《计算机算法设计与分析》第三章作业

姓名：任宇　　学号：33920212204567

算法分析题 3-1 设计一个 $O(n^2)$时间的算法，找出由 n 个数组成的序列的最长单调递增子序列。

答：算法设计：

1. 初始化一个长度为 n 的数组 dp，dp[i]表示以第 i 个元素为结束元素的最长单调递增子序列的长度。将 dp 数组的所有元素初始化为 1，因为每个元素自身都可以看作是长度为 1 的单调递增子序列。

2. 从第 2 个元素开始，依次遍历序列中的每个元素，对于每个元素，再从第 1 个元素开始，依次遍历到该元素之前的所有元素，如果前面的元素小于当前元素，那么就尝试更新当前元素的 dp 值。即 dp[i] = max(dp[i], dp[j] + 1)。

3. 遍历完所有元素后，dp 数组中的最大值就是序列的最长单调递增子序列的长度。

算法分析： 由算法中核心部分的两次循环易知，算法的时间复杂性为 $O(n^2)$，空间复杂性为 $O(n)$

算法分析题 3-4 给定 n 种物品和一背包。物品 i 的重量是 wi，其价值为 vi，背包的容量为 c，容积为 d。问应如何选择装入背包中的物品，使得装入背包中物品的总价值最大？在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分的物品 i。试设计一个解此问题的动态规划算法，并分析算法的计算复杂性。

答：算法设计：本题在经典的 0-1 背包问题增加了一个新的约束，即背包的容积 d。这样，不仅要考虑背包的容量 c，还要考虑背包的容积 d。

$$max\sum_{i=1}^{n} vixi \qquad \begin{cases} \sum_{i=1}^{n} wixi \le c \\ \sum_{i=1}^{n} bixi \le d \end{cases} \qquad xi \in \{0,1\}, 1 \le i \le n$$

容易证明这个问题具有最优子结构性质，由此接着求出这个问题的状态转移方程，定义一个三维的动态规划数组 dp，其中 dp[i][j][k] 表示前 i 个物品，使用容量 j，容积 k 所能得到的最大价值。

- 如果不选择物品 i，则 dp[i][j][k] = dp[i-1][j][k]
- 如果选择物品 i，则 dp[i][j][k] = dp[i-1][j-w[i]][k-v[i]] + p[i]，其中 p[i] 是物品 i 的价值，前提是 j >= w[i] 且 k >= v[i]

所以，dp[i][j][k] = max(dp[i][j][k], dp[i-1][j-w[i-1]][k-v[i-1]] + p[i-1])

经过三重循环即可得到最终的答案。

算法分析： 由算法核心部分的三重循环可以知道，算法的时间复杂性为 O(n*c*d)，空间复杂度即为用于存储结果的 dp 数组，即 O(n*c*d),但是这里可以优化，因为 dp 数组只用到上一层的数据，因此只需要两层来回交替即可，即可优化为 O(c*d)。

算法实现题 3-3 石子合并问题。

问题描述：在一个圆形操场的四周摆着 n 堆石子。现要将石子有序次地合并成一堆。规定每次只能选相邻的两堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的得分。试设计一个算法，计算出 n 堆石子合并成一堆的最小得分和最大得分。

答：算法设计： 本问题与书中例题中的矩阵连乘问题相似，但是需要注意的是，本问题中的石子为圆形排列，而矩阵连乘问题是线性问题，因此，需要先将本问题转化为线性问题。考虑到首尾相接的情况，我们可以通过将原序列复制一份接在其后，再在这个新序列上进行常规的区间 dp。

1. 首先将原序列复制一份接在其后，得到新序列 stone。
2. 初始化两个二维数组 min_dp 和 max_dp，min_dp[i][j]表示将 stone 中第 i 堆到第 j 堆石子合并的最小得分，max_dp[i][j]表示将 stone 中第 i 堆到第 j 堆石子合并的最大得分。初始时，min_dp[i][i]和 max_dp[i][i]都等于 0。
3. 从长度为 2 的区间开始，逐渐增加区间长度，对于每个区间[i, j]，枚举其中的每一堆石子 k，将区间[i, j]分为[i, k]和[k+1, j]两部分，计算将这两部分合并的得分，更新 min_dp[i][j]和 max_dp[i][j]。状态转移方程如下：

