题目描述

给一个二维数组nums，对于每一个元素nums[i]，找出距离最近的且值相等的元素，输出横纵坐标差值的绝对值之和，如果没有等值元素，则输出-1。

输入描述

输入第一行为二维数组的行
输入第二行为二维数组的列
输入的数字以空格隔开。

输出描述

数组形式返回所有坐标值。

用例

输入	3 5 0 3 5 4 2 2 5 7 8 3 2 5 4 2 4
输出	[[-1, 4, 2, 3, 3], [1, 1, -1, -1, 4], [1, 1, 2, 3, 2]]
说明	无

题目解析

我的解题思路如下：

首先遍历输入的矩阵，将相同数字的位置整理到一起。

然后再遍历一次输入的矩阵，找到和遍历元素相同的其他数字（通过上一步统计结果快速找到），求距离，并保留最小的距离。

上面逻辑的算法复杂度为O(n*m*k)，其中n是输入矩阵行，m是输入矩阵列，k是每组相同数字的个数，可能会达到O(N^3)的时间复杂度。

有一个优化点就是，遍历元素A找其他相同数字B时计算的距离可以缓存，这样的话，当遍历元素B时找其他相同数字A时，就可以从缓存中读取已经计算好的距离了，而不是重新计算。但是这样会浪费较多空间。

实际机试时，可以看用例数量级来决定是否要做上面这个优化。如果数量级很小，则无需上面优化。如果数量级较大，则有优化的必要。

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 === 2) {
    n = lines[0] - 0;
    m = lines[1] - 0;
  }

  if (n && lines.length === n + 2) {
    const matrix = lines.slice(2).map((line) => line.split(" ").map(Number));
    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 === 2) {
    n = lines[0] - 0;
    m = lines[1] - 0;
  }

  if (n && lines.length === n + 2) {
    const matrix = lines.slice(2).map((line) => line.split(" ").map(Number));
    console.log(getResult(matrix, n, m));
    lines.length = 0;
  }
});

function getResult(matrix, n, m) {
  const nums = {};

  // 统计输入矩阵中，相同数字的位置
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      const num = matrix[i][j];
      nums[num] ? nums[num].push([i, j]) : (nums[num] = [[i, j]]);
    }
  }

  // 遍历矩阵每一个元素，和其他相同数字的位置求距离，，取最小距离
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      const num = matrix[i][j];

      let minDis = Infinity;
      nums[num].forEach((pos) => {
        const [i1, j1] = pos;
        if (i1 !== i || j1 !== j) {
          let dis = Math.abs(i1 - i) + Math.abs(j1 - j);
          minDis = Math.min(minDis, dis);
        }
      });

      matrix[i][j] = minDis === Infinity ? -1 : minDis;
    }
  }

  return JSON.stringify(matrix).replace(/,/g, ", ");
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
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 m = sc.nextInt();

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

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

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

    // 统计输入矩阵中，相同数字的位置
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        Integer num = matrix[i][j];
        Integer[] arr = {i, j};
        nums.putIfAbsent(num, new ArrayList<>());
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
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 m = sc.nextInt();

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

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

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

    // 统计输入矩阵中，相同数字的位置
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        Integer num = matrix[i][j];
        Integer[] arr = {i, j};
        nums.putIfAbsent(num, new ArrayList<>());
        nums.get(num).add(arr);
      }
    }

    // 遍历矩阵每一个元素，和其他相同数字的位置求距离，，取最小距离
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        int num = matrix[i][j];

        int minDis = Integer.MAX_VALUE;
        for (Integer[] pos : nums.get(num)) {
          int i1 = pos[0];
          int j1 = pos[1];

          if (i1 != i || j1 != j) {
            int dis = Math.abs(i1 - i) + Math.abs(j1 - j);
            minDis = Math.min(minDis, dis);
          }
        }

        matrix[i][j] = minDis == Integer.MAX_VALUE ? -1 : minDis;
      }
    }

    return Arrays.toString(Arrays.stream(matrix).map(Arrays::toString).toArray(String[]::new));
  }
}
```

Python算法源码

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


# 算法入口
def getResult(matrix, n, m):
    nums = {}

    # 统计输入矩阵中，相同数字的位置
    for i in range(n):
        for j in range(m):
            num = matrix[i][j]
            if nums.get(num) is None:
                nums[num] = [[i, j]]
            else:
                nums[num].append([i, j])

    # 遍历矩阵每一个元素，和其他相同数字的位置求距离，取最小距离
    for i in range(n):
        for j in range(m):
            num = matrix[i][j]

            minDis = None
            for i1, j1 in nums[num]:
                if i1 != i or j1 != j:
                    dis = abs(i1 - i) + abs(j1 - j)
                    if minDis is None:
```

Python算法源码

复制

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


# 算法入口
def getResult(matrix, n, m):
    nums = {}

    # 统计输入矩阵中，相同数字的位置
    for i in range(n):
        for j in range(m):
            num = matrix[i][j]
            if nums.get(num) is None:
                nums[num] = [[i, j]]
            else:
                nums[num].append([i, j])

    # 遍历矩阵每一个元素，和其他相同数字的位置求距离，取最小距离
    for i in range(n):
        for j in range(m):
            num = matrix[i][j]

            minDis = None
            for i1, j1 in nums[num]:
                if i1 != i or j1 != j:
                    dis = abs(i1 - i) + abs(j1 - j)
                    if minDis is None:
                        minDis = dis
                    else:
                        minDis = min(minDis, dis)

            matrix[i][j] = -1 if minDis is None else minDis

    return matrix


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