题目地址(455. 分发饼干)

https://leetcode-cn.com/problems/assign-cookies/

入选理由

1. 贪心入门题目

标签

- 贪心

难度

- 简单

题目描述

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。对每个孩子 i ，都有一个胃口值 gi ，这是能让孩子们满足胃口

注意：

你可以假设胃口值为正。
一个小朋友最多只能拥有一块饼干。

示例 1:

输入: [1,2,3], [1,1]

输出: 1

解释:

你有三个孩子和两块小饼干，3个孩子的胃口值分别是：1,2,3。
虽然你有两块小饼干，由于他们的尺寸都是1，你只能让胃口值是1的孩子满足。
所以你应该输出1。

示例 2:

输入: [1,2], [1,2,3]

输出: 2

解释:

```
你有两个孩子和三块小饼干，2个孩子的胃口值分别是1,2。
你拥有的饼干数量和尺寸都足以让所有孩子满足。
所以你应该输出2.
```

前置知识

- 贪心算法
- 双指针

公司

- 阿里
- 腾讯
- 字节

思路

本题可用贪心求解。给一个孩子的饼干应当尽量小并且能满足孩子，大的留来满足胃口大的孩子。因为胃口小的孩子最容易得到满足，所以优先满足胃口小的孩子需求。按照从小到大的顺序使用饼干尝试是否可满足某个孩子。

算法：

- 将需求因子 g 和 s 分别从小到大进行排序
- 使用贪心思想，配合两个指针，每个饼干只尝试一次，成功则换下一个孩子来尝试，不成功则换下一个饼干 🍪 来尝试。

关键点

- 先排序再贪心

代码

语言支持：JS，Java,CPP,Python

JS Code:

```
/**
 * @param {number[]} g
 * @param {number[]} s
```

```
 * @return {number}
 */
const findContentChildren = function (g, s) {
  g = g.sort((a, b) => a - b);
  s = s.sort((a, b) => a - b);
  let gi = 0; // 胃口值
  let sj = 0; // 饼干尺寸
  let res = 0;
  while (gi < g.length && sj < s.length) {
    // 当饼干 sj >= 胃口 gi 时，饼干满足胃口，更新满足的孩子数并移动指针
    if (s[sj] >= g[gi]) {
      gi++;
      sj++;
      res++;
    } else {
      // 当饼干 sj < 胃口 gi 时，饼干不能满足胃口，需要换大的
      sj++;
    }
  }
  return res;
};
```

Java Code:

```
class Solution {
    public int findContentChildren(int[] g, int[] s) {
        int res = 0;
        Arrays.sort(g);
        Arrays.sort(s);
        int i = 0, j = 0;
        while (i < g.length && j < s.length) {
            if (g[i] <= s[j]) {
                res++;
                i++;
                j++;
            } else if (g[i] > s[j]) {
                j++;
            }
        }
        return res;

    }
}
```

CPP Code：

```
class Solution {
public:
```

```
    int findContentChildren(vector<int>& g, vector<int>& s) {

        sort(g.begin(), g.end());
        sort(s.begin(), s.end());

        int gleft =0;
        int sleft =0;

        int count =0;
        while(gleft< g.size() && sleft< s.size())
        {

            if(g[gleft]<=s[sleft])
            {
                count++;
                gleft++;
                sleft++;
            }
            else
                sleft++;
        }

        return count;

    }
};
```

Python Code:

```
class Solution:
    def findContentChildren(self, g: List[int], s: List[int]) -> int:
        s.sort(reverse=True)
        g.sort(reverse=True)
        gi, si = 0, 0
        count = 0
        while gi < len(g) and si < len(s):
            if s[si] >= g[gi]:
                count += 1
                si += 1
            gi += 1
        return count
```

复杂度分析

令 n 为数组长度

- 时间复杂度：由于使用了排序，因此时间复杂度大约为 O(nlogn)
- 空间复杂度：取决于具体的排序方法，大概是 O(1) 到 O(logn)

上一页 下一页