1st: Modeling, and Divide And Conquer

手写题

Longest Balanced Substring

（1）Modeling

该问题即要求找到满足下述条件的最长子串：每个出现在该子串中的字母，都要同时包含它的小写和大写形式。

（2）Algorithm description

采用分治法进行解题:

1. 对于给定字符串，遍历该串构建哈希集合，之后检查出现在字符串中的每一个字母，直到找到第一个不符合平衡条件的字母（若没有这样的字母，则说明原字符串即为最长平衡子串）
2. 则包含该字母的子串一定不是平衡子串，所以以该字母为分割点，将字符串分为左右两个子串
3. 递归地在这两个子串上继续寻找最长的平衡子串，直到子串长度小于2。
4. 返回所找到的最长平衡子串。

（3）Time complexity

对于长度为n的字符串s：
构建哈希集合并判断一个子串是否平衡的时间复杂度为$O(n)$，根据递归深度，最好情况时间复杂度为$O(nlogn)$，最坏情况时间复杂度为$O(n^2)$。

（4）Space complexity

由于字母数有限，构建哈希集合所需空间复杂度为常数级，根据递归深度，最好情况空间复杂度为$O(logn)$，最坏情况空间复杂度为$O(n)$。

Cutting Bamboo Poles

（1）Modeling

- 将每根竹竿看作一个数组元素，数组中的每个元素表示竹竿的长度。
- 我们需要找到一个长度L，使得能够从所有竹竿中砍出m根长度为L的竹竿，L越长越好。
- 这可以建模为一个优化问题：在最小长度和最大长度之间进行二分搜索，以找到能够砍出m根相同长度竹竿的最大长度L。

（2）Algorithm description

1. 初始化：设定一个搜索范围，最小值为0，最大值为所有竹竿中的最大长度的下取整。先判断这个长度能否满足题目要求，若能则结束，若不能则进行二分搜索。
2. 二分搜索：在这个范围内，通过下取整的方式计算选择中间值mid，计算能够砍出的竹竿数。如果可以砍出至少m根长度为mid的竹竿，设当前mid为最小值，继续在右半部分（较大值）搜索；否则设当前mid为最大值，继续在左半部分（较小值）搜索。
3. 判断条件：在每次二分搜索中，计算可以从每根竹竿中砍出的竹竿数，累加判断是否能够满足m根竹竿。
4. 终止条件：上述操作保证最小值都能满足题意，最大值都不能满足题意，当搜索范围收窄到无法继续，即区间的最小值与最大值仅相差1时，当前的最小值即为可以切出的最大长度。

（3）Time complexity

二分搜索时间复杂度为$O(log(maxLen))$，其中$maxLen$为最长竹竿的长度。对于每一个计算得到的中间值mid，需要遍历所有n根竹竿以判断是否能至少砍出m根该长度的竹竿来制作竹筏，时间复杂度为$O(n)$，所以总的时间复杂度为$O(nlog(maxLen))$

（4）Space complexity

只需要常数空间存储搜索范围的边界值和当前的最大长度，存储输入的每根竹竿的长度需要$O(n)$空间，所以总的空间复杂度为$O(n)$

Multiple Calculations

（1）Modeling

这道题目可以用分治递归思想去考虑，可以将每一个运算符作为表达式的左右两部分的分割部位，将问题拆分成相同的子问题，再去分别递归左右两部分的所有可能结果，最后所有结果通过运算符组合起来。

（2）Algorithm description

1. 初始化：给定一个表达式字符串，递归处理不同的运算符组合
2. 递归分解：遍历字符串中的每个运算符，将字符串分为运算符左侧和右侧两部分，分别递归两部分计算它们的结果
3. 递归终止：在递归中遇到当前字符串不包含运算符，说明是一个数字，直接返回为结果，作为递归的终止条件
4. 合并结果：当左右两部分的所有可能结果计算完成后，根据当前运算符对它们进行运算，将结果存入列表并返回。

（3）Time complexity

每个运算符都可能成为递归的分割点，表达式可以划分为左右两部分进行递归，生成的递归树近似于完全二叉树，最多会有$O(2^n)$ 个不同的计算结果，其中 n是运算符的数量。此外，每次组合左右两部分的结果的时间复杂度为$O(m^2)$，所以总的时间复杂度是：$O(2^n * m^2)$

