

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



Báo cáo bài tập lớn môn học:  
Phân tích và thiết kế thuật toán

---

NHÂN NHANH ĐA THỨC SỬ DỤNG  
BIẾN ĐỔI FOURIER NHANH

---

*Sinh viên thực hiện:*

Tạ Quang Tùng  
MSSV: 20154280

*Giáo viên hướng dẫn:*

TS. Đỗ Phan Thuận

Hà Nội, Ngày 18 tháng 12 năm 2018

# Mục lục

1	Giới thiệu về biến đổi Fourier nhanh . . . . .	1
2	Phép nhân đa thức và liên hệ với biến đổi Fourier nhanh . . . . .	1
3	Thuật toán biến đổi Fourier nhanh . . . . .	3
4	Bài toán A cộng B version 2 . . . . .	7
4.1	Đề bài . . . . .	7
4.2	Ý tưởng lời giải . . . . .	7
4.3	Code của thuật toán . . . . .	8
4.4	Bộ test và kết quả chạy . . . . .	9
4.5	Kết quả submit online . . . . .	12
5	Bài toán 993E - Nikita and Order Statistics . . . . .	12
5.1	Đề bài . . . . .	12
5.2	Tóm tắt . . . . .	13
5.3	Ý tưởng lời giải . . . . .	13
5.4	Code của thuật toán . . . . .	15
5.5	Bộ test và kết quả chạy . . . . .	16
5.6	Kết quả submit online . . . . .	19

<b>Tài liệu tham khảo</b>	<b>20</b>
---------------------------	-----------

# 1 Giới thiệu về biến đổi Fourier nhanh

Biến đổi Fourier (Fourier Transform) là một trong những công cụ toán học quan trọng bậc nhất với khoa học nói chung và xử lý tín hiệu nói riêng.

Trong toán học có 4 loại phép biến đổi Fourier khác nhau:

- Biến đổi Fourier (liên tục): Là loại biến đổi với tín hiệu liên tục, không tuần hoàn.
- Biến đổi Fourier thời gian rời rạc (Discrete-time Fourier Transform: Tín hiệu là rời rạc, không tuần hoàn, phổ tuần hoàn.
- Chuỗi Fourier (Fourier Series): Tín hiệu là liên tục, tuần hoàn, phổ rời rạc.
- Biến đổi Fourier rời rạc (Discrete Fourier Transform): Tín hiệu là rời rạc, tuần hoàn, phổ rời rạc, tuần hoàn.

Trong đó chỉ có phép biến đổi Fourier rời rạc là thực hiện được trên máy tính vì đầu vào và đầu ra đều là rời rạc. Một trong những thuật toán nhanh nhất để thực hiện phép biến đổi này có tên gọi là thuật toán biến đổi Fourier nhanh (Fast Fourier Transform). Thuật toán FFT ngoài việc được sử dụng rộng rãi trong xử lý tín hiệu số thì còn có thể được sử dụng trong các bài toán nhân nhanh đa thức được trình bày dưới đây.

Không những vậy, nghiên cứu chỉ ra rằng mắt và tai người, động vật có "cài đặt" sẵn thuật toán biến đổi Fourier để giúp chúng ta nhìn và nghe, vì vậy nó được GS Ronald Coifman của đại học Yale gọi là Phương pháp phân tích dữ liệu của tự nhiên ("Nature's way of analyzing data") [1] [4]

# 2 Phép nhân đa thức và liên hệ với biến đổi Fourier nhanh

Xét hai đa thức  $P(x)$ ,  $Q(x)$ , ta có:

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1} = \sum_{i=0}^{m-1} a_i x^i$$

$$Q(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1} = \sum_{j=0}^{n-1} b_j x^j$$

$$P(x) \times Q(x) = c_0 + c_1x + c_2x^2 + \dots + c_{p-1}x^{p-1} = \sum_{k=0}^{p-1} c_k x^k$$

Đồng nhất hệ số, ta được:

$$\Rightarrow c_k = \sum_{i=0}^k a_i \times b_{k-i}$$

$$\text{Với: } i \leq m-1$$

$$\text{Và: } k-i \leq n-1 \Leftrightarrow i \geq k-n+1$$

$$\text{Hay: } c_k = \sum_{i=\max(0, k-n+1)}^{\min(k, m-1)} a_i \times b_{k-i} \quad \forall k = \overline{0, m+n-2}$$

$$\text{Thời gian tính toán: } \mathcal{O}(m \times n)$$

(Do cần phải sử dụng  $m \times n$  phép nhân và số phép cộng nhỏ hơn số phép nhân).

Tuy nhiên, công thức trên khá phức tạp khi phải xét đến sự khác biệt giữa bậc của  $P(x)$  và  $Q(x)$ .

