Thuật toán và ứng dụng - TICHPX

Created: August 11, 2025 7:15 AM

Thuật toán và Ứng dụng

Chương 0. Tổng quan

1. Thuật toán

Khái niệm:

Thuật toán là dãy các thao tác biến đổi dữ liệu đầu vào thành đầu ra sau hữu hạn bước.

Ví dụ: Tính số Fibonacci

- Quy ước:
  - `F0 = 1`, `F1 = 1`
  - `Fn = F(n-1) + F(n-2)` với `n ≥ 2`

Cách 1: Chia để trị - đệ quy thuần

```
long long Fib(int n) {
    return n >= 2 ? Fib(n-1) + Fib(n-2) : 1;
}
```

Cách 2: Đệ quy có nhớ *memoization*

```
#include <map>
using namespace std;

map<int,long long> F = {{0,1},{1,1}};

long long Fib(int n) {
    if (F.find(n) != F.end()) return F[n];
```

```
    return F[n] = Fib(n-1) + Fib(n-2);
}
```

Cách 3: Quy hoạch động $bottom - up$

```
long long Fib(int n) {
    long long F[n+5] = {1,1};
    for (int i = 2; i <= n; i++) {
        F[i] = F[i-1] + F[i-2];
    }
    return F[n];
}
```

Cách 4: Dùng chứng minh quy nạp toán học, ma trận

Định lý:

$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^n = \begin{bmatrix}F_n & F_{n-1}\\ F_{n-1} & F_{n-2}\end{bmatrix}$$

Ví dụ implement nhanh Fibonacci:

```
void Fib(int n, long long &a, long long &b) {
    if (n == 0 || n == 1) {
        a = 1;
        b = 0;
        return;
    }
    long long x, y;
    Fib(n/2, x, y);
    a = x*x + y*y;
    b = 2*x*y - y*y;

    if (n % 2) {
        a = a + b;
        b = a - b;
    }
}
```

Bài tập: Tổng dãy Fibonacci

Cho dãy Fibonacci:

$$\{ F_1 = F_2 = 1,\ F_n = F_{n-1} + F_{n-2} \quad (n \geq 3)$$

Yêu cầu tính $S = F_1 + F_2 + \cdots + F_n$, kết quả lấy dư $10^9 + 9$.

Ràng buộc

- $1 \leq n \leq 10^9$

Input

- Một dòng duy nhất chứa số nguyên dương n.

Output

- Một số nguyên là $S \bmod 1\,000\,000\,009$.

Ví dụ

Input	Output
`5`	`12`
`100`	`724844561`

Ý tưởng

- Tổng `S = F_1 + … + F_n` có thể biến đổi thành `F_{n+1} - F_0 - F_1` nhờ tính chất `F_{k+1} = F_k + F_{k-1}` ⇒ chỉ cần biết `F_{n+1}` là ra kết quả.
- Dùng thuật toán đệ quy để tính nhanh `F_{n+1}` theo modulo `10^9+9`.

Các bước

1. Dùng đệ quy để nhận `(F_{n+1}, F_n)` với mọi phép toán đều mod `10^9+9`.
2. Tính `S = F_{n+1} - 2` (vì `F_0 = F_1 = 1`). Chuẩn hóa về modulo dương.
3. In kết quả `S mod 10^9+9`.

Biết:

- `F(n+1) = Fn + F(n-1)`
- Suy ra `Fn = F(n+1) - F(n-1)`

Do đó:

`S = F1 + F2 + ... + Fn`

`S = (F2 - F0) + (F3 - F1) + (F4 - F2) + ... + (F(n+1) - F(n-1))`

`S = F(n+1) - F0 - F1`

`S = F(n+1) - 2`

```
def Fibo(n):
    if n==0: return 1, 0
    x, y = Fibo(n//2)
    x, y = (x*x + y *y)%1000000009, (x*y+y*(x-y))%1000000009
    if n%2==0: return x%1000000009, y%1000000009
    return (x+y)%1000000009 , x%1000000009

if __name__ == '__main__':

    n = int(input())
    a, b = Fibo(n+1)
    print(a-1)
```

Bài tập: Tổng Fibonacci - mod 1e9+7

Xét dãy Fibonacci (F_n) với:

$$\{ F_1 = F_2 = 1,\ F_n = F_{n-1} + F_{n-2} \quad (n \geq 3).$$

Cho số nguyên n, hãy tính $S = F_1 + F_2 + \cdots + F_n$ lấy dư cho $10^9 + 7$.

Input

- Một số nguyên n. Dùng biến 64-bit khi nhập.

Output

- Một số nguyên duy nhất là $S \bmod 1\,000\,000\,007$.

Constraints

- Subtask 1: n thuộc phạm vi nhỏ ví dụ $(n \leq 10^5)$.
- Subtask 2: n lớn hơn, cần tối ưu hơn.
- Subtask 3: n tới giới hạn bài toán đủ lớn để cần $log(n)$.

Ví dụ

Input	Output
`4`	`7`

Ý tưởng

- Giống bài trước, tổng `S` rút gọn về `F_{n+2} - 1` (hoặc `F_{n+1} - 2` tùy quy ước), nên chỉ cần tính một giá trị Fibonacci lớn rồi trừ đi hằng số.
- Áp dụng đệ quy để tính `F_{n+1}` trong `O(log n)` và làm việc modulo `10^9+7`.

