Zayin and Bus 题解

不考虑不合法的限制，直接贪心得到的结果是合法的。

所有节点按深度排序，所有人按 $a[i] + i$ 排序，大小配对。

如果存在一个子树，其根上的人挡住了下面的某个节点的人进入。那么交换他们的目标点会使子树答案更优。而贪心可以保证对于任意一个子树都满足在不改变子树中的人的前提下，得到的方案是最优的。所以最优解是合法解。

Zayin and Elements 题解

一般图最大匹配

对于每个道具的每单位的 a，新建一个点 A，每单位的 b，新建点 B1，B2，每单位的 c，新建点 C1，C2

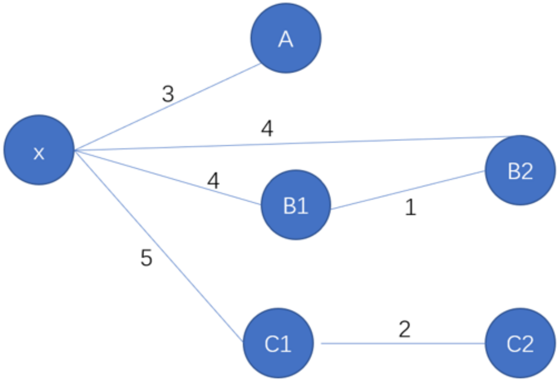
对于每个元素，新建一个点

假设某个 a_i, b_i, c_i 均大于零的道具可以作用于某个元素 x 上，x 与每单位的 a, b, c 有 x 与 A，x 与 B1，x 与 B2，x 与 C1，B1 与 B2，C1 与 C2 相连

然后按下图标号顺序从小到大匹配

带花树在匹配数不增加时不会丢弃之前能连上的边，所以会尽量满足 B1 与 B2，C1 与 C2 的边，而且也能按题意匹配

最后 B1 与 B2 以及 C1 与 C2 之间的边数就是答案



Zayin and Fireball 题解

计算每个圆的有效边界，通过 $\int ydx$ 将面积积出。

Zayin and Forest 题解

首先不难证明 F(x)=x+lowbit(x)
每次 Add 操作相当于在 log 个位置加 v
另一个操作是区间求和
不难想到哈希表 + 树状数组的做法

首先我们不难写出这样的两个 log 的代码（a 是哈希表）

for(;x<=n;x+=x&-x) for(LL y=x;y<=n;y+=y&-y) a[y]+=v;

注意到两个地方都是加 lowbit，所以它等价于以下代码

for(LL d=v;x<=n;x+=x&-x,d+=v) a[x]+=d;

这样就只需要一个 log 了

这部分只有一个 log 的话，不需要哈希表，用 unordered_map 就可以了

Zayin and Camp 题解

题目大意

要求你求出有多少有序的序列对 (A, B) ，满足

- A,B 中的每个元素要么是 $+1$, 要么是 $-m$
- A,B 中为 $-m$ 的的个数总共为 n。
- A 中所有数的和为 r，B 中所有数的和为 s
- A,B 序列的所有位置的前缀和都大于 0

题解

不妨只考虑一个序列 A、有 n 个 $-m$、序列总和为 r 的情况。此时序列总长度一定为 $len = n(m+1)+r$，有 $nm+r$ 个位置是 $+1$。

可以观察到，对于任意一个排列，它的所有 len 个循环左移同构序列中，一定有 r 个是满足“所有前缀和都 >0”的条件的。

证明

我们将序列 A 进行无限循环延拓：

$$A' = \{a_1, a_2, \ldots, a_{len}, a_1, a_2, \ldots\}$$

，并记

$$pre[n] = \sum_{i=1}^{n} a'[i]$$

记 $last_i = max\{j|pre[i] == j\}, i \in [1, r]$，也就是对于前缀和值 $1, 2, ..., r$，我们都记下它最后出现的位置。

