HƯỚNG DẪN GIẢI CODE TOUR 2024
Code Challenge #3

BÀI A: THỬ NGHIỆM SẢN PHẨM

Solution:

Python	https://ideone.com/jPhJPi
C++	https://ideone.com/Y3CYmC

Tóm tắt đề:

Cho N sản phẩm, khi phát hành sản phẩm i sẽ được tài trợ P_i đô la. Có M loại tài nguyên, chi phí khi sử dụng tài nguyên thứ j là C_j đô la. Mỗi sản phẩm sẽ yêu cầu một số tài nguyên nhất định để có thể phát hành, được biểu diễn bởi ma trận A $(N\text{x}M)$ với A_{ij} cho biết sản phẩm i có cần tài nguyên j không, mỗi loại tài nguyên có thể được sử dụng chung cho các sản phẩm khác nhau $(1 \leq N, M \leq 1000, 1 \leq P_i, C_j \leq 10^6)$.

Yêu cầu: tính lợi nhuận tối đa có thể đạt được, bằng tổng số tiền nhận được khi phát hành các sản phẩm được chọn trừ cho tổng chi phí các tài nguyên cần sử dụng. Ngoài ra, cần in ra các sản phẩm được chọn và tài nguyên sử dụng.

Ví dụ:

3 4 4 10 11 6 2 3 7 1 0 0 1 0 1 1 0 0 1 0 0	16 2 3 2 3

Giải thích:

Trong phương án tối ưu, các sản phẩm được phát hành là 2 và 3. Sản phẩm 2 cần tài nguyên 2 và 3, sản phẩm 3 cần tài nguyên 2, do đó tài nguyên cần sử dụng là 2 và 3. Tổng lợi nhuận là $10 + 11 - 2 - 3 = 16$.

Hướng dẫn giải:

Gọi tập N sản phẩm là X, và tập M tài nguyên là Y. Gọi tập các sản phẩm con của X được chọn để phát hành là X_1, cùng với tập sản phẩm X_1 được chọn đó, gọi Y_1 là tập các tài nguyên cần sử dụng.

Bài toán yêu cầu tìm một tập sản phẩm X_1 sao cho cực đại giá trị sau:

$$\mathrm{R} = \sum_{i \in X_1} P_i - \sum_{j \in Y_1} C_j$$

Ta có thể đưa bài toán trên trở về bài toán cực tiểu, bằng cách tính giá trị:

$$\mathrm{S} = \sum_{i \in X} P_i - R = \mathrm{R} = \sum_{i \in X} P_i - \left(\sum_{i \in X_1} P_i - \sum_{j \in Y_1} C_j \right)$$

$$= \left(\sum_{i \in X} P_i - \sum_{i \in X_1} P_i \right) + \sum_{j \in Y_1} C_j = \sum_{i \in X \setminus X_1} P_i + \sum_{j \in Y_1} C_j$$

$$= \sum_{i \in X_2} P_i + \sum_{j \in Y_1} C_j \quad (*)$$

Với $X_2 = X \setminus X_1$, là tập những sản phẩm *không được chọn*. Vì $\sum_{i \in X} P_i$ là tổng tài trợ của tất cả sản phẩm, không thay đổi và không phụ thuộc vào việc chọn tập X_1, do đó việc tìm R cực đại, tương đương với bài toán tìm S cực tiểu, lúc đó ta có thể suy ra R bằng công thức: $\mathrm{R} = \sum_{i \in X} P_i - \mathrm{S}$.

Như vậy, bài toán gốc (tìm cực đại) đã được đưa về một bài toán tương đương (tìm cực tiểu). Bài toán mới này có thể được đưa về bài toán lát cắt cực tiểu bằng cách dựng mạng:

- Đỉnh phát s nối với tập đỉnh sản phẩm $X = \{x_1, x_2, \dots, x_N\}$ với độ thông qua (capacity) là p_{x_i} $(1 \le i \le N)$
- Tập đỉnh tài nguyên $Y = \{y_1, y_2, \dots, y_M\}$ nối với đỉnh thu t với độ thông qua là c_{y_j} $(1 \le j \le M)$
- Tập đỉnh sản phẩm X nối với tập đỉnh tài nguyên Y với các cung (x_i, y_j) mà sản phẩm x_i cần tài nguyên y_j, với độ thông qua là vô cùng.

