算法分析第3次作业

小组编号：23

本次作业负责人:黄勖

1　算法分析题3-1 答案：

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。

算法输入:

- 一个包含n个元素的序列S：S[0], S[1], S[2], ..., S[n-1]。

算法输出:

- 序列S的最长单调递增子序列的长度（假设）。

算法描述:

1. 创建一个长度为 `n` 的数组 `dp` ，初始化所有元素为 1 ，表示以每个元素为结尾的最长递增子序列的长度。

2. 初始化一个变量 `max_length` ，用于追踪最长递增子序列的长度，初始值为 1 。

3. 对于 `i` 从 1 到 n-1 ：
   a. 对于 `j` 从 0 到 i-1 ：

   - 如果 S[i] > S[j]，则更新 `dp[i]` 为 max(dp[i], dp[j] + 1)。

   b. 更新 `max_length` 为 max(max_length, dp[i])。

4. 返回 `max_length` 作为最长递增子序列的长度，可以根据 `max_length` 在 `dp[i]` 的位置找到子序列的位置。

算法分析:

- 时间复杂度: 该算法的时间复杂度为 O(n^2) ，因为有两层嵌套的循环。
- 空间复杂度: 需要一个长度为 n 的 dp 数组，所以空间复杂度为 O(n) 。

这个算法使用动态规划来找到序列S的最长递增子序列的长度，通过比较每个元素与之前元素的大小关系，并维护一个dp数组来存储以每个元素结尾的最长递增子序列的长度。最后，返回dp数组中的最大值作为最长递增子序列的长度。

本题分工：小组共同讨论，黄勖编写

2 算法分析题3-4 答案:

3-4 给定 n 种物品和一背包。物品 i 的重量是 w_i，体积是 b_i，其价值为 v_i，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。

算法输入:

物品数目`n`

每个物品(重量，体积，价值) 即`(wi, bi, vi)` `i∈{1,2,……,n}`

背包容量`c`

背包容积`d`

算法输出:

序列`x1x2……xn`表示物品装入背包的方案，并使装入背包中物品的总价值最大，即:

$$max\sum_{i=1}^{n} x_i v_i$$

其中`xi∈{0, 1}`，0表示不装入，1表示装入，且有如下限制:

$$\begin{cases} \sum_{i=1}^{n} x_i b_i \leq d \\ \sum_{i=1}^{n} x_i w_i \leq c \end{cases}$$

算法描述:

`m[i][j][k]`：表示前 i 个物品装入容量为 j ，体积为 k 的背包中

边界条件: `i = 1`

$$m[1][j][k] = \begin{cases} v_i & w_i \leq j \ \&\& \ b_i \leq k \\ 0 & j < w_i \ || \ k < b_i \end{cases}$$

非边界条件: `i > 1 ^ i ∈ Z`

$$m[i][j][k] = \begin{cases} m[i-1][j][k] & w_i > j \ || \ b_i > k \\ max\{m[i-1][j][k], m[i-1][j-w_i][j-b_i]+v_i\} & w_i \leq j \ \&\& \ b_i \leq k \end{cases}$$

算法分析:

算法输出最优值`m[n][c][d]`，因此计算时间复杂度为

$$O(ncd)$$

本题分工：小组共同讨论，李嘉琪编写

3　算法实现题3-3 答案：

3-3　石子合并问题。

问题描述：在一个圆形操场的四周摆放着 n 堆石子。现要将石子有次序地合并成一堆。规定每次只能选相邻的 2 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，计算出将 n 堆石子合并成一堆的最小得分和最大得分。

算法设计：对于给定 n 堆石子，计算合并成一堆的最小得分和最大得分。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行是正整数 n（$1 \leqslant n \leqslant 100$），表示有 n 堆石子。第 2 行有 n 个数，分别表示每堆石子的个数。

结果输出：将计算结果输出到文件 output.txt。文件第 1 行的数是最小得分，第 2 行中的数是最大得分。

输入文件示例	输出文件示例
input.txt	output.txt
4	43
4 4 5 9	54

算法设计：

最小得分递推式：

$$dp_{\min}[i][j] = \min_{i \le k < j} (dp_{\min}[i][k] + dp_{\min}[k+1][j] + \sum_{l=i}^{j} \text{stones}[l]), \quad \text{for } i \le j, \text{cyclic}$$

