CHUYÊN ĐỀ HỘI THẢO TRẠI HÈ HÙNG VƯƠNG

LẦN THỨ XVIII, NĂM 2024

CÂY PHÂN KHÚC - SEGMENT TREE VÀ MỘT SỐ BÀI TẬP ÁP DỤNG

Người thực hiện: Nhóm Tin học

Tổ: Toán – Tin

Năm học: 2023 – 2024

Mã số: .....

MỤC LỤC

A. PHẦN MỞ ĐẦU

I. LÍ DO CHỌN ĐỀ TÀI

Chỉ cần qua câu nói "Algorithms+Data Structures = Program" của Niklaus Wirth ta đã có thể thấy được tầm quan trọng của các loại cấu trúc dữ liệu [data structures] trong giải các bài toán tin. Ứng dụng 1 cách thuần thục hiệu quả các loại cấu trúc sẽ đem đến những thuận lợi vô cùng lớn cho các học sinh và lập trình viên. Mặc dù vậy, tài liệu tiếng việt về những cấu trúc này lại khá ít, đặc biệt là Segment Tree, Binary Indexed Tree và Range minimum Query. Trong chuyên đề này sẽ đề cập tới cấu trúc thường xuyên được sử dụng: Segment Tree – Cây phân đoạn.

Cây phân đoạn - Segment Tree tên gọi chính xác là Segment Tree nhưng cái tên Segment Tree được sử dụng nhiều hơn ở Việt Nam. Là một công cụ rất hữu dụng được sử dụng nhiều trong các bài toán trên dãy số, hoặc được quy về các bài toán xử lý trên dãy số, đặc biệt là các bài toán có nhiều công việc cần xử lí và nhiều truy vấn xen kẻ nhau.

II. MỤC ĐÍCH, NHIỆM VỤ NGHIÊN CỨU

Nhằm làm rõ mức độ hiệu quả khi sử dụng cấu trúc Segment Tree thông qua một số bài tập có thể áp dụng cấu trúc này để giải quyết các bài toán trong chương trình bồi dưỡng học sinh lớp chuyên tin, học sinh giỏi các cấp, khu vực và quốc tế.

III. PHƯƠNG PHÁP NGHIÊN CỨU

- Thu thập tài liệu, nghiên cứu, phân tích.
- Áp dụng thực tế, rút kinh nghiệp và tổng hợp thông tin.

IV. ĐỐI TƯỢNG NGHIÊN CỨU

Cơ sở lý thuyết và bài tập trong các cấp độ lớp chuyên tin, HSG QG, khu vực và quốc tế.

V. ĐÓNG GÓP CỦA ĐỀ TÀI

Xây dựng tài tiệu học tập, nghiên cứu và tham khảo cho học sinh chuyên tin, học sinh thi chọn HSG các cấp và giáo viên say mê tin học lập trình.

B. PHẦN NỘI DUNG

I. CẤU TRÚC DỮ LIỆU CÂY PHÂN KHÚC (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.

Trong chuyên đề này tác giả đề cập tới các bài toán điển hình sau:

+ Cây phân khúc giải bài toán tìm Max của nhiều đoạn liên liếp

+ Cây phân khúc giải bài toán tính tổng của nhiều đoạn liên liếp

+ Cây phân khúc giải bài toán tìm số nhỏ nhất mà lớn hơn bằng k nhiều đoạn liên tiếp

+ Cây phân khúc giải bài toán tìm Max của nhiều đoạn liên liếp. Sử dụng kĩ thuật cập nhật lười để cập nhật giá trị cho nhiều đoạn liên tiếp.

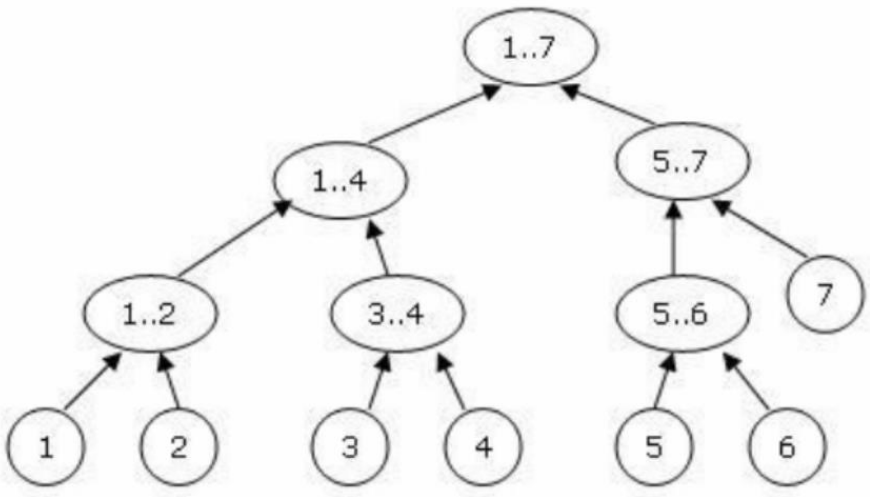
+ Cây phân khúc giải bài toán quy hoạch động tìm Max trên các đoạn liên tiếp độ dài k

