

HỘI THẢO KHOA HỌC

CHUYÊN ĐỀ

**CẤU TRÚC DỮ LIỆU CÂY PHÂN ĐOẠN
(SEGMENT TREE)**

MỤC LỤC

CẤU TRÚC DỮ LIỆU CÂY PHÂN ĐOẠN (SEGMENT TREE).....	1
A. PHẦN MỞ ĐẦU	4
B. PHẦN NỘI DUNG	5
I. CẤU TRÚC DỮ LIỆU CÂY PHÂN ĐOẠN (SEGMENT TREE).....	5
1. Giới thiệu.....	5
2. Segment tree cổ điển	9
3. Cập nhật lười (Lazy Propagation).....	13
II. BÀI TẬP ÁP DỤNG	16
1. Bài 1: Hiệu lớn nhất.....	16
1.1. Đề bài.....	16
1.2. Hướng dẫn giải thuật.....	17
1.3. Cài đặt.....	17
1.4. Test.....	19
2. Bài 2: Khoảng cách.....	19
2.1. Đề bài.....	19
2.2. Hướng dẫn giải thuật.....	20
2.3. Cài đặt.....	21
2.4. Test	24
3. Bài 3: Xếp nhóm.....	24
3.1. Đề bài.....	24
3.2. Hướng dẫn giải thuật.....	25
3.3. Cài đặt.....	26
3.4. Test	29
4. Bài 4: Khớp dữ liệu.....	30
4.1. Đề bài.....	30
4.2. Hướng dẫn giải thuật.....	31
4.3. Cài đặt.....	32
4.4. Test	34
5. Bài 5: Hoán đổi.....	35
5.1. Đề bài.....	35
5.2. Hướng dẫn giải thuật.....	35
5.3. Cài đặt.....	37

5.4. Test	40
6. Bài 6: Đếm tập con chẵn.....	40
6.1. Đề bài.....	40
6.2. Hướng dẫn giải thuật.....	41
6.3. Cài đặt.....	43
6.4. Test.....	45
III. MỘT SỐ BÀI TẬP LUYỆN THÊM	45
C. PHẦN KẾT LUẬN.....	46
D. TÀI LIỆU THAM KHẢO	47

A. PHẦN MỞ ĐẦU

Trong Tin học, khi thiết kế chương trình cho một bài toán, việc chọn cấu trúc dữ liệu là vấn đề rất quan trọng vì mỗi loại cấu trúc dữ liệu phù hợp với một vài loại bài toán ứng dụng khác nhau. Một cấu trúc dữ liệu được chọn cẩn thận sẽ cho phép thực hiện thuật toán hiệu quả hơn.

Trong chương trình bồi dưỡng học sinh giỏi, vấn đề sử dụng cấu trúc dữ liệu đặc biệt để giải các bài là một trong những vấn đề rất hay nhưng cũng rất khó. Hiện nay, phổ biến rất nhiều loại cấu trúc dữ liệu khác nhau như: Ngăn xếp (stack), hàng đợi (queue), băm (Hashing), cây phân đoạn (Segment Tree), BIT (Binary Indexed Tree),

...

Trong bài viết này, tôi chỉ trình bày cấu trúc dữ liệu cây phân đoạn (Segment Tree), không mang nặng vấn đề lí thuyết đầy đủ, chỉ ở mức tổng quát để đi đến các ứng dụng giải các bài toán cụ thể.

B. PHẦN NỘI DUNG

I. CẤU TRÚC DỮ LIỆU CÂY PHÂN ĐOẠN (SEGMENT TREE)

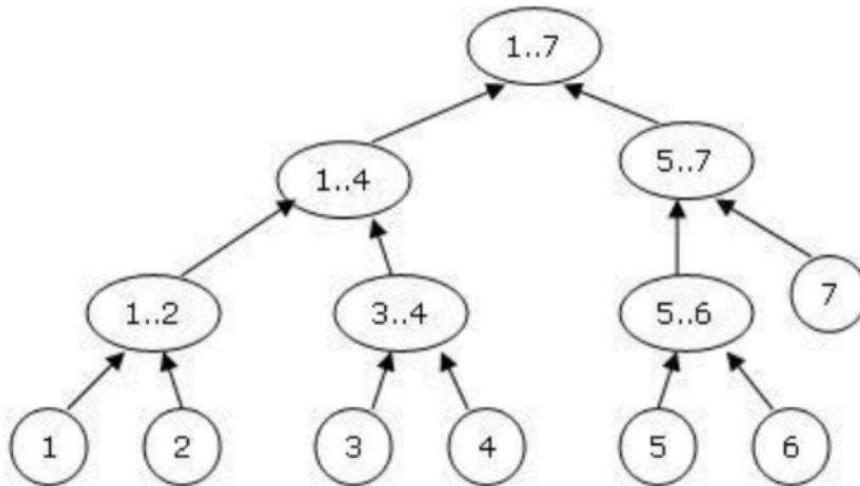
1. Giới thiệu

Segment Tree là một cấu trúc dữ liệu được sử dụng rất nhiều trong các kỳ thi, đặc biệt là trong những bài toán xử lý trên dãy số.

Segment Tree là một cây. Cụ thể hơn, nó là một cây nhị phân đầy đủ (mỗi nút là lá hoặc có đúng 2 nút con), với mỗi nút quản lý một đoạn trên dãy số. Với một dãy số gồm N phần tử, nút gốc sẽ lưu thông tin về đoạn $[1, N]$, nút con trái của nó sẽ lưu thông tin về đoạn $[1, \lfloor N/2 \rfloor]$ và nút con phải sẽ lưu thông tin về đoạn $[\lceil N/2 \rceil + 1, N]$. Tổng quát hơn: nếu nút A lưu thông tin đoạn $[i, j]$, thì 2 con của nó: A_1 và A_2 sẽ lưu thông tin của các đoạn $[i, \lfloor (i+j)/2 \rfloor]$ và $[\lceil (i+j)/2 \rceil + 1, j]$.

Ví dụ:

Xét một dãy gồm 7 phần tử, Segment Tree sẽ quản lý các đoạn như sau:



Cài đặt

Để cài đặt, ta có thể dùng một mảng 1 chiều, phần tử thứ nhất của mảng thể hiện nút gốc. Phần tử thứ id sẽ có 2 con là $2*id$ (con trái) và $2*id + 1$ (con phải). Với cách cài đặt này, người ta đã chứng minh được bộ nhớ cần dùng cho ST không quá $4*N$ phần tử.

Áp dụng

Để dễ hình dung, ta lấy 1 ví dụ cụ thể:

- Cho dãy N phần tử ($N \leq 10^5$). Ban đầu mỗi phần tử có giá trị 0.
- Có Q truy vấn ($Q \leq 10^5$). Mỗi truy vấn có 1 trong 2 loại:
 1. Gán giá trị v cho phần tử ở vị trí i.
 2. Tìm giá trị lớn nhất cho đoạn [i,j].

Cách đơn giản nhất là dùng 1 mảng A duy trì giá trị các phần tử. Với thao tác 1 thì ta gán $A[i]=v$. Với thao tác 2 thì ta dùng 1 vòng lặp từ i đến j để tìm giá trị lớn nhất. Rõ ràng cách này có độ phức tạp là $O(N*Q)$ và không thể chạy trong thời gian cho phép.

Cách dùng Segment Tree như sau:

- Với truy vấn loại 1, ta sẽ cập nhật thông tin của các nút trên cây ST mà đoạn nó quản lý chứa phần tử i.
- Với truy vấn loại 2, ta sẽ tìm tất cả các nút trên cây ST mà đoạn nó quản lý nằm trong [i,j], rồi lấy max của các nút này.

