算法-贪心

> 做题时不要太纠结于选择哪种方法，贪心算法没有特别明显的特征！
>
> 有时贪心算法和暴力算法类似

贪心算法概述

贪心的本质是 **通过选择每一阶段的局部最优，从而达到全局最优**

例如，有一堆钞票，你可以拿走十张，如果想达到最大的金额，你要怎么拿？指定每次拿最大的，最终结果就是拿走最大数额的钱。

每次拿最大的就是局部最优，最后拿走最大数额的钱就是推出全局最优。

再举一个例子如果是 有一堆盒子，你有一个背包体积为n，如何把背包尽可能装满，如果还每次选最大的盒子，就不行了。这时候就需要动态规划。

贪心一般解题步骤

1. 将问题分解为若干个子问题
2. 找出适合的贪心策略
3. 求解每一个子问题的最优解
4. 将局部最优解堆叠成全局最优解

典例-最大子数组和

53. 最大子数组和

难度 简单 4307

给你一个整数数组 `nums` ，请你找出一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

子数组 是数组中的一个连续部分。

示例 1：

```
输入：nums = [-2,1,-3,4,-1,2,1,-5,4]
输出：6
解释：连续子数组 [4,-1,2,1] 的和最大，为 6 。
```

示例 2：

```
输入：nums = [1]
输出：1
```

示例 3：

```
输入：nums = [5,4,-1,7,8]
输出：23
```

提示：

- `1 <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`

讲讲贪心：

如果 -2 1 在一起，计算起点的时候，一定是从1开始计算，因为负数只会拉低总和，这就是贪心贪的地方！

局部最优：当前“连续和”为负数的时候立刻放弃，从下一个元素重新计算“连续和”，因为负数加上下一个元素 “连续和”只会越来越小。

全局最优：选取最大“连续和”

局部最优的情况下，并记录最大的“连续和”，可以推出全局最优。

从代码角度上来讲：遍历nums，从头开始用count累积，如果count一旦加上nums[i]变为负数，那么就应该从nums[i+1]开始从0累积count了，因为已经变为负数的count，只会拖累总和。

```
class Solution {
public:
    int maxSubArray(vector<int>& nums)
    {
        //贪心
```

```
        //局部最优：当前“连续和”为负数的时候立刻放弃，从下一个元素重新计算“连续和”，因为负数加上下一个元素 “连续和”只会越来越小。
        //全局最优：选取最大连续和
        //局部最优的情况下，并记录最大的“连续和”，可以推出全局最优
        int result=INT32_MIN;
        int count=0;
        for(int i=0;i<nums.size();i++)
        {
            count+=nums[i];
            if(count > result)
            {
                //取区间累计的最大值（相当于不断确定最大子序终止位置）
                result = count;
            }
            if(count <= 0) count = 0;    //如果连续和变为0，那么重置count  相当于重置最大子序起始位置
        }
        return result;
    }
};
```

实验3-分发糖果（重点）

135. 分发糖果

难度 困难 756

`n` 个孩子站成一排。给你一个整数数组 `ratings` 表示每个孩子的评分。

你需要按照以下要求，给这些孩子分发糖果：

- 每个孩子至少分配到 `1` 个糖果。
- 相邻两个孩子评分更高的孩子会获得更多的糖果。

请你给每个孩子分发糖果，计算并返回需要准备的 **最少糖果数目** 。

示例 1：

输入： ratings = [1,0,2]
输出： 5
解释： 你可以分别给第一个、第二个、第三个孩子分发 2、1、2 颗糖果。

示例 2：

输入： ratings = [1,2,2]
输出： 4
解释： 你可以分别给第一个、第二个、第三个孩子分发 1、2、1 颗糖果。
第三个孩子只得到 1 颗糖果，这满足题面中的两个条件。

