2020ICPC・小米网络选拔赛第二场 [1]

October 30, 2020

[1]https://ac.nowcoder.com/acm/contest/7502

A. 2020

如果有 k 个 2020, 我们需要选择 $2k$ 个 20. 假设我们可以知道第 i 个 20 的 2 在 l_i 位置，0 在 r_i. 位置。如果假设 $l_1 < \cdots < l_{2k}$, 我们可以交换 r 保证 $r_1 < \cdots < r_{2k}$。同时，一定是 (l_i, r_i) 和 (l_{i+k}, r_{i+k}) 组成 2020.

因此我们可以二分答案 k, 之后只要让 $r_1, \ldots, r_k$ 尽量小，同时 $l_{k+1}, \ldots, l_{2k}$ 尽量大即可。我们可以先从左到右贪心，得到前 k 个 (l_i, r_i), 再从右到左贪心得到后 k 个 (l_i, r_i)，判断 $r_i < l_{i+k}$ 是否成立即可。

B. Bounding Box

对于每个 B_x，就是要求出有多少 B_y 他们有交点，但是并没有完全包含/被包含。先考虑对于每个 B_x 求出完全在 B_x 内部的 B_y 的 y 之和。对于每个连通块 y，我们随便选取一个位置 (r_y, c_y) 作为代表元。那么可以预处理二维前缀和，就可 $O(1)$ 求出 B_x 里面代表元之和。但是这里我们可能会有多算，并且这些多算的 B_y 肯定出现在 B_x 的边界上。于是就可以枚举 B_x 的边界，把这部分多算的减去。这样的复杂度是 $O(nm)$ 的，由于 B_x 里面恰好是 x 的位置至少是边界长度除以 2，所有边界长度之和会被总元素个数卡住。接下来考虑容斥计算答案，分成两部分：只有一个角在 B_x 内，有两个角在 B_x 内。对于只有一个角在 B_x 内，稍微画画图可以发现可以这么求：B_x 内所有角上之和 - 两个角在 B_x 里的 + 四个角都在 B_x 里面。四个角都在 B_x 里面在刚刚讲了，需要解决有两个角在 B_x 内。可以发现，这个时候 B_y 里某个 y 肯定出现在 B_x 的边界上。和刚才做法一样枚举下即可。整体可以在 $O(nm)$ 解决这个题了。

C. Data Structure Problem

可以发现 dp_x 一定是某个 $a_i + b_{i+1} + b_{i+2} + \cdots + b_x$。令 b 的前缀和为 s，那么 $dp_x = \max_{i=0}^{x}(a_i + s_x - s_i) = s_x + \max_{i=0}^{x}(a_i - s_i)$。用线段树维护 $a_i - s_i$ 即可，操作 1 对应单点修改，操作 2 对应区间修改，操作 3 就是区间取 $\max$。

D. Determinant

比较方便的推导是用 Matrix Determinant Lemma, 可以立刻得到答案就是 $(\sum_{i=1}^{n} a_i \cdot b_i)x^{n-1}$.

E. Query of Square

考虑离线 + 按照 x 坐标从大到小扫描线处理所有询问。可以发现，我们每次扫描线的时候会加入一段 y 边界或者删掉一段 y 边界。可以用线段树维护当前的所有边界，维护这个边界所在 x 坐标即可。对于一个询问 (u_j, v_j)，我们可以二分答案，然后直接线段树查查最值，这样复杂度是 $O(n \log n + m \log^2 n)$。但是注意到，如果单独把 $y \geq v_j$ 的边界拿出来，按照 $\text{border}(y) \leftarrow \max(\text{border}(y), \text{border}(y-1))$ 这样处理，这样就是一个单调的折线段了。然后就是要求出斜率为 1 的经过 (u_j, v_j) 的点和这条折线段的交点。可以简单的二分得出。事实上，这段过程可以在线段树上模拟。你可以先求出 $y \geq v_j$ 这段边界在线段树上对应的 $O(\log n)$ 个节点。这个节点本身的最大值，就可以构成一条虚拟的折线。可以挨个枚举过去，求出在哪个节点发生了相交。之后就可以从这个节点出发，在线段树上二分。这个样复杂度就是 $O((n+m)\log n)$ 了。

F. Modulo Nine

首先我们只要注意 3 的个数即可，0 到 9 这 10 个数字被分成了 3 类：

- 没有 3 的：$1, 2, 4, 5, 7, 8$
- 有 1 个 3 的：$3, 6$
- 有 2 个 3 的：$0, 9$

我们设 $f_i(j, k)$ $(i \geq j \geq k)$ 表示考虑到第 i 个数字，倒数第一个 3 在 j 位置，倒数第二个 3 在 k 位置的方案数，我们的限制条件对 k 有下界的要求。我们考虑当 $i-1$ 变到 i 会发生什么，有 3 种情况。

- $i > j \geq k$ 的情况，那么 $f_i(j,k) = 6f_{i-1}(j,k)$.
- $i = j > k$ 的情况，那么 $f_i(i,k) = 2\sum_j f_{i-1}(k,j)$
- $i = j = k$ 的情况，那么 $f_i(i,i) = 2\sum_{j,k} f_{i-1}(j,k)$.

