分块

树状数组和线段树虽然非常方便，但维护的信息必须满足信息合并特性（如区间可加、可减），若不满足此特性，则不可以使用树状数组和线段树。

分块算法可以维护一些线段树维护不了的内容，它其实就是**优化过后的暴力算法**。

分块可以解决几乎所有区间更新和区间查询问题，但效率相对于线段树等数据结构要差一些。

分块算法是将所有数据都分为若干块，维护块内信息，使得块内查询为 O(1)时间，而总询问可被看作若干块询问的的总和。

分块算法将长度为 n 的序列分成若干块，每一块都有 k 个元素，最后一块可能少于 k 个元素。为了使时间复杂度均摊，通常将块的大小设为 k= $\sqrt{n}$ ，用 pos[i]表示第 i 个位置所属的块，对每个块都进行信息维护。

分块可以解决以下问题：

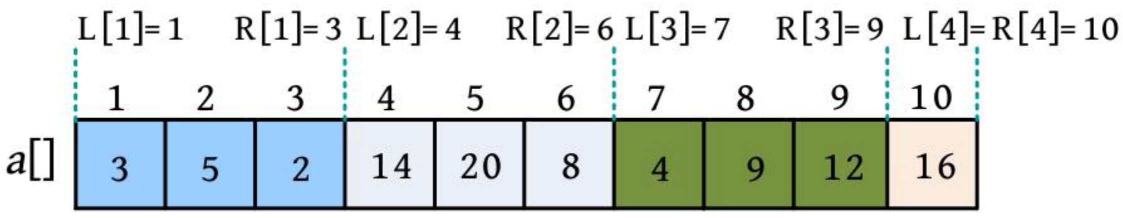
- **单点更新：**一般先将对应块的懒标记下传，再暴力更新块的状态，时间复杂度为 O($\sqrt{n}$)。
- **区间更新：**若区间更新横跨若干块，则只需对完全覆盖的块打上懒标记，最多需要修改两端的两个块，对两端剩余的部分暴力更新块的状态。每次更新都最多遍历个块，遍历每个块的时间复杂度都是 O(1)，两端的两个块暴力更新 $\sqrt{n}$ 次，总的时间复杂度是 O($\sqrt{n}$)。
- **区间查询：**和区间更新类似，对中间跨过的整个块直接利用块存储的信息统计答案，对两端剩余的部分可以暴力扫描统计。时间复杂度和区间修改一样，也是 O($\sqrt{n}$)。

将整个段分成多个块后进行修改或查询时，对完全覆盖的块直接进行修改，像线段树一样标记或累加；对两端剩余的部分进行暴力修改。

分块算法遵循“**大段维护、局部朴素**”的原则。

1. 预处理

（1）将序列分块，然后将每个块都标记左右端点 L[i]和 R[i]，对最后一块需要特别处理。n=10，t= $\sqrt{n}$ =3，每 3 个元素为一块，一共分为 4 块，最后一块只有一个元素。



算法代码：

```
t=sqrt(n*1.0);//float sqrt (float),double sqrt (double),double long sqrt(double long)
int num=n/t;
if(n%t)  num++;
for(int i=1;i<=num;i++){
     L[i]=(i-1)*t+1;//每一块的左右端点
     R[i]=i*t;
}
R[num]=n;
```

（2）用 pos[]标记每个元素所属的块，用 sum[]累加每一块的和值。

	1	2	3	4	5	6	7	8	9	10
a[]	3	5	2	14	20	8	4	9	12	16
pos[]	1	1	1	2	2	2	3	3	3	4
sum[]	10			42			25			16

算法代码：

```
for(int i=1;i<=num;i++)
    for(int j=L[i];j<=R[i];j++){
        pos[j]=i;//表示属于哪个块
        sum[i]+=a[j];//计算每块的和值
    }
```

2. 区间更新

区间更新，例如将[l, r]区间的元素都加上 d。

（1）求 l 和 r 所属的块，p=pos[l]，q=pos[r]。

（2）若属于同一块（p=q），则对该区间的元素进行暴力修改，同时更新该块的和值。

（3）若不属于同一块，则对中间完全覆盖的块打上懒标记，add[i]+=d，对首尾两端的元素进行暴力修改。

例如，将[3, 8]区间的元素都加上 5，操作过程：

① 读取 3 和 8 所属的块 p=pos[3]=1，q=pos[8]=3，不属于同一块，中间的完整块[p+1, q-1]为第 2 块，为该块打上懒标记 add[2]+=5；

