华为OD机试 - 宜居星球改造计划（Java & JS & Python）

原创 伏城之外 已于 2023-05-21 16:05:39 修改 746 收藏 版权
分类专栏： 华为OD机试（Java & JS & Python） # 华为OD机试2023B # 华为OD机试2023A
文章标签： 算法 华为机试 Java JavaScript Python

题目描述

2XXX年，人类通过对火星的大气进行宜居改造分析，使得火星已在理论上具备人类宜居的条件；

由于技术原因，无法一次性将火星大气全部改造，只能通过局部处理形式；

假设将火星待改造的区域为row * column的网格，每个网格有3个值，宜居区、可改造区、死亡区，使用YES、NO、NA代替，YES表示该网格已经完成大气改造，NO表示该网格未进行改造，后期可进行改造，NA表示死亡区，不作为判断是否改造完的宜居，无法穿过；

初始化下，该区域可能存在多个宜居区，并目每个宜居区能同时在每个大阳日单位向上下左右四个方向的相邻格子进行扩散，自动将4个方向相邻的真空区改造成宜居区；

请计算这个待改造区域的网格中，可改造区是否能全部成宜居区，如果可以，则返回改造的大阳日天教，不可以则返回-1

输入描述

输入row * column个网格数据，每个网格值 枚举值 如下: YES，NO，NA；

样例:

YES YES NO
NO NO NO
NA NO YES

输出描述

可改造区是否能全部变成宜居区，如果可以，则返回改造的太阳日天数，不可以则返回-1。

备注

grid[i][j]只有3种情况，YES、NO、NA

- row == grid.length
- column == grid[i].length
- 1 ≤ row, column ≤ 8

用例

输入	YES YES NO NO NO NO YES NO NO
输出	2
说明	经过2个太阳日，完成宜居改造

输入	YES NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO
输出	6
说明	经过6个太阳日，可完成改造

输入	NO NA
输出	-1
说明	无改造初始条件，无法进行改造

输入	YES NO NO YES NO NO YES NO NO YES NA NA YES NO NA NO
输出	-1
说明	-1 // 右下角的区域，被周边三个死亡区挡住，无法实现改造

题目解析

本题是图的多源 BFS 的经典题，解析可以参考：

华为OD机试 - 计算疫情扩散时间（Java & JS & Python）_伏城之外的博客-CSDN博客

注意本题没有输入截止条件，因此，我将输入空行作为输入截止条件。

Java算法源码

```java
import java.util.ArrayList;
import java.util.Scanner;

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

    ArrayList<String[]> matrix = new ArrayList<>();
    while (sc.hasNextLine()) {
      String line = sc.nextLine();
      if ("".equals(line)) {
        System.out.println(getResult(matrix));
        break;
      } else {
        matrix.add(line.split(" "));
      }
    }
  }

  private static int getResult(ArrayList<String[]> matrix) {
    int row = matrix.size();
    int col = matrix.get(0).length;

    // 记录宜居取坐标位置
    ArrayList<int[]> queue = new ArrayList<>();

    // 记录可改造区数量
    int need = 0;

    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        String val = matrix.get(i)[j];
        switch (val) {
          case "YES":
            queue.add(new int[] {i, j});
            break;
          case "NO":
            need++;
            break;
        }
      }
    }

    // 如果没有宜居区，则无法改造，直接返回-1
    if (queue.size() == 0) return -1;
    // 如果全是宜居区，则无需改造，直接返回0
    if (queue.size() == row * col) return 0;

    // 记录花费的天数
    int day = 0;

    // 上，下，左，右四个方向的偏移量
    int[][] offsets = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    // 图的多源BFS模板
    while (queue.size() > 0 && need > 0) {
      ArrayList<int[]> newQueue = new ArrayList<>();

      for (int[] pos : queue) {
        int x = pos[0], y = pos[1];
        for (int[] offset : offsets) {
          // 上，下，左，右四个方向扩散
          int newX = x + offset[0];
          int newY = y + offset[1];

          // 如果新位置没有越界，且为NO，则可以被改造
          if (newX >= 0
              && newX < row
              && newY >= 0
              && newY < col
              && "NO".equals(matrix.get(newX)[newY])) {
            matrix.get(newX)[newY] = "YES";
            newQueue.add(new int[] {newX, newY});
            need--;
          }
        }
      }

      day++;
      queue = newQueue;
    }

    if (need == 0) return day;
    else return -1;
  }
}
```