Cài đặt như sau:

```
// Truy vấn: A(i) = v
// Hàm cập nhật trên cây ST, cập nhật cây con gốc id quản lý đoạn [l, r]
void update(int id, int l, int r, int i, int v) {
    if (i < l || r < i) {
        // i nằm ngoài đoạn [l, r], ta bỏ qua nút i
        return ;
    }
    if (l == r) {
        // Đoạn chỉ gồm 1 phần tử, không có nút con
        ST[id] = v;
        return ;
    }

    // Gọi đệ quy để xử lý các nút con của nút id
    int mid = (l + r) / 2;
    update(id*2, l, mid, i, v);
    update(id*2 + 1, mid+1, r, i, v);
```

```

// Cập nhật lại giá trị max của đoạn [l, r] theo 2 nút con:
ST[id] = max(ST[id*2], ST[id*2 + 1]);
}

// Truy vấn: tìm max đoạn [u, v]
// Hàm tìm max các phần tử trên cây ST nằm trong cây con gốc id - quản lý đoạn [l, r]
int get(int id, int l, int r, int u, int v) {
    if (v < l || r < u) {
        // Đoạn [u, v] không giao với đoạn [l, r], ta bỏ qua đoạn này
        return -INFINITY;
    }
    if (u <= l && r <= v) {
        // Đoạn [l, r] nằm hoàn toàn trong đoạn [u, v] mà ta đang truy vấn, ta trả lại
        // thông tin lưu ở nút id
        return ST[id];
    }
    int mid = (l + r) / 2;
    // Gọi đệ quy với các con của nút id
    return max(get(id*2, l, mid, u, v), get(id*2 + 1, mid+1, r, u, v));
}

```

Phân tích thời gian chạy

Mỗi thao tác truy vấn trên cây ST có độ phức tạp $O(\log N)$. Để chứng minh điều này, ta xét 2 loại thao tác trên cây ST:

1. Truy vấn 1 phần tử trên ST (giống thao tác `update` ở trên)
2. Truy vấn nhiều phần tử trên ST (giống thao tác `get` ở trên)

Đầu tiên ta có thể chứng minh được:

- Độ cao của cây ST không quá $O(\log N)$.
- Tại mỗi độ sâu của cây, không có phần tử nào nằm trong 2 nút khác nhau của cây.

Thao tác loại 1

Với thao tác này, ở mỗi độ sâu của cây, ta chỉ gọi đệ quy các con của nó không quá 1 nút. Phân tích đoạn code trên, ta xét các trường hợp:

- Phần tử cần xét không nằm trong đoạn $[l, r]$ do nút id quản lý. Trường hợp này ta dừng lại, không xét tiếp.

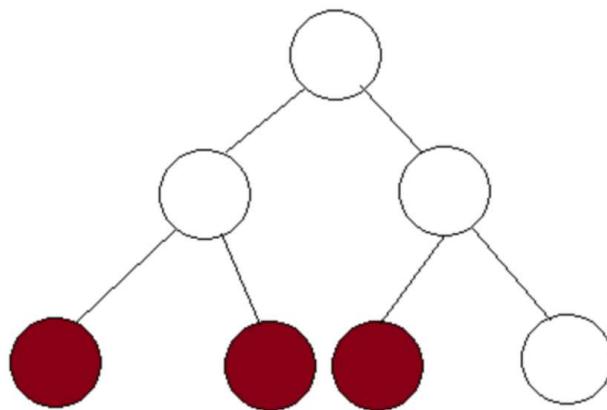
- Phần tử cần xét nằm trong đoạn $[l,r]$ do nút id quản lý. Ta xét các con của nút \boxed{id} . Tuy nhiên chỉ có 1 con của nút \boxed{id} chứa phần tử cần xét và ta sẽ phải xét tiếp các con của nút này. Với con còn lại, ta sẽ dừng ngay mà không xét các con của nó nữa.

Do đó độ phức tạp của thao tác này không quá $O(\log N)$.

Thao tác loại 2

Với thao này, ta cũng chứng minh tương tự, nhưng ở mỗi độ sâu của cây, ta chỉ gọi hàm đệ quy với các con của nó không quá 2 nút.

Ta chứng minh bằng phản chứng, giả sử ta gọi đệ quy với 3 nút khác nhau của cây ST (đánh dấu màu đỏ):



Trong trường hợp này, rõ ràng toàn bộ đoạn của nút ở giữa quản lý nằm trong đoạn đang truy vấn. Do đó ta không cần phải gọi đệ quy các con của nút ở giữa. Từ đó suy ra vô lý, nghĩa là ở mỗi độ sâu ta chỉ gọi đệ quy với không quá 2 nút.

Phân tích bộ nhớ

Ta xét 2 trường hợp:

- $N = 2^k$: Cây ST đầy đủ, ở độ sâu cuối cùng có đúng 2^k lá, và các độ sâu thấp hơn không có nút lá nào (và các nút này đều có đúng 2 con). Như vậy:
 - Tầng k : có 2^k nút

- Tầng $k-1$: có 2^{k-1} nút
- ... Tổng số nút không quá 2^{k+1} .
- Với $N > 2^k$ và $N < 2^{k+1}$. Số nút của cây ST không quá số nút của cây ST với $N=2^{k+1}$.

Do đó, số nút của cây cho dãy N phần tử, với $N \leq 2^k$ là không quá 2^{k+1} , giá trị này xấp xỉ $4*N$. Bằng thực nghiệm, ta thấy dùng $4*N$ là đủ.

2. Segment tree cỗ điển

Tại sao lại gọi là cỗ điển? Đây là dạng ST đơn giản nhất, chúng ta chỉ giải quyết truy vấn update một phần tử và truy vấn đoạn, mỗi nút lưu một loại dữ liệu cơ bản như số nguyên, boolean, ...

Ví dụ 1

Cho một dãy ngoặc độ dài N ($N \leq 10^6$), cho M truy vấn có dạng $[l_i, r_i]$ ($1 \leq l_i \leq r_i \leq N$). Yêu cầu của bài toán là với mỗi truy vấn tìm một chuỗi con (không cần liên tiếp) của chuỗi từ l_i đến r_i dài nhất mà tạo thành dãy ngoặc đúng.

Lời giải

Với mỗi nút(ví dụ như nút id, quản lý đoạn $[l, r]$) chúng ta lưu ba biến nguyên:

- **optimal**: Là kết quả tối ưu trong đoạn $[l, r]$.
- **open**: Số lượng dấu $($ sau khi đã xóa hết các phần tử thuộc dãy ngoặc đúng độ dài **optimal** trong đoạn.
- **close**: Số lượng dấu $)$ sau khi đã xóa hết các phần tử thuộc dãy ngoặc đúng độ dài **optimal** trong đoạn.

Ta tạo 1 kiểu dữ liệu cho 1 nút của cây ST như sau:

```
struct Node {
    int optimal;
    int open;
    int close;

    Node(int opt, int o, int c) { // Khởi tạo struct Node
```

```

        optimal = opt;
        open = o;
        close = c;
    }
};


```

Và ta khai báo cây ST như sau:

```
Node st[MAXN * 4];
```

Định lý

Để tính thông tin ở nút id quản lý đoạn [l,r], dựa trên 2 nút con $2*id$ và $2*id + 1$, ta định nghĩa 1 thao tác kết hợp 2 nút của cây ST:

```

Node operator + (const Node& left, const Node& right) {
    Node res;
    // min(số dấu "(" thừa ra ở cây con trái, và số dấu ")" thừa ra ở cây con phải)
    int tmp = min(left.open, right.close);

    // Để xây dựng kết quả tối ưu ở nút id, ta nối dãy ngoặc tối ưu ở 2 con, rồi thêm
    // min(số "(" thừa ra ở con trái, số ")" thừa ra ở con phải).
    res.optimal = left.optimal + right.optimal + tmp;

    res.open = left.open + right.open - tmp;
    res.close = left.close + right.close - tmp;

    return res;
}

```

Ban đầu ta có thể khởi tạo cây như sau:

```

void build(int id, int l, int r) {
    if (l == r) {
        // Đoạn [l, r] chỉ có 1 phần tử.
        if (s[l] == '(') st[id] = Node(0, 1, 0);
        else st[id] = Node(0, 0, 1);
        return ;
    }
    int mid = (l + r) / 2;
    build(id * 2, l, mid);
    build(id * 2 + 1, mid+1, r);

    st[id] = st[id * 2] + st[id * 2 + 1];
}

```

Để trả lời truy vấn, ta cũng làm tương tự như trong bài toán cơ bản:

```
Node query(int id, int l, int r, int u, int v) {
    if (v < l || r < u) {
        // Trường hợp không giao nhau
        return Node(0, 0, 0);
    }
    if (u <= l && r <= v) {
        // Trường hợp [l, r] nằm hoàn toàn trong [u, v]
        return st[id];
    }

    int mid = (l + r) / 2;
    return query(id * 2, l, mid, u, v) + query(id * 2 + 1, mid+1, r, u, v);
}
```

Ví dụ 2

- Cho một dãy số a_i ($1 \leq a_i \leq 10^9$) có N ($1 \leq N \leq 30,000$) phần tử
- Cho Q ($1 \leq Q \leq 200,000$) truy vấn có dạng 3 số nguyên là li, ri, ki ($1 \leq li \leq ri \leq N, 1 \leq k \leq 10^9$). Yêu cầu của bài toán là đếm số lượng số a_j ($li \leq j \leq ri$) mà $a_j \geq k$.