因为 $lim_{i\to\infty} pre[i] = \infty$，所以 last 值一定存在。

证明

可以看到，以 $last_i$ 为起点的 A′ 的子区间：$A'[last_1, last_1 + len), ..., A'[last_r, last_r + len)$，都是前缀和都 >0 的区间，都对应与原序列 A 的某个循环左移区间。

另一方面 $\forall i \neq j, last_i, last_j$ 对应的区间也不可能相同

(不然他们对应位置前缀和就会相差 r 的倍数，$pre[last_i] == pre[last_j] + c \times r, pre[last_i + 1] == pre[last_j + 1] + c \times r, \ldots$)

单个序列的情况

所以对于一个序列，满足“所有前缀和都 >0 “的序列个数，就为 $\binom{len}{n} \times \frac{r}{len}$

两个序列的情况

可以考虑枚举分 A 序列有 x 个 $-m$,B 序列有 $n-x$ 个 $-m$,用前述公式求卷积即可。

但其实两个序列的情况完全可以拼成一个新的大序列，其中 $-m$ 的位置有 n 个，序列总和为 $r+s$。对于某个合法的大序列 C，找到其前缀和 $pre[]$ 值为 r 的最后的位置 x 并从此处断开：$A=C[1:r], B=C[r+1, len]$

这个大序列的方案和原来的 AB 有序对一一对应。所以答案也就是这个大序列的合法方案数 $\binom{n(m+1)+r+s}{n} * \frac{r+s}{n(m+1)+r+s}$。

预处理阶乘之后每个询问就可以 $O(logn)$ 回答了。

Zayin and Dirichlet 题解

由多个函数 $f_1, f_2, \cdots, f_k$ 卷起来得到 f，问 $f(p^c)$，这相当于把 c 个 p 分配到这 k 个函数上，分别算然后乘起来，最后所有的分配方案得到的值再加起来。

μ 和 1 是不共存的，他们在狄利克雷卷积下互为逆函数（特判 $f(p^c) = 0$ 时由 $\mu * 1$ 得到)。接下来先考虑只有 1 和 id_k 的情况。(1 可看作 id_0)

首先，考虑在众多的 id_k 里面所用到的最大的 k 是多少（假设是 m)。考虑把 p^c 全部分配到 id_m，则会产生 p^{cm} 这一项，因此 $m = \frac{n}{c}$。若不整除则无解。

假设 id_0 到 id_m 使用的个数分别为 $num_0, \cdots, num_m$，接下来从高位往低位依次确定。

考虑把 p^c 全部分配到 id_m，那么 p^{cm} 的系数就是 $\binom{c+num_m-1}{num_m-1}$，因此就是解一个方程 $\binom{c+num_m-1}{num_m-1} = a_n$。又由于 $num_m \le 10^5$，可以枚举来确定。

然后逐次考虑 i 从 $m-1$ 到 0。把 p^{c-1} 分配到 id_m，把 p 分配到 id_i，那么会贡献到 $a_{m(c-1)+i}$ 这一项。而这一项其余的贡献一定来自于比 i 大的 id，不可能来自比 i 小的 id，因此可以用一个 DP 来算出 id_{i+1} id_m 对这一项的贡献，假设是 x，那么就会有方程 $\binom{c-1+num_m-1}{num_m-1} \cdot num_i + x = a_{m(c-1)+i}$，因此可以直接算出 num_i。

这个 DP 可以这样算，设 $f_{i,j,index}$ 表示考虑了 id_i id_m，已经分配了 j 个 p，对应到多项式的第 $index$ 项，所造成的贡献。转移就是枚举 id_i 分配多少个 p。上述的 x 就是 $f_{i+1,c,m(c-1)+i}$。这个 DP 的复杂度总共是 $O(mcn \cdot c) = O(n^2c)$。

