# 算法分析第三次作业

## 算法分析题

### 3-1 题答案：

**答：**

输入包含 n 个元素的序列 S：S[0],S[1],S[2],…,S[n−1]。

输出序列 S 的最长单调递增子序列的长度。

1.创建一个长度为 n 的数组 A，将所有元素初始化为 1，表示以每个元素为结尾的最长递增子序列的初始长度为 1。

2. 初始化变量:maxlen=1，追踪最长递增子序列的长度。

3. 外层循环，i 从 1 到 n−1：内层循环，j 从 0 到 i−1。

若 S[i]>S[j]，则更新

A[i]=max(A[i],A[j]+1)。

更新 maxlen=max(maxlen,A[i])。

4. 返回 maxlen 作为最长递增子序列的长度，根据 maxLen 在 A[i]中的位置找到子序列的位置。

时间复杂度：含两层嵌套循环，故时间复杂度为 $O(n^2)$。

### 3-4 题答案：

**答：**

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。

dp[i][j][k] = max(dp[i][j][k], dp[i-1][j-w[i-1]][k-v[i-1]]

+ p[i-1])

核心三重循环可知，O(ncd)

# 算法实现题

## 3-3 题答案：

**答：**

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：

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

cyclic 表示循环操作，确保首尾相连。用两层嵌套循环来计算 dp_min、dp_max。

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）。

初始化两个二维数组 dp_min 和 dp_max ，均为大小为 nxn 的数组，表示最小得分和最大得分。

dp_max [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，dp_min[i][i]和 dp_max[i][i]都等于 0。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]，分为[i,k]和[k+1,j]，计算两部分合并的得分，更新 min 和 max。(递推式)

min[i][i+n-1]和 max[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。遍历 dp_min 和 dp_max 两个数组，得到最小值和最大值。

## 3-13 题答案：

**答：**

定义数组 dp[i][j]表示将 I 的前 i 位划分为 j 段的最大乘积，num[i][j]表示 I 从第 i 位到第 j 位组成的数字。

状态转移方程：dp[i][j]=max(dp[i][j],dp[k][j-1] * num[k + 1][i])。当 j = 1 时，dp[i][1] = num[1][i]。最终答案是 dp[n][k]。

先计算 num 数组，再初始化 dp 数组，最后按状态转移方程计算。i 从 1 到 n，j 从 2 到 k，m 从 1 到 i 循环。

计算 num 数组时间复杂度： $O(n^2)$初始化 dp 数组时间复杂度为 $O(n)$，状态转移方程计算时间复杂度为 $O(kn^2)$，所以算法总复杂度为 $O(kn^2)$。

## 3-14 题答案：

## 答：

创建一个五维数组 dp[a][b][c][d][e],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。priceA、priceB、priceC、priceD、priceE 分别表示不同种商品的单价。对于每一种组合 I,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。

初始化 dp[a][b][c][d][e]为不考虑组合优惠时的花费:a*priceA + b*priceB + c*priceC + d*priced + e*priceE 。

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]。

在 s∈S、a∈Ka、b∈Kb、c∈Kc、d∈Kd、e∈Ke 情况下，计算所有可能的 dp[a][b][c][d][e]。最终的答案即为 dp[Ka][Kb][Kc][Kd][Ke] 。

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。

## 3-17 题答案：

## 答：

**数据输入**

从文件 input.txt 读取数据，第一行是字符串 A，第二行是字符串 B，第三行是空格与其他字符的距离定值 k。

**结果输出**

将字符串 A 和 B 的扩展距离输出到文件 output.txt　。

**算法描述**

定义 dq(i,j)表示 A 的子串 Ai=a1a2…ai 与 B 的子串 Bj=b1b2…bj 的扩展距离，d(a,b)表示字符 a 与 b 的距离。

当 ai 对应空格：dq(i,j)=dq(i−1,j)+k。

当 bj 对应空格：dq(i,j)=dq(i,j−1)+k。

当 ai 对应 bj：dq(i,j)=dq(i−1,j−1)+d(ai,bj)。

取上述三种情况的最小值：dq(i,j)=min{dq(i−1,j)+k, dq(i,j−1)+k, dq(i−1,j−1)+d(ai,bj)}。

**算法分析**

设 A 的长度为 m，B 的长度为 n，计算每个 dq(i,j)的时间复杂度为 O(1)，故算法时间复杂度为 O(mn)。