Trước tiên, chúng ta cùng tìm hiểu cấu trúc của cây phân khúc:

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: A_1 và A_2 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 ($N \leq 10^5$). 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(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:

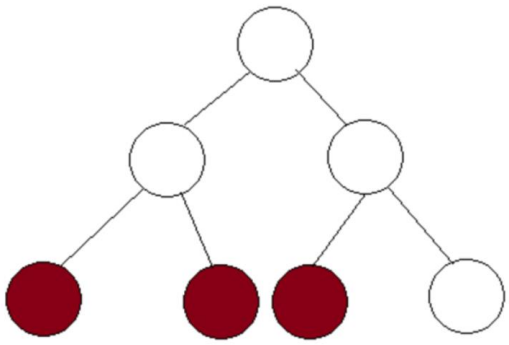
- Phần tử cần xét không nằm trong đoạn [l,r] do nút id quản lý. Trường hợp này ta dừng lại, không xét tiếp.
- Phần tử cần xét nằm trong đoạn [l,r] do nút id quản lý. Ta xét các con của nút id. Tuy nhiên chỉ có 1 con của nút id chứa phần tử cần xét và ta sẽ phải xét tiếp các con của nút này. Với con còn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

- $N = 2^k$: 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 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. Educational Segment Tree Contest - ITEZ1

Hảo Hảo, cảm nhận từng giây hạnh phúc!
Bạn là chủ một hệ thống nhà hàng chuyên bán mì ăn liền. Tại sao lại là mì ăn liền, đó là vì "dẫu thời gian mang đến nhiều đổi thay, những điều tuyệt vời vẫn còn đó trên mảnh đất này. Hảo Hảo, cảm nhận từng giây hạnh phúc".

Khi xem các quảng cáo mì ăn liền, ta đều thấy cả nhà cùng ngồi ăn mì và cười hạnh phúc, hay là người mẹ đi làm và mua mì về thì cả nhà cùng nhau reo hò. Mì thường được làm từ "trứng vàng" (≤15g/1kg) hay khoai tây (≈20g/1kg) rất tốt cho sức khỏe, trong khi đó thành phần sắn thì chỉ là thành phần phụ (≈900g/1kg). Nhà hàng của bạn có n gói mì đủ loại, mỗi gói có độ ngon là a_i($|a_i|\leq 10^9$). Bạn cần xử lý 2 loại truy vấn như sau:

- Loại 1 có dạng 1 x y: Nấu gói mì ở vị trí thứ x và mua gói mì có độ ngon y thay vào đó. ($1\leq x\leq n, |y|\leq 10^9$)
- Loại 2 có dạng 2 l r: In ra độ ngon lớn nhất của các gói mì từ vị trí l đến vị trí r ($1\leq l\leq r\leq n$)

Với mỗi truy vấn loại 2, hãy in ra câu trả lời trên một dòng.

Input

- Dòng đầu tiên là số n là số các gói mì ($1\leq n\leq 10^5$)
- Dòng 2 là n số nguyên là độ ngon của các gói mì ($|a_i|\leq 10^9$)
- Dòng 3 là 1 số q là số truy vấn($1\leq q\leq 10^5$)
- q dòng sau, mỗi dòng là một truy vấn thuộc một trong hai loại trên.

Output: Với mỗi truy vấn loại 2, in ra câu trả lời trên một dòng

Sample Input

Input	Output
5 1 4 2 3 5 6 2 1 3 1 3 3 2 1 5 2 3 5 1 2 3 2 2 4	4 5 5 3

Lời giải: Sử dụng cây phân khúc tìm Max. Mỗi nút it[id] lưu giá trị lớn nhất trong các phần tử của đoạn mà id quản lí.

+ Ban đầu xây dựng cây phân khúc quản lý độ ngon nhất của các đoạn. Với mỗi a[i] cập nhật giá trị a[i] vào vị trí i của cây phân khúc quản lý đoạn [1;n], nút 1 là gốc.

```
cin >> n;
For(i, 1, n) cin >> a[i], update(1, 1, n, i);
```

+ Xử lý các truy vấn: Với mỗi truy vấn đọc ra 3 số t, u, v

- Nếu t=1 thì a[u]=v và cập nhật giá trị v vào vị trí u trên cây phân khúc

- Nếu t=2 thì sử dụng hàm truy vấn get trả lời giá trị lớn nhất trên đoạn [u,v]

Độ phức tạp thuật toán cỡ: O(q*logn)

Code tham khảo:

