**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](#_Toc49002074)

[**A. PHẦN MỞ ĐẦU** 4](#_Toc49002075)

[**B. PHẦN NỘI DUNG** 5](#_Toc49002076)

[**I. CẤU TRÚC DỮ LIỆU CÂY PHÂN ĐOẠN (SEGMENT TREE)** 5](#_Toc49002077)

[**1. Giới thiệu** 5](#_Toc49002078)

[**2. Segment tree cổ điển** 9](#_Toc49002079)

[**3. Cập nhật lười (Lazy Propagation)** 13](#_Toc49002080)

[**II. BÀI TẬP ÁP DỤNG** 16](#_Toc49002081)

[**1. Bài 1: Hiệu lớn nhất** 16](#_Toc49002082)

[**1.1. Đề bài** 16](#_Toc49002083)

[**1.2. Hướng dẫn giải thuật** 17](#_Toc49002084)

[**1.3. Cài đặt** 17](#_Toc49002085)

[**1.4. Test** 19](#_Toc49002086)

[2. Bài 2: Khoảng cách 19](#_Toc49002087)

[2.1. Đề bài 19](#_Toc49002088)

[2.2. Hướng dẫn giải thuật 20](#_Toc49002089)

[2.3. Cài đặt 22](#_Toc49002090)

[2.4. Test 24](#_Toc49002091)

[3. Bài 3: Xếp nhóm 24](#_Toc49002092)

[3.1. Đề bài 24](#_Toc49002093)

[3.2. Hướng dẫn giải thuật 25](#_Toc49002094)

[3.3. Cài đặt 26](#_Toc49002095)

[3.4. Test 30](#_Toc49002096)

[4. Bài 4: Khớp dữ liệu 30](#_Toc49002097)

[4.1. Đề bài 30](#_Toc49002098)

[4.2. Hướng dẫn giải thuật 31](#_Toc49002099)

[4.3. Cài đặt 32](#_Toc49002100)

[4.4. Test 35](#_Toc49002101)

[5. Bài 5: Hoán đổi 35](#_Toc49002102)

[5.1. Đề bài 35](#_Toc49002103)

[5.2. Hướng dẫn giải thuật 35](#_Toc49002104)

[5.3. Cài đặt 37](#_Toc49002105)

[5.4. Test 40](#_Toc49002106)

[**6. Bài 6: Đếm tập con chẵn** 40](#_Toc49002107)

[**6.1. Đề bài** 40](#_Toc49002108)

[**6.2. Hướng dẫn giải thuật** 41](#_Toc49002109)

[**6.3. Cài đặt** 43](#_Toc49002110)

[**6.4. Test** 45](#_Toc49002111)

[III. MỘT SỐ BÀI TẬP LUYỆN THÊM 45](#_Toc49002112)

[**C. PHẦN KẾT LUẬN** 47](#_Toc49002113)

[**D. TÀI LIỆU THAM KHẢO** 48](#_Toc49002114)

# **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 (Binarry 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)**

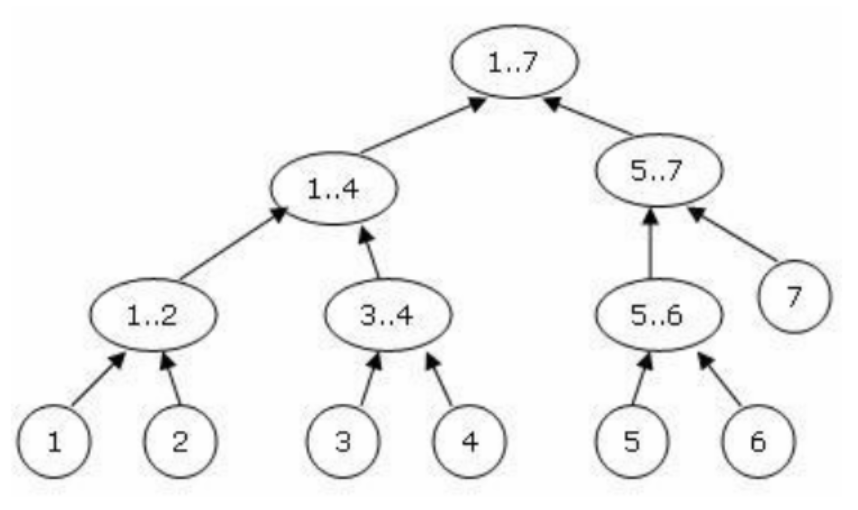
## **1. Giới thiệu**

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,⌊N/2⌋] và nút con phải sẽ lưu thông tin về đoạn [⌊N/2⌋+1,N]. Tổng quát hơn: nếu nút A lưu thông tin đoạn [i,j], thì 2 con của nó: A1 và A2 sẽ lưu thông tin của các đoạn [i,⌊(i+j)/2⌋] và đoạn [⌊(i+j)/2⌋+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 ≤ 105). Ban đầu mỗi phần tử có giá trị 0.
* Có Q truy vấn (Q ≤ 105). 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 đọan [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(logN). 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(logN).
* 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:

