算法基础

第五次作业（DDL：2024 年 11 月 5 日 23:59）

解答过程中请写出必要的计算和证明过程

Q1.(20 分) 若正整数序列 $a_1, a_2, \ldots, a_n$ 满足：

1. $a_1 = 1$;
2. $a_j \le \max\limits_{1\le i\le j-1} a_i + 1, \forall j \in [2, n]$,

我们称这个正整数序列具有限制增性质。

请设计一个多项式时间复杂度的动态规划算法，计算长度为 n 的正整数序列中满足限制增性质的序列数目。

例如，长度为 1 的正整数序列中满足限制增性质的序列数目为 1 个，为 [1]; 长度为 2 的正整数序列中满足限制增性质的序列数目为 2 个，为 $[1,1]$,$[1,2]$; 长度为 3 的正整数序列中满足限制增性质的序列数目为 5 个，为 $[1,1,1]$,$[1,1,2]$,$[1,2,1]$,$[1,2,2]$,$[1,2,3]$.

简述算法过程，给出递归式，并简单分析算法复杂度。不需要说明算法正确性。

solution

令 $num(l,m)$ 为长度 l 时的序列数，m 为长度 l 的正整数序列的最大值。由限制增的性质，注意到长度为 n 的整数列中元素的最大值不会超过 n。

对长度为 l 中，前 $l-1$ 个元素中是否存在最大值 m 分情况进行讨论：1. 若存在，则 a_l 可以取 $[1,m]$ 中的任意一个值；2. 若不存在，则 $a_l = m$。这意味着前 $l-1$ 个元素中的最大值只可能为 $m-1$。于是可以得到递归式：

$$
\begin{aligned}
&num(l,m) = num(l-1,m-1) + m \cdot num(l-1,m),\\
&(num(l,1) = 1,\ num(l,m) = 0 \text{ if } l < m)
\end{aligned}
$$

由递归式，我们需要解决 n^2 个子问题。可以设计一个 $n \cdot n$ 的表格（l 与 m 的取值范围均为 $[1,n]$）解决每一个子问题只需要常数时间。在最后一步，对这个表格的最后一行相加即可得到解：$\sum_{i=1}^{n} num(n,i)$，而这只需要线性时间。故总的时间复杂度为 $O(n^2)$。

Q2.(20 分) 在 3-划分问题中，目标是将集合 S 划分成 3 个和相等的子集。

例如，

$S = \{7, 3, 2, 1, 5, 4, 8\}$

我们可以将集合 S 划分成 3 个子集，每个子集的和为 10。

$S_1 = \{7, 3\}$

$S_2 = \{5, 4, 1\}$

$S_3 = \{8, 2\}$

请设计一个 3-划分问题的动态规划算法, 简述算法过程，给出递归式，并简单分析算法复杂度。不需要说明算法正确性。

solution

算法过程：

1. 初始化：计算集合 S 的总和。如果元素个数小于 3 或总和不为 3 的倍数，则无法划分。

2. 递归求解：使用递归函数 `SubsetSum` 递归判断能否划分。

3. 记录状态：通过字典 `lookup` 存储状态 (a, b, c, n) 的结果，避免重复计算。

4. 递归条件：

- 基础情况：如果 $a = b = c = 0$，表示成功划分。
- 否则，将当前元素尝试加入不同子集，并递归检查。

递归式：

$$\texttt{SubsetSum}(S, n, a, b, c) = \begin{cases} \text{True}, & \text{if } a = 0, b = 0, c = 0 \\ \text{False}, & \text{if } n < 0 \\ \texttt{SubsetSum}(S, n-1, a - S[n], b, c) \vee & \\ \texttt{SubsetSum}(S, n-1, a, b - S[n], c) \vee & \\ \texttt{SubsetSum}(S, n-1, a, b, c - S[n]), & \text{if } a, b, c \geq 0 \end{cases}$$

复杂度分析：状态空间是 $\mathcal{O}(n \times (\frac{\text{sum}(S)}{3})^3)$，其中 n 是集合大小。

Q3. (30 分) 给定一个整数数组 $A[1:n]$, 找到一个具有最大和的连续子数组(子数组最少包含一个元素), 比如数组 $[-1,7,-2,3]$ 的一个具有最大和的连续子数组为 $[7,-2,3]$.

(a) 基于分治思想设计算法，并分析其时间复杂度（算法时间复杂度不得超过 $O(n\log n)$）。

(b) 用动态规划的方法在 $O(n)$ 时间内求解该问题, 根据你的思路列出你用到的边界条件和状态转移方程。

(c) 我们将一维的整数数组扩展到二维的矩阵, 试用动态规划的方法找到整数矩阵 M[1:m,1:n] 中具有最大和的子矩阵。简要说明你的算法并给出时间复杂度。

solution

(a) 算法思想：将数组分为两个相等子数组，则最大子数组要么出现在左边的数组里，要么出现在右边的子数组里，要么横跨左右数组。所以每个子数组分别递归求的最大子数组为 S_1，S_2，从原数组的中间出发，沿向数组首尾方向分别累加求和，分别找出左右两边的最大和 S_3，比较 S_1,S_2,S_3 的大小，最大的值对应的子数组即为所求。时间复杂度的递归方程为 $T(n)=2T(n/2)+O(n)$，可以求得时间复杂度为 $O(n\log n)$。

