NOI模拟赛solution

samjia2000

March 10, 2018

前言

这是一套非常简单的模拟赛。
大家AK愉快。

小A的树

首先对树进行点分治，对于每一个分治重心，bfs求出分治区间内所有点到它的距离，将这些距离以及对应的节点按照从小到大存起来放到一个序列q中，对于每个点，记录下它是属于这个分治重心的哪颗子树，对于q中每个位置，我们要存下这个位置之前最近的一个跟当前点来自不同的子树的位置。

小A的树

接着，我们二分第K大的距离，设这个值为Lim.

小A的树

接着，我们二分第K大的距离，设这个值为Lim.
记录一个cnt，初始为0，枚举每个节点x，在点分树上沿途查找，对于一个分治重心now，我们暴力在对应的q中用一个指针扫一遍，找到与x距离大于等于Lim的点，并在途中给cnt 累加，如果cnt中途大于等于K了，那么就直接退出函数返回K。

小A的树

接着，我们二分第K大的距离，设这个值为Lim.
记录一个cnt，初始为0，枚举每个节点x，在点分树上沿途查找，对于一个分治重心now，我们暴力在对应的q中用一个指针扫一遍，找到与x距离大于等于Lim的点，并在途中给cnt 累加，如果cnt中途大于等于K了，那么就直接退出函数返回K。
最后返回cnt。

小A的树

这样，做一次check的复杂度就是$O(n\ log\ n + K)$。
所以总的时间复杂度就是$O((n\ log\ n + K) log\ L)$。

小B的序列

用线段树维护每个区间的最大值。

小B的序列

用线段树维护每个区间的最大值。
首先，我们肯定会有一个很naive的想法，就是，如果一个区间做完这次操作后所有的数的结果都变成相等的了，那么就直接打上一个区间赋值操作即可。

小B的序列

用线段树维护每个区间的最大值。
首先，我们肯定会有一个很naive的想法，就是，如果一个区间做完这次操作后所有的数的结果都变成相等的了，那么就直接打上一个区间赋值操作即可。
具体的，就是记录一个区间所有值的or的和，以及and的和，然后就可以得到一个区间内既有1也有0的位的状态。

小B的序列

用线段树维护每个区间的最大值。
首先，我们肯定会有一个很naive的想法，就是，如果一个区间做完这次操作后所有的数的结果都变成相等的了，那么就直接打上一个区间赋值操作即可。
具体的，就是记录一个区间所有值的or的和，以及and的和，然后就可以得到一个区间内既有1也有0的位的状态。
对于区间修改的时候，判断是否与这个状态有交即可。

小B的序列

用线段树维护每个区间的最大值。
首先，我们肯定会有一个很naive的想法，就是，如果一个区间做完这次操作后所有的数的结果都变成相等的了，那么就直接打上一个区间赋值操作即可。
具体的，就是记录一个区间所有值的or的和，以及and的和，然后就可以得到一个区间内既有1也有0的位的状态。
对于区间修改的时候，判断是否与这个状态有交即可。
但是，这样做很naive，完全是可以被卡成$O(n^2 log\ n)$的。

小B的序列

考虑一个看起来更加靠谱的做法。

小B的序列

考虑一个看起来更加靠谱的做法。
对每个区间，维护区间内的最大值，区间的数or的和以及and的和。

小B的序列

考虑一个看起来更加靠谱的做法。
对每个区间，维护区间内的最大值，区间的数or的和以及and的和。
对于每个区间，维护两个懒标记Tag_or和Tag_and，并且我们强制先做and再做or。

小B的序列

考虑一个看起来更加靠谱的做法。
对每个区间，维护区间内的最大值，区间的数or的和以及and的和。
对于每个区间，维护两个懒标记Tag_or和Tag_and，并且我们强制先做and再做or。
((A and B)or C)and D = (A and (B and D))or (C and D)
((A and B)or C)or D = (A and B)or (C or D)

小B的序列

在对区间取or的时候，如果or上的值val与这个区间内所有数既有1也有0的状态没有交集，那么，这个区间内的数和val取or之后的结果的相对顺序是不会发生改变的，所以可以直接将最大值对val取or，否则就递归下去。
取and的时候类似。

小B的序列

时间复杂度？

小B的序列

时间复杂度？
对每一个二进制分开考虑，即假设我们的做法是这样的：只关心这一位，无论其他位是否有交集，都不会影响当前位置是否继续递归。
这样的操作次数显然是要大于我们前面所说做法的操作数的。

