

动画与数字艺术学院 20数媒网络 潘东逸杰

区间DP & 状态压缩DP

REVIEW

- ▶ 什么是动态规划 (Dynamic Programming) ?
- ▶ 学术定义：动态规划是一种通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。
- ▶ 而在ACM算法竞赛中，很多时候大家可以把之理解成一种结合了贪心算法的递推
- ▶ 甚至很多时候，我们也默认把一些不带贪心算法（纯递推）的题目归类到动态规划当中

REVIEW

- ▶ 回顾一个经典问题LCS（最长公共子序列）的状态转移方程

$$f(i, j) = \begin{cases} f(i-1, j-1) + 1 & A_i = B_j \\ \max(f(i-1, j), f(i, j-1)) & A_i \neq B_j \end{cases}$$

当上面一个等式成立时，这就是一个递推；当下面一个等式成立时，这就是一个带选择（你可以理解成贪心算法）的递归

WHY IMPORTANT

- ▶ 为什么花这么多篇幅讲这么多的动态规划?
- ▶ 从老队员的角度来看: 解决DP问题的能力与你在ACM上取得的成绩成正相关
- ▶ DP题可体现出 分析问题 化繁为简 逐步推理 数学功底等 算法竞赛选手的必备能力
- ▶ DP在非常多的比赛中会成为铜牌题/银牌题的坎
- ▶ 大厂面试 非常喜欢 考察DP能力

重视DP 重视DP 重视DP

区间DP

INTRO & PROBLEM 1

- ▶ NOI 1995 石子合并 (SIMPLE)
- ▶ 有 $N(N \leq 100)$ 堆石子摆成一条线，要将石子合并，每次只能选相邻的两堆合并成新的一堆，并将新堆的石子数记为该次操作的代价，求将所有堆合并为一堆的最小代价。

INTRO

- ▶ 养成习惯 看数据范围猜时间复杂度
- ▶ $O(n^3)$ 的时间复杂度非常显然
- ▶ 看上去很像DP
- ▶ 如何DP?

TEMPLATE

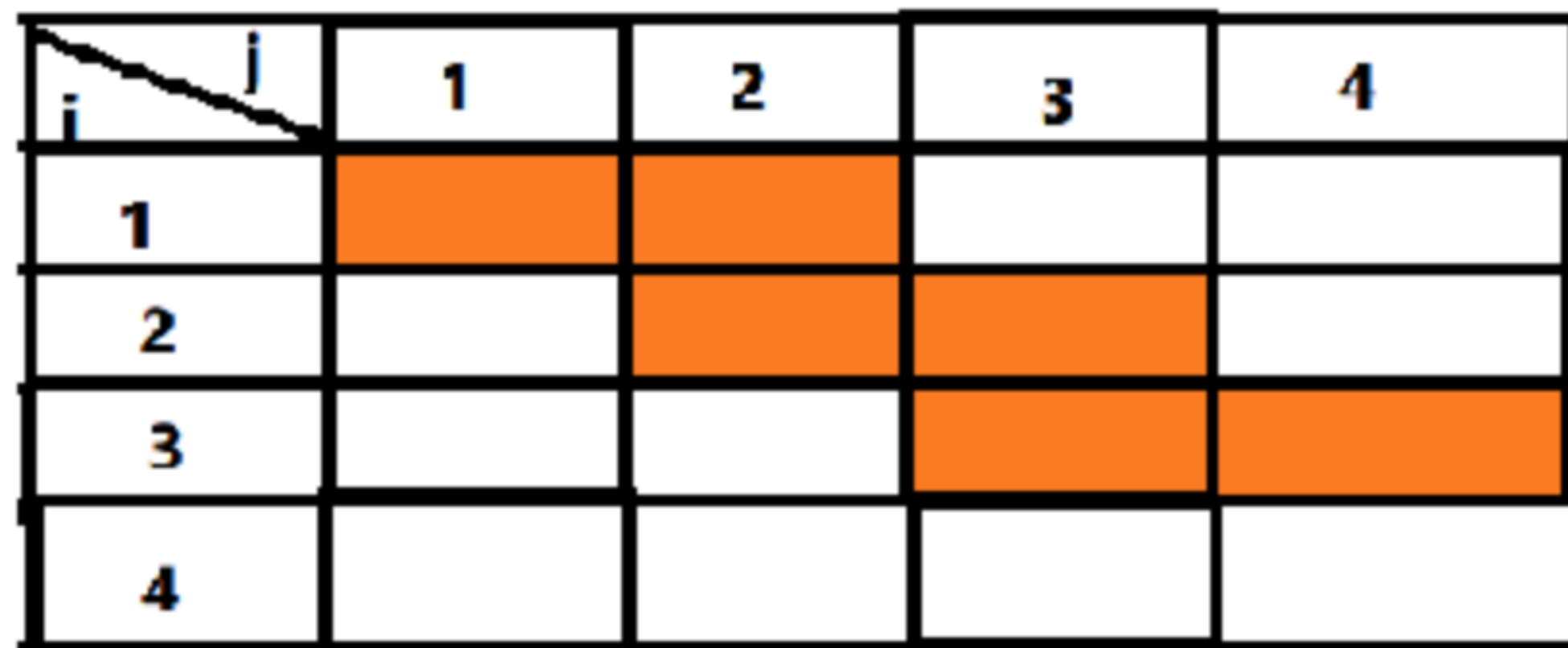
- ▶ 由此引入今天的第一个主角：区间动态规划
- ▶ 区间DP一般有它的模板 由小区间转移到大区间
- ▶ 假设 $N=4$
- ▶ 第一步咱们首先要考虑两个石子合并的情况

