

数位DP&状压DP

PKU Hzyoi



01 数位DP




数位DP


- 在信息学竞赛中，有这样一类问题：求给定区间中，满足给定条件的某个 D 进制数或此类数的数量。
- 所求的限定条件往往与数位有关，例如数位之和、指定数码个数、数的大小顺序分组等等。
- 题目给定的区间往往很大，无法采用朴素的方法求解。
- 简单的题从左到右枚举过去，枚举每一位选什么(要小于 x 的这一位)，然后后面的乱选
- 但是稍难的复杂度不允许枚举，要快速计算这一位选什么的答案。



经典老题 windy数 (SCOI2009)

- 定义不含前导零且相邻两个数字之差至少为2的正整数为windy数。求 $[A,B]$ 中的windy数个数。
- $A,B \leq 2,000,000,000$


- 
- 数位DP模板题。
 - 我们先把Query(A,B)拆成两个问题Query(1,B)-Query(1,A-1)。
 - 设 $f[i][j]$ 表示以第i位为最高位（可以有前导零），且第i位上的数字为j的windy数个数。易得转移方程为：
 - $f[i][j] = \sum (f[i-1][k])$ 其中， $0 \leq k \leq 9$ 且 $\text{abs}(k-j) \geq 2$

- 
- 那么如何利用f数组求出Query(1,x)的值呢？
 - 我们可以逐位确定。
 - 什么是逐位确定呢？
 - 逐位确定就是从高位到低位依次确定下去。例如：
 - $Q(1,432)=Q(1,399)+Q(400,429)+Q(430,432)$ 。



[Ahoi2009]同类分布


- 求出 $[a,b]$ 中各位数字之和能整除原数的数的个数。
- $a,b \leq 10^{18}$


- 
- 可以拆分成两个前缀和的问题
 - 不难发现数位之和最多162
 - 所以可以直接在状态里记数位之和记为 p ，只有18位数位DP的部分不会太慢。
 - 数位DP查询的时候把左边的已确定部分和右边接起来。
 - 左边的部分可以直接得到数位和和 $\text{mod } p$ 是多少。
 - 右边的部分肯定是00000(k 个0)到99999(k 个9)的形式，这部分只要考虑数位之和是多少，还有 $\text{mod } p$ 是多少。
 - 然后只要枚举中间那位选多少，把用左边的情况去询问右边的答案即可。



随意魔改的题

- 给定 n, m, x , 求 $\sum_{i=0}^n \sum_{j=0}^m (i \text{ xor } j \text{ xor } x)^2$
- $N, m, x \leq 1e10, \text{mod } 998244353$

- 
- 数位dp本质上是一个难做的限制拆成 \log 个可以预处理的限制
 - 既然是限制，两维的限制显然可以拆开
 - 具体的， $<a$ 的一个限制，我们可以将它拆解为 \log 个“前若干位为abc，后若干位任意”的限制。
 - 两维都这样拆解，然后 $O(\log^2 n)$ 进行枚举。

- 
- 对于i和j的两个这样的限制如何计算 $\sum d(i \text{ xor } j \text{ xor } x)$ 。
 - 首先考虑只有i的限制， $\sum d(i \text{ xor } x)$ 如何计算，那么我们可以注意到“后若干位任意”的那些位异或完仍然是“后若干位任意”，只要将前面的若干位进行异或，后面若干位仍然任意。
 - 然后注意到例如所有形如010xxxx的d之和可以简单地用0101111和(0100000-1)的前缀和相减得到，所以我们可以直接计算平方和的前缀和，相减即可。



[908G] New Year and Original Order

- 定义函数 $S(n)$ 的值为将 n 中所有数位上数字排序后形成的新数。
- 求 $\sum_{i \in [1, N]} S(i)$ 。
- $N \leq 10^{700}$ 。



