HƯỚNG DẪN GIẢI CODE CHALLENGE 2
KỲ THI CODE TOUR 2022

BÀI A: VNG CLOUD VÀ HACKER BÍ ẨN

Solution:

C++	https://ideone.com/LZUXhT

Tóm tắt đề:

Cho danh sách N điểm nguyên trên mặt phẳng p_1, p_2, …, p_N. Biết đa giác tạo thành từ N đoạn thẳng được tạo bởi 2 điểm liên tục trong danh sách, tức ta có N đoạn thẳng là (p_1, p_2), (p_2,p_3), …, (p_{N-1}, p_N), (p_N, p_1).

Biết rằng không có 2 cặp đoạn thẳng nào cắt nhau trong N đoạn thẳng đã cho.

Tính diện tích đa giác tạo bởi N đoạn thẳng đó.

Input:

Gồm nhiều testcase trên nhiều dòng.

- Dòng đầu tiên của mỗi testcase gồm một số nguyên **N** ($1 \le N \le 10^5$) là số lượng điểm nguyên trên mặt phẳng, cũng là số đoạn thẳng được cho.

- N dòng tiếp theo chứa 2 số nguyên **x, y** ($0 \le x, y \le 10^5$) là tọa độ nguyên của **N** điểm. Biết rằng các điểm phân biệt và không có 3 điểm cùng nằm trên 1 đường thẳng.

Chuỗi testcase kết thúc bằng số 0.

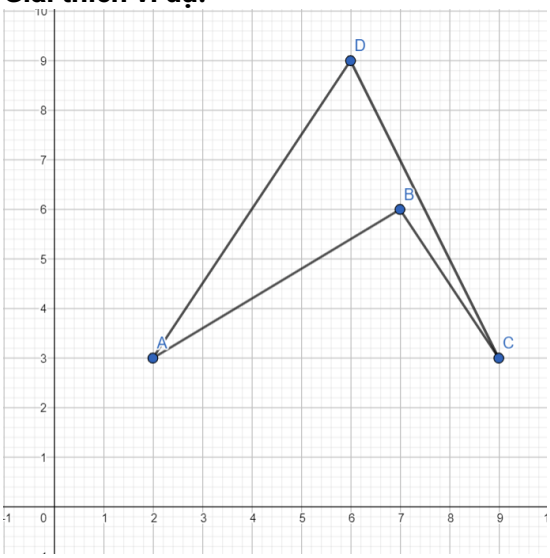
Output:

Diện tích đa giác đã cho, kết quả làm tròn 2 chữ số đằng sau dấu thập phân.

Ví dụ:

4 2 3 7 6 9 3 6 9 0	10.50

Giải thích ví dụ:



Diện tích tứ giác ABCD là 10.50

Hướng dẫn giải:

Sử dụng *Shoelace Formula*, gọi A là diện tích đa giác được cho bởi N điểm p_i với i = 1..N liên tục theo thứ tự, khi đó

$$2A = \left| \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_N & x_1 \\ y_N & y_1 \end{vmatrix} \right|$$

$$2A = |(x_1y_2 - x_2y_1) + (x_2y_3 - x_3y_2) + \cdots + (x_Ny_1 - y_Nx_1)|$$

Do đó ta chỉ cần dùng 1 vòng lặp duyệt qua tất cả các điểm liên tiếp của đa giác để tính tổng định thức của các ma trận kích thước 2x2 chứa tọa độ hai điểm.

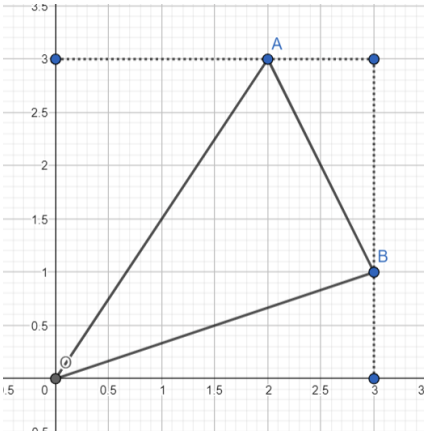
Vì đa giác tạo bởi các điểm theo thứ tự input đã cho có thể *định hướng âm* (phụ thuộc vào chiều thứ tự của các điểm tạo ra đa giác), nên tổng các định thức trên có thể có giá trị âm, do đó diện tích đa giác được tính dựa trên giá trị tuyệt đối của tổng trên.