Nếu đặt:

$$p = m + n - 1$$

$$N = 2^{\lceil \log_2(p) \rceil}$$

$$W_u = e^{-j \frac{2\pi u}{N}}$$

Thay  $W_u$  vào hai đã thức đã cho, ta được:

$$P(W_u) = \sum_{i=0}^{m-1} a_i W_u^i$$

$$Q(W_u) = \sum_{j=0}^{n-1} b_j W_u^j$$

$$P(W_u) \times Q(W_u) = \sum_{k=0}^{p-1} c_k W_u^k$$

Đồng thời nếu mở rộng các hệ số của  $P(x)$  và  $Q(x)$  cho đến bậc  $N-1$ , hay:

$$a_i = 0 \quad \forall i = \overline{m, N-1}$$

$$b_j = 0 \quad \forall j = \overline{n, N-1}$$

$$c_k = 0 \quad \forall k = \overline{p, N-1}$$

Ta sẽ được công thức đơn giản hơn như sau:

$$c_k = \sum_{i=0}^k a_i \times b_{k-i} \quad \forall k = \overline{0, N-1}$$

Và đồng thời:

$$\begin{aligned} P(W_u) &= \sum_{i=0}^{N-1} a_i W_u^i = \mathcal{F}[a_i]_u \\ Q(W_u) &= \sum_{i=0}^{N-1} b_i W_u^i = \mathcal{F}[b_i]_u \\ P(W_u) \times Q(W_u) &= \sum_{i=0}^{N-1} c_i W_u^i = \mathcal{F}[c_i]_u \end{aligned}$$

Trong đó  $\mathcal{F}[x_i]_u$  là phép biến đổi Fourier rời rạc của dãy  $x_i$ .  
 Từ đó ta có thể tính được dãy  $c_i$  như sau:

$$\begin{aligned} \Rightarrow c_i &= \mathcal{F}^{-1} [\mathcal{F}[c_i]_u]_i \\ &= \mathcal{F}^{-1} [P(W_u) \times Q(W_u)]_i \\ &= \mathcal{F}^{-1} [\mathcal{F}[a_i]_u \times \mathcal{F}[b_i]_u]_i \end{aligned}$$

Với thuật toán tính biến đổi Fourier nhanh (thuận và ngược), thời gian tính toán  $\mathcal{O}(N \log N)$ , ta có thời gian tổng của thuật toán nhân đa thức sử dụng FFT như sau:

$$\begin{aligned} \mathcal{F}[a_i]_u &\quad \mathcal{O}(N \log N) \\ \mathcal{F}[b_i]_u &\quad \mathcal{O}(N \log N) \\ P(W_u) \times Q(W_u) &\quad \mathcal{O}(N) \\ \mathcal{F}^{-1}[P(W_u) \times Q(W_u)]_i &\quad \mathcal{O}(N \log N) \\ \text{Tổng:} &\quad \mathcal{O}(N \log N) < \mathcal{O}(m \times n) \end{aligned}$$

### 3 Thuật toán biến đổi Fourier nhanh

Xét dãy số  $x_i$  có  $N$  phần tử,  $i = \overline{0, N-1}$ ,  $N = 2^p$ , ta có:

$$\begin{aligned} X_u &= \mathcal{F}[x_i]_u = \sum_{i=0}^{N-1} x_i e^{-j \frac{2\pi i u}{N}} \\ &= \sum_{i=0}^{\frac{N}{2}-1} x_{2i} e^{-j \frac{2\pi 2i u}{N}} + \sum_{i=0}^{\frac{N}{2}-1} x_{2i+1} e^{-j \frac{2\pi (2i+1) u}{N}} \\ &= \sum_{i=0}^{\frac{N}{2}-1} x_{2i} e^{-j \frac{2\pi i u}{N/2}} + e^{-j \frac{2\pi u}{N}} \sum_{i=0}^{\frac{N}{2}-1} x_{2i+1} e^{-j \frac{2\pi i u}{N/2}} \end{aligned}$$

Đặt:  $a_i$  là dãy các số có chỉ số chẵn lấy ra từ  $x_i$ .

$b_i$  là dãy các số có chỉ số lẻ lấy ra từ  $x_i$ .

Ta được:

$$\begin{aligned}\sum_{i=0}^{\frac{N}{2}-1} x_{2i} e^{-j \frac{2\pi i u}{N/2}} &= \sum_{i=0}^{\frac{N}{2}-1} a_i e^{-j \frac{2\pi i u}{N/2}} \\ &= \mathcal{F}[a_i]_u \quad \left( \text{Có } \frac{N}{2} \text{ phần tử} \right) \\ \sum_{i=0}^{\frac{N}{2}-1} x_{2i+1} e^{-j \frac{2\pi i u}{N/2}} &= \sum_{i=0}^{\frac{N}{2}-1} b_i e^{-j \frac{2\pi i u}{N/2}} \\ &= \mathcal{F}[b_i]_u \quad \left( \text{Có } \frac{N}{2} \text{ phần tử} \right)\end{aligned}$$

Nhưng lại có:  $u = \overline{0, N-1}$

Tuy nhiên, ta có thể thấy:

$$\begin{aligned}e^{-j \frac{2\pi i u}{N/2}} &= e^{-j \frac{2\pi i u}{N/2} + j 2\pi i} \\ &= e^{-j \frac{2\pi i (u - N/2)}{N/2}}\end{aligned}$$

Tuần hoàn với chu kỳ  $\frac{N}{2}$

Nên ta chỉ cần tính  $\frac{N}{2}$  giá trị  $u$  cho  $\mathcal{F}[a_i]_u$  và  $\mathcal{F}[b_i]_u$  là đủ. Đây cũng chính là mấu chốt làm cho thuật toán FFT có thể thực hiện được với thời gian  $\mathcal{O}(N \log N)$ .

