TRƯỜNG THPT CHUYÊN HẠ LONG

HƯỚNG DẪN CHẤM ĐỀ THI ĐỀ XUẤT
HỌC SINH GIỎI DHBB NĂM 2023

Môn thi: **TIN HỌC 11**

I. HƯỚNG DẪN CHUNG

1. Giám khảo nghiên cứu đề, hướng dẫn chấm, test, code;
2. Chép bài của thí sinh vào máy chấm. Kiểm tra bài thí sinh trên giấy in và trên máy;
3. Thực hiện chấm bài trên máy chấm (bài thi của thí sinh được chấm trên máy tính bằng phần mềm chấm thi Themis, bản quyền của Cục Quản lý chất lượng);
4. Kiểm tra kết quả chấm bài (kiểm tra lại các bài có kết quả như không tồn tại bài, lỗi biên dịch, các bài bị 0 điểm, …);
5. Tổng hợp kết quả chấm bài của thí sinh. Điểm bài thi được xuất từ phần mềm chấm thi, không quy tròn điểm của từng câu, điểm của bài thi.

II. ĐÁP ÁN VÀ BIỂU ĐIỂM

Bài 1. Mật khẩu (7 điểm)

➢ **Phân bổ điểm**

- Có 20% số test ứng với 20% số điểm của bài có $m \le 10; n_i \le 3; k_i \le 2 \times 10^5$;
- Có 20% số test ứng với 20% số điểm của bài có $m \le 2 \times 10^5$; $n_i + k_i \le 50$;
- Có 30% số test khác ứng với 30% số điểm của bài có $m \le 5000$; $n_i, k_i \le 5000$;
- Có 30% số test còn lại ứng với 30% số điểm của bài có giới hạn như dữ kiện bài ra.

➢ **Hướng dẫn thuật toán**

Subtask 1:

- Nếu $n = 1$, ta có: $\rho(1,k) = k.\rho(0,k) = k$;
- Nếu $n = 2$, ta có: $\rho(2,k) = \frac{k[\rho(0,k) + \rho(1,k)]}{2} = \frac{k(1+k)}{2}$;
- Nếu $n = 3$, ta có: $\rho(3,k) = \frac{k[\rho(0,k) + \rho(1,k) + \rho(2,k)]}{3} = \frac{k[1 + k + \frac{k(k+1)}{2}]}{3} = \frac{k(1+k)(2+k)}{6}$.

Căn cứ vào các công thức trên, tính kết quả của biểu thức lấy theo modulo $10^9 + 7$.

Độ phức tạp thuật toán: $O(m)$.

Subtask 2:

Với $n \ge 1$, ta có:

$$\rho(n,k) = \frac{k[\rho(0,k) + \rho(1,k) + \ldots + \rho(n-1,k)]}{n} = \frac{n-1}{n} \cdot \frac{k[\rho(0,k) + \rho(1,k) + \ldots + \rho(n-2,k)]}{n-1} + \frac{k\rho(n-1,k)}{n}$$
$$= \frac{(n-1+k)\rho(n-1,k)}{n}.$$

Từ công thức truy hồi ta xây dựng hàm đệ quy để tính giá trị biểu thức và lấy kết quả theo modulo $10^9 + 7$. Nhưng với cách tính toán xây dựng hàm đệ quy để tính biểu thức, kết quả có thể bị tràn số khi $n + k > 50$.

Độ phức tạp thuật toán: $O(m.n)$.

Subtask 3:

Từ công thức truy hồi ở trên, ta có:

$$\rho(n,k) = \frac{(n-1+k)(n-2+k)...(1+k)k}{n!}$$

Đến đây ta tính $\rho(n,k)\%(10^9+7)$ theo các bước sau:

Bước 1. Dùng vòng lặp tính:

$$a = (n-1+k)(n-2+k)...(1+k)k \ \%\ (10^9+7), b = n!\%(10^9+7).$$

Bước 2. Dùng định lí Ferma nhỏ tính c là nghịch đảo mô đun của $n!$ (theo modulo 10^9+7).

