动态规划

___debug

2019 年 1 月 30 日

大纲

DP 的一般形式是表示出能代表出至于后续或答案有关的当前的**状态**，并在这些状态之中进行**转移**。
而 DP 的优劣就用状态数以及转移的时间复杂度来评定了，好的 DP 应该是深挖了题目或数据的性质，从而达到时间与空间上的良好效果。
DP 的难点主要分为两类：一类以状态设计为难点，一类以转移的优化为难点。

(bzoj 2298)

一次考试共有 n 个人参加，第 i 个人说：“有 a_i 个人分数比我高，b_i 个人分数比我低。”。
可以有相同分数，问最多有多少人在讲真话。

$n \leq 10^5$

转换模型：一个人说的话如果为真说明从第 $a_i + 1$ 个人到 $n - b_i$ 个人的分数都是一样的。
如果我们把他看作一段线段，那么在这 n 条线段中，相交且不重合的线段一定不能同时为真话，所以即为求问题就转换成选出尽可能多的不向交或完全重合的线段，且对于某条线段完全重合的个数要小于等于线段长。
将线段按某左端点排序，令 $f[i]$ 为到了 i 这个坐标以及以前所能选的最多线段：

$$f[i] = f[i - l] + \min(\text{线段个数}, \text{线段长度})$$

数位 DP

数位 DP 常用来统计或查找一个区间满足条件的数，然后按数位顺序 DP，一般需要仔细分情况讨论，常见处理如将区间拆为 $[1, R], [1, L)$，记忆化，预处理等。

(bzoj 3131)

有一个二维坐标系，X 轴，Y 轴坐标范围均为 $1..N$。初始的时候，所有的整数坐标点上均有一块金子，共 N^2 块。
一阵风吹过，初始在 (i,j) 坐标处的金子会变到 $(f(i),f(j))$ 坐标处。其中 $f(x)$ 表示 x 各位数字的乘积（去除前导零），例如 $f(99)=81$，$f(12)=2$，$f(10)=0$。如果金子变化后的坐标不在 $1..N$ 的范围内，我们认为这块金子已经被删除。
同时可以发现，对于变化之后的游戏局面，某些坐标上的金子数量可能不止一块，而另外一些坐标上可能已经没有金子。
求出风吹过之后金块数量前 K 大的坐标的金块数量和。答案可能很大，输出对 10^9+7 取模之后的答案。

$N \leq 10^12, K \leq 100000$

我们发现实际的 f 可能的取值很少，且取值的质因子只有 $2,3,5,7$ 几种。
于是我们可以用 $f(i,S,j)$ 表示从高到低到了第 i 位，j 表示已经考虑过得位置是否小于 N，S 为当前乘积是否是 0 以及不是 0 时 $2,3,5,7$ 因子的次数，最后得到答案后用优先队列即可统计答案.

概率 DP

概率 DP 是一类求事件概率或期望的 DP 的总称。
对于求概率问题，有时利用补集转化，或者将其转化为计数问题；而对于求期望则大多利用期望的线性性来解决问题。

(bzoj 4008)

给你 N 张牌, 每张牌有一个发动概率 P_i 以及伤害 D_i. 共有 R 回合, 每回合会按编号从小到大依次考虑本次游戏中还未发动的卡牌, 依次尝试发动; 如果成功发动, 进入下一回合. 求期望伤害之和. 共 T 组数据.

$T \leq 444, N \leq 220, R \leq 132$

直接 DP 显然只能状压, 必须要改变状态.

直接 DP 显然只能状压, 必须要改变状态.

不妨依次考虑每个卡牌而非每局, 分别计算对答案的贡献.
注意到对于卡牌 i, 之前具体那些卡牌已经发动了是不需记录的, 我们所需要知道的仅仅是之前有多少张卡牌已经发动, 也即还剩下多少局会考虑到第 i 张牌.

直接 DP 显然只能状压，必须要改变状态.

不妨依次考虑每个卡牌而非每局，分别计算对答案的贡献.
注意到对于卡牌 i，之前具体那些卡牌已经发动了是不需记录的，我们所需要知道的仅仅是之前有多少张卡牌已经发动，也即还剩下多少局会考虑到第 i 张牌.

那么我们设状态 $dp(i,j)$ 表示考虑到第 i 张牌时，还剩下 j 局没有发动过卡牌 (即前 $i-1$ 张牌都没有在这 j 局中发动). 转移也是很显然的，考虑 $i-1$ 有没有发动即可:

$$dp(i,j) = dp(i-1,j)(1-p_{i-1})^j + dp(i-1,j+1)(1-(1-p_{i-1})^{j+1})$$

最后答案的计算就十分简单了，直接枚举 i,j，考虑贡献即可.

(codeforces 113D)

有一个 n 个点 m 条边的无向图，两个人分别从 x, y 出发，每个人每分钟有 p_i 的概率不动，有 $1 - p_i$ 的概率走到随机一个相邻的点. 当他们在同一时刻选择前往同一个房间，他们就会在那个房间相遇并停止.
求在每个点相遇的概率.

$n \leq 22$

这道题有两个做法.

这道题有两个做法.
第一个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j, 此时开始, 之后在 t 点相遇的概率.

