HW 5

Q1 限制增性质的序列数目

1. 用dp[j][M]表示长度为 j ，最大值为M 的序列数目。
2. 初始化：dp[1][1]=1。
3. 状态转移：
   i. j从2到n，M从1到j。
   ii. 如果aj <= M，那么最大值为M，贡献 M * dp[j-1][M]个。
   iii. 如果aj=M+1，那么最大值为M+1，贡献dp[j-1][M-1]个。
   iv. 综合，dp[j][M]=M * dp[j-1][M]+dp[j-1][M-1]。
4. 最终结果：$\Sigma_{M=1}^{n} dp[n][M]$
5. 递归式：

$$dp[j][M] = \begin{cases} 0 & \text{若 } j = 0 \text{或} M > j \\ 1 & \text{若 } j = 1, M = 1 \\ M \times dp[j-1][M] + dp[j-1][M-1] & \text{若 } M > 1 \end{cases}$$

6. 时间复杂度：填充一个n * n的dp数组，每个状态的时间复杂度为O(1)，总时间复杂度为O(n^2)。

Q2 3-划分问题

1. 计算总和并检查：$Total = \Sigma_{i=1}^{n} S_i$，如果Total不是3的倍数，那么无法划分，返回false，如果Total是3的倍数，那 $Target = Total/3$。
2. dp[i][s1][s2]表示前i个元素中，分配到子集1的和为s1，分配到子集2的和为s2的方案数，子集3自动由 $Total - s1 - s2$ 确定。
3. 初始化：dp[0][0][0]=1。
4. 状态转移：
   i. Si加入子集1：若s1+Si<=Target，那么 dp[i][s1+Si][s2]+=dp[i-1][s1][s2]。
   ii. Si加入子集2：若s2+Si<=Target，那么 dp[i][s1][s2+Si]+=dp[i-1][s1][s2]。
   iii. Si加入子集3：若Total-s1-s2-Si>=0，那么 dp[i][s1][s2]+=dp[i-1][s1][s2]。
5. 结果计算：方案数为 dp[n][Target][Target]。
6. 递归式：

$$dp[i][s1][s2] = \begin{cases} dp[i-1][s1-Si][s2] + dp[i-1][s1][s2-Si] + dp[i-1][s1][s2] & \text{若 } s1 >= Si \text{且} s2 >= Si \\ dp[i-1][s1-Si][s2] + dp[i-1][s1][s2] & \text{若 } s1 >= Si \text{且} s2 < Si \\ ]dp[i-1][s1][s2-Si] + dp[i-1][s1][s2] & \text{若 } s1 < Si \text{且} s2 >= Si \\ ]dp[i-1][s1][s2] & \text{若 } s1 < Si \text{且} s2 < Si \end{cases}$$

7. 时间复杂度：状态数为 n * Target * Target，每个状态的时间复杂度为O(1)，总时间复杂度为O(n * Target^2)，由于Target=Total/3，所以时间复杂度为O(n*Total^2)。
8. 空间复杂度：dp数组大小为n * Target * Target，所以空间复杂度为O(n * Target^2)。

Q3 具有最大和的连续子数组

(a)基于分治思想

1. 分解：左部分A[low ... mid] 右部分A[mid+1 ... high]。
2. 递归：求解左右两个部分的最大子数组和，记为leftMax，rightMax。

3. 合并：计算跨越mid的最大子数组和，记为crossMax：
   i. 从mid向左遍历，计算累加和，记录左边部分的最大累加和crossleftMax。
   ii. 从mid+1向右遍历，计算累加和，记录右边部分的最大累加和crossrightMax。
   iii. crossMax = crossleftMax + crossrightMax。
4. 返回结果：取max(leftMax，rightMax和crossMax)。
5. 递归式：

$$\mathrm{MaxSubArray}(A, \mathrm{low}, \mathrm{high}) = \begin{cases} A[\mathrm{low}] & \text{若 } \mathrm{low} = \mathrm{high} \\ \max\begin{pmatrix} \mathrm{MaxSubArray}(A, \mathrm{low}, \mathrm{mid}), \\ \mathrm{MaxSubArray}(A, \mathrm{mid}+1, \mathrm{high}), \\ \mathrm{MaxCrossingSubArray}(A, \mathrm{low}, \mathrm{mid}, \mathrm{high}) \end{pmatrix} & \text{否则} \end{cases}$$