Các bước

1. Viết hàm `Fibo` đệ quy trả về `(F_n, F_{n-1})` (hoặc `(F_{n+1}, F_n)`) dưới modulo `10^9+7`.
2. Gọi hàm với `n+1`, lấy `F_{n+1}`.
3. Tính `S = (F_{n+1} - 1) mod 10^9+7` (hoặc `F_{n+1} - 2` theo triển khai), in ra kết quả.

```
MOD = 1_000_000_007

def Fibo(n):
    if n == 0:
        return 1, 0
    x, y = Fibo(n // 2)
    x, y = (x*x + y*y) % MOD, (x*y + y*(x - y)) % MOD
    if n % 2 == 0:
        return x, y
    return (x + y) % MOD, x

if __name__ == '__main__':
    n = int(input())
```

```
    a, _ = Fibo(n + 1)
    print((a - 1) % MOD)
```

CHƯƠNG 2: Python

1. Nhập xuất trong Python

- Nhập từ **1 dòng**: `input()`
- Nhập từ **nhiều dòng**: `sys.stdin.read()`

2. Các lệnh điều khiển

2.1. Lệnh `if`

```
if bt_logic:
    A

if a > b:
    max = a
    min = b

if bt_logic:
    A
else:
    B

if Bt1:
    A1
elif Bt2:
    A2
elif Bt3:
    A3
# ...

A if BT else B
```

- Chú ý: Biểu thức logic khác 0 → `True`, ngược lại `False`

- Toán tử logic: `not`, `and`, `or`
- So sánh chuỗi điều kiện:
  - `a < x < b`
  - `a == b == c` được chấp nhận trong Python

Ví dụ: Ghép 2 hình chữ nhật thành hình vuông

Cho hai hình chữ nhật cạnh `a, b` và `c, d`. Hỏi có ghép thành hình vuông được không?

```
a, b = map(int, input().split())
c, d = map(int, input().split())

if a < b:
    a, b = b, a
if c < d:
    c, d = d, c

print("yes" if a == c == b + d else "no")
```

Ví dụ: Biện luận phương trình bậc 2 `ax^2 + bx + c = 0`

```
a, b, c = map(float, input().split())
b /= -2
d = b*b - a*c

if a == b == c == 0:
    print("vo so nghiem")
elif (a == b == 0) or d < 0:
    print("vo nghiem")
elif a == 0:
    print("%.3f" % (c/2/b))
elif d == 0:
    print("%.3f" % (b/a))
else:
    d = d**0.5
    x1 = (b - d) / a
    x2 = (b + d) / a
```

```
    if x1 > x2:
        x1, x2 = x2, x1
    print("%.3f\n%.3f" % (x1, x2))
```

Ý tưởng

- Dựa vào giá trị `a` và biệt thức `d = b^2 - a*c` để xác định số nghiệm.
- Xem xét các trường hợp đặc biệt khi hệ số bằng 0 để kết luận chính xác.

Các bước

1. Nhập `a, b, c`, chuẩn hóa `b` theo công thức đang dùng.
2. Tính `d = b*b - a*c`.
3. Kiểm tra tuần tự các trường hợp: vô số nghiệm, vô nghiệm, phương trình bậc nhất, nghiệm kép, hoặc hai nghiệm phân biệt.
4. Với hai nghiệm, tính `x1`, `x2`, sắp thứ tự trước khi in.

Ví dụ: Số Armstrong - 3 chữ số

```
x = input()
a, b, c = int(x[0]), int(x[1]), int(x[2])
print("yes" if int(x) == a**3 + b**3 + c**3 else "no")
```

Ý tưởng

- Một số Armstrong 3 chữ số thỏa: tổng lập phương của từng chữ số bằng chính số đó.
- Chỉ cần tách 3 chữ số và so sánh tổng.

Các bước

1. Đọc chuỗi `x` gồm 3 ký tự số.
2. Chuyển từng ký tự thành số, tính `a^3 + b^3 + c^3`.
3. So sánh với `int(x)` và in `yes` hoặc `no`.

2.2. Vòng lặp `for`

`range`:

- `range(n)` → `0 … n-1`
- `range(start, end, step=1)`
- `range(1, n+1)` → `1 … n`

Ví dụ: Giai thừa số lớn

```
n = int(input())
s = 1
for i in range(1, n+1):
    s *= i
print(s)
```

Ví dụ: Đếm số ước của n giai thừa

Cho số nguyên dương `n`, hãy tính số ước nguyên dương của `n!`. Vì kết quả có thể rất lớn, chỉ in phần dư khi chia cho `10^9 + 7`.

Input

- Một dòng duy nhất chứa số nguyên dương `n`.

Output

- Một số nguyên là `d(n!) mod 1_000_000_007`, với `d(x)` là số ước nguyên dương của `x`.

Ví dụ

Input	Output
`4`	`8`
`1000`	`972926972`

Giải thích: `4! = 24`, các ước là `{1, 2, 3, 4, 6, 8, 12, 24}` ⇒ tổng cộng 8 ước.

```
def sang(n):
    s = [1] * (n + 1)
    for i in range(2, n + 1):
```

```
        if s[i]:
            for j in range(i * i, n + 1, i):
                s[j] = 0
    return s

if __name__ == '__main__':
    MOD = 10**9 + 7
    n = int(input())
    primes = sang(n)
    res = 1
    for p in range(2, n + 1):
        if primes[p]:
            exp = 1
            m = n
            while m > 0:
                m //= p
                exp += m
            res = (res * exp) % MOD
    print(res)
```

Ý tưởng

- Biết rằng nếu `n! = ∏ p_i^{e_i}` thì số ước `d(n!) = ∏ (e_i + 1)`.
- Tính `e_i` bằng cách cộng `⌊n/p⌋ + ⌊n/p^2⌋ + …` cho từng số nguyên tố `p`.

Các bước

1. Dùng sàng để xác định các số nguyên tố ≤ `n`.
2. Với mỗi nguyên tố, tính số mũ xuất hiện trong `n!` bằng cách chia liên tiếp cho `p`.
3. Nhân `res = res * (exp + 1) mod 1e9+7`.
4. In `res`.

Ví dụ: Tích cấp số cộng

Cho số nguyên dương `n` và số `k` thỏa `0 ≤ k < n`. Xét các số nguyên dương không vượt quá `n` và có cùng số dư với `n` khi chia cho `k`, hay tương đương là dãy:

`n, n-k, n-2k, …` *lấy đến khi còn dương.*

Hãy tính tích của toàn bộ dãy số này.

Input

- Hai số nguyên `n` và `k`.

Output

- Một số nguyên dương là tích cần tìm.

Ví dụ

Input	Output
`100 10`	`36288000000000000`
`100 11`	`1099886703552000`

Ý tưởng

- Các phần tử cần nhân tạo thành cấp số cộng lùi với công sai `k`.
- Duyệt giảm dần từ `n` về `>0`, bước `k`, nhân dồn kết quả.