这道题有两个做法.
第一个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j, 此时开始, 之后在 t 点相遇的概率.
枚举终点, 做 n 次高斯消元即可, 不过直接这样做是 $O(n^7)$ 的.

这道题有两个做法.
第一个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j, 此时开始, 之后在 t 点相遇的概率.
枚举终点, 做 n 次高斯消元即可, 不过直接这样做是 $O(n^7)$ 的. 注意到每次高斯消元时, $Ax=B$ 只有 B 发生了变化. 于是把 B 改成一个矩阵就可以 $O(n^6)$ 了.

这道题有两个做法.
第一个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j, 此时开始, 之后在 t 点相遇的概率.
枚举终点, 做 n 次高斯消元即可, 不过直接这样做是 $O(n^7)$ 的. 注意到每次高斯消元时, $Ax=B$ 只有 B 发生了变化. 于是把 B 改成一个矩阵就可以 $O(n^6)$ 了.
第二个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j 这种情况的期望出现次数.

这道题有两个做法.
第一个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j, 此时开始, 之后在 t 点相遇的概率.
枚举终点, 做 n 次高斯消元即可, 不过直接这样做是 $O(n^7)$ 的. 注意到每次高斯消元时, $Ax=B$ 只有 B 发生了变化. 于是把 B 改成一个矩阵就可以 $O(n^6)$ 了.
第二个做法是, 设 $f(i,j)$ 为第一个人在 i, 第二个人在 j 这种情况的期望出现次数.
因为终止状态只会出现 $0/1$ 次, 于是期望就是概率了...

树形 DP

树形 DP 是指基于树的结构的动态规划，基础的有：

- 树的直径：DP 记录子树内的最长路
- 树的重心：DP 记录子树大小
- 树上最大权独立集：DP 记录子树的根是否选择
- 树形依赖背包：在 dfs 序上 DP(即每次选择是否跳过子树)，或通过将父节点的 DP 值传入孩子 DP
- 虚树：在原树上只保留需要的点与他们的 LCA 的树称为虚树，建树方法为：将点按 dfs 序排序，一次将点加入，用一个栈维护树上的已加入的点和他们 LCA 的右链，每次加入一个点，与栈顶比较，便可以建出虚树
- ...

(bzoj 3611)

给出一 n 个点棵树, 树上边的长度都为 1, 每次询问 k 个点, 在 k 个点之间有 $\binom{k}{2}$ 条路径, 请求出:

1. 这些路径的代价和
2. 这些路径中代价最小的是多少
3. 这些路径中代价最大的是多少

$n \leq 10^6, \sum k \leq 2n$

有着询问的和小于的限制的题是虚树的一个显著的特征。
建出虚树, 然后直接在上面树形 DP。设 $f(i)$ 为以 i 为根的子树内路径的代价和, $size(i)$ 为 i 的子树中询问点的个数, $max(i)$ 为 i 子树内的到 i 的最长路, $min(i)$ 为最短路，则:

$$f(i) = f(son(i)) + size(i) \times (k - size(son(i)))$$

$$max(i) = \max\{max(son(i)) + 1\}$$

路径和答案是 $f(root)$, 对于每个点, 用它的最大与次大的儿子更新答案. 如果他是被询问的点, 则还可以用它每个儿子更新答案.

状态压缩 DP

基于状态压缩的 DP 是由于状态用单个简单的变量直接存储存在空间的浪费，而采用压缩的状态的动态规划，例如：

- 插头 DP：维护当前已决策和未决策的一条 Z 字形的轮廓线的插头状态，用括号序列配对插头，每次只需分情况讨论即可，但是这类 DP 的显著特点就是情况繁多，使用时须细心

(bzoj 3836)

给你一个 N 个点 M 条边的无向图，每个点有点权 A_i. 保证任意两点之间距离不超过 10.
现在要你选取一些点，使得每个点要么自己被选，要么相邻的点被选.
一个方案的代价定义为选取的点的点权和. 请你最小化代价.

$N \leq 20000, M \leq 25000, 0 \leq A_i \leq 10000$

考虑如何利用“距离不超过 10”这个限制.
不妨把生成树搞出来, 那么这个树的深度 ≤ 10. 一个很显然的想法是 DP 时, 对每个点设状态 $dp(u, S)$, 表示以 u 为根的子树, 当 u 到根这条路径上的点的选取情况是 S 时的最小代价.
由于, 一个点可能会对通过非树边覆盖祖先, 对于 S 中每个未被选取的点, 我们可以再记录一下它是否已经被覆盖. 对于一个点状态数是 3^{10} 的.

用 3 进制来表示 S. 规定一下 S 中某一位的值的意义:

- 0: 这个点被选取了
- 1: 这个点未被选取, 但已经被覆盖
- 2: 这个点未被选取, 且仍未被覆盖

在 DFS 序上做即可. 转移并不困难, 注意的是当 DFS 序中前一个点不是当前点的父亲时, 要把 dp 值维护一下.
使用滚动数组, 空间复杂度 $O(N + M + 3^{10})$, 时间复杂度 $O(M + 3^{10}N)$. 记得对每个连通块都要做一遍.