粗暴的算法一

- 考虑数位型动态规划的一般思路， $F[i][0/1]$ 表示已经在前 i 位上填好数字，前 i 位是/否均与 N 对应相同，形成的新数之和。
- 转移时枚举第 i 位上的数字，将其插入新数的某个位置，需要给新数的一部分乘上一个10的幂，而幂指数不能用状态表示。
- 不妨把所有的情况都记录下来， $F[i_1][i_2] \dots [i_9][0/1][10]$ 表示前几位中有 i_1 个1， i_2 个2， \dots ，前 i 位是/否均与 N 对应相同，数字 j 对答案的贡献。
- 总时间复杂度 $O(10^2 * N^9)$ 。



一种想法

- 状态数与 i 中数字的顺序有关，而 i 对答案的贡献与 i 中数字顺序无关。
- 也就是说，相同数字只需要关心其出现次数。
- 还是将 N 拆分成 $10 \cdot \log N$ 个前半段确定、后半段均未确定的数， $F[i][j]$ 表示答案的前 i 位由数字 $[0..j]$ 组成，按 j 分段，结合组合数计数转移。总时间复杂度 $O(10^2 \cdot N^3)$ 。



emmmm

- 但是这个动态规划的数组难以应用到所有的状态，重新回到一般思路。
- 可以发现，算法一中计算10的幂指数只需要知道不小于该位置上填的数的个数即可。将其写入状态。
- $F[i][j][k][0/1]$ 表示已经确定了前i位的数字，其中 $\geq j$ 的数字有k个，前i位是/否均与N对应相同的方案数。
- 计算时对于每一位，假设前面有j个数大于k的数，那么就产生 $1 \dots 1$ (j个1)的贡献。
- 比如3459： ≥ 3 的有4个， ≥ 4 的有3个， ≥ 5 的有2个， $\geq 6..8$ 的有2个， ≥ 9 的有1个。
- 所以贡献就是 $1111*3+111*4+11*1+11*3+1*9=3456$ 。
- 总时间复杂度 $O(10^2*N^2)$ 。



02

状压DP

状态压缩



状压DP

- 顾名思义就是把某个状态压缩起来的DP。
- 常见的DP状态是 $dp[i]$ ， i 是一个 2^n 规模的数表示 n 个点的状态，一般为是否使用，是否覆盖这种东西。

*Noip2016 愤怒的小鸟

- Kiana最近沉迷于一款神奇的游戏无法自拔。
简单来说，这款游戏是在一个平面上进行的。
有一架弹弓位于 $(0,0)$ 处，每次Kiana可以用它向第一象限发射一只红色的小鸟，小鸟们的飞行轨迹均为形如 $y=ax^2+bx$ 的曲线，其中 a,b 是Kiana指定的参数，且必须满足 $a<0$ 。
当小鸟落回地面（即 x 轴）时，它就会瞬间消失。
在游戏的某个关卡里，平面的第一象限中有 n 只绿色的小猪，其中第 i 只小猪所在的坐标为 (x_i,y_i) 。
如果某只小鸟的飞行轨迹经过了 (x_i,y_i) ，那么第 i 只小猪就会被消灭掉，同时小鸟将会沿着原先的轨迹继续飞行；
如果一只小鸟的飞行轨迹没有经过 (x_i,y_i) ，那么这只小鸟飞行的全过程就不会对第 i 只小猪产生任何影响。
例如，若两只小猪分别位于 $(1,3)$ 和 $(3,3)$ ，Kiana可以选择发射一只飞行轨迹为 $y=-x^2+4x$ 的小鸟，这样两只小猪就会被这只小鸟一起消灭。
而这个游戏的目的，就是通过发射小鸟消灭所有的小猪。
这款神奇游戏的每个关卡对Kiana来说都很难，所以Kiana还输入了一些神秘的指令，使得自己能更轻松地完成这个游戏。这些指令将在【输入格式】中详述。
假设这款游戏一共有 T 个关卡，现在Kiana想知道，对于每一个关卡，至少需要发射多少只小鸟才能消灭所有的小猪。由于她不会算，所以希望由你告诉她。