Từ đó:

Nếu xét với  $u = \overline{0, \frac{N}{2} - 1}$

$$\begin{aligned}\mathcal{F}[x_i]_u &= \mathcal{F}[a_i]_u + e^{-j \frac{2\pi u}{N}} \mathcal{F}[b_i]_u \\ \mathcal{F}[x_i]_{u+N/2} &= \mathcal{F}[a_i]_u + e^{-j \frac{2\pi (u+N/2)}{N}} \mathcal{F}[b_i]_u \\ &= \mathcal{F}[a_i]_u - e^{-j \frac{2\pi u}{N}} \mathcal{F}[b_i]_u\end{aligned}$$

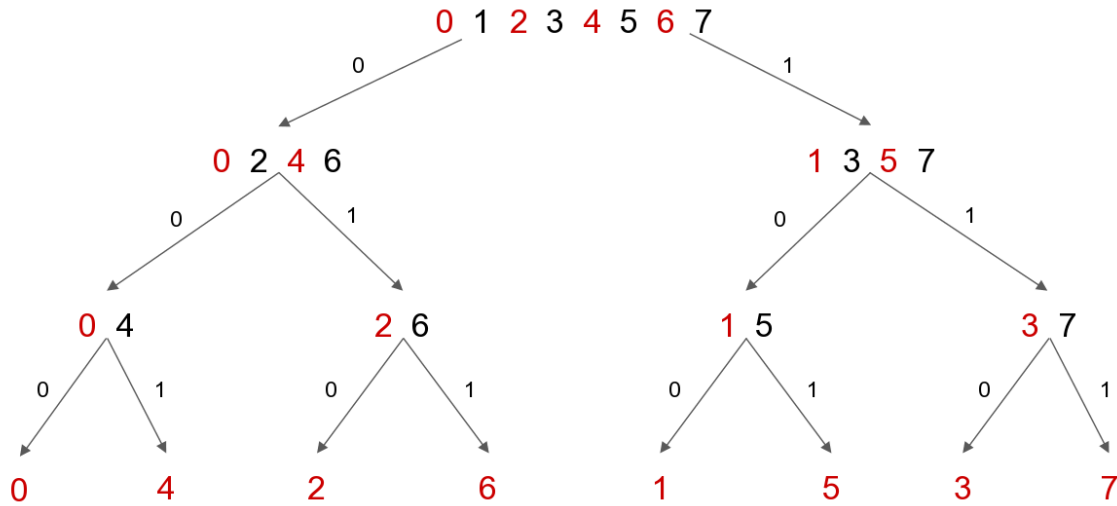
Thời gian tính toán:

Gọi  $T(n)$  là thời gian tính cho giải thuật này, áp dụng định lý thợ, ta được:

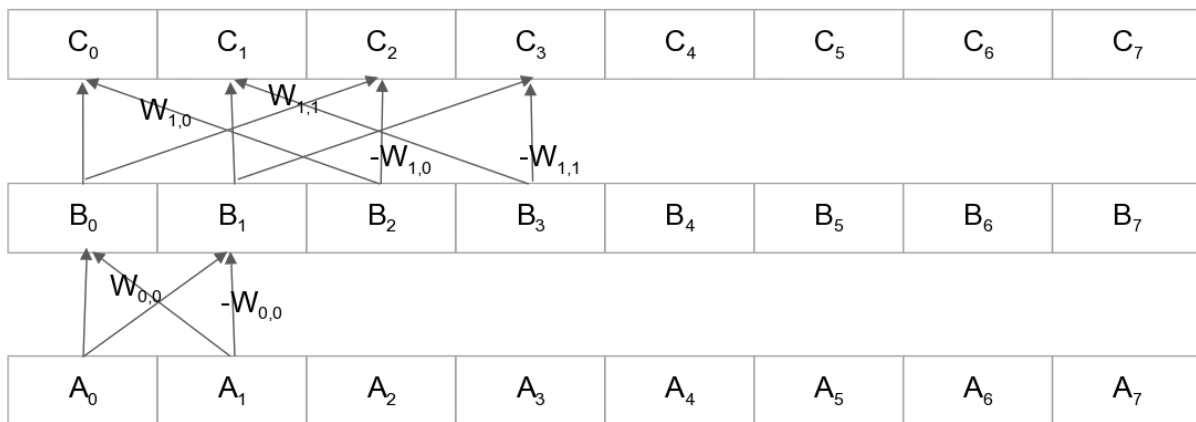
$$\begin{aligned}T(n) &= 2T\left(\frac{n}{2}\right) + \Theta(n) \\ \Rightarrow T(n) &= \Theta(n \log n)\end{aligned}$$

Ta có thể sử dụng đệ quy để implement thuật toán FFT. Tuy nhiên cách implement đó là không hiệu quả, trong các bài toán nhân nhanh đa thức và trong thực tế thì người ta thường sử dụng thuật toán như sau:

- Hoán vị các phần tử sử dụng đảo bit giá trị nhị phân các chỉ số trong mảng. Như được biểu diễn bằng hình sau:



