# 题目

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。

相邻两个孩子中，评分更高的那个会获得更多的糖果。

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 \* 104

0 <= ratings[i] <= 2 \* 104

# 分析

## 方法一：两次遍历

要解决“给孩子分发糖果”的问题，核心思路是两次遍历数组：先保证“左到右”相邻孩子的评分规则（评分高的糖果更多），再修正“右到左”的规则，最终得到满足所有条件的最少糖果分配方案。

解题思路

1、核心约束拆解：

- 每个孩子至少1颗糖果（基础条件）。

- 相邻孩子中，评分高的糖果更多（双向约束：左→右和右→左均需满足）。

2、两次遍历的必要性：

- 第一次遍历（左→右）：仅保证“左边孩子评分 < 右边孩子评分时，右边糖果更多”。

初始化糖果数组candies为全1（满足基础条件），从左到右遍历：若ratings[i] > ratings[i-1]，则candies[i] = candies[i-1] + 1（确保右边比左边多）。

此时可能存在“右边孩子评分 < 左边孩子评分，但左边糖果未比右边多”的问题（如示例2：[1,2,2]左→右遍历后为[1,2,1]，已满足，但需验证更复杂场景）。

- 第二次遍历（右→左）：修正“右边孩子评分 < 左边孩子评分时，左边糖果更多”的情况。

从右到左遍历：若ratings[i] > ratings[i+1]，则candies[i] = max(candies[i], candies[i+1] + 1)（取当前值与“右边+1”的最大值，避免覆盖左→右遍历的有效结果）。

3、求和得到结果：两次遍历后，candies数组满足所有约束，累加其元素即为最少糖果总数。

代码：

class Solution {

public:

int candy(vector<int>& ratings) {

int n = ratings.size();

if (n == 0) return 0; // 边界：无孩子，返回0（题目中n≥1，可省略）

// 初始化糖果数组：每个孩子至少1颗糖果

vector<int> candies(n, 1);

// 第一次遍历：左→右，保证右边评分高的糖果更多

for (int i = 1; i < n; ++i) {

if (ratings[i] > ratings[i-1]) {

candies[i] = candies[i-1] + 1;

}

}

// 第二次遍历：右→左，保证左边评分高的糖果更多（修正左→右的不足）

for (int i = n - 2; i >= 0; --i) {

if (ratings[i] > ratings[i+1]) {

// 取当前值与“右边+1”的最大值，避免覆盖左→右的有效结果

candies[i] = max(candies[i], candies[i+1] + 1);

}

}

// 累加所有糖果，得到最少总数

int total = 0;

for (int num : candies) {

total += num;

}

return total;

}

};

代码解释

1、初始化：candies数组初始化为全1，直接满足“每个孩子至少1颗糖果”的基础条件。

2、左→右遍历：

- 从索引1开始（对比当前与前一个孩子），若当前孩子评分更高，则糖果数为前一个孩子+1，确保“左低右高”时右边糖果更多。

- 示例1（[1,0,2]）左→右遍历后：candies = [1,1,2]（第三个孩子评分>第二个，糖果+1）。

3、右→左遍历：

- 从索引n-2开始（对比当前与后一个孩子），若当前孩子评分更高，需确保糖果数>后一个孩子。此时用max函数保留左→右遍历的结果（如示例1中第一个孩子评分>第二个，candies[0] = max(1, 1+1) = 2）。

- 示例1右→左遍历后：candies = [2,1,2]，累加得5，符合预期。

4、求和：遍历candies数组累加，得到最少糖果总数。

复杂度分析

- 时间复杂度：O(n)，其中n是孩子数量（ ratings 数组长度）。两次遍历数组（各O(n)）+ 一次求和（O(n)），总时间为O(n)。

- 空间复杂度：O(n)，额外使用大小为n的candies数组存储每个孩子的糖果数（可优化为O(1)，但O(n)更直观，且n≤2×10⁴时空间开销可接受）。