(b) 我们用 $f(i)$ 代表以第 i 个数结尾的连续子数组的最大和，我们要求的答案就是 $\max_{1\leq i\leq n}\{f(i)\}$ 而状态转移方程是 $f(i)=\max\{f(i-1)+A[i],A[i]\}$. 我们在计算 f(i) 的时候，更新 f(i) 对应的子数组开始和结束的位置即可。

(c) 假设子矩阵的开始行和结束行为 i 和 j, 那么我们把矩阵 $M[i:j,1:n]$ 按照列的方向求和，得到长度为 n 的序列, 然后求其最大子序和，对应的就是矩阵 $M[i:j,1:n]$ 的最大子矩阵，这里需要遍历所有的 i 和 j, 时间复杂度为 $O(m^2\times n)$。

Q4. (30 分) 叠叠乐

(a) 在一次图书漂流活动中，小明需要将收集到的旧书叠起来以充分利用存储室的空间。为了保证叠起来的书保持稳定，要求上层书的尺寸（长宽分别为 $p \times q$）必须严格小于下层书的尺寸（长宽分别为 $r \times s$），即 $p < r$ 且 $q < s$。此外，为了便于统计书名和书的数量，所有书的书脊必须朝向同一面（即书的长宽不能调换）。现在已知有 n 本旧书，它们的长宽分别为 $a_i \times b_i (1 \le i \le n,\ a_i > b_i)$，试设计最坏时间复杂度尽可能优的算法求这些书最多可能叠多少层。

(b) 积木同样可以堆叠。不过，积木的尺寸有三个维度，分别是长、宽、高。每块积木都能够以任意方向进行堆叠（即可以选择任意一面作为底面，底面的长宽也可以调换）。为了保持稳定，要求上层积木的底面（尺寸为 $p \times q$）必须严格小于下层积木的顶面（尺寸为 $r \times s$），即 $p < r$ 且 $q < s$，或 $p < s$ 且 $q < r$。现在已知有 n 种尺寸各异的积木（尺寸为 $a_i \times b_i \times c_i (1 \le i \le n)$），每种尺寸的积木至少有三块，试设计最坏时间复杂度为 $O(n^2)$ 的算法求这些积木最多可能叠多高。

solution

(a)

解法 1，同 (b), 只需要简化摆放方式为 $(a_i, b_i, 1)$ 即可。时间复杂度为 $O(n^2)$

解法 2，基于二分查找的动态规划，时间复杂度为 $O(n \lg n)$

设 $f[j]$ 表示经过排序后的第二维度 s 的前 i 个元素可以组成的长度为 j 的最长严格递增子序列的末尾元素的最小值，如果不存在长度为 j 的最长严格递增子序列，对应的 f 值无定义。在定义范围内，可以看出 f 值是严格单调递增的，因为越长的子序列的末尾元素显然越大。

在进行状态转移时，我们考虑当前的元素 s_i：

- 如果 s_i：大于 f 中的最大值，那么 s_i：就可以接在 f 中的最大值之后，形成一个长度更长的严格递增子序列；
- 否则我们找出 f 中比 s_i 严格小的最大的元素 $f[j_0]$，即 $f[j_0] < s_i \le f[j_0 + 1]$，那么 s_i 可以接在 $f[j_0]$ 之后，形成一个长度为 $j_0 + 1$ 的严格递增子序列，因此需要对 $f[j_0] + 1$ 进行更新：

$$f[j_0 + 1] = s_i$$

  我们可以在 f 上进行二分查找，找出满足要求的 j_0。

在遍历所有的 s_i 之后，f 中最后一个有定义的元素的下标增加 1（下标从 0 开始）即为最长严格递增子序列的长度。

(b)

将每种积木的尺寸 a_i, b_i, c_i 整理为 $x_i \le y_i \le z_i$ 的形式，并构造 3 种摆放方式 (x_i, y_i, z_i)、(x_i, z_i, y_i) 和 (y_i, z_i, x_i)。（每种积木可能的摆放方式）（或者构造 6 种摆放方式）

对所有摆放方式（共 $O(n)$ 种）按 x, y, z 的字典顺序进行排序，时间复杂度为 $O(n \lg n)$。得到排序后的摆放方式序列 $\{P_i | P_i = (r_i, s_i, t_i)\}$。

采用动态规划，求解以各个排列方式的积木为最底部的积木最大高度 $h(P_i)$。

$h(P_1) = t_1$（P_1 不能叠放在其他积木上，最大高度为自身高度 t_1）。积木 P_i 只可能叠放在积木 $P_j (1 \le j \le i-1)$ 下方（因为经过排序，后续积木的底面比积木 P_i 长或宽），且要求 $r_i > r_j$ 且 $s_i > s_j$（前面的积木的地面至少有一个维度小于等于积木 P_i）。故有，通过动态规划，每次都选取使得积木塔尽可能高的排列方式：

$$h(P_i) = t_i + \max\{0\} \cup \{h(P_j) | 1 \le j \le i-1, r_i > r_j, s_i > s_j\}$$

求 $h(P_i)$ 的时间复杂度为 $O(n)$。共求 $O(n)$ 次，时间复杂度为 $O(n^2)$。

最后对所有 $h(P_i)$ 求 max，时间复杂度为 $O(n)$。

总时间复杂度为 $O(n^2)$。