- Thực hiện vòng lặp tính các DFT từ  $n = 2$  và tăng dần  $\times 2$  mỗi vòng lặp, như hình dưới đây:



$$W_{s,u} = e^{-j \frac{2\pi u}{2(1 \ll s)}}$$

$$\mathcal{F}[x_i]_u = \mathcal{F}[a_i]_u + e^{-j \frac{2\pi u}{N}} \mathcal{F}[b_i]_u$$

$$\mathcal{F}[x_i]_{u+N/2} = \mathcal{F}[a_i]_u - e^{-j \frac{2\pi u}{N}} \mathcal{F}[b_i]_u$$

Code của thuật toán FFT:

```

std::bitset<64> reverse(std::bitset<64> set, ll bit_count) {
    for (ll i = 0; i < bit_count / 2; i++) {
        bool tmp = set[i];
        set[i] = set[bit_count - 1 - i];
        set[bit_count - 1 - i] = tmp;
    }
    return set;
}

void init_T(ll bit_count) {
    ll N = 1 << bit_count;
    for (ll i = 0; i < N; i++)
        T[i] = reverse(i, bit_count).to_ullong();
}

void fft(cp *A, ll bit_count, bool inverse = false) {
    static cp W[FFT_MAX / 2];

    ll N = 1 << bit_count;

    for (ll i = 0; i < N; i++) {
        ll j = T[i];
        if (i < j)
            std::swap(A[i], A[j]);
    }

    for (ll step = 1; step < N; step <= 1) {
        double angle = -PI / step;
        if (inverse)
            angle = -angle;

        cp w(1, 0), wn(std::cos(angle), std::sin(angle));
        for (ll i = 0; i < step; i++) {
            W[i] = w;
            w *= wn;
        }

        for (ll start_even = 0; start_even < N; start_even += step * 2) {
            auto start_odd = start_even + step;
            for (ll i = 0; i < step; i++) {
                cp U = A[start_even + i];
                cp V = W[i] * A[start_odd + i];
                A[start_even + i] = U + V;
                A[start_odd + i] = U - V;
            }
        }
    }

    if (inverse) {
        for (ll i = 0; i < N; i++)
            A[i] /= N;
    }
}

```

Trong đó init\_T dùng để khởi tạo mảng chuyển vị T. step là biến biểu thị rằng các giá trị FFT với kích thước step đã được tính. Biến inverse = true khi mà ta muốn tính FFT ngược.



## 4 Bài toán A cộng B version 2

### 4.1 Đề bài

[2]

## POST2 - A cộng B version 2

### Giới hạn

- Thời gian: 0.25s
- Bộ nhớ: 1536MB
- Mã nguồn: 50000 bytes

Cho 3 dãy N số nguyên  $A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_n$  và  $C_1, C_2, \dots, C_n$ . Hãy đếm số bộ 3 ( $A_i, B_j, C_k$ ) thỏa mãn  $A_i + B_j = C_k$ .

### Input

- Dòng đầu ghi số N. (  $N \leq 10^5$  )
- Dòng thứ 2 ghi N số nguyên có giá trị tuyệt đối không vượt quá 50000 thể hiện dãy A.
- Dòng thứ 3 và thứ 4 lần lượt ghi dãy B và C theo quy cách tương tự.

### Output

- Số bộ 3 thỏa mãn.

### 4.2 Ý tưởng lời giải

- Mỗi giá trị của các dãy nằm trong khoảng từ  $-50000$  đến  $50000$   
 $\Rightarrow$  Có tổng cộng 100001 giá trị.
- Mỗi dãy A, B, C có thể đại diện bằng một vector (nguyên) 100001 chiều.
- Gọi U, V, W lần lượt là 3 vector đó.
- $\Rightarrow$  Số bộ 3 thỏa mãn là tổng:

$$N = 2^{\lceil \log_2(4n+1) \rceil}$$

$$R_k = \sum_{i=\max(-n, k-n)}^k U_i \times V_{k-i} \quad \mathcal{O}(N \log N)$$

$$\sum_{k=-n}^n W_k R_k \quad \mathcal{O}(n)$$

$i = \max(-n, k-n)$  do  $k-i \leq n$ . Công thức của  $R_k$  giống công thức của phép nhân đa thức. Trong đó  $n = 50000$ ,  $\Rightarrow 2n$  là bậc của đa thức tương ứng với 3 vector U, V, W. Nên bậc của đa thức ứng với  $R_k$  là  $4n$ .

