LeetCode 1399. 统计最大组的数目

1399. 统计最大组的数目

难度 简单 5 收藏 分享 切换为英文 关注 反馈

给你一个整数 `n` 。请你先求出从 `1` 到 `n` 的每个整数 10 进制表示下的数位和（每一位上的数字相加），然后把数位和相等的数字放到同一个组中。

请你统计每个组中的数字数目，并返回数字数目并列最多的组有多少个。

示例 1：

输入： n = 13
输出： 4
解释： 总共有 9 个组，将 1 到 13 按数位求和后这些组分别是：
[1,10]，[2,11]，[3,12]，[4,13]，[5]，[6]，[7]，[8]，[9]。总共有 4 个组拥有的数字并列最多。

示例 2：

输入： n = 2
输出： 2
解释： 总共有 2 个大小为 1 的组 [1]，[2]。

示例 3：

输入： n = 15
输出： 6

示例 4：

输入： n = 24
输出： 5

```cpp
class Solution {
public:
    int countLargestGroup(int n) {
        vector<int> sum(40, 0);
        for (int i = 1; i <= n; i++) {
            int x = i, y = 0;
            while (x) {
                y += x % 10;
                x /= 10;
            }
            sum[y]++;
        }

        int ma = 0;
        for (int i = 1; i <= 36; i++)
```

```
            ma = max(ma, sum[i]);

        int ans = 0;
        for (int i = 1; i <= 36; i++)
            if (ma == sum[i])
                ans++;

        return ans;
    }
};
```

LeetCode 1400. 构造 K 个回文字符串

1400. 构造 K 个回文字符串

难度 中等 7 收藏 分享 切换为英文 关注 反馈

给你一个字符串 `s` 和一个整数 `k` 。请你用 `s` 字符串中 **所有字符** 构造 `k` 个非空 **回文串** 。

如果你可以用 `s` 中所有字符构造 `k` 个回文字符串，那么请你返回 **True** ，否则返回 **False** 。

示例 1:

输入： `s = "annabelle", k = 2`
输出： `true`
解释： 可以用 `s` 中所有字符构造 `2` 个回文字符串。
一些可行的构造方案包括：`"anna" + "elble"`，`"anbna" + "elle"`，`"anellena" + "b"`

示例 2:

输入： `s = "leetcode", k = 3`
输出： `false`
解释： 无法用 `s` 中所有字符构造 `3` 个回文串。

示例 3:

输入： `s = "true", k = 4`
输出： `true`
解释： 唯一可行的方案是让 `s` 中每个字符单独构成一个字符串。

示例 4:

输入： `s = "yzyzyzyzyzyzyzy", k = 2`
输出： `true`
解释： 你只需要将所有的 `z` 放在一个字符串中，所有的 `y` 放在另一个字符串中。那么两个字符串都是回文串。

```
class Solution {
public:
    bool canConstruct(string s, int k) {
        if(k > s.size()) return false;

        vector<int> cnt(26,0);
```

```
        for(auto c : s)
            cnt[c - 'a'] ++;

        int odd = 0;
        for(int i = 0;i < 26;i ++)
            if(cnt[i] & 1)
                odd ++;
        return odd <= k;
    }
};
```

1401. 圆和矩形是否有重叠

1401. 圆和矩形是否有重叠

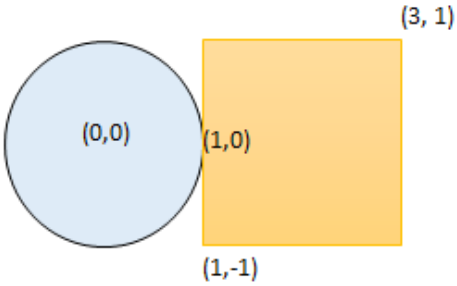
难度 中等 10 收藏 分享 切换为英文 关注 反馈

给你一个以（`radius`，`x_center`，`y_center`）表示的圆和一个与坐标轴平行的矩形（`x1`，`y1`，`x2`，`y2`），其中（`x1`，`y1`）是矩形左下角的坐标，（`x2`，`y2`）是右上角的坐标。