(bzoj 2734)

对于任意一个正整数 $N \leq 100000$, 求出 $\{1, 2, ..., n\}$ 的满足" 若 x 在该子集中，则 $2x$ 和 $3x$ 不能在该子集中" 的子集的个数 (只需输出对 1000000001 取模的结果).

$N \leq 10^5$

比较明显的状压 DP.

考虑 $x = t2^a3^b b$, 当 t 不同时互不影响, 相同时 t 没有作用.
然就转化为网格图上的" 不能选相邻的点" 的一个计数问题, 用大小为 $\log_3 N$ 的那一维状压, 做一次是 $O(3^{\log_3 N} \log_2 N) = O(N \log N)$.
所以总复杂度 $O(N \log^2 N)$.

DP 套 DP

某些 DP 问题的子判定问题不能简单的解决，而必须用另一个 DP 解决，此时就只能使用 DP 套 DP 的方法，即：外面的 DP 的状态是存的里层 DP 各个状态的值，利用里层的状态来判断外层的 DP 是否合法，类似的问题有 LCS 为定值的序列的方案数等等。

(bzoj 3864)

给你一个只由 ACGT 组成的字符串 S, 对于每个 $0 \leq k \leq |S|$, 问有多少个只由 ACGT 组成的长度为 M 的字符串 T, 使得 $\mathrm{LCS}(S, T) = k$.

$|S| \leq 15, M \leq 10^3$

DP 套 DP.

考虑算 LCS 的 DP

$$dp(i,j) = \max\left(dp(i-1,j-1) + [T_i = S_j], dp(i-1,j), dp(i,j-1)\right)$$

这题的 $|S|$ 很小, 不妨对于每个 i, 将 j 不同时的 DP 值都记录下来, 计个数即可.
但是直接记 DP 值状态数会爆炸, 但是注意到相邻的 DP 值只会差 1, 所以我们可以用 $2^{|S|}$ 的状态数将这些值记录下来.

大纲

DP 的形式优化

有时在做一个 DP 问题时将会遇到时间或者空间复杂度过高的问题，而在 DP 的形式上优化便是有效的优化技巧，典型的有预处理，分阶段 DP 等:

- 预处理: 我们可能发现，在 DP 的过程中，出现了重复的运算，浪费了时间，所以我们可以通过 DP 前预处理，或者 DP 过程总处理出最值，而达到为后面的 DP 提供便捷的功能与作用，达到优化的目的
- 分阶段 DP: 在某些 DP 中将 DP 拆为一个个有特点阶段也许比将整个 DP 放在一起更加节省时间与空间，所以对于彼此相对无关的转移，可以分开考虑

给你长度分别为 N, M 的数组 A, B, 求最长公共上升子序列.

$N, M \leq 5000$

不妨设状态 $dp(i,j)$ 表示 A 数组考虑到 i, B 数组考虑到 j 并且**必须选** j 的最大长度.
当 $A_i = B_j$ 时, 显然有转移

$$dp(i,j) = \max\{dp(i-1,k) \mid k < j, B_k < B_j\} + 1$$

由于 $A_i = B_j$, 那么这个限制实际上就是 $B_k < A_i$, 对于同一个 i 的限制是相同的.
我们可以从小到大枚举 j, 动态维护 $\max\{dp(i-1,k) \mid B_k < A_i\}$ 直接转移即可.

(bzoj 2436)

现在有 n 个活动，每个活动需要占用 $[l_i, r_i)$ 的时间. 现在有两个会场，规定两个会场不能同时有活动，但是一个会场可以同时举办多个活动.
要你安排每个活动在哪个会场举行，或是不举行，使得举行活动较少的会场举办的活动最多. 同时，你还要对 $i \in [1, n]$ 求出如果强制活动 i 必须举行，那么举行活动较少的会场最多举办的活动是多少.

$n \leq 200$

首先考虑第一问. 非常简单. 预处理出 $cnt(l,r)$ 表示 $[l,r)$ 包含了多少段区间. 然后设状态为 $f(i,j)$ 表示考虑了 $[0,i)$ 这个区间, 第一个会场举办了 j 个活动的前提下, 第二个会场最多举办多少个活动. $O(n^3)$ 解决.

对于第二问这种问题，比较套路的做法就是：原来用 DP 解决的序列上的问题，要询问改变其中一个元素的答案，可以考虑预处理出前缀后缀 DP 数组，合并一下即可

令 $g(i,j)$ 表示考虑了 $[i,size)$ 这个区间时的 DP 数组.
不难想到这样做：处理出 $max(l,r)$ 表示 $[l,r)$ 这个区间强制选时的答案. 但是这样直接做，首先枚举 l,r 要 $O(n^2)$，然后合并前缀和后缀 (枚举 j_1,j_2) 要 $O(n^2)$，一共是 $O(n^4)$ 的，无法承受.

考虑优化合并. 因为我们并不需要像在 DP 的时候一样要知道对于每个第一个会场的 j，对应的第二个会场举办的最多活动是多少；我们只需要知道一个最大值. 而且这个数组又有很好的单调性. 即：随着 j 的增大，$f(i,j)$ 必定不增. $g(i,j)$ 同理. 所以利用这个我们就可以先枚举 j_1，然后二分出 j_2 的最优位置. 进一步，由于 f,g 都有单调性，我们可以直接 TwoPointers.

