

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。

```
1 // 核心逻辑片段
2 for (int i = 0; i < n; i++) {
3     for (int j = i + 1; j < n; j++) {
4         // 宽度 * 高度瓶颈
5         ans = max(ans, (j - i) * min(a[i], a[j]));
6     }
7 }
```

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。

```

1 // 记录原始下标并按高度降序排序
2 sort(p, p + n, [&](int x, int y) { return a[x] > a[y]; });
3
4 int mn = 1e9, mx = -1;
5 for (int i = 0; i < n; i++) {
6     int idx = p[i]; // 获取原始位置
7     if (mx != -1) {
8         // 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
9         int width = max(abs(idx - mn), abs(idx - mx));
10        ans = max(ans, a[idx] * width);
11    }
12    mn = min(mn, idx); // 更新已见过的最左边界
13    mx = max(mx, idx); // 更新已见过的最右边界
14}

```

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

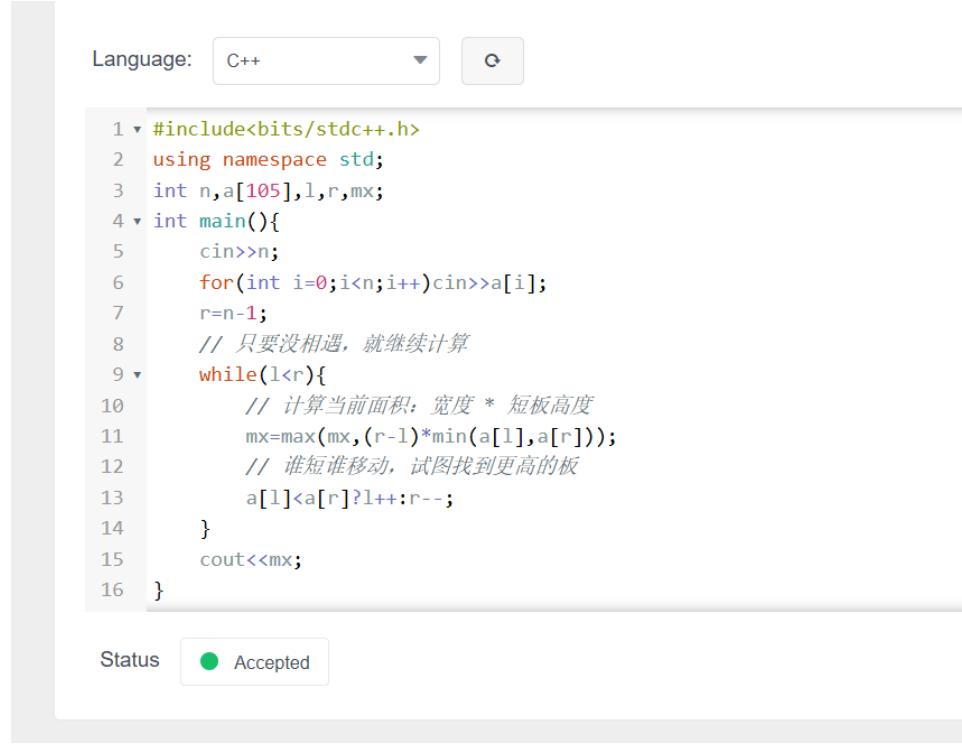
方法	时间复杂度	空间复杂度	评价
暴力枚举	$O(n^2)$	$O(1)$	数据量大时超时 (TLE)
排序贪心	$O(n \log n)$	$O(n)$	较优，但在原本有序时非最优
双指针	$O(n)$	$O(1)$	最优解，一次遍历

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int n;
6     cin >> n;
7     vector<int> a(n);
8     for(int i=0; i<n; i++) cin >> a[i];
9
10    int l = 0, r = n - 1, mx = 0;
11
12    while (l < r) {
13        // 计算当前面积
14        int h = min(a[l], a[r]);
15        mx = max(mx, (r - l) * h);
16
17        // 贪心策略：谁短移动谁
18        if (a[l] < a[r]) l++;
19        else r--;
20    }
21
22    cout << mx << endl;
23    return 0;
24 }
```

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++ ↻
1 #include<bits/stdc++.h>
2 using namespace std;
3 int n,a[105],l,r,mx;
4 int main(){
5     cin>>n;
6     for(int i=0;i<n;i++)cin>>a[i];
7     r=n-1;
8     // 只要没相遇，就继续计算
9     while(l<r){
10         // 计算当前面积：宽度 * 短板高度
11         mx=max(mx,(r-l)*min(a[l],a[r]));
12         // 谁短谁移动，试图找到更高的板
13         a[l]<a[r]?l++:r--;
14     }
15     cout<<mx;
16 }

```

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）。



3. 双指针法(最优解) O(n) Time.

算法核心与证明

这是对搜索空间的完美剪枝。我们从数组两端 $L=0, R=n-1$ 开始。

为什么移动短板？(Proof)

假设当前左边 L 比右边 R 矮。
此时容器高度由 $H[R]$ 决定。
如果我们将保持 L 不动而向右移动 R ，无论右边遇到多高的板，容器的高度永远不会超过 $H[L]$ ，但宽度变小了。
结论：只要 L 是短板， $(L, L+1), \dots, (L, R-1)$ 这一系列组合的面积一定小于当前面积。因此我们可以安全地舍弃 L ，将其右移。

```

#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;
}

```

→ 点击此处跳转博客查看动画演示 ←

· 2025 年 12 月 10 日