2021 Nowcoder Training
Chenjb and His Friends' Contest

2021 年 8 月 16 日

前言

首先对出题人及验题人表示感谢，包括且不限于：Quailty, Tangjz, Claris, Subconscious, Zyb, Acesrc, Suika_Predator, Heltion, Tommyr7, Orenji.Sora, UESTC_Nocturne，以及浙江大学 2021 届校队。特别感谢 Quailty 提供了非常大的帮助。

在组题的时候，我和 qls 就发现这套题对于国内这么大型的训练赛来说，必然是难度过高的，难免出现前排美如画，后排靠手速的情况。但是如果我们对标东欧训练营题，或者 EC-Final、OpenCup，这套题无疑从区分度还有难度、知识点上都比较合适。这也得益于出题人们为这套题提供了非常优质的题目，使得我们在筛选题目的时候都仿佛从身上割肉一般。于是，作为牛客多校的最后一场，我们觉得还是值得让大家体验一下，见识一下，于是最后就让大家做到了一场这样的比赛。

在整个比赛过程中，实际上除了前排的几个队伍，中后排队伍的表现比我们想象中还要不理想。我们深深感觉到自己已经和这个世代的选手们脱节了。一些我们以前天天强调的做题策略和习惯，现在仿佛越来越难以在选手身上看到。更甚，一些戾气和不良习惯则从中滋生。举几个例子，比如看不懂 F 或者 H 题，实际上我们也担心选手理解造成了困扰，所以要么提供样例解释，要么尽量写得浅显易懂，而不是加上一堆的无聊背景绕人耳目。甚至当我们将原文放进 google 翻译时，得到的中文题目直接就是能读懂的。

再比如有人抱怨“全场都是数学题”，我在想，但凡是一支训练有素的队伍，有强调过读题习惯，跟榜习惯的队伍，一定能发现诸如 A、E、F、J 这样的题目其实也非常可做。对于 J 这样的题目，要是我把 J 拆成两道题，第一题是求凸包切线，第二题是

求最小环覆盖，我猜过题人数翻 5 倍不止。你会说“这不是模板题/原题吗”，那大家不是很喜欢贴板子，以找到原题为傲吗？那怎么两个东西贴一起就搞不定了？当然，这样的观点无疑是偏激而苛责的。但我想引用叉姐出 CCPC FINAL 的时候对我说的一句话：“我们这个比赛是 Programming(编程) 比赛，不是 Algorithm(算法) 比赛，更不是 Mathematic(数学) 比赛”。当你对诸如 A，F 这样的题束手无策的时候，我觉得比起学习新奇的算法，倒不如想想，如何提高自己的建模能力和代码能力，我一直觉得这些东西才是最应该得到锻炼的东西。

我希望大家都扪心自问一下，你们平日里训练赛的目的到底是什么？是拿到小奖品，提高 rating，还是磨合队伍，积累经验，提高实力？我们甚至发现在比赛过程中有个别队伍可能是从 cf gym 上复制了原题 (J 题简化版) 的复杂度较优的代码过来提交并得到了 AC，这种行为对于训练赛来说是毫无意义的，与初衷背道而驰。

所谓忠言逆耳，作为一位出题人，我一直都很希望选手能够享受我出的比赛，能够有所收获，也从来不希望把自己放在选手之上，只顾自己的爽快。但我们这帮老人，随着年龄增长，自身发展的原因，即使再不情愿，也会慢慢地远离这个圈子。我们非常希望能够利用有限的机会，将我们的一些经验和心得留下来，帮助后辈们提高自己的水平，能够体会到我们当年的那种畅快感，若如此，则无怨无悔。

我们衷心地希望各位选手能够通过这场训练赛更加正视自己的水平，认识到自己与优秀队伍之间的差距所在，以更加开放的心态认真地对待每一场训练赛，争取在未来的正式比赛中取得更好的成绩。

