POJ3648

题目描述（POJ3648）：有 N 个整数 $A_1, A_2, \ldots, A_N$，需要对其进行两种操作，一种操作是对给定区间中的每个数都添加一个给定的数，另一种操作是查询给定区间中数的总和。

输入：第 1 行包含两个数 N 和 Q（$1 \le N, Q \le 10^5$）；第 2 行包含 N 个数，为 $A_1, A_2, \ldots, A_N$ 的初始值（$-10^9 \le A_i \le 10^9$）；接下来的 Q 行，每行都表示一种操作，“C a b c”表示将 $A_a, A_{a+1}, \ldots, A_b$ 中的每一个数都加 c（$-10^4 \le c \le 10^4$），“Q a b”表示查询 $A_a, A_{a+1}, \ldots, A_b$ 的总和。

输出：对每个查询，都单行输出区间和的值。

输入样例	输出样例
10 5	4
1 2 3 4 5 6 7 8 9 10	55
Q 4 4	9
Q 1 10	15
Q 2 4	
C 3 6 3	
Q 2 4	

提示：总和可能超过 32 位整数的范围。

题解：本题有两种操作：区间更新和区间查询，可采用用分块算法解决。

算法设计如下：

（1）分块预处理。将序列分块，然后对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；标记每个元素所属的块，累加每一块的和值。

（2）区间更新。首先取 l 和 r 所属的块，p=pos[l]，q=pos[r]；若属于同一块，则对该区间的所有元素都进行暴力修改，同时更新该块的和值。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；对首尾两端的元素暴力修改即可。

（3）区间查询。首先取 l 和 r 所属的块，p=pos[l]，q=pos[r]；若属于同一块，则对该区间的所有元素都进行暴力累加，然后加上懒标记上的值。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，然后对首尾两端的元素暴力累加元素值及懒标记值。

POJ1019

题目描述（POJ1019）：给出单个正整数 i，编写程序以找到位于数字组 $S_1, S_2, \ldots, S_k$ 序列中第 i 位上的数字。每个组 S_k 都由一系列正整数组成，范围为 1～k，一个接一个地写入。序列的前 80 位数字如下：

11212312341234512345612345671234567812345678912345678910123456789101112345678910。

输入：第 1 行包含一个整数 t（1≤t≤10），表示测试用例的数量。每个测试用例后都跟一行，包含单个整数 i（1≤i≤2, 147, 483, 647）。

输出：对每个测试用例，都单行输出第 i 位上的数字。

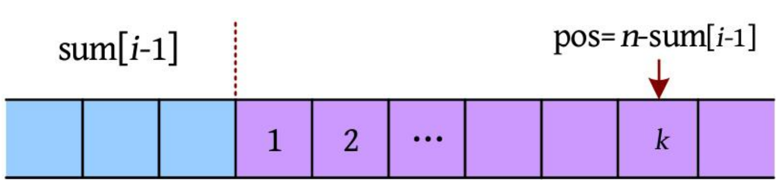
题解：在测试用例中，序列的第 8 位和第 3 位都是 2：

112123123412345123456123456712345678123456789123456789101234567891011…

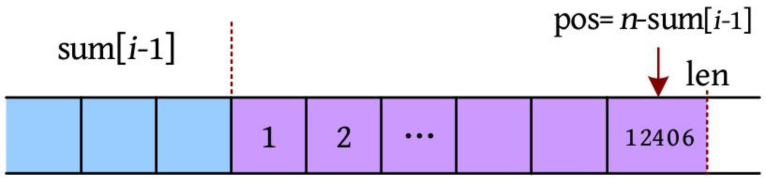
将每个组都看作一个分块，每个组（分块）的长度都为 a[i]：当组内的每个数都由一位数字组成时，当前组的长度等于前一组的长度+1；当组内出现两位数 10～99 时，当前组的长度等于前一组的长度+2，以此类推。

- 1 12 123 1234 12345 123456 1234567 12345678 123456789　前一组的长度+1
- 12345678910 1234567891011 123456789101112……　前一组的长度+2a[i]为第 i 块的长度，sum[i]为前 i（包括 i）块的总长度。