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

```
#define fi first
#define se second
#define ll long long
#define ii pair<int,int>
#define pb push_back
#define rep(i, n) for(int i = 0; i < (n); i++)
#define For(i, a, b) for(int i = (a); i <= (b); i++)
using namespace std;
const int N = 2e5 + 1;
int n, a[N];
ll it[4 * N];
void update(int id, int l, int r, int pos){
	if(l > r || l > pos || r < pos) return;
	if(l == r) return void(it[id] = a[pos]);
	int m = (l + r) / 2;
	update(id * 2, l, m, pos);
	update(id * 2 + 1, m + 1, r, pos);
	it[id] = max(it[id * 2], it[id * 2 + 1]);
}
ll get(int id, int l, int r, int u, int v){
	if(l > v || r < u) return -1e18;
	if(l >= u && r <= v) return it[id];
	int m = (l + r) >> 1;
	return max(get(id * 2, l, m, u, v), get(id * 2 + 1, m + 1, r, u, v));
}
void solve(){
	cin >> n;
	For(i, 1, n) cin >> a[i], update(1, 1, n, i);
	int q; cin >> q;
	while(q--){
		int t, u, v; cin >> t >> u >> v;
		if(t == 1) a[u] = v, update(1, 1, n, u);
		else cout << get(1, 1, n, u, v) << '\n';
	}
}
int main(){
	ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
```

```
    int T = 1;
    //cin >> T;
    while(T--){
       solve();
    }
    return 0;
}
```

Ví dụ 2: Educational Segment Tree Contest - ITEZ2

Bạn được cho một mảng gồm n số nguyên. Ban đầu tất cả các số của mảng đều là 0. Nhiệm vụ của bạn là xử lí 2 loại truy vấn:

- Loại 1 có dạng 1 x y: Gán phần tử ở vị trí thứ x trong dãy thành số y ($1 \le x \le n$, $|y| \le 10^9$)
- Loại 2 có dạng 2 l r: In ra tổng các phần tử trong đoạn từ l đến r ($1 \le l \le r \le n$)

Với mỗi truy vấn loại 2, hãy in ra câu trả lời trên một dòng.

Input

- Dòng đầu tiên là hai số n và q lần lượt là số phần tử của mảng ($1 \le n \le 10^5$) và số truy vấn ($1 \le q \le 10^5$).
- q dòng sau, mỗi dòng là một truy vấn thuộc một trong 2 loại trên.

Output

Với mỗi truy vấn loại 2, in ra câu trả lời trên một dòng

Sample Input

Input	Output
5 8 1 1 2 1 3 5 2 1 3 1 5 4 2 1 5 1 4 3 1 3 5 2 1 4	7 11 10

Lời giải: Sử dụng cây phân khúc tính tổng. Nút it[id] lưu tổng các phần tử của đoạn mà id quản lí

+ Ban đầu đọc n và số lượng truy vấn q

```
cin >> n>>q;
```

+ Xử lý các truy vấn: Với mỗi truy vấn đọc ra 3 số t, u, v

- Nếu t=1 thì a[u]=v và cập nhật giá trị v vào vị trí u trên cây phân khúc quản lí tổng các phần tử trong đoạn

- Nếu t=2 thì sử dụng hàm truy vấn get trả lời giá trị tổng các phần tử trong đoạn [u,v]

Độ phức tạp thuật toán cỡ: O(q*logn)

Code tham khảo:

```
#include<bits/stdc++.h>
#define fi first
#define se second
#define ll long long
#define ii pair<int,int>
#define rep(i, n) for(int i = 0; i < (n); i++)
#define For(i, a, b) for(int i = (a); i <= (b); i++)
using namespace std;
const int N = 2e5 + 1;
int n, a[N];
ll it[4 * N];
void update(int id, int l, int r, int pos){
	if(l > r || l > pos || r < pos) return;
	if(l == r) return void(it[id] = a[pos]);
	int m = (l + r) / 2;
	update(id * 2, l, m, pos);
	update(id * 2 + 1, m + 1, r, pos);
	it[id] = it[id * 2] + it[id * 2 + 1];
}
ll get(int id, int l, int r, int u, int v){
	if(l > v || r < u) return 0;
	if(l >= u && r <= v) return it[id];
	int m = (l + r) >> 1;
	return get(id * 2, l, m, u, v) + get(id * 2 + 1, m + 1, r, u, v);
}
void solve(){
	cin >> n;
	int q; cin >> q;
	while(q--){
		int t, u, v; cin >> t >> u >> v;
		if(t == 1) a[u] = v, update(1, 1, n, u);
		else cout << get(1, 1, n, u, v) << '\n';
	}
}
int main(){
	ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
	int T = 1;
```

```
    //cin >> T;
    while(T--){
       solve();
    }
    return 0;
}
```

Ví dụ 3. Educational Segment Tree Contest - ITCount

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

Giả sử chúng ta có một mảng b với b_i=1 nếu a_i>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:

Ví dụ 4. Educational Segment Tree Contest - ITLAZY

Bạn được cho một mảng gồm n số nguyên. Nhiệm vụ của bạn là xử lí 2 loại truy vấn:

- Loại 1 có dạng 1 x y val: Tăng giá trị của các phần tử từ vị trí thứ x đến vị trí thứ y trong dãy lên val đơn vị ($1 \le x \le y \le n, 1 \le val \le 10^9$)
- Loại 2 có dạng 2 l r: In ra phần tử lớn nhất của dãy từ phần tử thứ l đến phần tử thứ r ($1 \le l \le r \le n$)

Với mỗi truy vấn loại 2, hãy in ra câu trả lời trên một dòng.

