差分与前缀和

湖南师大附中
ruanxingzhi

区间加

给定一个序列a(初值全为0)。有很多次操作，每个操作形如：

A l r k　将$a_{[l,r]}$每个值加上k.

最后输出整个数组。复杂度要求$O(n)$.允许离线。

区间加

代码：（已经预先指定a[0]=0）

```
void add(int l,int r,int k)
{p[l]+=k,p[r+1]-=k;}

void get_a()
{
    for(int i=1;i<=n;i++)
        a[i]=a[i-1]+p[i];
}
```

区间加

我们很自然地想到：

如果我们知道每一个元素比前一个元素大多少，我们显然可以推出整个序列。

e.g. 已知$a_1 = 2$。a_2比a_1大3，a_3比a_2小4.

那么可以推出：$a_2 = a_1 + 3 = 5, a_3 = a_2 - 4 = 1.$

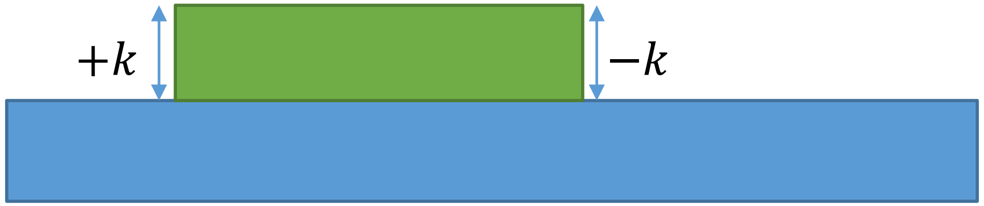
区间加

区间加[l,r]，实际上是发生了这两件事：

a[l]比前一个元素多了k;

a[r+1]比前一个元素少了k.

麻烦自己脑补一下(:



区间加

我们用数组p表示刚刚的差值，p[i]=a[i]-a[i-1].

那么：区间加$[l,r]$，可以化为这两个操作：

p[l]+=k;

p[r+1]-=k;

因此，一次区间加只修改这两个元素；

最后利用p数组求出a数组，即为答案。

差分

$$\Delta a_i = a_i - a_{i-1}$$

差分

在上面的题目中，我们需要维护的数据是“相邻两个数之差”。

它有一个名字：差分。

令 $p_i = a_i - a_{i-1}$ ，我们称p为a的差分数组。

差分

现在，我们拿着p数组就能确定a数组：

a_i相当于$p_{[1,i]}$这个前缀和。

想一想，为什么？

你原本在0楼。往上走了3楼，再往上走了4楼，再往下走了2楼，你一共上了3+4-2=5楼。

因此你目前在5楼。

区间加等差

给定一个序列a(初值全为0)。有很多次操作，操作有两种，强制在线：

- 区间加等差数列

指定$[l,r]$，指定等差数列f的首项和公差。$a_{[l,r]}$每一个元素加上对应的元素：

$a_{[l]} += f_1, a_{[l+1]} += f_2, \cdots$

- 单点查询

指定x，求a_x.

e.g. 目前a={1,2,3,3,3,3}，给[3,5]加上首项为2、公差为1的等差数列。

A变为：{1,2,5,6,7,3}.

区间加等差

显然：a区间加等差，相当于p区间加常数、端点单点修改。

e.g. 目前a={1,2,3,3,3,3}，给[3,5]加上首项为2、公差为1的等差数列。

原来的p数组：{1,1,1,0,0,0}

之后的p数组：{1,1,3,1,1,-4}

A变为：{1,2,5,6,7,3}.

区间加等差

区间加自然是给$p_{[l+1,r]}$加上d；

p_l自然要加上f_1；

那么右端点应该怎么改呢？

显然是要把刚才的操作复原。

我们给p_l加上了f_1，给之后的$(r-l)$个数加了d.

因此：$p_{r+1} \mathrel{-}= f_1 + (r-l)d$.

区间加等差

因此我们要解决的问题就很明显了。

维护p数组，支持单点修改、区间加。

显然可以用线段树解决。

思考题：如果要询问a数组的区间和，怎么做?

提示：思路可以参考
http://www.cnblogs.com/acmsong/p/7225903.html

区间加等差

如果题目还要单点加，应该如何实现？

$p_x \mathrel{+}= k, p_{x+1} \mathrel{-}= k$.

