# 算法设计与分析第一次作业

**姓名：王泽黎**
**学号：2022K8009929011**

# 1.Longest Balanced Substring

## 1.1 Modeling

核心要求：
1. 如果每个字母的大小写形式至少出现一次，字符串被称为平衡字符串。
2. 在给定字符串 s 中，找到最长的平衡子串。

关键变量：
1. 输入变量：
字符串 s，表示给定的字符串
2. 输出变量：
longestSubstring 最长的平衡子串

分析思路：
1. 该问题本质是一个从字符串中选择一个满足特殊条件的子串的问题。其中，满足条件的子串是指每个字母的大小写形式至少出现一次。
2. 该问题可以通过暴力搜索字符串 s 的所有子串，判断每个子串是否满足条件，从而找到最长的平衡子串。但是这种方法的时间复杂度较高，不适合大规模数据。
3. 考虑滑动窗口的方法，可以有效降低时间复杂度。滑动窗口的思想是维护一个窗口，窗口内的元素满足某种条件，窗口的大小可以根据条件的变化而变化。

## 1.2 Algorithm description

1. 初始化变量：
    - 定义两个指针 left 和 right，分别表示滑动窗口的左右边界，且均指向字符串 s 的第一个字符。
    - 定义一个字典 count，用于记录窗口内每个字母的大小写形式出现的次数。
    - 定义变量 maxLength，用于记录最长的平衡子串的长度。
    - 定义变量 longestSubstring，用于记录最长的平衡子串。
2. 向右拓展窗口：
    - 使用 right 指针向右移动，逐个字符加入窗口。

    - 更新 count 字典，统计窗口内每个字母的大小写形式出现的次数。
3. 检查窗口内的子串是否满足条件：
    - 如果 count 字典中的所有值均大于等于 1，则表示窗口内的子串满足条件。
    - 如果满足条件，则更新 maxLength 和 longestSubstring。
    - 如果不满足条件，则向右移动 left 指针，缩小窗口，直到满足条件。
4. 返回最长的平衡子串 longestSubstring。

## 1.3 Complexity analysis

- 时间复杂度：O(n)，时间复杂度主要需要考虑遍历字符串 s 和窗口调整的过程，其中 n 表示字符串 s 的长度，遍历字符串 s 的时间复杂度为 O(n)，窗口调整的时间复杂度也为 O(n)，因此总的时间复杂度为 O(n)。
- 空间复杂度：O(1)，空间复杂度主要需要考虑 count 字典的空间占用，count 字典的大小不会超过 52 （26 个小写和 26 个大写），因此空间复杂度为 O(1)。

# 2. Cutting Bamboo Poles

## 2.1 Modeling

核心要求：
1. 对于给定的长度 L，需要验证是否可以从所有竹竿中切割出至少 m 根长度为 L 的竹竿。这个过程涉及对每根竹竿进行遍历，计算能够切割出的竹竿数量。

关键变量：
1. 输入变量：
n: 竹竿的数量
m: 需要的竹竿数量
l_i: 每根竹竿的长度
2. 输出变量：
maxLength 满足要求的最长的竹竿长度

分析思路：
1. 该问题本质是一个对每根竹竿进行遍历的问题，需要计算能够切割出的竹竿数量。
2. 考虑二分查找的方法，通过在可能的长度范围内（1 到最长竹竿的长度）进行搜索，逐步缩小范围，直到找到最大可行长度。

## 2.2 Algorithm description

1. 初始化变量：

    - 定义变量 low，表示二分查找的左边界，初始值为 1。
    - 定义变量 high，表示二分查找的右边界，初始值为 max(l_1, l_2, ...)。
    - 定义变量 maxLength，表示满足要求的最长的竹竿长度。
2. 进行二分查找：
    - 在 low 和 high 之间进行二分查找，计算中间值 mid。
    - 对每根竹竿进行遍历，计算能够切割出的竹竿数量。
    - 如果能够切割出的竹竿数量大于等于 m，则更新 maxLength，并将 low 设置为 mid + 1。
    - 否则，将 high 设置为 mid - 1。
    - 重复上述步骤，直到 low 大于 high。
3. 返回 maxLength。

