高级数据结构-线段树

一、AcWing 1275. 最大数

【题目描述】

给定一个正整数数列$a_1, a_2, \ldots, a_n$，每一个数都在$0 \sim p-1$之间。

可以对这列数进行两种操作：

- 添加操作：向序列后添加一个数，序列长度变成$n+1$；
- 询问操作：询问这个序列中最后L个数中最大的数是多少。

程序运行的最开始，整数序列为空。

一共要对整数序列进行m次操作。

写一个程序，读入操作的序列，并输出询问操作的答案。

【输入格式】

第一行有两个正整数m, p，意义如题目描述；

接下来m行，每一行表示一个操作。

如果该行的内容是`Q L`，则表示这个操作是询问序列中最后L个数的最大数是多少；

如果是`A t`，则表示向序列后面加一个数，加入的数是$(t+a) \ mod \ p$。其中，t是输入的参数，a是在这个添加操作之前最后一个询问操作的答案（如果之前没有询问操作，则$a=0$）。

第一个操作一定是添加操作。对于询问操作，$L>0$且不超过当前序列的长度。

【输出格式】

对于每一个询问操作，输出一行。该行只有一个数，即序列中最后L个数的最大数。

【数据范围】

$1 \le m \le 2 \times 10^5$

$1 \le p \le 2 \times 10^9$

$0 \le t < p$

【输入样例】

```
10 100
A 97
Q 1
Q 1
A 17
Q 2
A 63
Q 1
Q 1
Q 3
A 99
```

【输出样例】

```
97
97
97
60
60
97
```

【样例解释】

最后的序列是$97, 14, 60, 96$。

【分析】

线段树模板题，各函数功能详见代码注释部分。

【代码】

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>
using namespace std;

const int N = 200010;
int m, p;

struct Node

{
```

```
    int l, r, v;//表示区间[l, r]的最大值为v
}tr[N << 2];//线段树节点数一般开成N * 4

void pushup(int u)//由子节点的信息来计算父节点的信息
{
    tr[u].v = max(tr[u << 1].v, tr[u << 1 | 1].v);
}

//以u为根节点,在区间[l, r]上建立线段树的节点
void build(int u, int l, int r)
{
    tr[u].l = l, tr[u].r = r;
    if (l == r) return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

//从节点u开始找,查询区间[l, r]中的最大值
int query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].v;//线段树该节点的区间已被完全包含在[l, r]中了
    int mid = tr[u].l + tr[u].r >> 1, v = 0;
    if (l <= mid) v = query(u << 1, l, r);//如果要查询的区间与节点的左半区间有交集则递归查询左半区间
    if (r > mid) v = max(v, query(u << 1 | 1, l, r));//若与右半区间有交集则递归查询右半区间,整个区间的最大值为两个半区间的最大值
    return v;
}

//从节点u开始找,将节点x处的值修改为y
void modify(int u, int x, int y)
{
    if (tr[u].l == x && tr[u].r == x) tr[u].v = y;//如果已经找到叶节点且编号为x则将其值修改为y
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, y);//如果x在节点u的左半区间则递归修改其左儿子
        else modify(u << 1 | 1, x, y);//否则修改其右儿子
        pushup(u);//修改完u的儿子后需要通过儿子维护父节点

    }
```

```
}

int main()
{
    cin >> m >> p;
    build(1, 1, m);
    int n = 0, last = 0;//n表示当前节点数,last表示上次查询的结果
    while (m--)
    {
        string op;
        int x;
        cin >> op >> x;
        if (op == "Q") { last = query(1, n - x + 1, n); cout << last << endl; }
        else modify(1, ++n, ((long long)x + last) % p);//两数相加可能会爆int
    }
    return 0;
}
```

二、AcWing 245. 你能回答这些问题吗

【题目描述】

给定长度为N的数列A，以及M条指令，每条指令可能是以下两种之一：

1. `1 x y`，查询区间$[x,y]$中的最大连续子段和，即$max_{x\le l\le r\le y}\{\sum_{i=l}^{r}A[i]\}$。
2. `2 x y`，把$A[x]$改成y。

对于每个查询指令，输出一个整数表示答案。

【输入格式】

第一行两个整数N,M。

第二行N个整数$A[i]$。

接下来M行每行3个整数k,x,y，$k=1$表示查询（此时如果$x>y$，请交换x,y），$k=2$表示修改。

【输出格式】

对于每个查询指令输出一个整数表示答案。

每个答案占一行。

【数据范围】

$N \leq 500000, M \leq 100000$

$-1000 \leq A[i] \leq 1000$

【输入样例】