Định lí Ferma nhỏ: *Với số nguyên dương u và số nguyên tố p, ta có: $u^{p-1} = 1(mod\ p)$.*

Từ đó suy ra u^{p-2} là nghịch đảo mô đun của u theo mô đun p.

Do $p = 10^9+7$ là số nguyên tố nên $c = b^{p-2}\%p$ (tính bằng phương pháp chia để trị).

Bước 3. Kết quả của $\rho(n,k)\%p$ là: $(a.c)\%p$.

Độ phức tạp thuật toán: $O(m.n)$.

Subtask 4:

Ta có: $\rho(n,k) = \frac{(n-1+k)(n-2+k)...(1+k)k}{n!} = \frac{(n-1+k)!}{n!(k-1)!}$.

Để giảm độ phức tạp tính toán ta tính trước 2 mảng sau:

$$fc[i] = i!\%p,\ rfc[i] = (fc[i])^{p-2}\%p \text{ (với } p = 10^9+7, i = 0,1,...,400000).$$

Khi đó: $\rho(n,k)\%p = (fc[n-1+k].rfc[n]\%p).rfc[k-1]\%p$.

Tham khảo code lời giải mẫu để biết cách cài đặt thuật toán.

Độ phức tạp thuật toán: $O(m.(n+k))$.

Bài 2. Giá trị nhỏ nhất (7 điểm)

- **Phân bổ điểm**
  - Có 20% số test ứng với 20% số điểm của bài có $m,n \le 30$;
  - Có 20% số test ứng với 20% số điểm của bài có $m,n \le 100$;
  - Có 30% số test khác ứng với 30% số điểm của bài có $m,n \le 300$;
  - Có 30% số test còn lại ứng với 30% số điểm của bài có giới hạn như dữ kiện bài ra.
- **Hướng dẫn thuật toán**

Subtask 1:

Dễ thấy giá trị X của mỗi lưới ô vuông con thỏa mãn yêu cầu là giá trị của một ô nằm trong lưới ô vuông con đó. Vì vậy ta dùng 4 vòng lặp lồng nhau để duyệt vét cạn các lưới ô vuông con theo mô tả của bài toán để tìm kết quả.

Độ phức tạp thuật toán: $O(m.n.(h.w)^2)$.

Subtask 2:

Căn cứ vào tính chất trung vị của dãy các số, ta đưa các giá trị của lưới ô vuông con vào mảng $d[1..hw]$ và sắp xếp theo thứ tự không giảm thì số X cần tìm là $d[\frac{hw+1}{2}]$ ($\frac{hw+1}{2}$ là phép chia nguyên).

Độ phức tạp thuật toán: $O(m.n.h.w.log2(hw))$.

Subtask 3, 4:

Đây chính là bài toán tìm trung vị của dãy số có các giá trị trong lưới ô vuông con kích thước $h \times w$, khi sắp xếp các số này theo thứ tự không giảm. Ta thấy rằng kết quả của bài toán nằm trong phạm vi từ $l = 0$ đến $r = 10^9$ nên ta tìm kiếm nhị phân trên kết quả trên đoạn này.

Gọi $d = \frac{h.w+1}{2}$, khi đó với mỗi giá trị $mid = \frac{l+r}{2}$ ta xây dựng hàm $check(mid)$ để kiểm tra xem mid có là trung vị một lưới ô vuông con kích thước $h \times w$ hay không. Việc làm này được thực hiện bằng cách đếm số lượng các số trong lưới nhỏ hơn hoặc bằng k nếu giá trị này nếu lớn hơn hoặc bằng d thì ta trả về cho hàm $check(mid)$ giá trị là $true$, ngược lại trả về kết quả $false$.

Để giảm độ phức tạp tính toán khi đếm số lượng các số trong lưới nhỏ hơn hoặc bằng mid, ta cần tính trước toán mảng cộng dồn 2 chiều $s[.][.]$ như sau:

$$s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + (a[i][j] \le mid), \forall i = 1..m, j = 1..n.$$

Trong đó: $s[0][j] = s[i][0], \forall i = 0..m, j = 0..n$.