Các bước

1. Đọc `n, k`.
2. Khởi tạo `s = 1`.
3. Lặp `i` từ `n` xuống `>0`, mỗi lần `i -= k`, nhân `s *= i`.
4. In `s`.

```
n, k = map(int, input().split())
s = 1
for i in range(n, 0, -k):
    s *= i
print(s)
```

List và tuple

- List: dùng ngoặc vuông `[]`
- Tuple: dùng ngoặc tròn `()`

```
x = [4, 7, 2, 8]
y = (4, 7, 2, 8)
```

Nhập và tách:

```
x = list(map(int, input().split()))
print(x)

n, m, *y = x
print(n)
print(m)
print(y)
```

Đảo list:

```
y = x[::-1]
```

Ví dụ: Đảo chuỗi ký tự

```
s = input()
print(s[::-1])
```

Ví dụ: Đảo thứ tự từ

```
x = input().split()
y = x[::-1]
print(*y)
```

Ví dụ:

Nhập `n, a1, ..., an`.

Tính `S = a1 + (a1+a2) + (a1+a2+a3) + …`

```
n = int(input())
a = list(map(int, input().split()))

a = a[::-1]
s = 0
for i, x in enumerate(a, 1):
    s += i * x
print(s)
```

---

map

```
# map(f, [a0, a1, ..., an]) → [f(a0), f(a1), ..., f(an)]

n = int(input())
a = range(1, n+1)
b = map(lambda x: x*x, a)
print(sum(b))
```

---

zip

```
x = [1, 2, 3]
y = [4, 5, 6]
z = list(zip(x, y))       # [(1, 4), (2, 5), (3, 6)]
t = list(zip(x, y, x[::-1]))
```

Ví dụ: Tổng 3 đa thức

```
n = int(input())
a = list(map(int, input().split()))

m = int(input())
b = list(map(int, input().split()))

k = int(input())
```

```
c = list(map(int, input().split()))

p = max(m, n, k)
a = a + [0]*(p - n)
b = b + [0]*(p - m)
c = c + [0]*(p - k)

d = list(map(lambda x, y, z: x + y + z, a, b, c))

print(*d)
```

Ý tưởng

- Để cộng 3 đa thức, cần căn chỉnh số hạng bậc cao nhất bằng cách thêm các hệ số 0.
- Sau khi đồng bộ độ dài, cộng từng hệ số tương ứng.

Các bước

1. Đọc chiều dài và hệ số của 3 đa thức.
2. Xác định độ dài lớn nhất `p`, bù 0 cho đa thức ngắn hơn.
3. Dùng `map` $hoặc vòng lặp$ để cộng từng vị trí.
4. In kết quả.

Ví dụ: Kiểm tra dãy không giảm `a1 ≤ a2 ≤ … ≤ an`

```
n = int(input())
a = list(map(int, input().split()))

d = sum(1 for x, y in zip(a, a[1:]) if x <= y)
print("yes" if d == n-1 else "no")
```

Ý tưởng

- Dãy không giảm khi mọi cặp liền nhau đều thỏa `a_i ≤ a_{i+1}`.
- Đếm số cặp hợp lệ rồi so sánh với `n-1`.

Các bước

1. Đọc `n` và danh sách `a`.

2. Dùng `zip(a, a[1:])` để duyệt từng cặp liền kề.
3. Đếm số cặp mà `x ≤ y`.
4. Nếu đếm = `n-1` → `yes`, ngược lại `no`.

`filter` + list comprehension

```
x = [4, 7, 2, 8, 3, 7, 2, 1]
y = [t for t in x if t % 2 != 0]
print(y)
```

Cú pháp `for` với `else`

```
x = [4, 8, 6, 4]
for t in x:
    if t % 2:
        print(t)
else:
    print("khong so le")
```

2.3. Vòng lặp `while`

Cú pháp:

```
while bt_logic:
    A

while bt_logic:
    A
else:
    B
```

Ví dụ: Đếm số số 0 tận cùng của n!

Ý tưởng: đếm số lần xuất hiện của thừa số 5 trong n!

```
n = int(input())
d = 0
for i in range(5, n+1, 5):
    while i % 5 == 0:
        i //= 5
        d += 1
print(d)
```

CHƯƠNG 3: Lập trình hướng chức năng và đối tượng

I. Hướng chức năng

1. Hàm

- Kiểu trả về linh hoạt
- Hàm có thể trả về nhiều giá trị
- Có giá trị `None` khi không trả về gì

Ví dụ: Đếm số nghiệm trùng phương

Cho phương trình trùng phương:

$$ax^4 + bx^2 + c = 0$$

với các hệ số thực `a, b, c`. Đếm số nghiệm thực phân biệt của phương trình $nghiệm kép tính một lần$. Nếu có vô số nghiệm, in `-1`.

Input

- Một dòng gồm ba số thực `a b c`, cách nhau ít nhất một khoảng trắng.

Output

- Một số nguyên là số nghiệm thực phân biệt, hoặc `-1` nếu vô số nghiệm.

Ví dụ

Input	Output
`0.0 0.0 0.0`	`-1`
`1 -4 0`	`3`

Ý tưởng

- Phương trình trùng phương đặt `y = x^2` để về dạng bậc hai `a y^2 + b y + c = 0`.
- Tính nghiệm theo `y`, sau đó với mỗi `y` xác định số nghiệm thực của `x` bằng hàm `dem`.
- Xử lý các trường hợp đặc biệt (`a = b = c = 0`, vô nghiệm, bậc nhất) trước.