Input

- Dòng đầu tiên là số n là số phần tử của mảng ($1 \le n \le 10^5$)
- Dòng 2 là n số nguyên là các phần tử của mảng a ($|a_i| \le 10^9$)
- Dòng 3 là 1 số q là số truy vấn($1 \le q \le 10^5$)

- q dòng sau, mỗi dòng là một truy vấn thuộc một trong hai loại trên.

Output: Với mỗi truy vấn loại 2, in ra câu trả lời trên một dòng

Sample Input

Input	Output
3 1 2 3 7 1 1 3 1 1 2 3 4 2 1 3 1 1 1 5 2 1 2 1 1 2 2 2 2 3	8 7 9

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.

Giả sử nút it[id] quản lý giá trị lớn nhất đoạn [u, v] thì khi tăng tất cả các phần tử của đoạn [u, v] lên val đơn vị thì it[id] cũng tăng lên val đơn vị và vẫn là giá trị lớn nhất của đoạn sau khi tăng. Do đó, thay vì cập nhật val cho tất cả các phần tử của đoạn [u, v] thì ta chỉ cần cập nhật cho nút cha quản lí đoạn đó. Vì thế ta gọi là cập nhật lười.

Khi cần truy cập vào các nút con của id thì giá trị cập nhật lười sẽ được truyền xuống 2 con của nó, sau đó giá trị cập nhật lười ở nút cha trả về 0 để tránh cập nhật chồng.

```
void up(int id){
    For(i, id * 2, id * 2 + 1) it[i] += lazy[id], lazy[i] += lazy[id];
    lazy[id] = 0;
```

```
}
```

Code tham khảo:

```
#include<bits/stdc++.h>
#define ll long long
#define rep(i, n) for(int i = 0; i < (n); i++)
#define For(i, a, b) for(int i = (a); i <= (b); i++)
using namespace std;
const int N = 2e5 + 1;
int n, a[N];
ll it[4 * N], lazy[4 * N];
void up(int id){
	For(i, id * 2, id * 2 + 1) it[i] += lazy[id], lazy[i] += lazy[id];
	lazy[id] = 0;
}
void Update(int id, int l, int r, int u, int v, int val){
	if(l > v || r < u) return;
	if(l >= u && r <= v){
		lazy[id] += val;
		it[id] += val;
		return;
	}
	up(id);
	int m = (l + r) / 2;
	Update(id * 2, l, m, u, v, val);
	Update(id * 2 + 1, m + 1, r, u, v, val);
	it[id] = max(it[id * 2], it[id * 2 + 1]);
}
ll get(int id, int l, int r, int u, int v){
	if(l > v || r < u) return -1e18;
	if(l >= u && r <= v) return it[id];
```

```
        up(id);
        int m = (l + r) >> 1;
        return max(get(id * 2, l, m, u, v), get(id * 2 + 1, m + 1, r, u, v));
}
void solve(){
        cin >> n;
        For(i, 1, n) cin >> a[i], Update(1, 1, n, i, i, a[i]);
        int q; cin >> q;
        while(q--){
                int t, u, v; cin >> t >> u >> v;
                if(t == 1) { int val; cin >> val;
                        Update(1, 1, n, u, v, val);
                }
                else cout << get(1, 1, n, u, v) << '\n';
        }
}
int main(){
        ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
        int T = 1;
        //cin >> T;
        while(T--){
           solve();
        }
        return 0;
}
```

II. BÀI TẬP ÁP DỤNG

1. Bài 1: Educational Segment Tree Contest - ITDS1

1.1. Đề bài

Cho một dãy số có N số a_1, a_2, …, a_N.

Bạn cần xử lí 2 loại truy vấn.

• Truy vấn loại 1 có dạng 1 i v, ta đổi số ở vị trí i thành v.

• Truy vấn loại 2 có dạng 2 L R k, ta cần in ra số nhỏ nhất mà lớn hơn bằng k trong khoảng [L..R].

Dữ liệu vào:

• Dòng đầu tiên ghi 2 số N và M là số phần tử trong dãy và số truy vấn ($1 \le N, M \le 10^5$).

• Dòng tiếp theo gồm N số là dãy số ban đầu.

• M dòng tiếp theo, mỗi dòng ghi một truy vấn nằm trong 1 trong 2 loại trên.

Các số trong input đều lớn hơn hoặc bằng 1 và nhỏ hơn hoặc bằng 10^9.

Kết quả ra: Với mỗi truy vấn loại 2 ghi trên 1 dòng đáp số.

Sample Input

Input	Output
5 4 1 5 3 4 5 2 1 3 2 2 3 5 6 1 2 2 2 1 5 2	3 -1 2

1.2. Hướng dẫn giải thuật

Trong bài toán này, mỗi nút của cây phân đoạn là một ***multiset*** chứa các phần tử trong đoạn mà nó quản lí. Khi đó, để hợp nhất các nút con, ta chỉ cần *insert* toàn bộ phần tử của cả 22 nút con vào nút cha.

Mỗi khi cập nhật cây phân đoạn, nếu ta duyệt từng phần tử của các nút con để *insert* vào nút cha thì với số lượng truy vấn M, ta sẽ mất độ phức tạp lên tới hơn O(M×N) *(chưa kể độ phức tạp của các thao tác trên multiset)*, vì số lượng phần tử của mỗi nút có thể lên tới N.