Shoelace Formula có thể được hình dung như sau,
Xét bài toán đơn vị tính diện tích tam giác OAB với A(x,y), B(u,v), O(0,0). Để đơn giản trong việc tính toán, giả sử 0 < x < u, 0 < v < y.

Khi đó, diện tích tam giác OAB được tính bằng: $uy - {(xy + uv + (y - v)(u - v))}/{2} = {uy - xv}/{2}$

Tổng quát hơn ta có diện tích tam giác OAB = $|{xv - uy}/{2}|$.

Vậy để tính diện tích một đa giác bất kỳ ta sẽ chia đa giác này thành các tam giác đơn vị (gồm 3 đỉnh là gốc tọa độ O, 2 đỉnh liên tiếp của đa giác)



Độ phức tạp: **O(N)**

BÀI B: VIRUS

Solution:

C++	https://ideone.com/rIlicR

Tóm tắt đề:

Cho n là dân số của một vùng, mảng a với a[i] là khả năng nhiễm bệnh của người thứ i. Định nghĩa khả năng lây lan dịch bệnh cấp k như sau:

- Chia mảng a ra làm các đoạn liên tiếp, mỗi đoạn k phần tử (trừ đoạn cuối cùng có thể ít hơn nếu n không chia hết cho k).
- Với mỗi đoạn trên, tính tổng các tích 2 số trong đoạn đấy.
- Khả năng lây lan dịch bệnh cấp k sẽ là tổng của các giá trị tính được ở bước trên.

Ta cần tính tổng các khả năng lây lan dịch bệnh cấp k với k từ 1 đến n.

Input:

Dòng đầu tiên gồm số nguyên n – dân số của khu vực.

Dòng thứ 2 gồm n số a[1], a[2], ..., a[n].

Điều kiện:

- $1 <= n <= 2 * 10^5$
- $-10^9 <= a[i] <= 10^9$

Subtask:

- 10% số test tương ứng với 1 <= n <= 400.
- 20% số test tương ứng với 1 <= n <= 5000.
- 20% số test tương ứng với $1 <= n <= 2 * 10^5$, các giá trị trong mảng bằng nhau.
- 50% số test còn lại không ràng buộc gì thêm.

Output:

Tổng các khả năng lây lan dịch bệnh theo modulo $10^9 + 7$.

Ví dụ:

10 0 1 3 -1 2 4 5 -4 -2 3	17

Hướng dẫn giải:

Subtask 1: Vét cạn #1

$1 \leq n \leq 400$

Ta có thể tính tổng các tích của các số trên một đoạn con từ L đến R của mảng a như sau:

```
int sumOfProd(int L, int R) {
    int sum = 0;
    for(int i = L; i <= R; i++) {
        for(int j = L; j < i; j++) {
            int prod = ((int64_t)a[i] * a[j]) % MOD;
            sum = (sum + prod) % MOD;
        }
    }

    return sum;
}
```

Hàm trên có độ phức tạp là $O(k^2)$. Với k là độ dài của đoạn $[L, R]$.

Khi đó, để tính độ tập trung cấp k, ta có thể làm như sau:

```
int calc(int k) {
    int ans = 0;
    for(int i = 1; i <= n; i += k) {
        int j = min(i + k - 1, n);
        ans = (ans + sumOfProd(i, j)) % MOD;
    }

    return ans;
}
```

Hàm trên có độ phức tạp: $O\left(\frac{n}{k} \cdot k^2\right) = O(nk)$.

Để tính kết quả bài toán, ta duyệt qua hết các k từ 1 đến n, với mỗi k ta tính *độ tập trung cấp* k và cộng vào kết quả:

```
void solve() {
    int ans = 0;
    for(int k = 1; k <= n; k++)
        ans += calc(k);

    cout << ans << '\n';
}
```

Hàm trên có độ phức tạp: $O(n + 2n + 3n + \ldots + n \cdot n) = O\left(n \cdot \frac{n(n+1)}{2}\right) = O(n^3)$.

