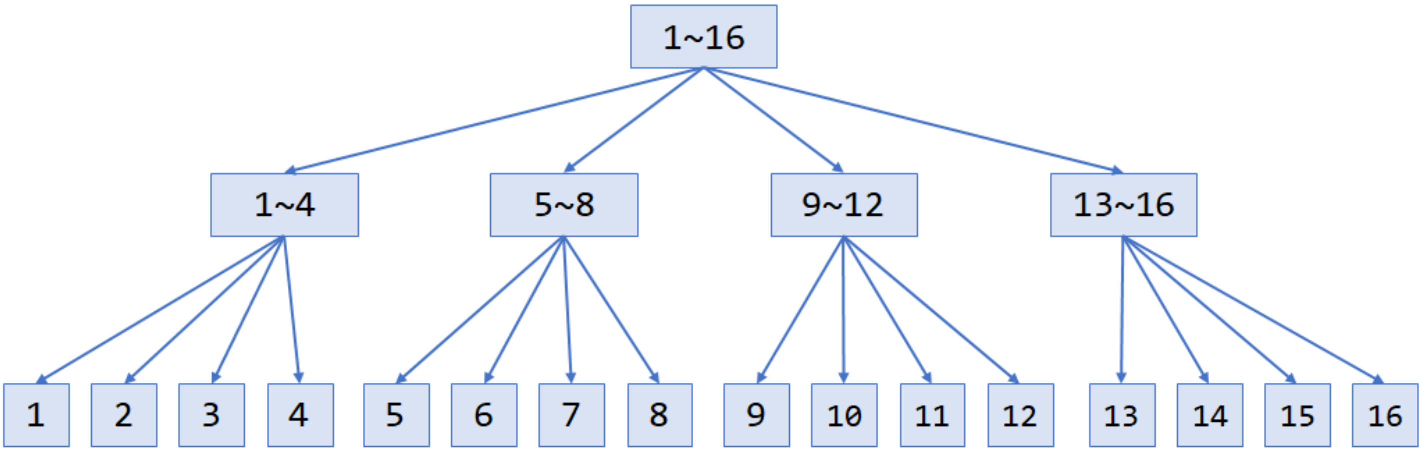


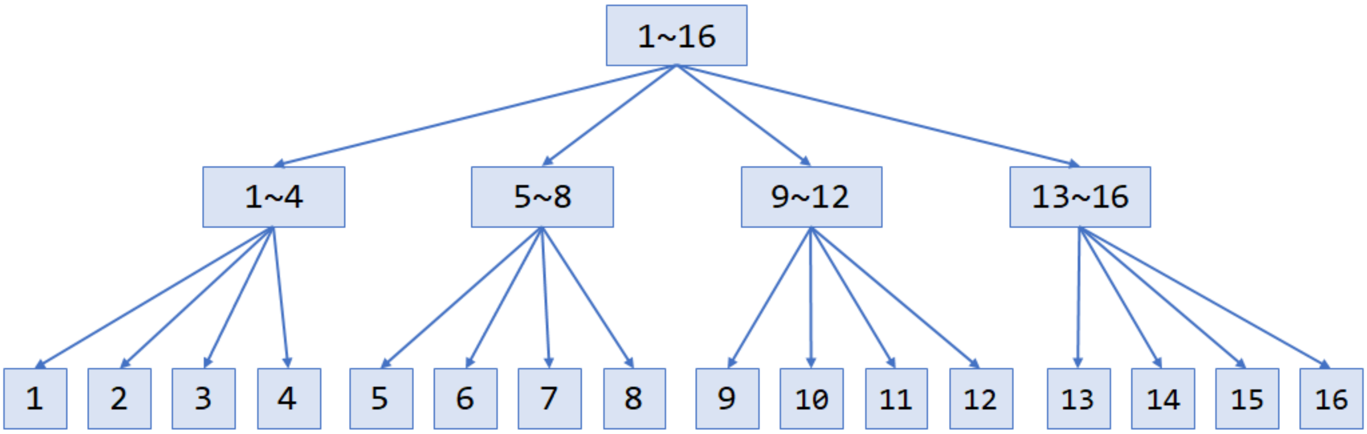
分块

分块是一种思想，把一个整体划分为若干个小块，对整块整体处理，零散块单独处理。本文主要介绍**块状数组**——利用分块思想处理区间问题的一种数据结构。

块状数组把一个长度为 n 的数组划分为 a 块，每块长度为 $\frac{n}{a}$ 。对于一次区间操作，对区间内部的整块进行整体的操作，对区间边缘的零散块单独暴力处理。（所以分块被称为“优雅的暴力”）

这里，块数既不能太少也不能太多。如果太少，区间中整块的数量会很少，我们要花费大量时间处理零散块；如果太多，又会让块的长度太短，失去整体处理的意义。一般来说，我们取块数为 $\sqrt{n}$ ，这样在最坏情况下，我们要处理接近 $\sqrt{n}$ 个整块，还要对长度为 $\frac{2n}{\sqrt{n}}=2\sqrt{n}$ 的零散块单独处理，总时间复杂度为 $O(\sqrt{n})$ 。这是一种**根号算法**。

