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

```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) {
```

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(" "));

    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++) {
      if (matrix[i][j] == "1") {
        ufs.union(i, j);
      }
    }
  }

  return ufs.count;
}

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.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++) {
```

Java算法源码

```java
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 = 0; j < n; j++) {
                if (matrix[i][j] == 1) {
                    ufs.union(i, j);
                }
            }
        }

        return ufs.count;
    }
}

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)

        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())))
```

Python算法源码

复制

```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):
        if matrix[i][j] == 1:
            ufs.union(i, j)

print(ufs.count)
```