   - min_dp[i][j] = min(min_dp[i][j], min_dp[i][k] + min_dp[k + 1][j] + total);
   - max_dp[i][j] = max(max_dp[i][j], max_dp[i][k] + max_dp[k + 1][j] + total);

4. 对于每个长度为 n 的区间，min_dp[i][i+n-1]和 max_dp[i][i+n-1]就分别是将这个区间内的石子合并的最小得分和最大得分。

5. 最后遍历 min_dp 和 max_dp 两个数组，分别获取最小值和最大值即为题目的答案。

算法分析：算法用到了三层嵌套循环遍历所有的区间和分割点，而循环体内的每次计算时间复杂度为 O(1)，所以总的时间复杂度是 $O(n^3)$。算法使用了两个二维数组 min_dp 和 max_dp 来存储状态，所以空间复杂度是 $O(n^2)$。

算法实现题 3-13 最大 k 乘积问题。

问题描述：设 I 是一个 n 位十进制整数。如果将 I 划分为 k 段，则可得到 k 个整数。这 k 个整数的乘积称为 I 的一个 k 乘积。试设计一个算法，对于给定的 I 和 k，求出 I 的最大 k 乘积。

答：算法设计：假设最大 k 乘积是将前 x 位划分为 k-1 段，再乘以最后的整数。若前 x 位的划分不是最优，则其乘积必然小于最优方法所得乘积 S_max,则其与最后的整数所得结果也并非最大 k 乘积，与前提矛盾。因此，将前 x 位划分为 k-1 段所得结果必为最大乘积。由此可知问题满足最优子结构性质。

我们定义 dp 和 num 两个数组，dp[i][j]为将整数 I 的前 i 位划分为 j 段得到的最大乘积，num[i][j]为整数 I 从第 i 位到第 j 位组成的数字。可知状态转移方程为：

dp[i][j] = max(dp[i][j], dp[k][j-1] * num[k+1][i])。当 j 为 1 时，dp[i][1]即为整数 I 的前 i 位组成的数字，因此 dp[i][1] = num[1][i]。答案就是 dp[n][k]。

因此，算法的主要流程即为计算 num 数组，接着初始化 dp 数组，最后进行状态转移方程的计算。进行状态转移方程计算时，第一重循环 i 从 1 遍历到 n，第二重循环 j 从 2 遍历到 k，第三重循环 m 从 1 遍历到 i。

算法分析：计算 num 数组需要用到两重循环，因此时间复杂度为 $O(n^2)$,初始化 dp 数组用到一重循环，时间复杂度为 O(n)，而进行状态转移方程的计算需要用到三重循环，时间复杂度为 $O(kn^2)$,因此，算法总的时间复杂度为 $O(kn^2)$。

算法实现题 3-14 最少费用购物问题。

问题描述：商店中每种商品都有标价。例如，一朵花的价格是 2 元，一个花瓶的价格是 5 元。为了吸引顾客，商店提供了一组优惠商品价。优惠商品是把一种或多种商品分成一组，并降价销售。例如，3 朵花的价格不是 6 元而是 5 元，2 个花瓶加 1 朵花的优惠价是 10 元。试设计一个算法，计算出某顾客所购商品

应付的最少费用。

答：算法设计：定义状态 dp[a][b][c][d][e]，表示购买 a 件第一种商品，b 件第二种商品，c 件第三种商品，d 件第四种商品，e 件第五种商品所需的最小费用。A[k],B[k],C[k],D[k],E[k]表示第 k 种优惠方案的商品组合，而 offer(m)是第 m 种优惠方案的价格。如果 dp[a][b][c][d][e]使用了第 m 种优惠方案，则找出最优子问题的递归表达式：dp[a][b][c][d][e]=min(dp[a][b][c][d][e], dp[a-A[m]][b-B[m]][c-C[m]][d-D[m]][e-E[m]]+offer(m))。可知本题具有最优子结构性质，可以用动态规划算法来实现。