JS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 = [];
rl.on("line", (line) => {
  if (line == "") {
    const matrix = lines.map((arr) => arr.split(" "));
    console.log(getResult(matrix));
    lines.length = 0;
  } else {
    lines.push(line);
  }
});

function getResult(matrix) {
  const row = matrix.length;
  const col = matrix[0].length;

  // 记录宜居取坐标位置
  let queue = [];
  // 记录可改造区数量
  let need = 0;

  for (let i = 0; i < row; i++) {
    for (let j = 0; j < col; j++) {
      switch (matrix[i][j]) {
        case "YES":
          queue.push([i, j]);
          break;
        case "NO":
          need++;
          break;
      }
    }
  }

  // 如果没有宜居区，则无法改造，直接返回-1
  if (queue.length == 0) return -1;
  // 如果全是宜居区，则无需改造，直接返回0
  if (queue.length == row * col) return 0;

  // 记录花费的天数
  let day = 0;
  // 上，下，左，右四个方向的偏移量
  const offsets = [
    [-1, 0],
    [1, 0],
    [0, -1],
    [0, 1],
  ];

  // 图的多源BFS模板
  while (queue.length > 0 && need > 0) {
    const newQueue = [];

    for (let [x, y] of queue) {
      for (let offset of offsets) {
        // 上，下，左，右四个方向扩散
        const newX = x + offset[0];
        const newY = y + offset[1];

        // 如果新位置没有越界，且为NO，则可以被改造
        if (
          newX >= 0 &&
          newX < row &&
          newY >= 0 &&
          newY < col &&
          "NO" == matrix[newX][newY]
        ) {
          matrix[newX][newY] = "YES";
          newQueue.push([newX, newY]);
          need--;
        }
      }
    }

    day++;
    queue = newQueue;
  }

  if (need == 0) return day;
  else return -1;
}
```

Python算法源码

```python
# 算法入口
def getResult(matrix):
    row = len(matrix)
    col = len(matrix[0])

    # 记录宜居取坐标位置
    queue = []
    # 记录可改造区数量
    need = 0

    for i in range(row):
        for j in range(col):
            if matrix[i][j] == "YES":
                queue.append([i, j])
            elif matrix[i][j] == "NO":
                need += 1

    # 如果没有宜居区，则无法改造，直接返回-1
    if len(queue) == 0:
        return -1
    # 如果全是宜居区，则无需改造，直接返回0
    elif len(queue) == row * col:
        return 0

    # 记录花费的天数
    day = 0
    # 上，下，左，右四个方向的偏移量
    offsets = ((-1, 0), (1, 0), (0, -1), (0, 1))

    # 图的多源BFS模板
    while len(queue) > 0 and need > 0:
        newQueue = []

        for x, y in queue:
            for offsetX, offsetY in offsets:
                # 上，下，左，右四个方向扩散
                newX = x + offsetX
                newY = y + offsetY

                # 如果新位置没有越界，且为NO，则可以被改造
                if row > newX >= 0 and col > newY >= 0 and matrix[newX][newY]
                    matrix[newX][newY] = "YES"
                    newQueue.append([newX, newY])
                    need -= 1

        day += 1
        queue = newQueue

    if need == 0:
        return day
    else:
        return -1


# 输入获取
matrix = []
while True:
    line = input()
    if line == "":
        print(getResult(matrix))
        break
    else:
        matrix.append(line.split())
```