第 130 场双周赛

1. 100299. 判断矩阵是否满足条件

给你一个大小为 `m x n` 的二维矩阵 `grid` 。你需要判断每一个格子 `grid[i][j]` 是否满足：

- 如果它下面的格子存在，那么它需要等于它下面的格子，也就是 `grid[i][j] == grid[i + 1][j]` 。
- 如果它右边的格子存在，那么它需要不等于它右边的格子，也就是 `grid[i][j] != grid[i][j + 1]` 。

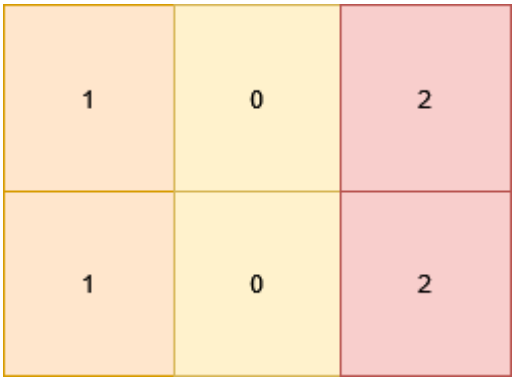
如果 **所有** 格子都满足以上条件，那么返回 `true` ，否则返回 `false` 。

示例 1：

输入： grid = [[1,0,2],[1,0,2]]

输出： true

解释：



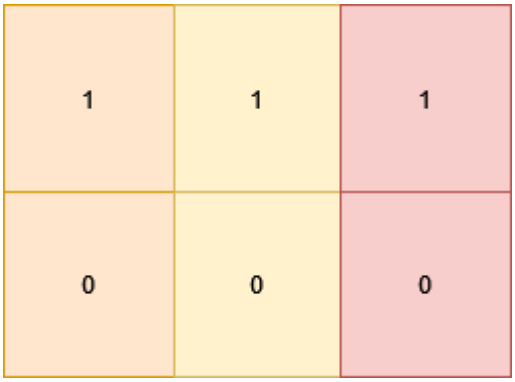
网格图中所有格子都符合条件。

示例 2：

输入： grid = [[1,1,1],[0,0,0]]

输出： false

解释：



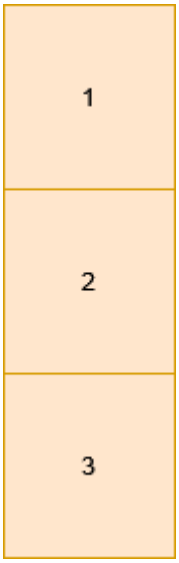
同一行中的格子值都相等。

示例 3：

输入： grid = [[1],[2],[3]]

输出： false

解释：



同一列中的格子值不相等。

提示：

- `1 <= n, m <= 10`
- `0 <= grid[i][j] <= 9`

思路：

按照题意，遍历矩阵依次判断即可

1. Java

```java
class Solution {
    public boolean satisfiesConditions(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        boolean check = true;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (i + 1 < m && grid[i + 1][j] != grid[i][j]) {
                    check = false;
                    break;
                }
                if (j + 1 < n && grid[i][j + 1] == grid[i][j]) {
                    check = false;
                    break;
                }
            }
        }
        return check;
    }
}
```

2. Python

```python
class Solution:
    def satisfiesConditions(self, grid: List[List[int]]) -> bool:
        m, n = len(grid), len(grid[0])
        check = True
        for i in range(m):
            for j in range(n):
                if i + 1 < m and grid[i + 1][j] != grid[i][j]:
                    check = False
                    break
                if j + 1 < n and grid[i][j + 1] == grid[i][j]:
                    check = False
                    break
        return check
```

2. 100302. 正方形中的最多点数

给你一个二维数组 `points` 和一个字符串 `s` ，其中 `points[i]` 表示第 `i` 个点的坐标， `s[i]` 表示第 `i` 个点的 **标签** 。