A. Browser Games

加强自 2020 年 ICPC 银川赛区 K 题

首先我们不考虑空间限制，先对所有字符串建立一棵字典树，并将每个串对应字典树上的节点标为黑点，每次操作会将一个黑点变为白点，目标是将尽可能少的非根节点标红使得任意一个白点到根的路径上都有被标红的节点，且任意一个黑点到根的路径上都没有被标红的节点。我们记录下每个点的子树里有多少个黑点，每次有一个黑点变为白点的时候，首先标红这个白点，然后我们将所有的红色标记尽可能往上推就能得到最少被标红的节点数。

为了优化空间，我们观察到字符串只有 n 个，比字典树的节点数少一个串长的系数，如果我们能缩掉只有一个儿子的非关键点，只保留 n 个关键点以及字典树上的分叉节点，就能得到只有 $O(n)$ 个节点的经过压缩的字典树，从而通过此题。我们考虑自顶向下地递归构建这棵树，并且在构建的过程中对 n 个字符串进行排序。当我们位于某个节点时，维护节点的深度信息，以及当前节点的子树中的字符串，注意到在对所有字符串排序之后，这些字符串的下标会构成一个区间，我们可以通过深度信息得到这些字符串对应位置的字符，并使用桶排序完成对区间的划分，如果能划分出多个区间则递归下去构建各个子树，否则不需要新建节点，当某个节点只剩一个串时即可退出递归过程。

B. Child' s play

考虑一个状态数为 m^{n-1} 的 dp：令 $f(\{a_1, a_2, a_3, ..., a_{n-1}\}) = t$ 表示当前我看到别人的数字分别是 $\{a_1, a_2, a_3, ..., a_{n-1}\}$ 时，我可以在别人连续 t 轮无法判断他的 rank 后确定我自己的 rank。

转移是显然的，我们考虑枚举自己的数字所在的区间 I

$$\begin{aligned} f(\{a_1, a_2, a_3, ..., a_{n-1}\}) = second\_max_I\{\max_{x\in I}\{\min\{&f(\{x, a_2, a_3, ..., a_{n-1}\}),\\ &f(\{a_1, x, a_3, ..., a_{n-1}\}),\\ &\cdots,\\ &f(\{a_1, a_2, a_3, ..., x\})\}\}\} + 1 \end{aligned}$$

其中，$second\_max$ 表示第二大的区间。

考虑将这个 dp 优化到 $O(n^3)$。

1. 简单思考后会发现，表达式中的 $\max_{x\in I}$ 是可以简化的，因为只有当你的数取为 $sup(I)$ 或 $inf(I)$ 时后面表达式才有可能取最大值。

2. 我们将这个集合的状态用一个二进制字符串来表示，第 i 位为 1 表示有一个你以外的玩家持有着这个数，第 i 位为 0 表示不存在你以外的玩家持有着这个数。

如：$m = 6$ 时，$f(\{1, 4, 5\}) = f('100110')$

我们考虑上例中持有 4 这个数的玩家，对他来说有意义的状态只有 $f('110010')$，$f('100011')$

状态可以被 $O(n^2)$ 记录，然后 dp 即可。

C. Dance Party

题目大意

给一个二分图 $G = \langle X, E, Y\rangle, (X, Y$是两个点集$, E$是边集$,$且$|X| = |Y| = n)$。

X 中的每个点只与 Y 中的不超过 k 个点**没有边**。

求 G 的一个完美匹配，如果不存在输出 -1

$n \leq 30000, k = 100$

题解 1

做法可以分成 3 部分：

1.

首先，对 Y 中的点按照度数从小到大排序，并且重编号为 $1, 2, 3, \cdots, n$

找到**最小的** t，满足 $\deg(t) \geq k + (t - 1)$。(deg 是点的度数)

考虑不存在的边的数量，这个最小的 t 满足这个不等式：

$$\sum_{j=1}^{t-1} \max(0, n - (k + t - 2)) \leq k \times n \tag{1}$$

这个 t 非常的小，在 $k = 100$ 时，$t \leq 370$。对于度数较小的前 $t - 1$ 个点，它们的度数都小于 $k + (t - 1)$，用网络流进行暴力匹配，如果无法匹配，则输出-1

因为 $t = O(k)$，因此这部分的时间复杂度约为 $O(k^{2.5})$

2.