## 2.3 Complexity analysis

- 时间复杂度：O(nlogL)，时间复杂度主要需要考虑二分查找和检查可行性的过程，二分查找的时间复杂度为 O(logL)，而检查可行性的时间复杂度为 O(n)，其中 n 表示竹竿的数量，L 表示最长竹竿的长度，因此总的时间复杂度为 O(nlogL)。
- 空间复杂度：O(1)，空间复杂度主要需要考虑变量 low、high 和 maxLength 的空间占用，因此空间复杂度为 O(1)。

# 3. Multiple Calculations

## 3.1 Modeling

核心要求：
1.根据给定字符串 s，通过计算数值和运算符的不同分组方式，输出所有可能的计算结果。

关键变量：
1. 输入变量：
字符串 s，表示给定的字符串
2. 输出变量：
result 所有可能的计算结果

分析思路：
1. 该问题本质是一个对字符串进行分组和计算的问题，需要考虑不同的分组方式和运算符的优先级。
2. 考虑使用递归的方法，对字符串进行分组，计算每个分组的结果，然后合并结果。

## 3.2 Algorithm description

1. 初始化变量：

    - 定义函数 diffWaysToCompute(s)，用于递归计算字符串 s 的所有可能的计算结果。
    - 定义变量 result，用于存储所有可能的计算结果。
2. 递归计算字符串 s：
    - 判断当前字符串 s 是否只有数字，如果是，则直接返回 s。
    - 否则，遍历字符串 s，根据运算符将字符串分为左右两部分，并调用 diffWaysToCompute 计算左右两部分的所有可能的计算结果。
    - 对左右两部分的计算结果进行组合，根据运算符计算结果，并将结果添加到 result 中。
3. 返回 result。

## 3.3 Complexity analysis

- 时间复杂度：O(n!)，时间复杂度主要需要考虑递归计算的过程，递归的深度为 n，每次递归都需要遍历字符串 s，因此时间复杂度为 O(n!)。
- 空间复杂度：O(n)，空间复杂度主要需要考虑递归的过程中的栈空间占用，递归的深度为 n，因此空间复杂度为 O(n)。

# 4. N-sum

## 4.1 Modeling

核心要求：
1. 判断数组 B 中是否存在数值之和等于给定值 m，且仅需判断是否存在，不需要输出具体的组合。

关键变量：
1. 输入变量：
数组 B，表示给定的数组
m，表示给定的值
2. 输出变量：
result，表示是否存在数值之和等于 m

分析思路：
根据提示，我们可以利用指数编码的技巧，将问题转化为多项式乘法的问题，思路大致如下：
1. 构建多项式：将数组 B 中的每个元素 B[i] 转换为多项式的形式。具体来说，对于每个 B[i]，我们构建一个多项式 P(x) = x^(B[i]) + 1。
2. 多项式乘法：将所有的多项式相乘，得到一个最终的多项式 Q(x)。多项式 Q(x) 的系数表示不同和的可能性。
3. 检查结果：检查多项式 Q(x) 中是否存在 x^m 的系数不为零。如果存在，则表示存在一组索引使得 B[i1] + B[i2] + ... = m。

## 4.2 Algorithm description

1. 初始化变量：
   - 定义函数 nSumExists(B, m)，用于判断数组 B 中是否存在数值之和等于 m。
   - 定义变量 result，表示是否存在数值之和等于 m。
2. 构建多项式：
   - 对数组 B 中的每个元素 B[i]，构建一个多项式 P_i(x) = x^(B[i]) + 1。
3. 多项式乘法：
   - 将所有的多项式相乘，得到一个最终的多项式 Q(x)。
4. 检查结果：
   - 检查多项式 Q(x) 中是否存在 x^m 的系数不为零。
   - 如果存在，则返回 True，表示存在数值之和等于 m。
   - 否则，返回 False，表示不存在数值之和等于 m。