Giả sử chúng ta có một mảng b với $b_i = 1$ nếu $a_i \geq k$ và bằng 0 nếu ngược lại. Thì chúng ta có thể dễ dàng trả lời truy vấn (i, j, k) bằng cách lấy tổng từ i đến j .

Cách làm của bài này là xử lý các truy vấn theo thứ tự khác, để ta có thể dễ dàng tính được mảng b. Kĩ năng này được gọi là **xử lý offline** (tương tự nếu ta trả lời các truy vấn theo đúng thứ tự trong input, thì được gọi là **xử lý online**):

- Sắp xếp các truy vấn theo thứ tự tăng dần của k .
- Lúc đầu mảng b gồm toàn bộ các số 1.
- Với mỗi truy vấn, ta xem trong mảng a có những phần tử nào lớn hơn giá trị k của truy vấn trước, và nhỏ hơn giá trị k của truy vấn hiện tại, rồi đánh dấu các vị trí đó trên mảng b thành 0. Để làm được việc này một cách hiệu quả, ta cũng cần sắp xếp lại mảng a theo thứ tự tăng dần.

Ta tạo kiểu dữ liệu cho truy vấn:

```
struct Query {
```

```

    int k;
    int l, r;
};

// so sánh 2 truy vấn để dùng vào việc sort.
bool operator < (const Query& a, const Query &b) {
    return a.k < b.k;
}

```

Phản xử lý chính sẽ như sau:

```

vector< Query > queries; // các truy vấn
// Đọc vào các truy vấn
readInput();

// Sắp xếp các truy vấn
sort(queries.begin(), queries.end());

// Khởi tạo mảng id sao cho:
// a[id[1]], a[id[2]], a[id[3]] là mảng a đã sắp xếp tăng dần.

// Khởi tạo Segment Tree

for(Query q : queries) {
    while (a[id[i]] <= q.k) {
        b[id[i]] = 0;
        // Cập nhật cây Segment Tree.
        ++i;
    }
}

```

Vậy ta có thể viết hàm xây dựng cây như sau:

```

void build(int id, int l, int r) {
    if (l == r) {
        // Nút id chỉ gồm 1 phần tử
        st[id] = 1;
        return ;
    }
    int mid = (l + r) / 2;
    build(id * 2, l, mid);
    build(id * 2, mid+1, r);

    st[id] = st[id*2] + st[id*2+1];
}

```

Một hàm cập nhật khi ta muốn gán lại một vị trí bằng 0:

```
void update(int id, int l, int r, int u) {
    if (u < l || r < u) {
        // u nằm ngoài đoạn [l, r]
        return ;
    }
    if (l == r) {
        st[id] = 0;
        return ;
    }
    int mid = (l + r) / 2;
    update(id*2, l, mid, u);
    update(id*2 + 1, mid+1, r, u);

    st[id] = st[id*2] + st[id*2+1];
}
```

Và cuối cùng là thực hiện truy vấn lấy tổng một đoạn:

```
int get(int id, int l, int r, int u, int v) {
    if (v < l || r < u) {
        // Đoạn [l, r] nằm ngoài đoạn [u, v]
        return 0;
    }
    if (u <= l && r <= v) {
        // Đoạn [l, r] nằm hoàn toàn trong đoạn [u, v]
        return st[id];
    }
    int mid = (l + r) / 2;
    return get(id*2, l, mid, u, v)
        + get(id*2+1, mid+1, r, u, v);
}
```

3. Cập nhật lười (Lazy Propagation)

Đây là kĩ thuật được sử dụng trong ST để giảm độ phức tạp của ST với các truy vấn cập nhật đoạn.

Tư tưởng

Giả sử ta cần cập nhật đoạn $[u, v]$. Để thấy ta không thể nào cập nhật tất cả các nút trên Segment Tree (do tổng số nút nằm trong đoạn $[u, v]$ có thể lên đến $O(N)$). Do đó, trong quá trình cập nhật, ta chỉ thay đổi giá trị ở các nút quản lý các đoạn to

nhanh nhất nằm trong $[u,v]$. Ví dụ với $N=7$, cây Segment tree như hình minh họa ở đầu bài. Giả sử bạn cần cập nhật $[1,6]$:

- Bạn chỉ cập nhật giá trị ở các nút quản lý các đoạn $[1,4]$ và $[5,6]$.
- Giá trị của các nút quản lý các đoạn $[1,2], [3,4], [1,1], [2,2], [5,5], \dots$ sẽ không đúng. Ta sẽ chỉ cập nhật lại giá trị của các nút này khi thật sự cần thiết (Do đó kĩ thuật này được gọi là lazy - lười biếng).

Cụ thể, chúng ta cùng xem bài toán sau:

Bài Toán

Cho dãy số A với N phần tử ($N \leq 50,000$). Bạn cần thực hiện 2 loại truy vấn:

- Cộng tất cả các số trong đoạn $[l,r]$ lên giá trị val .
- In ra giá trị lớn nhất của các số trong đoạn $[l,r]$.

Phân tích

Thao tác 2 là thao tác cơ bản trên Segment Tree, đã được ta phân tích ở bài toán đầu tiên.

Với thao tác 1, truy vấn đoạn $[u,v]$. Giả sử ta gọi $F(id)$ là giá trị lớn nhất trong đoạn mà nút id quản lý. Trong lúc cập nhật, muốn hàm này thực hiện với độ phức tạp không quá $O(\log N)$, thì khi đến 1 nút id quản lý đoạn $[l,r]$ với đoạn $[l,r]$ nằm hoàn toàn trong đoạn $[u,v]$, thì ta không được đi vào các nút con của nó nữa (nếu không độ phức tạp sẽ là $O(N)$ do ta đi vào tất cả các nút nằm trong đoạn $[u,v]$). Để giải quyết, ta dùng kĩ thuật Lazy Propagation như sau:

- Lưu $T(id)$ với ý nghĩa, tất cả các phần tử trong đoạn $[l,r]$ mà nút id quản lý đều được cộng thêm $T(id)$.
- Trước khi ta cập nhật hoặc lấy 1 giá trị của 1 nút id' nào đó, ta phải đảm bảo ta đã "đẩy" giá trị của mảng T ở tất cả các nút tổ tiên của id' xuống id' . Để làm được điều này, ở các hàm get và update, trước khi gọi đệ quy xuống các con $2*id$ và $2*id + 1$, ta phải gán:
 - $T[id*2] += T[id]$
 - $T[id*2+1] += T[id]$

- $T[id] = 0$ chú ý ta cần phải thực hiện thao tác này, nếu không mỗi phần tử của dãy sẽ bị cộng nhiều lần, do ta đẩy xuống nhiều lần.