```
5 3
1 2 -3 4 5
1 2 3
2 2 -1
1 3 2
```

【输出样例】

```
2
-1
```

【分析】

区间最大连续字段和是一道非常经典的问题。

首先假设u表示父节点，l, r分别表示左右儿子节点。根据区间可见性，我们知道这里面必然会增加两个变量$lmax$和$rmax$分别管理前缀最大子段和和后缀最大子段和。然后根据区间可见性，显然$[l, r]$区间的最大连续子段和$u.tmax$就是左区间的最大连续子段和$l.tmax$、右区间的最大连续子段和$r.tmax$、左右两区间结合在一起的跨区间的最大连续子段和。跨区间的最大连续子段和为左区间的最大后缀和加上右区间的最大前缀和，即$l.rmax + r.lmax$。综上，$u.tmax = max(max(l.tmax, r.tmax), l.rmax + r.lmax)$

计算最大前缀和与最大后缀和还需要维护一个信息就是每个区间的区间和sum，某个区间的最大前缀和有以下两种情况：

1. 最大前缀和完全在其左儿子的区间范围内，则该区间的最大前缀和等于其左儿子的最大前缀和，即$u.lmax = l.lmax$；
2. 最大前缀和跨区间，则该区间的最大前缀和等于其左儿子的区间和加上右儿子的最大前缀和，即$u.lmax = l.sum + r.lmax$。

因此区间$[l, r]$的最大前缀和$u.lmax = max(l.lmax, l.sum + r.lmax)$，最大后缀和的维护方式同理。区间和的维护方式很简单，$u.sum = l.sum + r.sum$。

【代码】

```
#include <cstdio>
#include <cstring>
```

```cpp
#include <algorithm>
using namespace std;

const int N = 500010;
int w[N];
int n, m;

struct Node
{
    int l, r;
    int sum, lmax, rmax, tmax;//区间和,最大前缀和,最大后缀和,最大连续子段和
}tr[N << 2];

void pushup(Node& u, Node& l, Node& r)
{
    u.sum = l.sum + r.sum;
    u.lmax = max(l.lmax, l.sum + r.lmax);
    u.rmax = max(r.rmax, r.sum + l.rmax);
    u.tmax = max(max(l.tmax, r.tmax), l.rmax + r.lmax);
}

void pushup(int u)
{
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = { l, r, w[r], w[r], w[r], w[r] };
    else
    {
        tr[u] = { l, r };
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int x, int y)
{
    if (tr[u].l == x && tr[u].r == x) tr[u] = { x, x, y, y, y, y };
    else

    {
```

```
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, y);
        else modify(u << 1 | 1, x, y);
        pushup(u);
    }
}

Node query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u];
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (r <= mid) return query(u << 1, l, r);
        else if (l > mid) return query(u << 1 | 1, l, r);
        else
        {
            Node left = query(u << 1, l, r);
            Node right = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
    build(1, 1, n);
    while (m--)
    {
        int k, x, y;
        scanf("%d%d%d", &k, &x, &y);
        if (k == 1)
        {
            if (x > y) swap(x, y);
            printf("%d\n", query(1, x, y).tmax);
        }
        else modify(1, x, y);
    }

    return 0;
```

```
89  }
```

三、AcWing 246. 区间最大公约数

【题目描述】

给定一个长度为N的数列A，以及M条指令，每条指令可能是以下两种之一：

1. `C l r d`，表示把$A[l], A[l+1], \ldots, A[r]$都加上$d$。
2. `Q l r`，表示询问$A[l], A[l+1], \ldots, A[r]$的最大公约数(GCD)。
   对于每个询问，输出一个整数表示答案。

【输入格式】

第一行两个整数N, M。

第二行N个整数$A[i]$。

接下来M行表示M条指令，每条指令的格式如题目描述所示。

【输出格式】

对于每个询问，输出一个整数表示答案。

每个答案占一行。

【数据范围】

$N \leq 500000, M \leq 100000$

$1 \leq A[i] \leq 10^{18}$

$|d| \leq 10^{18}$

【输入样例】

