盛最多水的容器

Container With Most Water

Problem ID: 3123 | Source: ZJUT OJ

题目描述

给定一个长度为 n 的整数数组 `height`。有 n 条垂线，第 i 条线的两个端点是 $(i, 0)$ 和 $(i, \text{height}[i])$。

找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

目标：返回容器可以储存的最大水量。

$$\text{Area} = (j - i) \times \min(\text{height}[i], \text{height}[j])$$

一、暴力枚举法 (Brute Force)

算法思路

直接使用两层循环枚举所有可能的左边界 i 和右边界 j。对于每一对 (i, j)，计算其面积并更新最大值。虽然逻辑简单，但计算量极大。

```
// 核心逻辑片段
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        // 宽度 * 高度瓶颈
        ans = max(ans, (j - i) * min(a[i], a[j]));
    }
}
```

二、排序贪心法 (Sorting Strategy)

算法思路

将柱子按**高度从大到小**排序。处理第 k 高的柱子时，它一定是当前构成的容器的“短板”。为了使面积最大，我们需要找到原数组中距离当前柱子**最远**的且高度大于等于它的柱子。通过维护已遍历柱子的最小和最大索引即可实现。

```cpp
// 记录原始下标并按高度降序排序
sort(p, p + n, [&](int x, int y) { return a[x] > a[y]; });

int mn = 1e9, mx = -1;
for (int i = 0; i < n; i++) {
    int idx = p[i]; // 获取原始位置
    if (mx != -1) {
        // 计算到最左端和最右端的最大跨度，因为这些柱子都比当前高
        int width = max(abs(idx - mn), abs(idx - mx));
        ans = max(ans, a[idx] * width);
    }
    mn = min(mn, idx); // 更新已见过的最左边界
    mx = max(mx, idx); // 更新已见过的最右边界
}
```

三、双指针法 (Two Pointers - Optimal)

算法核心与证明

这是本题的最优解。定义双指针 $L = 0, R = n - 1$。每次移动两个指针中**较短**的那一个。

证明：假设 $h[L] < h[R]$，当前面积由 $h[L]$ 决定。如果保持 L 不动而左移 R，容器宽度变小，而高度受限于 $h[L]$ 绝不可能增加，因此面积必然减小。为了寻找更大的面积，必须舍弃当前的短板 L，尝试 $L + 1$。

表 1: 算法性能对比

方法	时间复杂度	空间复杂度	评价
暴力枚举	$O(n^2)$	$O(1)$	数据量大时超时 (TLE)
排序贪心	$O(n \log n)$	$O(n)$	较优，但在原本有序时非最优
双指针	$O(n)$	$O(1)$	**最优解，一次遍历**

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for(int i=0; i<n; i++) cin >> a[i];

    int l = 0, r = n - 1, mx = 0;

    while (l < r) {
        // 计算当前面积
        int h = min(a[l], a[r]);
        mx = max(mx, (r - l) * h);

        // 贪心策略：谁短移动谁
        if (a[l] < a[r]) l++;
        else r--;
    }

    cout << mx << endl;
    return 0;
}
```

四、结果验证与扩展阅读

评测结果

上述三种方法中，双指针法与排序贪心法均可通过测试。以下为最终提交通过的截图：

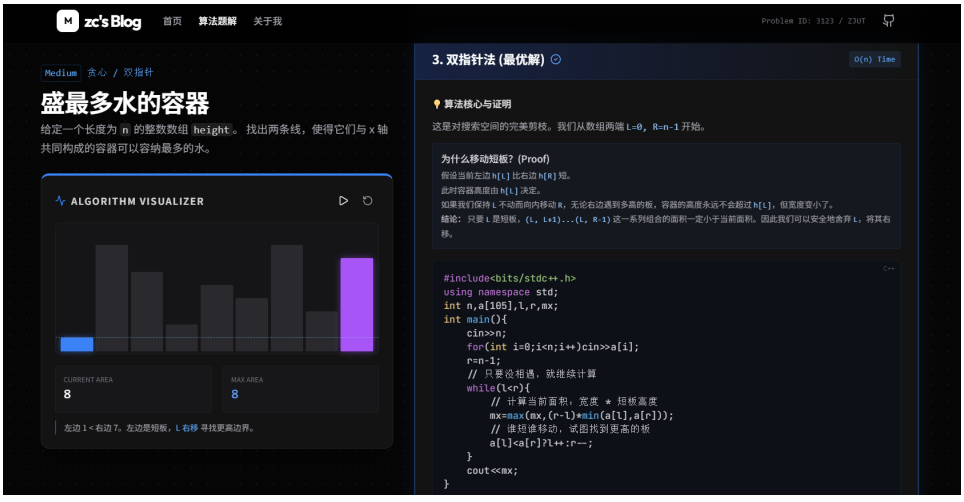
Language: C++

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,a[105],l,r,mx;
int main(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>a[i];
    r=n-1;
    // 只要没相遇，就继续计算
    while(l<r){
        // 计算当前面积：宽度 * 短板高度
        mx=max(mx,(r-l)*min(a[l],a[r]));
        // 谁短谁移动，试图找到更高的板
        a[l]<a[r]?l++:r--;
    }
    cout<<mx;
}
```

Status Accepted

算法可视化演示

为了更直观地理解“双指针”是如何移动并剪枝搜索空间的，我在博客中制作了详细的**动态演示 (Animation)**。



→ 点击此处跳转博客查看动画演示 ←

· 2025 年 12 月 10 日