Cài đặt

Ta có kiểu dữ liệu cho 1 nút của ST như sau:

```
struct Node {
    int lazy; // giá trị T trong phân tích trên
    int val; // giá trị lớn nhất.
} nodes[MAXN * 4];
```

Hàm "đẩy" giá trị T xuống các con:

```
void down(int id) {
    int t = nodes[id].lazy;
    nodes[id*2].lazy += t;
    nodes[id*2].val += t;

    nodes[id*2+1].lazy += t;
    nodes[id*2+1].val += t;

    nodes[id].lazy = 0;
}
```

Hàm cập nhật:

```
void update(int id, int l, int r, int u, int v, int val) {
    if (v < l || r < u) {
        return ;
    }
    if (u <= l && r <= v) {
        // Khi cài đặt, ta LUÔN ĐÀM BẢO giá trị của nút được cập nhật ĐỒNG THỜI với
        // giá trị Lazy propagation. Như vậy sẽ tránh sai sót.
        nodes[id].val += val;
        nodes[id].lazy += val;
        return ;
    }
    int mid = (l + r) / 2;

    down(id); // đẩy giá trị Lazy propagation xuống các con

    update(id*2, l, mid, u, v, val);
    update(id*2+1, mid+1, r, u, v, val);

    nodes[id].val = max(nodes[id*2].val, nodes[id*2+1].val);
```

}

Hàm lấy giá trị lớn nhất:

```
int get(int id, int l, int r, int u, int v) {
    if (v < l || r < u) {
        return -INFINITY;
    }
    if (u <= l && r <= v) {
        return nodes[id].val;
    }
    int mid = (l + r) / 2;
    down(id); // đẩy giá trị Lazy propagation xuống các con

    return max(get(id*2, l, mid, u, v),
               get(id*2+1, mid+1, r, u, v));
    // Trong các bài toán tổng quát, giá trị ở nút id có thể bị thay đổi (do ta đẩy Lazy
    // propagation
    // xuống các con). Khi đó, ta cần cập nhật lại thông tin của nút id dựa trên thông tin của
    // các con.
}
```

II. BÀI TẬP ÁP DỤNG

1. Bài 1: Hiệu lớn nhất

1.1. Đề bài

Cho dãy số nguyên a_1, a_2, \dots, a_n và số nguyên dương k . Thực hiện phép xóa k phần tử, sau đó sắp xếp các phần tử theo thứ tự tăng dần, gọi W là hiệu lớn nhất giữa hai phần tử liên tiếp.

Yêu cầu: Tìm cách xóa để W nhận giá trị nhỏ nhất.

Input

- Dòng đầu chứa hai số nguyên dương n, k ($k \leq n - 2$);
- Dòng thứ hai chứa n số nguyên a_1, a_2, \dots, a_n ($|a_i| \leq 109$);

Output

- Gồm một dòng chứa một số là giá trị W nhỏ nhất tìm được.

MAXDIF.INP	MAXDIF.OUT
5 1 4 1 2 3 9	1

Subtask 1: $n \leq 100$;

Subtask 2: $n \leq 2000$;

Subtask 3: $n \leq 10^5$;

1.2. Hướng dẫn giải thuật

Ta có n phần tử, suy ra có $n - 1$ khoảng cách giữa hai phần tử liên tiếp nhau.

Khi xóa đi k phần tử, ta còn lại $n - k$ phần tử, suy ra có $n - k - 1$ khoảng cách giữa hai số liên tiếp nhau.

Gọi a_i là khoảng cách giữa hai phần tử i và $i + 1$ ($i: 1 \rightarrow n-1$).

Vậy bài toán trở thành: Chọn đoạn gồm $n - k - 1$ phần tử liên tiếp trong $n - 1$ phần tử trong mảng a sao cho max của $n - k - 1$ phần tử là nhỏ nhất.

Dùng **Segment Tree** để tìm phần tử lớn nhất trong đoạn gồm m phần tử liên tiếp

```
void use_ST()
{
    n--; // có n phần tử => có n-1 khoảng cách giữa 2 phần tử liên tiếp
    build(1, 1, n); // xây dựng ST, mỗi nút lưu giá trị lớn nhất của đoạn
    for (long i=m; i<=n; i++) // thử từng đoạn từ i-m+1 đến i
    {
        long long h=gett(1, 1, n, i-m+1, i); // tìm phần tử lớn nhất trong đoạn
        kq=min(kq, h); // cập nhật kết quả
    }
    cout << kq;
}
```

1.3. Cài đặt

```
#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
long n, i, day, top, m;
long long kq, k;
long long a[100013], it[500013];
```

```

pair <long long,long> s[100013];

void build(long k,long l,long r)
{
    if (l==r)
    {
        it[k]=a[r];
        return;
    }
    long m=(l+r)/2;
    build(k*2,l,m);
    build(k*2+1,m+1,r);
    it[k]=max(it[k*2],it[k*2+1]);
}

long long gett(long k,long l,long r,long u,long v)
{
    if (u>r || v<l) return 0;
    if (u<=l && v>=r) return it[k];
    long m=(l+r)/2;
    return max(gett(k*2,l,m,u,v),gett(k*2+1,m+1,r,u,v));
}

int main()
{
    freopen("maxdif.inp","r",stdin);
    freopen("maxdif.out","w",stdout);
    cin >> n >> k;
    for (i=1; i<=n; i++)
        cin >> a[i];
    sort(a+1,a+1+n);
    for (i=1; i<n; i++)
        a[i]=a[i+1]-a[i];
}

```

```

m=n-k-1; //xoa k phan tu thi co n - k - 1 khoang cach
n--;
build(1, 1, n); //cay ST, moi nut luu gia tri max cua doan
kq=2234567890;

for (i=m; i<=n; i++)
{
    k=gett(1, 1, n, i-m+1, i); //xet doan n-k-1 phan tu
    kq=min(kq, k);
}
cout << kq;
}

```

1.4. Test

<https://drive.google.com/drive/folders/196EylfcWGwkpH8qKJjr-S2w7bn-VUXZ0?usp=sharing>

2. Bài 2: Khoảng cách

2.1. Đề bài

Cho n điểm trên mặt phẳng, điểm thứ i có tọa độ (x_i, y_i) . Định nghĩa khoảng cách giữa điểm thứ i với điểm thứ j là **MIN** ($|x_i - x_j|, |y_i - y_j|$). Xét tất cả các cặp điểm, tạo ra dãy gồm $n \times (n-1)/2$ giá trị là khoảng cách tất cả các cặp điểm, sắp xếp các khoảng cách theo thứ tự tăng dần, hãy xác định giá trị thứ k .

Input

- Dòng đầu chứa hai số nguyên n, k ;
- Tiếp theo là n dòng, dòng thứ i chứa hai số nguyên không âm x_i, y_i ($x_i, y_i \leq 10^5$).

Output

- Gồm một dòng chứa một số là giá trị thứ k tìm được.

DIST.INP	DIST.OUT
4 2	0

0 0	
1 0	
0 1	
1 1	

SubTask 1: $n \leq 1000$;

SubTask 2: $n = p \times q \leq 10^5$, các điểm lần lượt nằm trên lưới điểm gồm p hàng q cột.

SubTask 3: $n \leq 10^5$.

2.2. Hướng dẫn giải thuật

Subtask1: $N \leq 1000$

Duyệt qua tất cả các điểm

```
For(i,1,n)
```

```
    For(j,1,n)
```

Tính khoảng cách giữa điểm i và điểm j , lưu vào mảng KQ

Sắp xếp lại mảng KQ và in ra phần tử thứ k

SubTask2 và SubTask3

Chặt nhị phân tìm D ($1..10^5$) nhỏ nhất mà số lượng cặp điểm có khoảng cách bé hơn hoặc bằng D là lớn hơn hoặc bằng k;

Xét ví dụ

6 8

$X = 1 3 4 6 7 8$ (sắp xếp tăng dần)

$Y = 4 7 5 2 3 9$ (chạy theo x)

Khoảng cách được sx tăng dần như sau:

1 1 1 1 1 1 2 2 2 2 3

Giả sử D = 3, tìm được cặp (i,j) đầu tiên là (1,3)

$Res += j - i + Get(Y_j - D, Y_j + D)$ //truy vấn trên cây ST

Dùng Cây Segment Tree: mỗi nút chứa số lần xuất hiện của y_i , ban đầu tất cả các nút đều bằng 0;

Trước khi tăng i để tìm cặp (i,j) thỏa mãn, cập nhật y_i vào cây ST (nút lá y_i được cập nhật thành 1);

```
long long Cnt(int D)
{
    int d=1;  long long res=0;
    for (int c=1;c<=n;c++) //đếm số (d,c) thỏa mãn
    {
        while (a[c].x-a[d].x>D) //nếu không tìm thấy
        {
            UPDATE(1, 1, 1e5, a[d].y); //cập nhật y_d vào cây ST
            d++;
        }
        if (a[c].x-a[d].x<=D) res+= (c-d) + Get(1, 1, 1e5,
a[c].y-D, a[c].y+D); //tìm tần suất cặp (d,c) → cộng số cặp vào kq
    }
    return res;
}

int l=0, r=100000, m;
while (l<=r) //chặt nhị phân tìm D nhỏ nhất thỏa mãn
{
    RESET(1, 1, 1e5);
    m = (l+r)/2;
    if (Cnt(m)>=k) r=m-1;
    else l=m+1;
}
cout<<l; //khoảng cách ở vị trí thứ k
```