### 4.3 Code của thuật toán

```
int N;
std::cin >> N;

for (int i = 0; i < N; i++) {
    int x; std::cin >> x;
    A[x + MAX] += 1;
}

for (int i = 0; i < N; i++) {
    int x; std::cin >> x;
    B[x + MAX] += 1;
}

for (int i = 0; i < N; i++) {
    int x; std::cin >> x;
    C[x + 2*MAX] += 1;
}

fft(A, MAX_BIT_COUNT);
fft(B, MAX_BIT_COUNT);

for (ll i = 0; i < FFT_MAX; i++)
    A[i] *= B[i];

fft(A, MAX_BIT_COUNT, true);

ll count = 0;
for (int x = MAX; x <= 3 * MAX; x++)
    count += std::llround(A[x].real()) * C[x];

std::cout << count << std::endl;
```

Trong đó,  $MAX = 50000$ ,  $MAX\_BIT\_COUNT = 18$  do  $\lceil \log_2(4N+1) \rceil = 18$ . Ta thực hiện đọc vào các mảng A, B, C. Cứ mỗi giá trị x, ta tăng giá trị các mảng tại vị trí tương ứng lên 1. Mảng A, B phải cộng  $MAX$  do giá trị nhỏ nhất của x là  $-MAX$ . Mảng C phải cộng  $2 \times MAX$  do A, B

đã cộng  $MAX$  và ta lại muốn cộng các phần tử của A, B và so sánh với C. Thực hiện nhân nhân đa thức sử dụng FFT bằng gọi FFT cho mảng A, B, nhân lần lượt từng phần tử của hai mảng với nhau, và thực hiện FFT ngược để ra được đa thức kết quả.

Ta tính tổng từ  $x = MAX$  đến  $3MAX$  do các giá trị của mảng C chỉ có nghĩa tại các vị trí  $x = MAX$  đến  $3MAX$ .

#### 4.4 Bộ test và kết quả chạy

Test được chạy trên chương trình chấm tự xây dựng, trong đó các test bao gồm:

- 5 test tay (kết quả được tính bằng tay).
- 2 test có quy luật (tổng bằng và tổng không bằng), kích thước 1000 phần tử.
- 2 test có quy luật (tổng bằng và tổng không bằng), kích thước 100000 phần tử.
- 12 test được sinh ngẫu nhiên bằng thuật toán đơn giản (không dùng FFT) để kiểm tra tính đúng đắn, kích thước 1000 phần tử.
- 5 test được sinh ngẫu nhiên bằng chương trình đích để kiểm tra thời gian chạy, kích thước 100000 phần tử.

Kết quả chạy test:

```
% main ../fft.cpp
=====
Test case: 00
PASSED
Time: 0s 121ms
=====
Test case: 01
PASSED
Time: 0s 127ms
=====
Test case: 02
PASSED
Time: 0s 133ms
=====
Test case: 03
PASSED
Time: 0s 125ms
=====
Test case: 04
PASSED
Time: 0s 127ms
=====
Test case: 05
PASSED
Time: 0s 123ms
=====
Test case: 06
PASSED
Time: 0s 121ms
=====
Test case: 07
PASSED
Time: 0s 124ms
=====
Test case: 08
PASSED
Time: 0s 122ms
=====
Test case: 09
PASSED
Time: 0s 146ms
=====
```

## Bộ test chương trình

- 5 test tay (kết quả được tính bằng tay).
- 2 test có quy luật (tổng bằng và không bằng), kích thước 1000 phần tử.
- 2 test có quy luật (tổng bằng và không bằng), kích thước 10000 phần tử.
- 12 test được sinh ngẫu nhiên bằng chương trình đơn giản (không dùng FFT) để kiểm tra tính đúng đắn, kích thước 1000 phần tử.
- 5 test được sinh ngẫu nhiên bằng chương trình đích để kiểm tra tính đúng đắn, kích thước 100000 phần tử.

Nhấp để thêm ghi chú của người thuyết trình

```

=====
Test case: i16 >> N;
PASSED
Time: 0s (121ms = 0; i < N; i++) {
=====
Test case: x17 MAX] += 1;
PASSED
Time: 0s 137ms
=====
Test case: t18; std::cin >> x;
PASSED B[x + MAX] += 1;
Time: 0s 119ms
=====
Test case: n19 i = 0; i < N; i++) {
PASSED int x; std::cin >> x;
Time: 0s (119ms 2*MAX] += 1;
=====
Test case: 20
PASSED t(A, MAX_BIT_COUNT);
Time: 0s (120ms BIT_COUNT);
=====
Test case: l21 = 0; i < FFT_MAX; i++)
PASSED A[i] *= B[i];
Time: 0s 161ms
=====
Test case: 22
PASSED count = 0;
Time: 0s (162ms = MAX; x <= 3 * MAX; x++)
=====
Test case: 23
PASSED d::cout << count << std::endl;
Time: 0s 167ms
=====
Test case: 24
PASSED g -X550CC ~/Desktop/design-algorithms/A-plus-B
Time: 0s 167ms
=====
Test case: x25 0CC ~/Desktop/design-algorithms/A-plus-B
PASSED
Time: 0s (164ms job

```

## 4.5 Kết quả submit online

https://vn.spoj.com/status/ ☆ B

**S**phere online judge Giải bài trực tuyến PROFILE ▾

News

Problems

acm

challenge

main

oi

tutorial

Status

Ranking

Forum

info

rules

links

Biên bản chấm bài

Xem lại 1 2 3 4 5 Xem tiếp >

ID	GIỜ NỘP	Tài khoản:	PROBLEM	RESULT	TIME	MEM	NGÔN NGỮ
22818945	2018-12-04 11:55:08	Tung Quang	A cộng B version 2	accepted edit run	0.09	31M	C++14
22818934	2018-12-04 11:51:51	Ngot	Pha chế	93.33	0.09	17M	C++14
22818927	2018-12-04 11:50:40	Ngot	Pha chế	0	0.00	0k	C++14
22818923	2018-12-04 11:49:55	hoangkings100	Đi xem phim	100	0.00	2.7M	C++ 4.3.2
22818907	2018-12-04 11:47:32	hoangkings100	Đi xem phim	90 WA-test-4	0.00	2.7M	C++ 4.3.2
22818840	2018-12-04	Ngot	Pha chế	6.67	0.02	18M	C++14