（4)Space complexity

递归的最大深度取决于表达式中的运算符数量。假设表达式有 n个运算符，那么递归的最大深度为$O(n)$,每个子表达式的结果都会被储存。由于递归的分治方法，会生成接近$O(2^n)$个子表达式，所以总的空间复杂度为：$O(2^n)$

N-sum

(1)Modeling

定义

$p(x) = xB[0] + xB[1] + ... + xB[n-1]$
注意到
$p(x)n$ 的每一项为 $\prod_{j=1}^{n}x^{B[i_{j}]}=x^{\sum_{j=1}^{n}{B[i_{j}]}}$，因此我们只需要检查 $p(x)n$ 是否包含 xm 这一项即可。

1. Algorithm description

令 $p(x) := \sum_{j=1}^{n}x^{B[j]}$ 。
令
$q(x) := p(x)n$, 并使用FFT计算。

检查
q 中 xm 的系数是否为0。

1. Time complexity

FFT的时间复杂度为 $\mathcal{O}(n\log n)$, 则做 $n-1$ 次FFT的时间复杂度为 $\mathcal{O}(n2\log n)$ 。

1. Space complexity

FFT的空间复杂度为 $\mathcal{O}(n)$ 。

Ex. Unary Cubic Equation

1. Modeling
   - 三个答案都在$[-100,100]$范围内，两个根的差的绝对值>=1,保证了每一个大小为1的区间里至多有1个解，也就是说当区间的两个端点的函数值异号时区间内一定有一个解，同号时一定没有解。
   - 因此可以利用分治思想划分查找解的范围，在区间$[i,i+1]$内进行二分查找。
   - 终止条件设定为区间长度足够小，从而获得根的近似解。
2. Algorithm description
   - 确定区间范围：设定一个初始的搜索区间，题目要求根的范围在 $[-100,100]$ 间。
   - 将当前的区间分成两部分，$[left,mid]$和$[mid,right]$。
   - 根据题目中的提示，如果在区间两端点处函数值的乘积小于零，则在该区间内必然存在一个根。因此查找上一步中划分的两个区间。
   - 递归缩小区间：根据上一步判断的结果，选择包含根的区间，并递归地重复步骤 2 和 3，继续缩小包含根的区间范围。
   - 终止条件：当区间的长度小于某个很小的阈值（例如 0.001）时，可以认为找到了根。此时可以将区间中点作为根的近似值输出。
3. Time complexity
   - $O(logN)$
   - 分析：最基础的二分过程，每次查找都可以筛选掉一半范围
4. Space complexity

- $O(1)$
- 分析：查找到输出即可，没有中间变量需要存储

Ex. Distance

1. Modeling

- 因为要满足对于arr1中任意的元素都不满足|arr1[i] - arr2[j]| <= d这个条件，所以：对于arr1中任意元素arr1[i]，只要arr2中有一个元素满足了 arr1[i] - d <= arr2[j] <= arr1[i] + d，那么arr1[i]就不符合题意。
- 可以将问题转化为：在arr2中找满足 “arr1[i] - d <= arr2[j] <= arr1[i] + d”这个条件的元素，如果能找到，则说明arr1[i]不合题意。如果找不到，则说明arr1[i]符合题意。
- 因此可以用排序+二分查找的方式求解。

1. Algorithm description
2. 对数组 arr2 进行升序排序。
3. 二分查找
    a. 遍历arr1中的所有元素，当前元素为arr[1],left = 0, right = len(arr2) - 1，mid = (left + right) / 2.
    b. 如果arr2[mid] 满足 arr1[i] - d <= arr2[j] <= arr1[i] + d，则arr1[i]不合题意
    c. 如果arr2[mid] > arr1[i] + d， right = mid -1
    d. 如果arr2[mid] < arr1[i] - d, left = mid+1
    e. 当left >= right跳出循环
4. 统计符合要求的arr1元素的个数，返回最终的结果。
5. Time complexity

- 排序：对 arr2 进行排序的时间复杂度为 O(mlogm)，其中 m 是 arr2 的长度。
- 二分查找：对 arr1中的每个元素进行二分查找，每次查找的时间复杂度为 O(logm)，总共有 n 个元素要查找，因此查找的总时间复杂度为 O(nlogm)，其中 n 是 arr1 的长度。