将上一步中成功匹配的点对从图中删去，此时剩余未匹配的图 $G = \langle X', E', Y' \rangle$，设 $m = |X'|$，满足 $\forall v \in X', \deg(v) \geq m - k$ 且 $\forall v \in Y', \deg(v) \geq k$，对于这样一个二分图，可用霍尔定理证明一定存在完美匹配，问题就转变成了如何构造一组完美匹配。

先在 G' 中进行贪心匹配 (按顺序枚举 X' 中的每个点，在 Y' 中挑选一个能与它匹配且仍未被匹配的点进行匹配)，设最后成功匹配了 s 对点 $x_1 - y_1, x_2 - y_2, \cdots, x_s - y_s$，两个点集中各剩下 $m - s$ 个点未被匹配 $\{x_{s+1}, x_{s+2}, \cdots, x_m\}, \{y_{s+1}, y_{s+2}, \cdots, y_m\}$

因为 $m \leq n$，所以这部分的时间复杂度是 $O(nk)$

3.

对于 $s + 1 \leq j \leq m$ 的每个 j，都能够挑选出一个 i，满足 $1 \leq i \leq s$，x_j 可以和 y_i 匹配且 x_i 可以和 y_j 匹配。

因为 $m - s \leq k$，所以这部分的时间复杂度是 $O(nk)$

整个做法的总复杂度是 $O(nk + k^{2.5})$

题解 2

考虑优化匈牙利算法，记 $k = 100$。前 $n - k$ 个点可以按顺序直接匹配，不需要反悔，或者说增广路长为 1，所以简单 set 维护即可。最后 k 个点，模拟匈牙利算法：首先把待匹配的女士点的出边全部访问一遍，这样就只有 k 个点（即待匹配的女士不想匹配的男士）没有访问过，所以之后找出边的时候只需要考虑每个点的出边和那 k 个点的交集。于是，除了待匹配点，每个其它点的有意义的出边最多只有 k 条。这样的话，总边数是 $O(nk)$ 级别的，直接套用匈牙利算法即可。因为要对 k 个点求增广路，总复杂度是 $O(nk^2)$。

D. Diameter Counting

我们先来了解一个算树直径的算法.

这个算法会执行若干轮, 每一轮要做的事情就是先找出该树目前有哪些叶子, 然后把列出来的节点全部删除.

假设执行了 x 轮删除, 树变成只剩一个点或者两个点.

如果是只剩下一个点, 那么原本树的直径为 $2x$. 如果剩下两个点, 那么原本树的直径是 $2x+1$.

这里的原理是每一轮将所有叶子删除之后, 树的直径会减少 2.

我们将这个想法反过来, 从一个核心开始 (一个点或两个点的结构被称为核心), 执行若干轮加叶子操作, 就能造出指定直径的树.

每轮加叶子的要求是什么呢? 要保证原来的旧叶子都必须被添加一个以上的新叶子.

令 $f[i][j]$ 表示有 j 个叶子的 i 标号树的个数.

当我们想执行一轮添加新叶子操作时, 我们枚举添加 k 个新叶子, 这样 $f[i][j]$ 就能转移到 $f[i+k][k]$.

但是该怎么转移呢? 现在有这样一个子问题:

我们需要构建一个长度为 k 的子序列 (这里的物理意义是将新叶子的父亲节点按新叶子编号写下来得到的序列, 该序列与加叶子方案一一对应), 其中序列中元素的取值有 i 种可能, 但有 j 种取值必须出现 (要保证 j 个旧叶子必须被至少一个新叶子连接). 记这个的方案数为 $g[i][j][k]$.

有了 g 我们可以方便地计算 f: $f[i+k][k]+=f[i][j]\cdot g[i][j][k]\cdot C(i+k,i)$. 最后的 $C(i+k,i)$ 指的是从 $i+k$ 里面选出 i 个节点构成原本的 i 标号树.

现在问题变成了怎么计算 $g[i][j][k]$. $g[i][j][k]$ 的计算我们可以用序列 dp 的想法.

考虑序列中第 k 个元素. 分类讨论一下.