TEMPLATE

$i \backslash j$	1	2	3	4
1				
2				
3				
4				

- ▶ 区间长度为2的时候，似乎不需要转移，这是一个定值
- ▶ 考虑从区间为2到区间为3的转移，也就是3个石子合并时候的最优解

TEMPLATE



<i>i</i> \ <i>j</i>	1	2	3	4
1				
2				
3				
4				

- ▶ 比如区间[1,3] 它可以从[1,2]和[3,3]合并而来，也可以从[1,1]和[2,3]合并而来
- ▶ 我们需要对这两个抉择取max

TEMPLATE

$i \backslash j$	1	2	3	4
1				
2				
3				
4				

- ▶ 现在我们有合并连续3个石子时候的最优解，我们需要从它转移到合并4个石子
- ▶ 这个时候选择就多了起来

TEMPLATE

$i \backslash j$	1	2	3	4
1				
2				
3				
4				

- ▶ 考虑4个石子可能由哪些方案合并而来?
- ▶ $\{[1,1][2,4]\} \{[1,2][3,4]\} \{[1,3],[4,4]\}$ 我们需要比较它们中的最大值

TEMPLATE

$i \backslash j$	1	2	3	4
1				
2				
3				
4				

- ▶ 然后呢？问题解决了！
- ▶ 发现区间规律的一般模板了吗，当当前状态是一个大区间时，你需要分割这个大区间为两个小区间，然后比较所有分割方法中的最优解

TEMPLATE

令状态 $f(i, j)$ 表示将下标位置 i 到 j 的所有元素合并能获得的价值最大值，那么 $f(i, j) = \max\{f(i, k) + f(k + 1, j) + cost\}$ ， $cost$ 为将这两组元素合并起来的代价。

- ▶ 这样，我们可以得到一个区间动态规划型题目的一般模板

INTRO

- ▶ NOI 1995 石子合并 (SIMPLE)
- ▶ 有 $N(N \leq 100)$ 堆石子摆成一条线，要将石子合并，每次只能选相邻的两堆合并成新的一堆，并将新堆的石子数记为该次操作的代价，求将所有堆合并为一堆的最小代价。
- ▶ 回到我们开头的问题，如何求解？
- ▶ <https://github.com/PrimoPan/CUC-ACM-Training-14>

EXTENDED

- ▶ 回到刚刚那个问题，如果我们的N个石子，不是按照线性排列的，而是按照环状排列的，我们应该怎么办？
- ▶ 算法竞赛中面对环状问题，最常用的方法是——拆环为链
- ▶ 将原数组复制一遍，COPY到现在的数组的后面
- ▶ 然后取区间长度最大值为N，进行N次刚刚的那种DP
- ▶ 当然，同学们如果之后进行探索，会发现这个问题可以利用四边形不等式优化，感兴趣的同学可以深究，在此不做赘述

EXTENDED

- ▶ 这道题会留成课后练习题，思路已经讲了，大家自己尝试着把Problem 1的代码改进成Problem 2的代码。

PROBLEM 2

- ▶ 给出一串括号（包括圆括号和方括号），要求输出添加括号最少，并使所有括号均有匹配的串。
- ▶ SAMPLE INPUT: `[([]`
- ▶ SAMPLE OUTPUT: `()[(())]`
- ▶ 输入字符串长度最长为100

PROBLEM 2

- ▶ 这种括号匹配类问题八九不离十都是用区间DP去做，当然变种很多，这是最最基础的模板题
- ▶ Hint：用 $dp[i][j]$ 表示区间 $[i,j]$ 需要添加多少个括号，使得 $[i,j]$ 这个区间能是一个合法的括号序列，怎么写状态转移方程？给大家五分钟时间思考

PROBLEM 2

- ▶ 用 $dp[i][j]$ 表示区间 $[i,j]$ 需要添加多少个括号，使得 $[i,j]$ 这个区间能是一个合法的括号序列
- ▶ 如果当前这个 i 和 j 正好匹配，那么 $dp[i][j]=dp[i+1][j-1]$ ，因为不需要添加序列
- ▶ 如果不匹配，我们就需要枚举区间断点 k
- ▶ $dp[i][j]=\min(dp[i][k]+dp[k+1][j])$
- ▶ 因为我们需要输出解决方案，所以要记录这个 k 的位置，记录为 $pos[i][j]$ 。