## 5 Bài toán 993E - Nikita and Order Statistics

### 5.1 Đề bài

[3]

## E. Nikita and Order Statistics

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

Nikita likes tasks on order statistics, for example, he can easily find the  $k$ -th number in increasing order on a segment of an array. But now Nikita wonders how many segments of an array there are such that a given number  $x$  is the  $k$ -th number in increasing order on this segment. In other words, you should find the number of segments of a given array such that there are exactly  $k$  numbers of this segment which are less than  $x$ .

Nikita wants to get answer for this question for each  $k$  from 0 to  $n$ , where  $n$  is the size of the array.

### Input

The first line contains two integers  $n$  and  $x$  ( $1 \leq n \leq 2 \cdot 10^5$ ,  $-10^9 \leq x \leq 10^9$ ).

The second line contains  $n$  integers  $a_1, a_2, \dots, a_n$  ( $-10^9 \leq a_i \leq 10^9$ ) — the given array.

### Output

Print  $n + 1$  integers, where the  $i$ -th number is the answer for Nikita's question for  $k = i - 1$ .

## 5.2 Tóm tắt

Cho một mảng  $n$  phần tử và một giá trị  $x$ .

Ứng với mỗi giá trị  $k$  từ 0 đến  $n$ , tìm số lượng các đoạn (segment) mà có đúng  $k$  giá trị trong đoạn đó là nhỏ hơn  $x$ .

Điều kiện:

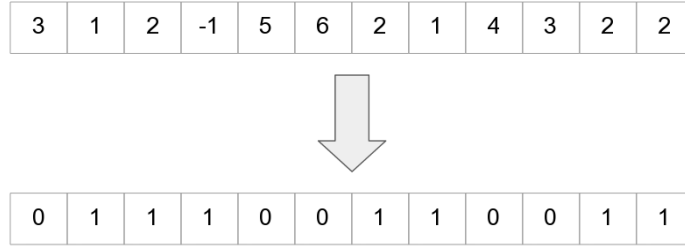
$$\begin{aligned} 1 &\leq n \leq 2 \times 10^5 \\ -10^9 &\leq a_i \leq 10^9 \\ -10^9 &\leq x \leq 10^9 \end{aligned}$$

Giới hạn thời gian: 2s, bộ nhớ: 256MB.

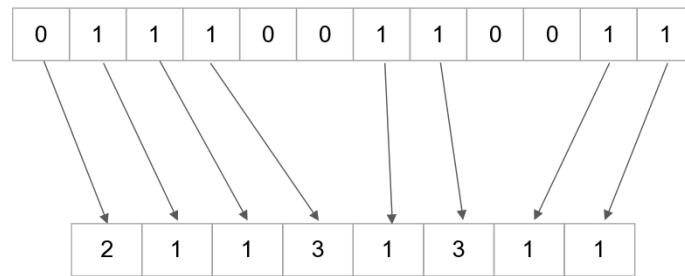
## 5.3 Ý tưởng lời giải

Ý tưởng lời giải:

- Do ta chỉ quan tâm các vị trí trong mảng có nhỏ hơn  $x$  hay không, nên ta có thể chuyển mảng đầu vào thành chuỗi nhị phân. Ví dụ với  $x = 3$ :



- Gọi  $C_j$  là số các số 0 sau phần tử thứ  $j$ , cộng 1. ( $j = \overline{1, m}$ ) và  $C_0$  là số phần tử 0 liên tiếp bắt đầu xâu nhị phân  $M$ , cộng 1.



- Số các segment gồm  $k$  ( $k \geq 1$ ) số 1 bao lấy đoạn ứng với  $C_j$  đến  $C_{j+k-1}$  là:

$$\begin{aligned}
 X_k &= \sum_{j=1}^{m+1-k} C_{j-1} \times C_{j-1+k} \\
 &= \sum_{p=0}^{m-k} C_p \times C_{p+k}
 \end{aligned}$$

- Ta chuyển công thức trên thành nhân đa thức:

Gọi  $D$  là dãy nghịch đảo của  $C$ .

Hay:  $D_i = C_{m-i} \quad \forall i = \overline{0, m}$

Ta có:

$$\begin{aligned}
 C_{p+k} &= D_{m-k-p} \\
 \Rightarrow X_k &= \sum_{p=0}^{m-k} C_p \times C_{p+k} \\
 &= \sum_{p=0}^{m-k} C_p \times D_{m-k-p}
 \end{aligned}$$



Công thức này chính là công thức nhân đa thức, từ đó ta có thể sử dụng FFT để tính nhanh các giá trị  $X_k$ .

