题目描述

给定一个 m x n 的矩阵，由若干字符 'X' 和 'O'构成，'X'表示该处已被占据，'O'表示该处空闲，请找到最大的单入口空闲区域。

解释：

空闲区域是由连通的'O'组成的区域，位于边界的'O'可以构成入口，

单入口空闲区域即有且只有一个位于边界的'O'作为入口的由连通的'O'组成的区域。
如果两个元素在水平或垂直方向相邻，则称它们是“连通”的。

输入描述

第一行输入为两个数字，第一个数字为行数m，第二个数字为列数n，两个数字以空格分隔，1<=m,n<=200。

剩余各行为矩阵各行元素，元素为'X'或'O'，各元素间以空格分隔。

输出描述

若有唯一符合要求的最大单入口空闲区域，输出三个数字

- 第一个数字为入口行坐标（0~m-1）
- 第二个数字为入口列坐标（0~n-1）
- 第三个数字为区域大小

三个数字以空格分隔；

若有多个符合要求，则输出区域大小最大的，若多个符合要求的单入口区域的区域大小相同，则此时只需要输出区域大小，不需要输出入口坐标。

若没有，输出NULL。

用例

输入	4 4 X X X X X O O X X O O X X O X X
输出	3 1 5
说明	存在最大单入口区域，入口坐标(3,1)，区域大小5

输入	4 5 X X X X X O O O O X X O O O X X O X X O
输出	3 4 1
说明	存在最大单入口区域，入口坐标（3,4），区域大小1

输入	5 4 X X X X X O O O X O O O X O O X X X X X
输出	NULL
说明	不存在最大单入口区域

	5 4 X X X X X O O O

用例

输入	4 4 X X X X X O O X X O O X X O X X
输出	3 1 5
说明	存在最大单入口区域，入口坐标(3,1)，区域大小5

输入	4 5 X X X X X O O O O X X O O O X X O X X O
输出	3 4 1
说明	存在最大单入口区域，入口坐标（3,4），区域大小1

输入	5 4 X X X X X O O O X O O O X O O X X X X X
输出	NULL
说明	不存在最大单入口区域

输入	5 4 X X X X X O O O X X X X X O O O X X X X
输出	3
说明	存在两个大小为3的最大单入口区域，两个入口坐标分别为(1,3)、(3,3)

题目解析

本题可以使用 深度优先搜索 来解题。

首先，我们可以遍历矩阵元素，当遍历到“O”时，已该“O”的坐标位置开始向其上、下、左、右方向开始深度优先搜索，每搜索到一个“O”，则该空闲区域数量+1，如果搜索到的“O”的坐标位置处于矩阵第一列，或最后一列，或者第一行，或者最后一行，那么该“O”位置就是空闲区域的入口位置，我们将其缓存到out数组中。

当所有深度优先搜索的分支都搜索完了，则判断out统计的入口数量，

1. 如果只有1个，则该空闲区域是符合题意得单入口空闲区域，输出入口坐标位置，以及空闲区域数量。
2. 如果有多个，则该区域不符合要求

另外，我们还需要定义一个check集合来缓存，已经被递归过的"O"位置，避免重复的深度优先搜索。

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

  if (lines.length === 1) {
    [n, m] = lines[0].split(" ").map(Number);
  }

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

function getResult(matrix, n, m) {
```

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, m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [n, m] = lines[0].split(" ").map(Number);
  }

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

function getResult(matrix, n, m) {
  const checked = new Set();

  const offset = [
    [0, -1],
    [0, 1],
    [-1, 0],
    [1, 0],
  ];

  function dfs(i, j, count, out) {
    const pos = `${i}-${j}`;

    if (
      i < 0 ||
      i >= n ||
      j < 0 ||
      j >= m ||
      matrix[i][j] === "X" ||
      checked.has(pos)
    )
      return count;

    checked.add(pos);

    if (i === 0 || i === n - 1 || j === 0 || j === m - 1) out.push([i, j]);

    count++;

    for (let k = 0; k < offset.length; k++) {
      const [offsetX, offsetY] = offset[k];
      const newI = i + offsetX;
      const newJ = j + offsetY;
      count = dfs(newI, newJ, count, out);
    }

    return count;
  }

  const ans = [];

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      if (matrix[i][j] === "O" && !checked.has(`${i}-${j}`)) {
        const out = [];
        const count = dfs(i, j, 0, out);
        if (out.length === 1) {
          ans.push([...out[0], count]);
        }
      }
    }
  }

  if (!ans.length) return "NULL";

  ans.sort((a, b) => b[2] - a[2]);

  if (ans.length === 1 || ans[0][2] > ans[1][2]) {
    return ans[0].join(" ");
  } else {
    return ans[0][2];
  }
}
```