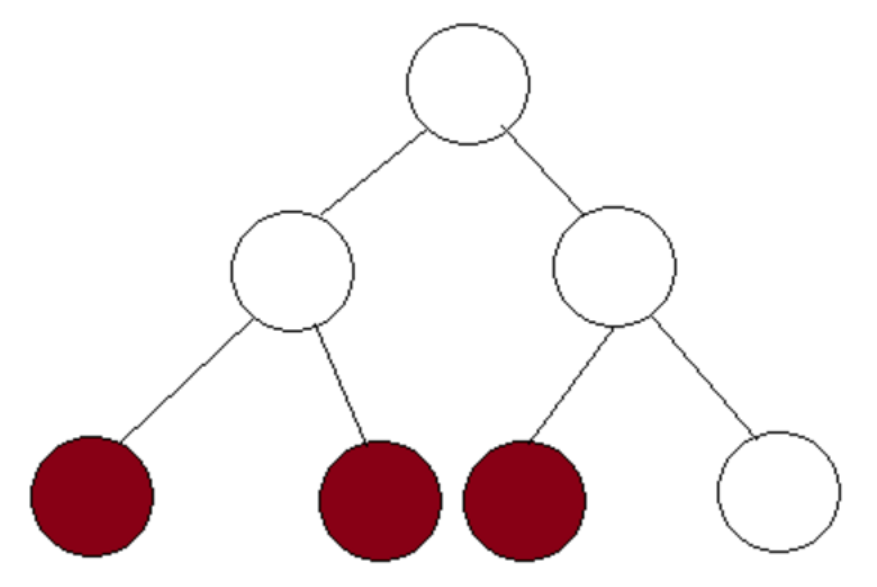
* Phần tử cần xét không nằm trong đoạn [l,r] do nút id quản lý. Trường hợp này ta dừng lại, không xét tiếp.
* Phần tử cần xét nằm trong đoạn [l,r] do nút id quản lý. Ta xét các con của nút id. Tuy nhiên chỉ có 1 con của nút id chứa phần tử cần xét và ta sẽ phải xét tiếp các con của nút này. Với con còn 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(logN).

**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 = 2k: Cây ST đầy đủ, ở độ sâu cuối cùng có đúng 2k 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 2k nút
* Tầng k−1: có 2k−1 nút
* ... Tổng số nút không quá 2k+1.
* Với N > 2k và N < 2k+1. Số nút của cây ST không quá số nút của cây ST với N=2k+1.

Do đó, số nút của cây cho dãy N phần tử, với N ≤ 2k là không quá 2k+1, giá trị này xấp xỉ 4∗N. Bằng thực nghiệm, ta thấy dùng 4∗N là đủ.

## **2. Segment tree cổ điển**

Tại sao lại gọi là cổ 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* ≤ 106), cho *M* truy vấn có dạng *li,ri (1 ≤ li ≤ ri ≤ N)*. Yêu cầu của bài toán là với mỗi truy vấn tìm một chuỗi con (không cần liên tiếp) của chuỗi từ *li* đến *ri* 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 *ai (1 ≤ ai ≤ 109)* có N (1 ≤ N ≤ 30,000) phần tử
* Cho *Q (1 ≤ Q ≤ 200,000)* truy vấn có dạng 3 số nguyên là *li, ri, ki (1 ≤ li ≤ ri ≤ N, 1 ≤ k ≤ 109)*. Yêu cầu của bài toán là đếm số lượng số *aj (li ≤ j ≤ ri)* mà aj ≥ k.