6. 时间复杂度：T(n)=2T(n/2)+O(n)，根据主定理，递推关系的解为O(nlogn)，即时间复度为O(nlogn)。

(b)基于动态规划思想

1. dp[i]表示以第i个元素结尾的子数组的最大和，first[i]表示最大和子数组的首元素。
2. 初始化：dp[1]=A[1]，first[1]=1。
3. 状态转移方程：

$$dp[i] = \begin{cases} A[i] & \text{若 } dp[i-1] < 0 \\ dp[i-1] + A[i] & \text{若} dp[i-1] >= 0 \end{cases}$$

$$first[i] = \begin{cases} i & \text{若 } dp[i-1] < 0 \\ first[i-1] & \text{若} dp[i-1] >= 0 \end{cases}$$

4. 返回结果：返回dp[n]，first[n]表示最大和子数组的首元素。
5. 边界条件：当n=1时，dp[1]=A[1]，first[1]=1；对于第i个元素，如果dp[i-1]<0，那么开始一个新的子数组，first[i]=i。

(c)具有最大和的子矩阵

1. 遍历行区间：top从1到m，对于每个top，bottom从top到m。
2. 计算压缩的列数组：$temp[j] = \Sigma_{i=top}^{bottom} A[i][j], \quad \forall j \in [1, n]$。
3. 应用一维最大子数组和算法：
   i. 对于每个 temp 数组，使用 Kadane 算法（动态规划方法）找到其最大子数组和，并记录对应的top，bottom，j和first[j]。
   ii. 更新全局的最大子数组和，即最大子矩阵和，同时更新top，bottom，j和first[j]。
4. 时间复杂度：
   i. 行区间：总共O(m^2)可能行区间
   ii. 列数组最大和的子数组：根据前面的算法，时间复杂度为O(n)。
   iii. 总时间复杂度为O(m^2*n)。

Q4 叠叠乐

(a)书叠层

1. 长度降序排序：将所有书按照长度(ai)降序排序，如果长度相同，按照宽度(bi)降序排序。
2. 宽度序列 B = [b1, b2, ..., bn]。
3. 在宽度序列中寻找最长严格递减子序列：
   i. dp[i]表示以第i本书为顶的最大叠层数；
   ii. 递归式

$$dp[i] = \begin{cases} 1 & 若\ i = 1 \\ max\{dp[j] + 1 | 1 <= j < i, bj > bi\} & 若\ i > 1 \end{cases}$$

iii. 返回结果：$\max_{i=1}^{n} dp[i]$。

4. 时间复杂度：
   i. 排序：O(nlogn)。
   ii. 动态规划：O(n^2)。
   iii. 总时间复杂度为O(nlogn+n^2)=O(n^2)。

(b)积木叠层

1. 生成3种可能的方向：
   i. 底面ai*bi，高ci
   ii. 底面ai*ci，高bi
   iii. 底面bi*ci，高ai
   iv. 为了简化比较，确保每个方向满足$长 >= 宽$
2. 将每个积木的3个方向的对应的长、宽、高分别加入长、宽、高数组。
3. 仿照(a)的过程，只需要将递归式修改为：

$$dp[i] = \begin{cases} height_1 & 若\ i = 1 \\ max\{dp[j] + height_j | 1 <= j < i, bj > bi\} & 若\ i > 1 \end{cases}$$

4. 最后遍历dp[3n] 获取最大值即可。