PROBLEM 2

- ▶ 代码见: <https://github.com/PrimoPan/CUC-ACM-Training-14> Problem 2

EXTENDED

- ▶ 课堂时间有限，希望同学们认真阅读 <https://www.luogu.com.cn/blog/BreakPlus/ou-jian-dp-zong-jie-ti-xie> 这个blog，把题单刷完

状压DP

REVIEW

- ▶ 需要大家首先复习一下位运算的知识
- ▶ <https://oi-wiki.org/math/bit/>
- ▶ 为什么需要状态压缩？当我们需要去表达一个长度并不长的序列的状态时（举个例子，16个路灯，每个路灯的开关情况是开还是关，如果我们用一个数组表示，那就有 2^{16} 种状态，我们需要开 2^{16} 个长度为16的数组来记录这些状态
- ▶ 那为啥不直接用二进制数表示呢？

INTRO

- ▶ 当所有路灯都开着的时候 我们的状态就是二进制数 11111111111111111111
- ▶ 当所有路灯都关着的时候 我们的状态就是二进制数 00000000000000000000
- ▶ 打开某一个路灯，即将某一位二进制数 设为1
- ▶ 关闭某一个路灯，即将某一位二进制数 设为0
- ▶ 对某一个路灯进行开关操作，即对二进制数的某一位进行取反操作，如何取反，刚才的review里聪明的你们应该已经复习到了

INTRO

例题



「SCOI2005」互不侵犯



在 $N \times N$ 的棋盘里面放 K 个国王 ($1 \leq N \leq 9, 1 \leq K \leq N \times N$)，使他们互不攻击，共有多少种摆放方案。

国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共 8 个格子。

INTRO

解释

设 $f(i, j, l)$ 表示前 i 行，第 i 行的状态为 j ，且棋盘上已经放置 l 个国王时的合法方案数。

对于编号为 j 的状态，我们用二进制整数 $sit(j)$ 表示国王的放置情况， $sit(j)$ 的某个二进制位为 0 表示对应位置不放国王，为 1 表示在对应位置上放置国王；用 $sta(j)$ 表示该状态的国王个数，即二进制数 $sit(j)$ 中 1 的个数。例如，如下图所示的状态可用二进制数 100101 来表示（棋盘左边对应二进制低位），则有 $sit(j) = 100101_{(2)} = 37, sta(j) = 3$ 。



设当前行的状态为 j ，上一行的状态为 x ，可以得到下面的状态转移方程：

$$f(i, j, l) = \sum f(i - 1, x, l - sta(j)).$$

设上一行的状态编号为 x ，在保证当前行和上一行不冲突的前提下，枚举所有可能的 x 进行转移，转移方程：

$$f(i, j, l) = \sum f(i - 1, x, l - sta(j))$$

PROBLEM 4

- ▶ 现有 n 盏灯，以及 m 个按钮。每个按钮可以同时控制这 n 盏灯——按下了第 i 个按钮，对于所有的灯都有一个效果。按下 i 按钮对于第 j 盏灯，是下面3中效果之一：如果 $a[i][j]$ 为1，那么当这盏灯开了的时候，把它关上，否则不管；如果为-1的话，如果这盏灯是关的，那么把它打开，否则也不管；如果是0，无论这灯是否开，都不管。现在这些灯都是开的，给出所有开关对所有灯的控制效果，求问最少要按几下按钮才能全部关掉。
- ▶ $n \leq 10$ $m \leq 100$

PROBLEM 4

- ▶ 现有 n 盏灯，以及 m 个按钮。每个按钮可以同时控制这 n 盏灯——按下了第 i 个按钮，对于所有的灯都有一个效果。按下 i 按钮对于第 j 盏灯，是下面3中效果之一：如果 $a[i][j]$ 为1，那么当这盏灯开了的时候，把它关上，否则不管；如果为-1的话，如果这盏灯是关的，那么把它打开，否则也不管；如果是0，无论这灯是否开，都不管。现在这些灯都是开的，给出所有开关对所有灯的控制效果，求问最少要按几下按钮才能全部关掉。
- ▶ $n \leq 10$ $m \leq 100$

PROBLEM 4

- ▶ 现可以把灯的开和关视作1和0，则用一串01串（二进制）表示这一串灯的一个总的状态。
- ▶ 因为这个01串是2进制，所以他们所对应的10进制数最大不会超过 $(2^{10} - 1) = 1023$
- ▶ 也就是说最多有1023种状态，所以当然是可行的

PROBLEM 4

- ▶ 那么这题就可以直接BFS解决
- ▶ 利用之前的计算方法，开灯（1）就是把对应的那一位0变成1： $x |= 1 << (i-1)$
- ▶ 如果本身是1的话当然没有任何影响
- ▶ 同理，关灯（-1）的话就是把对应的那一位1变成0
- ▶ $x = x \& \sim(1 << (i-1))$
- ▶ 当然如果本身是0，也没有影响

PROBLEM 4

- ▶ 代码见上文github链接

感谢聆听