本质和区间加是一致的。

乱七八糟求和

给定序列a，有n次询问。每次询问指定$[l,r]$.
要你回答：

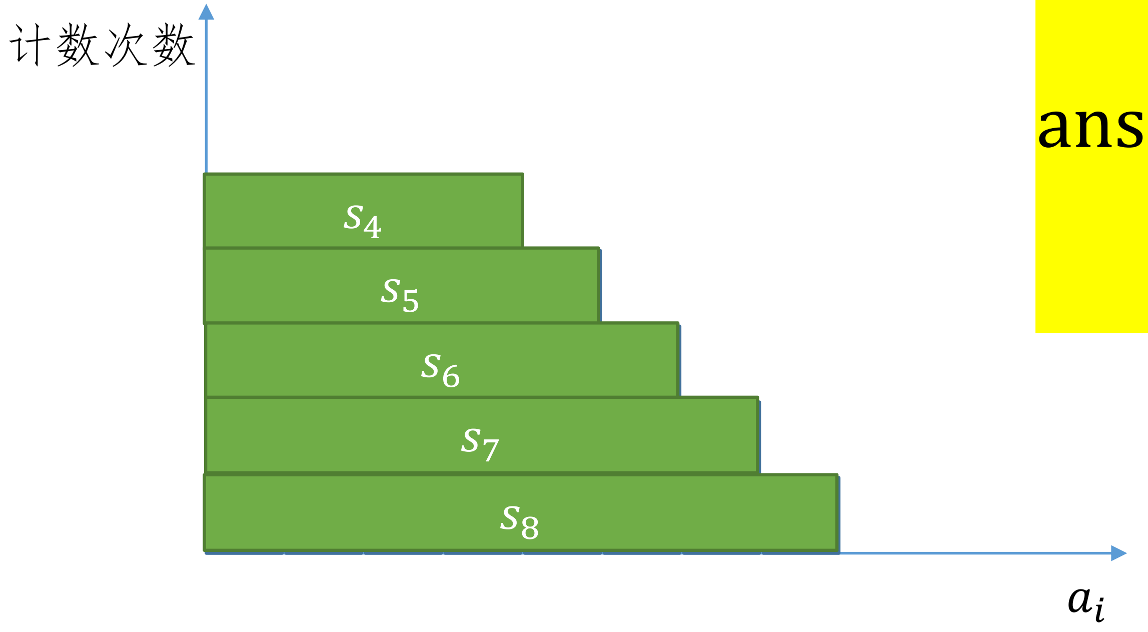
$$\text{ans} = \sum_{i=l}^{r}(r-i+1)a_i$$

e.g. 查询[3,5]，则有：

ans = 3*a[3]+2*a[4]+1*a[5].

乱七八糟求和

我们画个图：



$$
\begin{aligned}
ans &= 5a_4 + 4a_5 + 3a_6 + 2a_7 + a_8 \\
&= 5a_4 + 4a_5 + 3a_6 + 2a_7 + a_8 \\
&\quad +5s_3 - 5s_3 \\
&= s_4 + s_5 + s_6 + s_7 + s_8 - 5s_3
\end{aligned}
$$

乱七八糟求和

求出a的前缀和数组s，接下来的询问就是：

- s数组单点查询
- s数组区间和查询

我们再求出s的前缀和，就能$O(1)$回答s的区间和查询。

综上，本题$O(n)$预处理，$O(1)$每次回答。

区间加二次函数

给定一个序列a(初值全为0)。有很多次操作，操作只有一种：

- 区间加二次函数

指定$[l,r]$，指定一个二次函数$f(x)$，生成数列$\{f(1),f(2)\cdots\}$。

$a_{[l,r]}$每一个元素加上对应的元素。

最后输出a序列。

为了避免炸int，所有运算在模998244353意义下进行。

区间加二次函数

做法：多阶差分。

令$p = \Delta a$,$r = \Delta p$.

那么：a加二次函数，相当于p加等差数列、单点修改；

p加等差数列、单点修改，可以转化为r的单点修改和区间加；

再对r进行一次差分，按照《区间加》那个做法就能解决问题。

每次修改达到$O(1)$.

最后求出r,通过r求出p,再通过p求出a.这件事是$O(n)$的。

三个思想

做了这么一些题，我们大致可以归纳出三个思想：

1. 差分降次，前缀和升次。关于常数的操作可以方便地完成。
2. 知道p数组，可以$O(n)$求出a数组。
3. 知道s数组，可以$O(1)$求出a数组的区间和。

智障题

今有$n \times m$的网格，初始状态是空的。有k次操作。
每次操作指定一个矩形（给出四个顶点），铺上地毯。
最后询问有多少个点没有被铺地毯。

二维差分，二维前缀和，$O(nm+nk)$.
差分：左上角加，右上角减，左下角减，右下角加.
最后$p[x][y]$的二维前缀和即为$a[x][y]$.