算法的主要实现：

- 因为最多为五种商品，所以 dp 是一个五维数组，使用五重循环遍历所有状态，每层循环对应每种商品的数量。
- 首先尝试不使用优惠组合，直接购买单件商品，更新 dp[a][b][c][d][e] 的值。
- 然后尝试使用优惠组合，使用状态转移方程更新 dp[a][b][c][d][e] 的值。
- 最终 dp[a][b][c][d][e] 将存储购买 a 件第一种商品，b 件第二种商品，c 件第三种商品，d 件第四种商品，e 件第五种商品所需的最小费用。

算法分析：由于题目中规定最多购买 25 件商品，每种商品最多购买 5 件，所以状态的总数是 6^5 = 7776。对于每个状态，有两种方式更新最小费用：

- 不使用优惠组合，直接购买单件商品。这种情况下，时间复杂度是 O(B)，
- 使用优惠组合。这种情况下，时间复杂度是 O(S * B)，因为需要枚举每种优惠组合（S 种），并且对于每种优惠组合，需要计算购买组合中每种商品后的剩余数量，这需要枚举每种商品（B 种）。

因此，每个状态的时间复杂度是 O(B + S * B) = O(S * B)。综上，总的时间复杂度是 O(7776 * S * B)。

算法实现题 3-17 字符串比较问题。

问题描述：对于长度相同的两个字符串 A 和 B，其距离定义为相应位置字符距离之和。两个非空格字符的距离是它们的 ASCII 编码之差的绝对值。空格与空格的距离为 0，空格与其他字符的距离为一定值 k。

在一般情况下，字符串 A 和 B 的长度不一定相同。字符串 A 的扩展是在 A 种插入若干空格字符所产生的字符串。在字符串 A 和 B 的所有长度相同的扩展中，有一对距离最小的扩展，该距离称为字符串 A 和 B 的扩展距离。

对于给定的字符串 A 和 B，试设计一个算法，计算其扩展距离。

答：算法设计：对于给定的字符串 A 和 B，定义 dp[i][j]为 A 的前 i 个字符和 B 的前 j 个字符的最小扩展距离。可以通过以下三种方式来计算 dp[i][j]：

- 将 A 的第 i 个字符与 B 的第 j 个字符匹配，那么 dp[i][j] 可以由 dp[i - 1][j - 1] 转移得到，转移的费用为 A[i] 与 B[j] 的距离。
- 将 A 的第 i 个字符与空格匹配，那么 dp[i][j] 可以由 dp[i - 1][j] 转移得到，转移的费用为 k。
- 将 B 的第 j 个字符与空格匹配，那么 dp[i][j] 可以由 dp[i][j - 1] 转移得到，转移的费用为 k。

可以发现，dp[i][j] 的值完全取决于其子问题 dp[i - 1][j - 1]、dp[i - 1][j] 和 dp[i][j - 1] 的解。因此，这个问题具有最优子结构性质，即原问题能够使用动态规划来解决。

由以上的分析，易得问题的状态转移方程为：

dp[i][j] = min(dp[i - 1][j - 1] + dist(A[i], B[j]), dp[i - 1][j] + k, dp[i][j - 1] + k)

由此方程设计算法:

- 初始化字符串 A 和 B 的长度，分别记为 lenA 和 lenB。接着初始化一个二维数组 dp，大小为 (lenA + 1) * (lenB + 1)，所有元素初始值为 0。
- 使用双重循环，分别遍历字符串 A 和 B 的每个字符。
- 对于每个 dp[i][j]，根据状态转移方程来更新其值。
- 最终，dp[lenA][lenB] 的值即为字符串 A 和 B 的最小扩展距离。

算法分析： 算法中嵌入了一个双重循环，循环体内部进行状态更新的时间复杂度是 O(1),因此算法总的时间复杂度为 O(lenA*lenB),分别对应字符串 A 和 B 的长度，空间复杂度也为 O(lenA*lenB)。