Thay vào đó, ta nhận thấy rằng, khi thay đổi giá trị a[i] thành v thì tất cả *multiset* của các nút quản lí phân đoạn chứa phần tử a[i] *(các nút nằm trên đường đi đơn từ gốc đến nút lá tương ứng)* đều sẽ phải xóa đi một giá trị a[i] và thêm vào một giá trị v. Do đó, với truy vấn loại 1, khi cập nhật một nút, ta không cần phải *insert* lại toàn bộ phần tử của cả 2 nút con, mà chỉ cần xóa đi một giá trị cũ trong *multiset* và chèn thêm giá trị mới.

Với truy vấn loại 2, ta thực hiện tương tự như thao tác lấy giá trị. Tuy nhiên, mỗi khi xét đến nút mà đoạn nó quản lí nằm hoàn toàn bên trong đoạn cần truy vấn *(trường hợp cơ sở)*, ta sử dụng àm lower_bound()lower_bound() để trả ra giá trị nhỏ nhất mà vẫn lớn hơn hoặc bằng k trong *multiset* của nút đó.

Độ phức tạp của thuật toán là $O(N \times \log^2 N + M \times \log^2 N)$.

1.3. Code tham khảo

```
#include<bits/stdc++.h>
#define base 1000000007
#define fi first
```

```
#define se second
#define ll long long
#define ii pair<int,int>
#define all(x) (x).begin(), (x).end()
#define pb push_back
#define el cout << '\n'
#define rep(i, n) for(int i = 0; i < (n); i++)
#define For(i, a, b) for(int i = (a); i <= (b); i++)
#define data map <int, int>
using namespace std;
const int N = 2e5 + 1;
int n, q, a[N];
data it[4 * N];
data operator + (data &A, data &B){
   data ans;
   for(auto &v : A) ans[v.fi] += v.se;
   for(auto &v : B) ans[v.fi] += v.se;
   return ans;
}
void build(int id, int l, int r){
   if(l == r){
      it[id][a[l]]++;
      return;
   }
   int m = (l + r) >> 1;
   build(id * 2, l, m);
   build(id * 2 + 1, m + 1, r);
   it[id] = it[id * 2] + it[id * 2 + 1];
}

void update(int id, int l, int r, int pos, int New){
        if(l > pos || r < pos) return;
        if(l <= pos && r >= pos){
                it[id][New]++;
                it[id][a[pos]]--;
                if(it[id][a[pos]] <= 0) it[id].erase(a[pos]);
                if(l == r) return;
        }
        int m = (l + r) >> 1;
   update(id * 2, l, m, pos, New);
   update(id * 2 + 1, m + 1, r, pos, New);
}
int k, val;

void get(int id, int l, int r, int u, int v){
        if(l > v || r < u || l > r) return;
        if(l >= u && r <= v){
```

```
                auto ide = it[id].lower_bound(k);
                if(ide != it[id].end()) val = min(val, (*ide).fi);
                return;
        }
        int m = (l + r) >> 1;
        get(id * 2, l, m, u, v);
        get(id * 2 + 1, m + 1, r, u, v);
}
void solve(){
        cin >> n >> q;
   for(int i = 1; i <= n; i++) cin >> a[i];
   build(1, 1, n);
   while(q--){
        int t, u, v; cin >> t >> u >> v;
        if(t == 1) (update(1, 1, n, u, v), a[u] = v);
        else{
                cin >> k; val = base;
                get(1, 1, n, u, v);
                cout <<  (val == base ? -1 : val);
                el;
                }
        }
}

int main(){
        ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
        int T = 1;
        //cin >> T;
        while(T--){
           solve();
        }
        return 0;
}
```

2. Bài 2: Educational Segment Tree Contest - ITMED

2.1. Đề bài

Trò chơi này thật đơn giản: Bạn có n hộp được đánh số từ 1 đến n và một số k. Trong mỗi hộp có một tấm séc ghi số a_i. Nếu bạn chọn hộp i và $a_i \geq 0$ thì bạn sẽ nhận được a_i đồng còn $a_i \leq 0$ thì bạn sẽ phải "nộp" cho chương trình $|a_i|$ đồng.
Bằng khả năng "see the future" khủng của mình, bạn đã biết được con số ghi trên tấm séc trong mỗi hộp. Nhiệm vụ của bạn bây giờ chỉ còn là kiếm nhiều tiền nhất. Nhưng, trò chơi này có 2 điều luật như sau:

- Ví dụ bạn chọn hộp thứ i, thì hộp tiếp theo bạn chọn phải có chỉ số ≤i+k. Với hộp đầu tiên thì bạn có thể chọn tùy ý.
- Bạn chỉ được chọn các hộp từ trái qua phải. Nói cách khác, nếu bạn chọn hộp thứ i thì hộp tiếp theo bạn chọn phải có số thứ tự >i.

Bạn hãy tìm cách chơi để đem về nhà được nhiều tiền nhất nhé.

