莫队

——西电双创周 div1

Made by：谭升阳（2019计科）

Codeforces ID: et3 tsy, et3 tsy2.0, et3 tsy3.0

哔哩哔哩：电音抖腿不能改

目录 CONTENTS

如果没特别说明，默认n = 1e5, m = 1e5, 默认n、m同阶

n 一般指的是数据元素的个数，m 一般指的是进行查询的次数
大家有问题可以直接提出。
如果对根号数据结构、莫队有一定了解的同学，可以自己学些别的，不要听tsy浪费时间。

本ppt借鉴了部分 lxl 的内容，仅用于学术交流，无商业用途。
（ps：lxl 是莫队二次离线的发明者）

PART ONE

引入

大家平常看到 1e5 级别的题一般采用的是 O(n log n) 的处理方式或者数据结构。

所以，我们今天讲点不同的，O(n sqrt(n))的数据结构如何来解决这类问题？

分块基础

我们把红色块这样的叫做整块，绿色的叫做散块。

那么，我们如果要对区间内容进行询问、修改，就可以利用这个结构来处理问题。

分块基础

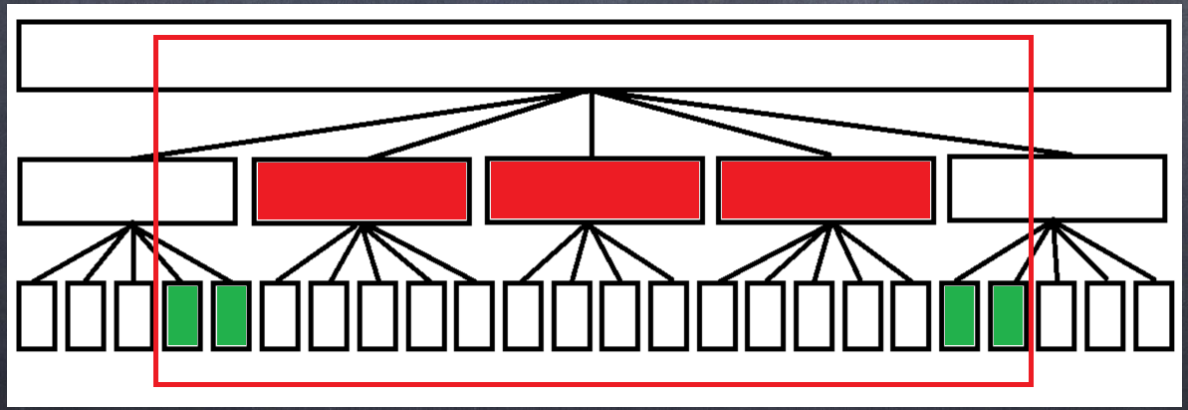
我们可以把这种结构看成一个二层或者三层的树。

单点修改的复杂度：O（1）

区间修改：O(sqrt(n))

（整块的数量至多为 sqrt(n)，散块的数量至多为 2 sqrt(n)，修改可以类似线段树的pushdown）

区间查询：O(sqrt(n))



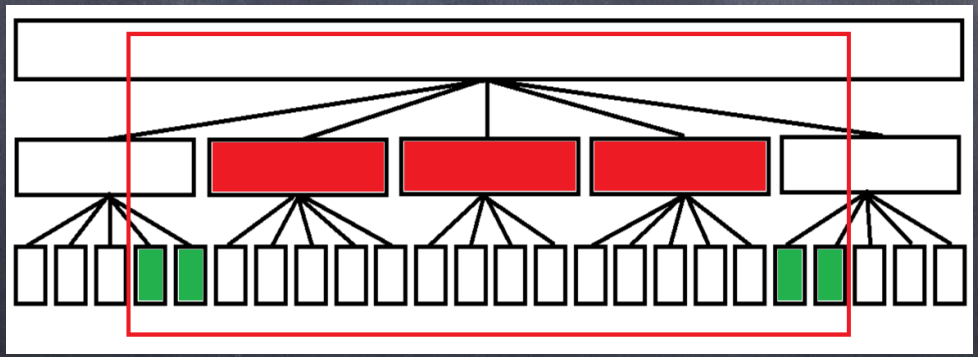
分块基础

基础问题：

维护一个序列

1.区间加

2.查询区间小于x的数个数

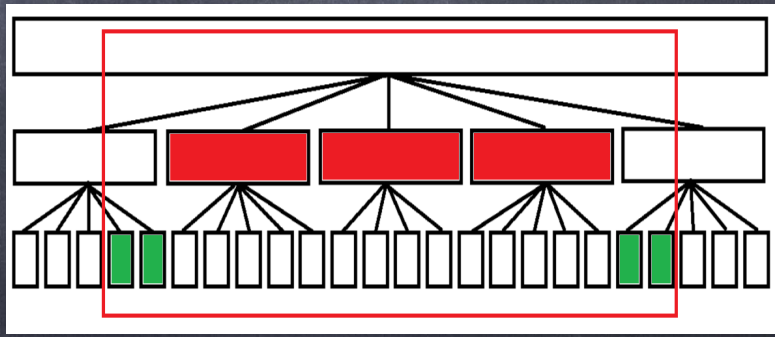


分块基础

1）整块：进行排序。每次区间加的时候，只打tag，不pushdown，整块内的相对相对大小是不变的。查询时，二分即可。（块数：sqrt(n), 二分 O(log sqrt(n))

2）散块：零散块在修改时可以把整块重构，最多两个整块会被修改。查询直接查询即可。（采用归并，线性，复杂度 m sqrt(n))

复杂度O(m sqrt(n) log sqrt(n))



使用上面的分块结构，我们可以处理很多问题，但是这都是基于维护的信息是**可进行区间线性运算的**，如和，乘积，异或和，类似于线段树的基本操作。

但是，面对另一类问题，即**不可进行区间线性运算的**，那么我们就会比较棘手，比如统计区间出现的颜色种类数，

所以，前IOI国家队队长莫涛就这类问题做了总结，后人以他的名字命名解决这类问题的方法，并不断延伸他总结的东西。

在国外题解里，莫队算法叫**mo's algorithm**

莫队是谁?

ACM相关。

关注问题 | 写回答 | 邀请回答 | 好问题 6 | 添加评论 | 分享 | ...

