const x_fa = this.find(x);
    const y_fa = this.find(y);

    if (x_fa !== y_fa) {
      this.fa[y_fa] = x_fa;
      this.count--;
    }
  }
}
```

Java算法源码

```java
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    int[][] matrix = new int[n][n];

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        matrix[i][j] = sc.nextInt();
      }
```

Java算法源码

```java
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();

    int[][] matrix = new int[n][n];

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        matrix[i][j] = sc.nextInt();
      }
    }

    System.out.println(getResult(matrix, n));
  }

  public static int getResult(int[][] matrix, int n) {
    UnionFindSet ufs = new UnionFindSet(n);

    for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) { // j从i+1开始，是因为矩阵是对称的
        if (matrix[i][j] == 1) {
          ufs.union(i, j);
        }
      }
    }

    // connected的key代表某个连通分量的顶级父节点，value代表该连通分量下的节点个数
    HashMap<Integer, Integer> connected = new HashMap<>();

    for (int i = 0; i < n; i++) {
      Integer fa = ufs.find(ufs.fa[i]);
      connected.put(fa, connected.getOrDefault(fa, 0) + 1);
    }

    // 返回最大节点数
    return connected.values().stream().max((a, b) -> a - b).get();
  }
}

// 并查集实现
class UnionFindSet {
  int[] fa;
  int count;

  public UnionFindSet(int n) {
    this.count = n;
    this.fa = new int[n];
    for (int i = 0; i < n; i++) this.fa[i] = i;
  }

  public int find(int x) {
    if (x != this.fa[x]) {
      return (this.fa[x] = this.find(this.fa[x]));
    }
    return x;
  }

  public void union(int x, int y) {
    int x_fa = this.find(x);
    int y_fa = this.find(y);

    if (x_fa != y_fa) {
      this.fa[y_fa] = x_fa;
      this.count--;
    }
  }
}
```

Python算法源码

```python
# 并查集
class UnionFindSet:
    def __init__(self, n):
        self.fa = [idx for idx in range(n)]
        self.count = n

    def find(self, x):
        if x != self.fa[x]:
            self.fa[x] = self.find(self.fa[x])
            return self.fa[x]
        return x

    def union(self, x, y):
        x_fa = self.find(x)
        y_fa = self.find(y)
```

Python算法源码

```python
# 并查集
class UnionFindSet:
    def __init__(self, n):
        self.fa = [idx for idx in range(n)]
        self.count = n

    def find(self, x):
        if x != self.fa[x]:
            self.fa[x] = self.find(self.fa[x])
            return self.fa[x]
        return x

    def union(self, x, y):
        x_fa = self.find(x)
        y_fa = self.find(y)

        if x_fa != y_fa:
            self.fa[y_fa] = x_fa
            self.count -= 1


n = int(input())

matrix = []
for i in range(n):
    matrix.append(list(map(int, input().split())))

ufs = UnionFindSet(n)

for i in range(n):
    for j in range(i + 1, n):  # 这里j从i+1开始，是因为矩阵是对称的
        if matrix[i][j] == 1:
            ufs.union(i, j)

# connected字典的属性代表某个连通分量的顶级父节点，属性值代表该连通分量下的节点个数
connected = {}

for i in range(n):
    fa = ufs.find(ufs.fa[i])
    connected[fa] = connected.get(fa, 0) + 1

# 返回最大节点数
print(max(connected.values()))
```