例如，查询第 n 位上的数字，首先定位到第 i 块，然后在当前块内查找具体的数 k。



k 可能是多位数，例如 k=12406，如下图所示。



第 pos 位的数字应为 $k/10^{len-pos}=124$，124%10=4。

1. 算法设计

（1）计算每一块的长度 a[i]及前 i 块的总长度 sum[i]。

（2）定位到第 i 块，在块内查找第 pos 位所在的数 k。

（3）数 k 有可能是多位数，第 pos 位为 k/(int)pow(10.0, len-pos)%10。

2. 算法实现

```
LL a[maxn],sum[maxn];//a[i]为第 i 组（分块）的长度，sum[i]为前 i（包括 i）组的总长度
int main(){
    int i,j;
    sum[0]=a[0]=0;
    for(i=1;i<maxn;i++){
        a[i]=a[i-1]+(int)log10((double)i)+1;
        sum[i]=sum[i-1]+a[i];
    }
    int t,n;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        i=0;
        while(sum[i]<n) i++; //确定 n 在第 i 块
        int pos=n-sum[i-1];  //确定 n 在第 i 块的第 pos 个位置
        int len=0,k=0;
        while(len<pos){
            k++;
            len+=(int)log10((double)k)+1;
        }
        printf("%d\n", k/(int)pow(10.0,len-pos)%10);
    }
    return 0 ;
}
```

POJ3264

题目描述（POJ3264）：每天挤奶时，约翰的 N 头奶牛（1≤N≤50,000）都以相同的顺序排队。他挑选一系列连续的奶牛来玩游戏。为了让所有奶牛都玩得开心，它们的高度差异不应太大。约翰列出了 Q 组（1≤Q≤200,000）奶牛和它们的高度（1≤height≤1,000,000）。他希望确定每个小组中最高和最矮的奶牛之间的高度差异。

输入：第 1 行包含两个整数 N 和 Q。接下来 N 行，每行都包含一个整数，表示奶牛的高度。最后 Q 行，每行都包含两个整数 A 和 B（1≤A≤B≤N），代表从 A 到 B 的奶牛范围。

输出：输出 Q 行，每行都包含一个整数，表示该范围内最高和最矮奶牛的高度差。

输入样例	输出样例
6 3 1 7 3 4 2 5 1 5 4 6 2 2	6 3 0

题解：本题是典型的区间最值查询问题，可采用线段树、ST 或分块解决。

1. 算法设计

（1）分块。划分块，记录每个元素所属的块，以及每一块的左右端点下标、最大值和最小值。

（2）查询。查询[l, r]区间最大值和最小值的差值。

• 若该区间属于同一块，则暴力统计最大值和最小值，返回两者的差值。

• 若该区间包含多个块，则统计中间每个块的最大值和最小值，然后暴力统计左端点和右端点的最大值和最小值，返回两者的差值。

2. 算法实现

```
void build(){//分块预处理
    int t=sqrt(n*1.0);
    int num=n/t;
    if(n%num) num++;
    for(int i=1;i<=num;i++)
        L[i]=(i-1)*t+1,R[i]=i*t;
    R[num]=n;
    for(int i=1;i<=n;i++)
        belong[i]=(i-1)/t+1;
    for(int i=1;i<=num;i++){//求每一块的最值
        int MIN=inf,MAX=-inf;
        for(int j=L[i];j<=R[i];j++){
            MAX=max(MAX,a[j]);
            MIN=min(MIN,a[j]);
        }
        block_max[i]=MAX;
        block_min[i]=MIN;
    }
}
```

```
int query(int l,int r){//查询区间最值差
    int MIN=inf,MAX=-inf;
    if(belong[l]==belong[r]){
        for(int i=l;i<=r;i++){
            MAX=max(MAX,a[i]);
            MIN=min(MIN,a[i]);
        }
        return MAX-MIN;
    }
    else{
        for(int i=l;i<=R[belong[l]];i++){//左端点
            MAX=max(MAX,a[i]);
            MIN=min(MIN,a[i]);
        }
        for(int i=belong[l]+1;i<belong[r];i++){//中间
            MAX=max(MAX,block_max[i]);
            MIN=min(MIN,block_min[i]);
        }
        for(int i=L[belong[r]];i<=r;i++){//右端点
            MAX=max(MAX,a[i]);
            MIN=min(MIN,a[i]);
        }
    }
    return MAX-MIN;
}
```

HDU4417

题目描述（HDU4417）：可怜的公主陷入困境，马里奥需要拯救他的情人。把通往城堡的道路视为一条线（长度为 n），在每个整数点 i 上都有一块高度为 h_i 的砖，马里奥可以跳的最大高度是 H，求他在[L, R]区间可以跳过多少砖块。

输入：第 1 行是整数 T，表示测试用例的数量。每个测试用例的第 1 行都包含两个整数 n、m（1≤n，m≤10^5），n 是道路的长度，m 是查询的数量。下一行包含 n 个整数，表示每个砖的高度（范围是[0, 10^9]）。接下来的 m 行，每行都包含三个整数 L、R、H（0≤L≤R<n，0≤H≤10^9）。

输出：对每种情况都输出“Case X：”（X 是从 1 开始的案例编号），后跟 m 行，每行都包含一个整数。第 i 个整数是第 i 个查询中马里奥跳过的砖块数。

输入样例	输出样例
1	Case 1:
10 10	4
0 5 2 7 5 4 3 8 7 7	0
2 8 6	0
3 5 0	3
1 3 1	1
1 9 4	2
0 1 0	0
3 5 5	1
5 5 1	5
4 6 3	1
1 5 7	
5 7 3	