登录后你可以

不限量看优质回答　私信答主深度交流　精彩内容一键收藏

登录

查看全部 3 个回答

莫涛

可以叫我莫队

406 人赞同了该回答

我。

发布于 2014-08-04

▲ 赞同 406 ▼　24 条评论　分享　收藏　喜欢

考虑下面的问题：

给定一个大小为n序列，m次询问，每次询问你区间 [l,r] 有多少种不同的颜色

给定一个大小为n序列，m次询问，每次询问查询一个区间中每个数出现次数的平方和

给定一个大小为n序列，每次询问查询给定l, r, a, b, 查询区间[l,r]中值在[a,b]内的不同数个数

这些题乍一看，怎么好像只会暴力去做？

那恭喜你，直觉是对的，莫队就是运用分块思想的暴力。

莫队处理问题一般分为两类：

一是莫队维护区间答案

二是维护区间内的数据结构（常与分块结构组合食用）

PART TWO

普通莫队

给定一个大小为n序列，m次询问，每次询问你区间[l,r] 有多少种不同的颜色

我们考虑最朴素的做法。我们设置一个左端 l，右端 r，维护区间 [l, r] 的情况，即开一个数组 cnt[i], 统计颜色 i 出现的次数。

但是在下面的询问会退化：

1 100000
2 3
3 99999
4 5
5 99998
……

所以，莫队算法是如何减少这种“彻头彻尾”的移动开销呢？

我们对n个数进行分块，再对原询问进行如下排序。

```
block_size=sqrt(n);
bool operator<(const Query& sec)const
{
    if(l/block_size!=sec.l/block_size)return l<sec.l;
    else return r<sec.r;
}
```

对左端进行按块分区，区号越小越靠前，同区就看右端，右端越小越靠前。

```
block_size=sqrt(n);
bool operator<(const Query& sec)const
{
    if(l/block_size!=sec.l/block_size)return l<sec.l;
    else return r<sec.r;
}
```

在进行查询的时候，我们按照按**递增左端区号，再递增右端区号**的顺序查询

我们来看下这样移动左右两端的好处是什么：

1）左端：每两次查询之间如果左端对应区号不变，那么左端指针的移动的最大距离为块大小，考虑到m次查询，复杂度 O(n * sqrt (n))

如果区号改变了，因为我们区号递增，至多会改变 sqrt (n) 次，这里复杂度 O(n)

2）右端：每两次查询之间如果左端对应区号不变，右端只会单调往右走，每当左端区号改变，右端才会回来。因为我们区号最多改变 sqrt (n) 次，所以右端的移动复杂度 O(n * sqrt (n))

我们看到莫队总体复杂度大概为 n log (n) [排序] + m sqrt (n) [查询]

不难发现，莫队的常数比较大，并且块的大小比较有讲究（要分析复杂度）

具体实现也有挺多细节值得注意的，让我们看道例题吧。

P1494 [国家集训队]小Z的袜子

给定 n 只袜子，m 次询问，每只袜子有不同颜色 Ai，依次给定。每次询问给定 [l, r]，求该区间能抽到两只相同颜色袜子的概率。

```
6 4                2/5
1 2 3 3 3          0/1
2                  1/1
2 6                4/15
1 3
3 5
1 6
```

P1494 [国家集训队]小Z的袜子

我们先考虑怎么解决单次查询。显然，区间 [l,r] 的区间大小为 r – l + 1, 则拿袜子的总方案数是 C_{r-l+1}^{2}

颜色 i 的数量为 k， 对应能拿到匹配的袜子的方案数是 C_{k}^{2}，如果多一只，即 k + 1，那么对应的方案数是 C_{k+1}^{2}，多了k种方案数。那么少一只，就是逆过程。

```
void add(int cur)
{
    sum+=cnt[cur];
    cnt[cur]++;
}
void del(int cur)
{
    --cnt[cur];
    sum-=cnt[cur];
}
```

P1494 [国家集训队]小Z的袜子

我们再考虑怎么解决多次查询。利用莫队思想。

```
struct Query
{
    int l,r,id;
    bool operator<(const Query& sec)const
    {
        if(l/block_size!=sec.l/block_size)return l<sec.l;
        return r<sec.r;
    }
}q[maxn];
```

注意这里左端右端的移动顺序。先扩张后收缩，先左端后右端。
可以思考一下为什么。
如：从 [1,5] 移动到 [11,15]，不能出现 [11,5] 这样的非法过渡态

```
void add(int cur)
{
    sum+=cnt[cur];
    cnt[cur]++;
}
void del(int cur)
{
    --cnt[cur];
    sum-=cnt[cur];
}
```

```
sort(q,q+m);
for(int i=0;i<m;i++)
{
    if(q[i].l==q[i].r)
        continue;
    while(l>q[i].l)add(color[--l]);
    while(r<q[i].r)add(color[++r]);
    while(l<q[i].l)del(color[l++]);
    while(r>q[i].r)del(color[r--]);
    ans1[q[i].id]=sum;                    // 分子
    ans2[q[i].id]=1ll*(r-l+1)*(r-l)/2;    // 分母
}
```

让我们重新来思考一下开头看的三道题, 当作练习：

给定一个大小为n序列，m次询问，每次询问你区间 [l,r] 有多少种不同的颜色

给定一个大小为n序列，m次询问，每次询问查询一个区间中每个数出现次数的平方和

给定一个大小为n序列，每次询问查询给定l, r, a, b, 查询区间[l,r]中值在[a,b]内的不同数个数

给定一个大小为n序列，m次询问，每次询问你区间 [l,r] 有多少种不同的颜色

裸题。如果使用树状数组，可以参考下面思路。

P1972 [SDOI2009]HH的项链

此题首先应考虑到这样一个结论：

对于若干个询问的区间[l,r]，如果他们的r都相等的话，那么项链中出现的同一个数字，一定是只关心出现在最右边的那一个的，例如：

项链是：1 3 4 5 1

那么，对于r=5的所有的询问来说，第一个位置上的1完全没有意义，因为r已经在第五个1的右边，对于任何查询的[L,5]区间来说，如果第一个1被算了，那么他完全可以用第五个1来替代。