显然，分块的时间复杂度比不上线段树和树状数组这些**对数级算法**。但由此换来的，是更高的灵活性。与线段树不同，块状数组并不要求所维护信息满足结合律，也不需要一层层地传递标记。但它们又有相似之处，线段树是一棵高度约为 $\log_2 n$ 的树，而块状数组则可被看成一棵高度为3的树：



只不过，块状数组最顶层的信息不用维护。

预处理

具体地使用块状数组，我们要先划定出每个块所占据的范围：

```
int sq = sqrt(n);
for (int i = 1; i <= sq; ++i)
{
    st[i] = n / sq * (i - 1) + 1; // st[i]表示i号块的第一个元素的下标
    ed[i] = n / sq * i; // ed[i]表示i号块的最后一个元素的下标
}
```

但是，数组的长度并不一定是一个完全平方数，所以这样下来很可能会漏掉一小块，我们把它们纳入最后一块中：

```
ed[sq] = n;
```

然后，我们为每个元素确定它所归属的块：

```
for (int i = 1; i <= sq; ++i)
    for (int j = st[i]; j <= ed[i]; ++j)
        bel[j] = i; // 表示j号元素归属于i块
```

最后，如果必要，我们再预处理每个块的大小：

```
for (int i = 1; i <= sq; ++i)
    size[i] = ed[i] - st[i] + 1;
```

好了，准备工作做完了，后面的事情就很简单了。分块的代码量也许不比线段树小多少，但看起来要好理解很多，我们先来搞线段树模板题。

（洛谷P3372 【模板】线段树1）

> **题目描述**
> 如题，已知一个数列，你需要进行下面两种操作：
>
> 将某区间每一个数加上 k。
> 求出某区间每一个数的和。
>
> **输入格式**
> 第一行包含两个整数 n, m，分别表示该数列数字的个数和操作的总个数。
> 第二行包含 n 个用空格分隔的整数，其中第 i 个数字表示数列第 i 项的初始值。
> 接下来 m 行每行包含 3 或 4 个整数，表示一个操作，具体如下：
> `1 x y k`：将区间 [x, y] 内每个数加上 k 。
> `2 x y`：输出区间 [x, y] 内每个数的和。

这个题数据范围只有 10^5，可以用分块。我们用一个 `sum` 数组来记录每一块的和，`mark` 数组来做标记（注意这两者要分开，因为处理零散块时也要用到标记）。

读入和预处理数据

```
for (int i = 1; i <= n; ++i)
    A[i] = read();
for (int i = 1; i <= sq; ++i)
    for (int j = st[i]; j <= ed[i]; ++j)
        sum[i] += A[j];
```

`sum[i]` 保存第 `i` 个块的和。

区间修改

首先是区间修改，当x与y在同一块内时，直接暴力修改原数组和 `sum` 数组：

```
if (bel[x] == bel[y])
    for (int i = x; i <= y; ++i)
    {
        A[i] += k;
        sum[bel[i]] += k;
    }
```

否则，先暴力修改左右两边的零散区间：

```
for (int i = x; i <= ed[bel[x]]; ++i)
{
    A[i] += k;
    sum[bel[i]] += k;
}
for (int i = st[bel[y]]; i <= y; ++i)
{
    A[i] += k;
    sum[bel[i]] += k;
}
```

然后对中间的整块打上标记：

```
for (int i = bel[x] + 1; i < bel[y]; ++i)
    mark[i] += k;
```

区间查询

同样地，如果左右两边在同一块，直接暴力计算区间和。

```
if (bel[x] == bel[y])
    for (int i = x; i <= y; ++i)
        s += A[i] + mark[bel[i]]; // 注意要加上标记
```

否则，暴力计算零碎块：

```
for (int i = x; i <= ed[bel[x]]; ++i)
    s += A[i] + mark[bel[i]];
for (int i = st[bel[y]]; i <= y; ++i)
    s += A[i] + mark[bel[i]];
```

再处理整块：

```
for (int i = bel[x] + 1; i < bel[y]; ++i)
    s += sum[i] + mark[i] * size[i]; // 注意标记要乘上块长
```