```
5 5
1 3 5 7 9
Q 1 5
C 1 5 1
Q 1 5
C 3 3 6
Q 2 4
```

【输出样例】

```
1
2
4
```

【分析】

本题题面上涉及区间修改及区间查询，若是直接入手求解则难度很大，在求解之前我们需要了解最大公约数的一个性质：$gcd(a,b)=gcd(a,b-a)$。此性质又名更相减损术，可以推广至n个元素：$gcd(a_1,a_2,\dots,a_n)=gcd(a_1,a_2-a_1,\dots,a_n-a_{n-1})$。

有了这个式子就说明可以通过维护序列的差分序列来达到求gcd同样的效果，而使用差分序列也可以将区间修改转换成单点修改，假设$b[1],b[2],\dots,b[n]$为序列$a[1],a[2],\dots,a[n]$的差分序列，那么序列a在区间$[l,r]$上的最大公约数为：$gcd(a[l,r])=gcd(a[l],gcd(b[l+1,r]))$，因此使用线段树维护一个差分序列，$a[l]$即为区间$[1,l]$的和，$gcd(b[l+1,r])$即为该序列在区间$[l+1,r]$上的最大公约数，所以线段树的每个节点记录区间和以及区间最大公约数即可。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
using namespace std;

typedef long long LL;
const int N = 500010;
LL w[N];
int n, m;

struct Node
{
    int l, r;
    LL sum, d;//sum表示区间和,d表示区间gcd
}tr[N << 2];

LL gcd(LL a, LL b)
{
    return b ? gcd(b, a % b) : a;
}


void pushup(Node& u, Node& l, Node& r)
```

```
{
    u.sum = l.sum + r.sum;
    u.d = gcd(l.d, r.d);
}

void pushup(int u)
{
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = { l, r, w[r] - w[r - 1], w[r] - w[r - 1] };
    else
    {
        tr[u] = { l, r };
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int x, LL y)
{
    if (tr[u].l == x && tr[u].r == x) tr[u].sum += y, tr[u].d += y;
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, y);
        else modify(u << 1 | 1, x, y);
        pushup(u);
    }
}

Node query(int u, int l, int r)
{
    if (l > r) return { 0, 0, 0, 0 };
    if (tr[u].l >= l && tr[u].r <= r) return tr[u];
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (r <= mid) return query(u << 1, l, r);

        else if (l > mid) return query(u << 1 | 1, l, r);
```

```
        else
        {
            Node left = query(u << 1, l, r);
            Node right = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    while (m--)
    {
        string op;
        int l, r;
        cin >> op >> l >> r;
        if (op == "Q")
        {
            //gcd([l, r]) = gcd(a[l], gcd(b[l + 1, r]))
            Node left = query(1, 1, l), right = query(1, l + 1, r);
            cout << abs(gcd(left.sum, right.d)) << endl;
        }
        else
        {
            LL d;
            cin >> d;
            modify(1, l, d);
            if (r < n) modify(1, r + 1, -d);
        }
    }
    return 0;
}
```

四、AcWing 243. 一个简单的整数问题2

【题目描述】

给定一个长度为N的数列A，以及M条指令，每条指令可能是以下两种之一：

- `C l r d`，表示把$A[l], A[l+1], \ldots, A[r]$都加上$d$。
- `Q l r`，表示询问数列中第$l \sim r$个数的和。

对于每个询问，输出一个整数表示答案。

【输入格式】

第一行包含两个整数N和M。

第二行包含N个整数$A[i]$。

接下来M行表示M条指令，每条指令的格式如题目描述所示。

【输出格式】

对于每个询问，输出一个整数表示答案。

每个答案占一行。

【数据范围】

$1 \leq N, M \leq 10^5$

$|d| \leq 10000$

$|A[i]| \leq 10^9$

【输入样例】

```
10 5
1 2 3 4 5 6 7 8 9 10
Q 4 4
Q 1 10
Q 2 4
C 3 6 3
Q 2 4
```

【输出样例】

```
4
55
9
15
```

【分析】

对于区间修改操作，我们可以给线段树加上懒标记***add***，表示以当前节点为根的不包含根节点的子树中的每一个点加上***add***，此外维护一个***sum***值表示区间和。当我们进行修改与查询操作时，如果需要分裂区间，那么在分裂之前如果根节点有懒标记，则需要在递归前先将根节点的懒标记传递给子节点并更新子节点的***sum***值，同时清空根节点的懒标记，完成此操作的函数即为***pushdown***函数，具体实现代码如下。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#define ls u << 1
#define rs u << 1 | 1
using namespace std;