Như vậy, ta có đồ thị G với tập đỉnh có $N + M + 2$ đỉnh là $V = \{s\} \cup X \cup Y \cup \{t\}$.

Bài toán trở thành, tìm một lát cắt qua các cạnh, chia đồ thị trên thành 2 tập đỉnh rời nhau (S, T) sao cho $s \in S, t \in T, S \cup T = V, S \cap T = \emptyset$.

Ta có các nhận xét về lát cắt (S, T) như sau:

- Vì độ thông qua của các cung từ X đến Y là vô cùng, nên không thể tồn tại cung bị cắt từ X đến Y, đồng nghĩa với nếu một đỉnh sản phẩm được chọn ($\in X_1$) thì các tài nguyên yêu cầu ($\in Y_1$) cũng sẽ được chọn.

- Chỉ tồn tại hai loại cung bị cắt là (s, x_i) hoặc (y_j, t). Nên đồ thị sẽ được chia thành hai tập, tập S chứa đỉnh nguồn s và tập các đỉnh sản phẩm được chọn (cùng các đỉnh tài nguyên mà các sản phẩm này yêu cầu, như đã giải thích ở ý trên), tập T chứa đỉnh thu t và tập các tài nguyên không được chọn (cùng các đỉnh sản phẩm không được chọn). Nói cách khác, cung nối từ S đến T chỉ có thể là (s, x_i) với $x_i \in X_2$ và (y_j, t) với $y_j \in Y_1$.

- Giá trị của lát cắt (S, T) bằng $\sum_{i \in X_2} P_i + \sum_{j \in Y_1} C_j$, cũng chính là kết quả $(*)$ của bài toán cần tìm cực tiểu.

Với những phân tích trên, bài toán tóm lại chính là tìm giá trị của lát cắt cực tiểu trên mạng G. Để tìm lát cắt cực tiểu, ta có định lý quan trọng sau: *giá trị của luồng cực đại bằng trọng số của lát cắt cực tiểu* (tham khảo thêm tại đây). Do đó ta chỉ cần tìm giá trị của luồng cực đại trên G.

Để tìm luồng cực đại trên đồ thị, ta có nhiều thuật toán để giải quyết (tham khảo ở đây). Đối với giới hạn bài toán này, $N, M \leq 1000$, ta cần thuật toán đủ nhanh như Dinic để có thể được điểm tuyệt đối.

Về việc xây dựng lại danh sách sản phẩm và tài nguyên được chọn, trên lát cắt cực tiểu tìm được, nếu cung (s, x_i) không bị cắt thì sản phẩm x_i được chọn, nếu cung (y_j, t) bị cắt thì tài nguyên y_j được sử dụng. Nói cách khác, các đỉnh x_i đến được từ s là các sản phẩm được chọn, và các đỉnh y_j đến được từ s sẽ là các tài nguyên cần sử dụng.

BÀI B: BRIDGE BUILDING

Solution:

C++	https://ideone.com/srX2aP

Tóm tắt đề:

Cho 1 dãy a có độ dài là n, tìm cách xếp lại dãy a sao cho $\sum_1^{n-1}\sqrt{1+(a_i+a_{i+1})^2}$ là nhỏ nhất

Input:

- Dòng đầu tiên là số nguyên n ($1 \le n \le 10^5$)
- Dòng thứ hai chứa n số nguyên dương $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^4$)

Output:

- In ra duy nhất 1 số thực tổng chiều dài nhỏ nhất của các tấm gỗ
- Câu trả lời sẽ được chấp nhận nếu sai số $\le 10^{-6}$

Ví dụ:

2 3 4	1.414214

Hướng dẫn giải:

Subtask 1:

$$n \le 15, a_i \le 100$$

Sử dụng thuật toán backtracking để tìm tất cả các chỉnh hợp có thể của dãy, từ đó tính $\sum_1^{n-1}\sqrt{1+(a_i+a_{i+1})^2}$ và lưu lại kết quả nhỏ nhất.