Subtask 2: Vét cạn #2

Ta có thể viết lại hàm `sumOfProd` ở Subtask đầu tiên bằng cách đặt $a[i]$ ở mỗi bước duyệt i ra làm nhân tử chung như sau:

```
int sumOfProd(int L, int R) {
    int sum = 0;
    for(int i = L; i <= R; i++) {
        int current = 0;
        for(int j = L; j < i; j++)
            current = (current + a[j]) % MOD;

        current = (((int64_t)a[i]) * current) % MOD;
        sum = (sum + current) % MOD;
    }

    return sum;
}
```

Nhận xét là việc tính toán `current` tương đương với lấy $a[i]$ nhân với tổng của các số từ L đến $i-1$. Do đó, ta có thể tăng tốc hàm trên bằng cách dùng thêm một biến `pre` để tính tổng của các số từ L đến $i-1$:

```
int sumOfProd(int L, int R) {
    int sum = 0;
    int pre = 0;
    for(int i = L; i <= R; i++) {
        int current = (((int64_t)a[i]) * pre) % MOD;
        pre = (pre + a[i]) % MOD;
        sum = (sum + current) % MOD;
    }

    return sum;
}
```

Khi này hàm `sumOfProd` có độ phức tạp $O(k)$ Với k là độ dài đoạn $[L, R]$.

Giữ nguyên các hàm `calc` và `solve` như Subtask 1, ta giải được bài toán với độ phức tạp $O(n^2)$.

Subtask 3:

Ở subtask này, mảng a chỉ có duy nhất 1 giá trị phân biệt.

Do đó, ta có thể dễ dàng tính được tổng các tích giữa hai số phân biệt trên một đoạn $[L, R]$ trong thời gian hằng số.

Cụ thể, giả sử mảng a chỉ có duy nhất một giá trị là x. Khi đó, tổng các tích giữa hai số phân biệt trên đoạn $[L, R]$ có độ dài k sẽ là: $x^2 \cdot \frac{k(k-1)}{2}$. Ta có hàm `sumOfProd` như sau:

```
int sumOfProd(int L, int R) {
    int x = a[L];
    int x2 = (((int64_t)x) * x) % MOD;
    int k = R - L + 1;
    int64_t numOfPairs = ((((int64_t)k) * (k - 1)) / 2) % MOD;

    return (x2 * numOfPairs) % MOD;
}
```

Khi đó, hàm `calc` sẽ có độ phức tạp là $O\left(\frac{n}{k}\right)$.

Do đó, ta có thể giải được bài toán với độ phức tạp là $O\left(\frac{n}{1}+\frac{n}{2}+\ldots+\frac{n}{n}\right) = O(n\log n)$.

Subtask 4:

Ta sẽ thử làm giống như Subtask 3. Tức là ta muốn tìm cách để tính `sumOfProd` trong thời gian $O(1)$.

Để làm được điều đấy, ta dùng công thức sau:

$$\left(\sum_{i=L}^{R} a_i\right)^2 = \sum_{i=L}^{R} a_i^2 + 2\sum_{L\le i<j\le R} a_i \cdot a_j$$

$$\frac{1}{2}\left[\left(\sum_{i=L}^{R} a_i\right)^2 - \sum_{i=L}^{R} a_i^2\right] = \sum_{L\le i<j\le R} a_i \cdot a_j$$

Từ biểu thức trên, ta sẽ cần tính được tổng và tổng các bình phương trên một đoạn $[L, R]$ bất kỳ một cách nhanh chóng. Ta có thể tính tổng một đoạn bất kỳ bằng cách tính trước mảng `psum` với `psum[i]` là $a_1 + a_2 + \ldots + a_i$. Khi đó tổng một đoạn $[L, R]$ bất kỳ sẽ là `psum[R] - psum[L - 1]`. Ta làm tương tự để tính tổng các bình phương.