typedef long long LL;
const int N = 100010;
int n, m;
int w[N];

struct Node
{
    int l, r;
    LL sum, add;
}tr[N << 2];

void pushup(int u)
{
    tr[u].sum = tr[ls].sum + tr[rs].sum;
}

void pushdown(int u)
{
    if (tr[u].add)
    {
        tr[ls].add += tr[u].add, tr[ls].sum += (LL)(tr[ls].r - tr[ls].l + 1) * tr[u].add;
        tr[rs].add += tr[u].add, tr[rs].sum += (LL)(tr[rs].r - tr[rs].l + 1) * tr[u].add;
        tr[u].add = 0;

    }
```

```
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = { l, r, w[r], 0 };
    else
    {
        tr[u] = { l, r };
        int mid = l + r >> 1;
        build(ls, l, mid), build(rs, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, int y)
{
    if (tr[u].l >= l && tr[u].r <= r) tr[u].add += y, tr[u].sum += (LL)(tr[u].r - tr[u].l + 1) * y;
    else//必须要分裂
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(ls, l, r, y);
        if (r > mid) modify(rs, l, r, y);
        pushup(u);
    }
}

LL query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
    else
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        LL res = 0;
        if (l <= mid) res += query(ls, l, r);
        if (r > mid) res += query(rs, l, r);
        return res;
    }
}


int main()
```

```
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    while (m--)
    {
        string op;
        int l, r, d;
        cin >> op >> l >> r;
        if (op == "C") { cin >> d; modify(1, l, r, d); }
        else cout << query(1, l, r) << endl;
    }
    return 0;
}
```

五、AcWing 247. 亚特兰蒂斯

【题目描述】

有几个古希腊书籍中包含了对传说中的亚特兰蒂斯岛的描述。

其中一些甚至包括岛屿部分地图。

但不幸的是，这些地图描述了亚特兰蒂斯的不同区域。

您的朋友Bill必须知道地图的总面积。

你自告奋勇写了一个计算这个总面积的程序。

【输入格式】

输入包含多组测试用例。

对于每组测试用例，第一行包含整数n，表示总的地图数量。

接下来n行，描绘了每张地图，每行包含四个数字x_1,y_1,x_2,y_2（不一定是整数），(x_1,y_1)和(x_2,y_2)分别是地图的左上角位置和右下角位置。

注意，坐标轴x轴从上向下延伸，y轴从左向右延伸。

当输入用例$n=0$时，表示输入终止，该用例无需处理。

【输出格式】

每组测试用例输出两行。

第一行输出`Test case #k`，其中k是测试用例的编号，从1开始。

第二行输出`Total explored area: a`，其中a是总地图面积（即此测试用例中所有矩形的面积并，注意如果一片区域被多个地图包含，则在计算总面积时只计算一次），精确到小数点后两位数。

在每个测试用例后输出一个空行。

【数据范围】

$1 \leq n \leq 10000$

$0 \leq x_1 < x_2 \leq 100000$

$0 \leq y_1 < y_2 \leq 100000$

注意，本题n的范围上限加强至10000。

【输入样例】