Giả sử chúng ta có một mảng b với bi=1 nếu ai>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 một 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;

}

Phần 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]. Dễ 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 nhất nằm trong [u,v]. Ví dụ với N=7, cây Segment tree như hình minh hoạ 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], ... 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≤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(logN), 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 ĐẢM 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 𝑎1,𝑎2,…,𝑎𝑛 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 ≤ 𝑛 − 2);
  + Dòng thứ hai chứa 𝑛 số nguyên 𝑎1,𝑎2,…,𝑎𝑛 (|𝑎𝑖| ≤ 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 ≤ 100;

**Subtask 2**: 𝑛 ≤ 2000;

**Subtask 3**: 𝑛 ≤ 105;

### **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 *ai* là khoảng cách giữa hai phần tử *i* và *i* + 1 (*i*: 1 🡪 *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--; //xet tren mang khoang cach a con lai n-1 phan tu  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 (𝑥𝑖,𝑦𝑖). Định nghĩa khoảng cách giữa điểm thứ 𝑖 với điểm thứ 𝑗 là 𝐌𝐈𝐍 (|𝑥𝑖 − 𝑥𝑗|,|𝑦𝑖 − 𝑦𝑗|). Xét tất cả các cặp điểm, tạo ra dãy gồm 𝑛×(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 𝑥𝑖,𝑦𝑖 (𝑥𝑖, 𝑦𝑖 ≤ 105).

**Output**

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

|  |  |
| --- | --- |
| DIST.INP | DIST.OUT |
| 4 2  0 0  1 0  0 1  1 1 | 0 |

**SubTask 1**: 𝑛 ≤ 1000;

**SubTask 2**: 𝑛 = 𝑝 × 𝑞 ≤ 105, các điểm lần lượt nằm trên lưới điểm gồm 𝑝 hàng 𝑞 cột.

**SubTask 3**: 𝑛 ≤ 105.

### 2.2. Hướng dẫn giải thuật

**Subtask1**: N ≤ 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..105) 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 2 3 ….

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

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

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

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

|  |
| --- |
| long long Cnt(int D)  {  int d=1; long long res=0;  for (int c=1;c<=n;c++)//duyệt tìm cặp chỉ 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 yd 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 tìm thấy 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 𝑠𝑖 là số người trong đoàn thứ 𝑖 (𝑖 = 1,2,…,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,…,𝑅 tham gia. Trò chơi rất thú vị nên sau mỗi một lượt chơi, các đoàn có số hiệu 𝑅 + 1,𝑅 + 2,…,𝑛 lần lượt đăng kí tham gia. Để trò chơi thêm phần hấp dẫn, mỗi khi có đoàn mới đăng kí 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 𝑠1,𝑠2,…,𝑠𝑛 và 𝑅, hãy giúp Ban tổ chức tính số nhóm nhiều nhất có thể xếp được sau mỗi lượt các nhóm đăng kí 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 ≤ 105; 𝑘 ≤ 100);
* Dòng thứ hai chứa 𝑛 số nguyên dương 𝑠𝑖 (1 ≤ 𝑠𝑖 ≤ 109 ,𝑖 = 1,2,…,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  4 4 4 4 4 | 5 |

### 3.2. Hướng dẫn giải thuật

**Tính số nhóm có thể chia được**:

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

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

* Nếu ai < x 🡪 S=S + ai
* Nếu ai ≥ x 🡪 S=S + x

Số nhóm x là khả thi khi S ≥ 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 (ai, 1), lưu ý *ai* 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 x=gett(k\*2,l,m,u,v);  ii y=gett(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=200000000000000;  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 (𝑎1,𝑎2,…,𝑎𝑛) được gọi là khớp với dãy số nguyên không âm (𝑏1,𝑏2,…,𝑏𝑛) qua chuẩn 𝑀 nếu 𝑎𝑖% 𝑀 = 𝑏𝑖 % 𝑀 với mọi 𝑖 = 1,2,…,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 (𝑎1,𝑎2,…,𝑎𝑛), (𝑏1,𝑏2,…,𝑏𝑛) và 𝑘 cặp chỉ số (𝐿𝑗,𝑅𝑗) với 1 ≤ 𝐿𝑗 ≤ 𝑅𝑗 ≤ 𝑛, 𝑗 = 1,2,…,𝑘. Với mỗi cặp chỉ số (𝐿𝑗,𝑅𝑗), hãy tìm số nguyên dương 𝑀𝑗 lớn nhất là chuẩn của hai dãy (𝑎𝐿𝑗,𝑎𝐿𝑗+1,…,𝑎𝑅𝑗) và (𝑏𝐿𝑗,𝑏𝐿𝑗+1,…,𝑏𝑅𝑗).

