246458-池聪哲

问题 1：给定N个 1-9 的数字，插入K个乘号和$N-K-1$个加号，使最终结果尽量大。

伪代码设计

首先，我们需要理解这个问题的优化子结构和子问题重叠性：

- **优化子结构**：我们可以从每一个数字出发，选择最优的方式将其与前面的数字进行组合（加法或乘法）。这样，通过合适的分割点，子问题的最优解可以组合成全局问题的最优解。
- **子问题重叠性**：我们可以通过存储中间结果避免重复计算。每次我们处理一个子问题时，都会基于前面的解来计算。

我们可以使用动态规划来解这个问题。假设 `dp[i][k]` 表示从第 1 个数字到第 i 个数字，使用 k 个乘号时的最大结果。

伪代码：

```
def maxExpression(nums, N, K):
    # dp[i][k] 表示前i个数字，使用k个乘号的最大值
    dp = [[-float('inf')] * (K + 1) for _ in range(N + 1)]
    dp[0][0] = 0  # 初始状态，0个数字，0个乘号，结果为0

    # 填充dp数组
    for i in range(1, N + 1):  # 遍历所有数字
        for k in range(0, min(K, i - 1) + 1):  # 遍历使用的乘号数量
            # 遍历前一个位置
            for j in range(0, i):  # 遍历i前面的所有位置
                if k > 0:  # 如果k>0，则可以尝试将当前数字与前面的乘号组合
                    dp[i][k] = max(dp[i][k], dp[j][k-1] * calc(nums[j:i]))  # 乘法
                # 如果没有乘号，直接加法
                dp[i][k] = max(dp[i][k], dp[j][k] + calc(nums[j:i]))  # 加法

    # 返回最终答案
    return dp[N][K]

# 计算从nums[l]到nums[r]的表达式值，是否加法或乘法
def calc(nums):
    if len(nums) > 1:
        # 计算加法或乘法表达式
        return sum(nums)  # 加法
    return nums[0]  # 单一数字时返回自身
```

时间复杂度分析：

- 状态 `dp[i][k]` 是根据之前的状态计算的。
- 内层循环遍历前面的每个数字，因此时间复杂度为 $O(N^2K)$，其中 N 是数字个数，K 是乘号的个数。

因此，总的时间复杂度为 $O(N^2K)$。

问题 2：将整数序列划分为多个子序列，使每个子序列的和不大于一个给定值B，最小化所有子序列中最大值的和。

伪代码设计

此问题的优化子结构和子问题重叠性如下：

- **优化子结构**：通过动态规划来选择在某一个位置分割，使得子序列和不超过 B，同时最小化所有子序列最大值的和。
- **子问题重叠性**：每个子问题的解都依赖于前面某些子问题的解，因此可以用动态规划存储每个子问题的解，避免重复计算。

我们可以用动态规划来定义 `dp[i]`，表示将前 i 个数分成若干个子序列的最优解。

伪代码：

```python
def minMaxSum(nums, N, B):
    # dp[i] 表示将前i个数分成子序列，且每个子序列和不超过B时，最小化最大子序列和的总和
    dp = [float('inf')] * (N + 1)
    dp[0] = 0  # 0个数时，最大和为0

    # 遍历所有位置
    for i in range(1, N + 1):
        current_sum = 0
        max_subsequence = 0
        # 从j到i位置计算和
        for j in range(i, 0, -1):  # j 从 i 到 1 遍历
            current_sum += nums[j-1]  # 计算当前子序列和
            if current_sum > B:  # 如果当前和超过B，则跳出
                break
            max_subsequence = max(max_subsequence, current_sum)  # 更新当前子序列的最大和
            dp[i] = min(dp[i], dp[j-1] + max_subsequence)  # 更新最小化最大和的结果

    return dp[N]
```

时间复杂度分析：

- 外层循环遍历每个位置 `i`，内层循环从 `i` 遍历到 1，计算每个子序列的和。因此，时间复杂度为 $O(N^2)$。

总结

1. **问题 1** 使用动态规划，时间复杂度为 $O(N^2K)$，通过在每个数字间插入加法和乘法符号来最大化结果。
2. **问题 2** 使用动态规划，时间复杂度为 $O(N^2)$，通过划分子序列来最小化所有子序列最大值的和。

两者都体现了动态规划的优化子结构和子问题重叠性，通过保存子问题的解来避免重复计算。