```
2
10 10 20 20
15 15 25 25.5
0
```

【输出样例】

```
Test case #1
Total explored area: 180.00
```

【分析】

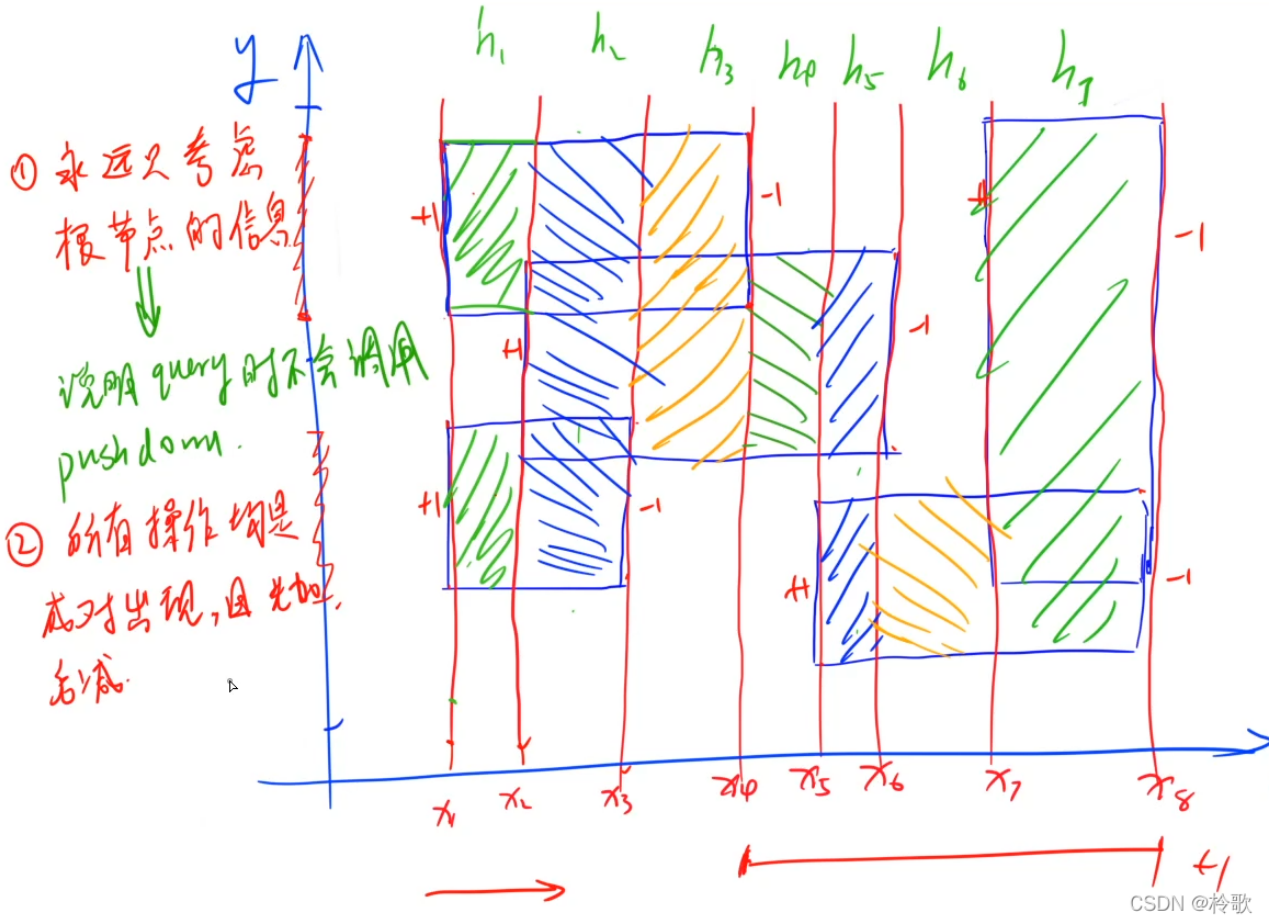
首先由于坐标是实数，可以有无穷多种取值，数量太大了，我们需要离散化。

对于每个线段树的节点，我们需要维护两个值：

- cnt：这个点所代表的线段被覆盖了多少次。
- len：以这个点为根的子树中被覆盖的区间一共有多长。

当一条线段进来的时候，在代表它的那若干个节点上$cnt++$，其它节点cnt不用加。

然后len维护的就是这个区间内那些$cnt > 0$的节点所覆盖的区间总长。



操作1：将某个区间 $[L, R]$ + k.

操作2：整个区间中长度大于0的区间总长是多少？

线段树中的节点信息：

① cnt. 当前区间整个被覆盖次数.

② len. 不考虑祖先节点 cnt 的前提下，cnt > 0 的区间总长

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <vector>
#define ls u << 1
#define rs u << 1 | 1
using namespace std;

const int N = 10010;
int n, T = 1;
vector<double> ys;//用于将y离散化

struct Segment//每一条竖直的线段
{
    double x, y1, y2;
    int k;//每个矩形左边的竖直线权值为1,右边的权值为-1
    bool operator< (const Segment& t)const
    {
        return x < t.x;//按横坐标为每条直线排序
    }
}seg[N << 1];

struct Node
{
    int l, r, cnt;//cnt表示这个点所代表的线段被覆盖了多少次
    double len;//len表示以这个点为根的子树中被覆盖的区间一共有多长
}tr[N << 3];

int find(double y)
{
    return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
}

void pushup(int u)
{
    if (tr[u].cnt) tr[u].len = ys[tr[u].r + 1] - ys[tr[u].l];
    else if (tr[u].l == tr[u].r) tr[u].len = 0;
    else tr[u].len = tr[ls].len + tr[rs].len;
}

void build(int u, int l, int r)
{