2.3. Cài đặt

```
#include <bits/stdc++.h>
```

```

#define x first
#define y second
using namespace std;
int n, IT[400001];
long long k;
pair < int , int > a[100001];

void RESET(int id, int l, int r)
{
    IT[id]=0;
    if (l==r) return;
    int m = (l+r)/2;
    RESET(id*2, l, m);
    RESET(id*2+1, m+1, r);
}

void UPDATE(int id, int l, int r, int i)
{
    if (r<i || i<l) return;
    if (l==i && i==r)
    {
        IT[id]++;
        return;
    }
    int m = (l+r)/2;
    UPDATE(id*2, l, m, i);
    UPDATE(id*2+1, m+1, r, i);
    IT[id] = IT[id*2] + IT[id*2+1];
}

int Get(int id, int l, int r, int u, int v)
{
    if (r<u || v<l) return 0;
    if (u<=l && r<=v) return IT[id];
    int m = (l+r)/2;

```

```

        return Get(id*2, l, m, u, v) + Get(id*2+1, m+1, r, u, v);
    }

long long Cnt(int D)
{
    int d=1;  long long res=0;
    for (int c=1;c<=n;c++)
    {
        while (a[c].x-a[d].x>D)
        {
            UPDATE(1, 1, 1e5, a[d].y);
            d++;
        }
        if (a[c].x-a[d].x<=D) res+= (c-d) + Get(1, 1, 1e5,
a[c].y-D, a[c].y+D);
    }
    return res;
}

int main()
{
    #ifndef ONLINE_JUDGE
    freopen("DIST.inp","r",stdin);
    freopen("DIST.out","w",stdout);
    #endif
    ios_base::sync_with_stdio(0); cin.tie(0);
    cin>>n>>k;
    for (int i=1;i<=n;i++) cin>>a[i].x>>a[i].y;
    sort(a+1, a+1+n);
    int l=0, r=100000, m;
    while (l<=r)
    {
        RESET(1, 1, 1e5);
        m = (l+r)/2;
        if (Cnt(m)>=k) r=m-1;

```

```

        else l=m+1;
    }
    cout<<l;
}

```

2.4. Test

<https://drive.google.com/drive/folders/196EylfcWGwkpH8qKJjr-S2w7bn-VUXZ0?usp=sharing>

3. Bài 3: Xếp nhóm

3.1. Đề bài

Tham dự Đại hội thể thao quốc tế, có n đoàn được mời tham gia. Các đoàn được đánh số hiệu từ 1 đến n , biết s_i là số người trong đoàn thứ i ($i = 1, 2, \dots, n$). Trong buổi giao lưu giữa các đoàn, Ban tổ chức lên kế hoạch tổ chức một trò chơi. Trò chơi cần nhiều nhóm tham gia, mỗi nhóm có đúng k người và không có nhóm nào có hai người cùng đoàn. Chú ý là có thể có người không được xếp vào bất cứ nhóm nào. Ban đầu, chỉ có R đoàn có số hiệu $1, 2, \dots, R$ tham gia. Trò chơi rất thú vị nên sau mỗi một lượt chơi, các đoàn có số hiệu $R + 1, R + 2, \dots, n$ lần lượt đăng ký tham gia. Đề trò chơi thêm phần hấp dẫn, mỗi khi có đoàn mới đăng ký tham gia Ban tổ chức muốn xếp lại các nhóm để có nhiều nhóm nhất mà mỗi nhóm có đúng k người và không có nhóm nào có hai người cùng đoàn.

Yêu cầu: Cho s_1, s_2, \dots, s_n và R , hãy giúp Ban tổ chức tính số nhóm nhiều nhất có thể xếp được sau mỗi lượt các nhóm đăng ký tham gia.

Input

Gồm T bộ dữ liệu, mỗi bộ theo khuôn dạng sau:

- Dòng thứ nhất gồm ba số nguyên n, k, R ($n \leq 10^5; k \leq 100$);
- Dòng thứ hai chứa n số nguyên dương s_i ($1 \leq s_i \leq 10^9, i = 1, 2, \dots, n$);

Output

Gồm T dòng, mỗi dòng gồm $n - R$ số nguyên, số thứ j là số nhóm tối đa xếp được khi đoàn $R + j$ đăng kí tham gia.

Ví dụ:

GROUP.INP	GROUP.OUT
5 4 4	5
4 4 4 4 4	

3.2. Hướng dẫn giải thuật

Tính số nhóm có thể chia được:

Tính tổng dòn của các a_i , chặt nhị phân ($0..tổng$) tìm số nhóm có thể chia được;

Xét số nhóm là x , duyệt dãy số từ $1..n$:

- Nếu $a_i < x \rightarrow S = S + a_i$
- Nếu $a_i \geq x \rightarrow S = S + x$

Số nhóm x là khả thi khi $S \geq K*x$

Áp dụng:

Ban đầu chia tất cả n đoàn, xem được bao nhiêu nhóm;

Sắp xếp n đoàn theo số người tăng dần để giảm độ phức tạp;

Xây dựng cây ST , mỗi nút quản lý hai thuộc tính:

- Tổng số người;
- Số đoàn tham gia;

Các nút lá chứa hai thuộc tính ($a_i, 1$), lưu ý a_i được sắp tăng dần.

Cập nhật nút cha như sau: cộng tổng lần lượt của hai thuộc tính;

```
void build(long k, long l, long r)
{
    if (l==r)
    {
        it[k].fi=a[r].fi;
        it[k].se=1;
        return;
    }
```

```

    }

    long m=(l+r)/2;
    build(k*2,l,m);
    build(k*2+1,m+1,r);
    it[k]=cong(it[k*2],it[k*2+1]);
}

```

Sau đó, bỏ đi đoàn thứ n, cập nhật lại cây **ST**, xem thử chia được bao nhiêu nhóm;

Sau đó, bỏ đi đoàn thứ n-1, cập nhật lại cây **ST**, xem thử chia được bao nhiêu nhóm

Tiếp tục, cho đến khi bỏ đi đoàn thứ $n - R + 1$, tính số nhóm chia được.

3.3. Cài đặt

```

#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
typedef pair<long long, long long> ii;
long long i,n,h,q,sl;
long long k;
long long b[100013],vt[100013];
ii it[500013],a[100013];
vector<long long> kq;

ii cong(ii a,ii b)
{
    return make_pair(a.fi+b.fi,a.se+b.se);
}

void build(long k,long l,long r)
{
    if (l==r)
    {
        it[k].fi=a[r].fi;
    }
}

```

```

        it[k].se=1;
        return;
    }

    long m=(l+r)/2;
    build(k*2,l,m);
    build(k*2+1,m+1,r);
    it[k]=cong(it[k*2],it[k*2+1]);
}

void update(long k,long l,long r,long i)
{
    if (l>i || r<i) return;
    if (l==r)
    {
        it[k].fi=0;
        it[k].se=0;
        return;
    }

    long m=(l+r)/2;
    update(k*2,l,m,i);
    update(k*2+1,m+1,r,i);
    it[k]=cong(it[k*2],it[k*2+1]);
}

ii gett(long k,long l,long r,long u,long v)
{
    if (u>r || v<l) return make_pair(0,0);
    if (u<=l && v>=r) return it[k];
    long m=(l+r)/2;
    ii xgett(k*2,l,m,u,v);
    ii ygett(k*2+1,m+1,r,u,v);
    return cong(x,y);
}

```

```

bool check(long long x)
{
    long i=upper_bound(b+1,b+1+n,x)-b;
    i--;
    ii tmp=gett(1,1,n,1,i);
    long long t=tmp.fi+(sl-tmp.se)*x;
    if (t>=k*x) return true;
    return false;
}

void np()
{
    long long l=0;
    long long r=2000000000000000;
    while (l<=r)
    {
        long long m=(l+r)/2;
        if (check(m)) l=m+1; else r=m-1;
    }
    kq.push_back(r);
}

int main()
{
    freopen("group.in","r",stdin);
    freopen("group.out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin >> q;
    while (q--)
    {
        cin >> n >> k >> h;
        for (i=1; i<=n; i++)

```

```

{
    cin >> a[i].fi;
    a[i].se=i;
}
sort(a+1,a+1+n);
for (i=1; i<=n; i++)
{
    vt[a[i].se]=i;
    b[i]=a[i].fi;
}
build(1,1,n);
kq.clear();
for (i=n; i>h; i--)
{
    sl=i;
    np();
    update(1,1,n,vt[i]);
}
for (i=kq.size()-1; i>=0; i--)
    cout << kq[i] << " ";
cout << endl;
}
return 0;
}

```

3.4. Test

<https://drive.google.com/drive/folders/196EylfcWGwkpH8qKJjr-S2w7bn-VUXZ0?usp=sharing>

4. Bài 4: Khớp dữ liệu

4.1. Đề bài

Dãy số nguyên không âm (a_1, a_2, \dots, a_n) được gọi là khớp với dãy số nguyên không âm (b_1, b_2, \dots, b_n) qua chuẩn M nếu $a_i \% M = b_i \% M$ với mọi $i = 1, 2, \dots, n$, trong đó $\%$ là phép chia lấy dư.