题解：

- 这题显然是状压dp。
- 由于两点可以确定一条抛物线，所以最普通的转移是枚举两个点确定一条抛物线，然后枚举剩余的点，验证一下是否在抛物线上。复杂度 $O(2^n * n^3)$ 。
- 优化：
 1. 可以预处理每条抛物线能打到哪些点。
 2. 只需要枚举一个点，另一个点则是没被打中的点中编号最小的，这样就可以确定一条抛物线了。时间复杂度 $O(2^n * n)$

[SDOI2009]学校食堂Dining

- 小F 的学校在城市的一个偏僻角落，所有学生都只好在学校吃饭。学校有一个食堂，虽然简陋，但食堂大厨总能做出让同学们满意的菜肴。当然，不同的人口味也不一定相同，但每个人的口味都可以用一个非负整数表示。由于人手不够，食堂每次只能为一个人做菜。做每道菜所需的时间是和前一道菜有关的，若前一道菜的对应的口味是 a ，这一道为 b ，则做这道菜所需的时间为 $(a \text{ or } b) - (a \text{ and } b)$ ，而做第一道菜是不需要计算时间的。其中，or 和 and 表示整数逐位或运算及逐位与运算，C语言中对应的运算符为“|”和“&”。学生数目相对于这个学校还是比较多的，吃饭做菜往往就会花去不少时间。因此，学校食堂偶尔会不按大家的排队顺序做菜，以缩短总的进餐时间。虽然同学们能够理解学校食堂的这种做法，不过每个同学还是有一定容忍度的。也就是说，队伍中的第 i 个同学，最多允许紧跟他身后的 B_i 个人先拿到饭菜。一旦在此之后的任意同学比当前同学先拿到饭，当前同学将会十分愤怒。因此，食堂做菜还得照顾到同学们的情绪。现在，小F 想知道在满足所有人的容忍度这一前提下，自己的学校食堂做完这些菜最少需要多少时间。
- 数据范围：
- $1 \leq N \leq 1,000, 0 \leq T_i \leq 1,000, 0 \leq B_i \leq 7, 1 \leq C \leq 5$




题解：

- 简单的状压DP。
- 用 $f[i][j][k]$ 表示前 $i-1$ 人都吃过饭， i 与 i 之后 7 人的吃饭情况为 j ，上一个吃饭的人与 i 的相对位置为 k 的最小时间。
- 然后枚举 $i \sim i+B_i$ 中没吃过饭的人转移一下即可。
- 时间复杂度 $O(n \cdot 2^8 \cdot 8 \cdot 8)$ 。



俄罗斯套娃 (Matrjoschka)

- 输入 $n, k (0 \leq n; k \leq 10^9)$, 我们有 $k + 1$ 个集合 $S_0 \dots S_k$ 。
- 其中 $|S_0| = n$ 即 S_0 的大小为 n , 并且有 $S_i \subseteq S_{i-1} (1 \leq i \leq k)$
- 即 S_i 是 S_{i-1} 的子集。
- 问 (S_0, S_1, \dots, S_k) 有多少种可能? 结果模 10^9+7 输出

- 
- 每个元素显然是独立的。
 - 根据乘法原理，答案一定是 x^n
 - 分析一下情况每个元素根据所属集合有 $k+1$ 种情况，那么答案就是 $(k+1)^n$
 - 可以魔改成复杂一点的版本。
 - 比如 $B \subseteq A, C \subseteq A$
 - 这个时候答案是 5^n




讲这个干啥


- 用来计算状态和转移的复杂度
- 常见的
- 在 2^n 状态中，转移需要枚举每个状态的所有子集，套到上面那个公式，复杂度为 3^n
- 学会计算状态之后就可以计算状态压缩DP的复杂度了。
- 在转移状态时一般也会把点分成若干种点来计算复杂度。





一个进阶题目


- 给定一个图，定义一个有根生成树的代价是这个生成树中的 $\sigma(w[u] \cdot \text{deep}[u])$
- $w[u]$ 表示从u点的父边，即往根方向移动的边的权值， deep 表示生成树中u的深度，根节点 深度为0
- 求最小代价生成树的代价
- $1 \leq n \leq 12$