    tr[u] = { l, r, 0, 0 };
```

```
    if (l == r) return;
    int mid = l + r >> 1;
    build(ls, l, mid), build(rs, mid + 1, r);
}

void modify(int u, int l, int r, int y)
{
    if (tr[u].l >= l && tr[u].r <= r)
    {
        tr[u].cnt += y;
        pushup(u);
    }
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(ls, l, r, y);
        if (r > mid) modify(rs, l, r, y);
        pushup(u);
    }
}

int main()
{
    while (cin >> n, n)
    {
        ys.clear();
        for (int i = 0, j = 0; i < n; i++)
        {
            double x1, x2, y1, y2;
            cin >> x1 >> y1 >> x2 >> y2;
            seg[j++] = { x1, y1, y2, 1 }, seg[j++] = { x2, y1, y2, -1 };
            ys.push_back(y1), ys.push_back(y2);
        }
        sort(seg, seg + n * 2);
        sort(ys.begin(), ys.end());
        ys.erase(unique(ys.begin(), ys.end()), ys.end());
        build(1, 0, ys.size() - 1);
        double res = 0;
        for (int i = 0; i < n << 1; i++)
        {
            if (i > 0) res += tr[1].len * (seg[i].x - seg[i - 1].x);
            modify(1, find(seg[i].y1), find(seg[i].y2) - 1, seg[i].k);

        }
```

```
87          printf("Test case #%d\n", T++);
88          printf("Total explored area: %.2lf\n\n", res);
89      }
90      return 0;
91  }
```

