动态规划基础

1190201926 曲喆平

母牛的故事

- 有一头母牛，它每年年初生一头小母牛。每头小母牛从第四个年头开始，每年年初也生一头小母牛。问在第n年的时候，共有多少头母牛?

母牛的故事

- 设第n年有a_n头母牛。
- n=1时，只有一头老母牛，a_1=1。
- 2≤n≤4时，每年大母牛都会生一头小母牛，但小母牛无法生育，所以a_n=n
- n＞4时，第n年较前一年而言，增加的小母牛数等于此时所有的大母牛数。而第n年的大母牛总数等于第n-3年的总母牛数，因为在n-3年之后出生的牛并没有生育能力。因此$a_n=a_{n-1}+a_{n-3}$

动态规划（Dynamic Programming）

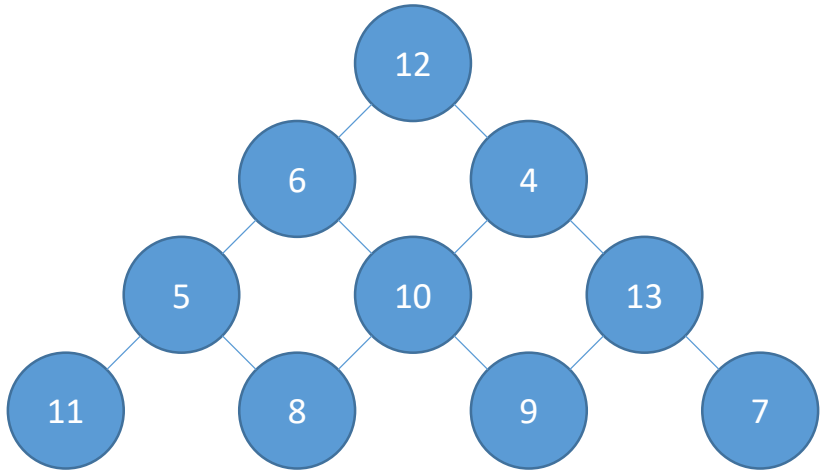
- 动态规划不是一种特定的算法，更是一种利用递推去解决问题的思想。
- 利用动态规划，我们会从一个或多个初始状态出发，根据某个递推公式，让每一个子问题的解都由之前子问题的解推出。
- 动态规划的思想是“空间换时间”，尽量避免解决重复的问题。

动态规划（Dynamic Programming）

- 在进行动态规划之前，要求问题具有三个特征：
- 1.**最优子结构**：指问题的最优解一定是由子问题的最优解推导过来的。也就是说，如果全局最优，一定有局部最优。
- 2.**重复子问题**：不同的决策可能会达到某个相同的状态。正是因为我们在动态规划的过程中记录下这些状态，才能减少重复计算，降低复杂度。
- 3.**无后效性**：有两层含义，第一层含义是，在利用某个状态去推导后面的状态时，我们不去关心这个状态的过程状态，也就是它是怎么被推导来的；第二层含义是，某个状态一旦确定，就不会受未来某个决策的影响而发生改变。

数字三角形

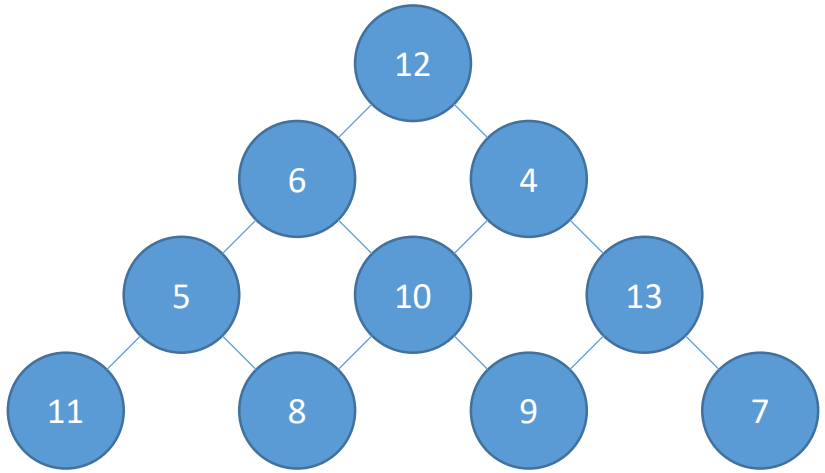
- 给定一个数字三角形，求一条从三角形最高点到底部任意处的路径，使沿途经过的数字的和最大。每一步可以向左下走也可以向右下走。