Subtask 2:

$$n \le 10^5, a_i \le 10^4$$

Ta cần tìm min của $\sum_1^{n-1}\sqrt{1+(a_i+a_{i+1})^2}$.

Hàm $\sum_1^{n-1}\sqrt{1+(a_i+a_{i+1})^2}$ là một dạng của khoảng cách Euclidean giữa hai điểm (i, a_i) và $(i+1, a_{i+1})$.

Ta cần tìm cách xếp sao cho các $|a_i - a_{i+1}|$ được tối ưu.

Sắp xếp tăng dần: Các phần tử nhỏ hơn luôn đứng trước các phần tử lớn hơn. Do đó, sự thay đổi giữa hai giá trị liên tiếp sẽ là nhỏ nhất có thể trong toàn bộ dãy số.

Vậy ta chỉ cần sort lại dãy a theo tăng dần hoặc giảm dần và tính $\sum_1^{n-1} \sqrt{1 + (a_i + a_{i+1})^2}$

Độ phức tạp: O(nlogn)

BÀI C: DÃY BẰNG KHÔNG

Solution:

C++	https://ideone.com/DYX6Tq

Tóm tắt đề:

Cho dãy gồm ***N*** số nguyên $A_1, A_2,...,A_N$. Có ***Q*** truy vấn, mỗi truy vấn gồm 2 số nguyên ***L, R***.

Yêu cầu: Với mỗi truy vấn, in ra số lượng đoạn con liên tiếp nhiều nhất trong đoạn ***[L, R]*** mà tổng các số trong mỗi đoạn con đó bằng **0**.

Lưu ý các đoạn con không được giao nhau.

Input:

Dòng đầu gồm hai số nguyên dương ***N***, ***Q*** ($1 \le N, Q \le 10^6$) - độ dài dãy số và số lượng truy vấn.

Dòng tiếp theo gồm N số nguyên A_i ($-10^6 \le A_i \le 10^6$).

Q dòng tiếp theo, mỗi dòng gồm một cặp số nguyên ***L, R*** ($1 \le L \le R \le N$).

Output:

Gồm Q dòng, mỗi dòng gồm một số nguyên là số lượng đoạn con liên tiếp nhiều nhất trong đoạn ***[L, R]*** có tổng bằng 0.

Ví dụ:

5 4 -2 2 -2 8 -6 2 5 1 5 1 4	1 2 1 1

1 2	

Giải thích:

Trong ví dụ, dãy gồm 5 phần tử và 4 truy vấn:

- Với truy vấn thứ nhất gồm đoạn [2, -2, 8, -6], đoạn con có tổng bằng 0 là [[2, -2], 8, -6] hoặc [2, [-2, 8, -6]]
- Với truy vấn thứ hai gồm đoạn [-2, 2, -2, 8, -6], số lượng được có tổng bằng 0 lớn nhất là [[-2, 2], [-2, 8, -6]]
- Với truy vấn thứ ba gồm đoạn [-2, 2, -2, 8], đoạn con có tổng bằng 0 là [[-2, 2], -2, 8] hoặc [-2, [2, -2], 8]
- Với truy vấn cuối cùng gồm đoạn [-2, 2], có duy nhất chính nó có tổng là 0

Hướng dẫn giải:

Với mỗi truy vấn, ta duyệt từ ***L*** sang ***R*** hoặc từ ***R*** sang ***L***, sau đó tìm đoạn con trái nhất hoặc phải nhất có tổng bằng 0. Từ vị trí kết thúc của đoạn con trước đó, ta tiếp tục tìm đoạn con trái nhất hoặc phải nhất có tổng bằng 0 tiếp theo.