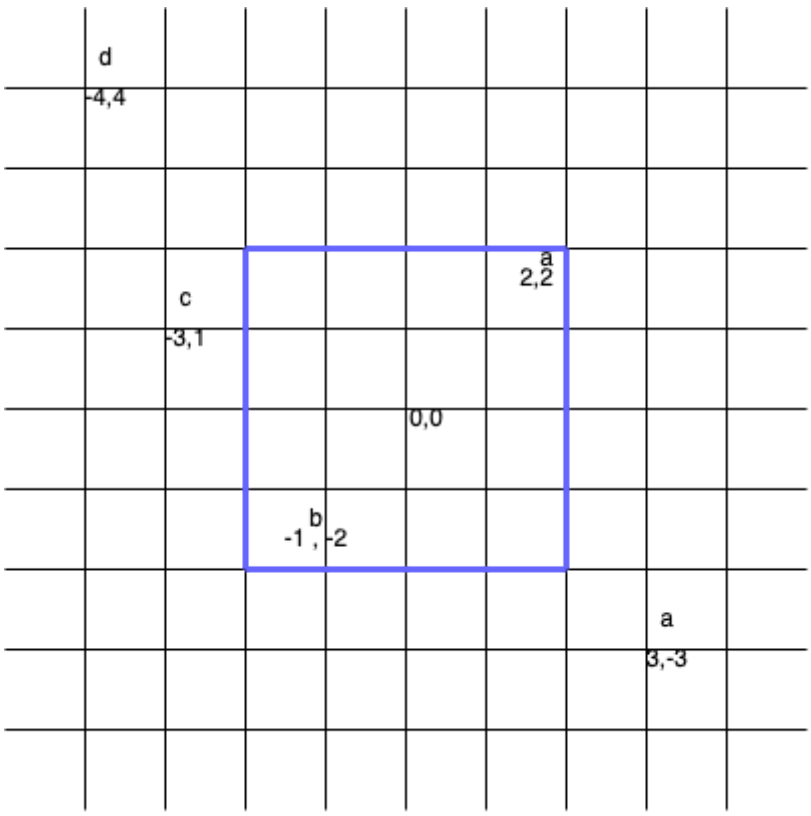
如果一个正方形的中心在 `(0, 0)` ，所有边都平行于坐标轴，且正方形内 **不** 存在标签相同的两个点，那么我们称这个正方形是 **合法** 的。

请你返回 **合法** 正方形中可以包含的 **最多** 点数。

注意：

- 如果一个点位于正方形的边上或者在边以内，则认为该点位于正方形内。
- 正方形的边长可以为零。

示例 1：



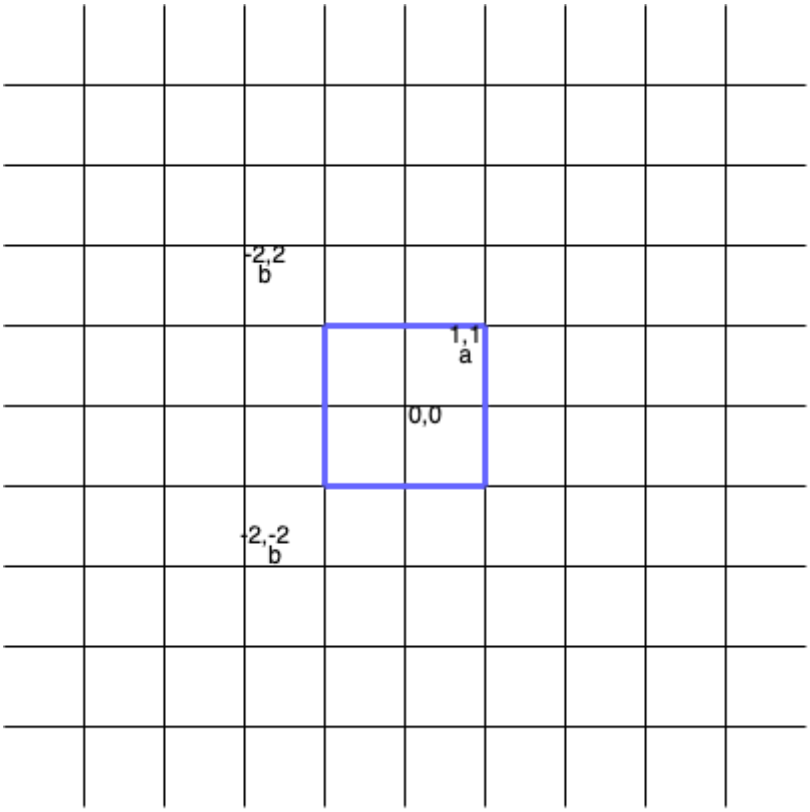
输入： points = [[2,2],[-1,-2],[-4,4],[-3,1],[3,-3]], s = "abdca"

输出： 2

解释：

边长为 4 的正方形包含两个点 `points[0]` 和 `points[1]` 。

示例 2：



输入： points = [[1,1],[-2,-2],[-2,2]], s = "abb"

输出： 1

解释：

边长为 2 的正方形包含 1 个点 `points[0]` 。

示例 3：

输入： points = [[1,1],[-1,-1],[2,-2]], s = "ccd"