- 三角形的高度≤1000

数字三角形

- 设数字三角形上第i行第j个数的值为$a_{i,j}$，从三角形最高点到(i,j)点的路径最大和为f[i][j]。则有f[i][j] = max(f[i-1][j-1] , f[i-1][j]) + a[i][j]。



数字三角形（变式一）

- 要求必须经过三角形上的某个点（？）

数字三角形（变式一）

- 要求必须经过三角形上的某个点（？）
- 两种办法：
- 第一种：先从求这个节点往下走的最大值，再由这个点向上转移或者把同层其他节点打个不可访问标记。
- 第二种办法，给这个节点加上一个特别大的数，最后答案再减去它，这样就能保证一定会经过这个点（？）

数字三角形（变式二）

- 其它条件不变，问题改为使沿途经过的数字之和对p（p≤100）取模后最大（？）

数字三角形（变式二）

- 其它条件不变，问题改为使沿途经过的数字之和对p（p≤100）取模后最大（？）
- 显然，如果直接按照原来的方式进行，无法保证每次决策都从子问题的最优解中转移过来，即不满足最优子结构。
- 我们可以加一维状态：
- 令f[i][j][k]表示从最高点到(i,j)点的路径和对p取模后能否为k。
- 方程：f[i][j][k] = f[i-1][j-1][(k-a[i][j]+p)%p] || f[i-1][j][(k-a[i][j]+p)%p]

最长上升子序列（LIS）

- 给你一个包含n个正整数的序列A，让你找到一个最长的子序列，满足序列中的每一项都严格大于前一项。
- 输出这个最长上升子序列的长度。
- $n \leq 2000$ 。

最长上升子序列（LIS）

- 令f[i]表示以Ai为结尾元素的最长上升子序列的长度。我们只需要考虑f[i]是从之前哪个状态转移过来的就可以了。
- 转移方程：f[i] = max(f[j]) + 1 (j<i 且 Aj<Ai)
- 注意：最后的答案是所有f[i]的最大值。

最长上升子序列（LIS）

- 令f[i]表示以Ai为结尾元素的最长上升子序列的长度。我们只需要考虑f[i]是从之前哪个状态转移过来的就可以了。
- 转移方程：f[i] = max(f[j]) + 1 (j<i 且 Aj<Ai)
- 注意：最后的答案是所有f[i]的最大值。
- 如果题目要求输出一个最长上升子序列该怎么办呢?

最长上升子序列（LIS）

- 令f[i]表示以Ai为结尾元素的最长上升子序列的长度。我们只需要考虑f[i]是从之前哪个状态转移过来的就可以了。
- 转移方程：f[i] = max(f[j]) + 1 (j<i 且 Aj<Ai)
- 注意：最后的答案是所有f[i]的最大值。
- 如果题目要求输出一个最长上升子序列该怎么办呢?
- 每当用f[j]更新f[i]时，我们就用一个状态last[i]=j记录下f[i]是由之前的f[j]更新过来的。这样我们可以从最大值的点开始，沿着last[]一步一步找出状态更新的整个路径，逆向输出就是一个最长上升子序列。

最长上升子序列（LIS）

- 这个问题还有一个二分的解法，利用二分可以将时间复杂度从$O(n^2)$降低到O(nlogn)。
- 令f[i]表示长度为i的上升子序列中最后一个数的最小值。
- 可以发现f[i]是单调递增的，转移的时候直接二分就可以了。