Các bước

1. Chuẩn hóa phương trình, tính biệt thức `d = b^2 - a c`.
2. Phân loại: vô số nghiệm, vô nghiệm, bậc nhất, bậc hai.
3. Với mỗi nghiệm `y`, gọi `dem(y)` để đếm số nghiệm `x`.
4. In tổng số nghiệm tìm được.

```
def dem(t):
    if t < 0:
        return 0
    if t == 0:
        return 1
    return 2

if __name__ == "__main__":
    a, b, c = map(float, input().split())

    b /= 2
    d = b*b - a*c
    if a == b == c == 0:
        print(-1)
    elif a == b == 0 or d < 0:
        print(0)
    elif a == 0:
        print(dem(-c/2/b))
    elif d == 0:
```

```
        print(dem(-b/a))
    else:
        x1 = (-b - d**0.5) / a
        x2 = (-b + d**0.5) / a
        print(dem(x1) + dem(x2))
```

Ví dụ: Giải và biện luận phương trình trùng phương

```
def dem(t):
    if t < 0:
        return 0
    if t == 0:
        return 1
    return 2

if __name__ == "__main__":
    a, b, c = map(float, input().split())

    b /= 2
    d = b*b - a*c
    if a == b == c == 0:
        print("vo so nghiem")
    elif a == b == 0 or d < 0:
        print("vo nghiem")
    else:
        if a == 0:
            res = dem(-c/2/b)
        elif d == 0:
            res = dem(-b/a)
        else:
            res = dem((-b + d**0.5)/a) + dem((-b - d**0.5)/a)
        if res == []:
            print("vo nghiem")
        else:
            for x in res:
                print("%.3f" % x)
```

Ý tưởng

- Tương tự ví dụ trên nhưng cần in cụ thể nghiệm tùy trường hợp.
- Hàm `dem` trợ giúp để biết số nghiệm của mỗi nghiệm trung gian.

Các bước

1. Tính `d = b^2 - a c` và xét các trường hợp đặc biệt.
2. Khi còn nghiệm, xác định giá trị `y` rồi dùng `dem` để suy ra nghiệm `x`.
3. Nếu không thu được nghiệm thực, in `vo nghiem`; nếu có, in từng nghiệm với định dạng `%.3f`.

Ví dụ: Uống bia chai

Bố cu Ấm có `n` đồng, giá mỗi chai bia là `m`. Quán đang khuyến mại: cứ đủ `k` nắp chai thì được đổi thêm 1 chai mới. Sau một ngày chạy xe, anh ấy muốn biết tổng cộng uống được bao nhiêu chai bao gồm cả các chai đổi từ nắp.

Input

- Dòng đầu: số bộ test `t`.
- Mỗi dòng tiếp theo: ba số nguyên `n, m, k` (`n` là số tiền, `m` giá một chai, `k` số nắp cần để đổi 1 chai).

Output

- Với mỗi test, in ra tổng số chai có thể uống được.

Ví dụ

Input	Output
`1` `52 4 3`	`19`

Giải thích ví dụ: mua `52 / 4 = 13` chai ⇒ 13 nắp. Đổi 12 nắp lấy 4 chai ⇒ 17 chai, còn 5 nắp. Đổi tiếp 3 nắp lấy 1 chai ⇒ 18 chai, còn 3 nắp. Đổi thêm lấy 1 chai nữa ⇒ 19 chai, nắp còn lại không đủ đổi thêm.

Ý tưởng

- Số chai ban đầu = `n // m`. Sau khi uống, thu được số nắp tương ứng để đổi.

- Liên tục đổi `k` nắp lấy 1 chai mới, mỗi lần đổi lại sinh thêm nắp mới -> giống bài toán đổi vỏ chai.

Các bước

1. Với mỗi test, tính `res = n // m`, `nap = res`.
2. Khi `nap ≥ k`, đổi lấy chai mới: `res += nap // k`, cập nhật `nap = nap % k + nap // k`.
3. In `res`.

```
def sol():
    m, n, k = map(int, input().split())
    res = m // n
    nap = res
    while nap >= k:
        res += nap // k
        nap = nap % k + nap // k
    print(res)

if __name__ == "__main__":
    t = int(input())
    for _ in range(t):
        sol()
```

Ví dụ: Tính số ngày

Ngân hàng cần tính số ngày giữa hai mốc thời gian kể cả ngày gửi, không tính ngày rút . Biết rằng các ngày nhập vào đều hợp lệ và nằm trong khoảng `1930 ≤ y1 ≤ y2 ≤ 2020`.

Input

- Dòng 1: `d1 m1 y1` – ngày gửi tiền.
- Dòng 2: `d2 m2 y2` – ngày rút tiền.

Output

- Một số nguyên không âm là số ngày từ ngày gửi đến trước ngày rút.

Ví dụ

Input	Output
`8 3 2017` `8 3 2017`	`0`
`7 5 1954` `30 4 1975`	`7663`
`28 2 2000` `1 3 2000`	`2`

Lưu ý: Năm nhuận có thêm ngày 29/2, xác định bằng: chia hết cho 400 hoặc chia hết cho 4 nhưng không chia hết cho 100.

Ý tưởng

- Quy đổi mỗi ngày sang số ngày tuyệt đối từ mốc ví dụ $1/1/0001$, sau đó lấy hiệu.
- Có thể tự viết hàm `totalDays` hoặc dùng thư viện `datetime`.

Các bước

1. Viết hàm kiểm tra năm nhuận và hàm cộng số ngày theo từng năm/tháng.
2. Tính tổng ngày của `(d2, m2, y2)` và `(d1, m1, y1)`, lấy hiệu.
3. Hoặc sử dụng `datetime.date` để trừ trực tiếp và lấy `days`.

```
#1
def isLeapYear(y):
    return y % 400 == 0 or (y % 4 == 0 and y % 100 != 0)

def totalDays(d, m, y):
    s = sum(365 + isLeapYear(i) for i in range(1, y))
    t = [0, 31, 28 + isLeapYear(y), 31, 30, 31, 30, 31, 31, 30, 31, 3
    return s + sum(t[:m]) + d

if __name__ == '__main__':
    d1, m1, y1 = map(int, input().split())
    d2, m2, y2 = map(int, input().split())

    print(totalDays(d2, m2, y2) - totalDays(d1, m1, y1))
```

```
#2
import datetime

d1, m1, y1 = map(int, input().split())
d2, m2, y2 = map(int, input().split())

s = datetime.date(y2, m2, d2) - datetime.date(y1, m1, d1)
print(s.days)
```

Ví dụ: Tính chất đơn điệu của dãy số

Xác định tính chất của một dãy số nguyên tối thiểu 3 phần tử theo các quy tắc:

- `Day don dieu tang ngat`: $a_1 < a_2 < \cdots < a_n$
- `Day don dieu tang`: $a_i \leq a_{i+1}$ với ít nhất một cặp bằng nhau
- `Day don dieu giam ngat`: $a_1 > a_2 > \cdots > a_n$
- `Day don dieu giam`: $a_i \geq a_{i+1}$ với ít nhất một cặp bằng nhau
- `Day bang nhau`: mọi phần tử bằng nhau
- `Day khong don dieu`: các trường hợp còn lại

Input

- Dòng 1: số nguyên dương `n` (số phần tử, `n ≥ 3`).
- Dòng 2: `n` số nguyên không âm.

