差分

区间修改与最少操作

第一题：区间增加

题目描述

给定长度为 n 的序列 a，再给定 m 次操作。每次输入 l, r, d，将 $a_l, a_{l+1}, \ldots, a_r$ 全部增加 d。所有操作结束后，输出最终序列。

d 可以为负数，此时表示减少相应的数值。

输入格式

第一行输入 n, m，$1 \leq n, m \leq 2 \times 10^5$。

第二行输入 n 个整数 a_i，$1 \leq a_i \leq 10^4$。

接下来 m 行，每行输入整数 l, r, d，满足 $1 \leq l \leq r \leq n$，$-10^4 \leq d \leq 10^4$。

输出格式

输出 n 个整数，表示操作结束后的序列。

样例输入

```
6 3
1 2 2 1 2 1
1 3 1
3 5 1
1 6 1
```

样例输出

```
3 4 5 3 4 2
```

先考虑暴力的方法

每次操作都枚举区间里的位置，逐个增加 d：

```
for (int i = l; i <= r; ++i) a[i] += d;
```

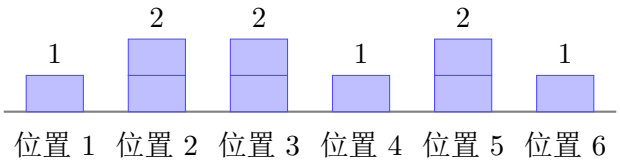
如果每次都改所有位置，位置数和操作数都达到 200000，就要做约 400 亿次加法，太慢了。

本题只在全部修改结束后输出，不需要在修改之间回答查询。可以先记下修改，最后统一算出结果。

原数组和差分，都可以画成积木

先看原数组：一块积木代表 1，每根柱子的高度就是这个位置的数。

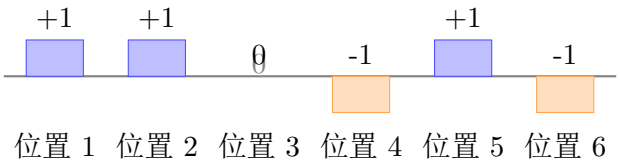
原数组：每个位置有多高



差分记录的是“这一根比左边那根高多少”。第一根前面，想象有一根高度为 0 的柱子。

例如第二根比第一根高 1，记 $+1$；第三根和第二根一样高，记 0；第四根比第三根矮 1，记 -1。

差分数组：与左边相比，增加还是减少



蓝色积木画在横线上方，表示要增加；橙色积木画在横线下方，表示要减少；零没有积木。

下面的橙色积木是“减少的记号”，不是原数组里真的有负数块积木。差分图中每个位置都从横线开始画，不能把它的柱高当成原数组的数。

前缀和，就是把变化从头累加

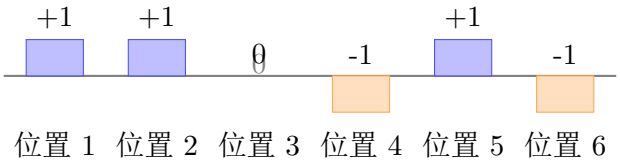
现在遮住原数组，只看差分的积木。手里一开始有 0 块，按位置从左向右走：

走到哪里	按照差分的记号做	手里现在有几块
位置 1	添 1 块	1
位置 2	再添 1 块	2
位置 3	不添也不取	2
位置 4	取走 1 块	1
位置 5	添 1 块	2
位置 6	取走 1 块	1

每走到一个位置，就记下手里有多少块，得到 $1, 2, 2, 1, 2, 1$，正好是原数组。

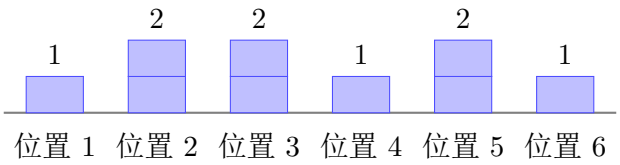
这种“从第一个位置开始，一项一项加进去”的动作，就叫**求前缀和**。不需要记复杂公式：**添的记号就加，取的记号就减，边走边记下结果。**

先看差分：这一步添几块或取几块



⇓ 从左到右累加 ⇓

累加后：得到原数组的积木



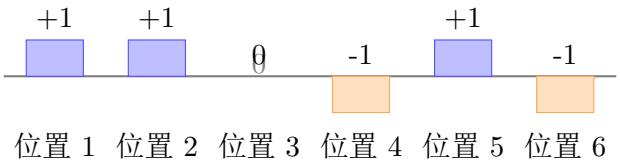
代码中的 `now` 就像手里当前的积木数量，每次执行 `now += b[i]`，再输出 `now`。