Sau khi tính xong mảng $s[.][.]$, ta dễ dàng tính được số lượng các số trên lưới có giá trị nhỏ hơn hoặc bằng mid bằng hàm

$$getsum(i,j,u,v) = s[u][v] - s[i-1][v] - s[u][j-1] + s[i-1][j-1],$$

với (i,j), (u,v) lần lượt là ô trên trái, dưới phải của lưới ô vuông kích thước $h \times w$ đang xét.

Việc cài đặt hàm $check(mid)$ và tìm kiếm nhị phân kết quả xin tham khảo code lời giải mẫu.

Độ phức tạp thuật toán: $O(m.n.log2(10^9))$.

Bài 3. Truy vấn trên cây (6 điểm)

- **Phân bổ điểm**
  - Có 30% số test ứng với 30% số điểm của bài có $1 \le n,q \le 7000$;
  - Có 25% số test ứng với 25% số điểm của bài có truy vấn loại 1 với $l = 0$;
  - Có 25% số test khác ứng với 25% số điểm của bài không có truy vấn loại 3;
  - Có 20% số test còn lại ứng với 20% số điểm của bài có giới hạn như dữ kiện bài ra.
- **Hướng dẫn thuật toán**

Kỹ thuật biến cây thành đoạn thẳng

Trên một cây, nếu ta dfs cây này và đánh số các đỉnh theo thứ tự ta thăm thì khi đó, mỗi một cây con luôn chứa các nút có nhãn liên tiếp nhau. Do đó, các bài truy vấn trên cây (đặc biệt là trên cây con gốc u) sẽ quy về bài toán truy vấn trên đoạn thẳng và sử dụng CTDL BIT/IT.

Subtask 1:

Duyệt độ phức tạp thuật toán: $O(NQ)$.

Subtask 2:

$l = 0 \Rightarrow$ các truy vấn làm tăng một lượng như nhau với các số trong một đoạn liên tiếp (cây con gốc u ứng với một đoạn liên tiếp trên dãy số). Đây chính là bài toán IT cơ bản: thao tác tăng một đoạn $l..r$ lên x và tính tổng các số từ l tới r.

Độ phức tạp thuật toán: $O(Qlog2(N))$.

Subtask 3:

Nhận xét: Độ tăng của giá trị của một đỉnh phụ thuộc vào khoảng cách từ đỉnh đó tới v (phụ thuộc vào độ cao của đỉnh đó trên cây). Một nút u thuộc cây con gốc v được tăng lên một lượng bằng

$$k - (high[u] - high[v]) * l = k + high[v] * l - high[u] * l$$

Trong biểu thức này, phần $k - high[v] * l$ là cố định với mọi đỉnh thuộc cây con gốc v, phần $- high[u] * l$ thì tỷ lệ thuận với độ cao của nút u. Do đó, ta dùng 2 cây IT khác nhau để duy trì hai đại lượng này: Một cây IT dùng để update lượng cố định ($k + high[v] * l$), cây IT còn lại update phần $-l$.

- Với truy vấn $update\ (v, k, l)$: $IT1.update(x..y, k + high[v] * l).\ IT2.update(x..y, -l)$.
- Với truy vấn get(v): $IT1.get(v) + high[v] * IT2.get(v)$.

Độ phức tạp thuật toán: $O(Qlog2(N)$.

Subtask 4

Ý tưởng giải giải thuật như subtask 3: Ta chia phần thay đổi giá trị của mỗi thao tác update ra làm 2 phần: phần tĩnh (thay đổi như nhau với mọi nút) và phần động (phần tỷ lệ thuận với chiều cao của nút đó). Với mỗi phần, ta sử dụng một cây IT/Segment Tree để quản lý phần đó.

- IT1 (quản lý phần tĩnh) giống như loại Segment Tree quen thuộc (tăng đoạn $l..r$ thêm x, tính tổng đoạn $l..r$).
- IT2 (quản lý phần động) là một cây Segment Tree mà có thể giải bài toán sau đây:
  Cho hai dãy số $a_1, a_2,..., a_n$ và $h_1, h_2,..., h_n$. Cần thực hiện 2 loại truy vấn sau:
  - $update\ l\ r\ x$: tăng các giá trị $a_l, a_{l+1}, ..., a_r$ thêm x.
  - $sum\ l\ r$: tính tổng $a_l * h_l + a_{l+1} * h_{l+1} + ... + a_r * h_r$.