Output

- Một chuỗi mô tả tính chất của dãy như trên.

Ví dụ

Input	Output
`5` `1 3 6 9 15`	`Day don dieu tang ngat`
`5` `1 3 3 6 9`	`Day don dieu tang`

Ý tưởng

- Dùng hàm `check(a, f)` để kiểm tra toàn bộ dãy thỏa điều kiện nào đó.
- Lần lượt kiểm tra các tính chất theo thứ tự ưu tiên; nếu không thuộc tính chất nào thì kết luận “không đơn điệu”.

Các bước

1. Đọc `n` và danh sách `a`.
2. Định nghĩa hàm `check` và `count` hỗ trợ.
3. Với phiên bản #1: gọi `check` với các so sánh tương ứng cho từng tính chất.
4. Với phiên bản #2: dùng `count` để xác định sự tồn tại của quan hệ `<`, `>`, `=` và lập bảng kết luận.

```
#1
def check(a, f):
    return all(f(x, y) for x, y in zip(a, a[1:]))

if __name__ == '__main__':
    n = int(input())
    a = list(map(int, input().split()))
    if check(a, lambda x, y: x < y):
        print("Day don dieu tang ngat")
    elif check(a, lambda x, y: x > y):
        print("Day don dieu giam ngat")
    elif check(a, lambda x, y: x == y):
        print("Day bang nhau")
    elif check(a, lambda x, y: x >= y):
        print("Day don dieu giam")
    elif check(a, lambda x, y: x <= y):
        print("Day don dieu tang")
    else:
        print("Day khong don dieu")
```

```
#2
def count(a, f):
    return any(f(x, y) for x, y in zip(a, a[1:]))

if __name__ == '__main__':
    n = int(input())
```

```
    a = list(map(int, input().split()))
    t = count(a, lambda x, y: x < y)
    g = count(a, lambda x, y: x > y)
    b = count(a, lambda x, y: x == y)
    s = t*4 + g*2 + b
    kl = [
        "Day it hon 2 phan tu",
        "Day bang nhau",
        "Day don dieu giam ngat",
        "Day don dieu giam",
        "Day don dieu tang ngat",
        "Day don dieu tang",
        "Day khong don dieu",
        "Day khong don dieu"]
    print(kl[s])
```

Ví dụ: Dãy gần cấp số cộng

Dãy số nguyên `a1, a2, ..., an` là cấp số cộng nếu tồn tại `d` sao cho `a_{i+1} - a_i = d` với mọi `i`. Dãy được gọi là “gần cấp số cộng” nếu chỉ cần thay **đúng một** phần tử để trở thành cấp số cộng.

Input

- Dòng 1: số nguyên dương `n`.
- Dòng 2: `n` số nguyên có giá trị tuyệt đối ≤ `10^9`.

Output

- Nếu dãy đã là cấp số cộng: in `Day la day cap so cong`.
- Nếu dãy gần cấp số cộng: in `Day la gan cap so cong tai vi tri k` - vị trí cần thay đổi.
- Ngược lại: in `Day khong la day gan cap so cong`.

Ví dụ

Input	Output
`5` `1 2 3 4 5`	`Day la day cap so cong`
`5` `1 2 3 -6 5`	`Day la gan cap so cong tai vi tri 4`
`5` `4 7 2 8 1`	`Day khong la day gan cap so cong`

```
def csc(a):
    if len(a) < 2:
        return False
    d = a[1] - a[0]
    for i in range(len(a) - 1):
        if a[i+1] - a[i] != d:
            return False
    return True

if __name__ == "__main__":
    n = int(input())
    a = list(map(int, input().split()))

    if csc(a):
        print("Day la day cap so cong")
    elif csc(a[1:]):
        print("Day la gan cap so cong tai vi tri 1")
    elif csc(a[:-1]):
        print("Day la gan cap so cong tai vi tri %d" % n)
    else:
        for i in range(1, n-1):
            if (a[i-1] + a[i+1]) % 2 == 0:
                x = (a[i-1] + a[i+1]) // 2
                y = a[i]
                a[i] = x
                if csc(a):
                    print("Day la gan cap so cong tai vi tri %d" % (i
                    break
                a[i] = y
```

```
        else:
            print("Day khong la day gan cap so cong")
```

Ví dụ: Chăn bò

Toto đứng ở gốc tọa độ `(0, 0)` quan sát các con bò đặt ở các điểm nguyên trên mặt phẳng. Hai con bò nằm trên cùng một tia xuất phát từ gốc - cùng hướng so với Toto thì chỉ con đứng gần gốc hơn là nhìn thấy. Hỏi Toto nhìn được bao nhiêu con.

Ý tưởng

- Hai điểm cùng hướng nếu tọa độ của chúng tỉ lệ với nhau. Chuẩn hóa bằng cách chia cho `gcd(|x|, |y|)`.
- Số hướng khác nhau chính là số con có thể thấy.

Các bước

1. Với mỗi con bò `(x, y)`, tính `d = gcd(|x|, |y|)` và chuẩn hóa thành `(x/d, y/d)`.
2. Đưa các cặp chuẩn hóa vào tập `set`.
3. Kết quả là kích thước của tập.