## 5.4 Code của thuật toán

```
void init() {
    std::fill(C, C + n + 1, 0);
    C[0] = 1;

    C_len = n + 1;
    bit_count = ceil_log2(C_len * 2);
    N = 1 << bit_count;

    init_T(bit_count);
}

void input() {
    std::cin >> n >> x;

    init();

    ll sum = 0;
    for (ll i = 0; i < n; i++) {
        ll v;
        std::cin >> v;
        sum += v < x;
        C[sum]++;
    }

    for (ll i = 0; i <= sum; i++)
        if (C[i] > 1)
            zero_count += C[i] * (C[i] - 1) / 2;
}

int main() {
    std::ios::sync_with_stdio(false);

    input();
    std::cout << zero_count << " ";
    static cp A[FFT_MAX], B[FFT_MAX];

    std::copy(C, C + C_len, A);
    std::fill(A + C_len, A + N, 0);

    std::fill(B, B + 2 * n - C_len, 0);
    std::reverse_copy(C, C + C_len, B + 2 * n - C_len);
    std::fill(B + 2 * n, B + N, 0);

    fft(A, bit_count);
    fft(B, bit_count);
    for (ll i = 0; i < N; i++)
        A[i] *= B[i];
    fft(A, bit_count, true);
    for (ll k = 1; k <= n; k++)
```

```

    std::cout << (ll)std::round(A[2 * n - k - 1].real()) << " ";
    std::cout << '\n';
    return 0;
}

```

Trong đó `init_T` là hàm để khởi tạo bảng chuyển vị bit. ( $T[i]$  sẽ là số tương ứng với các bit đảo ngược của  $i$ ). Hàm `input` sẽ thực hiện đọc đầu vào và chuyển luôn thành mảng  $C_i$ , không thông qua bước chuyển thành chuỗi nhị phân. Thời gian tính toán  $\mathcal{O}(n)$ .

Thực hiện sao chép giá trị của  $C$  cho vào mảng  $A$ , đảo ngược giá trị của mảng  $C$ , cho vào mảng  $B$ . Thực hiện nhân nhanh đa thức bằng FFT và in kết quả.

Giá trị `zero_count` là số các đoạn mà không chứa số 1 nào phải được tính riêng lẻ.

