NOIP2024 模拟赛

GDFZ

2024 年 11 月 25 日

目录

A 树（tree）

假设 u, v 在树上的距离为 x，则当以 u 为根的时候，每在 v 子树内加一个点，$f(u) - f(v)$ 就增加 x。先用这样加点增加到不能再增加，然后再根据 x 与 n 的奇偶性简单讨论即可。

x 有范围 $[\sqrt{n} - O(1), \sqrt{n} + O(1)]$，实际上可以说明 $|x - \sqrt{n}| \leq 3$，枚举即可。注意使用 `sqrtl`，也可以使用二分。

B 序列（seq）

考虑 DP，设 $f_{i,j}$ 表示 $p_i = j$ 是否可行。转移枚举一个 k，若 $t_{i+1} = s_k$ 且 $[j+1, k-1]$ 只有一种字符，就可以从 $f_{i,j}$ 转移到 $f_{i+1,k}$。

设 x 为满足 $f_{i,x} = 1$ 的最小整数，y 为满足 $f_{i,y} = 1$ 的最大整数。容易发现 $f_{i,x\sim y}$ 中，除了 $\neq t_i$ 的位置，其他位置的 DP 值都是 1。也就是说若忽略 $\neq t_i$ 的位置，DP 值为 1 的位置可以构成一个连续段。

那么我们只用对于每个 i 维护 (x, y)，转移是 $O(1)$ 的。构造方案直接倒推即可。时间复杂度 $O(n + m)$。

C 转圈（circ）

不难发现其实我们只在乎最后环能到多大。对答案最主要的限制来自边，如果我们知道点 u 缩点后在环上的点 k（为了方便，下面称此时 u 的颜色是 k），则如果有边 (u,v)，则 v 缩点后在 $(k+1) \bmod t$ 上，其中 t 是环的大小。在树上我们随便给一个点颜色染成 0，则可以将其他点的颜色求出来，所以可以缩成一条链，然后将这条链首尾相接就得到了最大的环，答案是链长 -1，结合暴搜可以拿到 30pts。

发现对于任意图，不能硬跑树的做法的原因是一个点容易被重新染色，那不妨考虑检查一个答案 k 是否可行，对于每个连通块，如果没出现矛盾也就成一条链，把这些链全部首尾相连，判断一下最长长度是否达到了 k 即可。注意特判某个连通块缩点后的链本身已经有一条从尾连到首的边（而且此时这条链长度一定是 k）的情况，复杂度 $O(Tnm)$。

发现随便跑 check 太浪费了，我们考虑一个连通块取出一个生成树，这样每个点就有个颜色了，再考虑所有非树边的限制，如果有一条边从颜色 a 连向颜色 b，则最终环长必须是 $|a+1-b|$ 的因数，而且至少已经有 $|a+1-b|$ 种颜色了，所以全部非树边带来的限制取一个 gcd 就一定是最终环长。

但是还有一种特殊情况，就是所有非树边的限制都是 0，也就是没有限制，此时答案就和树的情况差不多，把每个链首尾相连就是最终答案了。

若干个数取 gcd 是 $O(n+\log V)$ 的，所以最终复杂度 $O(T(n+m))$。

D 游戏（game）

1. 算法一

首先，最优策略一定是，先进行若干次区间 -1 操作，在进行若干次区间 $+1$ 操作。若不然，若在 $[l_1, r_1]$ 先进行了 $+1$，又在 $[l_2, r_2]$ 进行了 -1：

- 若 $[l_1, r_1] \cap [l_2, r_2] = \emptyset$ ，则这两个操作先后顺序没有影响，可以直接交换。
- 否则，设 $R = [l_1, r_1] \cap [l_2, r_2]$ ，则我们将操作改为在 $[l_2, r_2] \setminus R$ 上先减 1 ，再从 $[l_1, r_1] \setminus R$ 上加 1。

我们考虑从左到右扫描整个序列，维护 x 表示在此时还有多少个区间 -1 操作跨过了 i，y 表示此时还有多少个区间 $+1$ 操作跨过了 i 。

1. 若 $a_i = 0$ 但 $b_i > 0$ ，显然无解。
2. 若 $b_i > 0$，则必须有 $x < a_i$，因此首先更新 $x \leftarrow \min(x, a_i - 1)$。随后，我们可以将 a_i 更新为 $a'_i \leftarrow a_i - x + y$。

考虑如果 $a'_i < b_i$ ，则我们需要增大 a'_i 的值。我们可以做的操作有:

1. 在 i 处新增一个区间 $+1$ 的操作。
2. 在 i 前删除一个区间 -1 的操作。

考虑如果我们在 i 处删除了一个区间 -1 ，但又在 i 后的一个位置 j 加入了一个区间 -1 和区间 $+1$，则我们需要消耗 3 的代价。而如果我们将删除区间 -1 替换为在 $[i, j]$ 增加一个区间 $+1$ ，并在 j 开头保持需要增加的区间 -1 ，我们只需要消耗 2 代价。因此，我们进行一次操作 2，相当于获得了一次以 1 的代价同时获得一个 -1 与 $+1$ 的机会。