Input

- Dòng đầu tiên là 2 số n và k ($1 \le n, k \le 10^5$)
- Dòng thứ 2 là n số nguyên là số ghi trên tấm séc trong mỗi hộp ($|a_i| \le 10^9$)

Sample Input

Input	Output
5 2 -1 -2 3 -4 5	8

2.2. Hướng dẫn giải thuật

Sử dụng thuật toán QHĐ cơ bản như sau:

Gọi dp[i] là số tiền lớn nhất khi chọn đến hộp thứ i. Có 2 khả năng xảy ra là:

Nếu a[i] được chọn đầu tiên thì dp[i] = a[i]

Nếu a[i] được chọn sau a[j] thì trong đoạn k hộp liên tiếp trước i ta chọn hộp mà dp[j] tại đó lớn nhất, tức là dp[i] = max(dp[j]| i-k ≤ j ≤ i-1) + a[i]

➔ Độ phức tạp thuật toán cỡ O(N*k)

Cài tiến: Sử dụng cây phân khúc quản lí Max để tìm nhanh giá trị lớn nhất của đoạn dp[i-k…i-1] để tìm dp[i], sau đó update giá trị dp[i] tại vị trí i

2.3. Code tham khảo

```
#include<bits/stdc++.h>
#define fi first
#define se second
#define ll long long
#define ii pair<int,int>
#define heap_max(a) priority_queue<a>
#define heap_min(a) priority_queue<a, vector<a>, greater <a>>
#define all(x) (x).begin(), (x).end()
#define pb push_back
#define rep(i, n) for(int i = 0; i < (n); i++)
#define For(i, a, b) for(int i = (a); i <= (b); i++)
using namespace std;
const int N = 2e5 + 1;
int n, k;ll a[N];ll it[4 * N];
void update(int id, int l, int r, int pos){
        if(l > r || l > pos || r < pos) return;
```

```
        if(l == r) return void(it[id] = a[pos]);
        int m = (l + r) / 2;
        update(id * 2, l, m, pos);
        update(id * 2 + 1, m + 1, r, pos);
        it[id] = max(it[id * 2], it[id * 2 + 1]);
}
ll get(int id, int l, int r, int u, int v){
        if(l > v || r < u) return -1e18;
        if(l >= u && r <= v) return it[id];
        int m = (l + r) >> 1;
        return max(get(id * 2, l, m, u, v), get(id * 2 + 1, m + 1, r, u, v));
}
void solve(){
        cin >> n >> k;
        For(i, 1, n) cin >> a[i];
        update(1, 1, n, 1);
        For(i, 2 ,n){
                a[i] += max(get(1, 1, n, max(i - k, 1), i - 1), 0ll);
                update(1, 1, n, i);
        }
        ll res = 0;
        For(i, 1, n) res = max(res, a[i]);
        cout << res;
}
int main(){   ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
        int T = 1; //cin >> T;
        while(T--){
           solve();
        return 0;}
```

3. Bài 3: GSS - Đoạn con có tổng lớn nhất

3.1. Đề bài

Cho dãy số a[1], a[2], ..., a[n] (|a[i]| <= 15000, n <= 50000).

Hàm $q(x, y) = \max\left(\sum_{k=i}^{j} a_k, \text{ với } x \le i \le j \le y\right)$.

Cho m câu hỏi dạng x, y ($1 \le x \le y \le n$). ($m \le 50000$). Hãy tính các q(x, y).

Input

- Dòng đầu là n.

- Dòng thứ hai là dãy a.
- Dòng thứ 3 là m.
- m dòng tiếp theo mỗi dòng là 1 cặp số x, y.

Output

-> Lần lượt ghi ra các q(x, y) tương ứng. Mỗi kết quả ghi trên 1 dòng.

Sample Input

Input	Output
3 -1 2 3 1 1 2	2

3.2. Hướng dẫn giải thuật

Xây dựng cây Segment Tree mà mỗi nút của cây lưu 4 thông tin. Nút id quản lí đoạn [l, r] có các thông tin tương ứng là:

+ q[l, r] để trả lời câu hỏi của đề bài
+ MaxLeft(l, r) đoạn con liên tiếp có tổng lớn nhất bắt đầu từ vị trí l
+ MaxRight(l,r) đoạn con liên tiếp có tổng lớn nhất kết thúc ở vị trí r
+ sum(l,r) tổng các phần tử của đoạn [l, r]

Xét 3 trường hợp:

TH1: Đoạn con có tổng lớn nhất nằm hoàn toàn bên trong nút con bên trái.

TH2: Đoạn con có tổng lớn nhất nằm hoàn toàn bên trong nút con bên phải.

TH3: Một phần của đoạn con có tổng lớn nhất nằm ở nút con bên phải, và phần còn lại nằm ở nút con bên trái.

Ở 2 trường hợp đầu tiên thì ta chỉ cần trực tiếp lấy giá trị của đoạn con lớn nhất từ các nút con.

Ở trường hợp thứ 3, giá trị của đoạn con lớn nhất sẽ là tổng hậu tố lớn nhất của nút con bên trái với tiền tố lớn nhất của nút con bên phải.