修改时，先让差分的两处积木变化

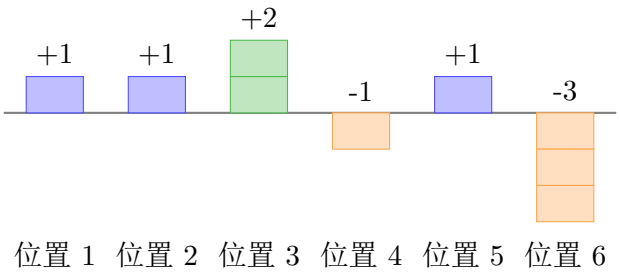
用一个具体例子：让位置 3 到位置 5 都增加 2。先不一根一根改原数组，只改差分的两个位置。

- 位置 3：原来是 0，加 2 后变为 +2，画两块向上的积木。
- 位置 6：原来是 −1，再减 2 后变为 −3，画三块向下的积木。
- 其他位置的差分积木都不变。

修改前的差分积木



修改后的差分积木：只改位置 3 和位置 6



位置 3 的记号表示“从这里开始，多带两块”；位置 6 的记号表示“从这里开始，取消多带的两块”。

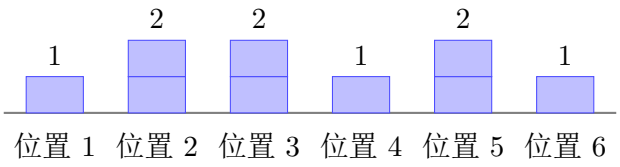
所以，区间内每个位置都会增加两块，而后面的地方不受影响。第二个记号要写在区间结束的**下一格**。

差分改好后，再累加成新的原数组

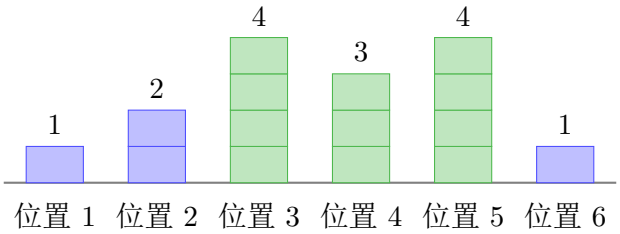
新的差分是 $1, 1, 2, -1, 1, -3$，从头累加：

位置	原来的数	新差分	重新累加得到的数
1	1	1	$0+1=1$
2	2	1	$1+1=2$
3	2	2	$2+2=4$
4	1	-1	$4-1=3$
5	2	1	$3+1=4$
6	1	-3	$4-3=1$

修改前的原数组



修改后的原数组：位置 3 到 5 都高了两块



差分只改了两格，累加得到的原数组却改变了一整段。这就是差分能加快区间修改的原因。

用代码记下这两个变化：

```
b[l] += d;
b[r + 1] -= d;
```

如果区间一直到最后一格，结束后的记号仍要有地方存。数组多开一个位置，最后输出时只输出原来的那些位置即可。

跟着样例计算

初始差分为 $[1,1,0,-1,1,-1]$，同时准备好额外位置 7，初值为 0。

步骤	b_1	b_2	b_3	b_4	b_5	b_6	b_7
初始	1	1	0	-1	1	-1	0
$[1,3]$ 加 1	2	1	0	-2	1	-1	0
$[3,5]$ 加 1	2	1	1	-2	1	-2	0
$[1,6]$ 加 1	3	1	1	-2	1	-2	-1

最后依次累加前六个差分，得到 $3,4,5,3,4,2$。

完整代码

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAX_N = 2e5 + 10;
ll b[MAX_N];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    ll last = 0;
    for (int i = 1; i <= n; ++i) {
        ll x;
        cin >> x;
        b[i] = x - last;
        last = x;
    }
    while (m--) {
        int l, r;
        ll d;
        cin >> l >> r >> d;
        b[l] += d;
        b[r + 1] -= d;
    }
    ll now = 0;
    for (int i = 1; i <= n; ++i) {
        now += b[i];
        if (i > 1) cout << ' ';
        cout << now;
    }
    cout << '\n';
    return 0;
}
```

复杂度与检查

开始时把数组读一遍，算好差分；每次操作只改两格；最后再从头累加一遍。需要的工作量大约是“两遍数组，加上每次操作的两次修改”，不用反复扫描整段。保存一份差分数组即可。

代码使用 `long long` 保存数值，便于应对多次修改的累计；位置和操作次数用 `int`。

- $l = r$ 时，只修改一个数，仍然在 l 加、$l + 1$ 减。
- $d < 0$ 时，原来的两句代码不变。例如加 -3 就是减少 3。
- $n = 1$ 时，额外位置 2 仍然可用，只输出位置 1 的累计值。
- 修改记在差分数组上，最终必须累加，不能直接输出差分。

思考

如果初始数组全为 0，差分应该如何初始化？为什么第二个标记必须放在 $r + 1$，而不能放在 r？

第二题：最少多少次减到 1？

题目描述

给定长度为 n 的数组 a。每次任选一个非空连续区间，将其中所有数减少 1。求使数组所有数都变成 1 的最少操作次数。

$1 \le n \le 10^5$，$1 \le a_i \le 10^5$，数据保证有解。

输入格式

第一行输入 n，第二行输入 n 个整数 a_i。

输出格式

输出一个整数，表示最少操作次数。

样例输入