最长公共子序列（LCS）

- 给你两个长度分别为n和m的序列A和B，$n,m \leq 2000$。
- 找到一个最长公共子序列，使它在A和B中都出现过。
- 问这个最长公共子序列的长度。

最长公共子序列（LCS）

- 给你两个长度分别为n和m的序列A和B，$n,m \le 2000$。
- 找到一个最长公共子序列，使它在A和B中都出现过。
- 问这个最长公共子序列的长度。
- 令f[i][j]表示A的前i个数和B的前j个数匹配所得的最长公共子序列的长度。
- 转移的时候，如果Ai和Bj匹配，就让f[i][j] = f[i-1][j-1] +1；如果不匹配，就保留之前计算的结果，即f[i][j] = max(f[i-1][j] , f[i][j-1])。
- 转移方程： f[i][j] = max(f[i-1][j] , f[i][j-1] , f[i-1][j-1] + (a[i]==b[j]))

背包问题

- 有n件物品和一个容量为C的背包。第i件物品的重量是w[i]，价值是v[i]。求背包能容纳的最大价值和。
- n,C,w[i],v[i] ≤ 2000。

背包问题

- 有n件物品和一个容量为C的背包。第i件物品的重量是w[i]，价值是v[i]。求背包能容纳的最大价值和。
- n,C,w[i],v[i] ≤ 2000。
- f[i][c]表示前i件物品放入一个容量为c的背包可以获得的最大价值。
- 转移方程： f[i][c] = max(f[i-1][c] , f[i-1][c-w[i]] + v[i])。

背包问题

- 有n件物品和一个容量为C的背包。第i件物品的重量是w[i]，价值是v[i]。求背包能容纳的最大价值和。
- n,C,w[i],v[i] ≤ 2000。
- f[i][c]表示前i件物品放入一个容量为c的背包可以获得的最大价值。
- 转移方程： f[i][c] = max(f[i-1][c] , f[i-1][c-w[i]] + v[i])。
- 如果不进行优化，储存所有状态需要n×C的空间。
- 前i个物品的状态只由前i-1个状态转移过来，储存的时候可以省掉前面的一维，只保留一个背包容量的状态。
- 注意，循环的时候，需要的递推顺序为背包容量从最大到0。

多重背包

- 有n种物品和一个容量为C的背包。第i件物品的重量是w[i]，价值是v[i]，数量为a[i]。求背包能容纳的最大价值和。
- n,C,w[i],v[i],a[i] ≤ 2000。

多重背包

- 有n种物品和一个容量为C的背包。第i件物品的重量是w[i]，价值是v[i]，数量为a[i]。求背包能容纳的最大价值和。
- n,C,w[i],v[i],a[i] ≤ 2000。
- 如果全都拆成一个物品的情况来做，相当于最多有2000×2000个物品。
- 二进制法：按照二进制拆分物品。比方说物品i有13个，就把它拆成1、2、4、6四个物品。
- 这样复杂度就降为O(C·Σloga[i])。

乘积最大

- 设有一个长度为n的数字串，要求选手使用k个乘号将它分成k+1个部分，找出一种分法，使得这k+1个部分的乘积最大。
- n≤40，k<n，保证最大乘积在long long范围内。

乘积最大

- 设有一个长度为n的数字串，要求选手使用k个乘号将它分成k+1个部分，找出一种分法，使得这k+1个部分的乘积最大。
- n≤40，k<n，保证最大乘积在long long范围内。
- 令f[i][k]表示考虑到第i个数字，一共使用了k个乘号，所能得到的最大乘积。
- 转移：f[i][k] = max(f[p][k-1] × c[p+1][i])，其中c[p+1][i]表示从p+1位置到i位置的数字。
- 初始条件：f[i][0] = c[1][i]。

