浙大城市学院第二十届程序设计新生赛暨第十五届蓝桥杯全国软件和信息技术专业人才大赛选拔赛 - Solution

HZCU ACM Group

HangZhou City University

2023.12.23

A - Hile and Start

- 输入字符串后直接输出原字符串后加上“, start!”即可，注意逗号后面有个空格。

B - Bit Magic

- 看到异或操作，考虑二进制，从高位往低位走，先维护高位，有四种情况
  首先，若 x 和 y 在这一位上相同，就跳过
  若 x 为 1，y 为 0，则第一操作无用，只能用第二种操作将 1 变 0
  若 x 为 0，y 为 1，那么我们既可以用第一种操作，也可以用第二种操作，我们可以发现第一种操作的花费一定小于等于第二种操作，所以我们选择第一种操作，花费是 $2^i - 2^t$(t 是 x 在低位上为 1 的位置) 用完后记得置 0。

C - Sinemory I

- 给出一些数组，求元素之积等于数组标号之积的有序下标对个数。

C - Sinemory I

- 给出一些数组，求元素之积等于数组标号之积的有序下标对个数。
- 我们令 P_x 表示元素 x 的数组标号，则可以写出式子

$$x * y = P_x * P_y$$

C - Sinemory I

- 给出一些数组，求元素之积等于数组标号之积的有序下标对个数。
- 我们令 P_x 表示元素 x 的数组标号，则可以写出式子

$$x * y = P_x * P_y$$

- 稍微转换一下

$$\frac{x}{P_x} = \frac{P_y}{y}$$

C - Sinemory I

- 给出一些数组，求元素之积等于数组标号之积的有序下标对个数。
- 我们令 P_x 表示元素 x 的数组标号，则可以写出式子

$$x * y = P_x * P_y$$

- 稍微转换一下

$$\frac{x}{P_x} = \frac{P_y}{y}$$

- 因此只需要将所有元素的 $\frac{x}{P_x}$ 存储到 map 里然后对每个元素查询可以和其匹配的元素个数即可，即对于元素 i ，查询 $\frac{P_i}{i}$ 的个数即可。
注意不能直接将 double 存储在 map 里，由于精度的问题会导致答案错误。需要以分式的形式存储，可以用 $gcd()$ 得到最简分式

D - Sinemory II

- 给出一些数组，求元素之积等于数组标号之积且乘积不超过 n 的有序下标对个数。

D - Sinemory II

- 给出一些数组，求元素之积等于数组标号之积且乘积不超过 n 的有序下标对个数。
- 本题有两种方法：

D - Sinemory II

- 给出一些数组，求元素之积等于数组标号之积且乘积不超过 n 的有序下标对个数。
- 本题有两种方法：
- 方法 1：可以利用 C 题结论，从标号较大的数组向标号较小的数组遍历所有元素，用一个指针从标号小数组向标号大数组遍历，仅将标号乘积不超过 n 的元素加入 map 中，复杂度为 $O(log_2(\sum m_i) * \sum m_i)$。

- 方法 2：转为枚举所有有序对 (i,j) 对答案的贡献，满足 $i \le j$ 并且 $i*j \le n$,$a_i * a_j = a_{i*j}$。最后把这个个数乘以二再减去所有 (i,i) 形式的贡献即为答案。易证所有这样的有序对 (i,j) 中，仅有小于 $\sqrt{n}$ 的数会出现在 i 的位置上，而所有数在 j 位置上出现次数不超过 $\sqrt{n}$ 次。故直接两层循环枚举 i,j，外层循环枚举 i 后对数组 i 用 $O(1)$ 的桶存各种值的出现次数，然后内层循环遍历数组 j，在桶里统计答案即可。外层循环下的复杂度总和小于 $\sum m_i$，内层复杂度总和小于 $\sum_{i=1}^{\sqrt{n}} n/i$，最后形如 (i,i) 的数对个数为 $\sqrt{n}$ 个，直接暴力枚举所有，且复杂度总和小于 $\sum m_i$，故总复杂度小于 $\ln n * \sum_{i=1}^{n} m_i$，比方法 1 更快。