思考题

《区间加二次函数》改成区间加k次多项式。不一定要用差分啊。

将函数$f(x)$平移成$f(x-l)$，这样的话，a_i增加的值只与i有关，而与首项无关。这一步耗时$O(k^2)$.

将a_x看作一个分段函数，每个点都有对应的一组多项式的系数。

那么每次区间加就是给$a_{[l,r]}$加上一组系数。

显然这又是区间加常数了，用《区间加》的做法就能了事。

思考题

上面的做法瓶颈在于函数平移。

用FFT来加速平移，每次操作可以达到$O(k\log k)$.

至此问题解决。每次操作$O(k\log k)$,最后$O(kn+kq)$统计结果。

代码细节：拿结构体来存系数，重载一下加号。

然后就能直接套《区间加》的模板啦。

思考题

这题不要你最后输出a数组了。要你支持在线查询a_i.

上面我们是用差分实现区间加系数的。

那现在就用线段树来加咯。

$O(k\log k+k\log n)$每次区间加。

$O(k\log n)$每次查询。

思考题

现在n取10^{18}.
操作仍然只有区间加多项式。
最后指定一些点，查询a_i的值.

扫描线即可。复杂度与n无关了。

致谢：riteme [https://riteme.github.io]

树上差分

$a, b, \mathrm{LCA}(a, b)$.

树上差分

有一棵树，点有点权。有m个操作，每个操作是：

- 链加　指定u,v,k，u到v的路径上每个点点权加k.

最后输出每个点的权值。

树上差分

$u \to v$的链加，可以拆成两部分：$u \to \mathrm{LCA}, v \to \mathrm{LCA}$.