最大得分递推式：

$$dp_{\max}[i][j] = \max_{i \le k < j} (dp_{\max}[i][k] + dp_{\max}[k+1][j] + \sum_{l=i}^{j} \text{stones}[l]), \quad \text{for } i \le j, \text{cyclic}$$

考虑到圆形操场的情况，"cyclic" 表示循环操作，确保首尾相连。

对于更新方式，用两层嵌套循环来计算 dp_min 和 dp_max 的值

算法描述：

1. 读取输入数据：`n`（石子堆数）和一个包含 n 个整数的数组 `stones`（每堆石子的个数）。
2. 初始化两个二维数组 `dp_min` 和 `dp_max`，均为大小为 n x n 的数组，表示最小得分和最大得分。
3. 对于 i 从 0 到 n-1 ：
    - 将 `dp_min[i][i]` 和 `dp_max[i][i]` 初始化为 `stones[i]`，因为每堆石子本身就是一个合并步骤。

4. 开始填充 dp_min 和 dp_max:

- 使用两层循环，外层循环遍历合并的堆数 `num_heaps` 从 2 到 n ：
  - 内层循环遍历每堆石子 i 从 0 到 n-1，对于每堆石子 i 计算 dp_min 和 dp_max:
    - 初始化 min_val 和 max_val 为正无穷大。
    - 使用一个循环 j 从 i 到 (i + num_heaps - 1) 取余 n，以处理圆形合并:
      - 计算 cost = dp_min[i][j] + dp_min[(j+1) % n][i+num_heaps-1]。
      - 如果 cost 小于 min_val，则更新 min_val 为 cost。
      - 计算 cost = dp_max[i][j] + dp_max[(j+1) % n][i+num_heaps-1]。
      - 如果 cost 大于 max_val，则更新 max_val 为 cost。
    - 更新 dp_min[i][i+num_heaps-1] 为 min_val，dp_max[i][i+num_heaps-1] 为 max_val。

5. 最小得分为 dp_min[0][n-1]，最大得分为 dp_max[0][n-1]。

6. 输出最小得分和最大得分。

算法分析:

- 时间复杂度：该算法中使用了三重循环，其中最内层循环的迭代次数为n，因此时间复杂度为O(n^3)。
- 空间复杂度：需要两个二维数组 dp_min 和 dp_max，每个数组的大小为n x n，因此空间复杂度为O(n^2)。

本题分工：小组共同讨论，黄勖编写

4 算法实现题3-13 答案:

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。

算法设计： 对于给定的 I 和 k，计算 I 的最大 k 乘积。

数据输入： 由文件 input.txt 提供输入数据。文件的第 1 行中有 2 个正整数 n 和 k。正整数 n 是序列的长度，正整数 k 是分割的段数。接下来的一行中是一个 n 位十进制整数($n \leq 10$)。

结果输出： 将计算结果输出到文件 output.txt。文件第 1 行中的数是计算出的最大 k 乘积。

输入文件示例	输出文件示例
input.txt	output.txt
2 1	15
15	

算法描述:

给定整数 `I = I(n,1)` 获得整数 `I(i,j)`, `I(i,j)` 表示从高位 `i` 到低位 `j` 表示的正整数:

$$I(i,j) = \frac{I}{10^{j-1}} \% 10^{n-i+1} \qquad (1)$$

设最优值`f(i,j)`表示从整数`I(i,1)`分为`j`段的最大j乘积，存在 `p∈[1,i-1]`

$$f(i,j) = min\{I(i,i-p) \times f(p,j-1)\} \qquad (2)$$

算法分析:

每次计算`f(i,j)`时，求(1)的时间复杂度为`O(1)`，求解(2)的时间复杂度为`O(n)`

算法求`f(n,k)`为最优解，因此时间复杂度为

$$O(n) * O(1) * O(nk) = O(n^2 k)$$

本题分工：小组共同讨论，李嘉琪编写

5 算法实现题3-14 答案:

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。

算法设计：对于给定欲购商品的价格和数量，以及优惠商品价，计算所购商品应付的最少费用。

数据输入：由文件 input.txt 提供欲购商品数据。文件的第 1 行中有 1 个整数 B（$0 \leqslant B \leqslant 5$），表示所购商品种类数。在接下来的 B 行中，每行有 3 个数 C、K 和 P。C 表示商品的编码（每种商品有唯一编码），$1 \leqslant C \leqslant 999$；$K$ 表示购买该种商品总数，$1 \leqslant K \leqslant 5$；$P$ 是该种商品的正常单价（每件商品的价格），$1 \leqslant P \leqslant 999$。注意，一次最多可购买 $5 \times 5 = 25$ 件商品。