Input

- Dòng 1: số nguyên dương `n` – số con bò.
- `n` dòng tiếp theo: mỗi dòng là hai số nguyên `x y` (tọa độ của một con bò, không trùng gốc, `|x|, |y| ≤ 1000`).

Output

- Một số nguyên là số con bò Toto nhìn thấy.

Ví dụ

Input	Output

`6`
`3 3`
`7 7`
`4 6`
	`3`

Input	Output

`-1 -1`
`6 9`
`3 3`	

Giải thích: các con ở `(3,3)`, `(4,6)` và `(-1,-1)` hiển thị; các con khác bị che bởi con nằm cùng hướng gần hơn.

```
from collections import namedtuple
import math

diem = namedtuple("Diem", "x,y")
n = int(input())
A = set()
for _ in range(n):
    x, y = map(int, input().split())
    d = abs(math.gcd(x, y))
    A.add(diem(x//d, y//d))
print(len(A))
```

Ví dụ: Trao giải chung kết marathon

Khoa Điện - Điện tử tổ chức cuộc thi lập trình với sự tham gia của sinh viên nhiều khoa. Kết thúc cuộc thi, cần tìm top 3 sinh viên thuộc khoa DDT $điểmcaonhất$ để trao giải chính, và chọn thêm 1 sinh viên thuộc các khoa còn lại $CNTT, CK, CT, KTXD$ có điểm cao nhất cho giải giao lưu.

Ý tưởng

- Tách danh sách thí sinh thành hai nhóm: DDT và ngoài DDT.
- Sắp xếp nhóm DDT theo điểm giảm dần lấy 3 người đầu; nhóm còn lại tìm người có điểm lớn nhất.

Các bước

1. Đọc `n`, duyệt từng sinh viên, bỏ vào danh sách `D` hoặc `K`.
2. Sắp xếp `D` giảm dần theo điểm.
3. Dùng `max` để lấy sinh viên điểm cao nhất trong `K`.
4. In tên theo thứ tự giải.

Input

- Dòng 1: số nguyên dương `n` – tổng số thí sinh.
- `n` dòng tiếp theo: mỗi dòng chứa `Họ tên` $có khoảng trắng$, `diem` $nguyên dương < 1000$ và `khoa` (một trong `DDT, CNTT, CK, CT, KTXD`).

Output

- In 4 dòng:
  - `Giai nhat :<tên>` - sinh viên DDT điểm cao nhất
  - `Giai nhi :<tên>`
  - `Giai ba :<tên>`
  - `Giai giao luu :<tên>` - sinh viên không thuộc DDT, điểm cao nhất

Ví dụ

Input $rút gọn$	**Output**

`18` ... _danh sách như đề bài_	`Giai nhat :Tran Ba Quang`
	`Giai nhi :Nguyen Huu Manh`
	`Giai ba :Do Van Manh`
	`Giai giao luu :Nguyen Duc Thanh`

```
from collections import namedtuple

sv = namedtuple("sv", "ten diem")

if __name__ == "__main__":
    n = int(input())
    D = []
    K = []
    for i in range(n):
        ten, diem, khoa = input().rsplit(" ", 2)
        if khoa == "DDT":
            D.append(sv(ten, int(diem)))
        else:
            K.append(sv(ten, int(diem)))
    D.sort(key=lambda x: x.diem, reverse=True)
    k = max(K, key=lambda x: x.diem)
    print("Giai nhat :%s" % D[0].ten)
```

```
    print("Giai nhi :%s" % D[1].ten)
    print("Giai ba :%s" % D[2].ten)
    print("Giai giao luu :%s" % k.ten)
```

Ví dụ: Kiểm tra xem có hai điểm trùng nhau không

Cho `n` điểm có tọa độ nguyên trên mặt phẳng. Hãy kiểm tra xem có tồn tại ít nhất một cặp điểm trùng nhau không.

Input

- Dòng 1: số nguyên dương `n`.
- `n` dòng tiếp theo: mỗi dòng gồm hai số nguyên `x y` (`|x|, |y| ≤ 10^9`) là tọa độ của một điểm.

Output

- In `YES` nếu tồn tại hai điểm trùng nhau, ngược lại in `NO`.

Ví dụ

Input	Output

`4`
`1 2`
`-3 4`
`1 2`
`3 5`
	`YES`

`4`
`1 2`
`-3 4`
`1 -2`
`3 5`
	`NO`

```
from collections import namedtuple

diem = namedtuple("Diem", "x,y")
n = int(input())
```

```
A = set()
for i in range(n):
    x, y = map(int, input().split())
    A.add(diem(x, y))
print("NO" if len(A) == n else "YES")
```

Ý tưởng

- Dùng `set` để lưu các điểm duy nhất; nếu có điểm trùng, kích thước set < `n`.

Các bước

1. Đọc `n` và từng tọa độ.
2. Thêm vào set `A`.
3. So sánh `len(A)` với `n` rồi in `YES` nếu có trùng, `NO` nếu không.

Ví dụ: Tính điểm thi lập trình

Cho `n` sinh viên tham gia thi 5 bài `A, B, C, D, E`. Mỗi bài có hệ số điểm lần lượt là 1, 2, 3, 4, 5. Dữ liệu đầu vào gồm một dòng tiêu đề chứa các cột `TEN, A, B, C, D, E` nhưng thứ tự các cột có thể bị đảo. Hãy tính tổng điểm của từng sinh viên và in ra danh sách theo thứ tự tên tăng dần.

Input

- Dòng 1: số nguyên `n`.
- Dòng 2: tên các cột (hoán vị của `TEN, A, B, C, D, E`, phân tách bởi khoảng trắng hoặc tab).
- `n` dòng tiếp theo: thông tin mỗi sinh viên theo đúng thứ tự cột vừa nêu.

Output

- In danh sách `tên + điểm` sắp xếp tăng dần theo tên.