  Chú ý: Dãy $h_1, h_2,..., h_n$ không bị thay đổi trong mọi truy vấn.

Nhận xét: IT1 có bản chất giống IT2, nếu như ta coi các hệ số $h_1 = h_2 = ... = h_n = 1$.

IT lazy update (tổng quát)

Xét bài toán cơ bản: Tăng đoạn $l..r$ lên x và tính tổng đoạn $l..r$.

Ý tưởng:

Vì sao cần lazy update: Ý tưởng của cây IT(segment tree) là mỗi nút trên cây quản lý đoạn $l..r$ nào đó và trên nút này cần lưu lại tổng giá trị các phần tử từ l tới r mà nó quản lý.

Tuy nhiên, trong trường hợp xấu, thao tác tăng đoạn $l..r$ lên x có thể làm ảnh hưởng tới cả N phần tử dẫn tới toàn bộ các nút trên cây IT đều phải thay đổi (nếu không có lazy update, mọi nút trên cây đều lưu lại tổng các số trong đoạn nó quản lý thì khi mọi số đc tăng x, mọi nút tương ứng cũng tăng theo). Làm cho việc update sẽ mất $O(N)$.

Do đó, để thao tác update chỉ mất độ phức tạp $O(logN)$, ta không thể cập nhật lại toàn bộ cây IT được. Thay vào đó, ở mỗi nút, ngoài thông tin "sum", ta còn lưu thêm một thông tin nữa gọi là "lazy", với mục tiêu như sau: Giá trị "sum" ở một nút trên cây IT có thể không phản ánh tổng thực tế của các phần tử của cây IT, nhưng nếu "gộp" sum ở một nút với mọi lazy ở các nút là tổ tiên của nó, ta sẽ có được tổng thực sự. Việc gộp các thông tin lazy tại các tổ tiên của một nút là hoàn toàn thực hiện được, vì để đi tới một nút, ta cần đi từ gốc, xuống dần tới nút đó.

Cài đặt:

Vấn đề khó nhất trong các bài toán dùng IT có lazy update là xác định các thông tin được lưu ở mỗi nút. Các thông tin được lưu ở mỗi nút được chia làm 2 loại:

- Loại 1: Thông tin dạng tích lũy, dùng để tính ra kết quả ("sum" trong ví dụ trên)
- Loại 2: Thông tin lazy update, dùng để lưu thông tin các truy vấn cập nhật.

```
void pushDown(int i, int l, int r) {
        int m = (l + r) >> 1;
        sum[2 * i] += lazy[i] * (m - l + 1);
        sum[2 * i + 1] += lazy[i]  * (r - m);
        lazy[2 * i] += lazy[i];
        lazy[2 * i + 1] += lazy[i];
        lazy[i] = 0;
}
```

```
// truy vấn tăng đoạn u..v thêm c, đang xét nút i của cây IT quản lý đoạn l..r
void update(int i, int l, int r, int u, int v, int c) {
	// nếu đoạn l..r, u..v không tồn tại hoặc hai đoạn này không giao nhau
	if (l > v || r < u || l > r || v < u) return;

	// nếu l..r nằm trong u..v (mọi phần tử thuộc đoạn l..r đều bị thay đổi)
	if (u <= l && r <= v) {
		sum[i] += (r - l + 1) * c;
		lazy[i] += c;
		return;
	}

	pushDown(i, l, r);
	// gọi đệ quy xuống 2 con để cập nhật
	int m = (l + r) >> 1;
	update(2 * i, l, m, u, v, c);
	update(2 * i + 1, m + 1, r, u, v, c);

	// tổng nút i bằng tổng con trái cộng tổng con phải
	sum[i] = sum[2 * i] + sum[2 * i + 1];
}
// truy vấn tính tổng đoạn u..v, đang xét nút i của cây IT quản lý đoạn l..r
int getSum(int i, int l, int r, int u, int v) {
	// nếu đoạn l..r, u..v không tồn tại hoặc hai đoạn này không giao nhau
	if (l > v || r < u || l > r || v < u) return 0;

	// nếu l..r nằm trong u..v (mọi phần tử trong đoạn l..r đều đem vào tổng)
	if (u <= l && r <= v) return sum[i];

	pushDown(i, l, r);
	// gọi đệ quy xuống 2 con để tính tổng
	int m = (l + r) >> 1;
	int L = getSum(2 * i, l, m, u, v, c);
	int R = getSum(2 * i + 1, m + 1, r, u, v, c);

	return L + R;
}
```