提示：

- `n == ratings.length`
- `1 <= n <= 2 * 10^4`
- `0 <= ratings[i] <= 2 * 10^4`

这道题目一定是要确定一边之后，再确定另一边，例如比较每一个孩子的左边，然后再比较右边，**如果两边一起考虑一定会顾此失彼。**

先确定右边评分大于左边的情况（也就是从前向后遍历）

此时局部最优：只要右边评分比左边大，右边的孩子就多一个糖果，全局最优：相邻的孩子中，评分高的右孩子获得比左边孩子更多的糖果

局部最优可以推出全局最优。

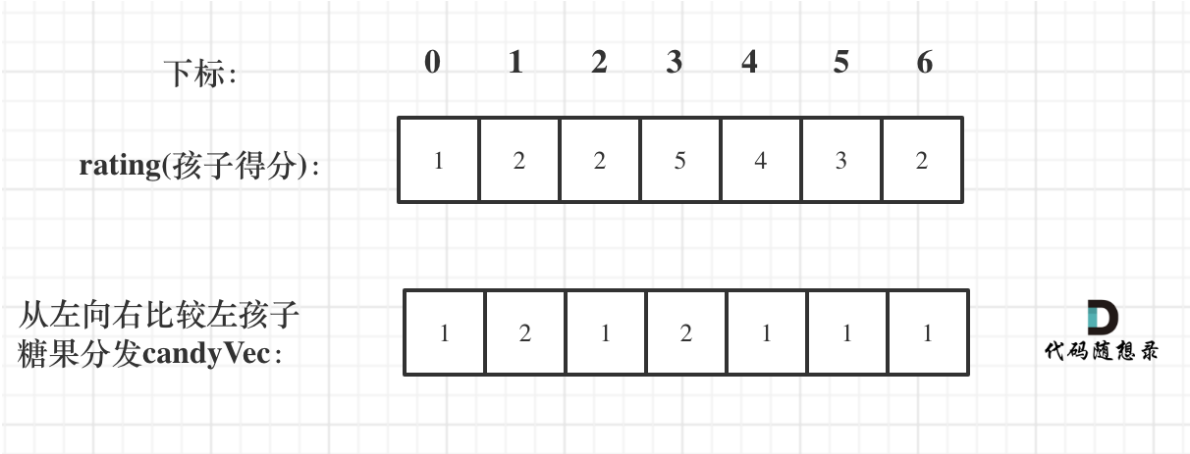
如果ratings[i] > ratings[i - 1] 那么[i]的糖 一定要比[i - 1]的糖多一个，

所以贪心：candyVec[i] = candyVec[i - 1] + 1

代码如下：

```
// 从前向后
for (int i = 1; i < ratings.size(); i++) {
    if (ratings[i] > ratings[i - 1]) candyVec[i] = candyVec[i - 1] + 1;
}
```