$u \to \mathrm{LCA}, v \to \mathrm{LCA}$各加上$k$，然后LCA减去$k$（因为被加了两次）。

也就是说，我们需要实现：点到祖先的链加、单点修改。

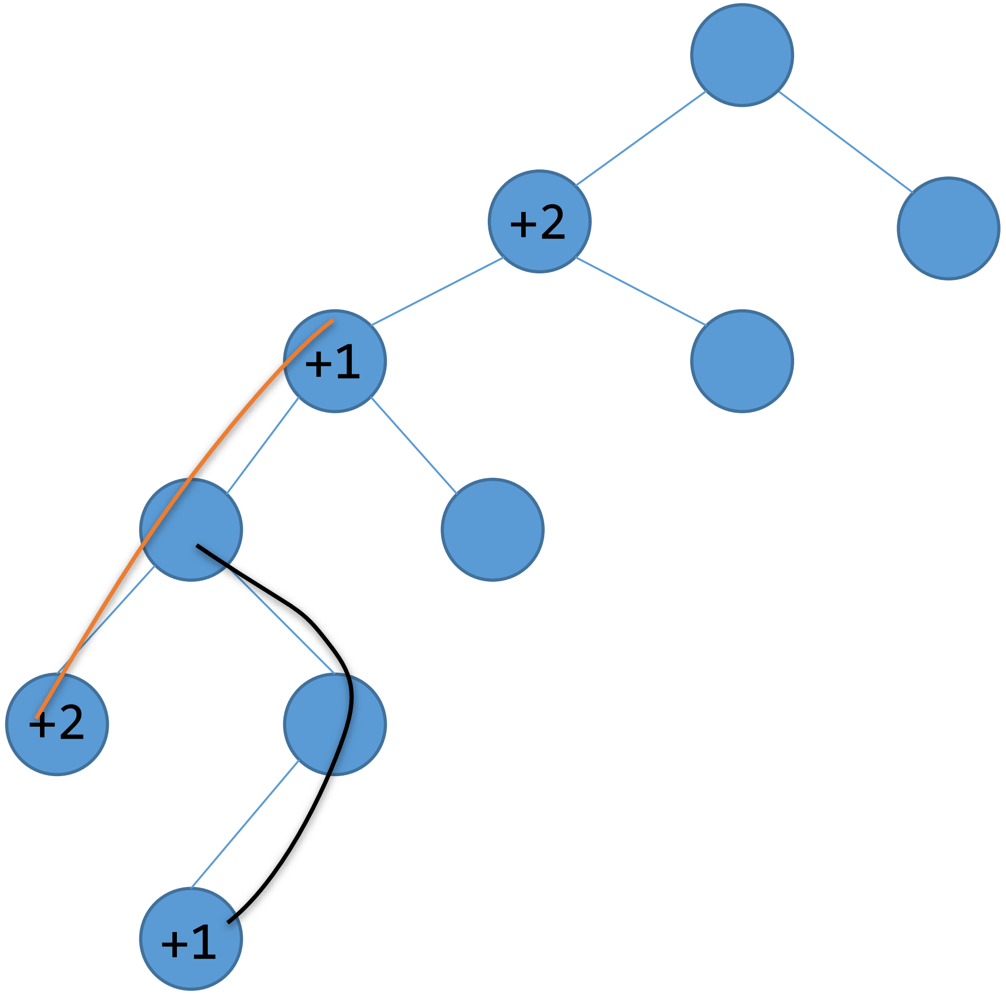
树上差分

如何实现点到祖先的链加呢？

思路和序列差分基本一致。

$u \to p$链加k:

```
flag[u]+=k;
flag[dad[p]]-=k;
```



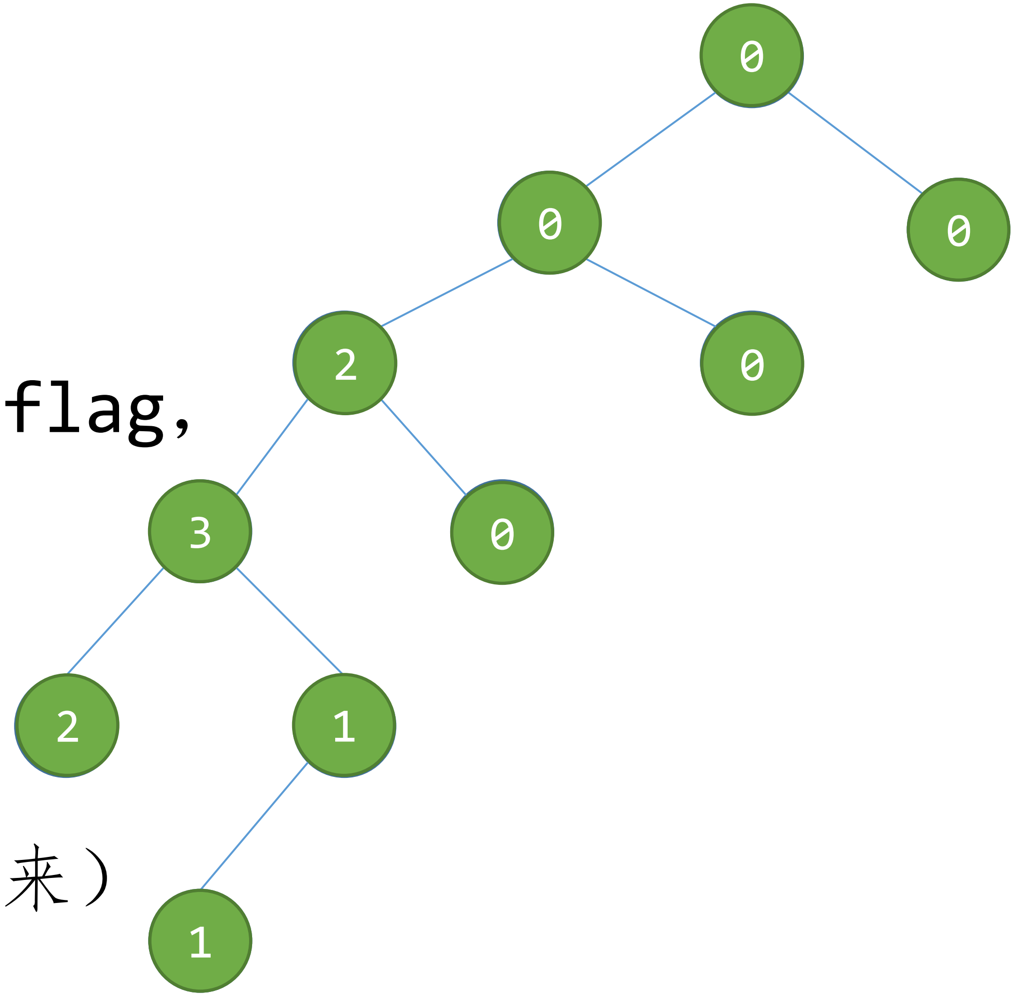
树上差分

最后对于每个点，干下面两件事：

- 处理所有子节点；
- 把所有子节点的value加上自己的flag，
    作为自己的value.

一遍DFS即可做到。

（也可以倒着拓扑排序，从叶子推上来）



链求和

给定一棵n个点的树，点有点权。有m次询问。

每次询问：求树上两点之间路径上的点权和。

链求和

我们显然可以把$u \to v$拆成把$u \to \text{LCA}, v \to \text{LCA}$.