因此，整个算法的时间复杂度为：
O(*mlogm*+*nlogm*)

1. Space complexity

O(1) 【+分析】

OJ题

Nearest Point

(1) Modeling

- 可用分治策略：将点按x坐标排序，然后分别处理左右两边和中间部分。
- 在中间部分，设$delta$为左右两边中更小的结果，那便只有处于距中间线$delta$以内的点才有更新结果的机会。
- 搜索中间部分时，可按y坐标排序并依据$delta$进行剪枝，此排序利用分治过程进行归并排序以减少运算。

(2) Algorithm description

1. 将station和agent的坐标合并成一个点集$point$。
2. 按x坐标对点集进行排序。
3. 分治法求解最近点对$F(l, r)$:
   a. 边界条件：当点的数量为1时，返回距离无穷大。
   b. 递归：将点集按x坐标分成左右两部分，递归计算左右两部分的最小距离分别为$F(l, mid)$、$F(mid+1, r)$。
   c. 按y坐标，执行归并排序的合并操作。（此时按x递归的操作已经完成，重新排序不会影响递归）
   d. 取中间线$pivot = (point[mid].x + point[mid+1].x)/2$，将$x$坐标距离中间线小于$delta = min(F(l, mid), F(mid+1, r))$的点加入$point_pivot$。
   e. 利用y坐标剪枝搜索$point_pivot$中stations和agents间的最小距离dis_pivot。

    f. 返回值：则区间内的最小距离$F(l,r)$为$delta$、dis_pivot中的较小值。

4. 剪枝搜索$Search(point_pivot)$:
    a. 枚举$node_i$。
    b. 嵌套枚举$node_j$（从$node_i$后一个开始）。
        i. 若$node_j.y - node_i.y \geq delta$，则剪枝，即break掉枚举$node_j$。
        ii. 根据$node.tag$，若$node_i$和$node_j$分别为station和agent才计算并更新最小值。
        iii. 返回值：若搜索不到则返回无穷大，否则返回搜索到的最小值。
5. 最终结果:$F(1,2N)$。

(3) Time complexity

易见步骤2、3.c、3.d的时间复杂度分别为$O(NlogN)$、$O(n)$、$O(n)$。

证：剪枝搜索$Search(n)$的复杂度为$O(n)$

- 根据剪枝规则可知，Search函数中，对每个点$node_i$的搜索空间为$2delta * delta$矩形范围。
- 由$delta$定义可知在该范围内，绝不会超过8个点。

已知枚举$node_i$的复杂度为$O(n)$，则证。

分治法时间复杂度$F(n)$: $O(n) + 2F(n/2)$ → $O(nlogn)$。

主函数: $O(N) + O(NlogN) + F(2N)$ → $O(2Nlog2N)$。

则总体复杂度为$O(NlogN)$。

(4) Space complexity

开数组计算在递归操作后，因此为$O(N)$。

A unique permutation

1. Modeling
    - 维护一个$0 \sim n-1$的数列，使其选出长度大于3的子序列（可以不连续）都不能是等差数列。

- 正难则反，分析发现一个非法的序列至少包含一个长度为3的等差数列，可以将原序列拆分成两个等差数列，得知两序列中间部分要么是两个偶数一个奇数，要么是两个奇数一个偶数，判断这一部分一定是合法的。
- 所以每次将序列拆分成两段不合法等差序列时，会构造出两序列中间部分的合法区间。
- 因此使用分治法，重复这一步，直至将整个区间处理为合法，当处理区间长度小于2时就不必拆分了。
- 注：此题答案不唯一，构造合理即可

2. Algorithm description
   - 判断当前区间 $[l, r]$ 的长度，如果小于等于 2，则认为不再需要拆分和处理，直接返回。
   - 计算区间的中点 $mid = (l + r)/2$，将当前区间拆分成左右两部分 $[l, mid]$ 和 $[mid + 1, r]$。
   - 初始化 t_1 和 t_2，分别为当前区间的起始值 $a[l]$ 和 $a[l + 1]$，用于构造左右部分的合法等差序列。
   - 构造左半部分 $[l, mid]$：从 l 开始赋值，每次步长为 p_l，逐渐递增赋值给 $a[i]$，构造出合法的等差序列。
   - 构造右半部分 $[mid + 1, r]$：从 $mid + 1$ 开始赋值，同样每次步长为 p_l，逐渐递增，形成右半部分的合法序列。
   - 递归调用：对左右两部分再进行递归处理，步长 p_l 每次翻倍 $p_l * 2$，这样每一层递归构造的序列递增速度加快，保证等差性。
3. Time complexity
   - $O(NlogN)$
   - 分析：最外层是简单的二分递归逻辑，每层遍历时步长 p_l 翻倍，时长减半。
4. Space complexity
   - $O(N)$
   - 分析：只需要存储一次序列 $a[0, N]$，通过移动指针确定递归查找范围，并在原序列上修改即可。