Độ phức tạp của Subtask 4 tương tự với Subtask 3: $\boldsymbol{O(n\log n)}$

BÀI C: LR-COOL SUBARRAY

Solution:

Python	https://ideone.com/jwm0S9

Tóm tắt đề:

Cho một dãy số A gồm N chữ số $A_1, A_2, \dots, A_n$. Có Q truy vấn, mỗi truy vấn gồm 2 số L và R, yêu cầu hãy tìm số dãy con liên tiếp của A thỏa điều kiện tồn tại dãy số B gồm các **số nguyên** sao cho.

$$L \leq B_1 * A_i + B_2 * A_{i+1} + \cdots + B_{j-i+1} * A_j \leq R$$

Input:

Dòng đầu tiên gồm 1 số N, độ dài của dãy số A.

Dòng tiếp thứ 2 gồm N số, biểu diễn dãy A.

Dòng thứ 3 gồm 1 số là Q.

Q dòng tiếp theo mỗi dòng gồm 2 số L và R.

Ràng buộc:

- $1 \leq \mathrm{N} \leq 10^5$
- $1 \leq \mathrm{A_i} \leq 10^5$
- $1 \leq \mathrm{Q} \leq 10^5$
- $1 \leq \mathrm{L} \leq \mathrm{R} \leq 10^5$

Output:

Gồm Q dòng, mỗi dòng gồm kết quả tìm được của mỗi truy vấn.

Ví dụ:

3 1 2 3 3 1 2 1 3 3 3	5 6 5

Giải thích ví dụ:

Trong test ví dụ, truy vấn đầu tiên có 5 dãy thỏa mãn đó là {1, 2, 3}, {1, 2}, {2, 3}, {1}, {2}.

Hướng dẫn giải:

Để giải được bài toán, ta phải biết được

$$B_1 * A_1 + B_2 * A_2 + B_3 * A_3 + \cdots + B_p * A_p = K \ (1)$$

có nghiệm khi nào, theo giải thuật Euclid thì phương trình trên có nghiệm nếu như K chia hết cho G = gcd$(A_1, A_2, \dots, A_p)$

Với mỗi K để đếm số dãy con A có index (l, r) thỏa mãn ta có thể sử dụng binary search và sparse table. Cụ thể như sau, cố định r ($1 \leq r \leq n$), giờ ta phải đếm số l ($l \leq r$) sao cho K chia hết cho gcd$(a_l \dots a_r)$. Nhận thấy nếu ta cố định r, thì gcd$(a_{l_1} \dots r)$ chia hết cho gcd$(a_{l_2} \dots a_r)$ với $l_2 < l_1$.

Do đó với mỗi vị trí cố định r, ta dùng binary search để tìm các giá trị l và sparse table để truy vấn gcd trong $O(1)$. Gọi $cnt(x)$ là số đoạn (l, r) có gcd = x. Như vậy với mỗi giá trị K ta chỉ cần đếm số các số x sao cho K chia hết cho x.

Tuy nhiên bài toán gốc cần tìm là

$$L \leq B_1 * A_1 + B_2 * A_2 + B_3 * A_3 + \cdots + B_p * A_p \leq R \ (2)$$

Lưu ý rằng phương trình trên không tương đương với kết quả của (1) với số K trong [L, R]. Bởi nếu tồn tại 2 số $K_1 < L \leq K_2 \leq R$ và K_2 chia hết cho K_1 thì $cnt(K_1)$ cũng là kết quả. Như vậy với mỗi $cnt(x) > 0$, ta sẽ cần cộng vào kết quả của (2) nếu như trong đoạn từ L tới R có số chia hết cho x, nói theo một cách toán học là tồn tại 1 số nguyên dương k sao cho $L \leq k * x \leq R$ thì ta sẽ cộng thêm $cnt(x)$ vào kết quả.

Với mỗi truy ván, để giải quyết nhanh ta có thể làm bằng cách đó là xử lý truy vấn offline.

Đầu tiên, ta sẽ sort lại truy vấn tăng dần theo L, nếu L bằng nhau thì tăng dần theo R.

Các giá trị x ta có thể lưu vào trong 1 heap. Nếu heap.top() < L thì ta sẽ cộng đến khi nào mà >= L (biến đổi x = x * k sao cho >= L). Sau đó ta chỉ cần lấy tổng cnt của những số x <= R. Để tính tổng nhanh ta có thể sử dụng Fenwick Tree (lưu ý mỗi lần cập nhật x = x*k thì phải xóa đi giá trị cũ).