总时间复杂度 $O(n^3)$.

决策单调性优化

DP 问题的转移往往需要大量的时间，如果我们能发现一些性质，找到一些规律来优化 DP 决策转移的过程，那么在时间上我们便能得到很大的优化，常见的有四边形不等式优化，以及一些 1D/1D 动态规划的优化。

四边形不等式优化

四边形不等式优化: 对于形如以下的 DP

$$f(i,j) = f(i,k-1) + f(k,j) + w_{i,j}$$

如果 w 满足四边形不等式:

- 任意 $i \le i' \le j \le j'$, 有 $w_{i,j} + w_{i',j'} \le w_{i',j} + w_{i,j'}$
- 任意 $i' \le i \le j \le j'$, 有 $w_{i,j} \le w_{i',j'}$

那么也可证明

$$f(i,j) + f(i',j') \le f(i',j) + f(i,j')$$

而如果得到了这样的式子，则就可以证明 $f(i,j)$ 的决策一定在 $f(i,j-1)$ 与 $f(i-1,j)$ 的决策之间:

$$s_{i,j-1} \leq s_{i,j} \leq s_{i-1,j}$$

四边形不等式的证明网络上十分详细，这里就不再演示，这里提供一篇证明。http://wenku.baidu.com/link?url=344UHCQdTP9z2dFTCCGB3eBYHnlBeF0IAYdFeLmA_p0QU9nGv3L-6A_yISk4zUKcTMBDrokvx_i-5BHh7H5ZFfjS3hf2j9jHdPCgUXwQjqS

1D/1D 动态规划方程的优化

$$f(i) = \min\{f(j) + w_{j,i} | j \leq i\}$$

若 w 满足四边形不等式，则可以证明 $f(i)$ 的决策也一定单调。

(bzoj 1563)

小 G 是一个出色的诗人，经常作诗自娱自乐。但是，他一直被一件事情所困扰，那就是诗的排版问题。
一首诗包含了若干个句子，对于一些连续的短句，可以将它们用空格隔开并放在一行中，注意一行中可以放的句子数目是没有限制的。
小 G 给每首诗定义了一个行标准长度 (行的长度为一行中符号的总个数)，他希望排版后每行的长度都和行标准长度相差不远。显然排版时，不应改变原有的句子顺序，并且小 G 不允许把一个句子分在两行或者更多的行内。
在满足上面两个条件的情况下，小 G 对于排版中的每行定义了一个不协调度，为这行的实际长度与行标准长度差值绝对值的 P 次方，而一个排版的不协调度为所有行不协调度的总和。小 G 最近又作了几首诗，现在请你对这首诗进行排版，使得排版后的诗尽量协调 (即不协调度尽量小)，并把排版的结果告诉他。

n 为诗句个数，L 为标准长度，每一句诗长度小于等于 30
$n \leq 100000, L \leq 3000000, P \leq 10$

首先可以推得最直接的 DP:

$$f(i) = \min\{f(j) + |sum_i - sum_j - l|^p\}$$

令 $w_{i,j} = |sum_i - sum_j - l|^p$, 可以证明 w 满足四边形不等式。
由前面讲到的结论, 对于任意 $i \leq j$, $f(i)$ 的决策一定小于等于 $f(j)$ 的决策, 于是我们可以用一个栈来维护 DP 的决策.

一开始，只有 f(1)的函数值被计算出来，于是所有状态的当前最优决策都是 1。

1111111111111111111111111111111111111111111111111111111111111

现在，显然 f(2)的值已经确定了：它的最有决策只能是 1。我们用决策 2 来更新这个决策表。由于决策单调性，我们知道新的决策表只能有这样的形式：

1111111111111111111111111112222222222222222222222222222

这意味着我们可以使用二分法来查找“转折点”，因为如果在一个点 x 上，如果决策 2 更好，则所有比 x 大的状态都是决策 2 更好；如果 x 上决策 1 更好，则所有比 x 小的状态都是决策 1 更好。

现在决策 1 和决策 2 都已经更新完毕，则 f(3)业已确定，现在用决策 3 来更新所有状态。根据决策单调性，现在的决策表只能有以下 2 种类型：

1111111111111111111111111111112222222222222222233333333333

1111111111111111111111113333333333333333333333333333333333

而这样的决策表示绝对不会出现的：

1111111111133333333333333333322222222222222222222222222222，不可能。

那么，我们的更新算法就是：

1、考察决策 2 的区间[b,e]的 b 点上是否决策 3 更优，如果是，则全部抛弃决策 2，将此区间划归决策 3；如果否，则在决策 2 的区间[b,e]中二分查找转折点。

2、如果第 1 问的回答是“是”，则用同样的方法考察决策 1。

斜率优化

斜率优化 DP 是当 DP 转移式形如

$$f(i) = \min\{f(j) + k_i x_j + c_i + b_j\}$$

将与 j 无关的常数提出 $\min$，我们就是要求

$$\min\{k_i x_j + f(j) + b_j\}$$

令 $y_j = f(j) + b_j$，每次我们实际上是在所有过 (x_j, y_j) 且斜率为 $-k_i$ 的直线

$$y_j = -k_i x_j + B$$

中找到一个直线具有最小的 B，即纵截距最小。显然最优的 (x_j, y_j) 一定在凸壳上。于是我们便可以使用维护凸壳来将时间复杂度变得更优。

根据 x_j 和 k_i 的单调性，我们可以：

- x_j 与 k_i 同时单调：单调队列/单调栈 (hdu 3507)
- x_j 单调，k_i 不单调：单调队列/单调栈 + 二分斜率 (bzoj 2726)
- x_j 不单调：Splay 维护凸壳 (bzoj 1492)