六、AcWing 1277. 维护序列

【题目描述】

老师交给小可可一个维护数列的任务，现在小可可希望你来帮他完成。

有长为N的数列，不妨设为$a_1, a_2, \ldots, a_N$。

有如下三种操作形式：

1. 把数列中的一段数全部乘一个值；
2. 把数列中的一段数全部加一个值；
3. 询问数列中的一段数的和，由于答案可能很大，你只需输出这个数模P的值。

【输入格式】

第一行两个整数N和P；

第二行含有N个非负整数，从左到右依次为$a_1, a_2, \ldots, a_N$；

第三行有一个整数M，表示操作总数；

从第四行开始每行描述一个操作，输入的操作有以下三种形式：

- `1 t g c`，表示把所有满足$t \le i \le g$的a_i改为$a_i \times c$；
- `2 t g c`，表示把所有满足$t \le i \le g$的a_i改为$a_i + c$；
- `3 t g`，询问所有满足$t \le i \le g$的a_i的和模P的值。

同一行相邻两数之间用一个空格隔开，每行开头和末尾没有多余空格。

【输出格式】

对每个操作**3**，按照它在输入中出现的顺序，依次输出一行一个整数表示询问结果。

【数据范围】

$1 \le N, M \le 10^5$

$1 \le t \le g \le N$

$0 \le c, a_i \le 10^9$

$1 \le P \le 10^9$

【输入样例】

```
7 43
1 2 3 4 5 6 7
5
1 2 5 5
3 2 4
2 3 7 9
3 1 3
3 4 7
```

【输出样例】

```
2
35
8
```

【样例解释】

初始时数列为$\{1,2,3,4,5,6,7\}$；

经过第1次操作后，数列为$\{1,10,15,20,25,6,7\}$；

对第2次操作，和为$10+15+20=45$，模43的结果是2；

经过第3次操作后，数列为$\{1,10,24,29,34,15,16\}$；

对第4次操作，和为$1+10+24=35$，模43的结果是35；

对第5次操作，和为$29+34+15+16=94$，模43的结果是8。

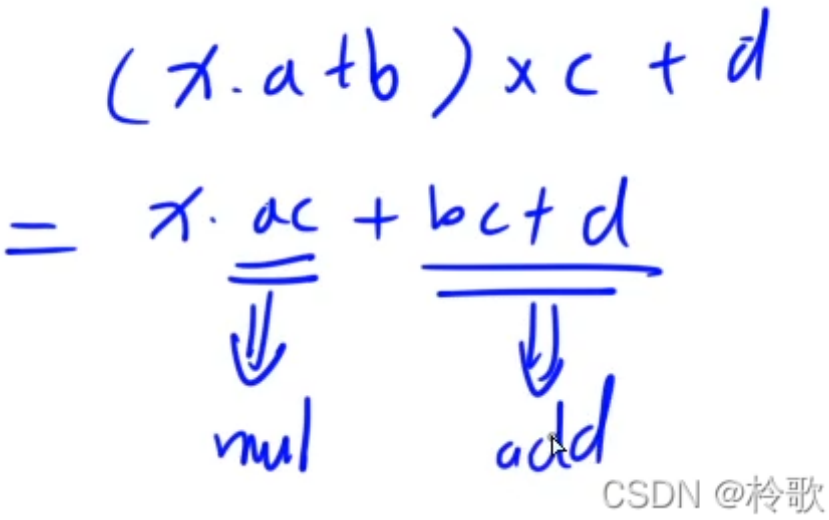
【分析】

首先我们考虑将加和乘合并成一个修改操作，即使用一个$modify$函数完成，线段树节点需要记录的信息有：区间和sum、区间加法懒标记add、区间乘法懒标记mul，假设当前懒标记情况为先乘一个数再加一个数，即$x\times mul+add$，对节点添加懒标记时有以下两种情况：

1. 添加一个加法懒标记，即$x\times mul+add+add'$，此时可以把$add+add'$看成新的加法懒标记；
2. 添加一个乘法懒标记，即$(x\times mul+add)\times mul'=x\times mul\times mul'+add\times mul'$，此时可以把$mul\times mul'$看成新的乘法懒标记，把$add\times mul'$看成新的加法懒标记。

因此这种添加新的懒标记是很好融合进去的，很容易变成统一的形式。

将加法与乘法合并成先乘后加的统一操作后，即对于$x \times c + d$，如果当前操作为加法，则令$c = 1$，如果当前操作为乘法，则令$d = 0$。



【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#define ls u << 1
#define rs u << 1 | 1
using namespace std;

typedef long long LL;
const int N = 100010;
int n, m, p;
LL w[N];

struct Node
{
    int l, r;
    LL sum, add, mul;
}tr[N << 2];

//对节点u先乘mul再加add
void eval(Node& u, LL add, LL mul)
{
    u.sum = (u.sum * mul + (u.r - u.l + 1) * add) % p;
    u.add = (u.add * mul + add) % p;

    u.mul = (u.mul * mul) % p;
```

```
}

void pushup(int u)
{
    tr[u].sum = (tr[ls].sum + tr[rs].sum) % p;//注意求余
}

void pushdown(int u)
{
    eval(tr[ls], tr[u].add, tr[u].mul);//由节点u向左儿子传递标记
    eval(tr[rs], tr[u].add, tr[u].mul);//向右儿子传递标记
    tr[u].add = 0, tr[u].mul = 1;//复原父节点标记
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = { l, r, w[r], 0, 1 };
    else
    {
        tr[u] = { l, r, 0, 0, 1 };//注意非叶节点也要初始化add和mul
        int mid = l + r >> 1;
        build(ls, l, mid), build(rs, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, LL add, LL mul)
{
    if (tr[u].l >= l && tr[u].r <= r) eval(tr[u], add, mul);
    else
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(ls, l, r, add, mul);
        if (r > mid) modify(rs, l, r, add, mul);
        pushup(u);
    }
}

LL query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;

    else
```

```
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        LL res = 0;
        if (l <= mid) res += query(ls, l, r);
        if (r > mid) res = (res + query(rs, l, r)) % p;
        return res;
    }
}

int main()
{
    cin >> n >> p;
    for (int i = 1; i <= n; i++) cin >> w[i];
    build(1, 1, n);
    cin >> m;
    while (m--)
    {
        int op, l, r, d;
        cin >> op >> l >> r;
        if (op == 1)//区间乘法
        {
            cin >> d;
            modify(1, l, r, 0, d);//区间的数先乘d再加0
        }
        else if (op == 2)//区间加法
        {
            cin >> d;
            modify(1, l, r, d, 1);//区间的数先乘1再加d
        }
        else cout << query(1, l, r) << endl;
    }
    return 0;
}
```