E - Hile and Her Codes

- 由题意可知只有区间包含是合法的如果出现区间交叉就是非法的
  具体的来说我们定义 $st[i]$ 为第 i 个文件的起始位置,$ed[i]$ 为第 i 个文件的结束位置
  那么合法就是要满足 $st[i] < st[j], ed[j] < ed[i]$ 对于所有的 $j \in (st[i], ed[i])$ 成立
  因此我们只要用栈模拟这个过程即可
  具体的来说如果当前点是起始点那么把它入栈
  如果是终点就和栈顶比较如果不是这个节点就是出现了区间交叉 (非法) 然后出栈
  如果当前节点既不是起始点也不是终点那么这个点应该和栈顶元素相等
  那么合法方案就是入栈的顺序, 时间复杂度 O(n)。

F - Why do Vila play Haruhikage?

- 初始分数不变，因此答案要求最大化特殊音符的收益。根据 k 和 a_i 得到每秒可获得的初始分数 $score_i$，那么将第 i 秒的 1 个普通音符转化为特殊音符后的收益为 $\sum_{j=i+1}^{\min(i+x,n)} score_j$，前缀和预处理每一秒的收益后排序即可。注意 1 个普通音符只能转化 1 次，因此第 i 秒的收益最多选取 a_i 次。

G - masttf's tree

- 首先我们可以 dfs 预处理出每个节点以它为根的子树中需要经过它的才能到达最近白色祖先节点的黑色节点的个数和这个节点到它最近白色祖先节点的距离
  记总的价值和为 sum,$num[i]$ 表示以 i 为根的子树中经过它的黑色节点个数,$f[i]$ 为 i 到最近白色祖先节点的距离,now 为当前节点,$father$ 为当前节点的父亲节点,v 为当前节点的儿子
  那么有 $f[now] = f[father] + 1$ 如果当前节点是白色节点 $f[now] = 0$, $num[i] += num[v]$ v 不是白色节点
  然后我们枚举黑色节点如果把它变白这时候 sum 就会变成 $sum - num[i] * f[i]$ 一直取 min 就好了
  时间复杂度 $O(n)$

H - Physical constant

- 首先尝试 dp，但是观察数据范围完全背包不可解。对每个数算贡献也无法进行，于是考虑直接暴力枚举所有情况计算期望。因为是任取数的乘积，所以首先猜测符合答案的情况数很少。先进行一次暴力，发现在数据量大的情况下无法出结果，再考虑进行优化。因为是乘积，所以可以进行根号分治。将小于根号 k 和大于根号 k 的结果分开，最后计算时再进行组合。这样优化后发现其实仍然会 TLE，但是尚且可以跑出答案。

H - Physical constant

- 首先尝试 dp，但是观察数据范围完全背包不可解。对每个数算贡献也无法进行，于是考虑直接暴力枚举所有情况计算期望。因为是任取数的乘积，所以首先猜测符合答案的情况数很少。先进行一次暴力，发现在数据量大的情况下无法出结果，再考虑进行优化。因为是乘积，所以可以进行根号分治。将小于根号 k 和大于根号 k 的结果分开，最后计算时再进行组合。这样优化后发现其实仍然会 TLE，但是尚且可以跑出答案。
- 所以再次对搜索进行优化，还是因为乘法的存在，所以可以再次分治，考虑 meet-in-middle（折半搜索），将所有的组合情况拆分成两半，在最后计算答案时再组合到一起，这样可以少跑很多无效结果。最后请注意自己的代码是否有除数为 0 的情况。

I - 单马擒王

- 根据马的移动方式模拟即可。可以用 map<pair<int, int>, int> 记录坐标以及该坐标的棋子情况。

I - 单马擒王

- 根据马的移动方式模拟即可。可以用 map<pair<int, int>, int> 记录坐标以及该坐标的棋子情况。
- 注意需要特判没有马或者两匹马的情况，以及不能攻击友军。如果得到了格式错误，确保你给出的马的移动字符串与题面中的移动字符串长度一致。

J - Hile and Paopao

- 模拟即可，枚举每个位置作为插入点，然后利用双指针从插入点左右开始，如果两个指针指向位置的数值相同就向两侧扩展，直至一方触及边界。最后请留意 $n = 1$ 时答案应该为 1。

K、L - Candle I、Candle II

- 对于一块长为 l 宽为 w 的布上放置蜡烛的方案数记作 f(l,w)