可以看到，正常情况下基础函数的使用方法其实是唯一的。

至于求答案，答案已经被 DP 出来了。

然后考虑如果最低位的函数不是 id_0 而是 μ 的话会发生什么，其实对于 num_1 num_m 的计算是没有影响的，对于 $i = 0$ 时的 DP 会有点不一样，若 $m = 0$ 则要单独枚举全由 μ 卷起来的情况（注意这里要取最小解)。

最后还有一个挺强的 corner case，就是在模 998244353 意义下组合数的结果是会重的，即枚举 num_m 的时候是可能有多解的。但打表发现对于一个 k 来说 $\binom{x}{k}$ 的值最多重复 3 次，因此对于每种解都求一次答案即可。

Zayin and Count 题解

Zayin 和 Taotao 从 0 开始数数，他们会忽略掉含有自己不认识的阿拉伯数字的数字，且他们至少认识两个阿拉伯数字，已知 Zayin 在 x$(x \leq 2^{64})$ 秒写下的数字，问此时 Taotao 写下的数字是?

这是一道类似进制转换的题，当涛涛和洋洋认识 0 时，就是一道普通的进制转换，考虑 Zayin 不认识 0、只认识 1 和 2 的情况，他会依次写下 1、2、11、12、21、22、111，很容易发现这种时候，我们不能直接当进制转换做，要稍微计一下数。

Zayin and Obstacles 题解

题意很简单，就是给 $N \times N \times N$ $(N \leq 100)$ 的三维网格图，给定起点终点和障碍物的位置，求从起点到终点最少要经过多少个方格。

障碍物的位置以长方体顶点的形式给出，所以我们可以通过三维前缀和的方法得到每个方格有没有被长方体覆盖。

知道每个位置的方格有没有障碍物之后 BFS 就可以得到答案。

单组数据时间复杂度 $O(N^3)$。

Zayin and Coin Game 题解

首先我们思考一个问题：给定 n 和 k，能不能从全 0 变成全 1

经过思考，我们不难证明，能的充要条件是 $k/gcd(n,k)$ 是奇数

情况 1：$k/gcd(n,k)$ 是奇数

这时我们可以有这样一个操作：翻转连续的 n 个硬币

如果我们既可以翻转连续 a 个硬币，也可以翻转连续 b 个硬币，那我们就可以翻转连续 $|a-b|$ 个硬币

根据数论知识可推出，我们可翻转连续 $gcd(a,b)$ 个硬币

所以，我们可以翻转 $gcd(n,k)$ 个硬币

因为 n 和 k 都是 $gcd(n,k)$ 的倍数，所以等价于只有翻转连续 $gcd(n,k)$ 个硬币的操作

我们还不难发现，如果把环断开成链，跨越断点的操作可以由许多不跨越断点的操作得到

所以现在我们就只有翻转不跨越断点的连续 $gcd(n,k)$ 个硬币的操作了

直接模拟即可

情况 2：$k/gcd(n,k)$ 是偶数

这时我们就不能由全 0 变成全 1 了

如果我们强行增加一个翻转连续 n 个硬币的操作（这个称为 n 操作，原来的操作称为 k 操作），那我们用 n 操作和 k 操作组合得到 $gcd(n,k)$ 操作，其中 n 操作一定用了奇数次（否则就抵消了，相当于我们可以从全 0 变成全 1，这和 $k/gcd(n,k)$ 是偶数矛盾）

如果我们把其中的 n 操作去掉，只保留 k 操作的话，我们组合得到的就是翻转连续 $n-gcd(n,k)$ 个硬币的操作，也就是先翻连续 $gcd(n,k)$ 个，再把全部硬币翻一遍