1. 如果这个元素不是全新的特殊值 (特殊值就是必须出现一次的值), 意思是这个元素要么是出现过的特殊值, 要么就不是特殊值, 那么去掉这个元素我们能得到一个类似的子问题结构. 所以这种情况下方案数是 $i\cdot g[i][j][k-1]i$.

2. 如果这个元素是全新的特殊值, 意思是在之前已经出现过 $j-1$ 个特殊值, 这也变成了一个子问题. 所以这种情况下方案数是 $j\cdot g[i-1][j-1][k-1]$.

两种情况合并一下就是 $g[i][j][k]$.

现在我们有了 f 和 g, 接下来怎么求解最终我们要的直径的和呢? 我们再引入一个量 $h[i][j]$, 表示有 j 个叶子的 i 标号树的直径和.

同样的我们枚举新加 k 个叶子, 考虑 $h[i][j]$ 对 $h[i+k][k]$ 的贡献. 他的贡献其实分两部分.

比如有一棵树 1 - 2 - 3 - 4, 在加了新叶子之后变成 5 - 1 - 2 - 3 - 4 - 6, 那么 1 到 4 的路径就是旧树原本的直径, 5 和 6 这两个叶子所产生的是新叶子的固定为 2 的新叶贡献.

1. 旧树直径的贡献: 一棵有 j 个叶子的 i 标号树在添加 k 个新叶子之后, 会衍生出

$g[i][j][k]$ 棵新树, 意味着它的旧树直径贡献会被放大 $g[i][j][k]$ 倍. 所以这部分的计算是 $h[i+k][k]+=h[i][j]\cdot g[i][j][k]$.

2. 新叶贡献: 由于新叶的贡献是固定的 2, 所以有多少种新树, 就有相应数量的新叶贡献. 这部分的计算是 $h[i+k][k]+=2f[i+k][k]$.

得到 h 之后我们就能获得最终答案了.

所有数组的计算都是 $O(n^3)$ 的.

E. More Fantastic Chess Problem

题意

问 K 维象棋中给定位置的王/后/车/象/马, q 次移动, 每次移动前后能攻击到几个格子

题解

没有移动的情况

车显然

王只和每维是否贴边界、贴几个边界有关，预处理 2^i 和 3^i 算贡献

后可以枚举长度，然后转换为王的情况

象也是枚举长度，维护出每个长度有多少维有一个方向，有多少维有两个方向，组合数算一下再减掉两个方向的维数

马是选一个长度为 1 的维度，再选一个长度为 2 的维度，再把选到同一维的情况减掉

有移动的情况

拿线段树维护所有长度的 C_{x+y}^2 之和、2^x3^y 之和、y 之和就可以支持单维度修改了，其中 x 是只有一边离边界距离 $\geq len$ 的答案，y 是两边离边界距离都 $\geq len$ 的答案。

F. Train Wreck

将栈操作视为树，要求转化为给每一个节点染色，使得从根到每一个节点的链所构成的颜色序列两两不同。注意到两个都在第 i 层的节点，如果它们的父亲不同，则从根

到它们父亲的链所构成的颜色序列不同，这使得从根到它们的链所构成的颜色序列一定不同。(因为删去序列最后一项后得到的序列不同）因此，条件可以转化为每个点的各个儿子颜色互不相同。对于一个节点，假设其有 k 个儿子，我们选取剩余火车数最多的 k 个颜色对应上去即可。

G. Game of Death

来源

`https://www.zhihu.com/question/472917138`

题意

n 人围成一圈, 每个人独立随机地选择一名其他玩家并向其开一枪, 开枪是同时的, 被命中者立刻阵亡退出游戏. 假设所有人的命中概率都为 p, 问还剩 $k = 0, 1, \cdots, n$ 个人的概率.

题解

记 $f(S)$ 为被击中的集合恰为 S 被击中的概率,$g(S)$ 为被击中的集合是 S 子集概率. 那么

$$f(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} g(T).$$

显然 f 和 g 对于所有相同大小的集合 S 值都相等, 可以求出

$$g(S) = (1 - p + \frac{p(|S| - 1)}{n - 1})^{|S|} (1 - p + \frac{p|S|}{n - 1})^{n-|S|},$$

那么

$$f(S) = \sum_{i=0}^{|S|} (-1)^{|S|-i} C_{|S|}^{i} g(i),$$

可以用 FFT 求出.