```
6
1 3 5 2 7 1
```

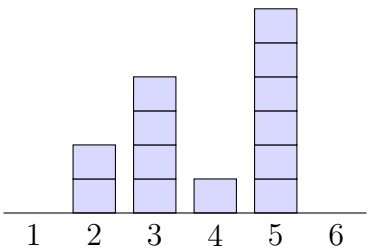
样例输出

```
9
```

先减去不需要拆掉的底座

每根柱子最后要留下 1 块底座。先把这块底座放到一旁，只看上面需要拆掉的积木。例如原来有 5 块，就还需要拆掉 4 块。代码用 exttth 表示这些多余积木的数量。

样例对应的高度为 $[0, 2, 4, 1, 6, 0]$。把它想成一排积木柱，每次可以从连续的几根柱子中各取走一块积木。



小数据可以一层一层拆

每一层中，连续有积木的位置组成一段，就可以一起拆。高度为 0 的柱子不能包含在操作中，因为它已经没有积木可拆。

但是柱子可能有 100000 层，每层都重新检查所有柱子，可能需要检查 100 亿次。能不能只看柱子之间的高低变化，直接数出要拆几次？

把拆积木倒过来看：从零搭起来

拆掉一段积木的动作倒过来，就是给一段连续柱子各添一块。能用几次搭起来，就能反过来用几次拆掉。

从左向右看：第一根有多少块多余积木，就要开始多少次添积木的动作。以后每遇到一根更高的柱子，**比左边多出的几块，就需要在这里新开始几次动作**。

例如左边有 2 块，当前有 4 块。左边已有的两层可以延续过来，但上面还差两层，必须新开始两次。

如果当前更矮，只需让一些动作在左边结束，不必新开动作。所以只把“比左边多的块数”加到答案里，下降不加。

为什么这样数，一定不多也不少？

一根柱子比左边高出的每一块，都不能由覆盖左边的动作来补，否则左边也会变高。因此这些新动作是省不掉的。

再按层看积木：同一层里，连续相连的一段，可以用一次动作一起搭出来。遇到空缺就分开，不能跨过去。这些横着的一段，恰好都从“比左边高”的地方开始。

所以，在每个上升位置数新开始的层数，既没有漏掉需要的动作，也没有多算。程序只需要比较相邻高度，不用真的搭一遍或拆一遍。

样例为什么是 9？

位置	原数	多余高度	从这里新开的层数
1	1	0	0
2	3	2	2
3	5	4	2
4	2	1	0
5	7	6	5
6	1	0	0

所以答案为 $0+2+2+0+5+0=9$。

一种实际的减法方案是：$[2,5]$ 减一次，$[2,3]$ 减一次，$[3,3]$ 减两次，$[5,5]$ 减五次，共九次。每次都从仍有多余高度的区间减，不会将任何数减到 1 以下。

与差分的关系

把多余积木画成差分图：上方的积木表示“这里比左边多了几块”，下方表示“这里比左边少了几块”。只把横线上方的积木数加起来，就是答案。

“上升”表示需要开始新的一段，“下降”表示某些段在这里结束。第一题用两端标记记录区间操作；第二题通过上升的差值，数出最少需要开始多少段操作。

第二题完整代码

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin >> n;
    ll last = 0, ans = 0;
    for (int i = 1; i <= n; ++i) {
        ll a;
        cin >> a;
        ll h = a - 1;
        if (h > last) ans += h - last;
        last = h;
    }
    cout << ans << '\n';
    return 0;
}
```

每根柱子只读一次，只记住上一根柱子的多余高度和当前答案，不必保存整个数组。

答案可能超过 32 位整数范围。例如 10^5 个数在 10^5 和 1 之间交替，每个高柱都需单独操作，答案可达 $50000 \times 99999 = 4999950000$，应使用 `long long`。

边界情况

- 所有数都是 1：没有多余积木，答案为 0。
- 只有一个数 a_1：答案为 $a_1 - 1$。
- 所有数都等于 k：可以对整个区间操作 $k - 1$ 次。
- 数组逐渐下降：第一根柱子需要的操作足以分段延续到后面，不需要额外开段。

自己试一试

求 $[2, 2, 2]$、$[5, 4, 2]$、$[1, 4, 1, 4]$ 的答案。参考结果依次为 $1, 4, 6$。

两题都没有提供部分分分组。逐个修改、逐层拆积木适合先在纸上理解。两份完整代码都不用反复扫描区间，可以处理题目给出的全部规模。