因此，我们可以对所有查询的区间按照r来排序，然后再来维护一个树状数组，这个树状数组是用来干什么的呢？看下面的例子：

1 2 1 3

对于第一个1，insert(1,1)；表示第一个位置出现了一个不一样的数字，此时树状数组所表示的每个位置上的数字（不是它本身的值而是它对应的每个位置上的数字）是：1 0 0 0

对于第二个2，insert(2,1)；此时树状数组表示的每个数字是1 1 0 0

对于第三个1，因为之前出现过1了，因此首先把那个1所在的位置删掉insert(1,-1),然后在把它加进来insert(3,1)。此时每个数字是0 1 1 0

如果此时有一个询问[2,3]，那么直接求sum(3)-sum(2-1)=2就是答案。

代码可参考（https://www.luogu.com.cn/problem/solution/P1972）

给定一个大小为n序列，m次询问，每次询问你区间 [l,r] 有多少种不同的颜色

裸题。如果使用主席树，可以参考下面思路。

主席树每个结点维护当前版本下位置 i 的颜色出现情况

更新：以数组a为基础建立线段树，然后从左往右扫描 a

当扫到颜色 Ai 时，如果 Ai 是第一次出现，那么直接在新的线段树上在 i 位置 +1

如果值 Ai 在前面位置 p 出现过，那么在新的线段树上将 p 位置 -1，在 i 位置 +1

（这样保证了同种颜色在同一个版本对应的树上产生的贡献为1，且是由最右边对应位置贡献的）

询问：[l, r]

在第 r 棵线段树的 rt[1] 是 [1,r] 区间上的不同的数的个数

要求 [l, r]上不同的数的个数，询问第 r 棵线段树区间 [l, r] 的和即可

代码可参考（https://www.cnblogs.com/zsben991126/p/10750164.html）

给定一个大小为n序列，m次询问，每次询问查询一个区间中每个数出现次数的平方和

与例题类似，分析平方差。

给定一个大小为n序列，每次询问查询给定l, r, a, b, 查询区间[l, r]中值在[a, b]内的不同数个数

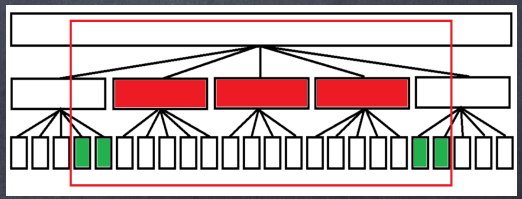
跑个莫队，维护树状数组，O(n sqrt(n) log n) 慢

这类莫队就是维护数据结构的题：用莫队进行移动区间，用数据结构来维护值域

莫队总共修改O(n sqrt(n))次，而且是单点修改的
查询只有O(m)次

如果单点修改复杂度为O(1), 区间查询复杂度是O(Sqrt(n)), 那么总复杂度就是 O(n sqrt(n))

这不就是我们刚刚讲过的根号数据结构吗?



总结一下莫队的使用场景：

1）当我们有 ans[l, r], 我们可以在常数时间获得 ans[l ±1, r], ans[l, r ± 1]

2）数据范围在1e5左右

3）离线，即所有询问一次给出

PART THREE

带修莫队

P1903 [国家集训队]数颜色 / 维护队列

墨墨购买了一套N支彩色画笔(其中有些颜色可能相同)，摆成一排，你需要回答墨墨的提问。墨墨会向你发布如下指令：

1、 Q L R代表询问你从第L支画笔到第R支画笔中共有几种不同颜色的画笔。

2、 R P Col 把第P支画笔替换为颜色Col

```
6 5                4
1 2 3 4 5 5        4
Q 1 4              3
Q 2 6              4
R 1 2
Q 1 4
Q 2 6
```

P1903 [国家集训队]数颜色 / 维护队列

带修莫队的本质实际上是在原有l，r的基础上再引入一维，即时间t(版本)，每个询问记录对应的l，r，t。与主席树有类似之处。

我们的排序就变成了以下的形式：

```
struct Query
{
    int l,r,id,t;
    int posl,posr;
    bool operator<(const Query& sec)const
    {
        if(posl!=sec.posl)return l<sec.l;
        if(posr!=sec.posr)return r<sec.r;
        return t<sec.t;
    }
}q[maxn];
q[tot].posl=q[tot].l/block_size;
q[tot].posr=q[tot].r/block_size;
```

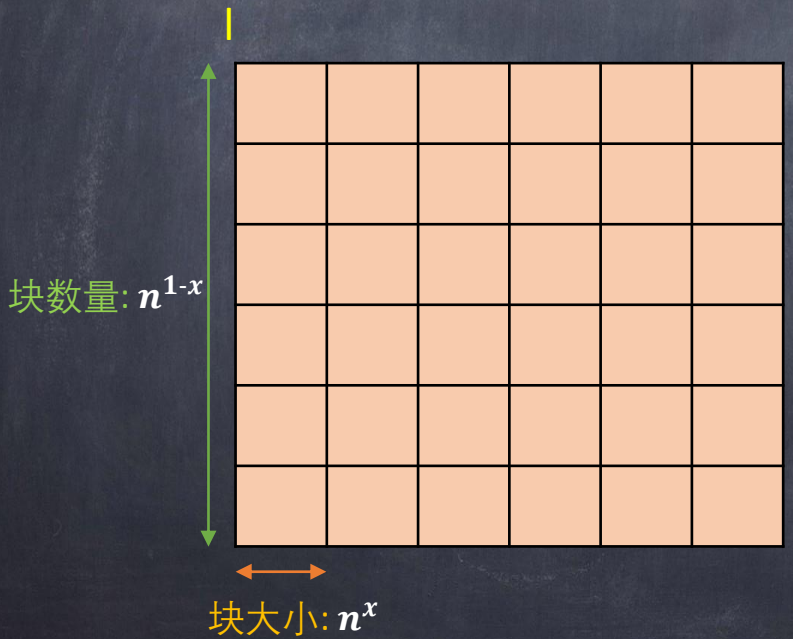
原来的l, r 移动，就变成了三维l，r，t 的移动

```
while(l>q[i].l)add(color[--l]);
while(r<q[i].r)add(color[++r]);
while(l<q[i].l)del(color[l++]);
while(r>q[i].r)del(color[r--]);
while(t<q[i].t)work(++t);
while(t>q[i].t)work(t--);
// 值得注意，t要初始化为0
```