Độ phức tạp:

Tính toán mảng cnt ta sẽ tốn $O(n \ (\log n)^2)$. Do số gcd khác nhau từ r $(1 \leq r \leq n)$ về 1 chỉ tối đa khoảng $\log_2 n$.

Mỗi giá trị x ta sẽ phải cập nhật liên tục cho mỗi L nên sẽ tốn tối đa khoảng $O(M \log M)$ với M là giá trị tối đa của x (M không quá 10^5). Cập nhật và tính toán với cấu trúc dữ liệu sẽ tốn khoảng $O(\log M)$. Nên độ phức tạp các truy vấn sẽ là $O(Q * logM + M \ (logM)^2)$

$O((n\,(\log n)^2 + \ Q * logM + M\ (logM)^2)$ sẽ là độ phức tạp cuối cùng.

BÀI D: BRACKET

Solution:

C++	https://ideone.com/R7cSJo

Tóm tắt đề:

Biểu thức ngoặc là xâu chỉ gồm 2 ký tự '(' và ')'. Biểu thức ngoặc là đúng nếu :

- Biểu thức là rỗng
- Biểu thức ngoặc A đúng thì (A) đúng
- Biểu thức ngoặc A và B đúng thì AB đúng

Với biểu thức ngoặc có độ dài n. Hãy tìm biểu thức ngoặc sai độ dài n thứ k theo thứ tự từ điển

Input:

Gồm 2 số nguyên n, k ($n \le 10^6$, $k \le 10^{18}$, $k \le \min(2^n, 10^{18})$)

Output:

Biểu thức ngoặc sai có độ dài n thứ k theo thứ tự từ điển

Ví dụ:

5 2	(((()

Giải thích ví dụ:

Biểu thức ngoặc sai có độ dài là 5 lần lượt là (((((, ((((), ….

Hướng dẫn giải:

Coi 1 dấu '(' = 1 và ')' = -1. Vậy dãy ngoặc có độ dài n $c_1, c_2, ..., c_n$ là dãy ngoặc sai nếu tồn tại 1 vị trí j sao cho tổng $c_1, c_2, ..., c_j$ nhỏ hơn 0

Ta bắt đầu đặt các dấu ngoặc từ 1 đến n. Tại vị trí i:

- Nếu dãy ngoặc đặt từ vị trí 1 đến $i-1$ đã sai thì tại vị trí i đến n ta đặt vào đó dãy ngoặc có thứ tự từ điển lớn thứ k (không kể đúng sai).

- Nếu dãy ngoặc đặt từ vị trí 1 đến $i-1$ chưa sai thì ta giả sử đặt ngoặc '(' ở vị trí i và gọi x là số cách đặt tiếp dãy ngoặc từ $i+1$ đến n sao cho dãy sai:

- Nếu $k \le x$ thì ta đặt ở vị trí i là ngoặc '('.

- Nếu $k > x$ thì ta đặt ở vị trí i là ngoặc ')' và gán $k = k - x$ (loại x dãy có thứ tự từ điển bé hơn nếu đặt dấu ngoặc thứ i là '(').

Để tính x là số cách đặt tiếp từ $i + 1$ đến n sao cho dãy sai.

Ta có nhận xét:

Tổng số cách đặt dấu ngoặc (không kể đúng sai) là: 2n

→ Số cách đặt sai = 2n - số cách đặt đúng

Gọi f(i, j) là số cách đặt ngoặc có độ dài là i với j ngoặc '(' mà dãy đó chưa sai:

f(i, j) += f($i - 1, j - 1$) -> trường hợp đặt ở vị trí i là '('.

f(i, j) += f($i - 1, j$) -> trường hợp đặt ở vị trí i là ')', với điều kiện $2 * j - i \geq 0$ (số ngoặc mở lớn hơn ngoặc đóng).

Ta cần 2 biến *open* và *close* để lưu số ngoặc mở và đóng từ 1 đến $i - 1$.

Vậy $x = 2^{n-1}$ - f($n-i, n/2 - open - 1$)

Độ phức tạp: $O(n^2)$ với $n \leq 120$

---------- HẾT ----------