2019牛客暑期多校训练营 第三场

Reconquista

牛客竞赛
AC.NOWCODER.COM

A - Graph Games

- **显然，我们要用hash等技巧来显式地“表现”S(x)。**
  - 用到一个技巧：给每一个点一个long long范围内的随机数key，把一个集合S(x)的值定义为其里面的点的key的异或值，那么判集合相等就可以用判数值相等来替代。
- **然后考虑如何维护这些异或值**
  - 有一个比较显然的 $Q\sqrt{M}\log\boldsymbol{M}$的做法。
  - 我们按点（在原图）的度数分块。对于所有“小点”，每次我们暴力枚举与它相邻的边，看看是否存在，存在的话，用另一侧的点更新它的xor值。至于判断如何存在，其实是一个区间异或、单点查询的问题，我们可以用树状数组做到 logM 的快速查询。
  - 对于所有“大点”，总数不会超过 √M。我们在每次修改的时候，暴力维护每一个大点的答案即可。这里提供一种可能的方法：对每一个“大点”，维护一个它的边的vector（按标号排序）。那么每次一个操作对其的贡献是vector里连续一段。我们只要把这连续的一段的xor值给异或掉即可。这样可能需要一个 logM 的定位。

A - Graph Games

- **数据没有特意卡带log的复杂度。**
- **其实优化一下，就能得到一个 √M 的做法了。**
  - 对于“小点”，瓶颈是维护“区间异或，单点查询”。注意到修改只有Q个，但询问有Q√M 个。所以我们可以匀一匀复杂度，改成对边的序列分块。每次修改时，块间打标记，块内暴力；询问时可以直接O(1)得出结果。
  - 对于“大点”，我们还是在每次修改时暴力维护。考虑一开始对边序列分块，那么一个块最多只会对2*√M个点产生贡献。我们记录一下每一个块对它能影响的每一个大点的xor值的贡献，以及如果翻转了这个块后的贡献。每次修改时，在块间打翻转标记，两侧块暴力重构对大点的贡献。每次询问一个大点时，我们枚举所有的块，把所有的块对它的贡献都异或起来即可。
- **综上，总复杂度是O(Q√M)，内存O(M)，支持在线。**

B - Crazy Binary String

- **子序列显然能取满，就是** min(1**的个数，**0**的个数**)
- **子串的话，需要稍微统计一下**
  - 如果字符是 ‘1’ 值设为1，否则设为-1，求一遍前缀和。
  - 一个子串 (l,r) 满足要求，即为 S[r]-S[l-1]=0
  - 直接扫过去，一边记录 S[l-1] 的情况，一边每次询问是否有 =S[r] 的存在。

- **时间复杂度：** $O(N)$

牛客竞赛
AC.NOWCODER.COM

C - Guessing ETT

- **这是一道很有趣的构造题，可能有很多做法。**
- **首先考虑递归解决问题。**
- **如果某个数字存在超过一次，先对于相邻相同数字构成的那段递归下去解决。**
  - 里面依然可以出现这个数字。
  - 求出一个解后递归上来时，我们可以将这两个数看做同一个数，重新又变成了和原来等价的问题。
- **现在我们手头的子问题是：已知一个序列，两头是k，中间没有任何相同的数字出现，让你将-1填完，使得它是一个合法的欧拉序。**

C - Guessing ETT

- **一个想当然的做法是，实时维护一个栈，表示目前所到的点到根路径的所有点。**
  - 一开始先把k放进去。
- **如果遇到一个非-1的数，很简单：**
  - 如果在栈的倒数第二个位置，就弹出栈的最后一个数字。
  - 否则，将这个数加入栈。
- **遇到一个-1，看起来是优先弹栈，如果不能弹了，就用一个没用过的数字？**

C - Guessing ETT

- 看一个简单的例子：1 ？3 ？？4 ？2 1
- 怎么知道第二格要填1呢？
- 如果不填对不对？好像可以！1 5 3 5 1 4 1 2 1！
- 但是如果是这个例子：1 ？3 4 ？2 ？？1
- 就不能直接写5了，只能填4或者2。
- 有些时候需要强制填一个后面出现过的数字。
- 假设当前位置是i，后面那个数的位置是j，那么有判定条件：
- $(Sum[j-1]-Sum[i])*2=j-i$

C - Guessing ETT

- **注意还有一个条件是：I 和 j 的奇偶性要相同。**
- **上述式子可以简单地线性维护。**
- **这样 -1 的流程也就出来了：**
  - 如果存在一个>i的j满足上述式子，挑一个最小的j，填上对应数字。
  - 否则，看能否退栈，能就退，不能就进。
- **这题细节还是比较繁琐的，建议都去实现以下。**