提供两种复杂度的说明

我们假设块大小为n^x，n，m，t 同阶，那么块的数量为n^{1-x}

从数形结合上，把l，r，t看成三维



l: 块内 $n^x \times n$
块外 $n^x \times n^{1-x} = n^1$

r: 块内 $n^x \times n$
块外 $n^1 \times n^{1-x} = n^{2-x}$

t: $n^{1-x} \times n^{1-x} \times n = n^{3-2x}$

要使max{1+x,2-x,3-2x}最小

因为0<x<1, x取 $\frac{2}{3}$ 最优

从代码逻辑上

我们假设块大小为n^x，n，m，t 同阶，那么块的数量为n^{1-x}

- 对于L指针
  - 在块内移动时，每一次移动的复杂度应为$O(n^x)$，由于共有m次询问，因此总复杂度为$O(n^{x+1})$。
  - 到下一个块时，每一次移动的复杂度应为$O(n^x)$，由于块的大小为$O(n^x)$，因此总块数为$O(\frac{n}{n^x})$，因此总复杂度为$O(n)$。

$\therefore$ L指针的总复杂度为$O(n^{x+1})$。

我们假设块大小为n^x，n，m，t同阶，那么块的数量为n^{1-x}

- 对于R指针
  - L和R全都在块内移动时，每一次移动的复杂度应为$O(n^x)$，由于这样的情况共有$O((\frac{n}{n^x})^2)$，即$O(n^{2-2x})$次，因此总复杂度为$O(n^{2-x})$。
  - L块相同且R到下一块时，每一次移动的复杂度应为$O(n^x)$，由于总块数为$O(\frac{n}{n^x})$，即$O(n^{1-x})$，因此总复杂度为$O(n^{2-x})$
  - L指针移动到下一个块时，每一次移动的复杂度应为$O(n)$，由于这样的情况共有$O(\frac{n}{n^x})$，即$O(n^{1-x})$次，因此总复杂度为$O(n^{2-x})$。

$\therefore$ R指针的总复杂度为$O(n^{2-x})$。

我们假设块大小为n^x，n，m，t同阶，那么块的数量为n^{1-x}

- 对于K指针
  - L和R全都在块内移动时，此时K指针应该是递增的（因为排序时对于这样的情况我们$return\ x.k < y.k$），所以总复杂度为$O(n)$。
  - L块相同且R到下一块时，每一次移动的复杂度应为$O(n)$，由于这样的情况有$O(n^{2-2x})$次，因此总复杂度为$O(n^{3-2x})$。
  - L指针移动到下一个块时，每一次移动的复杂度应为$O(n)$，由于这样的情况共有$O(n^{1-x})$次，因此总复杂度为$O(n^{2-x})$。

$\therefore$ K指针的总复杂度为$O(n^{max(2-x,3-2x)})$。

我们假设块大小为n^x，n，m，t同阶，那么块的数量为n^{1-x}

综上所述，算法的总时间复杂度应为$O(n^{max(x+1,2-x,3-2x)})$，那么我们的目的就是找到一个x（$0 < x < 1$）使$max(x+1, 2-x, 3-2x)$最小。

此时的x应取$\frac{2}{3}$，所以块的大小就是$O(n^{\frac{2}{3}})$。

l, r 的移动依然一样

```
inline void add(int cur)
{
    if(++cnt[cur]==1)sum++;
}
inline void del(int cur)
{
    if(--cnt[cur]==0)sum--;
}
```

值得注意的是版本的移动，进行的修改会稍微麻烦点

```
pos[t] // 记录版本号为t的更新的位置号
nxt[t] // 暂存版本t对换信息，新换旧要把旧的存回来，旧换新要把新的存回来

inline void work(int cur)
{
    if(l<=pos[cur]&&pos[cur]<=r)
    {
        if(--cnt[color[pos[cur]]]==0)sum--;
        if(++cnt[nxt[cur]]==1)sum++;
    }
    std::swap(nxt[cur],color[pos[cur]]);
}
```

PART FOUR

回滚莫队

AT1219 歴史の研究

给定一个大小为 n 的序列，给定 m 次查询，每次查询给定 [l, r]，问此区间重要度最大是多少。重要度定义：该区间某个数出现的次数乘以这个数本身

```
5 5
9 8 7 8 9
1 2
3 4
4 4
1 4
2 4
```

```
9
8
8
16
16
```

这道题我们很容易发现，当区间扩大时，只需要统计每个数出现的次数，用次数乘以数大小来更新mx即可。但是，当区间减小时，我们就无法处理了，我们无法得知第二大对应的内容。

