又是一道线段树模板题。不过本题有点特殊。区间更改要取模，而取模运算是不满足交换律和结合律的，这意味着要修改到叶子节点，单次时间复杂度为O(n)，本体1e5的数据量恐怕不能过去(除非数据过水)，所以考虑优化。

先看几组式子。

19 mod 5 = 4 19 mod 5 mod 6 = 1 19 mod 5 mod 5 = 4 19 mod 5 mod 3=1

通过数学归纳法，我们可以发现以下规律：

a mod b mod c = a mod b 当且仅当b<=c 时成立

因此，我们维护一个变量lz(初始值为Inf)，表示当前访问区间历史取mod的最小值，例如5次取模的值分别为：5 4 32 252 545 ，最后lz的值为4。当区间取mod的值(设为k)大于lz的值时，该区间就不需要更新了，直接递归返回(剪枝)。具体实现嘛，在递归进入时额外判断一下就行了，并不会增加时间复杂度，相反，均摊情况下这种优化能去掉大多数无用的更新操作，从而节省时间(奈何数据不行，优化效果不好)。

还能更优吗？

维护一个变量mx，表示当前访问区间的最大值，当k(如上)的值大于mx时，该区间也不需要更新，递归返回，再次剪枝。但是一重优化就够了，双重优化不一定更快(两种优化效果都不错)。

区间更新update\_range代码如下：

void

update\_range(int p,int L,int R,int k)

{

if(mx(p)<k||lz(p)<=k) return ;*//剪枝优化*

if(l(p)>=L&&r(p)<=R&&lz(p)>k) lz(p)=k;*//这一步很重要，lz在这里更新*

if(l(p)==r(p)) return sum(p)=mx(p)=mx(p)%k,void();*//到达叶子节点，返回*

int mid=l(p)+r(p)>>1;*//由于没有区间整体更新，所以没有push\_down函数* if(L<=mid)update\_range(lc(p),L,R,k);*//左区间* if(R>mid)update\_range(rc(p),L,R,k);*//右区间*

push\_up(p);*//线段树修改必备*

}

然后就是单点更新，比较简单，但要注意维护lz的值，单点修改成k时，访问路径上的区间的lz要更新为max(lz,k+1)，这是一个坑点，因为在更新后lz的值不一定还有效(不好说清，自行理解)，至于为什么lz要更新为max(lz,k+1)嘛，可以理解成先将lz变回初始值Inf，再对k+1取模，然而这次取模是不会更新区间上的值的(自行理解)。于是

void

update\_node(int p,int x,int k)

{

lz(p)=k>lz(p)?k:lz(p);*//重点*

if(l(p)==r(p)) return sum(p)=mx(p)=k,void();*//到达目标*

int mid=l(p)+r(p)>>1;*//折中*

if(x<=mid) update\_node(lc(p),x,k);*//左区间*

else update\_node(rc(p),x,k);*//右区间*

push\_up(p);*//维护*

}

其余的都简单了，完整代码如下：

#include<cstdio>

#include<algorithm>

#include<cstring>

using namespace std;

const int N=1e5+10,Inf=1e8;

typedef long long LL;

struct

SegTree

{

int l,r;

LL sum;

int mx;

int lz;

#define l(p) tree[p].l

#define r(p) tree[p].r

#define sum(p) tree[p].sum

#define mx(p) tree[p].mx

#define lz(p) tree[p].lz

#define lc(p) (p<<1)

#define rc(p) (p<<1|1)

}tree[N<<2];

int a[N+1];

int n,m;

void

push\_up(int p)

{

sum(p)=sum(lc(p))+sum(rc(p));

mx(p)=max(mx(lc(p)),mx(rc(p)));

}

void

build(int p,int L,int R)

{

l(p)=L;r(p)=R;lz(p)=Inf;

if(L==R) return mx(p)=sum(p)=a[L],void();

int mid=L+R>>1;

build(lc(p),L,mid);

build(rc(p),mid+1,R);

push\_up(p);

}

void

update\_node(int p,int x,int k)

{

lz(p)=k+1>lz(p)?k+1:lz(p);*//重点*

if(l(p)==r(p)) return sum(p)=mx(p)=k,void();*//到达目标*

int mid=l(p)+r(p)>>1;*//折中*

if(x<=mid) update\_node(lc(p),x,k);*//左区间*

else update\_node(rc(p),x,k);*//右区间*

push\_up(p);*//维护*

}

void

update\_range(int p,int L,int R,int k)

{

if(mx(p)<k||lz(p)<=k) return ;*//剪枝优化*

if(l(p)>=L&&r(p)<=R&&lz(p)>k) lz(p)=k;*//这一步很重要，lz在这里更新*

if(l(p)==r(p)) return sum(p)=mx(p)=mx(p)%k,void();*//到达叶子节点，返回*

int mid=l(p)+r(p)>>1;*//由于没有区间整体更新，所以没有push\_down函数* if(L<=mid)update\_range(lc(p),L,R,k);*//左区间* if(R>mid)update\_range(rc(p),L,R,k);*//右区间*

push\_up(p);*//线段树修改必备*

}

LL

query(int p,int L,int R)

{

if(l(p)>R||r(p)<L) return 0;

if(l(p)>=L&&r(p)<=R) return sum(p);

return query(lc(p),L,R)+query(rc(p),L,R);

}

int

Read()

{

int x=0;char ch=getchar();

while(ch>'9'||ch<'0') ch=getchar();

while(ch>='0'&&ch<='9') x=(x<<3)+(x<<1)+ch-'0',ch=getchar();

return x;

}

int

main()

{

freopen("review.in","r",stdin);

freopen("review.out","w",stdout);

n=Read();

for(int i=1;i<=n;i++) a[i]=Read();

build(1,1,n);

m=Read();

int op,l,r,k;

for(int i=1;i<=m;i++)

{

op=Read();

if(op==1)

{

l=Read();r=Read();k=Read();

*//scanf("%d%d%d",&l,&r,&k);*

update\_range(1,l,r,k);

}else if(op==2)

{

l=Read();k=Read();

*//scanf("%d%d",&l,&k);*

update\_node(1,l,k);

}else

{

l=Read();r=Read();

*//scanf("%d%d",&l,&r);*

printf("%lld\n",query(1,l,r));

}

}

return 0;

}

PS:可以合并lz和mx，不过没必要