一、分块

考虑这样一个问题：

> 长度为n的数列，考虑两种操作，区间整体加一个数和询问一段区间的和。

前面我们学过了一种叫线段树的数据结构，可以进行区间加法和区间求和。但是在这里我们介绍另一种方式，分块。

而分块，顾名思义，就是把一段序列分成一整块一整块得来处理，预处理一整块的答案，维护一段作为一个整体，只记录维护整体的有关信息，就是分块。

查询：左边界的块内暴力查询，右边界块内暴力查询，中间覆盖整块的区间用块信息整体查询。

修改：左边界的块内暴力修改，右边界块内暴力修改，中间覆盖整块的区间用持久化标记等方式整体修改。

因为是暴力维护和整块取信息，所以可以处理一些线段树处理不了的类型。

复杂度： 设每个块长度为m,分为n/m块

块内遍历两次，每次最多m次，整块合并最多n/m次。那么总的时间复杂度就是 O(m+n/m),当m为sqrt(n)时，取最小值。故应该分为sqrt(n),块，每块sqrt(n)个元素。

二、莫队

一种离线算法，在预先知道所有询问的情况下，通过合理的安排询问的顺序，来降低复杂度。比如询问区间 [l , r] ，如果从[l , r]转移到 [l + 1 , r] , [l - 1 , r] , [l , r + 1] , [l , r - 1],这四个区间的答案只需要O(1)的复杂度，我们就可以通过合适的转移，来很快的从已知的[l , r]转向一个还未知的询问区间 [l ' , r '] , 那么就可以使用莫队算法。

1: 普通莫队

对询问排序，以 L为第一关键字，若L在同一个块内，则将R从左向右排序。对于排序后的询问，先处理第一个询问，然后暴力从上一个区间的答案转移到下一个区间的答案。分块大小应为sqrt(n),若n与m差距很大，那么可以适当调整块的大小。

例题：P1494 小Z的袜子

> 作为一个生活散漫的人，小 Z 每天早上都要耗费很久从一堆五颜六色的袜子中找出一双来穿。终于有一天，小 Z 再也无法忍受这恼人的找袜子过程，于是他决定听天由命……
>
> 具体来说，小 Z 把这 N只袜子从1到 N编号，然后从编号 L 到 R (L 尽管小 Z 并不在意两只袜子是不是完整的一双，甚至不在意两只袜子是否一左一右，他却很在意袜子的颜色，毕竟穿两只不同色的袜子会很尴尬。
>
> 你的任务便是告诉小 Z，他有多大的概率抽到两只颜色相同的袜子。当然，小 Z 希望这个概率尽量高，所以他可能会询问多个 (L,R) 以方便自己选择。

对于区间 [l , r]的答案，很容易得出为

segma(sum [i] * (sum [i] - 1) /2) / ((r - l +1) * (r - l))

然后暴力转移。

优化方案：奇偶排序。

2: 带修莫队

P1903：

> 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。
>
> 为了满足墨墨的要求，你知道你需要干什么了吗？

树套树：时间：O(nlog(n^2)) 空间：O(nlog(n^2))

莫队 ：时间：O(n sqrt(n)) 空间：n

于是乎我们可以套用普通莫队，在l和r两个指针的基础上再加一个指针作为时间指针，相当于添加一个时间轴，按照顺序存下每次的修改，将时间每次O(1)的时间复杂度，在当前修改到前一次修改、后一次后盖之间改动，直到时间在当前查询的前一次修改的时间位置。查询的排序按照 l , r , t ，将 l所处块作为第一键值，r所处块作为第二键值，t作为第三键值来进行排序。

```
int cmp(node a,node b){
    if(a.l/divi!=b.l/divi)return a.l/divi<b.l/divi;
    if(a.r/divi!=b.r/divi)return a.r/divi<b.r/divi;
    return a.t<b.t;
}
```