Lost items

1. Modeling

穷举法的时间复杂度为$O(N*2N)$，当N较大时复杂度太高。因此我们可以使用meet in the middle方法，将问题分解为两个较小的子问题来解决。在该问题中，我们可以把物品集合分成两半，分别计算两部分所有可能的重量组合。对于一个集合中的每个子集，在另一个集合中找配对的子集，使得两个子集的重量和等于D。

1. Algorithm description

1.物品集合分为两部分：集合A包含前半部分物品，集合B包含后半部分物品，每个集合的大小为 $N/2$。

2.枚举两部分集合的所有组合：

- 对于集合A，枚举所有可能的子集，计算每个子集的总重量，并记录下这些重量及其对应的物品数量。
- 对于集合B，做相同操作，计算所有可能的子集重量。

3.寻找满足条件的组合：

为了快速查找集合B中满足条件的子集，可以对集合B的子集总重量进行排序，然后使用**二分查找**来高效找到合适的配对。

4.分析结果：

- 如果找到一个配对子集，计算其物品数量。
- 如果有多个组合满足 WA+WB=DW_A + W_B = DWA+WB=D，输出 “AMBIGIOUS”。
- 如果找不到满足条件的组合，则输出 “IMPOSSIBLE”。

1. Time complexity

1.枚举子集的复杂度：

集合A和集合B各有$(N/2)$个物品，每个集合有$2N/2$种可能的子集，计算所有子集的和需要 $(N/2)$的时间，总复杂度为$O(2N/2 \cdot (N/2)) = O(2N/2 \cdot N)$。

2.排序：

B集合有$2N/2$个子集，对B集合排序，复杂度为$O(2N/2 log 2N/2) = O(2N/2 \cdot N)$。

3.二分查找

A集合有$2N/2$个子集，对每个子集进行一次二分查找的复杂度为$O(log2N/2)=O(N)$。

总的时间复杂度即，上述步骤的时间复杂度之和$O(2N/2 \cdot N)$。相比穷举法指数复杂度的指数从N降低到$N/2$。

1. Space complexity

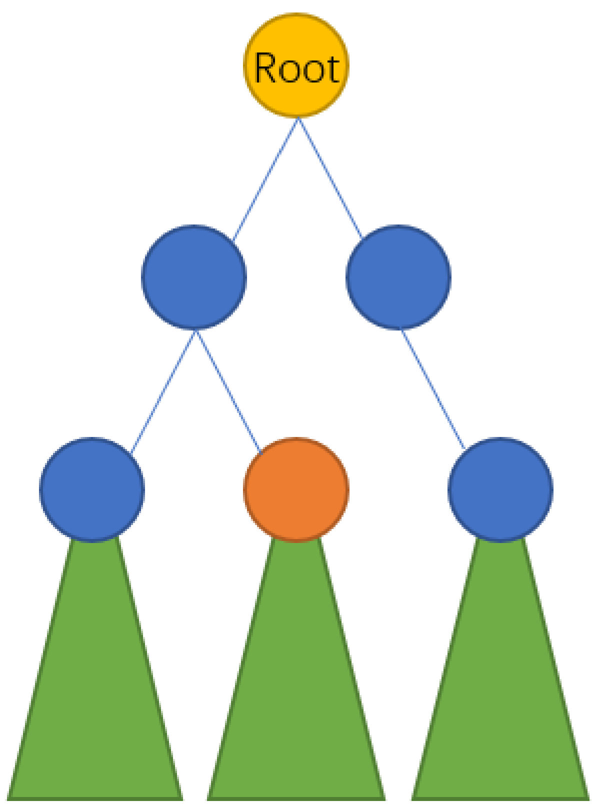
分治法将物品集合分为两部分，每一部分枚举所有可能的子集，并将每个子集的总重量存储下来，以便后续进行匹配和查找。因此，存储这些中间结果是空间复杂度的主要来源。