Với hai dãy số nguyên không âm, việc tìm chuẩn M đối với Hoàng không phải là công việc khó, Hoàng còn muốn tìm chuẩn M lớn nhất một cách hiệu quả.

Yêu cầu: Cho hai dãy số nguyên không âm $(a_1, a_2, \dots, a_n), (b_1, b_2, \dots, b_n)$ và k cặp chỉ số (L_j, R_j) với $1 \leq L_j \leq R_j \leq n, j = 1, 2, \dots, k$. Với mỗi cặp chỉ số (L_j, R_j) , hãy tìm số nguyên dương M_j lớn nhất là chuẩn của hai dãy $(a_{L_j}, a_{L_{j+1}}, \dots, a_{R_j})$ và $(b_{L_j}, b_{L_{j+1}}, \dots, b_{R_j})$.

Dữ liệu: Vào từ file văn bản **seq.inp** có định dạng:

- Dòng đầu chứa số hai số nguyên dương n, k ($n \leq 10^5$);
- Dòng thứ hai gồm n số nguyên không âm a_1, a_2, \dots, a_n ;
- Dòng thứ ba gồm n số nguyên không âm b_1, b_2, \dots, b_n ($b_i \neq a_i$ với $i = 1, 2, \dots, n$);
- Tiếp theo là k dòng, dòng thứ j ($1 \leq j \leq k$) gồm 2 số nguyên dương L_j, R_j với $1 \leq L_j \leq R_j \leq n, j = 1, 2, \dots, k$.

Kết quả: Ghi ra file văn bản **seq.out** gồm k dòng, dòng thứ j là giá trị M_j lớn nhất là chuẩn của hai dãy $(a_{L_j}, a_{L_{j+1}}, \dots, a_{R_j})$ và $(b_{L_j}, b_{L_{j+1}}, \dots, b_{R_j})$.

Chú ý:

- Có 30% số test có $k = 1$ và các giá trị a_i, b_i không vượt quá 10^3 ;
- Có 50% số test khác có $k \leq 10$ và các giá trị a_i, b_i không vượt quá 10^9 ;
- Có 20% số test còn lại có $k \leq 10^5$ và các giá trị a_i, b_i không vượt quá 10^{15} .

SEQ.INP	SEQ.OUT
3 3	3
1 3 10	4
10 15 2	1

1 2	
2 3	
1 3	

4.2. Hướng dẫn giải thuật

SubTask1: duyệt đơn giản

SubTask2 và SubTask3:

Quy hai dãy số về thành một dãy:

```
for (long i=1; i<=n; i++)
    a[i]=abs(a[i]-b[i]);
```

Xây dựng cấu trúc cây **ST**, mỗi nút chứa UCLN của hai nút con, nút lá chứa giá trị của chính nó.

```
void build(long k, long l, long r)
{
    if (l==r)
    {
        it[k]=a[r];
        return;
    }
    long m=(l+r)/2;
    build(k*2, l, m);
    build(k*2+1, m+1, r);
}
```

```

        build(k*2+1,m+1,r);

        it[k]=__gcd(it[k*2],it[k*2+1]);

    }

```

Dựa vào cấu trúc cây ST, tìm UCLN trên từng đoạn nhờ hàm **Gett**

```

long long gett(long k,long l,long r,long u,long v)
{
    if (u>r || v<l) return 0;

    if (u<=l && v>=r) return it[k];

    long m=(l+r)/2;

    return __gcd(gett(k*2,l,m,u,v),gett(k*2+1,m+1,r,u,v));
}

```

4.3. Cài đặt

```

#include <bits/stdc++.h>
using namespace std;
long n,k,i;
long long a[100013],b[100013],it[500013],f[2013];

void sub1()
{
    long l,r;
    cin >> l >> r;
    for (long i=l; i<=r; i++)
    {
        long x=a[i];

```

```

        long y=b[i];
        for (long j=1; j<=1000; j++)
            if (x%j==y%j) f[j]++;
    }

    for (long j=1000; j>=1; j--)
        if (f[j]==r-l+1)
    {
        cout << j;
        break;
    }
}

void build(long k,long l,long r)
{
    if (l==r)
    {
        it[k]=a[r];
        return;
    }

    long m=(l+r)/2;
    build(k*2,l,m);
    build(k*2+1,m+1,r);
    it[k]=__gcd(it[k*2],it[k*2+1]);
}

long long gett(long k,long l,long r,long u,long v)
{
    if (u>r || v<l) return 0;
    if (u<=l && v>=r) return it[k];
    long m=(l+r)/2;
    return __gcd(gett(k*2,l,m,u,v),gett(k*2+1,m+1,r,u,v));
}

void sub3()

```

```

{
    for (long i=1; i<=n; i++)
        a[i]=abs(a[i]-b[i]);
    build(1,1,n);
    long l,r;
    while (k--)
    {
        cin >> l >> r;
        cout << gett(1,1,n,l,r) << endl;
    }
}

int main()
{
    freopen("seq.inp","r",stdin);
    freopen("seq.out","w",stdout);
    cin >> n >> k;
    for (i=1; i<=n; i++)
        cin >> a[i];
    for (i=1; i<=n; i++)
        cin >> b[i];
    if (k==1)
    {
        sub1();
        return 0;
    }
    sub3();
}

```

4.4. Test

<https://drive.google.com/drive/folders/196EylfcWGwkpH8qKJjr-S2w7bn-VUXZ0?usp=sharing>

5. Bài 5: Hoán đổi

5.1. Đề bài

Trên dãy số nguyên dương a_1, a_2, \dots, a_n , xét thao tác đổi chỗ hai phần tử kế nhau. Cho số nguyên không âm k , hãy sử dụng không quá k thao tác đổi chỗ để đưa dãy a_1, a_2, \dots, a_n về dãy có thứ tự từ điển lớn nhất.

Input

- Dòng đầu chứa hai số nguyên n, k ;
- Dòng thứ hai gồm n số nguyên dương a_1, a_2, \dots, a_n ($a_i \leq 10^9$).

Output

- Gồm một dòng, chứa n số nguyên là dãy nhận được sau khi đổi chỗ.

SWAP.INP	SWAP.OUT
3 2 1 2 3	3 1 2

Subtask 1: $n \leq 1000; k = 1$;

Subtask 2: $n \leq 1000; k \leq 10^6$;

Subtask 3: $n \leq 10^5; k \leq 10^9$;

5.2. Hướng dẫn giải thuật

Sub1: Duyệt dãy số từ đầu tới cuối, nếu tìm thấy $a_{i+1} > a_i$ thì đổi chỗ 2 phần tử đó, đếm số lần đổi cho đến khi bằng k thì dừng.