输出： 0

解释：

任何正方形都无法只包含 `points[0]` 和 `points[1]` 中的一个点，所以合法正方形中都不包含任何点。

提示：

- `1 <= s.length, points.length <= 105`
- `points[i].length == 2`
- `-109 <= points[i][0], points[i][1] <= 109`
- `s.length == points.length`
- `points` 中的点坐标互不相同。
- `s` 只包含小写英文字母。

思路：

由于正方形边长越大，所能包括的点就越多，所以我们可以依次扩大正方形的边长，不断判断这个边长中有没有重复的点

具体算法思路实现：假设每个点的坐标为

$$(x, y)$$

因为我们依次扩大正方形的边长，所以我们就需要把一些可以归为一个边长的点分到一组中，如何判断可以归为一个边长，也就是相同的

$$max(abs(x), abs(y))$$

这样分组完毕之后，我们对边长进行排序，排序完成之后开始遍历，一旦遇到重复的点直接返回答案，因为小的都有重复点了，那么大的一定是包含重复点的

1. Java

```java
class Solution {
    public int maxPointsInsideSquare(int[][] points, String s) {
        Map<Integer, List<Character>> d = new HashMap<>();
        for (int i = 0; i < points.length; i++) {
            int x = points[i][0];
            int y = points[i][1];
            int m = Math.max(Math.abs(x), Math.abs(y));
            if (!d.containsKey(m)) {
                d.put(m, new ArrayList<>());
            }
            d.get(m).add(s.charAt(i));
        }
        int ans = 0;
        Map<Character, Integer> cnt = new HashMap<>();
        List<Integer> sortedKeys = new ArrayList<>(d.keySet());
        Collections.sort(sortedKeys);
        for (int k : sortedKeys) {
            for (char c : d.get(k)) {
                if (!cnt.containsKey(c)) {
                    cnt.put(c, 1);
                } else {
                    return ans;
                }
            }
            ans += d.get(k).size();
        }
        return ans;
    }
}
```

2. Python

```python
class Solution:
    def maxPointsInsideSquare(self, points: List[List[int]], s: str) -> int:
        d = defaultdict(list)
        for i, (x, y) in enumerate(points):
            m = max(abs(x), abs(y))
```

```
            d[m].append(s[i])
        ans = 0
        cnt = Counter()
        for k in sorted(d.keys()):
            for c in d[k]:
                if c not in cnt:
                    cnt[c] += 1
                else:
                    return ans
            ans += len(d[k])
        return ans
```

3. 100289. 分割字符频率相等的最少子字符串

给你一个字符串 `s` ，你需要将它分割成一个或者更多的 **平衡** 子字符串。比方说，`s == "ababcc"` 那么 `("abab", "c", "c")` ，`("ab", "abc", "c")` 和 `("ababcc")` 都是合法分割，但是 `("a", **"bab"**, "cc")` ，`(**"aba"**, "bc", "c")` 和 `("ab", **"abcc"**)` 不是，不平衡的子字符串用粗体表示。

请你返回 `s` **最少** 能分割成多少个平衡子字符串。

注意：一个 **平衡** 字符串指的是字符串中所有字符出现的次数都相同。

示例 1：

输入：s = "fabccddg"

输出：3

解释：

我们可以将 `s` 分割成 3 个子字符串：`("fab, "ccdd", "g")` 或者 `("fabc", "cd", "dg")` 。

示例 2：

输入：s = "ababaccddb"

输出：2

解释：

我们可以将 `s` 分割成 2 个子字符串：`("abab", "abaccddb")` 。

提示：

- `1 <= s.length <= 1000`
- `s` 只包含小写英文字母。

思路

划分型 DP 固定套路

注意本题答案是存在的，因为单个字母是平衡的，我们一定可以划分成 n 个子串。

方法一：记忆化搜索

定义 $dfs(i)$ 表示划分前缀 $s[0]$ 到 $s[i]$ 的最小划分个数，则有

$$dfs(i) = \min_{j=0}^{i} dfs(j-1) + 1$$

其中 $s[j]$ 到 $s[i]$ 是平衡子串。

我们可以在倒序枚举 j 的同时，用一个哈希表（或者数组）统计每个字符的出现次数。如果子串中每个字母的出现次数都相等，那么子串是平衡的。

递归边界：$dfs(-1) = 0$。

递归入口：$dfs(n-1)$，即答案。

考虑到整个递归过程中有大量重复递归调用（递归入参相同）。由于递归函数没有副作用，同样的入参无论计算多少次，算出来的结果都是一样的，因此可以用记忆化搜索来优化：

- 如果一个状态（递归入参）是第一次遇到，那么可以在返回前，把状态及其结果记到一个 $memo$ 数组中。
- 如果一个状态不是第一次遇到 $memo$ 中保存的结果不等于 $memo$ 的初始值），那么可以直接返回 $memo$ 中保存的结果。