Cứ như vậy đến hết đoạn ***[L, R]***. Tuy nhiên với cách này ta cần duyệt qua hết đoạn ***[L, R]***. Độ phức tạp có thể lên đến ***O(N * Q)***

Để dễ dàng lập trình thì ta sẽ chọn duyệt từ ***R*** sang ***L***. Ta chuẩn bị trước, với mỗi i ta lưu vị trí bắt đầu dãy con có tổng bằng 0 phải nhất. Gọi vị trí đó là pos[i], thì tồn tại vị trí j (pos[i] ≤ j ≤ i) sao cho tổng đoạn [pos[i], j] bằng 0, và trong đoạn [pos[i], j] không còn chứa đoạn con bằng 0 nào khác.

Với việc chuẩn bị trước như vậy, mỗi khi xử lí truy vấn thì với từ vị trí ***R*** ta có thể nhảy tới vị trí ***pos[R],*** xong từ đó nhảy tiếp đến khi tới ***L***. Với mỗi lần nhảy như vậy ta sẽ được một đoạn con có tổng bằng 0. Tuy nhiên trong trường hợp tệ thì có nhiều đoạn con có tổng bằng 0 thì ta sẽ thực hiện nhảy nhiều lần, làm độ phức tạp sẽ không đủ thời gian cho phép.

Để tối ưu số lần nhảy, ta sử dụng kỹ thuật **Binary Lifting**, mỗi lần nhảy sẽ nhảy 2^x lần. Với mỗi truy vấn ta chỉ nhảy tối đa $log_2(R - L + 1)$

Độ phức tạp: $O((N + Q)log(N))$

BÀI D: BE CAREFUL

Solution:

Python	Code Python Nộp bằng Pypy để tốc độ chạy nhanh hơn: https://ideone.com/za4WuT
C++	https://ideone.com/D1QJxF https://ideone.com/WjI2WO

Tóm tắt đề:

Cho một số N được tạo từ hàm $MakeNumber$ có mã giả như sau.

```
function isPrime(N):
    // Hàm dùng để kiểm tra N có phải số nguyên tố không.


function MakeNumber(S, K):
    N = 1
    while K > 0:
        #                 vv lấy ngẫu nhiên giữa 1 và 100.
        if isPrime(S) and random_int(1, 100) == 1:
            K -= 1
            N *= S
        S += 1
    return N
```

Trong đó, hàm $random\_int$ là hàm ngẫu nhiên phân phối đều.

Yêu cầu: Cho thông tin về hai thông số S, K được truyền vào, phân tích thừa số nguyên tố N.

Input:

Dòng đầu là hai số nguyên S, K.

Dòng thứ hai chứa số nguyên N.

Output:

In ra các thừa số nguyên tố của số nguyên N theo thứ tự tăng dần. Biết rằng giá trị của các thừa số nguyên tố trong đáp án sẽ không vượt quá $9{,}223{,}372{,}036{,}854{,}775{,}807$.

Ví dụ:

123 2 62291	167 373

Hướng dẫn giải:

Bài toán này trước hết cần tiếp cận phần tạo ra số nguyên N. Tuy có một phần xác suất của một số nguyên tố được lấy là 1%. Nhưng vì khoảng cách giữa các số nguyên tố không có chênh lệch quá lớn, nên thừa số nguyên tố cuối cùng với điểm bắt đầu S sẽ không quá 10^6.

Vì vậy, chúng ta có thể phân tích thừa số nguyên tố bằng cách cho giá trị S tăng dần và kiểm tra liệu đó là ước nguyên tố của N hay không. Nhưng không thể chỉ đơn thuần kiểm tra bằng modulo số lớn cho số nhỏ không vì chi phí cho việc tính modulo khá lớn trong (C/C++). Nên một vài cách để hạn chế là:

Cách 1: Miller Rabin.