- 
- 建树问题一般从下往上建,那么不妨用 $dp[i]$ 表示当前用的点状态是 i 的最小代价。
 - 但是不难发现, 添加一个父亲的时候, 添加的代价与整棵子树边权之和有关, 换而言之此时出现了两个单调状态, 代价越小越好, 边权之和越小越好。
 - 但边权之和显然规模过大无法维护。
 - 一条边的贡献可以在它那一层直接处理, 但是我们不知道这条边最终会在哪一层
 - 那么强制它在那一层, 最后建树的时候可以保证所有强制的条件满足, 这样层数的规模是 $O(n)$ 可以接受。


- 
- 每次转移可以认为是把A子树挂在B子树根节点下面。
 - 点分为三种，A子树中的，B子树中的，不在A、B子树中的。
 - DP状态要记录现在强制应该是第几层，还有根节点是什么，状态是 $O(2^n * n^2)$
 - 所以直接转移，枚举A节点集合和根，复杂度 $O(3^n * n^3)$
 - 可以优化，发现对于A子树，只要确定了根节点，所有的，只有根不同的B子树带来的贡献都是一样的，换言之对于所有根相同的A子树，都应该和同一个B子树转移。
 - 优化转移复杂度 $O(3^n * n^2 + 2^n * n^3)$

- 
- 已经能过了，但这题还有更优秀的解法
 - 钦定了层数之后，不只是一条父边可以直接确定最优的
 - 考虑从上往下一层层铺。
 - 不难发现我们不关心每个节点的father而只关心它在哪一层，因为它的father并不影响它的贡献，换言之我们一定可以选择最优的father。
 - 又因为同层边的贡献权都是相同的，所以假如一种方案放在某一层最优，放在另外层也一定最优。
 - 那么不妨对于上下两层的每种情况，直接求出最优的贡献，复杂度分析出来是 $O(3^n)$ 的

- 
- 然后考虑状态和转移，套用之前的复杂度分析理论，转移时点分为四种，上一层的，下一层的，在上一层上面的，在下一层下面的，还要考虑当前在第几层来算贡献
 - 所以复杂度是 $O(4^n \cdot n)$
 - 好像过不去啊。
 - 但是考虑一下常数， i 个点不可能构成超过 i 层的树，所以常数除以2，刚好能过

- 
- 基于贪心的DP会有基于反向贪心的优化。
 - 把上一层上面的点也当做上一层的，那么对于一个状态来说答案只有可能变劣而不会变优。
 - 但是最优方案一定会出在DP的过程里。而且点的种类少了一种。
 - 总复杂度 $O(3^n \cdot n)$

- 
- 状态不再是普通的 2^n 之类的，而是一些奇怪的东西
 - 常见的比如插头dp的最小表示法表示状态
 - 再比如说：
 - 给出一个序列。
 - 问有多少个 n 的排列的最长上升子序列是给出的序列。
 - $n \leq 12$

- 
- 这个题就是对单调栈进行了一个状压DP。
 - 我们并不关心原序列，因为只有单调栈的状态会影响最长上升子序列。
 - 每个元素三个状态，在栈里，从栈里出去了，还没出现过。
 - 每次从左到右哪一个还没用过的元素塞进去就行了。




其他的优化方向


- 有时候一些题目的特殊性质能使得有的看起来复杂度不对的做法可能状态和转移有限而可以通过



TC SRM 693 Div1 C

- 给一个 n 个点 m 条边的图，求有多少点集，使得点集内没有三角形
- $N, M \leq 60$

- 
- 朴素的想法： $F[i][s]$ 表示处理了前 i 个点， s 是个二进制数，记录每个点是否选择。
 - 每次转移判断一下是否构成三角形即可
 - 点数这么大，显然复杂度爆炸了

- 
- 优化方案：若做到一个点 i ，之前的所有点到之后的所有点之间都没有边相连了，那么之前的点的状态就可以忽略了
 - 可以 $O(n)$ 求出状态数
 - 边数和点数同级
 - 标算：多随机几次，随机出状态数相对较少的排列，状态数大概能到 $1e6$ 级别，即可通过本题



非常感谢您的观看

THANK YOU FOR YOUR WATCHING