我们再维护一个变量 z 来表示我们拥有的此类机会的数量。

- 当 $a_i > 0, b_i > 0$ 时，同时增加减 1 与加 1 没有意义，因此在这种情况下没有额外的贡献。
- 当 $a_i > 0, b_i = 0$ 的位置，我们可能先将 a_i 减到 0 ，随后再将 $+1$ 操作扩展到后方。因此，如果 $a'_i > 0$ ，我们需要将这个位置 -1，可能的操作仍然有：
  1. 增加一个区间 -1 的操作。
  2. 使用机会增加一个区间 -1 的操作，随后再增加一个区间 $+1$ 的操作。

容易发现，上述两种操作均需 1 的代价，因此操作 2 相当于我们可以获得一次免费 $+1$ 的机会。记 v 表示到当前位置所拥有的免费 $+1$ 操作的数量。则：

- 若 $a'_i < b_i$ ，则我们一定会先查看是否拥有免费的 1 ，如果有立刻使用，否则我们仍然按照上述优先级来操作。

```
i64 t = b[i] - a[i];
if (x < t) {
  add1(t - x);
  t = x;
}
x -= t;
i64 s = min(t, v);
y += s;
x += s;
v -= s;
z += t - s;
```

- 若 $a'_i > b_i$，则我们同样先通过增加区间减 1 操作，并看作后面我们可以获得区间减 1 和区间加 1 操作作为收益。

```
i64 t = a[i] - b[i];
if (y < t) {
  ans += t - y;
  x += t - y;
  a[i] -= t - y;
  t = y;
}
i64 d = a[i] - y - 1;
if (d < t) {
  y -= t - d;
  t = d;
}
y -= t;
z += t;
```

- 最终需要更新对应的边界信息。

```
z += v;
v = 0;
z = min(z, b[i] - 1 - y);
```

实现上述贪心过程即可，需要注意操作时的边界情况。

时间复杂度 $O(n)$。

2. 算法二

本题存在线性规划对偶的做法。

记 $\mathcal{I}$ 为所有区间的集合；对于 $S \in \mathcal{I}$，记 X_S 为这个区间上 -1 操作的数量、Y_S 为这个区间上 $+1$ 操作的数量。

那么题意即最小化 $\sum\limits_{S\in\mathcal{I}} X_S + Y_S$，对于每个点 i 限制操作后 $a_i = b_i$，分类讨论：

- $b_i = 0$，相当于要求减法操作减到 0，加法操作没有任何影响，即对于任意 i 有：

$$\sum_{S\ni i} X_S \geq a_i \tag{A_i}$$

- $b_i \neq 0$，要求减法操作不能减到 0，且加法操作后 $a_i = b_i$，即对任意 i 有：

$$\sum_{S\ni i} -X_S \geq -a_i + 1 \tag{B_i}$$

$$\sum_{S\ni i} -X_S + Y_S \geq b_i - a_i \tag{C_i}$$

$$\sum_{S\ni i} X_S - Y_S \geq a_i - b_i \tag{D_i}$$

那么其对偶问题即：

- 最大化 $\sum\limits_i A_i a_i + B_i(-a_i + 1) + C_i(b_i - a_i) + D_i(a_i - b_i)$。
- 对于任意区间 S，限制：

$$\sum_{i\in S} A_i[b_i = 0] + (-B_i - C_i + D_i)[b_i \neq 0] \leq 1$$

$$\sum_{i\in S} (C_i - D_i)[b_i \neq 0] \leq 1$$

注意到我们只关注 $C_i - D_i$ 的值，于是令 $C'_i = C_i - D_i$，将限制改写为：

$$\sum_{i\in S} A_i[b_i = 0] + (-B_i - C'_i)[b_i \neq 0] \leq 1$$

$$\sum_{i\in S} C'_i[b_i \neq 0] \leq 1$$

考虑分析 C'_i 的下限，我们发现如果 C'_i 减小，那么必然导致 B_i 增大，注意到若 $b_i \geq a_i$，次这么操作必然更劣；若 $b_i < a_i$，则 $b_i - a_i \leq a_i - 1$，那么也必然不优；所以这种调整是不会出现的。

于是有 $-1 \leq C'_i \leq 1, 0 \leq A_i \leq 1, \max(0, -C_i) \leq B_i \leq -C_i + 1$。

处理限制只需要考虑后缀最大值，这个必然为 0 或 1，直接状压，枚举 A, B, C，DP 即可。

时间复杂度 $\mathcal{O}(n)$。