H. War of Inazuma (Easy Version)

可以发现 n 维超立方体是一个二分图，直接按照二进制表示中 1 出现次数的奇偶性进行黑白染色即可，时间复杂度 $O(2^n \times n)$

I. War of Inazuma (Hard Version)

假设 $m = \lceil \sqrt{n} \rceil$.

我们可以将每个 $[0, 2^n - 1]$ 中的数 i 写成一个大小为 $m \times m$ 的 01 矩阵。具体做法如下：

将数 i 写成一个长度为 n 的 $0-1$ 串（即其二进制表示）

在该 $0-1$ 串的结尾补上 $m^2 - n$ 个 1 ，使其变成一个长度为 m^2 的 $0-1$ 串

将该 $0-1$ 串每 m 个一行分成 m 行，成为一个大小为 $m \times m$ 的 $0-1$ 矩阵

i 号点被染成黑色当且仅当以下两种情况中有一个成立：

1. 数 i 对应的 $0-1$ 矩阵中恰好有奇数个 1 且前 $\lceil \frac{n}{m} \rceil$ 行不存在一行全为 1（假设这样的数 i 有 A 个）

2. 数 i 对应的 $0-1$ 矩阵中恰好有偶数个 1 且前 $\lceil \frac{n}{m} \rceil$ 行存在一行全为 1（假设这样的数 i 有 B 个）

i 号点被染成白色当且仅当以下两种情况中有一个成立：

1. 数 i 对应的 $0-1$ 矩阵中恰好有奇数个 1 且前 $\lceil \frac{n}{m} \rceil$ 行存在一行全为 1（假设这样的数 i 有 C 个）

2. 数 i 对应的 $0-1$ 矩阵中恰好有偶数个 1 且前 $\lceil \frac{n}{m} \rceil$ 行不存在一行全为 1（假设这样的数 i 有 D 个）

容易发现，对于每个点 u ，和它同色且相邻的点最多只有 m 个

同时，$A + C = B + D = 2^{n-1}$ 为偶数，$A + D$ 为奇数，因此 $A + B$ 为奇数，则 $A + B \neq C + D$

时间复杂度 $O(2^n \times n)$

J. Illuminations

加强自 2018 年 ICPC 徐州赛区 M 题

首先观察到每个点光源能照亮凸多边形上连续的一段边，可以通过求解点到凸多边形的切线得到照亮区间，转化为在环上用最少的区间覆盖整个环的问题。

我们分别求解这两个子问题。

对于（凸多边形外的）点到凸多边形切线的求解，一个较为通用的做法是转化为凸多边形上的最值求解，参考 https://blog.csdn.net/u013279723/article/details/104239723。另一个基于分类讨论的做法是，记询问点 $P\ (x_0, y_0)$，先找出凸多边形横坐标最小的顶点 $Q_1\ (x_1, y_1)$ 和横坐标最大的顶点 Q_2，接下来分情况讨论：

- 如果 $x_0 \le x_1$，那么 P 到凸多边形的两条切线的切点会分别落在以 Q_1 和 Q_2 分界的两段边上，在两段边上分别二分切点即可；
- 如果线段 PQ_1 与凸多边形有异于 Q_1 的交点 R，那么 P 到凸多边形的两条切线的切点会分别落在以 Q_1 和 R 分界的两段边上，先二分出离 R 最近的凸多边形顶点，然后在两段边上分别二分切点即可；
- 否则 Q_1 就是一个切点，在整个凸多边形上二分另一个切点即可。在某些实现方式下这种情况可以与前一种情况合并，此时二分出的 R 恰好就是 Q_1。