K、L - Candle I、Candle II

- 对于一块长为 l 宽为 w 的布上放置蜡烛的方案数记作 f(l,w)
- 我们枚举第一块布的长 len 再计算两块布的方案数乘积，把这些乘积加起来就是答案，具体来说就是求

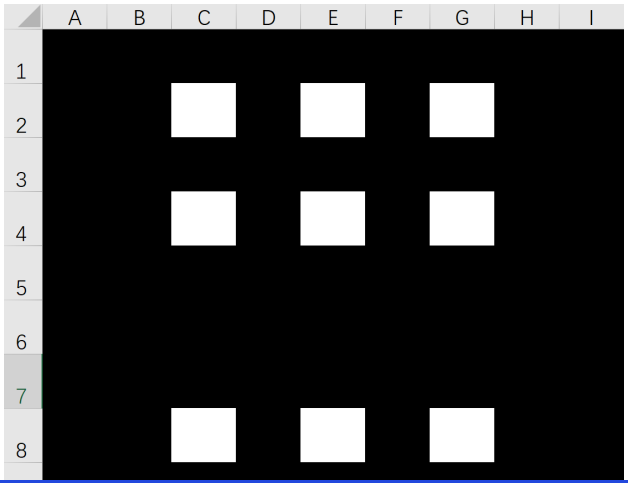
$$\sum_{len=0}^{c} f(len, a) * f(c - len, b)$$

K、L - Candle I、Candle II

- 对于一块长为 l 宽为 w 的布上放置蜡烛的方案数记作 f(l,w)
- 我们枚举第一块布的长 len 再计算两块布的方案数乘积，把这些乘积加起来就是答案，具体来说就是求

$$\sum_{len=0}^{c} f(len, a) * f(c - len, b)$$

- 如何求 f 函数？可以发现某个具体的蜡烛阵和 (行下标集合，列下标集合) 是一一对应的。列如：这个三阶蜡烛阵对应的是 (2,4,8,c,e,g)



- 那么我们就可以枚举阶数，在用组合数学技巧求出 f

$$f(l,w) = \sum_{i=0}^{min(l,w)} C_l^i * C_w^i$$

- 那么我们就可以枚举阶数，在用组合数学技巧求出 f

$$f(l,w) = \sum_{i=0}^{min(l,w)} C_l^i * C_w^i$$

- 对于这一个公式我们根据 l 和 w 的大小关系，用范德蒙德卷积加速：

$$f(l,w) = \begin{cases} C_{l+w}^{l}, & \text{if l<w} \\ C_{l+w}^{w}, & \text{if l>=w} \end{cases}$$

- 那么我们就可以枚举阶数，在用组合数学技巧求出 f

$$f(l,w) = \sum_{i=0}^{min(l,w)} C_l^i * C_w^i$$

- 对于这一个公式我们根据 l 和 w 的大小关系，用范德蒙德卷积加速：

$$f(l,w) = \begin{cases} C_{l+w}^{l}, & \text{if l<w} \\ C_{l+w}^{w}, & \text{if l>=w} \end{cases}$$

- 不难发现这两种情况的结果是一样的所以不妨：

$$f(l,w) = C_{l+w}^{l}$$

- 所以

$$res = \sum_{len=0}^{c} C_{len+a}^{a} * C_{c-len+b}^{b}$$

- 对于 easy version 至此已经结束了对于 hard version 我们需要 O(1) 计算这个公式用隔板法理解这个公式加速为

$$res = C_{a+b+c+1}^{a+b+1}$$

M-Love Comes From Reborn

- 我们可以枚举操作序列进行到某个点 i，那么我们可以知道经过前 i 次操作，当前点位于什么位置，同时计算出他还需要 x 次'R' 操作使得他能走到 n，那么我们的转变次数就是 i+1 到 i+x 中有几个'L'。

M-Love Comes From Reborn

- 我们可以枚举操作序列进行到某个点 i，那么我们可以知道经过前 i 次操作，当前点位于什么位置，同时计算出他还需要 x 次'R' 操作使得他能走到 n，那么我们的转变次数就是 i+1 到 i+x 中有几个'L'。
- 我们可以用前缀和直接求出这个区间有几个'L'，然后对每个点所需的转变次数取最小就是答案。

END

END