凸单调性 DP 优化

(uoj 240)

有一个 $m \times m$ 的正方形网格, 其中有 n 个关键点, 你可以花不超过 k 个小正方形去覆盖这 n 个点. 要求这些小正方形的主对角线必须落在大正方形的主对角线上, 并且**面积并**最小.

$k \leq n \leq 10^5, m \leq 10^6$

首先不妨把在对角线上面的点都翻到下面来，然后去处一些无用点.
显然有 DP:

$$dp(i,j) = \max_{k<j}\{dp(i-1,k) + (x_j - y_{k+1} + 1)^2 - \max(0, x_k - y_{k+1} + 1)^2\}$$

上面的这个很容易斜率优化，所以时间复杂度 $O(nk)$.
如果 $dp(k,n)$ 是一个关于 k 的凸函数 (也就是说增量是单调的)，那么不妨二分 $dp(k-1,n)$ 和 $dp(k,n)$ 这两个状态之间的差. (也就是用一个一次函数去切)

具体怎么实现呢？我们把 DP 方程改为:

$$dp(i) = \max_{j<i}\left\{dp(j) + (x_i - y_{j+1} + 1)^2 - \max(0, x_j - y_{j+1} + 1)^2\right\} + x$$

注意最后面加上的那个 x，也就是转移一次就有 x 的代价. 这样再记录一下 $dp(n)$ 转移达到最优值所需要的最少转移次数，就可以二分了. 问题至此迎刃而解.

大纲

容斥: 引入

关于容斥, 大家可能会有这样几种看法:

容斥: 引入

关于容斥, 大家可能会有这样几种看法:

- 容斥就是隔一个数加上一个负号, 然后答案正好就对了

容斥: 引入

关于容斥, 大家可能会有这样几种看法:

- 容斥就是隔一个数加上一个负号, 然后答案正好就对了
- 容斥就是 $\sum_{i=0}^{n} \binom{n}{i} (-1)^i = [n = 0]$

容斥: 引入

关于容斥, 大家可能会有这样几种看法:

- 容斥就是隔一个数加上一个负号, 然后答案正好就对了
- 容斥就是 $\sum_{i=0}^{n} \binom{n}{i} (-1)^i = [n = 0]$

容斥到底是什么?

容斥: 引入

关于容斥, 大家可能会有这样几种看法:

- 容斥就是隔一个数加上一个负号, 然后答案正好就对了
- 容斥就是 $\sum_{i=0}^{n} \binom{n}{i} (-1)^i = [n = 0]$

容斥到底是什么?
在接着讲之前, 我们先来看几道经典的例题.

容斥: 引入

经典问题 1

给定 n 个数 $a_1, ..., a_n$, 要求将所有数分成两组, 使得两组中元素 or 和相同.

$n \leq 50, 0 \leq a_i \leq 2^{20}$

容斥: 引入

经典问题 1 - 题解

按位考虑, 如果所有的数在某一位上都为 0, 显然可以不用考虑.

容斥: 引入

经典问题 1 - 题解

按位考虑, 如果所有的数在某一位上都为 0, 显然可以不用考虑.
对于其它的位, 如果要满足题目的要求, 则必须满足所有这一位为 1 的数不能全部在同一组里.

容斥: 引入

经典问题 1 - 题解

按位考虑, 如果所有的数在某一位上都为 0, 显然可以不用考虑.
对于其它的位, 如果要满足题目的要求, 则必须满足所有这一位为 1 的数不能全部在同一组里. 虽然这个条件不好计数, 但是它的反面是很好计数的!

容斥: 引入

经典问题 1 - 题解

按位考虑, 如果所有的数在某一位上都为 0, 显然可以不用考虑.
对于其它的位, 如果要满足题目的要求, 则必须满足所有这一位为 1 的数不能全部在同一组里. 虽然这个条件不好计数, 但是它的反面是很好计数的!
所以, 枚举至少有哪些二进制位不满足条件, 然后用并查集维护一下就行了.

容斥: 引入

经典问题 2

给你一个 N 维超立方体, 第 i 个维度的长度为 r_i, 同时给你一个 N 维超平面 $x_1 + x_2 + ... + x_n = S$. 这个超平面把超立方体切成至多两部分, 求原点所在的那一部分的面积.

$N \leq 500, A_i \leq 500, S \leq 10^9$

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.
发现 $A_n(x)$ 是 $A_{n-1}(x_0)$ 在 $[0, x]$ 上的定积分.

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.
发现 $A_n(x)$ 是 $A_{n-1}(x_0)$ 在 $[0, x]$ 上的定积分. 利用基本的微积分知识, 我们可以得到 $A_n(x) = \frac{x^n}{n!}$.