Ví dụ

Input $rútgọn$	**Output** $rútgọn$
`13` `C D A TEN B E`	`Bac 52` `Bang 75`

Input *rútgọn*	Output *rútgọn*
`...`	`...`

`13`
`A E D B C TEN`
`...`
	`Bac 86`
	`Bang 118`
	`...`

```
from collections import namedtuple

n = int(input())
sv = namedtuple("sv", input() + " S")
A = []

for i in range(n):
    a, b, c, d, e, f = input().split()
    x = sv(a, b, c, d, e, f, 0)
    t = int(x.A) + int(x.B)*2 + int(x.C)*3 + int(x.D)*4 + int(x.E)*5
    y = x._replace(S=t)
    A.append(y)

A.sort(key=lambda x: x.TEN)
for x in A:
    print(x.TEN, x.S)
```

Ý tưởng

- Mỗi hàng có thể bị đảo thứ tự cột, nên dùng `namedtuple` với tiêu đề động để truy cập theo tên.
- Tính điểm có trọng số rồi sắp xếp theo tên.

Các bước

1. Đọc `n` và dòng tiêu đề, khởi tạo `namedtuple`.
2. Với mỗi sinh viên, tạo bản ghi, tính điểm `A + 2B + ... + 5E`.
3. Lưu lại vào danh sách.
4. Sắp xếp theo `TEN` và in `TEN, S`.

Ví dụ: Rút gọn phân số *dạng phân số liên tiếp*

Cho dãy số nguyên dương `a1, a2, ..., an`. Hãy rút gọn phân số dạng phân số liên tiếp:

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots + \cfrac{1}{a_n}}}$$

Input

- Dòng 1: số nguyên dương `n` (`1 < n < 100`).
- Dòng 2: `n` số nguyên dương `a1, ..., an` (`1 ≤ ai ≤ 100`).

Output

- Phân số tối giản sau khi rút gọn biểu thức trên, in dạng `t/m`.

Ví dụ

Input	Output
`3` `1 2 3`	`10/7`

```
import math

class ps:
    def __init__(self, _t=0, _m=1):
        self.t = _t
        self.m = _m
    def __add__(self, other):
        return ps(self.t*other.m + self.m*other.t, self.m*other.m)
    def __str__(self):
        return str(self.t) + "/" + str(self.m)
    def rg(self):
        d = abs(math.gcd(self.t, self.m))
        if self.m < 0:
            d = -d
        self.t //= d
```

```
        self.m //= d
    def nghichdao(self):
        return ps(self.m, self.t)

if __name__ == '__main__':
    n = int(input())
    a = list(map(int, input().split()))[::-1]
    p = ps(a[0], 1)
    for x in a[1:]:
        q = p.nghichdao()
        p = ps(x, 1) + q
    p.rg()
    print(p)
```

Ý tưởng

- Phân số liên tiếp xử lý từ cuối về đầu: mỗi bước nghịch đảo kết quả hiện tại rồi cộng phần tử tiếp theo.
- Dùng lớp phân số để quản lý tử/mẫu và rút gọn cuối cùng.

Các bước

1. Đảo dãy `a` để bắt đầu từ phần tử cuối.
2. Khởi tạo `p = a_0/1`.
3. Lặp qua phần tử còn lại: `p = a_i + 1/p`.
4. Sau cùng rút gọn phân số và in dạng `t/m`.

Chương 3. Các cấu trúc dữ liệu trong Python

1. Ngăn xếp $Stack$ – `LifoQueue`

Khái niệm:

Cấu trúc dữ liệu động, tuyến tính, phần tử thêm / bớt ở cùng một đầu, theo nguyên tắc **LIFO** $LastInFirstOut$.

- `S.put(x)` → thêm vào
- `S.get()` → lấy ra
- `S.queue[-1]` → xem phần tử trên đỉnh mà không lấy

Ví dụ: Đổi sang cơ số 3

Tito cần chuyển một số nguyên dương từ hệ 10 sang hệ 3. Hãy biểu diễn từng số theo cơ số 3 bằng cách chia lấy dư liên tiếp hoặc dùng stack để đảo ngược các chữ số.

Input

- Dòng đầu: số bộ test `t` (`t ≤ 10^5`).
- `t` dòng tiếp theo: mỗi dòng là một số nguyên dương `n < 10^9`.

Output

- Với mỗi `n`, in một dòng là biểu diễn của `n` trong cơ số 3 (không có số 0 dẫn đầu, trừ khi `n = 0` thì in `0`).

Ví dụ

Input	Output

`5`
`354`
`551`
`536`
`561`
`459`
	`111010`
	`202102`
	`201212`
	`202210`
	`122000`

Ý tưởng

- Chia liên tiếp cho 3 để lấy phần dư $chữsốcơsố3$ theo thứ tự ngược.
- Dùng ngăn xếp để đảo thứ tự phần dư khi in ra.

Các bước

1. Với mỗi test, đọc `n`.
2. Nếu `n = 0`, in `0`; ngược lại, lặp chia `n` cho 3, push phần dư vào stack.
3. Pop tất cả phần tử để in thành chuỗi cơ số 3.

```
from queue import LifoQueue

def cs3():
    n = int(input())
```

```
    L = LifoQueue()
    while n > 0:
        L.put(n % 3)
        n //= 3
    while not L.empty():
        print(L.get(), end="")
    print()

if __name__ == '__main__':
    for _ in range(int(input())):
        cs3()
```

Ví dụ: Kiểm tra ngoặc đúng

Cho chuỗi chỉ gồm ký tự trong `{}`, `[]`, `()`. Cần kiểm tra xem chuỗi có hợp lệ theo quy tắc:

- Không có ngoặc nhọn bên trong ngoặc vuông hoặc tròn.
- Không có ngoặc vuông bên trong ngoặc tròn.
- Các cặp ngoặc phải đóng mở đúng thứ tự.

Input

- Dòng 1: số bộ test `t`.
- `t` dòng tiếp theo: mỗi dòng là một chuỗi ngoặc.

Output

- Với mỗi test, in `Dung` nếu chuỗi hợp lệ, ngược lại in `Sai`.

Ví dụ

Input	Output

`2`
`()()`
`(()`
	`Dung`
	`Sai`

`2`
`([]())`
`{[()]}`
	`Sai`
	`Dung`

Ý tưởng

- Dùng stack để đảm bảo ngoặc đóng trùng khớp với ngoặc mở gần nhất.
- Cần kiểm soát thêm thứ tự ưu tiên của ngoặc - ngoặc nhọn không nằm trong ngoặc vuông/tròn.