如果圆和矩形有重叠的部分，请你返回 True ，否则返回 False 。

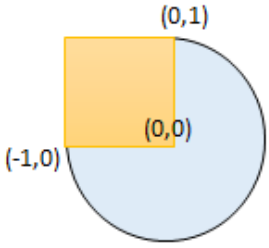
换句话说，请你检测是否 **存在** 点 (xi, yi)，它既在圆上也在矩形上（两者都包括点落在边界上的情况）。

示例 1：




```
输入：radius = 1, x_center = 0, y_center = 0, x1 = 1, y1 = -1, x2 = 3, y2 = 1
输出：true
解释：圆和矩形有公共点 (1,0)
```

示例 2：




```
输入：radius = 1, x_center = 0, y_center = 0, x1 = -1, y1 = 0, x2 = 0, y2 = 1
输出：true
```

```
class Solution {
public:
    double sqr(int x) {
```

```
        return x * x;
    }

    bool check(int r, int x0, int y0, int x, int y) {
        return sqr(x0 - x) + sqr(y0 - y) <= sqr(r);
    }

    bool checkOverlap(int radius, int x_center, int y_center, int x1, int y1,
int x2, int y2) {
        if (check(radius, x_center, y_center, x1, y1)
         || check(radius, x_center, y_center, x1, y2)
         || check(radius, x_center, y_center, x2, y1)
         || check(radius, x_center, y_center, x2, y2))
            return true;

        if (x1 <= x_center && x_center <= x2) {
            if (y1 <= y_center && y_center <= y2)
                return true;

            if (min(abs(y1 - y_center), abs(y2 - y_center)) <= radius)
                return true;
        }

        if (y1 <= y_center && y_center <= y2) {
            if (min(abs(x1 - x_center), abs(x2 - x_center)) <= radius)
                return true;
        }

        return false;
    }
};
```

LeetCode 1402. 做菜顺序

1402. 做菜顺序

难度 困难 22 收藏 分享 切换为英文 关注 反馈

一个厨师收集了他 `n` 道菜的满意程度 `satisfaction` ，这个厨师做出每道菜的时间都是 1 单位时间。

一道菜的 「喜爱时间」系数定义为烹饪这道菜以及之前每道菜所花费的时间乘以这道菜的满意程度，也就是 `time[i]*satisfaction[i]` 。

请你返回做完所有菜 「喜爱时间」总和的最大值为多少。

你可以按 **任意** 顺序安排做菜的顺序，你也可以选择放弃做某些菜来获得更大的总和。

示例 1:

```
输入：satisfaction = [-1,-8,0,5,-9]
输出：14
解释：去掉第二道和最后一道菜，最大的喜爱时间系数和为 (-1*1 + 0*2 + 5*3 = 14) 。每道菜都需要花费 1 单位时间完成。
```

示例 2:

```
输入：satisfaction = [4,3,2]
输出：20
解释：按照原来顺序相反的时间做菜 (2*1 + 3*2 + 4*3 = 20)
```

示例 3:

```
输入：satisfaction = [-1,-4,-5]
输出：0
解释：大家都不喜欢这些菜，所以不做任何菜可以获得最大的喜爱时间系数。
```

示例 4:

```
输入：satisfaction = [-2,5,-1,0,3,-3]
输出：35
```

```cpp
class Solution {
public:
    int maxSatisfaction(vector<int>& satisfaction) {
        sort(satisfaction.begin(), satisfaction.end());
        int n = satisfaction.size();

        vector<vector<int>> f(n + 1, vector<int>(n + 1, INT_MIN));
        f[0][0] = 0;

        for (int i = 1; i <= n; i++) {
            f[i][0] = f[i - 1][0];
            for (int j = 1; j <= i; j++)
                f[i][j] = max(f[i - 1][j], f[i - 1][j - 1] + j * satisfaction[i - 1]);
        }

        int ans = INT_MIN;
        for (int j = 0; j <= n; j++)
```

```
            ans = max(ans, f[n][j]);

        return ans;
    }
};
```