[NOIP2010]乌龟棋

- 乌龟棋的棋盘是一行N个格子，每个格子上一个分数（非负整数）。棋盘第1格是唯一的起点，第N格是终点，游戏要求玩家控制一个乌龟棋子从起点出发走到终点。
- 乌龟棋中M张爬行卡片，分成4种不同的类型（M张卡片中不一定包含所有4种类型的卡片），每种类型的卡片上分别标有1,2,3,4四个数字之一，表示使用这种卡片后，乌龟棋子将向前爬行相应的格子数。游戏中，玩家每次需要从所有的爬行卡片中选择一张之前没有使用过的爬行卡片，控制乌龟棋子前进相应的格子数，每张卡片只能使用一次。
- 游戏中，乌龟棋子自动获得起点格子的分数，并且在后续的爬行中每到达一个格子，就得到该格子相应的分数。玩家最终游戏得分就是乌龟棋子从起点到终点过程中到过的所有格子的分数总和。
- 很明显，用不同的爬行卡片使用顺序会使得最终游戏的得分不同，小明想要找到一种卡片使用顺序使得最终游戏得分最多。
- 现在，告诉你棋盘上每个格子的分数和所有的爬行卡片，你能告诉小明，他最多能得到多少分吗?
- 输入数据保证到达终点时刚好用光M张爬行卡片。N<=350，M<=120，四种爬行卡片，每张的数量都不会超过40。

[NOIP2010]乌龟棋

- 首先第一维状态是当前跳到的位置。
- 影响后续决策的因素：每种类型的牌剩余张数。
- 用f[i][a][b][c][d]表示当前在位置i，已经使用了a张1类型卡片，b张2类型卡片，c张3类型卡片，d张4类型卡片，能获得的最大的分数。
- 转移：假设最后一步要用1张2类型卡片，f[i][a][b][c][d]=f[i-2][a][b-1][c][d]+value[i]。
- 对于其它类型卡片同理，在四种决策中选择得分最多的那种。

[NOIP2010]乌龟棋

- 首先第一维状态是当前跳到的位置。
- 影响后续决策的因素：每种类型的牌剩余张数。
- 用f[i][a][b][c][d]表示当前在位置i，已经使用了a张1类型卡片，b张2类型卡片，c张3类型卡片，d张4类型卡片，能获得的最大的分数。
- 转移：假设最后一步要用1张2类型卡片，f[i][a][b][c][d]=f[i-2][a][b-1][c][d]+value[i]。
- 对于其它类型卡片同理，在四种决策中选择得分最多的那种。
- 但是这样做的话时空复杂度为350×40×40×40×40=896,000,000，这是无法接受的。
- 我们可以发现，当a,b,c,d确定的时候，实际上i也确定了。因此我们没必要记录i，只需要记录a,b,c,d即可。
- 即f[a][b][c][d] = max(f[a-1][b][c][d] , f[a][b-1][c][d] , f[a][b][c-1][d] , f[a][b][c][d-1]) + val[a*1+b*2+c*3+d*4]。

吃饭

- 一个人准备用勺子和筷子依次吃n道菜。勺子和筷子可以交替使用，但是每道菜只能用一种餐具。在开饭前他拿的是筷子。用勺子和筷子吃第i道菜的时间分别为ai和bi。吃第i道菜之前筷子和勺子交换的时间为ci。
  现在请你告诉这个人，按他的计划依次吃完这n道菜，最少需要多长时间。
- $1 \le n \le 100$
- $1 \le ai,bi,ci \le 10000$

吃饭

- 一个人准备用勺子和筷子依次吃n道菜。勺子和筷子可以交替使用，但是每道菜只能用一种餐具。在开饭前他拿的是筷子。用勺子和筷子吃第i道菜的时间分别为ai和bi。吃第i道菜之前筷子和勺子交换的时间为ci。
  现在请你告诉这个人，按他的计划依次吃完这n道菜，最少需要多长时间。
- f[i][0/1]表示吃前i个菜，手里拿的是勺子/筷子的最少时间。
- f[i][0] = min(f[i-1][0] , f[i-1][1] + c[i]) + a[i] ;
- f[i][1] = min(f[i-1][1] , f[i-1][0] + c[i]) + b[i] ;