如图：



再确定左孩子大于右孩子的情况（从后向前遍历）

遍历顺序这里有同学可能会有疑问，为什么不能从前向后遍历呢？

因为如果从前向后遍历，根据 ratings[i + 1] 来确定 ratings[i] 对应的糖果，那么每次都不能利用上前一次的比较结果了。

所以确定左孩子大于右孩子的情况一定要从后向前遍历！

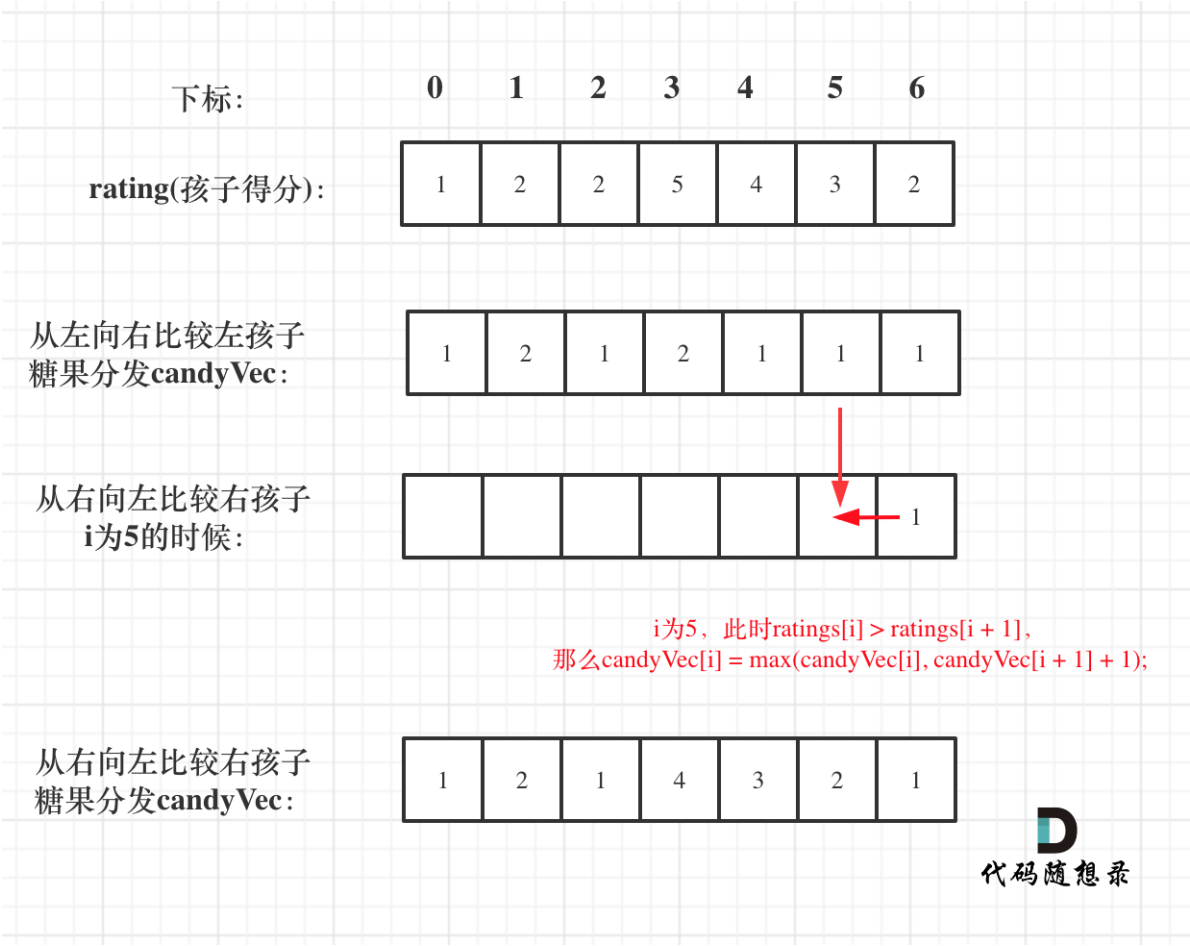
如果 ratings[i] > ratings[i + 1]，此时candyVec[i]（第i个小孩的糖果数量）就有两个选择了，一个是candyVec[i + 1] + 1（从右边这个加1得到的糖果数量），一个是candyVec[i]（之前比较右孩子大于左孩子得到的糖果数量）。

那么又要贪心了，局部最优：取candyVec[i + 1] + 1 和 candyVec[i] 最大的糖果数量，保证第i个小孩的糖果数量即大于左边的也大于右边的。全局最优：相邻的孩子中，评分高的孩子获得更多的糖果。

局部最优可以推出全局最优。

所以就取candyVec[i + 1] + 1 和 candyVec[i] 最大的糖果数量，**candyVec[i]只有取最大的才能既保持对左边candyVec[i - 1]的糖果多，也比右边candyVec[i + 1]的糖果多。**

如图：




```python
class Solution:
    def candy(self, ratings: List[int]) -> int:
        #贪心算法 分两个路线进行遍历

        candy = [1] * len(ratings)  #初始化candy数组，每个孩子最少分得一块糖果
        for i in range(1,len(ratings)): #先考虑右孩子比左孩子评分高的情况 从前向后遍历
            if ratings[i] > ratings[i-1]:
                candy[i] = candy[i-1] + 1
        for j in range(len(ratings) - 2,-1,-1): #先考虑左孩子比右孩子评分高的情况 从后向前遍历
            if ratings[j] > ratings[j+1]:
                candy[j] = max(candy[j],candy[j+1] + 1)
        return sum(candy)
```

实验3-最大数问题

179. 最大数

难度 中等 923

给定一组非负整数 `nums`，重新排列每个数的顺序（每个数不可拆分）使之组成一个最大的整数。

注意：输出结果可能非常大，所以你需要返回一个字符串而不是整数。

示例 1：

```
输入：nums = [10,2]
输出："210"
```

示例 2：

```
输入：nums = [3,30,34,5,9]
输出："9534330"
```

提示：

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 10^9`

通过次数 151,763 | 提交次数 369,077

请问您在哪类招聘中遇到此题?

社招 校招 实习 未遇到

贪心-局部最优推出全局最优

两两相比，大则交换

其实就像选择排序，但是这也属于贪心算法，体现了贪心的思想

```python
class Solution:
    def largestNumber(self, nums: List[int]) -> str:
        # 选择排序法解决
        length = len(nums)  #遍历范围
        nums = list(map(str,nums))  #利用map()函数转为str型
        for i in range(length):
            for j in range(i+1,length):
                if nums[i] + nums[j] < nums[j] + nums[i]:   #交换条件
                    nums[i],nums[j] = nums[j],nums[i]
        return str(int(''.join(nums)))  #.join()转str
```