5 6 3 6 5 1 2 5

所以，回滚莫队\不删除莫队解决的就是这类维护区间信息只增不减或者只减不增问题

回滚莫队的算法流程：依然是按照普通莫队的思想，对左端点按块进行分区，依据相同规则排序

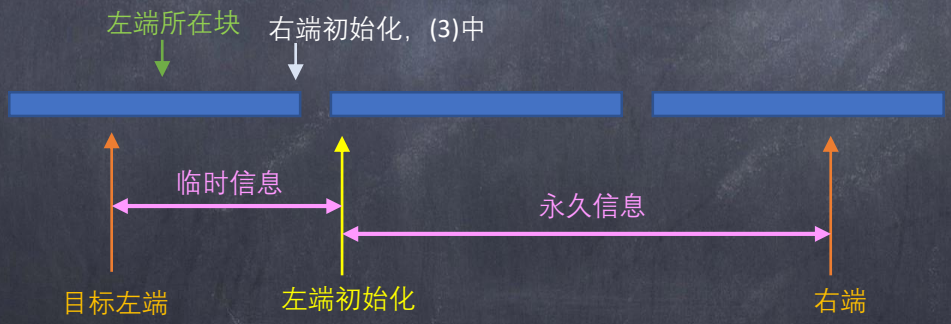
1）如果左端 l、右端 r 均在同一块中，直接暴力，扫描区间回答询问，结束。

2）如果当前左端 l 所属的块与前一次查询 l 不在同一块中，清空记录的内容，修改左右端，继续（3）

3）将左端移动到当前查询块的下一块的首位置，在移动的过程中将临时信息清空，永久信息保留

4）右端向右移动，更新永久信息

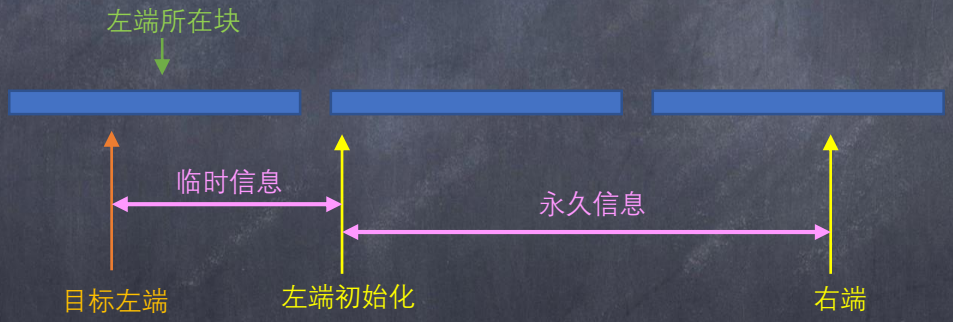
5）左端向左移动，更新临时信息，将临时信息和永久信息的内容合并得到最优解



我们来分析一下复杂度

1）右端。如果左端所在块不变，右端单调递增，是线性的。如果左端所在块变化，那么右端要消耗线性复杂度来清空，左端至多变化 sqrt(n) 次，因此，总复杂度为 O(n sqrt(n))

2）左端。显然每次查询过程中，遍历的范围最多是 sqrt(n), 因此，总复杂度为 n sqrt(n)



AT1219 歴史の研究

如果左端 l、右端 r 均在同一块中，直接暴力，扫描区间回答询问，结束

```
int brute(int l,int r)
{
    int mxx=0;
    rep(i,l,r)
    {
        __cnt[color[i]]++;
        mxx=max(mxx,__cnt[color[i]]*val[i]);
    }
    rep(i,l,r)__cnt[color[i]]=0;
    return mxx;
}
```

AT1219 歴史の研究

如果当前左端 l 所属的块与前一次查询 l 不在同一块中，清空记录的内容，修改左右端

```
if((q[i].l/block_size+1)*block_size!=l) // l goes into another block
{
    mx=0;
    while(r>=l)cnt[color[r]]=0,r--; // reset
    l=(q[i].l/block_size+1)*block_size;
    r=l-1;
}
```

AT1219 歷史の研究

移动右端，更新永久信息

```
while(r<q[i].r)
{
    ++r;
    cnt[color[r]]++;
    mx=max(mx,cnt[color[r]]*val[r]);
}
```

curmx记录当前答案，初值是右边永久信息对应的最大值mx，我们现在要统计 [q[i].l, l-1] 区间的临时信息

```
int curmx=mx;
rep(j,q[i].l,l-1)
{
    tmpcnt[color[j]]++;
    curmx=max(curmx,(cnt[color[j]]+tmpcnt[color[j]])*val[j]);  // 这里就体现了信息的合并
}
rep(j,q[i].l,l-1)
{
    tmpcnt[color[j]]=0;
}
```

P5906 【模板】回滚莫队&不删除莫队

题目描述

给定一个序列，多次询问一段区间 $[l,r]$，求区间中**相同的数的最远间隔距离**。

序列中两个元素的**间隔距离**指的是**两个元素下标差的绝对值**。

输入格式

第一行一个整数 n，表示序列长度。

第二行 n 个整数，描述这个序列。

第 $n+1$ 行一个整数 m，表示询问个数。

第 $n+2 \sim n+m+1$ 行，每行两个整数 l,r 表示询问区间。

```
8
1 6 2 2 3 3 1 6
5
1 4
2 5
2 8
5 6
1 7
```

```
1
1
6
1
6
```

P5906 【模板】回滚莫队&不删除莫队

这道题与上道题差不多，只是对每个颜色要维护两个信息，即最左端，最右端

```
int __l[maxn],__r[maxn];
int brute(int l,int r)
{
    int mxx=0;
    rep(i,l,r)
    {
        if(!__l[color[i]])__l[color[i]]=i;
        __r[color[i]]=i;
        mxx=max(mxx,__r[color[i]]-__l[color[i]]);
    }
    rep(i,l,r)__l[color[i]]=0,__r[color[i]]=0;
    return mxx;
}
```

P5906 【模板】回滚莫队&不删除莫队

```
if((q[i].l/block_size+1)*block_size!=l) // l goes into another block
{
    mx=0;
    while(r>=l)curr[color[r]]=0,curl[color[r]]=INF,r--; // reset
    l=(q[i].l/block_size+1)*block_size;r=l-1;
}
while(r<q[i].r)
{
    ++r;
    curl[color[r]]=min(r,curl[color[r]]);
    curr[color[r]]=r;mx=max(mx,curr[color[r]]-curl[color[r]]);
}
int curmx=mx;
rep(j,q[i].l,l-1)
{
    tmpl[color[j]]=min(tmpl[color[j]],j);
    tmpr[color[j]]=max(tmpr[color[j]],j);
    curmx=max(curmx,max(tmpr[color[j]],curr[color[j]])-min(tmpl[color[j]],curl[color[j]]));
}
rep(j,q[i].l,l-1)
{
    tmpl[color[j]]=INF;
    tmpr[color[j]]=0;
}
ans[q[i].id]=curmx;
```

CF1514D Cut and Stick(cf 2000)

原题地址：https://codeforces.com/contest/1514/problem/D

本题是作业题之一：有四种做法，建议都可以试一试，都有实际用途。核心问题：区间众数问题

题意翻译

题意

给定一个长度为 n 的序列 $(n \leq 3 \times 10^5)$，可以对其进行三种操作：

要求：给定 q 个询问 $(q \leq 3 \times 10^5)$，每次给出一个区间的左右端点，将这个区间分成若干个片段，使得每个片段内任意元素出现的次数不严格大于 $\lceil \frac{x}{2} \rceil$ (x 为该片段长度)。求可以分成的最少片段数目。

输入

第一行输入两个整数 n 和 p。第二行输入 n 个整数，对应在原序列中的值。第 $3 \sim q+2$ 行，每行输入两个整数，分别表示询问的区间左端点和右端点。