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.
发现 $A_n(x)$ 是 $A_{n-1}(x_0)$ 在 $[0, x]$ 上的定积分. 利用基本的微积分知识, 我们可以得到 $A_n(x) = \frac{x^n}{n!}$.
接下来考虑坐标的限制.

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.
发现 $A_n(x)$ 是 $A_{n-1}(x_0)$ 在 $[0, x]$ 上的定积分. 利用基本的微积分知识, 我们可以得到 $A_n(x) = \frac{x^n}{n!}$.
接下来考虑坐标的限制.
题目就是要求 $\sum x_i \le S$, 其中每个 $x_i \in [0, r_i]$.

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.
发现 $A_n(x)$ 是 $A_{n-1}(x_0)$ 在 $[0, x]$ 上的定积分. 利用基本的微积分知识, 我们可以得到 $A_n(x) = \frac{x^n}{n!}$.
接下来考虑坐标的限制.
题目就是要求 $\sum x_i \leq S$, 其中每个 $x_i \in [0, r_i]$.
考虑容斥. 我们把没有限制的情况, 分别减去 $1, 2, .., n$ 超过限制的情况, 再加上 $1, 2, 1, 3$ 等同时超过限制的情况...

容斥: 引入

经典问题 2 - 题解

设 $A_n(x)$ 为在 N 维空间里 $S = x$, 且长度没有限制的面积.
发现 $A_n(x)$ 是 $A_{n-1}(x_0)$ 在 $[0, x]$ 上的定积分. 利用基本的微积分知识, 我们可以得到 $A_n(x) = \frac{x^n}{n!}$.
接下来考虑坐标的限制.
题目就是要求 $\sum x_i \le S$, 其中每个 $x_i \in [0, r_i]$.
考虑容斥. 我们把没有限制的情况, 分别减去 $1, 2, .., n$ 超过限制的情况, 再加上 $1, 2, 1, 3$ 等同时超过限制的情况...
注意到 $N \times \max A_i$ 不会很大, 上面的容斥很容易利用 DP 优化.

容斥: 引入

UOJ 185

给你一个 n 个点 m 条边的无向图, 再给你一棵 n 个点的树, 问有多少种点编号的映射方式, 使得 n 个点恰好匹配, 且树上的边均存在于原图中.

$n \leq 17, m \leq \frac{n(n-1)}{2}$

容斥: 引入

UOJ 185 - 题解

首先考虑一个错误的树形 DP.

容斥: 引入

UOJ 185 - 题解

首先考虑一个错误的树形 DP. 设 $dp(u, p)$ 表示考虑了以 u 为根的这个子树, 并且根映射到原图的 p 点.

容斥: 引入

UOJ 185 - 题解

首先考虑一个错误的树形 DP. 设 $dp(u, p)$ 表示考虑了以 u 为根的这个子树, 并且根映射到原图的 p 点. 这个显然可以 $O(n^3)$ 转移, 但是有什么问题呢?

容斥: 引入

UOJ 185 - 题解

首先考虑一个错误的树形 DP. 设 $dp(u, p)$ 表示考虑了以 u 为根的这个子树, 并且根映射到原图的 p 点. 这个显然可以 $O(n^3)$ 转移, 但是有什么问题呢?
不同的点可能映射到同一个点.

容斥: 引入

UOJ 185 - 题解

首先考虑一个错误的树形 DP. 设 $dp(u, p)$ 表示考虑了以 u 为根的这个子树, 并且根映射到原图的 p 点. 这个显然可以 $O(n^3)$ 转移, 但是有什么问题呢?
不同的点可能映射到同一个点. 于是考虑容斥.

容斥: 引入

UOJ 185 - 题解

首先考虑一个错误的树形 DP. 设 $dp(u, p)$ 表示考虑了以 u 为根的这个子树, 并且根映射到原图的 p 点. 这个显然可以 $O(n^3)$ 转移, 但是有什么问题呢?

不同的点可能映射到同一个点. 于是考虑容斥.

求出 $dp(S)$ 表示映射的点集至多为 S 时的答案, 然后就可以 $O(2^n n^3)$ 做了.

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?
大家可以感受一下:

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?
大家可以感受一下:

- $=$ 和 $\neq$

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?
大家可以感受一下:

- $=$ 和 $\neq$
- $\min$ 和 $\max$

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?
大家可以感受一下:

- $=$ 和 $\neq$
- min 和 max
- gcd 和 lcm

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?
大家可以感受一下:

- $=$ 和 $\neq$
- min 和 max
- gcd 和 lcm
- “恰好” 和 “至少”

容斥: 引入

为什么容斥会有用

容斥什么时候会起作用?
大家可以感受一下:

- $=$ 和 $\neq$
- min 和 max
- gcd 和 lcm
- “恰好” 和 “至少”
- ...

容斥: 凑系数

刚才见到的是一种最经典的容斥:

给定一些条件, 问全部满足的对象的个数.
答案 = 所有对象 - 至少不满足其中一个的 + 至少不满足其中两个的 - 至少不满足其中三个的 + ...

容斥: 凑系数

刚才见到的是一种最经典的容斥:

给定一些条件, 问全部满足的对象的个数.
答案 = 所有对象 - 至少不满足其中一个的 + 至少不满足其中两个的 - 至少不满足其中三个的 + ...