Sub2: Xét dãy các phần tử từ phần tử thứ i tới phần tử thứ $i+k$, tìm phần tử lớn nhất trong dãy đó, tiến hành đổi chỗ để đưa a_{max} về vị trí thứ i .

```
void sub12 ()  
{  
    long vt,ma;  
    for (long i=1; i<=n; i++)  
    {  
        if (k<=0) break;  
        ma=-1;  
        vt=0;  
        for (long j=i; j<=min(i+k,n); j++)  
            if (a[j]>ma)  
            {  
                ma=a[j];  
            }  
    }  
}
```

```

        vt=j;
    }
    for (long j=vt; j>i; j--)
    {
        swap(a[j],a[j-1]);
        k--;
    }
}
for (long i=1; i<=n; i++)
    cout << a[i] << " ";
}

```

Sub3: Tương tự *sub2*, nhưng thay vì đưa phần tử *max* trong đoạn từ *i* đến *i+k* về vị trí *i* thì ta đẩy phần tử *max* vào hàng đợi, xóa phần tử *max* ra khỏi dãy và cập nhật lại cây ST.

Xây dựng cây ST, mỗi nút lưu hai thuộc tính (giá trị, vị trí của giá trị đó trong dãy ban đầu).

```

void build(long k, long l, long r)
{
    if (l==r)
    {
        it[k].vt=r;
        it[k].sl=1;
        return;
    }
    long m=(l+r)/2;
    build(k*2,l,m);
    build(k*2+1,m+1,r);
    if (a[it[k*2].vt]>=a[it[k*2+1].vt])
        it[k].vt=it[k*2].vt;
    else it[k].vt=it[k*2+1].vt;
    it[k].sl=it[k*2].sl+it[k*2+1].sl;
}

```

Có hai thủ tục cơ bản:

- Cập nhật giá trị Max trong đoạn (l,r);

```

void update(long k, long l, long r, long i)
{
    if (i<l || i>r) return;
    if (l==r)
    {
        it[k].vt=0;
        it[k].sl=0;
    }
}

```

```

        return;
    }
    long m=(l+r)/2;
    update(k*2,l,m,i);
    update(k*2+1,m+1,r,i);
    if (a[it[k*2].vt]>=a[it[k*2+1].vt])
        it[k].vt=it[k*2].vt;
    else it[k].vt=it[k*2+1].vt;
    it[k].sl=it[k*2].sl+it[k*2+1].sl;
}

```

- Truy vấn giá trị Max trong đoạn (l,r).

```

long gett(long k,long l,long r,long u,long v)
{
    if (u>r || v<l || l>r || u>v) return 0;
    if (u<=l && v>=r) return it[k].vt;
    long m=(l+r)/2;
    long x=gett(k*2,l,m,u,v);
    long y=gett(k*2+1,m+1,r,u,v);
    if (a[x]>=a[y]) return x;
    return y;
}

```

5.3. Cài đặt

```

#include <bits/stdc++.h>
using namespace std;
struct asd
{
    long vt, sl;
};
long n, k, i, m, x, d, j;
long a[100013];
asd it[500013];

void sub12()
{
    long vt, ma;
    for (long i=1; i<=n; i++)
    {
        if (k<=0) break;
        ma=-1;
        vt=0;
        for (long j=i; j<=min(i+k,n); j++)
            if (a[j]>ma)
            {
                ma=a[j];
                vt=j;
            }
    }
}

```

```

        for (long j=vt; j>i; j--)
        {
            swap(a[j],a[j-1]);
            k--;
        }
    }
    for (long i=1; i<=n; i++)
        cout << a[i] << " ";
}

long timvt(long k,long l,long r,long len)
{
    if (l==r) return r;
    long m=(l+r)/2;
    if (it[k*2].sl>=len) return timvt(k*2,l,m,len);
    return timvt(k*2+1,m+1,r,len-it[k*2].sl);
}

void build(long k,long l,long r)
{
    if (l==r)
    {
        it[k].vt=r;
        it[k].sl=1;
        return;
    }
    long m=(l+r)/2;
    build(k*2,l,m);
    build(k*2+1,m+1,r);
    if (a[it[k*2].vt]>=a[it[k*2+1].vt])
        it[k].vt=it[k*2].vt;
    else it[k].vt=it[k*2+1].vt;
    it[k].sl=it[k*2].sl+it[k*2+1].sl;
}

void update(long k,long l,long r,long i)
{
    if (i<l || i>r) return;
    if (l==r)
    {
        it[k].vt=0;
        it[k].sl=0;
        return;
    }
    long m=(l+r)/2;
    update(k*2,l,m,i);
    update(k*2+1,m+1,r,i);
    if (a[it[k*2].vt]>=a[it[k*2+1].vt])
        it[k].vt=it[k*2].vt;
}

```

```

        else it[k].vt=it[k*2+1].vt;
        it[k].sl=it[k*2].sl+it[k*2+1].sl;
    }

long gett(long k,long l,long r,long u,long v)
{
    if (u>r || v<l || l>r || u>v) return 0;
    if (u<=l && v>=r) return it[k].vt;
    long m=(l+r)/2;
    long x=gett(k*2,l,m,u,v);
    long y=gett(k*2+1,m+1,r,u,v);
    if (a[x]>=a[y]) return x;
    return y;
}

long cp(long k,long l,long r,long u,long v)
{
    if (u>r || v<l || l>r || u>v) return 0;
    if (u<=l && v>=r) return it[k].sl;
    long m=(l+r)/2;
    return cp(k*2,l,m,u,v)+cp(k*2+1,m+1,r,u,v);
}

void sub3()
{
    vector <long> b;
    bool f[100013];
    long dem=0;
    for (long i=1; i<=n; i++)
    {
        cin >> a[i];
        f[i]=false;
    }
    build(1,1,n);
    long i=1;
    while (k>0 && i<=n)
    {
        j=timvt(1,1,n,k+1);
        dgett(1,1,n,1,j);
        b.push_back(d);
        f[d]=true;
        k-=cp(1,1,n,1,d-1);
        update(1,1,n,d);
        i++;
    }
    for (i=0; i<b.size(); i++)
        cout << a[b[i]] << " ";
    for (i=1; i<=n; i++)
        if (f[i]==false) cout << a[i] << " ";
}

```

```

}

int main()
{
    freopen("swap.inp", "r", stdin);
    freopen("swap.out", "w", stdout);
    ios_base::sync_with_stdio(0);
    cin >> n >> k;
    for (i=1; i<=n; i++)
        cin >> a[i];
    /*if (n<=1000)
    {
        sub12();
        return 0;
    }*/
    sub3();
    return 0;
}

```

5.4. Test

<https://drive.google.com/drive/folders/196EylfcWGwkpH8qKJjr-S2w7bn-VUXZ0?usp=sharing>

6. Bài 6: Đếm tập con chẵn

Nguồn bài: <http://lqdoj.edu.vn/problem/subset>

6.1. Đề bài

Cho dãy số a_1, a_2, \dots, a_n và q thao tác thuộc 1 trong 2 loại sau:

1 k b: gán $a_k = b$

2 l r: Tính số lượng tập con $\{k_1, k_2, \dots, k_m\}$ khác rỗng của tập $\{l, l+1, \dots, r\}$ sao cho $a_{k_1} + a_{k_2} + \dots + a_{k_m}$ là số chẵn.

Yêu cầu: Với thao tác loại 2, hãy in ra số dư của kết quả khi chia cho $10^9 + 7$.

Dữ liệu

- Dòng đầu chứa 2 số nguyên n, q.
- Dòng tiếp theo chứa n số a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$).
- q dòng tiếp theo mỗi dòng chứa một trong hai loại thao tác :
 - **1 k b** ($1 \leq k \leq n; 1 \leq b \leq 10^9$)
 - **2 l r** ($1 \leq l \leq r \leq n$)

Kết quả: Với mỗi thao tác loại 2, in ra số lượng tập con thỏa mãn ($\text{mod } 10^9 + 7$) trên một dòng.

Ví dụ

Input	Output
5 6	7
2 4 6 8 10	1
2 1 3	3
2 4 4	0
2 4 5	1
1 4 7	
2 4 4	
2 4 5	

Giới hạn:

- 20% số test có $n, q \leq 20$.
- 20% số test có $n, q \leq 5000$.
- 20% số test có $n \leq 10^5, q \leq 100$.
- 40% số test có $n, q \leq 10^5$.

6.2. Hướng dẫn giải thuật

Tập trung vào truy vấn loại 2:

Ta có dãy [2 4 6 8 10]

Truy vấn 2 1 3 $\rightarrow \{2, 4, 6\} \rightarrow$ các tập con là: $\{\}, \{2\}, \{4\}, \{6\}, \{2, 4\}, \{2, 6\}, \{4, 6\}, \{2, 4, 6\}$

Truy vấn loại 2: $\{a_1, a_2, \dots, a_n\}$. Hỏi có bao nhiêu tập con có tổng là chẵn?