② 对首尾两端的元素（下标 3、7、8）进行暴力修改，并修改和值。

add[2]+=5

	1	2	3	4	5	6	7	8	9	10
a[]	3	5	7	14	20	8	9	14	12	16
pos[]	1	1	1	2	2	2	3	3	3	4
sum[]	15			42			35			16

算法代码：

```
void change(int l,int r,long long d){//[l,r]区间的元素加 d
    int p=pos[l],q=pos[r];//读取所属的块
    if(p==q) {//在同一块中
        for(int i=l;i<=r;i++)//暴力修改
            a[i]+=d;
        sum[p]+=d*(r-l+1);//修改和值
    }
    else{
        for(int i=p+1;i<=q-1;i++)//对中间完全覆盖的块打懒标记
            add[i]+=d;
        for(int i=l;i<=R[p];i++)//左端暴力修改
            a[i]+=d;
        sum[p]+=d*(R[p]-l+1); //修改和值
        for(int i=L[q];i<=r;i++)//右端暴力修改
            a[i]+=d;
        sum[q]+=d*(r-L[q]+1); //修改和值
    }
}
```

3. 区间查询

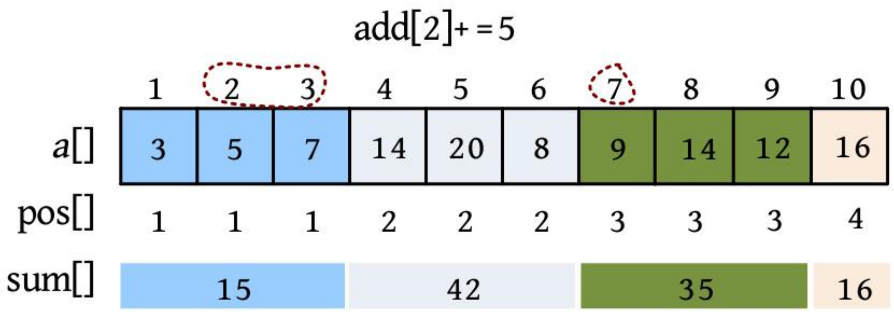
区间查询，例如查询[l, r]区间的元素和值。

（1）求 l 和 r 的所属块，p=pos[l]，q=pos[r]。

（2）若属于同一块（p=q），则对该区间的元素进行暴力累加，然后加上懒标记上的值。

（3）若不属于同一块，则对中间完全覆盖的块累加 sum[]值和懒标记上的值，然后对首尾两端暴力累加元素值及懒标记值。

例如，查询[2, 7]区间的元素和值，操作过程：①读 p=pos[2]=1，q=pos[7]=3，不属于同一块，则中间的完整块[p+1, q-1]为第 2 块，ans+=sum[2]+add[2]×(R[2]-L[2]+1)=42+5×3=57；②对首尾两端的元素暴力累加元素值及懒标记值。此时懒标记 add[1]=add[3]=0，ans+=5+7+add[1]×(3-2+1)+9+add[3]×(7-7+1)=78。



算法代码：

```
ll ask(int l,int r){//区间查询
    int p=pos[l],q=pos[r];
    ll ans=0;
    if(p==q) {//在同一块中
        for(int i=l;i<=r;i++)//累加
            ans+=a[i];
        ans+=add[p]*(r-l+1);//计算懒标记
    }
    else{
        for(int i=p+1;i<=q-1;i++)//累加中间段落
            ans+=sum[i]+add[i]*(R[i]-L[i]+1);
        for(int i=l;i<=R[p];i++)//左端暴力累加
            ans+=a[i];
        ans+=add[p]*(R[p]-l+1);
        for(int i=L[q];i<=r;i++)//右端暴力累加
            ans+=a[i];
        ans+=add[q]*(r-L[q]+1);
    }
    return ans;
}
```