即：$\text{ans} = d[u][\text{LCA}] + d[v][\text{LCA}] - d[\text{LCA}]$.

现在的任务是：求出一个点到自己某个祖先的点权和。

我们很自然地想到前缀和。

前缀的定义修改成：点x到根节点经过的所有点。前缀和记为s.

显然：$d[u][\text{anc}] = s[u] - s\big[\text{dad}[\text{anc}]\big]$.

链求和

$$\begin{aligned} \text{ans} &= d[u][\text{LCA}] + d[v][\text{LCA}] - d[\text{LCA}]. \\ &= s[u] - s\big[\text{dad}[\text{LCA}]\big] + s[v] - s\big[\text{dad}[\text{LCA}]\big] - d[\text{LCA}]. \\ &= s[u] + s[v] - 2s\big[\text{dad}[\text{LCA}]\big] - d[\text{LCA}]. \end{aligned}$$

我们懒得维护每个点的点权了，式子可以改写成：

$$\text{ans} = s[u] + s[v] - s[\text{LCA}] - s\big[\text{dad}[\text{LCA}]\big].$$

只需要知道s数组，就能O(1)回答$u \to v$的路径点权和。

COUNT ON A TREE

给定一棵n个点的树，点有点权。有m次询问。

每次询问：求树上两点之间路径上第k大的点权。

强制在线。

（这种题如果不强制在线，会被树上莫队之类的万能算法水过……）

[BZOJ题号：2588]

COUNT ON A TREE

如果这棵树是一条链，那显然是可持久化线段树模板题。
定义权值数组：w_i记录i出现了多少次。
那么我们显然是在维护数组的每个前缀的权值数组。

这也是一种前缀和与差分的思想。i在$[l,r]$中的出现次数，就是$w[r]_i - w[l-1]_i$.
$[l,r]$这个区间的权值数组，就是$w[r] - w[l-1]$.用线段树实现数组即可保证复杂度。

COUNT ON A TREE

既然链状的情况可以这样干，显然树上的也可以。

仍然把$u \to v$拆成$u \to \mathrm{LCA}$, $v \to \mathrm{LCA}$.

显然$w[u, \mathrm{LCA}] = w[u] - w[\mathrm{dad}[\mathrm{LCA}]]$. 与《链求和》同理，最后的式子是：

$$w[u, v] = w[u] + w[v] - w[\mathrm{LCA}] - w[\mathrm{dad}[\mathrm{LCA}]]$$

COUNT ON A TREE

子节点的前缀显然是从父亲那里继承过来的。
因此子节点的w由父亲节点的w派生出来。

用可持久化线段树实现就行了。

差分的数学应用

差分 <-> 求导

前缀和 <-> 定积分

数学上的差分

数学上的差分，定义为：$\Delta f(x)=f(x+1)-f(x)$.

为什么我们写代码的时候定义为a_i-a_{i-1}?

因为，这样我们就用p_1记录了a_1，从而可以唯一确定a.

差分的性质

差分有一些有趣的性质。比如，请看$a_n = n^3$的多阶差分：

```
(0, 1, 8, 27, 64, 125)        //a
-> (1, 7, 19, 37, 61)         //一阶差分
-> (6, 12, 18, 24)            //二阶差分
-> (6, 6, 6)                  //三阶差分
-> (0, 0)
```

差分的性质

为什么差分有这一项性质？可以手动玩一下。

$x^3-(x-1)^3=3x^2-3x+1$ ，变成二次多项式了。

这个性质可以用来找规律。

差分的性质

如果你感觉答案是关于n的多项式，你可以试试这个办法：

取数列的前几项，不断地进行差分，如果发现k阶差分所有项都是同一常数，那么这个多项式就是k次的。

然后手动高斯消元一发，就能找到$f(x)$.

有限微积分

在之前的《区间加二次函数》一题中，我们手动推出了p_{r+1}需要如何修改、p所加的等差数列的通项公式。

然而实际上，这样做效率是很低的。

差分可以类比求导，区间和可以类比定积分。

而且，区间和也与定积分一样，有类似的基本定理。

有限微积分

时间有限，无法细讲。请自行去了解相关知识。

关键词：

下降阶乘幂、有限微积分、第二类斯特林数。

资料：

- 我的博客 ruanx.pw/post/有限微积分.html [入门]
- 《具体数学》 [尤其安利这本书，讲了很多有用的知识]

致谢

- 感谢各位看完了这份课件。
- 感谢riteme教我区间加多项式。
- 感谢stdcall、HJWJBSR审查了这份课件。

END

反馈与建议：请联系

ruanxingzhi@gmail.com