我们如果把答案除以 6^n, 可以把 6 当作 1, 2 当作 $\frac{1}{3}$. 所以 f_i 和 f_{i-1} 的不同只有 $f_i(i,*)$ 这一行，我们可以通过维护这个矩阵每行的和，在 $O(n)$ 内算出这一行。至于 k 的下界要求，因为这个下界总是不断变大，我们可以暴力一列一列清 0, 总复杂度是 $O(n^2)$.

G. Shift and Reverse

把这些数放在一个环上。那么每次操作其实等价于翻转这个环，然后 shift 一下。总共有 $2n$ 种可能，用 hash 之类的判判是否一样即可。

H. Knapsack

仿照普通 01 背包的解法，我们设 $f(j)$ 表示大小为 j 的背包的最大价值。初始时 $f(j) = 0$. 考虑一个重量 w 的所有物品，假设它们的价值是 $v_1 \geq \cdots \geq v_k$. 我们设 $s(i) = v_1 + \cdots + v_i$. 同时，对于 $0 \leq r < w$, 我们设 $g(i) = f(r + i \cdot w)$. 显然，更新后的 $g'(i) = \max_{j=0}^{i} g(j) + s(i-j)$. 设 $g'(i)$ 时的最优决策是 b_i, 我们可以证明 $b_0 \leq b_1 \leq \cdots$ 成立，换言之 $g'(i)$ 具有决策单调性，可以使用 SMAWK 算法在 $O(m/w)$ 内解决。总的复杂度是 $O(m \max w_i)$.

I. Subsequence Pair

假设最后删完之后的字符串是 x 和 y，那么要么 x 是 y 的前缀，要么存在一个 i 使得 $x_i < y_i$。预处理出 $\mathrm{lcs}(i,j)$ 表示 $s_1 s_2 \ldots s_i$ 和 $t_1 t_2 \ldots t_j$ 的最长公共子序列长度。对于 x 是 y 的前缀，答案肯定是 $\max(\mathrm{lcs}(i,j) * 2 + (m - j))$。对于 $x < y$ 但是 x 不是 y 的前缀，那么肯定有个 i 和 j，使得 $s_i < t_j$。枚举这对 i 和 j，用 $\mathrm{lcm}(i-1, j-1) \cdot 2 + n - i + m - j + 2$ 来更新答案。时间复杂度 $O(nm)$。

J. Hamming Distance

可以发现 $S_i^m = \mathrm{ctz}(i) + 1$，其中 $\mathrm{ctz}(i)$ 表示 i 二进制表示末尾 0 的个数。先考虑求和，只需要算每一个位置对答案的贡献即可。也就是枚举每个 a_i $(0 \leq i < n)$，统计区间 $[i+1, 2^m - n + i]$ 里有多少位置对 2^{a_i} 取模为 0，但是对 2^{a_i+1} 取模不为 0。对于求最小值，我们可以发现对于 S^m 任意一个子串，里面的最大值近出现一次，并且以这个最大值为中心是个回文串。于是我们可以每个 a_i $(0 \leq i < n)$ 作为这个最大值，找到一个最小的 o，使得 $2^o - 1 \geq \max(i, n-1-i)$。那么对于任意在 $x \in [o, m]$ 的数都可以填在 a_i 上面。我们还可以发现，别的位置 j 应该是的字符为 $\mathrm{ctz}(|i-j|) + 1$，枚举 j 应该是的字符为 t，那么有 $j \equiv i \pmod{2^t}$ 和 $j \not\equiv i \pmod{2^{t+1}}$。只需要预处理下 $sum(t,x,y)$ 表示 $i \bmod 2^t = x$ 中有多少 $a_i = y$ 即可。时间复杂度 $O(n \log n)$。

K. Suffix Array for Fibonacci

有个广为人知的事实：根据后缀自动机/后缀树可以求出后缀数组。因此我们可以尝试建出 fib_n 的后缀自动机。幸运的是，fib_n 的后缀自动机很有规律：+ 每个 fibonacci 串交替以 `ab` 和 `ba` 结尾，$fib_n, fib_{n-2}, \ldots, fib_{n \bmod 2}$ 对应的位置都是接受态。+ 每个节点除去往相邻点有个出边外，fib_n 的后缀自动机恰好有 $\min(n-2, 0)$ 条额外的边。+ 每条额外的边要么是从 `ab` 结尾的 fibonacci 串的 `ab` 前出发，通过标号为 `a` 的边，跳到下一个 `ba` 结尾的 fibonacci 串的 `ba` 中间处；要么是从 `ba` 结尾的 fibonacci 串的 `ba` 前出发，通过标号为 `b` 的边，跳到下一个 `ab` 结尾的 fibonacci 串的 `ab` 中间处。+ 在这个后缀自动机上走的时候，肯定不会连续两次经过额外边。根据这个规则建出后缀自动机，并把出度为 1 的边都压缩，这样就可以 $O(n)$ 表示 fib_n 的后缀自动机。然后我们就可以求出每个节点 x 出发能够走到的接受态的数目 $ways(x)$。暴力做法就是从其实点开始，然后根据 $ways(x)$ 的大小决定往 `a` 边走还是 `b` 边走。但是由于 n 比较大，

我们显然只需要使用到这个自动机上最后若干个点。因此，仅建出最后部分对应的自动即可，然后在这部分上走。复杂度 $O(q\log A)$，其中 $A=\max(p_i)$。