同时，因为 $k/gcd(n,k)$ 是偶数，所以 $n/gcd(n,k)$ 是奇数

我们考虑 $n-gcd(n,k)$ 操作的补集 $gcd(n,k)$，跨越的也可以分解成不跨越的

所以现在等价于：有翻转不跨越断点的连续 $gcd(n,k)$ 个硬币的操作，但必须用偶数次

直接模拟即可

Zayin and Tree 题解

路径中的最大值和最小值一定在两端

Zayin and String 题解

题目要求 S 串的一个子序列 T，使得其中子串的甜蜜值的和与长度的比值最大，即最大化

$$\frac{\sum_{l=1}^{|T|}\sum_{r=l}^{|T|} love(T(l,r))}{|T|}$$

这是一个经典的 01 分数规划问题，我们二分答案 mid ，把长度乘到左边，就变成判断 S 串中是否存在一个子序列 T 使得

$$\sum_{l=1}^{|T|}\sum_{r=l}^{|T|} love(T(l,r)) - |T| * mid > 0$$

我们可以对字典的字符串构建一个 AC 自动机，并通过在 AC 自动机上 dp 解决这个问题。定义状态 $f[i][j]$ 表示字符串 S 前 i 位构成的子序列，最后一位在自动机的位置为 j 的最大甜蜜度值。定义 $g[i]$ 表示以自动机位置 i 为结尾的单词的甜蜜度的和。

转移的时候，考虑第 $i+1$ 位选或不选，如果选的话就在自动机上移动到对应位置 pos ，并加上 $g[pos]-mid$ 的贡献即可。

时间复杂度 $O(n * \sum |w| * log)$

Zayin and Raffle 题解

为方便描述，题解中的下标均从 0 开始。

我们用 $\mathrm{id}(i)$ 表示一个长度为 2^m，第 i 项为 1，其余项为 0 的向量. $0 \le i \le 2^m - 1$

那么题目可以转化成，给出 n 个向量，第 i 个向量 $v_i = \sum_{j=0}^{k-1} p_j \mathrm{id}(a_{ij})$. 求这 n 个向量做 or 卷积后的结果。通常我们可以采用快速莫比乌斯变换 FMT 进行加速。

记 FMT 变换矩阵为 F,

$$F(x,y) = \begin{cases} 0 & y \not\subseteq x \\ 1 & y \subseteq x \end{cases}$$

(FMT 其实就是一个子集前缀和。)

用 $\odot$ 表示逐项相乘，记

$$w = (Fv_0) \odot (Fv_2) \odot \ldots \odot (Fv_{n-1})$$

那么 $F^{-1}w$(即对 w 做 FMT 逆变换) 就是答案。如果对每一个向量都做一次 FMT 变换，再乘起来，复杂度比暴力还高。

但由于做卷积的向量都是只有 k 位有值，并且值都是一样的 (只是位置不一样)，是 $p_1, p_2 \ldots p_n$. 考虑到 F 是一个 01 矩阵，所以 Fv_i 的每一项有 2^k 种取值。我们用 $\mathrm{val}(x)$ 表示这些不同的取值，其中 x 是 0 到 $2^k - 1$ 的二进制 mask.

$$\mathrm{val}(x) = \sum_{i \in x} p_i$$

(x 是用二进制表示的一个 p_i 的集合。)

以下，对于某个向量 v, 我们用 $v(i)$ 表示它的第 i 项。
最后的向量 w 可以表示成

$$w(j) = \prod_{x=0}^{2^k-1} \mathrm{val(x)}^{\mathrm{cnt}(x,j)}$$

其中 cnt 是一个 $2^k \times 2^m$ 的矩阵，$\mathrm{cnt}(x, j)$ 的含义就是有多少个向量 v 做 FMT 变换后的第 j 项的值是 x.
我们的目标就是求出 cnt.