3.3. Code tham khảo

```
#include<bits/stdc++.h>
#define oo 99999999
using namespace std;
const int N = 5e4 + 5;
int n, q, a[N];
struct Node{
	int sum, left, right, res;
} it[4*N];
Node mn(int a, int b, int c, int d){
	Node ans;
	ans.sum = a;
	ans.left = b;
	ans.right = c;
	ans.res = d;
	return ans;
}
Node operator + (Node a, Node b){
```

```
	Node ans;
	ans.sum = a.sum + b.sum;
	ans.left = max(a.left, a.sum + b.left);
	ans.right = max(b.right, b.sum + a.right);
	ans.res = max(max(a.res, b.res), a.right + b.left);
	return ans;
}
void build(int id, int l, int r){
	if (l == r)
	{
		it[id] = mn(a[l], a[l], a[l], a[l]);
		return;
	}
	int m = (l+r) / 2;
	build(id*2, l, m);
	build(id*2+1, m+1, r);
	it[id] = it[id*2] + it[id*2+1];
}
Node get(int id, int l, int r, int u, int v)
{
	if(r < u || l > v) return mn(-oo, -oo, -oo, -oo);
	if(l >= u && 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);
}
main()
{
	ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
	cin >> n;
	for(int i = 1; i <= n; i++) cin >> a[i];
	build(1, 1, n);

	cin >> q;
	while(q--)
	{
		int u, v;
		cin >> u >> v;
		cout << get(1, 1, n, u, v).res << "\n";
	}
}
```

4. Bài 4: Tặng quà

4.1.Đề bài

Trong kỳ thi chọn học sinh giỏi cấp tỉnh năm học 2022 – 2023. Để động viên, khích lệ tinh thần cho học sinh ban tổ chức có chương trình tặng quà cho tất cả

học sinh tham dự kỳ thi. Ban tổ chức chuẩn bị sẵn n hộp đựng quà, mỗi hộp được đặt trên một bàn, các bàn đánh số từ 1 đến n. Trên hộp quà thứ i (i=1..n) có dán nhãn là a_i và trong đó có món quà trị giá w_i.

Học sinh có thể chọn một hay nhiều hộp quà liên tiếp hay không liên tiếp từ hộp quà ở bàn 1 đến bàn n, hộp quà chọn sau phải có nhãn lớn hơn nhãn trên hộp quà chọn trước, tức là:

$$\begin{cases} a_{i_1} < a_{i_2} < \cdots < a_{i_k} \\ 1 \le i_1 < i_2 < \cdots < i_k \le n \end{cases}$$

Em hãy chọn cho mình các món quà để có tổng trị giá là lớn nhất.

**** Dữ liệu vào:*** Đọc vào từ file văn bản QUA.INP gồm:

- Dòng 1 ghi số nguyên dương n ($n \le 5.10^5$);

- n dòng tiếp theo, dòng thứ i (i=1..n) ghi 2 số nguyên dương a_i ($a_i \le 10^9$) và w_i ($w_i \le 10^6$) là nhãn và trị giá của món quà trong hộp i. Các số trên cùng dòng cách nhau ít nhất một khoảng trống.

**** Kết quả ra:*** Ghi ra file văn bản QUA.OUT một số duy nhất là tổng trị giá các món quà được chọn.

**** Ví dụ:***

QUA.INP	QUA.OUT	Giải thích	QUA.INP	QUA.OUT	Giải thích
5 5 15 3 5 4 7 5 1 2 8	15	Chọn hộp quà thứ 1 có trị giá bằng 15	5 4 10 1 3 5 15 3 10 4 12	25	Có thể chọn các hộp quà thứ 1, 3 có tổng trị giá là: 10+15=25 hoặc chọn các hộp quà thứ 2, 4, 5 có tổng trị giá là: 3+10+12=25

**** Giới hạn:***

- Có 10/30 test, tương ứng 1,0 điểm với $n \le 10^3$;

- Có 20/30 test, tương ứng 2,0 điểm với $10^3 < n \le 5.10^5$.

4.2. Hướng dẫn giải thuật

- Thuật toán QHĐ tương tự như thuật toán tìm dãy con tăng dài nhất
- Thuật toán tầm thường độ phức tạp $O(n^2)$: Gọi f[i] = Tổng quà của dãy con tăng có tổng giá trị lớn nhất kết thúc tại a[i]

$$f[i] = \max_{\substack{j<i \\ a_j<a_i}}\{f[j]\} + w_i$$

*) Cải tiến thuật toán với độ phức tạp O(nlogn) như sau:

- Tính lần lượt các giá trị f[1 … n]
- Mỗi khi tính được một f[i], ta lưu trữ f[i] tại vị trí a[i] trên trục số