对于更加一般的容斥, 我们可以这样认为:

在所有物品中, 问在某个条件 C_0 下所有物品的贡献之和.
构造一些相对容易计算贡献的条件 $C_1, ..., C_n$, 再对于每个条件构造容斥系数 $f_1, ..., f_n$, 满足对于每个物品

$$\sum_{i=1}^{n} s(C_i) f_i = s(C_0)$$

其中 $s(C_i)$ 表示这个物品在条件 C_i 下所产生的贡献.
对于常见的计数问题, 物品的贡献只会是 $0/1$, 表示这个物品是否满足此条件.

容斥: 凑系数

这样理解有什么用呢?

容斥: 凑系数

这样理解有什么用呢?
先看经典的的错排问题

求长度为 n 的排列 $a_1, ...a_n$ 的个数, 满足 $a_i \neq i$

容斥: 凑系数

这样理解有什么用呢?
先看经典的的错排问题

求长度为 n 的排列 $a_1, ...a_n$ 的个数, 满足 $a_i \neq i$

构造 n 个条件: C_i 表示有多少个物品, 至少有 i 个位置满足 $a_j = j$.
我们用刚才的理解方式来构造容斥系数.

容斥: 凑系数

这样理解有什么用呢?
先看经典的的错排问题

求长度为 n 的排列 $a_1, ...a_n$ 的个数, 满足 $a_i \neq i$

构造 n 个条件: C_i 表示有多少个物品, 至少有 i 个位置满足 $a_j = j$.
我们用刚才的理解方式来构造容斥系数.
对于任意一个恰好有 m 个位置满足 $a_j = j$ 的排列, 需要满足

$$\sum_{i=0}^{m} \binom{m}{i} f_i = [m = 0]$$

容易看出 $f_i = (-1)^i$.

容斥: 凑系数

这样理解有什么用呢?
先看经典的的错排问题

求长度为 n 的排列 $a_1, ...a_n$ 的个数, 满足 $a_i \neq i$

构造 n 个条件: C_i 表示有多少个物品, 至少有 i 个位置满足 $a_j = j$.
我们用刚才的理解方式来构造容斥系数.
对于任意一个恰好有 m 个位置满足 $a_j = j$ 的排列, 需要满足

$$\sum_{i=0}^{m} \binom{m}{i} f_i = [m = 0]$$

容易看出 $f_i = (-1)^i$.
感觉没什么用啊?

容斥: 凑系数

其实还是有用的.
首先这是证明容斥的一种方式.

容斥: 凑系数

其实还是有用的.
首先这是证明容斥的一种方式.
然后, 它还给了我们一种 $O(n^2)$ 递推容斥系数的一般方法.(!)

容斥: 凑系数

其实还是有用的.
首先这是证明容斥的一种方式.
然后, 它还给了我们一种 $O(n^2)$ 递推容斥系数的一般方法.(!)
举个例子:

定义每个排列的价值为 a_k, 其中 k 为这个排列的错排数.
求所有排列的价值之和.

容斥: 凑系数

其实还是有用的.
首先这是证明容斥的一种方式.
然后, 它还给了我们一种 $O(n^2)$ 递推容斥系数的一般方法.(!)
举个例子:

定义每个排列的价值为 a_k, 其中 k 为这个排列的错排数.
求所有排列的价值之和.

我们只用构造 f 满足

$$\sum_{i=0}^{m} \binom{m}{i} f_i = a_m$$

容斥: 凑系数

其实还是有用的.
首先这是证明容斥的一种方式.
然后, 它还给了我们一种 $O(n^2)$ 递推容斥系数的一般方法.(!)
举个例子:

定义每个排列的价值为 a_k, 其中 k 为这个排列的错排数.
求所有排列的价值之和.

我们只用构造 f 满足

$$\sum_{i=0}^{m} \binom{m}{i} f_i = a_m$$

然后答案直接就是

$$\sum_{i=0}^{n} \binom{n}{i} (n-i)! f_i$$

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B

给定 m 个数 $a_1, ..., a_m$, 统计 $[1, n]$ 的整数中, 满足 $a_1, ..., a_m$ 中有奇数个数整除它的数的个数.

$n \leq 10^9, m \leq 15$

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B - 题解

首先肯定枚举 m 个数的一个子集, 算出 lcm, 然后容斥一下.

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B - 题解

首先肯定枚举 m 个数的一个子集, 算出 lcm, 然后容斥一下.
这不就是小学奥数题吗?

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B - 题解

首先肯定枚举 m 个数的一个子集, 算出 lcm, 然后容斥一下.
这不就是小学奥数题吗?
这道题跟小学奥数题的唯一区别就是要求 “奇数个”.

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B - 题解

首先肯定枚举 m 个数的一个子集, 算出 lcm, 然后容斥一下.
这不就是小学奥数题吗?
这道题跟小学奥数题的唯一区别就是要求 “奇数个”.
也就是要求容斥系数满足对于每个数, 如果它被 k 个数整除, 则有

$$\sum_{i=0}^{k} \binom{k}{i} f_i = k \bmod 2$$

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B - 题解

首先肯定枚举 m 个数的一个子集, 算出 lcm, 然后容斥一下.
这不就是小学奥数题吗?
这道题跟小学奥数题的唯一区别就是要求 “奇数个”.
也就是要求容斥系数满足对于每个数, 如果它被 k 个数整除, 则有

$$\sum_{i=0}^{k} \binom{k}{i} f_i = k \bmod 2$$

$O(n^2)$ 算就足够了.