于是我们用分块A掉了线段树的模板题（线段树：喵喵喵？）。洛谷上跑得还挺快，比较数据范围很小。

上面用分块解决问题的思路非常简单。然而，如果总是这么简单就好了……实际上，分块的题目可以出得很灵活、很难。我们来看“相对”简单的一道：

（洛谷P2801 教主的魔法）

> **题目描述**
> 教主最近学会了一种神奇的魔法，能够使人长高。于是他准备演示给XMYZ信息组每个英雄看。于是N个英雄们又一次聚集在了一起，这次他们排成了一列，被编号为1、2、……、N。
> 每个人的身高一开始都是不超过1000的正整数。教主的魔法每次可以把闭区间[L, R]（1≤L≤R≤N）内的英雄的身高全部加上一个整数W。（虽然L=R时并不符合区间的书写规范，但我们可以认为是单独增加第L（R）个英雄的身高）
> CYZ、光哥和ZJQ等人不信教主的邪，于是他们有时候会问WD闭区间 [L, R] 内有多少英雄身高大于等于C，以验证教主的魔法是否真的有效。
> WD巨懒，于是他把这个回答的任务交给了你。
> **输入格式**
> 第1行为两个整数N、Q。Q为问题数与教主的施法数总和。
> 第2行有N个正整数，第i个数代表第i个英雄的身高。
> 第3到第Q+2行每行有一个操作：
> （1）若第一个字母为“M”，则紧接着有三个数字L、R、W。表示对闭区间 [L, R] 内所有英雄的身高加上W。
> （2）若第一个字母为“A”，则紧接着有三个数字L、R、C。询问闭区间 [L, R] 内有多少英雄的身高大于等于C。
> **输出格式**
> 对每个“A”询问输出一行，仅含一个整数，表示闭区间 [L, R] 内身高大于等于C的英雄数。
> **数据范围**
> 对30%的数据，N≤1000，Q≤1000。
> 对100%的数据，N≤1000000，Q≤3000，1≤W≤1000，1≤C≤1,000,000,000。

区间加，询问区间大于等于C的元素个数。这个用线段树显然不方便（总不可能对每个C开一棵线段树吧……）。发现虽然N最大为 10^6 ，但Q较小，分块可行。

零散块暴力处理起来显然很容易，但如何对整块整体处理呢？实际上我们可以维护整块**排序**后的数组，然后对整块进行询问时直接**二分查找**。

注意，我们每次对零散块单独修改时，都需要**更新**排序后的数组，这听起来很暴力，但由于每个块相对较少，也可以接受。总时间复杂度 $O(q\sqrt{n}\log\sqrt{n})$ 。

主要代码如下：

```
/* ... */
const int MAXN = 1000005, SQ = 1005;
int st[SQ], ed[SQ], size[SQ], bel[MAXN];
void init_block(int n) // 初始化
{
    int sq = sqrt(n);
    for (int i = 1; i <= sq; ++i)
    {
        st[i] = n / sq * (i - 1) + 1;
        ed[i] = n / sq * i;
    }
    ed[sq] = n;
    for (int i = 1; i <= sq; ++i)
        for (int j = st[i]; j <= ed[i]; ++j)
            bel[j] = i;
    for (int i = 1; i <= sq; ++i)
        size[i] = ed[i] - st[i] + 1;
}
int A[MAXN], mark[SQ];
vector<int> v[SQ]; // 这里用vector存排序后的数组
void update(int b) // 更新排序后的数组
{
    for (int i = 0; i <= size[b]; ++i)
        v[b][i] = A[st[b] + i];
    sort(v[b].begin(), v[b].end());
}
int main()
{
    int n = read(), m = read();
    int sq = sqrt(n);
    init_block(n);
    for (int i = 1; i <= n; ++i)
        A[i] = read();
```

```
    for (int i = 1; i <= sq; ++i)
        for (int j = st[i]; j <= ed[i]; ++j)
            v[i].push_back(A[j]);
    for (int i = 1; i <= sq; ++i)
        sort(v[i].begin(), v[i].end());
    while (m--)
    {
        char o;
        scanf(" %c", &o);
        int l = read(), r = read(), k = read();
        if (o == 'M')
        {
            if (bel[l] == bel[r]) // 如果同属一块直接暴力
            {
                for (int i = l; i <= r; ++i)
                    A[i] += k;
                update(bel[l]);
                continue;
            }
            for (int i = l; i <= ed[bel[l]]; ++i) // 对零散块暴力处理
                A[i] += k;
            for (int i = st[bel[r]]; i <= r; ++i)
                A[i] += k;
            update(bel[l]);
            update(bel[r]);
            for (int i = bel[l] + 1; i < bel[r]; ++i) // 打上标记
                mark[i] += k;
        }
        else
        {
            int tot = 0;
            if (bel[l] == bel[r])
            {
                for (int i = l; i <= r; ++i)
                    if (A[i] + mark[bel[l]] >= k)
                        tot++;
                printf("%d\n", tot);
                continue;
            }
            for (int i = l; i <= ed[bel[l]]; ++i)
                if (A[i] + mark[bel[l]] >= k)
                    tot++;
            for (int i = st[bel[r]]; i <= r; ++i)
                if (A[i] + mark[bel[r]] >= k)
                    tot++;
```

```
            // 二分查找k-mark[i]的位置，因为整块都加上了mark[i]其实就相当于k减去mark[i]
            for (int i = bel[l] + 1; i < bel[r]; ++i)
               tot += v[i].end() - lower_bound(v[i].begin(), v[i].end(), k - mark[i]);
            printf("%d\n", tot);
        }
    }
    return 0;
}
```