由文件 offer.txt 提供优惠商品价数据。文件的第 1 行中有 1 个整数 S（$0 \leqslant S \leqslant 99$），表示共有 S 种优惠商品组合。接下来的 S 行，每行的第 1 个数描述优惠商品组合中商品的种类数 j。接着是 j 个数字对（C、K），其中 C 是商品编码，$1 \leqslant C \leqslant 999$；$K$ 表示该种商品在此组合中的数量，$1 \leqslant K \leqslant 5$。每行最后一个数字 P（$1 \leqslant P \leqslant 9999$）表示此商品组合的优惠价。

结果输出：将计算出的所购商品应付的最少费用输出到文件 output.txt。

输入文件示例		输出文件示例
input.txt	offer.txt	output.txt
2	2	14
7 3 2	1 7 3 5	
8 2 5	2 7 1 8 2 10	

算法描述:

1. 创建一个五维数组 `dp[a][b][c][d][e]` 表示选择 a 件第 1 种商品、b 件第 2 种商品、c 件第 3 种商品、d 件第 4 种商品、e 件第 5 种商品情况下的最少费用。
2. 定义 `priceA`、`priceB`、`priceC`、`priceD`、`priceE` 分别表示不同种商品的单价。

3. 对于每一种组合 i： a. 定义 `A[i]`、`B[i]`、`C[i]`、`D[i]`、`E[i]` 表示第 i 种组合下不同种类商品需要的数量，`price[i]` 则表示第 i 种组合的花费费用。

4. 初始化 `dp[a][b][c][d][e]` 为不考虑组合优惠时的花费，即 `a * priceA + b * priceB + c * priceC + d * priceD + e * priceE`。

5. 对于每一种组合 i，考虑应用组合优惠：

   - 更新 `dp[a][b][c][d][e]` 为 `dp[a - A[i]][b - B[i]][c - C[i]][d - D[i]][e - E[i]] + price[i]`，如果 `a >= A[i]` 且 `b >= B[i]` 且 `c >= C[i]` 且 `d >= D[i]` 且 `e >= E[i]`。

6. 在 s ∈ S、a ∈ K a 、b ∈ K b 、c ∈ K c 、d ∈ K d 、e ∈ K e 情况下，计算所有可能的 `dp[a][b][c][d][e]`。

7. 最终的答案即为 `dp[Ka][Kb][Kc][Kd][Ke]`。

算法分析:

- 时间复杂度：O(S * K^5)，其中 S 表示组合数，K 表示购买每种商品的最大数量。
- 空间复杂度：O(K^5)，需要一个五维数组来存储每种状态下的最少费用。

本题分工：小组共同讨论，黄勖编写

6 算法实现题3-17 答案:

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 中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。

算法设计：对于给定的字符串 A 和 B，计算其扩展距离。

数据输入：由文件 input.txt 给出输入数据。第 1 行是字符串 A，第 2 行是字符串 B，第 3 行是空格与其他字符的距离定值 k。

结果输出：将计算出的字符串 A 和 B 的扩展距离输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
cmc	10
snmn	
2	

算法描述:

`exd(i,j)` 表示 `A` 的子串 `Ai=a1a2……ai` 和 `B` 的子串 `Bj=b1b2……bj` 的扩展距离

`d(a,b)`表示字符`a`与字符`b`的距离

分析可得如下3种情况:

当`[ai]`对应`[ ]`:

$$exd(i,j)=exd(i-1,j)+k$$

当`[bj]`对应`[ ]`:

$$exd(i,j)=exd(i,j-1)+k$$

当`[ai]`对应`[bj]`:

$$exd(i,j)=exd(i-1,j-1)+d(ai,bj)$$

`exd(i,j)`应该为上述三种情况的最小值，即:

$$exd(i,j)=min\{exd(i-1,j)+k,exd(i,j-1)+k,exd(i-1,j-1)+d(ai,bj)\}$$

算法分析:

`A`的长度为`m`，`B`的长度为`n`，按照上述式子递推计算`exd(m,n)`为所求

计算每个`exd(i,j)`为`O(1)`，故时间复杂度为:

$$O(mn)$$

本题分工：小组共同讨论，李嘉琪编写