$$
\begin{aligned}
\operatorname{cnt}(x,j) &= \sum_{i=0}^{n-1}[(Fv_i)(j) = \operatorname{val}(x)] \\
&= \sum_{i=0}^{n-1}[(F(\sum_{t=0}^{k-1} p_t \operatorname{id}(a_{it})))(j) = \operatorname{val}(x)] \\
&= \sum_{i=0}^{n-1}[(\sum_{t=0}^{k-1} p_t F\operatorname{id}(a_{it}))(j) = \operatorname{val}(x)] \\
&= \sum_{i=0}^{n-1}\prod_{t=0}^{k-1}[(F\operatorname{id}(a_{it}))(j) = [t \in x]] \\
&= \sum_{i=0}^{n-1}(\prod_{t\in x}(F\operatorname{id}(a_{it}))(j))(\prod_{t\notin x}(1-(F\operatorname{id}(a_{it}))(j)))
\end{aligned}
$$

我们用向量表示上面的式子，$\mathrm{cnt}(x)$ 表示 cnt 矩阵的第 x 行，则有，

$$\mathrm{cnt}(x)=\sum_{i=0}^{n-1}\Big(\prod_{t\in x}F\mathrm{id}(a_{it})\Big)\odot\Big(\prod_{t\notin x}(\mathbf{1}-F\mathrm{id}(a_{it}))\Big)$$

这里的 $\mathbf{1}$ 表示一个全 1 向量, 累乘是 $\odot$ 的累乘，逐项相乘。

将上式的乘积展开，有

$$\operatorname{cnt}(x) = \sum_{i=0}^{n-1} \sum_{y=0}^{2^k-1} \operatorname{co}(x, y) \prod_{t \in y} F\mathrm{id}(a_{it})$$

其中 co 是 2^k 行, 2^k 列的系数矩阵.

$$\operatorname{co}(x, y) = \begin{cases} 0 & x \not\subseteq y \\ (-1)^{|x \oplus y|} & x \subseteq y \end{cases}$$

注意到 $F\mathrm{id}(x)$ 的含义是将矩阵 F 的第 x 列提取出来，由 FMT 变换的性质知，$(F\mathrm{id}(x)) \odot (F\mathrm{id}(y)) = F\mathrm{id}(x|y)$
我们可以利用这一性质化简上面的式子。

$$
\begin{aligned}
\mathrm{cnt}(x) &= \sum_{i=0}^{n-1} \sum_{y=0}^{2^k-1} \mathrm{co}(x, y) F\mathrm{id}(\mathop{\mathrm{OR}}_{t \in y} a_{it}) \\
&= \sum_{y=0}^{2^k-1} \sum_{i=0}^{n-1} \mathrm{co}(x, y) F\mathrm{id}(\mathop{\mathrm{OR}}_{t \in y} a_{it}) \\
&= \sum_{y=0}^{2^k-1} \mathrm{co}(x, y) F(\sum_{i=0}^{n-1} \mathrm{id}(\mathop{\mathrm{OR}}_{t \in y} a_{it}))
\end{aligned}
$$

其中 $\mathop{\mathrm{OR}}_{t \in y} a_{it}$ 表示将所有的 $a_{it}(t \in y)$ 做按位或运算的结果。

化到这一步，做法就呼之欲出了。我们只要对每个 y, 算出 $\sum_{i=0}^{n-1} \mathrm{id}(\mathop{\mathrm{OR}}_{t\in y} a_{it})$, 并做 FMT 变换。然后用系数矩阵 $\mathrm{co}(x, y)$ 来乘，就得到 $\mathrm{cnt}(x)$

如果我们用系数矩阵 co 暴力乘，就得到了一个 $O(2^k n + 2^{k+m} m + 2^{m+2k})$ 的做法。

由于 k 较大，这个复杂度还不够。

考虑 FMT 逆变换矩阵：

$$F^{-1}(x, y) = \begin{cases} 0 & y \not\subseteq x \\ (-1)^{|x \oplus y|} & y \subseteq x \end{cases}$$

恰好是我们的系数矩阵 co 的转置。于是最后一步不需要暴力乘 co, 做一次 FMT 逆变换即可。复杂度是 $O(2^k n + 2^{k+m}(k+m))$