Giả sử có a số chẵn và b số lẻ trong tập hợp.

Sub1,2: QHĐ

Gọi $d[a][b]$ là số tập con có tổng chẵn nếu có a số chẵn và b số lẻ

Gọi $e[a][b]$ là số tập con có tổng lẻ nếu có a số chẵn và b số lẻ

Xét số cuối:

- Nếu số này lẻ: $d[a][b] = d[a][b-1] + e[a][b-1]$
 $= d[a][b-1] + (2^{a+b} - 1 - d[a][b-1]); //$ tổng số tập con = tổng chẵn + tổng lẻ
 $= 2^{a+b} - 1; //$ trừ tập rỗng
- Nếu số này chẵn: $d[a][b] = d[a-1][b] + d[a-1][b] = 2 * d[a-1][b]$

ĐPT là $O(n^2)$.

Từ công thức QHĐ ở trên, ta thấy: Giả sử có a số chẵn và b số lẻ trong dãy đang xét thì đáp số là:

- $2^a * 2^{b-1} - 1 = 2^{a+b-1} - 1 = 2^{n-1} - 1$ với $b \geq 1 //$ có tồn tại số lẻ
- $2^n - 1$ nếu $b = 0 //$ không có số lẻ

Ví dụ: $a = 5, b = 3$: Đáp số là $2^7 - 1$

Vậy ta cần đếm số số lẻ trong đoạn từ $[a_1 \dots a_r] \rightarrow$ chỉ quan tâm là có số lẻ hay không?

Sub3,4:

Cách 1: Dùng CTDL **SET** trong C++

Xài **set** lưu tất cả vị trí là số lẻ.

$[1 \ 2 \ 1 \ 4 \ 7] \rightarrow \text{Set} = \{1, 3, 5\}$

Thao tác 2: l r

Kiểm tra xem có số x nào thuộc S mà $l \leq x \leq r$

--> Dùng `s.lower_bound(l);`

`if (s.lower_bound(l) != s.end()): // có tồn tại x không?`

`int x = *s.lower_bound(l);`

`if (x <= r) : có số lẻ trong [l, r]`

ĐPT $O(n \log n)$

Cách 2: dùng CTDL Segment Tree hoặc Fenwick Tree

Lưu một dãy nhị phân: $b = 00110$ với $b[i] = 0$ nếu $a[i]$ chẵn, và $= 1$ nếu ngược lại
Tính số số lẻ trong đoạn $[l, r] = \text{query}(r) - \text{query}(l - 1)$

Thao tác 1: $a[x] = y$

$b[x] = y \% 2$ hay $b[x] += (y \% 2 - b[x])$

ĐPT $O(\log n)$ nhanh hơn cách 1

6.3. Cài đặt

Ở đây xin trình bày cài đặt sử dụng CTDL Segment Tree

```
#include<bits/stdc++.h>
using namespace std;
const long long oo=1e9+7;
int arr[100014];
struct node{
    int counte=0;
    int counto=0;
};
node *tree=new node[5*100014];
long long power(long long x,long n)
{
    if (n==1) return x;
    if (n==0) return 1;
    long long k=power(x,n/2)%oo;
    if (n%2==0) return (k*k)%oo;
    else return (k*k*x)%oo;
}
void buildTree(int start,int end,int tn){
if(start==end){
if(arr[start]%2==0){
tree[tn].counte=1;
tree[tn].counto=0;
}else{
tree[tn].counto=1;
tree[tn].counte=0;
}
return;
}
int mid = (start+end)/2;
buildTree(start,mid,tn*2);
buildTree(mid+1,end,tn*2+1);

tree[tn].counte = tree[tn*2].counte+tree[tn*2+1].counte;
tree[tn].counto = tree[tn*2].counto+tree[tn*2+1].counto;
return;
}
void update(int start,int end,int tn,int id,int val){
if(start == end){
if(val%2==0)
tree[tn].counte=1;
else
tree[tn].counto=1;
}
```

```

arr[id] = val;
if(arr[id]%2==0) {
tree[tn].counte=1;
tree[tn].counto=0;
} else{
tree[tn].counto=1;
tree[tn].counte=0;
}
return;
}
int mid = (start + end)/2;
if(id>mid) {
update(mid+1,end,2*tn+1,id,val);
}
else{
update(start,mid,2*tn,id,val);
}
tree[tn].counte = tree[tn*2].counte+tree[tn*2+1].counte;
tree[tn].counto = tree[tn*2].counto+tree[tn*2+1].counto;
return;

}
node query(int start,int end,int tn,int l,int r){
node result;
result.counte=0;
result.counto=0;
if(start>r||end<l){
return result;
}
if(start>=l && end<=r) {
return tree[tn];
}
int mid = (start+end)/2;
if(l>mid) {
return query(mid+1,end,tn*2+1,l,r);
} if(r<=mid) {
return query(start,mid,tn*2,l,r);
}

node left = query(start,mid,tn*2,l,r);
node right = query(mid+1,end,tn*2+1,l,r);

result.counte = left.counte+right.counte;
result.counto = left.counto+right.counto;

return result;

}
int main() {
int n,q,ch,x,y;
cin>>n>>q;
for(int i=1;i<=n;i++) {

```

```

    cin>>arr[i];
}
buildTree(1,n,1);
for(int i=1;i<=q;i++) {
    cin>>ch>>x>>y;
    if(ch==1) {
        update(1,n,1,x,y);
    }
    else{
        node ans = query(1,n,1,x,y);
        if(ch==2){
            if (ans.counto==0) cout<<power(2,y-x+1)-1<<endl;
            else cout<<power(2,y-x)-1<<endl;
        }
    }
}
}

```

6.4. Test

<https://drive.google.com/drive/folders/196EylfcWGwkpH8qKJjr-S2w7bn-VUXZ0?usp=sharing>

III. MỘT SỐ BÀI TẬP LUYỆN THÊM

<https://vn.spoj.com/problems/NKLINEUP/>

<https://www.spoj.com/problems/KQUERY/>

<https://www.spoj.com/problems/GSS3/>

<https://codeforces.com/contest/356/problem/A> [mức độ dễ]

<https://codeforces.com/contest/380/problem/C>

<https://codeforces.com/contest/446/problem/C> [cập nhật lùi]

<https://codeforces.com/contest/501/problem/D>

<https://codeforces.com/problemset/problem/339/D>

<https://codeforces.com/contest/540/problem/E>

<https://codeforces.com/contest/609/problem/F>

<https://codeforces.com/contest/474/problem/F>

https://oj.uz/problem/view/COCI17_deda

C. PHẦN KẾT LUẬN

Trong chuyên đề, tôi đã trình bày về khái niệm cấu trúc dữ liệu cây phân đoạn, cách xây dựng cây phân đoạn và ứng dụng để giải một số bài toán trong Tin học giúp giảm độ phức tạp của bài toán.

Sau khi áp dụng cấu trúc dữ liệu cây phân đoạn vào một số bài toán học sinh giỏi, đặc biệt là các bài toán xử lí trên dãy số, tôi thấy nó mang lại hiệu quả rất rõ rệt. Bây giờ chúng ta đã có một phương pháp đơn giản, dễ dàng hơn để thực hiện. Do thời gian còn hạn chế và kiến thức còn chưa được sâu, rộng nên chắc chắn chuyên đề còn nhiều thiếu sót. Tôi rất mong quý thầy cô đồng nghiệp đóng góp ý kiến để chuyên đề được hoàn thiện hơn.

Hội An, ngày 2 tháng 9 năm 2020

Tác giả

VŨ THỊ MAI PHƯƠNG

SĐT : 0935219405

D. TÀI LIỆU THAM KHẢO

1. Tài liệu giáo khoa chuyên Tin tập 1, 2, 3.
2. Website: <https://vn.spoj.com/>
3. Website: <https://codeforces.com/>
4. Website: <http://lqdoj.edu.vn/>
5. Website: <https://vnoi.info/wiki/algo/data-structures/segment-tree-extend>
6. Website: https://cp-algorithms.com/data_structures/segment_tree.html