- **如果实现精细，复杂度是 O(N) 的。**

D - Big Integer

$11\cdots111 = \frac{10^n-1}{9} \equiv 0(mod\ p)$

等价于$10^n \equiv 1(mod\ 9p)$

当$p \neq 2,5$的时候，有$gcd(10,9p)=1$, 因此$10^{\phi(9p)} \equiv 1(mod\ 9p)$

我们需要找到$10^i\ mod\ 9p$ 的循环节d. 显然 $d \mid \phi(9p)$, 直接暴力枚举$\phi(9p)$的约数验证就好了。

找到循环节d之后，我们要知道有多少个$pair(i,j)$, $d \mid i^j$.

牛客竞赛
AC.NOWCODER.COM

D - Big Integer

对d做质因子分解, $d=p_1^{k_1}p_2^{k_2}\cdots p_l^{k_l}$，考虑$j$固定的时候，$i$需要满足什么条件。

i必须是$g=p_1^{\lceil\frac{k_1}{j}\rceil}p_2^{\lceil\frac{k_2}{j}\rceil}\cdots p_l^{\lceil\frac{k_l}{j}\rceil}$的倍数。因此一共有$\frac{n}{g}$个合法的$i$.

由于$k_i\le 30$, 所以j增大到30以上和$j=30$的结果是一样的，枚举j从1到30，分别计算g即可。

当$p=2,5$的时候，显然答案是0.

E - Trees in the Pocket II

- **DFS框架:**

  - 以第一棵树的边按照从大到小的顺序构造kruskal重构树，记为树A。
  - 按照重构树的dfs序重标号。
  - 在树A上dfs，考察A上非叶子节点及对应权值a_i的意义：对于左（右）子树的叶节点，若该节点是好点，那么在第二颗树上，它的标号对应点到每个右（左）子树的叶节点标号对应点要么存在一条只通过$b_i \le a_i$的边的路径，要么存在一条只通过$c_i \le a_i$的边的路径。
  - 即它所在的$b_i \le a_i$的边构成的树块和所在$c_i \le a_i$的边构成的树块中，包含了所有右（左）子树的叶节点。
  - 接下来考察左子树是否是好点，右子树的方法对照执行即可。

牛客竞赛
AC.NOWCODER.COM

E - Trees in the Pocket II

- 分别以第二颗树的边按照$b_i, c_i, max(b_i, c_i)$从小到大构造kruskal重构树，并以dfs序为时间轴构造可持久化线段树，依次记为树B，C，D。
- 任意取右子树的一个节点，在B上倍增找到$b_i \le a_i$的树块对应子树，并利用可持久化线段树找到一个不在该树块内的右子树叶节点（可能不存在），在C上同理，分别记子树为 ST_b 和 ST_c ，不在块内的节点为Out_b和Out_c 。
- **分四种情况讨论：**
  - Out_b和Out_c均不存在
  - 仅Out_b不存在
  - 仅Out_c不存在
  - Out_b和Out_c均存在

E - Trees in the Pocket II

- **Out_b和Out_c均不存在**

左子树的节点需要在ST_b或ST_c 内，标记记录两棵子树在B和C上对应的DFS序区间，称or标记，在dfs时下传。

该标记保留第一次遇到的or标记ori和另一个or标记oth。

标记叠加时，由于DFS时a_i不断增大，这些or标记在B和C上要么不交要么包含，讨论：

- 若新ST_b 包含ori_{b} ，将oth_{c}与新ST_c求交。
- 若新ST_c 包含ori_c ，将oth_b与新ST_b求交。
- 否则无解。

最终检查标记时，检查在$(ori_{\mathrm{b}}\ \ and\ oth_{\mathrm{c}})\ or\ (ori_{\mathrm{b}}\ \ and\ oth_{\mathrm{c}})$内即可。

E - Trees in the Pocket II

- **仅Out_b不存在**

左子树的节点需要在ST_b内，标记记录子树在B上对应的DFS序区间，称B标记，在dfs时下传。

标记叠加时直接求交。

检查时直接检查是否在标记对应区间内即可。

- **仅Out_c不存在**

同仅Out_b不存在，记录C标记。

E - Trees in the Pocket II

- Out_b**和**Out_c**均存在**

仍然可能存在最多一对（$b_i \le a_i$的树块， $c_i \le a_i$的树块），它们的并包含所有右子树叶节点，此时左子树的节点需要在它们的并树块内。唯一性可利用树上每条边均为割简单证明。

树块对只可能是（ ST_b 所在B树块， ST_c 所在C树块），（ ST_b 所在B树块，Out_b所在C树块），（Out_c所在B树块， ST_c 所在C树块）之一。