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

* Dòng đầu chứa số hai số nguyên dương 𝑛,𝑘 (𝑛 ≤ 105);
* Dòng thứ hai gồm 𝑛 số nguyên không âm 𝑎1,𝑎2,…,𝑎𝑛;
* Dòng thứ ba gồm 𝑛 số nguyên không âm 𝑏1,𝑏2,…,𝑏𝑛 (𝑏𝑖 ≠ 𝑎𝑖 với 𝑖 = 1,2,…,𝑛);
* Tiếp theo là 𝑘 dòng, dòng thứ 𝑗 (1 ≤ 𝑗 ≤ 𝑘) gồm 2 số nguyên dương 𝐿𝑗,𝑅𝑗 với 1 ≤ 𝐿𝑖 ≤ 𝑅𝑗 ≤ 𝑛, 𝑗 = 1,2,…,𝑘.

**Kết quả**: Ghi ra file văn bản ***seq.out*** gồm 𝑘 dòng, dòng thứ 𝑗 là giá trị 𝑀𝑗 lớn nhất là chuẩn của hai dãy (𝑎𝐿𝑗,𝑎𝐿𝑗+1,…,𝑎𝑅𝑗) và (𝑏𝐿𝑗,𝑏𝐿𝑗+1,…,𝑏𝑅𝑗).

***Chú ý***:

* Có 30% số test có 𝑘 = 1 và các giá trị 𝑎𝑖,𝑏𝑖 không vượt quá 103;
* Có 50% số test khác có 𝑘 ≤ 10 và các giá trị 𝑎𝑖,𝑏𝑖 không vượt quá 109;
* Có 20% số test còn lại có 𝑘 ≤ 105 và các giá trị 𝑎𝑖,𝑏𝑖 không vượt quá 1015.

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

### 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);  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 𝑎1,𝑎2,…,𝑎𝑛, xét thao tác đổi chỗ hai phần tử kề nhau. Cho số nguyên không âm 𝑘, hãy sử dụng không quá 𝑘 thao tác đổi chỗ để đưa dãy 𝑎1,𝑎2,…,𝑎𝑛 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 𝑎1,𝑎2,…,𝑎𝑛 (𝑎𝑖 ≤ 109).

**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 ≤ 1000;𝑘 = 1;

Subtask 2: 𝑛 ≤ 1000;𝑘 ≤ 106;

Subtask 3: 𝑛 ≤ 105;𝑘 ≤ 109;

### 5.2. Hướng dẫn giải thuật

**Sub1**: Duyệt dãy số từ đầu tới cuối, nếu tìm thấy ai+1 > ai 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 *amax* về 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);  d=gett(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 a1,a2,...,an và q thao tác thuộc 1 trong 2 loại sau:

1 k b: gán ak=b

2 l r: Tính số lương tập con {k1,k2,...,km} khác rỗng của tập {l,l+1,...,r} sao cho ak1+ak2+...+akm là số chẵn.

**Yêu cầu:** Với thao tác loại 2, hãy in ra số dư của kết quả khi chia cho 109+7.

**Dữ liệu**

* Dòng đầu chứa 2 số nguyên n,q.
* Dòng tiếp theo chứa n số a1,a2,...,an(1≤ ai ≤109).
* q dòng tiếp theo mỗi dòng chứa một trong hai loại thao tác :
  + 1 k b (1 ≤ k ≤ n;1 ≤ b ≤ 109)
  + 2 l r (1 ≤ l ≤ r ≤ n)

**Kết quả**: Với mỗi thao tác loại 2, in ra số lượng tập con thỏa mãn (mod 109+7) trên một dòng.

**Ví dụ**

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

**Giới hạn:**

* 20% số test có n, q ≤ 20.
* 20% số test có n, q ≤ 5000.
* 20% số test có n ≤ 105, q ≤ 100.
* 40% số test có n, q ≤ 105.

### **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 -> {2, 4, 6} 🡪 các tập con là: {2}, {4}, {6}, {2, 4}, {2, 6}, {4, 6}, {2, 4, 6}

Truy vấn loại 2: {a1, a2, ..., an}. 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] + (2a+b – 1 – d[a][b-1]); //tổng số tập con = tổng chẵn + tổng lẻ

=2a+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(n2).

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:

* 2a \* 2b - 1 - 1 = 2a + b - 1 - 1 = 2n - 1 - 1 với b >= 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 27 – 1

Vậy ta cần đếm số số lẻ trong đoạn từ [al … ar] 🡪 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] -> Set = {1, 3, 5}

Thao tác 2: l r

Kiểm tra xem có số x nào thuộc S mà l <= x <= 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(nlogn)

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] = query(r) - query(l - 1)

Thao tác 1: a[x] = y

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

ĐPT O(logn) 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){ |
| 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 lười]

<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>