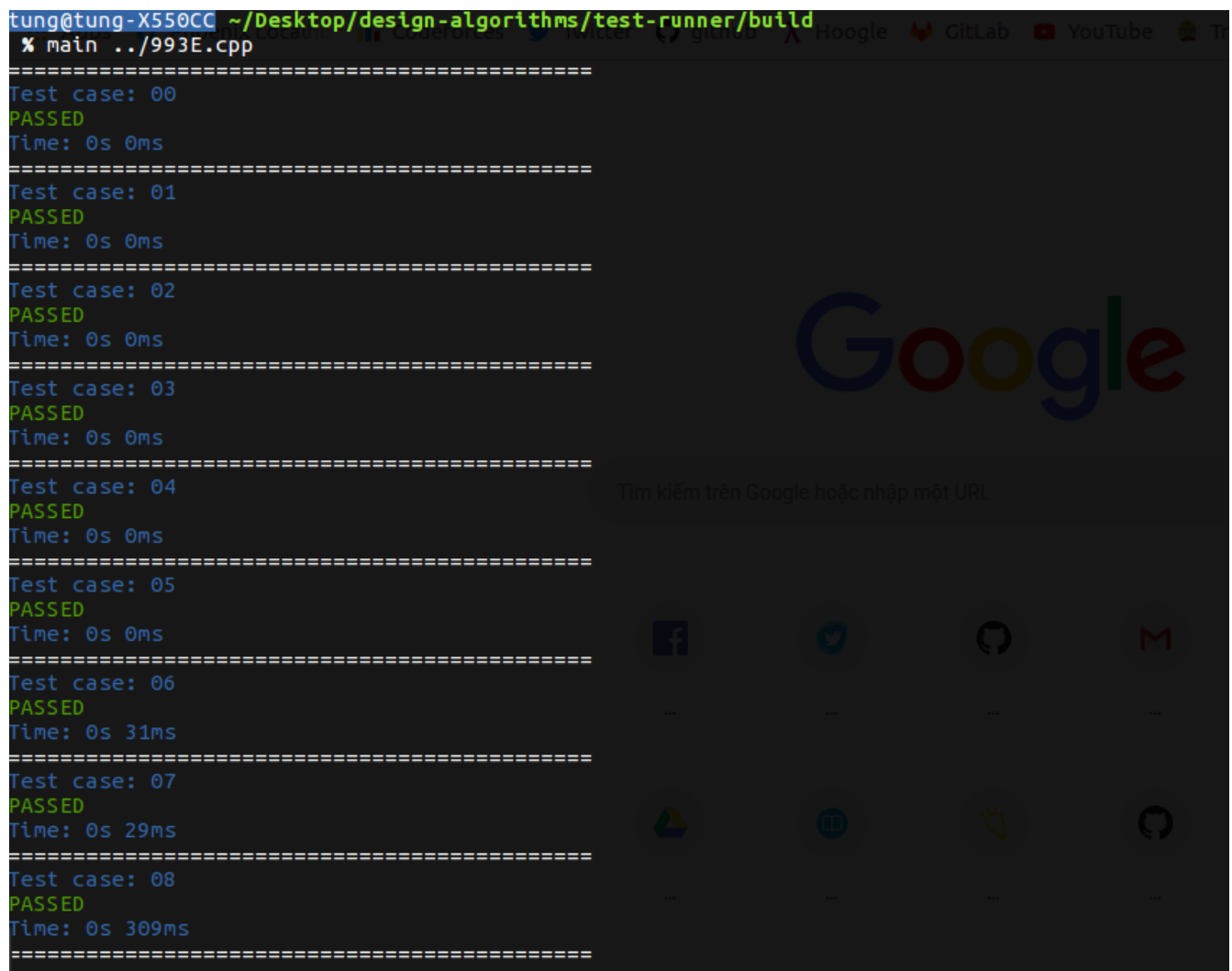
## 5.5 Bộ test và kết quả chạy

Bộ test chương trình:

- 6 test tay (kết quả được tính bằng tay), bao gồm một số test trên Codeforces.
- 2 test có quy luật (đều nhỏ hơn  $x$  và đều lớn hơn  $x$ ), kích thước 20000 phần tử.
- 2 test có quy luật (đều nhỏ hơn  $x$  và đều lớn hơn  $x$ ), kích thước 200000 phần tử.
- 10 test được sinh ngẫu nhiên bằng chương trình đơn giản (không dùng FFT) để kiểm tra tính đúng đắn, kích thước 20000 phần tử.
- 10 test được sinh ngẫu nhiên bằng chương trình đích để kiểm tra thời gian chạy, kích thước 200000 phần tử.

Kết quả chạy test:

```
tung@tung-X550CC ~/Desktop/design-algorithms/test-runner/build
% main ../993E.cpp
=====
Test case: 00
PASSED
Time: 0s 0ms
=====
Test case: 01
PASSED
Time: 0s 0ms
=====
Test case: 02
PASSED
Time: 0s 0ms
=====
Test case: 03
PASSED
Time: 0s 0ms
=====
Test case: 04
PASSED
Time: 0s 0ms
=====
Test case: 05
PASSED
Time: 0s 0ms
=====
Test case: 06
PASSED
Time: 0s 31ms
=====
Test case: 07
PASSED
Time: 0s 29ms
=====
Test case: 08
PASSED
Time: 0s 309ms
=====
```



```
=====
Test case: 20 https://docs.google.com/presentation/d/1JqFEaZkE6Y-NX95J80Fn5E3WJU-Ly0eeQveRJ55kkko/edit#slide=id.g4887858aa8
PASSED
Time: 0s 349ms
=====
Test case: 21 hình sửa Xem Chèn Định dạng Trang trình bày Sắp xếp Công cụ Tiện ích bổ sung Trợ giúp Đã lưu tại cả thay đổi
PASSED
Time: 0s 349ms
=====
Test case: 22
PASSED
Time: 0s 364ms
=====
Test case: 23
PASSED
Time: 0s 373ms
=====
Test case: 24
PASSED
Time: 0s 351ms
=====
Test case: 25
PASSED
Time: 0s 359ms
=====
Test case: 26
PASSED
Time: 0s 376ms
=====
Test case: 27
PASSED
Time: 0s 358ms
=====
Test case: 28
PASSED
Time: 0s 345ms
=====
Test case: 29
PASSED
Time: 0s 361ms
=====
tung@tung-X550CC ~/Desktop/design-algorithms/test-runner/build
```

## 5.6 Kết quả submit online

<a href="#">46203281</a>	2018-11-25 15:47:12	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	296 ms	30500 KB
<a href="#">46203052</a>	2018-11-25 15:40:09	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	296 ms	30500 KB
<a href="#">46202983</a>	2018-11-25 15:37:31	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	592 ms	30500 KB
<a href="#">46202786</a>	2018-11-25 15:30:52	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	686 ms	39900 KB
<a href="#">46202450</a>	2018-11-25 15:17:54	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	701 ms	35200 KB
<a href="#">46202129</a>	2018-11-25 15:06:40	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	982 ms	35200 KB
<a href="#">46201978</a>	2018-11-25 15:01:23	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	888 ms	35200 KB
<a href="#">46201844</a>	2018-11-25 14:56:43	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Wrong answer on test 9	31 ms	18800 KB
<a href="#">46201794</a>	2018-11-25 14:54:38	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Accepted	857 ms	35200 KB
<a href="#">46201721</a>	2018-11-25 14:51:28	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Compilation error	0 ms	0 KB
<a href="#">46198234</a>	2018-11-25 12:39:51	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU C++17	Time limit exceeded on test 12	2000 ms	2400 KB
<a href="#">46198234</a>	2018-11-14	quangtung97	<a href="#">E - Nikita and Order Statistics</a>	GNU			

# Tài liệu tham khảo

- [1] FFT. <http://vnoi.info/wiki/algo/trick/FFT>.
- [2] POST2 - VNOI. <http://vnoi.info/problems/show/POST2>.
- [3] Problem - 993E - Codeforces. <https://codeforces.com/problemset/problem/993/E>.
- [4] Rohit Thummalapalli. Fourier Transform: Nature's Way of Analyzing Data. *Yale Scientific Magazine*, Dec 2010.