Java算法源码

Java算法源码

```java
import java.util.*;

public class Main {
    static int n;
    static int m;
    static String[][] matrix;
    static int[][] offset = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};
    static HashSet<String> checked = new HashSet<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        m = sc.nextInt();

        matrix = new String[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.next();
            }
        }

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

    public static String getResult(String[][] matrix, int n, int m) {
        ArrayList<Integer[]> ans = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if ("O".equals(matrix[i][j]) && !checked.contains(i + "-" + j)) {
                    ArrayList<Integer[]> enter = new ArrayList<>();
                    int count = dfs(i, j, 0, enter);
                    if (enter.size() == 1) {
                        Integer[] pos = enter.get(0);
                        Integer[] an = {pos[0], pos[1], count};
                        ans.add(an);
                    }
                }
            }
        }

        if (ans.size() == 0) return "NULL";
        ans.sort((a, b) -> b[2] - a[2]);

        if (ans.size() == 1 || ans.get(0)[2] > ans.get(1)[2]) {
            StringJoiner sj = new StringJoiner(" ", "", "");
            for (Integer ele : ans.get(0)) {
                sj.add(ele + "");
            }
            return sj.toString();
        } else {
            return ans.get(0)[2] + "";
        }

    }

    public static int dfs(int i, int j, int count, ArrayList<Integer[]> enter) {
        String pos = i + "-" + j;

        if (i < 0 || i >= n || j < 0 || j >= m || "X".equals(matrix[i][j]) || checked.contains(pos)) {
            return count;
        }

        checked.add(pos);

        if (i == 0 || i == n - 1 || j == 0 || j == m - 1) enter.add(new Integer[]{i, j});

        count++;

        for (int k = 0; k < offset.length; k++) {
            int offsetX = offset[k][0];
            int offsetY = offset[k][1];

            int newI = i + offsetX;
            int newJ = j + offsetY;
            count = dfs(newI, newJ, count, enter);
        }

        return count;
    }
}
```

Python算法源码

```python
# 输入获取
m, n = map(int, input().split())
matrix = [input().split() for i in range(m)]


# 算法入口
```

Python算法源码

```python
# 输入获取
m, n = map(int, input().split())
matrix = [input().split() for i in range(m)]


# 算法入口
def getResult(matrix, m, n):
    checked = set()

    offsets = ((0, -1), (0, 1), (-1, 0), (1, 0))

    def dfs(i, j, count, out):
        pos = f"{i}-{j}"

        if i < 0 or i >= m or j < 0 or j >= n or matrix[i][j] == "X" or pos in checked:
            return count

        checked.add(pos)

        if i == 0 or i == m - 1 or j == 0 or j == n - 1:
            out.append([i, j])

        count += 1

        for offsetX, offsetY in offsets:
            newI = i + offsetX
            newJ = j + offsetY
            count = dfs(newI, newJ, count, out)

        return count

    ans = []

    for i in range(m):
        for j in range(n):
            if matrix[i][j] == "O" and f"{i}-{j}" not in checked:
                out = []
                count = dfs(i, j, 0, out)
                if len(out) == 1:
                    tmp = out[0][:]
                    tmp.append(count)
                    ans.append(tmp)

    if len(ans) == 0:
        return "NULL"

    ans.sort(key=lambda x: -x[2])

    if len(ans) == 1 or ans[0][2] > ans[1][2]:
        return " ".join(map(str, ans[0]))
    else:
        return ans[0][2]


# 算法调用
print(getResult(matrix, m, n))
```