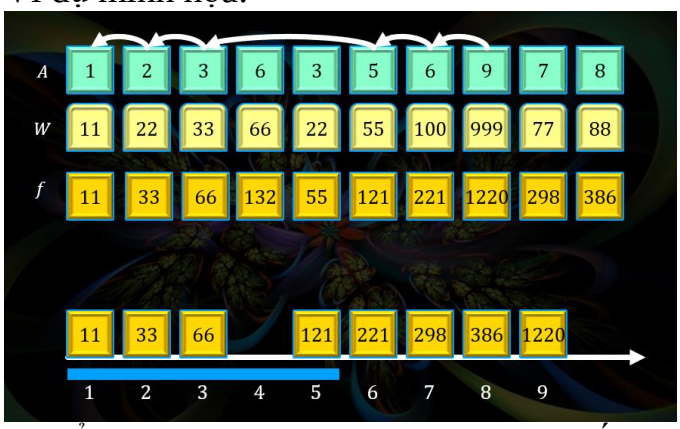
Để tính mỗi f[i] ta truy vấn giá trị lớn nhất trong các f[.] đã lưu trên trục số tại những điểm < a[i]. Đặt f[i] = giá trị đó + w[i]

Như vậy, nếu gọi g[a[i]] = Tổng giá trị của dãy con tăng trọng số lớn nhất kết thúc tại a[i]

g[a[i]] = Max{g[1 … a[i-1]] + w[i]

Việc tìm Max{g[1 … a[i-1]]} được thực hiện dễ dàng bằng cách sử dụng cây phân khúc Segment Tree

Ví dụ minh họa:



*) Để tìm được dãy con tăng có trọng số lớn nhất ta cần lưu lại giá trị f[i] = tổng trọng số lớn nhất của dãy con tăng kết thúc tại a[i]

4.3. Code tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector <int> vi;
typedef vector <ii> vii;
const int N = 1e6+5;
const int oo = 1e9+7;
const int MOD = 1e9+7;
const double PI = acos(-1);
int n, a[N],w[N],b[N];
ll f[N],g[N];
struct node
{
   int l,r;
   ll val;
};
vector <node> st;
void build(int l, int r, int id){
   st[id].val = 0;
   if ( (st[id].l=l) == (st[id].r=r) )
   {
      st[id].val = g[r];
      return;
   }
   int m = (l+r)/2;
```

```
    build(l,m,id*2);
    build(m+1,r,id*2+1);
    st[id].val = max(st[id*2].val, st[id*2+1].val);
}
void upd(int u, ll val, int id){
    if (u<st[id].l|| st[id].r<u) return;
    if (st[id].l == st[id].r){
        st[id].val = val;
        return;
    }
    upd(u,val,id*2);
    upd(u,val,id*2+1);
    st[id].val = max(st[id*2].val, st[2*id+1].val);
}
ll get(int u, int v, int id){
    if(v<st[id].l || st[id].r<u) return 0;
    if (u <= st[id].l && st[id].r <= v) return st[id].val;
    return max(get(u,v,id*2),get(u,v,id*2+1));
}
void nhap(){
    cin>>n;
    for(int i=1;i<=n;i++) {cin>>a[i]; b[i]=a[i];}
    for(int i=1;i<=n;i++) cin>>w[i];
    sort(b+1,b+n+1);
    for(int i=1;i<=n;i++)a[i] = lower_bound(b+1,b+n+1,a[i])-b;
    for(int i=1;i<=n;i++) g[i]=0;
    st.resize(4*n+1);
    build(1,n,1);
}
void xuly(){
   f[0]=0;g[0]=0;
   for(int i=1;i<=n;i++){
    int x=a[i];
    long long y = get(1,x-1,1);
    g[x] = max(y + w[i],g[x]);
    f[i] = y + w[i];
    upd(x,g[x],1);
   }
}
void Timkq(){
    ll res=0;
    for(int i=1;i<=n;i++) res=max(res,f[i]);
    ll tempA = N;
    cout<<res<<endl;
    int dem=0;
    for(int i=n;i>=1;i--){
        if (f[i]==res && a[i] < tempA){
```

```
            res = res - w[i];
            w[i] = -1;
            tempA = a[i];
            dem++;
        }
    }
    cout<<dem<<'\n';
    for(int i=1; i<=n;i++)
       if (w[i]==-1) cout<<i<<" ";
}
int main(){    ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    freopen("IS.INP","r",stdin);    freopen("IS.OUT","w",stdout);
    nhap();
    xuly();
    Timkq();
    return 0;}
```

5. Bài 5: Xếp nhóm Group

5.1. 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,…,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, + 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 $s_{1,2},…,s_n$ 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 ($n \leq 10^5$; $k \leq 100$);
- Dòng thứ hai chứa n số nguyên dương s_i ($1 \leq s_i \leq 10^9$, = 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

5.2. Hướng dẫn giải thuật

Tính số nhóm có thể chia được:

Tính tổng dồn 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 $ai \geq x \rightarrow S=S + x$

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.

5.3. 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;
}
```

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, chúng tôi đã trình bày về khái niệm cấu trúc dữ liệu cây phân khúc, cách xây dựng cây phân khúc 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 khúc vào một số bài toán học sinh giỏi, đặc biệt là các bài toán xử lí trên dãy số, chúng 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.

Chúng tôi rất mong quý thầy cô đồng nghiệp đóng góp ý kiến để chuyên đề được hoàn thiện hơn.

Xin trân trọng cảm ơn!

Bắc Giang, tháng 2 năm 2024

Tác giả

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