B题解题报告：线段树的基本应用

题意：区间加法，区间乘法，区间求和~~(线段树三个字已经快贴在脸上了)~~

很明显就是用线段树了。线段树进行区间加法、区间求和是最基本的操作。
首先我们需要构建一个线段树，一个完全二叉树，用于存储区间信息的数据结构，每个节点代表一个区间。
采用递归的方法构建。
利用一个结构体存线段树各个节点的信息，包括区间大小，区间和，还有懒惰标记。

```
void build(int l,int r,int tot){
 node[tot].r=r,node[tot].l=l;
 //node[tot].lazy_mul=1;
 if(r==l) {//找到了叶子节点，直接把原数组的值存进叶子节点
  node[tot].sum=arr[r]%mod;
  return;
 }
 int m=(r+l)/2;
 build(l,m,tot<<1);
 build(m+1,r,(tot<<1)+1);
 node[tot].sum=(node[tot<<1].sum+node[(tot<<1)+1].sum)%mod;
 //更新区间和
}
```

线段树有个广为人之的结论就是对于任意一个节点编号为tot，它的代表左半区间的子节点的编号是tot2,*代表右半区间的子节点的编号是*2tot+1。
利用这个性质可以准确的查找到各个节点的两个子节点。
构建好一颗线段树后，还要想着怎么使用它。
如果是区间加法的话，肯定不能通过线段树把每个叶子节点都找到然后一个一个更新。
这样的话还不如不构建线段树，复杂度反而上升了。
为了解决这个问题，线段树的一个精髓所在，懒惰标记便出现了。

```
struct Node{
 int r,l;
 ll sum;
 ll lazy_add;
 ll lazy_mul;
 Node(){
  sum=0,lazy_add=0,lazy_mul=1;
 }
}node[maxn<<2];
```

我们存储的节点中除了区间和和的信息外，还多了两个懒惰标记，分别用来标记加法和乘法。
懒惰标记是什么呢？
说得形象点，有了懒惰标记我们就不用打破沙锅问到底了。
举个例子，假如我们要给区间[2,9]都加上1，有一个非叶子节点维护的区间大小是[2,5]，如果我们找到了这个区间，
那么只要把这个节点的懒惰标记**lazy_add**加上1，表明这个区间内所有数都要加1，然后就可以不继续向下找了。
当我们要访问[2,5]区间以内的某个区间或数时，就可以把这个标记下传给他的两个子节点,
直到遇到了叶节点，直接修改叶节点的值，回溯的时候更新区间和就好了。

> 总的来说，懒惰标记帮我们省去了很多不必要的操作，因而实现了复杂度的降低。

如果仅仅只有加法的懒惰标记的话，这个题目几乎没有难度。但是有了乘法的懒惰标记，我们就要考虑如何让两种标记和谐共存。
假如一个区间已经有了大小为k的加法懒惰标记，此时他的父节点下传了大小为m的懒惰标记，我们应该怎么维护这个区间的信息？
其实就是个乘法分配律。

> (a+b)×c=a×c+b×c; 基于乘法分配律，下传时，这样更新子节点：

```
\\乘法懒惰标记相乘，加法懒惰标记相加；
\\区间和=（父节点加法懒惰标记*子节点区间长度+父节点乘法懒惰标记*（父节点加法懒惰标记+子节点原区间和）
\\每次更新乘法懒惰标记的时候原加法懒惰标记同时也要乘以下传的值。
```

> 时间复杂度O(nlogn):
> 区间查询和修改都是logn，区间乘法不过是多了些操作，对时间复杂度的贡献并没有本质上的提升。