输出

对于每个询问，输出一个整数，表示询问区间分成的最少片段。

```
6 2
1 3 2 3 3 2
1 6
2 5
```

```
1
2
```

CF1514D Cut and Stick

要求：给定 q 个询问 ($q \leq 3 \times 10^5$)，每次给出一个区间的左右端点，将这个区间分成若干个片段，使得每个片段内任意元素出现的次数不严格大于 $\lceil \frac{x}{2} \rceil$ (x 为该片段长度)。求可以分成的最少片段数目。

首先，本题的核心结论是，记某个区间的众数数量为m，该区间的大小为x

这个结论可以分奇偶看一看，当时官方题解好像没有证明，非重点，跳过。

```
int getans(int x,int m)
{
    if(m<=(x+1)/2)return 1;
    return 2*m-x;
}
```

所以，现在核心问题就是如何求解区间众数？四类思路

1）1/2的概率能抽到众数，抽不到就是1，多抽几次（jyz yyds!）

2）主席树上二分 （pllj yyds!）

3）普通莫队 （Wings yyds!）

4）回滚莫队

CF1514D Cut and Stick

普通莫队思路

普通莫队能很好的统计到每个颜色出现的次数，但是无法得知众数是谁，或者说，无法得知，众数出现的次数。

那么我们不妨开一个数组num记录一下每类数出现次数对应的个数。

例如：2 2 3 3 3 4 5 5

2出现了2次，3出现了3次，4出现了1次，5出现了2次

那么num[1] = 1, num[2] = 2, num[3] = 1

如果当前众数出现的次数为 x 次，那么显然 num[x] > 0

当我们 x 减少时，说明num[x]--, num[x-1]++

那么我们不妨开一个数组num记录一下每类数出现次数对应的个数。

例如：2 2 3 3 3 4 5 5

2出现了2次，3出现了3次，4出现了1次，5出现了2次

那么num[1] = 1, num[2] = 2, num[3] = 1

如果当前众数出现的次数为 x 次，那么显然 num[x] > 0

当我们 x 减少时，说明num[x]--, num[x-1]++

```
void add(int cur)
{
    num[cnt[cur]]--;
    cnt[cur]++;
    num[cnt[cur]]++;
    mx=max(mx,cnt[cur]);
}
void del(int cur)
{
    num[cnt[cur]]--;
    cnt[cur]--;
    num[cnt[cur]]++;
    if(!num[mx])mx--; // 通过这样移动找众数的个数，这里只需要if即可，不用while
}
```

复杂度说明：增加或删除这两个操作至多 O(n sqrt(n))次

CF1514D Cut and Stick

回滚莫队思路

比较显然，我们只需要合并临时信息和永久信息，找众数出现次数就简单了。

```
if((q[i].l/block_size+1)*block_size!=l)
{
    mx=0;
    while(r>=l)cnt[color[r--]]--;
    l=(q[i].l/block_size+1)*block_size;r=l-1;
}
while(r<q[i].r)
{
    ++r;cnt[color[r]]++;mx=max(mx,cnt[color[r]]);
}
int curmx=mx;
rep(j,q[i].l,l-1)
{
    cnt2[color[j]]++;curmx=max(curmx,cnt[color[j]]+cnt2[color[j]]);
}
rep(j,q[i].l,l-1)
{
    cnt2[color[j]]--;
}
ans[q[i].id]=getans(q[i].r-q[i].l+1,curmx);
```

PART FIVE

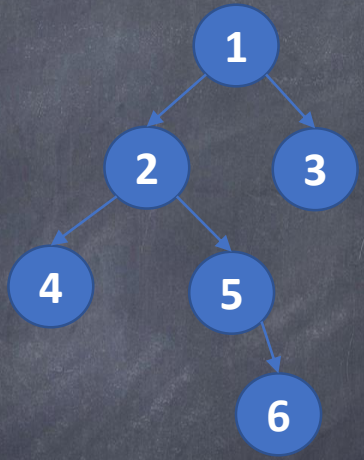
树上莫队

欧拉序介绍

欧拉序又叫括号序，与dfs序类似，记录每个元素在dfs中的相对顺序。
区别在于每个节点要记录入栈、出栈

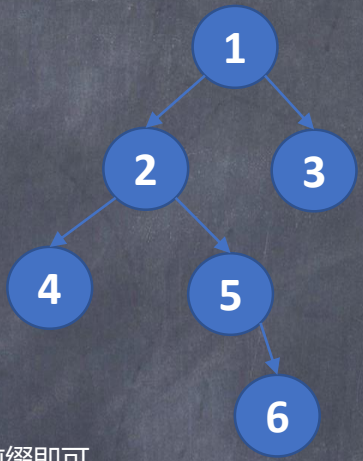
时间戳：1 2 3 4 5 6 7 8 9 10 11 12
欧拉序：1 2 4 4 5 6 6 5 2 3 3 1

每个元素记录对应入出的时间戳：
1 -> (1, 12)
2 -> (2, 9)
3 -> (10, 11)
4 -> (3, 4)
5 -> (5, 8)
6 -> (6, 7)



时间戳：1 2 3 4 5 6 7 8 9 10 11 12
欧拉序：1 2 4 4 5 6 6 5 2 3 3 1

1 -> (1, 12)
2 -> (2, 9)
3 -> (10, 11)
4 -> (3, 4)
5 -> (5, 8)
6 -> (6, 7)



用途：

1）**求某个点到根节点的权值和。**

方法：进做加法，出做减法，查询某点就只需要查询对应的前缀即可。

2）**求某个子树的权值和。**

方法：需要在进的点处做加法，求某个点最后一次出现的位置的前缀和减去第一次出现的位置的前一个位置的前缀和即可。

3）**记录任意两个点在树上的关系**

方法：pair记录入出，比较pair内容，包含关系——在外的是祖宗，不相交——不存在祖先关系。

4）树上莫队

将链关系转化为特定区间关系

Trees of Tranquillity(Codeforces #722 div1 C, 2300)

原题：https://codeforces.com/contest/1528/problem/C　视频讲解：https://www.bilibili.com/video/BV1QA411g7tg

题意翻译

给你两棵 n 个节点的树，要你找一个最大的点集 S，使得对于任意 $u, v \in S$，都满足 u, v 在第一棵树上一个是另一个的祖先并且在第二棵树上互相都没有祖先关系，输出 S 的大小即可。

