

# 问题简化

想象你有一个装满数字的盒子（数组 nums），比如 [2, 3, 1, 2, 4, 3]，还有一个目标和 target = 7。我们需要从这个盒子里挑出一段连续的数字（子数组），让这些数字加起来 ≥ 7，并且这段数字的个数（长度）是最小的。如果怎么挑都达不到 7，就返回 0。

这个方法可以应用到**任何数组**，只要满足题目给的约束条件：

* 数组长度在 1 到 10^5 之间。
* 每个数字在 1 到 10^4 之间。
* target 在 1 到 10^9 之间。

**直观方法：滑动窗口**

想象你手里有两个手指，一个是“左手指”（left），一个是“右手指”（right），它们在数组上滑动，夹住一部分数字。我们会：

1. 用“右手指”慢慢往右移动，累加数字，直到夹住的数字和 ≥ target。
2. 然后用“左手指”往右移动，减少数字，尽量让夹住的数字和仍然 ≥ target，但长度变短。
3. 记录每次夹住的最短长度。
4. 如果遍历完数组还没找到，就返回 0。

**举一个简单例子**

假设数组是 [1, 2, 3, 4]，target = 6。我们一步步试试：

**初始状态**

* left = 0, right = 0, window\_sum = 0, min\_length = 很大（比如 1000000）
* 夹住的数字：[]（空），和 = 0 (< 6)。

**步骤 1：移动右手指**

* right = 0: 夹住 [1]，和 = 1 (< 6)。
* right = 1: 夹住 [1, 2]，和 = 3 (< 6)。
* right = 2: 夹住 [1, 2, 3]，和 = 6 (>= 6)。
  + 现在和够了！长度 = 3，记录 min\_length = 3。
  + 试着移动左手指：left = 1，夹住 [2, 3]，和 = 5 (< 6)，不够了，停下。

**步骤 2：继续移动右手指**

* right = 3: 夹住 [1, 2, 3, 4]，和 = 10 (>= 6)。
  + 长度 = 4，min\_length = min(3, 4) = 3。
  + 移动左手指：left = 2，夹住 [3, 4]，和 = 7 (>= 6)。
    - 长度 = 2，min\_length = min(3, 2) = 2。
  + 再移动左手指：left = 3，夹住 [4]，和 = 4 (< 6)，不够了，停下。

**结束**

* 遍历完数组，min\_length = 2，说明最短子数组是 [3, 4]，和 = 7。
* 返回 2。

**再试一个随机数组**

假设随机数组是 [5, 1, 3, 2, 7]，target = 10。

* right = 0: [5]，和 = 5 (< 10)。
* right = 1: [5, 1]，和 = 6 (< 10)。
* right = 2: [5, 1, 3]，和 = 9 (< 10)。
* right = 3: [5, 1, 3, 2]，和 = 11 (>= 10)。
  + 长度 = 4，min\_length = 4。
  + 移动左手指：left = 1，[1, 3, 2]，和 = 6 (< 10)。
  + 继续移动右手指。
* right = 4: [5, 1, 3, 2, 7]，和 = 18 (>= 10)。
  + 长度 = 5，min\_length = min(4, 5) = 4。
  + 移动左手指：left = 2，[3, 2, 7]，和 = 12 (>= 10)。
    - 长度 = 3，min\_length = min(4, 3) = 3。
  + 再移动：left = 3，[2, 7]，和 = 9 (< 10)。
* 结束，min\_length = 3，子数组 [3, 2, 7] 和 = 12。

返回 3。

# 代码

#include <vector>

#include <climits>

#include <iostream>

using namespace std;

pair<int, vector<int>> minSubArrayLen(int target, vector<int>& nums) {

int minLength = INT\_MAX; // 最小长度

int windowSum = 0; // 窗口和

int left = 0; // 左指针

int startIdx = 0; // 最短子数组的起始索引

int n = nums.size();

// 遍历右指针

for (int right = 0; right < n; right++) {

windowSum += nums[right]; // 加入当前元素

// 当窗口和 >= target 时，尝试缩小窗口

while (windowSum >= target) {

int currentLength = right - left + 1;

// 更新最小长度和对应的起始索引

if (currentLength < minLength) {

minLength = currentLength;

startIdx = left; // 记录起始位置

}

// 缩小窗口

windowSum -= nums[left];

left++;

}

}

// 如果没有找到符合条件的子数组

if (minLength == INT\_MAX) {

return {0, {}}; // 返回长度 0 和空向量

}

// 构造最短子数组

vector<int> minSubarray(nums.begin() + startIdx, nums.begin() + startIdx + minLength);

return {minLength, minSubarray}; // 返回长度和子数组

}

// 测试代码

int main() {

// 测试用例 1

vector<int> nums1 = {2, 3, 1, 2, 4, 3};

int target1 = 7;

auto result1 = minSubArrayLen(target1, nums1);

cout << "Test 1 - Length: " << result1.first << ", Subarray: ";

for (int num : result1.second) {

cout << num << " ";

}

cout << endl; // 预期: Length: 2, Subarray: 4 3

// 测试用例 2

vector<int> nums2 = {1, 4, 4};

int target2 = 4;

auto result2 = minSubArrayLen(target2, nums2);

cout << "Test 2 - Length: " << result2.first << ", Subarray: ";

for (int num : result2.second) {

cout << num << " ";

}

cout << endl; // 预期: Length: 1, Subarray: 4

// 测试用例 3

vector<int> nums3 = {1, 1, 1, 1, 1, 1};

int target3 = 11;

auto result3 = minSubArrayLen(target3, nums3);

cout << "Test 3 - Length: " << result3.first << ", Subarray: ";

for (int num : result3.second) {

cout << num << " ";

}

cout << endl; // 预期: Length: 0, Subarray: (空)

return 0;

}