小B的序列

时间复杂度？
对每一个二进制分开考虑，即假设我们的做法是这样的：只关心这一位，无论其他位是否有交集，都不会影响当前位置是否继续递归。
这样的操作次数显然是要大于我们前面所说做法的操作数的。
然后，问题其实就变成了对01序列的区间max和区间min，这里需要用到势能分析，详见2016年吉如一的集训队论文，这里不展开讨论。

小B的序列

时间复杂度？
对每一个二进制分开考虑，即假设我们的做法是这样的：只关心这一位，无论其他位是否有交集，都不会影响当前位置是否继续递归。
这样的操作次数显然是要大于我们前面所说做法的操作数的。
然后，问题其实就变成了对01序列的区间max和区间min，这里需要用到势能分析，详见2016年吉如一的集训队论文，这里不展开讨论。
最后总的时间复杂度就是$O(nk\ log\ n)$。

小C的利是

熟悉多项式以及高斯消元求行列式的同学们，应该会写每个位置存一个最高次是K的多项式，然后求行列式的算法。

小C的利是

熟悉多项式以及高斯消元求行列式的同学们，应该会写每个位置存一个最高次是K的多项式，然后求行列式的算法。
但是这样做，要注意要对每个位置随机一个权值，不然你有很大概率会被出题人卡掉。

小C的利是

这里介绍另外一种做法。

小C的利是

这里介绍另外一种做法。
先找到一个质数P，满足$P \equiv 1(mod\ K)$。

小C的利是

这里介绍另外一种做法。
先找到一个质数P，满足$P \equiv 1(mod\ K)$。
设g是P的K次单位根。

小C的利是

这里介绍另外一种做法。
先找到一个质数P，满足$P \equiv 1(mod\ K)$。
设g是P的K次单位根。
对每个位置(i,j)随机一个值$rd_{i,j}$，将每个位置的权值设置为$rd_{i,j} \cdot x^{A_{i,j}}$。

小C的利是

这里介绍另外一种做法。
先找到一个质数P，满足$P \equiv 1(mod\ K)$。
设g是P的K次单位根。
对每个位置(i,j)随机一个值$rd_{i,j}$，将每个位置的权值设置为$rd_{i,j} \cdot x^{A_{i,j}}$。
将$x = g^i, i = 0..k$依次代入，然后计算每次的模P意义下的行列式的和sum。

小C的利是

这里介绍另外一种做法。
先找到一个质数P，满足$P \equiv 1(mod\ K)$。
设g是P的K次单位根。
对每个位置(i,j)随机一个值$rd_{i,j}$，将每个位置的权值设置为$rd_{i,j} \cdot x^{A_{i,j}}$。
将$x = g^i, i = 0..k$依次代入，然后计算每次的模P意义下的行列式的和sum。
如果最后sum不等于0，那么说明很大概率上存在一个我们需要找的排列。

小C的利是

下面来证明正确性。

小C的利是

下面来证明正确性。
考虑一个排列P的贡献，如果他们的和是S，前面的数的乘积是C，那么这个排列在最终的sum里的贡献就是：
$C \times (1 + g^S + g^{2S} + \cdots + g^{(K-1)S})$

小C的利是

下面来证明正确性。
考虑一个排列P的贡献，如果他们的和是S，前面的数的乘积是C，那么这个排列在最终的sum里的贡献就是：
$C \times (1 + g^S + g^{2S} + \cdots + g^{(K-1)S})$
当$S \equiv 1 (mod\ K)$时，这个式子的值是K。
否则，对这个式子等比数列求和，就会得到$\frac{g^{K \cdot S} - 1}{g^S - 1}$，模$P$意义下的结果就是0.

小C的利是

下面来证明正确性。
考虑一个排列P的贡献，如果他们的和是S，前面的数的乘积是C，那么这个排列在最终的sum里的贡献就是：
$C \times (1 + g^S + g^{2S} + \cdots + g^{(K-1)S})$
当$S \equiv 1 (mod\ K)$时，这个式子的值是K。
否则，对这个式子等比数列求和，就会得到$\frac{g^{K \cdot S} - 1}{g^S - 1}$，模$P$意义下的结果就是0.
做多几次就可以判断有解了。
时间复杂度$O(n^3 K)$