输入存储：同样需要$O(N)$的空间存储N个物品的重量。

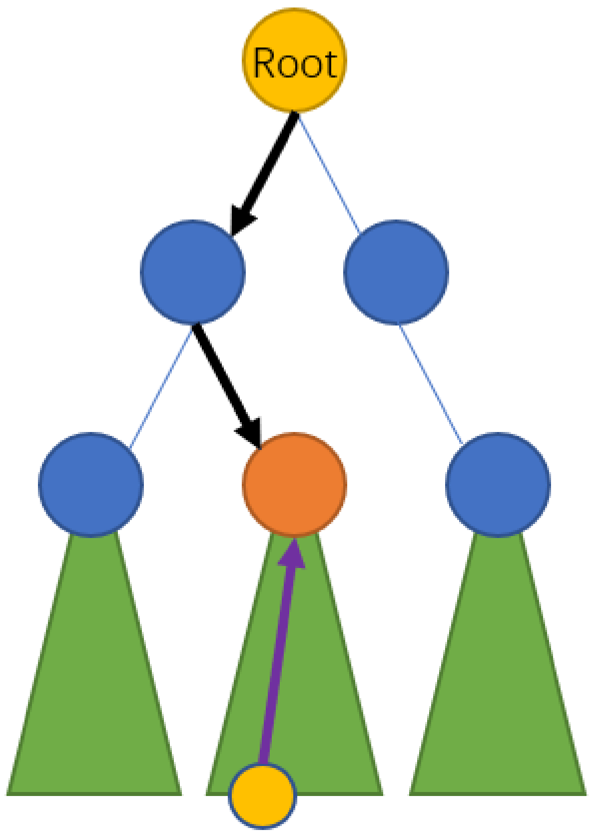
子集存储：集合A和集合B都有$2N/2$个物品，都需要$O(2N/2)$的空间保存子集和。所以总的空间复杂度是$O(2N/2)$。

Extra Problem

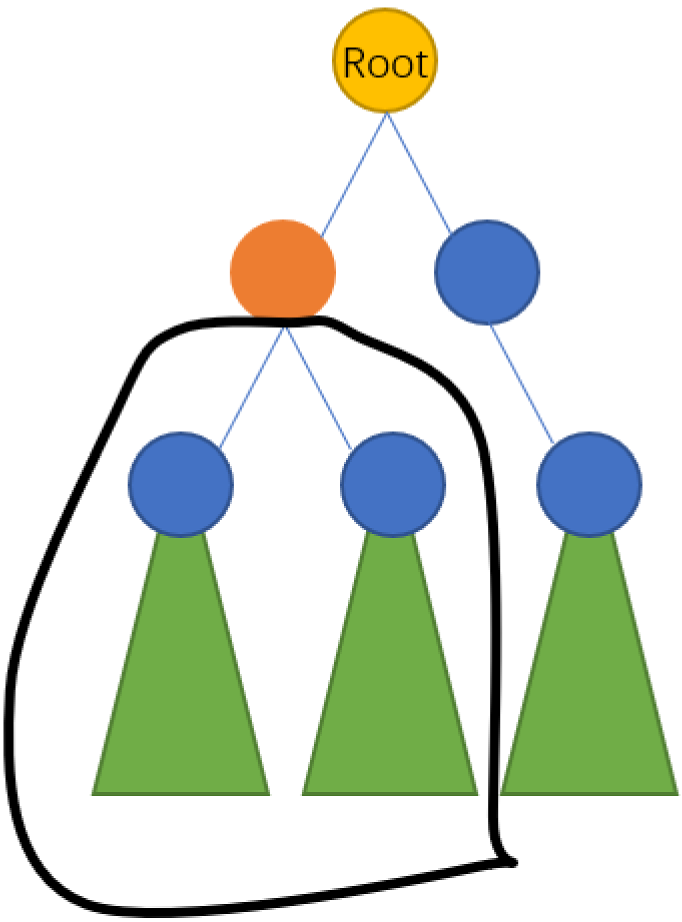
定义："封锁点"，就是在站在一个点上阻止贝西通行。一旦准时站在点 u上，就等于封锁了整个子树内的点，因为贝西将无法进入这棵子树内部。目标是封锁所有叶节点。



