原创 CF 242E (XOR ON SEGMENT)线段树维护区间异或，求和

2019-09-10 11:52:20 Laaahu_ 阅读数 51 更多

题目：进来看看呢，客官
题意：
一个定长的数组，然后他有两个操作；
①查询区间 【L，R】的区间和。
②一个更下新操作，对区间【L，R】内的数分别与一个固定的值X做异或操作，结果作为这个位置的新的值。

思路：
异或操作是在二进制的基础上进行的，所以我们开20棵线段树，每一颗线段树 去维护一个二进制位的1个数。
至于为什么存1的个数，我们来看看一个例子。
例如，一个数组有四个数字，1到4分别为5，6，7，8；
他们的 二进制分别是：
5=0101；6=0110，7=0111，8=1000；
这个数组的和是26； 5+6+7+8=26； 让我们来看一个 神奇的事情（其实很简单）
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
所有的二进制第一位 1 的个数之和为 1 ， 第二位 1 的个数之和 3 ， 第三位 1 的个数 2 ， 第四位一的个数 2；
$1 \times 2^3 + 3 \times 2^2 + 2 \times 2^1 + 2 \times 2^0 = 26$;
所以 我们知道了一的个数就可以计算得到在这个区间的区间和。
那么更新操作如何做呢 ， 我们为你试一下吧。
让我们用 4 对这个区间每个值求异或。
4 ^ 5 = 0100^0101 = 0001 = 1;
4 ^ 6 = 0100^0110 = 0010 = 2;
4 ^ 7 = 0100^0111 = 0011 = 3;
4 ^ 8 = 0100^1000 = 1100 = 12;
这时区间和为 18； 同样可以用上面的方法去验证。
这时我们来看看 一的个数的变化情况，
0 0 0 1
0 0 1 0
0 0 1 1
1 1 0 0
只有第二位一的个数发生了变化，因为这一位异或的是 1 而其余为为0 .
那么 它到底发生了什么变化。
之前为 3 现在为 1 而区间长度为 4 . 对 ， 就是你肉眼可见的简单。
每次异或为0可以不用管，因为0 XOR 0=0， 0 XOR 1=1;
异或位 为1 时 更新 区间的 1 的个数等于区间长度减去之前区间的 1 的个数。
因为区间更新 所以要 做懒惰标记 。
细节看代码吧 。
注意跟新时候的懒惰标记的更新。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;
typedef long long ll;
const int maxn = 1e5+10;
struct node{
    int b,e,num,lazy;
}t[25][maxn<<2];

int arr[maxn];
```

```
ll ans;

void push_up(int id,int node){
    t[id][node].num=t[id][node<<1].num+t[id][node<<1|1].num;
}

void create(int bb,int ee,int id,int node){
    t[id][node].b=bb;
    t[id][node].e=ee;
    t[id][node].lazy=0;
    t[id][node].num=0;
    if(bb==ee){
        if(arr[bb]&(1<<id)){
            t[id][node].num=1;
        }
        return ;
    }
    int mid = (bb+ee)>>1;
    create(bb,mid,id,node<<1);
    create(mid+1,ee,id,node<<1|1);
    push_up(id,node);
}

void push_down(int id,int node){
    if(t[id][node].lazy==1){
        int len;
        t[id][node<<1].lazy=t[id][node<<1].lazy^1;
        t[id][node<<1|1].lazy=t[id][node<<1|1].lazy^1;

        len=t[id][node<<1].e-t[id][node<<1].b+1;
        t[id][node<<1].num=len-t[id][node<<1].num;

        len=t[id][node<<1|1].e-t[id][node<<1|1].b+1;
        t[id][node<<1|1].num=len-t[id][node<<1|1].num;

        t[id][node].lazy=0;
    }
}

void update(int bb,int ee,int id,int node){
    if(bb<=t[id][node].b && ee>=t[id][node].e){
        t[id][node].lazy^=1;
        int len=t[id][node].e-t[id][node].b+1;
        t[id][node].num=len-t[id][node].num;
        return ;
    }
    int mid=(t[id][node].b+t[id][node].e)>>1;
    push_down(id,node);
    if(bb>mid) update(bb,ee,id,node<<1|1);
    else if(ee<=mid) update(bb,ee,id,node<<1);
    else {
        update(bb,mid,id,node<<1);
        update(mid+1,ee,id,node<<1|1);
    }
    push_up(id,node);
}

```


展开阅读全文