题解：本题为区间查询问题，查询[l, r]区间小于或等于 h 的元素个数，可以采用分块的方法解决。

1. 算法设计

（1）分块。划分块并对每一块进行非递减排序。在辅助数组 temp[ ]上排序，原数组不变。

（2）查询。查询[l, r]区间小于或等于 h 的元素个数。

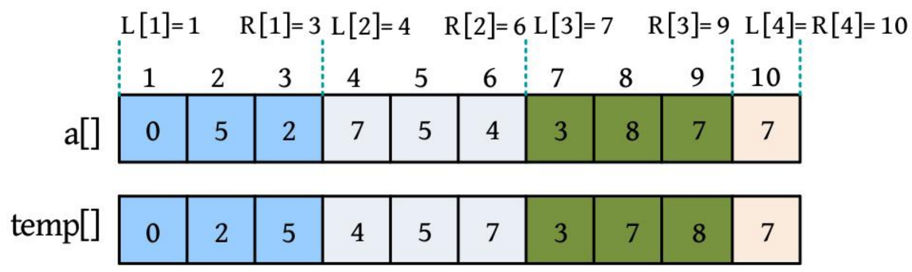
• 若该区间属于同一块，则暴力累加块内小于或等于 h 的元素个数。

• 若该区间包含多个块，则累加中间每一块小于或等于 h 的元素个数，此时可以用 upper_bound()函数统计，然后暴力累加左端和右端小于或等于 h 的元素个数。

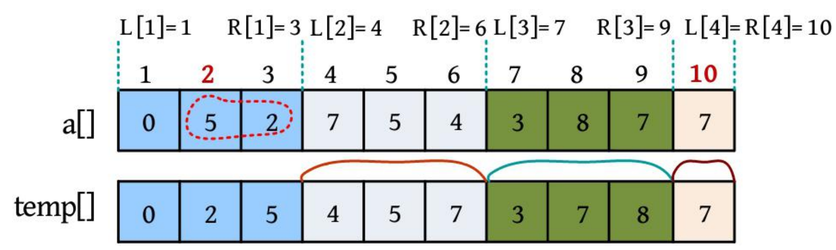
2. 图解

根据测试用例的输入数据，分块算法的求解过程如下。

（1）分块。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。原数组 a[ ]和每一块排序后的辅助数组 temp[ ]如下图所示。



（2）查询。1 9 4：因为题目中的下标从 0 开始，上图中的下标从 1 开始，所以实际上是查询[2, 10]区间高度小于或等于 4 的元素个数。[2, 10]区间跨 4 个块，左端第 1 个块没有完全包含，需要暴力统计 a[2]、a[3]小于或等于 4 的元素。后面 3 个块是完整的块，对完整的块可以直接用 upper_bound()函数在 temp 数组中统计，该函数利用有序性进行二分查找，效率较高。



2. 算法实现

upper_bound(begin, end, num)：从数组的 begin 位置到 end-1 位置二分查找第 1 个大于 num 的数字，若找到，则返回该数字的地址，否则返回 end。将返回的地址减去起始地址 begin，即可得到小于或等于 num 的元素个数。

```
void build(){//分块预处理
    int t=sqrt(n);
    int num=n/t;
    if(n%num) num++;
    for(int i=1;i<=num;i++)
        L[i]=(i-1)*t+1,R[i]=i*t;
    R[num]=n;
    for(int i=1;i<=n;i++)
        belong[i]=(i-1)/t+1;
    for(int i=1;i<=num;i++)
        sort(temp+L[i],temp+1+R[i]);//对每个块进行排序
}
```

```
int query(int l,int r,int h){//查询[l,r]区间有多少个数小于或等于 h
    int ans=0;
    if(belong[l]==belong[r]){
        for(int i=l;i<=r;i++)
            if(a[i]<=h) ans++;
    }
    else{
        for(int i=l;i<=R[belong[l]];i++)//左端
            if(a[i]<=h) ans++;
        for(int i=belong[l]+1;i<belong[r];i++)//中间
            ans+=upper_bound(temp+L[i],temp+R[i]+1,h)-temp-L[i];
        for(int i=L[belong[r]];i<=r;i++)//右端
            if(a[i]<=h) ans++;
    }
    return ans;
}
```

HDU5057

题目描述（HDU5057）：有由 N 个非负整数组成的序列：a[1], a[2], ..., a[N]，对该序列进行 M 个操作，操作形式：①S X Y，将 a[X]的值设置为 Y（a[X]=Y）；②Q L R D P，求[L, R]区间第 D 位是 P 的元素个数，L 和 R 是序列的索引。注意：第 1 位是最低有效位。