注意：$memo$ 数组的初始值一定不能等于要记忆化的值！例如初始值设置为 0，并且要记忆化的 $dfs(i)$ 也等于 0，那就没法判断 0 到底表示第一次遇到这个状态，还是表示之前遇到过了，从而导致记忆化失效。一般把初始值设置为 -1。本题由于 $dfs(i > 0)$，可以初始化成 0。

方法二：递推（1:1 翻译）

定义 $f[i+1]$ 表示划分前缀 $s[0]$ 到 $s[i]$ 的最小划分个数，则有

$$f(i+1) = \min_{j=0}^{i} f[j] + 1$$

其中 $s[j]$ 到 $s[i]$ 是平衡子串。

初始值 $f[0] = 0$，翻译自递归边界：$dfs(-1) = 0$。

答案为 $f[n]$，翻译自递归入口：$dfs(n-1)$。

1. Java

1. 记忆化搜索

```java
class Solution {
    Map<Integer, Integer> memo = new HashMap<>();

    public int minimumSubstringsInPartition(String s) {
        return dfs(s.length() - 1, s);
    }

    private int dfs(int i, String s) {
        if (i < 0)
            return 0;
        if (memo.containsKey(i))
            return memo.get(i);
        int res = Integer.MAX_VALUE;
        Map<Character, Integer> cnt = new HashMap<>();
        for (int j = i; j >= 0; j--) {
            cnt.put(s.charAt(j), cnt.getOrDefault(s.charAt(j), 0) + 1);
```

```
            if ((i - j + 1) % cnt.size() != 0)
                continue;
            int c0 = cnt.get(s.charAt(j));
            boolean allEqual = true;
            for (int count : cnt.values()) {
                if (count != c0) {
                    allEqual = false;
                    break;
                }
            }
            if (allEqual) {
                res = Math.min(res, dfs(j - 1, s) + 1);
            }
        }
        memo.put(i, res);
        return res;
    }
}
```

2. 递推

```
class Solution {
    public int minimumSubstringsInPartition(String s) {
        int n = s.length();
        int[] dp = new int[n + 1];
        for (int i = 1; i <= n; i++) {
            dp[i] = Integer.MAX_VALUE;
        }
        for (int i = 1; i <= n; i++) {
            Map<Character, Integer> cnt = new HashMap<>();
            int res = Integer.MAX_VALUE;
            for (int j = i; j >= 1; j--) {
                cnt.put(s.charAt(j - 1), cnt.getOrDefault(s.charAt(j - 1), 0) + 1);
                if ((i - j + 1) % cnt.size() != 0) {
                    continue;
                }
                int c0 = cnt.get(s.charAt(j - 1));
                boolean allEqual = true;
                for (int count : cnt.values()) {
                    if (count != c0) {
                        allEqual = false;
                        break;
                    }
                }
                if (allEqual) {
                    res = Math.min(res, dp[j - 1] + 1);
                }
            }
            dp[i] = res;
        }
        return dp[n];
    }
}
```

2. Python

1. 记忆化搜索

```python
class Solution:
    def minimumSubstringsInPartition(self, s: str) -> int:
        @cache
        def dfs(i: int) -> int:
            if i < 0:
                return 0
            res = inf
            cnt = Counter()
            for j in range(i, -1, -1):
                cnt[s[j]] += 1
                if (i - j + 1) % len(cnt):
                    continue
                c0 = cnt[s[j]]
                if all(c == c0 for c in cnt.values()):
                    res = min(res, dfs(j - 1) + 1)
            return res
        return dfs(len(s) - 1)
```

2.递推

```python
class Solution:
    def minimumSubstringsInPartition(self, s: str) -> int:
        n = len(s)
        f = [0] + [inf] * n
        for i in range(n):
            cnt = Counter()
            for j in range(i, -1, -1):
                cnt[s[j]] += 1
                if (i - j + 1) % len(cnt):
                    continue
                c0 = cnt[s[j]]
                if all(c == c0 for c in cnt.values()):
                    f[i + 1] = min(f[i + 1], f[j] + 1)
        return f[n]
```