容斥: 凑系数

“玲珑杯” 线上赛 Round 17 B - 题解

首先肯定枚举 m 个数的一个子集, 算出 lcm, 然后容斥一下.
这不就是小学奥数题吗?
这道题跟小学奥数题的唯一区别就是要求 “奇数个”.
也就是要求容斥系数满足对于每个数, 如果它被 k 个数整除, 则有

$$\sum_{i=0}^{k} \binom{k}{i} f_i = k \bmod 2$$

$O(n^2)$ 算就足够了.
不过也可以打表找规律:

$$f_i = [i \neq 0](-2)^{i-1}$$

容斥: 凑系数

BZOJ 4671

定义两个结点数相同的图 G_1 与 G_2 的异或为一个新的图 G, 其中如果 (u, v) 在 G_1 与 G_2 中的出现次数之和为 1, 那么边 (u, v) 在 G 中, 否则这条边不在 G 中.
现在给定 s 个结点数均为 n 的图 $G_1, ..., G_s$, 设 $S = G_1, ..., G_s$, 求 S 有多少个子集的异或为一个连通图.

$n \leq 10, s \leq 60$

容斥: 凑系数

BZOJ 4671 - 题解

连通图计数的一个经典思路就是容斥.

容斥: 凑系数

BZOJ 4671 - 题解

连通图计数的一个经典思路就是容斥.
对于这道题, 我们先用贝尔数的时间来枚举子集划分, 强制连通性“至少”是这个划分. 也就是说, 不同子集的两个点之间一定没有边, 相同子集的两个点则任意.

容斥: 凑系数

BZOJ 4671 - 题解

连通图计数的一个经典思路就是容斥.
对于这道题, 我们先用贝尔数的时间来枚举子集划分, 强制连通性“至少”是这个划分. 也就是说, 不同子集的两个点之间一定没有边, 相同子集的两个点则任意.
对于一个有 m 个连通块的图, 容斥系数需要满足

$$\sum_{i=1}^{m} \left\{ {m \atop i} \right\} f(i) = [m = 1]$$

容斥: 凑系数

BZOJ 4671 - 题解

连通图计数的一个经典思路就是容斥.
对于这道题, 我们先用贝尔数的时间来枚举子集划分, 强制连通性“至少”是这个划分. 也就是说, 不同子集的两个点之间一定没有边, 相同子集的两个点则任意.
对于一个有 m 个连通块的图, 容斥系数需要满足

$$\sum_{i=1}^{m} \left\{ {m \atop i} \right\} f(i) = [m = 1]$$

打表发现规律

$$f(i) = (-1)^{i-1}(i-1)!$$

技巧: 平方处理

BZOJ 1566

已知现在有两个 “输入管道”, 分别有 n, m 个珠子. 每个珠子是黑色或白色, 相同颜色的珠子被视作是相同的. 每个珠子的颜色也是已知的.

你每次可以从任意一个输入管道中取出最前面的那个珠子放入输出序列. 一共要取 $n+m$ 次, 将这些珠子都取完, 顺序任意.

现在定义两种取法不同, 当且仅当某一次操作中两种取法是在不同的管道里取的. 显然即使取法不同也有可能输出序列相同.

设最终可能产生的不同的输出序列有 k 种, 产生第 i 种输出序列的方案有 a_i 个, 现在要求:

$$\left(\sum_{i=1}^{k} a_i^2\right) \bmod 1024523$$

$n \leq 500$

技巧: 平方处理

BZOJ 1566 - 题解

统计平方的和, 转化成统计有序对.

技巧: 平方处理

BZOJ 1566 - 题解

统计平方的和, 转化成统计有序对.
即统计有多少对 (way_A, way_B) 使得 way_A, way_B 均能得到相同的结果.

技巧: 平方处理

BZOJ 1566 - 题解

统计平方的和, 转化成统计有序对.
即统计有多少对 (way_A, way_B) 使得 way_A, way_B 均能得到相同的结果.
$O(n^3)$ DP 即可.

技巧: 反射法

经典问题 3

给定 S, T, K, 求每次 $+1, -1$, 用不超过 K 次操作从 S 变成 T 的方案数. 每一时刻都不能为负.

$S, T, K \leq 10^5$

技巧: 反射法

经典问题 3 - 题解

在平面直角坐标系中画出图像 (x 轴代表时间, y 轴代表当前的数值), 发现所有不合法的路径都可以沿 $y = -1$ 反射到一条从 $(-(S+2), 0)$ 到 $(T, 0)$ 的路径.

技巧: 反射法

经典问题 3 - 题解

在平面直角坐标系中画出图像 (x 轴代表时间, y 轴代表当前的数值), 发现所有不合法的路径都可以沿 $y = -1$ 反射到一条从 $(-(S+2), 0)$ 到 $(T, 0)$ 的路径.

直接组合数计算即可. 当然直接减一下转化为不能穿过对角线也是一样的.