

题目描述

快递业务范围有 N 个站点，A 站点与 B 站点可以中转快递，则认为 A-B 站可达，如果 A-B 可达，B-C 可达，则 A-C 可达。
现在给 N 个站点编号 0、1、...n-1，用 $s[i][j]$ 表示 i-j 是否可达， $s[i][j] = 1$ 表示 i-j 可达， $s[i][j] = 0$ 表示 i-j 不可达。
现用二维数组给定 N 个站点的可达关系，请计算至少选择从几个主站点出发，才能可达所有站点（覆盖所有站点业务）。
说明： $s[i][j]$ 与 $s[j][i]$ 取值相同。

输入描述

第一行输入为 N，N 表示站点个数。 $1 < N < 10000$
之后 N 行表示站点之间的可达关系，第 i 行第 j 个数值表示编号为 i 和 j 之间是否可达。

输出描述

输出站点个数，表示至少需要多少个主站点。

用例

输入	4 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1
输出	1
说明	选择 0 号站点作为主站点，0 站点可达其他所有站点，所以至少选择 1 个站点作为主站才能覆盖所有站点业务。

输入	4 1 1 0 0 1 1 0 0 0 0 1 0 0 0 0 1
输出	3
说明	选择 0 号站点可以覆盖 0、1 站点，选择 2 号站点可以覆盖 2 号站点，选择 3 号站点可以覆盖 3 号站点，所以至少选择 3 个站点作为主站才能覆盖所有站点业务。

题目解析

本题其实就是求解 [连通分量](#) 的个数，可以用并查集求解。
本题类似于[华为机试 - 发广播_伏城之外的博客-CSDN博客_服务器广播](#) [华为机试题解](#)可以参考[链接](#)博客

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 1) {
```

 伏城之外 已关注 1 0 3 1 专栏目录 已订阅

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 1) {
15     n = lines[0] - 0;
16   }
17
18   if (n && lines.length === n + 1) {
19     lines.shift();
20     const matrix = lines.map((line) => line.split(" "));
21
22     console.log(getResult(matrix, n));
23
24     lines.length = 0;
25   }
26 });
27
28 function getResult(matrix, n) {
29   const ufs = new UnionFindSet(n);
30
31   for (let i = 0; i < n; i++) {
32     for (let j = i + 1; j < n; j++) {
33       if (matrix[i][j] == "1") {
34         ufs.union(i, j);
35       }
36     }
37   }
38
39   return ufs.count;
40 }
41
42 class UnionFindSet {
43   constructor(n) {
44     this.fa = new Array(n).fill(0).map((_, i) => i);
45     this.count = n;
46   }
47
48   find(x) {
49     if (x !== this.fa[x]) {
50       return (this.fa[x] = this.find(this.fa[x]));
51     }
52     return x;
53   }
54
55   union(x, y) {
56     const x_fa = this.find(x);
57     const y_fa = this.find(y);
58
59     if (x_fa !== y_fa) {
60       this.fa[y_fa] = x_fa;
61       this.count--;
62     }
63   }
64 }
```

Java算法源码

```
1  import java.util.Scanner;
2
3  public class Main {
4    public static void main(String[] args) {
5      Scanner sc = new Scanner(System.in);
6
7      int n = sc.nextInt();
8
9      int[][] matrix = new int[n][n];
10
11     for (int i = 0; i < n; i++) {
12       for (int j = 0; j < n; j++) {
13         matrix[i][j] = sc.nextInt();
14       }
15     }
16
17     System.out.println(getResult(matrix, n));
18   }
19
20   public static int getResult(int[][] matrix, int n) {
21     UnionFindSet ufs = new UnionFindSet(n);
22
23     for (int i = 0; i < n; i++) {
```



伏城之外 已关注



1



3



3



1



专栏目录

已订阅

Java算法源码

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         int n = sc.nextInt();
8
9         int[][] matrix = new int[n][n];
10
11         for (int i = 0; i < n; i++) {
12             for (int j = 0; j < n; j++) {
13                 matrix[i][j] = sc.nextInt();
14             }
15         }
16
17         System.out.println(getResult(matrix, n));
18     }
19
20     public static int getResult(int[][] matrix, int n) {
21         UnionFindSet ufs = new UnionFindSet(n);
22
23         for (int i = 0; i < n; i++) {
24             for (int j = 0; j < n; j++) {
25                 if (matrix[i][j] == 1) {
26                     ufs.union(i, j);
27                 }
28             }
29         }
30
31         return ufs.count;
32     }
33 }
34
35 class UnionFindSet {
36     int[] fa;
37     int count;
38
39     public UnionFindSet(int n) {
40         this.count = n;
41         this.fa = new int[n];
42         for (int i = 0; i < n; i++) this.fa[i] = i;
43     }
44
45     public int find(int x) {
46         if (x != this.fa[x]) {
47             return (this.fa[x] = this.find(this.fa[x]));
48         }
49         return x;
50     }
51
52     public void union(int x, int y) {
53         int x_fa = this.find(x);
54         int y_fa = this.find(y);
55
56         if (x_fa != y_fa) {
57             this.fa[y_fa] = x_fa;
58             this.count--;
59         }
60     }
61 }
```

Python算法源码

```
1 # 并查集
2 class UnionFindSet:
3     def __init__(self, n):
4         self.fa = [idx for idx in range(n)]
5         self.count = n
6
7     def find(self, x):
8         if x != self.fa[x]:
9             self.fa[x] = self.find(self.fa[x])
10        return self.fa[x]
11
12    def union(self, x, y):
13        x_fa = self.find(x)
14        y_fa = self.find(y)
15
16        if x_fa != y_fa:
17            self.fa[y_fa] = x_fa
18            self.count -= 1
19
20 n = int(input())
21
22 matrix = []
23 for i in range(n):
24     matrix.append(list(map(int, input().split())))
```



伏城之外 已关注



1



0



3



0



1



0

专栏目录

已订阅

Python算法源码

```
1 # 并查集
2 class UnionFindSet:
3     def __init__(self, n):
4         self.fa = [idx for idx in range(n)]
5         self.count = n
6
7     def find(self, x):
8         if x != self.fa[x]:
9             self.fa[x] = self.find(self.fa[x])
10            return self.fa[x]
11        return x
12
13    def union(self, x, y):
14        x_fa = self.find(x)
15        y_fa = self.find(y)
16
17        if x_fa != y_fa:
18            self.fa[y_fa] = x_fa
19            self.count -= 1
20
21
22 n = int(input())
23
24 matrix = []
25 for i in range(n):
26     matrix.append(list(map(int, input().split())))
27
28 ufs = UnionFindSet(n)
29
30 for i in range(n):
31     for j in range(i+1, n):
32         if matrix[i][j] == 1:
33             ufs.union(i, j)
34
35 print(ufs.count)
```

[复制](#)