- Chúng ta có thể cài đặt thuật toán Miller Rabin để kiểm tra số S là số nguyên tố hay không, sau đó thực hiện kiểm tra N có chia hết cho S hay không. Như thế hạn chế được các số S không phải là nguyên tố dư thừa.
- Như vậy, để kiểm tra Miller Rabin nhanh mình có thể dùng các hệ cơ số a (base witness) cụ thể để kiểm tra cho mọi số $S \leq 2^{64} - 1$.

Cách 2: BigNum

- Một cách khác để giải quyết vấn đề này chính là các template BigNum để tính toán số lớn, cụ thể hai phép tính ở đây chính là chia lấy nguyên và chia lấy dư (số lớn cho số nhỏ).

Vì phép tính chia lấy nguyên và dư cũng tương đối lớn, nên sẽ cần một vài cải tiến nho nhỏ như: S phải luôn lẻ, S là số có dạng $6k \pm 1$, … để bỏ qua những TH S chắc chắn không phải là nguyên tố cần tìm. Các cải tiến nho nhỏ này cũng có thể áp dụng với Miller-Rabin trên.

BÀI E: MAKE UP

Solution:

C++	https://ideone.com/R7k6U0

Tóm tắt đề:

Cho n sự kiện và m cách trang điểm. Mỗi cách trang điểm có 2 thông số là c_j và d_j với ý nghĩa cách trang điểm này có thể giúp Ngân tham gia được những sự kiện có thời điểm bắt đầu và kết thúc là a_i và b_i sao cho $c_j \leq a_i \leq b_i \leq d_j$. Hãy tìm một trong m cách trang điểm mà có thể giúp Ngân tham gia được nhiều sự kiện nhất.

Input:

- Dòng đầu tiên là 2 số nguyên n, m ($1 \leq n, m \leq 10^5$)
- n dòng tiếp theo mỗi dòng chứa n cặp số nguyên dương a_i, b_i ($1 \leq a_i, b_i \leq 10^9$)
- m dòng tiếp theo mỗi dòng chứa m cặp số nguyên dương c_j, d_j ($1 \leq c_j, d_j \leq 10^9$)

Output:

- In ra duy nhất 1 số nguyên dương là số sự kiện tối đa Ngân có thể tham gia được.

Ví dụ:

4 1 1 3 1 2 3 3 4 5 1 3	3

Hướng dẫn giải:

Như yêu cầu của đề, ta có thể thử từng cách trang điểm rồi đếm xem với cách trang điểm ta đang thử ta có thể tham gia được tối đa bao nhiêu sự kiện, sau đó lấy cách trang điểm giúp ta tham gia được nhiều sự kiện nhất. Độ phức tạp thời gian là O($n * m$).

Dựa theo cách tiếp cận trên, ta có thể thấy rằng ta có nên tối ưu độ phức tạp thời gian của việc đếm số sự kiện tham gia được của từng cách trang điểm.

Vì các sự kiện và cách trang điểm đều có 2 thông số thời gian nên ta không thể truy vấn trên đoạn chúng một cách thông thường. Ta sẽ sắp xếp cả sự kiện và các cách trang điểm theo thự tự giảm dần về thời gian, sau đó với mỗi cách trang điểm ta sẽ lưu lại những sự kiện i mà có a_i lớn hơn thời gian bắt đầu của cách trang điểm ta đang xét, sau đó cập nhật chúng trên một CTDL truy vấn trên đoạn. Sau khi kết thúc quá trình cập nhật số sự kiện tham gia được cho mỗi cách trang điểm, ta sẽ truy vấn tổng trên đoạn để lấy ra được số sự kiện mà cách trang điểm ta đang xét có hiệu quả.

Độ phức tạp: **O(mlogn)**

---------- **HẾT** ----------