输入：第 1 行包含一个整数 T，表示测试用例的数量。每个测试用例的第 1 行都包含两个整数 N 和 M。第 2 行包含 N 个整数：a[1], a[2], ..., a[N]。接下来的 M 行操作，若类型为 S，则在该行中将包含两个整数 X、Y；若类型为 Q，则将包含 4 个整数 L、R、D、P。其中：$1 \le T \le 50$，$1 \le N, M \le 10^5$，$0 \le a[i] \le 2^{31}-1$，$1 \le X \le N$，$0 \le Y \le 2^{31}-1$，$1 \le L \le R \le N$，$1 \le D \le 10$，$0 \le P \le 9$。

输出：对每个 Q 操作，都单行输出答案。

输入样例

```
1
5 7
10 11 12 13 14
Q 1 5 2 1
Q 1 5 1 0
Q 1 5 1 1
Q 1 5 3 0
Q 1 5 3 1
S 1 100
Q 1 5 3 1
```

输出样例

```
1
1
5
0
1
```

题解：根据测试用例的输入数据，序列如下图所示。

1	2	3	4	5
10	11	12	13	14

• Q 1 5 2 1：查询到[1, 5]区间第 2 位是 1 的元素有 5 个。

• Q 1 5 1 0：查询到[1, 5]区间第 1 位是 0 的元素有 1 个。

• Q 1 5 1 1：查询到[1, 5]区间第 1 位是 1 的元素有 1 个。

• Q 1 5 3 0：查询到[1, 5]区间第 3 位是 0 的元素有 5 个。

• Q 1 5 3 1：查询到[1, 5]区间第 3 位是 1 的元素有 0 个。

• S 1 100：将第 1 个元素修改为 100。

1	2	3	4	5
100	11	12	13	14

• Q 1 5 3 1：查询到[1, 5]区间第 3 位是 1 的元素有 1 个。

本题包括点更新和区间查询，区间查询比较特殊，需要查询第 D 位是 P 的元素个数，可以采用分块的方法来解决。

1. 算法设计

（1）分块。划分块，统计每一块每一位上的元素个数。block[i][j][k]表示第 i 块中第 j 位是 k 的元素个数。

（2）查询。查询[l, r]区间第 d 位是 p 的元素个数。

• 若该区间属于同一块，则暴力累加块内第 d 位是 p 的元素个数。

• 若该区间包含多个块，则累加中间每一块 i 的 block[i][d][p]，然后暴力累加左端和右端第 d 位是 p 的元素个数。

（3）更新。将 a[x]的值更新为 y。因为原来 x 所属的块已统计了 a[x]每一位上的元素个数，所以此时需要减去，再将新的值 y 累加上即可。

2. 算法实现

```
int a[maxn],belong[maxn],L[maxn],R[maxn],block[400][12][12],n,m;
//block[i][j][k]表示第 i 个块中第 j 位是 k 的元素个数
int ten[11]={0,1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000};
void build(){//分块预处理
    int t=sqrt(n);
    int num=n/t;
    if(n%t)  num++;
    for(int i=1;i<=num;i++){
        L[i]=(i-1)*t+1;//每块的左右
        R[i]=i*t;
    }
    R[num]=n;
    for(int i=1;i<=n;i++)
        belong[i]=(i-1)/t+1;//所属的块
    for(int i=1;i<=n;i++){
        int temp=a[i];
        for(int j=1;j<=10;j++){//位数最多有 10 位，1<=D<=10
            block[belong[i]][j][temp%10]++;//块、位、位上的数
            temp/=10;
        }
    }
}
```

```
int query(int l,int r,int d,int p){//查询[l,r]区间第 d 位是 p 的元素个数
    int ans=0;
    if(belong[l]==belong[r]){//属于同一块
        for(int i=l;i<=r;i++)//暴力统计
            if((a[i]/ten[d])%10==p)
                ans++;
        return ans;
    }
    for(int i=belong[l]+1;i<belong[r];i++)//累加中间的块
        ans+=block[i][d][p];
    for(int i=l;i<=R[belong[l]];i++){//左端暴力累加
        if((a[i]/ten[d])%10==p)
            ans++;
    }
    for(int i=L[belong[r]];i<=r;i++){//右端暴力累加
        if((a[i]/ten[d])%10==p)
            ans++;
    }
    return ans;
}
```

```
void update(int x,int y){//将a[x]的值更新为y
    for(int i=1;i<=10;i++){//原来的统计数减少
        block[belong[x]][i][a[x]%10]--;
        a[x]/=10;
    }
    a[x]=y;
    for(int i=1;i<=10;i++){//新的统计数增加
        block[belong[x]][i][y%10]++;
        y/=10;
    }
}
```