由于本题保证所有光源不会落在任意一条凸多边形边的延长线上，不需要再处理切线与某条边重合的情况。

对于环上区间覆盖问题的求解，首先破环成链，考虑完全覆盖某个长为 n 的区间 $[i, i+n)$ 所需要的最少区间数。我们可以先去除相互包含的区间，因为使用被其他区间所包含的区间不会更优，处理后余下的 $O(n)$ 个区间的右端点关于左端点是单调的，那么我们可以枚举选择的第一个区间，然后贪心地选择与当前区间有交的区间中往右延伸最远的区间作为下一个区间，直到长为 n 的区间被覆盖，从而得到一个 $O(n^2)$ 的做法。

为了优化复杂度，我们观察到选择一个区间后，贪心选择的下一个区间是唯一的，因此可以使用倍增算法，求出从每个区间开始选择 2^k 个区间之后能覆盖到的最右点，从而在枚举第一个区间后在 $O(\log n)$ 时间内求出覆盖长为 n 的区间所需要的区间数。另一个做法是，仍然观察到贪心选择区间的过程中每个区间的后继唯一，贪心选择的区间会构成树上的一条链，dfs 这棵的时候维护一个栈，在栈上二分即可。

K. Walking

首先我们可以发现两个维度可以分开.

令 $X(i)$ 为横向走 i 步不出界的方案数, $Y(i)$ 为纵向走 i 步不出界的方案数, 那么答案为 $\sum_i X(i)Y(t-i)C(t,i)$.

现在问题转变成子问题: 给定 n,t,x, 考虑一个数轴, 从 x 出发, 走 t 步, 要求不能走出 $[1,n]$ 范围的合法方案数 $K(t)$. 要求对 $1 \le t \le T$ 都求出 $K(t)$

Step 1

这个子问题有个显然的动态规划解法

令 $f(i,j)$ 为走 i 步之后到达点 j 的方案数. 那么 $K(t) = \sum_{1 \le j \le n} f(t,j)$.

而 $f(i,j)$ 的计算也非常简单. 不考虑 $n=1$ 的平凡情况.

当 $1 < j < n$, $f(i,j) = f(i-1,j-1) + f(i-1,j+1)$.

$f(i,1) = f(i,2)$.

$f(i,n) = f(i,n-1)$.

f 的计算复杂度是 $O(Tn)$, 当 T 或者 n 比较小的时候是不错的解法.

Step 2

我们从另一个角度思考.

我们直接考虑 K.

我们发现所有的 $f(i,j)$ 在 $K(i)$ 中的贡献系数是 1.

而对如果将 $f(i+1,j)$ 展开成 $f(i,j-1), f(i,j+1)$ 的形式, 我们可以发现大部分 $f(i,j)$ 对 $K(i+1)$ 的贡献系数是 2.

唯二的例外是 $f(i,1)$ 和 $f(i,n)$, 他们的贡献系数是 1.

也就是说 $K(i+1) = 2K(i) - f(i,1) - f(i,n)$.

所以我们只要求 $f(i,1), f(i,n)$ 这两个特殊值即可.

下面只考虑 $f(i,1)$ 的计算, $f(i,n)$ 同理.

假设没有任何限制地走, 从 x 走 i 步到达 y 的方案为 $g(y)$, 这是一个组合数值, 非常好算.

那么如果要从 $g(1)$ 算出 $f(i,1)$, 我们就要减去不合法的情况. 这里就需要用到容斥原理.

假设点 u 关于左边界 -1 的对称点为 $L(u) = -2-u$, 关于右边界 $n+1$ 的对称点为 $R(u) = 2(n+1)-u$.

稍微容斥一下我们可以得到:

$f(i,1) = g(1) - g(L(1)) - g(R(1)) + g(L(R(1))) + g(R(L(1))) - g(L(R(L(1)))) - -g(R(L(R(1))))...$

由于每次做对称会导致目标点到 x 的距离至少增加 n, 所以这个容斥式子直接计算复杂度是 $O(\frac{i}{n})$.

在这个做法下计算 K 的复杂度是 $O(\frac{T^2}{n})$

Step 3

把两个结合一下, 当 n 比较小的时候用动态规划, n 比较大的时候用第二种做法, 均衡系数之后就得到 $O(T\sqrt{T})$ 的做法.