本题有多组输入。

数据范围： $n \leq 3 \times 10^5$。

```
5
1 2 3 4
1 1 1 1
```

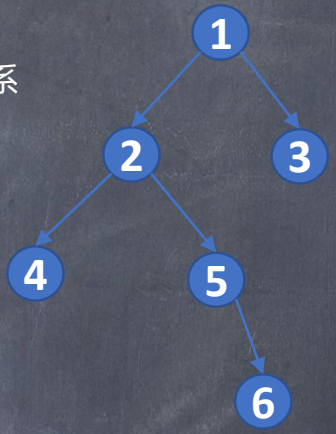
4

选2,3,4,5

Trees of Tranquillity

这道题就能很好体现欧拉序的引用——记录任意两个点在树上的关系

```
void dfs(int cur)
{
    in[cur]=++stamp;save[stamp]=cur;
    for(auto nxt:node[cur])
    {
        dfs(nxt);
    }
    out[cur]=stamp;
}
```



时间戳：1 2 3 4 5 6 7 8 9 10 11 12
欧拉序：1 2 4 4 5 6 6 5 2 3 3 1

1 -> (1, 12)
2 -> (2, 9)
3 -> (10, 11)
4 -> (3, 4)
5 -> (5, 8)
6 -> (6, 7)

考虑dfs过程，本质是栈进出的过程。

那么任意两个数的入出数对，只能有两种关系：

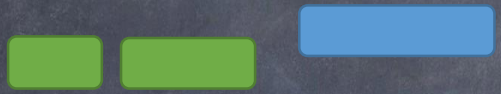
1）包含

2）不相交

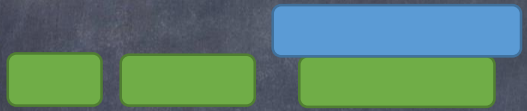
不存在 相交而不包含 的情况

Trees of Tranquillity

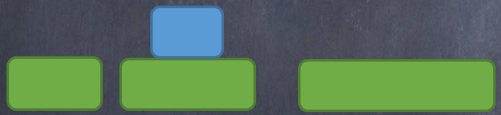
(1)插在尾部



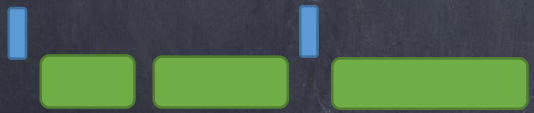
(2)覆盖了其他区间，劣于旧区间



(3)贪心，替换原区间



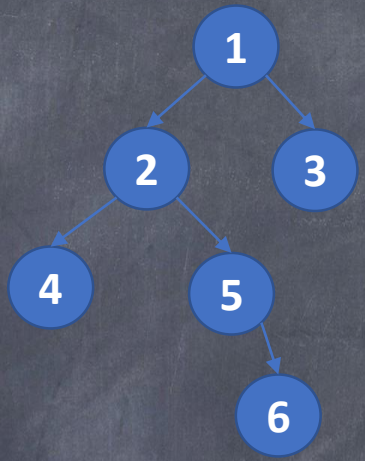
(4)插在旧区间中间或首部



```
void dfs2(int cur)
{
    int del=-1,add=-1;
    auto it=s.upper_bound(in[cur]);
    if(it==s.end()||*it>out[cur])
    {
        if(it!=s.begin())
        {
            it--; // save 时间戳
            if(out[save[*it]]>=out[cur])
            {
                del=*it;
                s.erase(del);
            }
        }
        add=in[cur];
        s.insert(add);
    }
    ans=max(ans,(int)s.size());
    for(auto nxt:node2[cur])
    {
        dfs2(nxt);
    }
    if(del!=-1)s.insert(del);  // 回溯还原
    if(add!=-1)s.erase(add);
}
```

树上莫队欧拉序的使用

时间戳：1 2 3 4 5 6 7 8 9 10 11 12
欧拉序：1 2 4 4 5 6 6 5 2 3 3 1
每个元素记录对应入出的时间戳，记为in，out：
1 -> (1, 12)
2 -> (2, 9)
3 -> (10, 11)
4 -> (3, 4)
5 -> (5, 8)
6 -> (6, 7)



树上莫队问题一般是对树上的链信息进行维护
甚至可以带修，即树上带修莫队，如WC2013糖果公园

链信息有两类，在使用欧拉序时，要分开处理：
即我们现在研究的是点x，y形成的链，他们的最近公共祖先是lca
1）两端存在祖先关系，即 x==lca 或 y==lca
不妨我们假设x==lca，那么 [in[x], in[y]] 这段区间刚好包含了一次链上元素，两次无关元素，例如1和5，12445，无关的4出现了两次。我们在区间统计时，可以利用异或的思想，将无关点的信息清除。

2）两端不存在祖先关系
那么x，y的 [in, out] 是不相交的。不妨假设 in[x] < in[y]，那么[out[x], in[y]] 这段区间刚好包含了一次除lca外链上元素，两次无关元素，如4和3，4566523，无关的5、6各出现了两次。在统计时，同样利用异或思想，还要补上lca的信息

我们看到欧拉序的好处就在于把树拍平，变成数组上莫队的问题，而且开销的空间只是原来的两倍

SP10707 COT2 - Count on a tree II

https://www.spoj.com/problems/COT2/

- 给定 n 个结点的树，每个结点有一种颜色。
- m 次询问，每次询问给出 u, v，回答 u, v 之间的路径上的结点的不同颜色数。
- $1 \leq n \leq 4 \times 10^4$，$1 \leq m \leq 10^5$。

```
8 2                    4
105 2 9 3 8 5 7 7      4
1 2
1 3
1 4
3 5
3 6
3 7
4 8
2 5
7 8
```

SP10707 COT2 - Count on a tree II


```
void dfs1(int pos,int fa)
{
    in[pos]=++stamp;save[stamp]=pos;
    /*
        dfs sons
    */
    out[pos]=++stamp;save[stamp]=pos;
}

for(int i=0;i<m;i++)
{
    int x=read(),y=read(),lca=query_LCA(x,y);
    if(x==lca||y==lca)
    {
        if(y==lca)swap(x,y);
        q[i].l=in[x],q[i].r=in[y];
    }
    else
    {
        if(in[y]<in[x])swap(x,y);
        q[i].lca=lca;
        q[i].l=out[x],q[i].r=in[y];
    }
    q[i].id=i;
}
```

```
// 因为是采用异或思想，add与del实质一样
void add(int cur)
{
    if(appr[cur])
    {
        cnt[color[cur]]--;
        if(cnt[color[cur]]==0)num--;
    }
    else
    {
        cnt[color[cur]]++;
        if(cnt[color[cur]]==1)num++;
    }
    appr[cur]^=1;
}
void del(int cur)
{
    add(cur);
}
```