找到树块交：两树块内分别各取一点，在第二棵树上lca找到路径，在（两段）树链上跳表找到交内的一点即可。

具体地，分别跳到B块和C块距离自身起点的最远位置，选择其中没有跳过lca的那个，都跳过lca时就选择lca。然后在树D上找到对应树块即可。

检查是否是合法树块对：利用B，C，D的可持久化线段树计算树块内右子树叶节点的数量，$B + C - D = all$。正确性和唯一性显然。

标记记录子树在D上对应的DFS序区间，称D标记，标记行为同前一情况。

牛客竞赛
AC.NOWCODER.COM

E - Trees in the Pocket II

- 总复杂度$O(n\log n)$。
- 小剪枝：
  - 某标记无解时退出；
  - 在另一侧子树的检查中，若检查到本侧的叶节点Out_b或Out_c不存在，说明本侧内部已经联通，本侧dfs下去时无需继续检查。

F - Planting Trees

- **枚举子矩形的上下边界，同时维护对于当前枚举的上下边界的每列最大最小值**
- **确定了上下边界后，枚举矩形的右边界，如何知道可行的最小左边界？**
- **可以使用数据结构或二分求，则复杂度为$O(N^3\log N)$，可能会超时**
- **更快的做法：枚举右边界同时维护两个单调队列，分别维护最大值和最小值**
  - 最小值的单调队列应为下标递增、大小递增，最大值为下标递增、大小递减
  - 以最小值为例，每次入队时弹出队尾不比当前入队元素小的元素
- **使用这两个单调队列就可维护一个当前可行最小左边界的指针**
  - 显然随着右边界右移，该指针是单调不降的
- **时间复杂度：** $O(N^3)$

G - Removing Stones

- 可以证明，原题目等价于：求有多少个长度不小于2的连续子数列，使得其中最大元素不大于所有元素和的1/2
- 考虑分治，对于区间(l, r)每次找出最大元素（多个任意取一个），设其下标为k
- 则所有包含最大元素的子数列的最大值都为该元素
- 枚举(l, k)和(k, r)中较短的区间的每个位置作为答案一个端点，在另一个区间中二分另一端点位置的最大/最小取值
    - 注意由于k的任意性，不能枚举较长的区间，否则复杂度会退化至$O(N^2)$
- 然后递归分治$(l, k-1)$和$(k+1, r)$即可
- 时间复杂度：$O(N\log^2 N)$，但实际达不到这个上界
- 比赛时候发现有线性做法orz……我也来试着讲一下……

H - Magic Line

- **这个题解法很多，可以做得很麻烦。**
- **考虑一个简单做法。**
- **假设没有x重复，直接按x排个序，画在最中间即可。**
- **如果有重复怎么办？我们对(x,y)进行双关键字排序，并定位最中间（偏左）的那个。**
  - 此时只要画一条略在它上面，微微倾斜的直线就行。

- **时间复杂度：** $O(N\log N)$

I - Median

- **一个结论是，如果有解，那么一定可以让每个位置最终的值，是它所能影响到的 3 个中位数之一。**
  - 证明：对于一个合法解中的每一个元素，如果它与 3 个相关中位数都不同，那么可以证明，它一定都比他们大或者都比他们小。调成他们的最大值/最小值后就会相等。
  - 因此每一个合法解都可以调整成一个满足结论的解。
- **因此我们可以用dp来判断解的合法性。**

I - Median

- 令 f[i][j][k]（其中 j 和 k 都是 0~2）表示能否让 a[i] = v[i][j], a[i-1]=v[i-1][k] 并让前 i-2 个中位数满足条件，其中 v[i][j] 表示与 a[i] 相关的 3 个中位数中第 j+1 大的。f[i][j][k] 可以通过 f[i-1][k][l](l=[0,2]）转移得到。
- 只要最终存在合法的f[N][j][k]，我们就可以从转移顺序中还原出一个合法的解。

- 复杂度为 $O(N)$

J - LRU management

- **从题面的LRU操作描述以及询问方式，很容易看出来用双向链表来维护。**
- **我们首先要定位一个 block，比较暴力的做法是 map，如果想速度快的话可以写哈希或 trie。**
- **然后这个结构会对应一个链表节点。**
  - 询问或增加block时，将新的block插入末尾。
  - 删除时直接扔掉head并重定位head。
- **一个特别优美的实现是，用C++的map和list，map的值指向 list 的迭代器。**
- **模拟起来比较繁琐，复杂度 O(N) 或 O(N log N)**

牛客竞赛
AC.NOWCODER.COM