我们先看这张图。我们希望确定橙点能不能成为一个封锁点。封锁点要求能**成功准时**地封锁住贝西，所以一定需要存在一个出口，使得农夫能**在贝西抵达并穿过封锁点进入子树前到达封锁点**。
因为可以说准时到达封锁点是让贝西不进入子树内地叶节点的
唯一机会。一旦比贝西晚到达，你就永远也追不上它了，它会长驱直入逃走。
所以我们必须制定这个规则。



黄色小点是最接近封锁点的出口，紫色路径是农夫到达封锁点的路径，黑色路径是贝西试图到达封锁点然后长驱直入逃跑的路径。
显然，我们需要紫色路径长度≤黑色路径长度。
还有，为了保证是最优解，一旦父亲可以做封锁点，那么儿子就不用再做封锁点了。



我们假设橙点是一个封锁点。因为父亲已经封锁了整个子树，所以儿子就不用（橙点的两个子节点）就不用再作为封锁点浪费农民资源了。
于是，我们得到了一个点 u应该是封锁点的条件：

$d_{u, \text{root}} \geq d_{u, f_u}$ 且 $d_{\text{fa}u, \text{root}} \leq d{u, f_{\text{fa}_u}}$

其中f_i代表离 i 最近的叶节点，d代表两点距离，fa_u表示封锁点u的父亲节点。

1. Modeling :
    - 贝西需要到达树的叶节点，因此我们将他的起点设为树的根节点。
    - 农民需要在特定的封锁点上站立，这些封锁点能有效阻止贝西进入其对应的子树，从而封锁所有叶节点。
2. Algorithm description :

a. **初始化**：

- 使用 DFS 遍历树，计算每个节点到根节点的距离 $d_{u, \text{root}}$、最近叶节点 f_u、父节点 fa_u 和每个节点的度数$deg(u)$。

b. **判断封锁点**：

- 对每个节点 *u*，判断是否可以成为封锁点，条件是：
  - $d_{u, \text{root}} \geq d_{u, f_u}$（农民到达封锁点的时间早于贝西到达相应子树）
  - $d_{\text{fa}u, \text{root}} \leq d{u, f_{\text{fa}_u}}$（父节点的条件也要满足）

c. **计数封锁点**：

- 如果节点 *u* 符合封锁点条件，则将计数器 `ans` 增加。

d. **优化统计**

目标是求满足以下条件的点的数量：

- 一个点 i 满足 ，其中：${dep}_i \geq f_i$
  - ${dep}_i$: 点 的深度（相对于当前树根）。
  - f_i: 点 到最近叶子的距离。

满足这个条件的点会形成子树，而最高的符合条件的点是这些子树的根。

关键性质

满足条件的点数，可以通过 ${deg}_i$（节点度数）和 ${dep}_i$（深度）的组合统计得到。

- 树的度数和满足 。

  $ \sum {deg}_i = 2n - 1$

- 通过数学变化，公式变为$\sum (2 - \text{deg}_i) = 1$ ，即所有符合条件的点的贡献之和为 1。

通过以上分析，答案可以表示为满足以下条件的点的加和：

${ans} = \sum_{\text{dep}_i \geq f_i} (2 - \text{deg}_i)$

这里 $2- \text{deg}_i $是每个点的贡献。

点分治的思路

点分治将整棵树拆分为多个部分，逐步计算满足条件的点数。

具体步骤

1. **深度和叶子距离分解：**
    - 每个点的深度 可以分解为两部分：

      ${dep}_i$

        - $\text{dis}(\text{lca}, i)$: 当前点到 LCA 的距离。
        - $\text{dis}(\text{lca}, \text{rt})$: LCA 到当前根的距离。
    - 目标条件变为：$\text{dis}(\text{lca}, \text{rt}) \geq f_i - \text{dis}(\text{lca}, i) 其中右边的部分 f_i - \text{dis}(\text{lca}, i) 可以预处理。$
2. **分治递归：**
    - 找到当前树的重心，将其作为根。
    - 切分树，递归处理每个子树。

3. Time complexity

- 时间复杂度为 $O(nlog^2n)$。

4. Space complexity

- 空间复杂度为 $O(n)$