WC2013 糖果公园

https://www.luogu.com.cn/problem/P4074

给你一棵树,每个点有个颜色

每次询问你一条路径求$\sum_c val_c \sum_{i=1}^{cnt_c} worth_i$

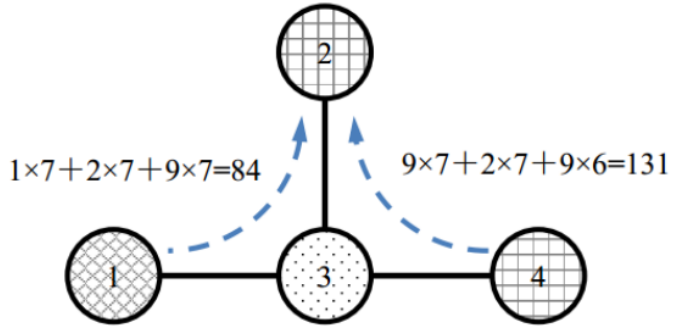
> val表示该颜色的价值,cnt表示其出现的次数,$woeth_i$表示第i次出现的价值

带修改

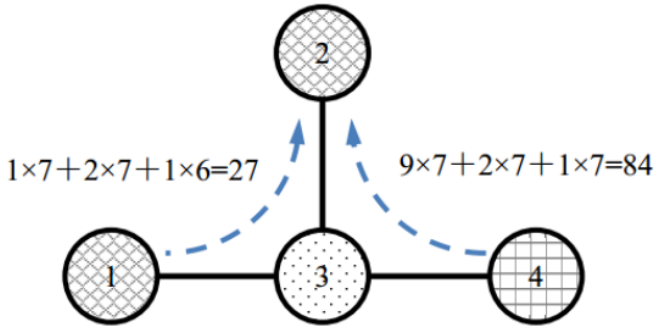
```
4 3 5          84
1 9 2          131
7 6 5 1        27
2 3            84
3 1
3 4
1 2 3 2
1 1 2
1 4 2
0 2 1
1 1 2
1 4 2
```

我们分别用

代表 C_i 为 1、 2、 3 的节点，在修改之前：



在将 C_2 修改为 1 之后：



WC2013 糖果公园

这是一道带修树上莫队，综合了我们前面所讲的树上莫队+带修莫队。

```
inline void add(int cur)  // cur: the node
{
    if(vis[cur]==0)
    {
        sum+=1ll*v[color[cur]]*w[++cnt[color[cur]]];
    }
    else sum-=1ll*v[color[cur]]*w[cnt[color[cur]]--];
    vis[cur]^=1;
}

inline void work(int cur) // cur: the version
{
    int num=ver[cur],clr=save[cur];
    if((l<=in[num]&&in[num]<=r)^(l<=out[num]&&out[num]<=r))
    {
        sum+=1ll*v[clr]*w[++cnt[clr]];
        sum-=1ll*v[color[num]]*w[cnt[color[num]]--];
    }
    swap(save[cur],color[num]);
}
```

一般写树上莫队时，求lca十分推荐采用树链剖分，常数小很多，不容易被卡

et3_tsy 2020-10-24 10:33	Accepted 100	P4074 [WC2013] 糖果公园	4.18s / 128.71MB / 3.94KB C++11 O2
Accepted 100	P4074 [WC2013] 糖果公园	9.04s / 13.05MB / 3.00KB C++11	
Accepted 100	P4074 [WC2013] 糖果公园	11.88s / 13.60MB / 3.00KB C++11	
Accepted 100	P4074 [WC2013] 糖果公园	10.39s / 13.06MB / 3.00KB C++11	
Accepted 100	P4074 [WC2013] 糖果公园	10.44s / 13.85MB / 2.91KB C++11	
Accepted 100	P4074 [WC2013] 糖果公园	19.43s / 108.62MB / 3.79KB C++11 O2	
Accepted 100	P4074 [WC2013] 糖果公园	25.13s / 107.91MB / 3.88KB C++11 O2	
Accepted 100	P4074 [WC2013] 糖果公园	14.09s / 15.24MB / 2.59KB C++11	
Accepted 100	P4074 [WC2013] 糖果公园	13.37s / 15.96MB / 2.57KB C++11 O2	

PART SIX

补充

提醒：开了O2的莫队，真的很快，如果数据不刻意卡，1e6甚至都可以跑。但是如果数据比较强，可能会T。（如：P1972 [SDOI2009]HH的项链）

如果你很自信，题目就是用莫队，然后你T了，你需要卡常技巧

卡常技巧

1）按照奇偶性排序，很玄学

```
bool cmp(const node &fir,const node &sec)
{
    if(fir.b!=sec.b) return fir.b<sec.b;
    if(fir.b&1) return fir.r<sec.r;
    return fir.r>sec.r;//请注意这两行
}
```

卡常技巧

2）仔细推导块大小

莫队的核心就是块大小，我们在带修莫队那里，很仔细的展示了怎么推导。设块大小为n^x，n，m，t 同阶，那么块的数量为n^{1-x} 。进一步去分析，每个指针移动的最劣复杂度是多少，最后让这些max的尽可能小对应的x取值就是我们要的。

以及网上有些给出的说法，可能会快？可以参考一下

```
block = n / sqrt( m * 2 / 3 );
```

卡常技巧

3）减少函数调用

比如这里：

```
while(l>q[i].l)add(id[--l]);
while(r<q[i].r)add(id[++r]);
while(l<q[i].l)add(id[l++]);
while(r>q[i].r)add(id[r--]);
```

冷门内容，比较偏，不细讲(主要是tsy太菜了不会)

大家可以参考一下这两篇blog

莫队二次离线:

https://blog.csdn.net/BWzhuzehao/article/details/115003628

莫队在线化改造:

https://www.luogu.com.cn/blog/asadashino/moqueue

THANKS!