双塔

- Mr. F有N块水晶，每块水晶有一个高度，他想用这N块水晶搭建两座有同样高度的塔，使他们成为一座双塔，Mr. F可以从这N块水晶中任取M（1≤M≤N）块来搭建。但是他不知道能否使两座塔有同样的高度，也不知道如果能搭建成一座双塔，这座双塔的最大高度是多少。所以他来请你帮忙。
- 给定水晶的数量N（1≤N≤100）和每块水晶的高度Hi（N块水晶高度的总和不超过2000），你的任务是判断Mr. F能否用这些水晶搭建成一座双塔（两座塔有同样的高度），如果能，则输出所能搭建的双塔的最大高度，否则输出“Impossible”。

双塔

- 如果开两维状态记录两个塔的高度，显然状态数量太多，再枚举一个n，就会超时。考虑削减状态。
- 当两个塔的高度差一定的时候，比如1和3、4和6，前面那一组高度小的就没用了。
- 令f[i][j]表示考虑前i块砖，两个塔的高度差为j时，矮的塔的高度。
- 对于每一块砖，四种情况：1、不取；2、取完放到高的；3、取完放到矮的，但是矮的依旧矮；4、取完放到矮的，矮的变成高的。

子串

- 有两个仅包含小写英文字母的字符串 A 和 B。现在要从字符串 A 中取出 k 个互不重叠的非空子串，然后把这 k 个子串按照其在字符串 A 中出现的顺序依次连接起来得到一个新的字符串，请问有多少种方案可以使得这个新串与字符串 B 相等？（答案对1,000,000,007取模）注意：子串取出的位置不同也认为是不同的方案。
- 字符串A长度为n，字符串B长度为m。
- $1 \le n \le 1000$ ， $1 \le m \le 200$ ， $1 \le k \le m$ 。
- 内存限制128mb。

子串

- 取状态f[i][j][p][0/1]表示当前考虑字符串A的1~i位置，用了p个不重叠非空子串，匹配了B的1~j位置。如果数组最后一位是0，表示A[i]这个字母并没有用；如果最后一位是1，表示A[i]这个字母作为子串的一部分。
- f[i][j][p][0] = f[i-1][j][p][0] + f[i-1][j][p][1]
- f[i][j][p][1] = f[i-1][j-1][p][1] + f[i-1][j-1][p-1][0] + f[i-1][j-1][p-1][1]

(条件是A[i]==B[j])

- 空间需求量大，需要开滚动数组。

K车问题

- 在n×m棋盘上放置K个车，使没有一个车同时被两个车攻击，求方案数。
- n,m,k ≤ 50。

K车问题

- f[i][j][k][p]四维状态，需要滚动数组：
- i：行数。 j：只有一个车且安全的列数。
  k：只有一个车且不安全的列数。 p：有两个车的列数。
- “安全”指这个车的同一行没有其它车。
- f[i][j][k][p] = f[i-1][j][k][p]
  + f[i-1][j-1][k][p] * (m-j-k-p+1)
  + f[i-1][j+1][k][p-1] * (j+1)
  + f[i-1][j][k-2][p] * (m-j-k-p+2) * (m-j-k-p+1) / 2;

结语

- 动态规划的题往往不会直接说这个题是用动态规划可以做的，需要你自己想明白到底能不能用，有没有后效性，满不满足最优子结构。
- 下午一共九道题。我是找到了去年的题，然后把比赛期间没有人A掉的题（或者只有郭主席和Owen才能A的题）删掉了，又添了几道更简单的。
- 所以难度从整体上来说很令人舒服，同学们不要轻易放弃。在遇到一道dp题的时候，要好好想想该怎么列状态、循环顺序是怎样的、转移方程中有没有漏掉的情况。
- “不要作弊，如果你很菜，作弊也进不了校队。你可以选择签到走人。”