## 4.3 Complexity analysis

- 时间复杂度：O(n^2 log n)，多项式乘法的时间复杂度为 O(n^2)，因为我们需要进行 n 次多项式乘法，每次乘法的复杂度为 O(n^2)。因此，总的时间复杂度为 O(n^2 * n) = O(n^3)。但是，通过快速傅里叶变换（FFT）可以将多项式乘法的复杂度降低到 O(n log n)，因此总的时间复杂度可以优化到 O(n^2 log n)。
- 空间复杂度：O(n^2)，存储多项式的空间复杂度为 O(n^2)，因为多项式的最高次项为 n^2。

# 5. Unary Cubic Equation

## 5.1 Algorithm description

核心要求：
1. 确定形如 ax^3 + bx^2 + cx + d = 0 的一元三次方程的系数 a, b, c, d ，保证方程有三个不同的实数根。
2. 三个实数根的取值范围为 [-100, 100]，且每对根之间的绝对差值大于等于1。

关键变量：
1. 输入变量：
无
2. 输出变量：
x1, x2, x3 三个实数根的值

分析思路：
由于只需要求出一组满足条件的实数根，不妨考虑通过根来构造方程的系数。具体来说，我们可以通过

枚举三个实数根的值，通过Vieta定理构造对应的方程，检查方程的系数是否满足条件。

## 5.2 Algorithm description

1. 初始化变量：
    - 定义函数 solveCubicEquation()，用于求解一元三次方程的系数。
    - 定义变量 x1, x2, x3，表示三个实数根的值。
2. 枚举实数根：
    - 对三个实数根的值 x1, x2, x3 进行枚举，范围为 [-100, 100], 且每对根之间的绝对差值大于等于 1。
3. 通过Vieta定理构造方程：
    - 根据Vieta定理，我们可以得到方程的系数 a, b, c, d。
4. 检查方程的系数：
    - 检查方程的系数是否满足条件，即方程有三个不同的实数根。
    - 如果满足条件，则返回实数根的值 x1, x2, x3。

## 5.3 Complexity analysis

- 时间复杂度：O(1)，时间复杂度主要需要考虑枚举实数根和通过Vieta定理构造方程以及检查，实数根的范围为 [-100, 100]，后续运算的时间复杂度也是 O(1)，因此总的时间复杂度为 O(1)。
- 空间复杂度：O(1)，空间复杂度主要需要考虑变量 x1, x2, x3, a, b, c, d 以及输出数组 root 的空间占用，因此空间复杂度为 O(1)。

# 6. Distance

## 6.1 Modeling

核心要求：
1. 求出满足以下条件的元素数量：对于 arr1 中的每个元素 arr1[i]，没有任何 arr2[j] 满足条件 |arr1[i] - arr2[j]| ≤ d。

关键变量：
1. 输入变量：
arr1，表示第一个数组
arr2，表示第二个数组
d，表示给定的值
2. 输出变量：
result，表示满足条件的元素数量

分析思路：
对于每个 arr1[i]，必须检查所有 arr2[j] 的值，需要高效地判断每个 arr1[i] 是否满足条件，避免暴力搜索。于是选择使用二分查找的方法，对 arr2 进行排序，然后对 arr1 中的每个元素 arr1[i]，在 arr2 中查找满足条件的元素。

## 6.2 Algorithm description

1. 初始化变量：
    - 定义变量 result，表示满足条件的元素数量，初始值为 0。
2. 排序：
    - 对 arr2 进行排序。
3. 递归分解arr1：
    - 将 arr1 分成两个子数组。对于每个子数组，递归地计算各自的距离值。
    - 使用二分查找来快速确定是否存在满足条件的元素。
4. 合并结果：
    - 逐级合并子数组的结果，返回最终的满足条件的元素数量。

## 6.3 Complexity analysis

- 时间复杂度：O(nlogn)，时间复杂度主要需要考虑排序 arr2 和二分查找的过程，其中 n 表示数组 arr1 和 arr2 的长度，排序 arr2 的时间复杂度为 O(nlogn)，二分查找的时间复杂度为 O(logn)，所以递归的时间复杂度为 O(lognlogn)，因此时间复杂度为 O(nlogn)。
- 空间复杂度：O(n)，空间复杂度主要需要考虑递归的过程中的栈空间占用，递归的深度为 O(logn)，因此空间复杂度为 O(logn)。