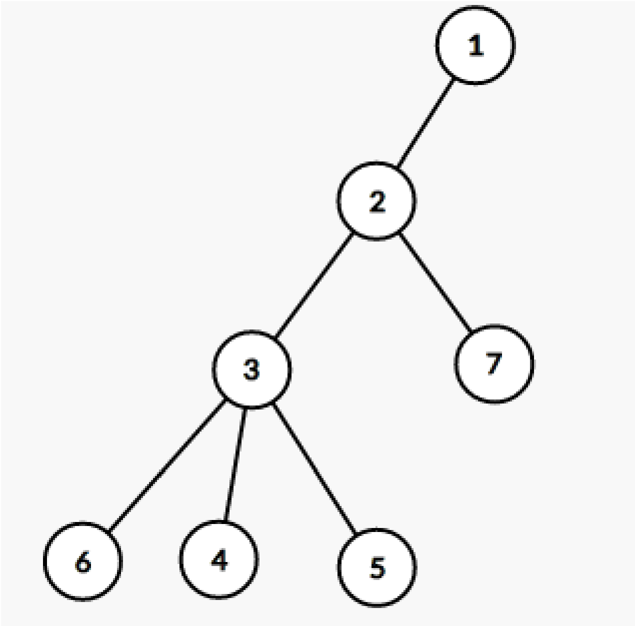
当询问块大小取到 pow (n , 2/3) 时，可达到最优时间复杂度O (pow (n ,5/3)) ，也有块大小取为pow(nt,1/3)的方法。

3:树上莫队

洛谷：SP10707

给一棵树，有点权，求任意两点间路径上不同整数的个数。

利用欧拉序，将树转换成线性结构，然后再对该线性结构使用莫队算法。记录结点出现次数。如果出现过欧数次，那么就不在答案中，如果出现奇数次，那么就在答案中。在线性结构上直接套用普通莫队统计次数的方法进行转移。



欧拉序：1 2 3 4 4 5 5 6 6 3 7 7 2 1

查询x~y，那么就相当于查询从x最后一次出现到y第一次出现的区间内答案，最后再特判一下lca(x,y).

三、树链剖分

考虑这样的问题：

> 对于一棵树进行以下操作：
>
> 操作1： 格式： 1 x y z 表示将树从x到y结点最短路径上所有节点的值都加上z
>
> 操作2： 格式： 2 x y 表示求树从x到y结点最短路径上所有节点的值之和
>
> 操作3： 格式： 3 x z 表示将以x为根节点的子树内所有节点值都加上z
>
> 操作4： 格式： 4 x 表示求以x为根节点的子树内所有节点值之和

将树形转换成线性结构，并用线段树维护。单纯的dfs序无法解决最短路径修改，所以要使用重链剖分。

1.处理节点信息

第一步：标记每个点的深度dep[ ]，父亲节点fa[ ]，子树大小siz[ ],重儿子(儿子中siz最大的)编号son[ ]。

```
void dfs1(int x,int f,int deep){//熟练操作，不做解释
    dep[x]=deep;
    fa[x]=f;
    siz[x]=1;
    int maxson=-1;
    for(Rint i=beg[x];i;i=nex[i]){
        int y=to[i];
        if(y==f)continue;
        dfs1(y,x,deep+1);
        siz[x]+=siz[y];
        if(siz[y]>maxson)son[x]=y,maxson=siz[y];
    }
}
```

2.重新标号，处理链

dfs序重新标号，优先标重儿子，保证连续处理。

记录每条链的top。重儿子的top为father的top，轻儿子的top为那个儿子。

```
void dfs2(int x,int topf){
    id[x]=++cnt;//标记树上节点与链新节点关系（标号）
    wt[cnt]=w[x];//复制权值
    top[x]=topf;//记录top
    if(!son[x])return;//叶子节点检测
    dfs2(son[x],topf);//先处理重边。
    for(int i=beg[x];i;i=nex[i]){//处理轻边
        int y=to[i];
        if(y==fa[x]||y==son[x])continue;
        dfs2(y,y);
    }
}
```

3.查询

路径，累积链的答案。

子树：连续区间的答案。

```
int qRange(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){                        //不在同一链
        if(dep[top[x]]<dep[top[y]])swap(x,y);    //从深的点处理
        res=0;
        query(1,1,n,id[top[x]],id[x]);            //线段树操作
        ans+=res;                                 //累计
        x=fa[top[x]];//                           //端点的父节点
    }
    //
    if(dep[x]>dep[y])swap(x,y);//                 //同一链上结果的求解
    res=0;
    query(1,1,n,id[x],id[y]);//
    ans+=res;
    return ans%mod;
}
int qSon(int x){
    res=0;
    query(1,1,n,id[x],id[x]+siz[x]-1);//子树区间右端点为id[x]+siz[x]-1
    return res;
}
```

4.修改

路径：按链向上跳，的路径

子树：连续区间的子树。

```
inline void updRange(int x,int y,int k){
    k%=mod;
    while(top[x]!=top[y]){
```

```
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        update(1,1,n,id[top[x]],id[x],k);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])swap(x,y);
    update(1,1,n,id[x],id[y],k);
}

inline void updSon(int x,int k){
    update(1,1,n,id[x],id[x]+siz[x]-1,k);
}
```