Trong đoạn code phía trên, các phần in nhạt là phần giống nhau với mọi bài IT lazy update (luôn có 3 hàm pushDown, update, get). Chỉ có các phần được tô đậm là khác nhau tùy vào bài sử dụng IT lazy update vào mục đích gì.

Các phần tô đậm khác nhau:

- Hàm pushDown: dùng để đẩy thông tin lazy update từ trên xuống dưới. Trong các bài khác nhau, các thông tin cần lưu ở mỗi nút trên cây IT khác nhau, do đó hàm lazy update này cũng khác nhau.
- Trong hàm update, phần trường hợp đoạn l..r nằm hoàn toàn trong đoạn u..v: Phần này mang ý nghĩa là nếu mọi phần tử trong nút bị cập nhật thì nút thay đổi như thế nào.

- Trong hàm update, cập nhật thông tin từ con lên cha (sum[i] = sum[2 * i] + sum[2 * i + 1]): đôi khi còn được gọi là pushUp.
- Trong hàm get, phần gộp thông tin kết quả từ hai nút con (return L + R).

Do đó, khi làm một bài sử dụng IT lazy update, sau khi xác định các loại truy vấn cần được hỗ trợ, ta cần trả lời các câu hỏi sau:

1. Các thông tin được lưu ở mỗi nút của cây IT là gì. Trong các thông tin đó, thông tin nào thuộc loại 1, thông tin nào thuộc loại 2.
2. Hàm pushDown được viết như thế nào (các thông tin loại một được cập nhật dựa trên thông tin loại hai như thế nào).
3. Các phần màu đỏ trong đoạn code phía trên cần viết như thế nào.

Quay trở lại phần subtask 4, cài IT2:

1. Các thông tin cần lưu ở các nút gồm:
   - Tổng hệ số h ở các vị trí được quản lý ("sumH") - Loại 1
   - Tổng giá trị ở các vị trí được quản lý - Loại 1
   - Lazy update: lượng được tăng thêm ở mỗi vị trí thuộc đoạn đang quản lý - Loại 2
2. pushDown

```
void pushDown(int i) {
        sum[2 * i] += lazy[i] * sumH[2 * i];
        sum[2 * i + 1] += lazy[i] * sumH[2 * i + 1];
        lazy[2 * i] += lazy[i];
        lazy[2 * i + 1] += lazy[i];
        lazy[i] = 0;
}
```

3. Các phần tô đậm khác:
   - Khi một nút được tăng thêm c: sum[i] += c * sumH[i]; lazy[i] += c;
   - Gộp thông tin từ hai con lên cha: sum[i] = sum[2 * i] + sum[2 * i + 1];
   - return L + R;
4. Chú ý: do mảng sumH là không đổi và không bị ảnh hưởng trong mọi truy vấn, nên ta có thể tạo hàm init để tính trước mảng sumH này.

```
void calcSumH(int i, int l, int r) {
        if (l > r) return;
        if (l == r) {
                sumH[i] = h[l];
                return;
        }
        int m = (l + r) >> 1;
        calcSumH(2 * i, l, m);
        calcSumH(2 * i + 1, m + 1, r);
        sumH[i] = sumH[2 * i] + sumH[2 * i + 1];
}
```

Tham khảo code lời giải mẫu để xem cách cách đặt.

Độ phức tạp thuật toán: $O(Qlog2(N))$.

---------- HẾT ----------