Các bước

1. Tạo stack `L` và map `C` cho cặp ngoặc, map `D` cho độ ưu tiên.
2. Duyệt từng ký tự: nếu là ngoặc mở, kiểm tra thứ tự ưu tiên rồi push ngoặc đóng mong đợi.
3. Nếu là ngoặc đóng, kiểm tra đúng với phần tử trên đỉnh; nếu sai → “Sai”.
4. Sau khi duyệt, nếu stack rỗng → “Dung”, ngược lại “Sai”.

```
from queue import LifoQueue

def check(x):
    L = LifoQueue()
    D = {'{':3, '[':2, '(':1, '}':3, ']':2, ')':1}
    C = {'{':'}', '[':']', '(':')'}

    for c in x:
        if c in C.keys():
            if L.qsize() and D[L.queue[-1]] < D[c]:
                return "Sai"
            L.put(C[c])
        elif c in C.values():
            if L.empty() or L.queue[-1] != c:
                return "Sai"
            L.get()
    return "Dung" if L.empty() else "Sai"

if __name__ == '__main__':
    for i in range(int(input())):
        print(check(input()))
```

Ví dụ: Khối lượng hóa chất

Cho công thức hóa học hữu cơ chỉ gồm các ký tự `C`, `H`, `O`, dấu ngoặc `()` và các số $chỉsố$ từ 2 đến 9. Khối lượng nguyên tử: `C=12`, `H=1`, `O=16`. Hãy tính khối lượng phân tử của chất đó.

Input

- Dòng 1: số bộ test `t`.
- Mỗi dòng tiếp theo: một chuỗi công thức $độdài \leq 500$.

Output

- Với mỗi công thức, in khối lượng phân tử (số nguyên không vượt quá `10^9`).

Ví dụ

Input	Output

`2`
`((CHOH)2(CO2H)2H2O)4`
`CH3((CHCO2H)2(CH2)8CO2H)3H2CO3H`
	`672`
	`897`

Ý tưởng

- Dùng stack để xử lý ngoặc: `(` đánh dấu bắt đầu nhóm, khi gặp `)` thì cộng các giá trị trong nhóm.
- Khi gặp số, nhân khối lượng trên đỉnh stack với số đó.

Các bước

1. Duyệt từng ký tự công thức:
   - Nếu là `C/H/O`, push khối lượng tương ứng.
   - Nếu là `(`, push 0 - đánh dấu.
   - Nếu là `)`, pop đến khi gặp 0 để tính tổng nhóm.
   - Nếu là số, nhân phần tử trên đỉnh với số.
2. Tổng khối lượng = tổng các phần tử còn lại trong stack.

```
from queue import LifoQueue

def hoachat(x):
    L = LifoQueue()
    K = {'(': 0, 'C':12, 'H':1, 'O':16}
```

```
    for c in x:
        if c in K.keys():
            L.put(K[c])
        elif c == ')':
            s = 0
            while L.queue[-1] != 0:
                s += L.get()
            L.queue[-1] = s
        else:
            L.queue[-1] *= int(c)
    return sum(L.queue)

if __name__ == '__main__':
    for i in range(int(input())):
        print(hoachat(input()))
```

Ví dụ: Xếp hàng - đếm số cặp nhìn thấy nhau

Trong hàng thẳng, hai người được xem là “nhìn thấy nhau” nếu giữa họ không có ai cao hơn một trong hai người đó - tức là chiều cao dãy ở giữa không vượt quá chiều cao thấp hơn trong cặp. Đếm tổng số cặp như vậy.

Input

- Dòng 1: số nguyên dương `n` số người.
- Dòng 2: `n` chiều cao (nguyên dương, ≤ `10^9`).

Output

- Một số nguyên là số cặp người nhìn thấy nhau.

Ví dụ

Input	Output
`6` `7 6 5 1 4 2`	`6`
`8` `4 7 2 8 4 4 4 6`	`14`

Ý tưởng

- Dùng stack để lưu các cặp {chiều cao, số lần xuất hiện}. Khi gặp người cao hơn, pop các người thấp hơn và cộng số cặp họ nhìn thấy.
- Nếu chiều cao bằng nhau, cộng thêm số cặp giữa họ và xử lý số lượng.

Các bước

1. Duyệt từng chiều cao:
    - While top thấp hơn hiện tại: pop và cộng số cặp = count.
    - Nếu top bằng nhau: cộng count + $stack > 1$ rồi tăng count.
    - Nếu top cao hơn: cộng 1 vì nhìn thấy top và push mới.
2. Tổng kết quả `res` sau vòng lặp.

Giải thích ví dụ 2: xét từng người nhìn về phía sau, ta có tổng 14 cặp như mô tả trong đề.

```
from queue import LifoQueue

S = LifoQueue()
n = int(input())
res = 0
a = list(map(int, input().split()))
for x in a:
    while S.qsize() and S.queue[-1][0] < x:
        res += S.queue[-1][1]
        S.get()
    if S.qsize() and S.queue[-1][0] == x:
        res += S.queue[-1][1] + (S.qsize() > 1)
        S.queue[-1][1] += 1
    else:
        res += (S.qsize() > 0)
        S.put([x, 1])
print(res)
```

Ví dụ: Bài toán tìm phần tử gần nhất - "Chào đón tân sinh viên K59"

Cho dãy chiều cao sinh viên `a`. Với mỗi vị trí `i`, cần tìm chỉ số của sinh viên gần nhất có chiều cao lớn hơn `a[i]`. Nếu có hai phía thỏa mãn, lấy chỉ số nhỏ hơn bên trái. Nếu không tồn tại, in `-1`.

Input

- Dòng 1: `n` – số sinh viên.
- Dòng 2: `n` số nguyên dương – chiều cao từng sinh viên.

Output

- Một dòng gồm `n` số nguyên; vị trí `i` là chỉ số cần tìm cho sinh viên `i`.

Ví dụ

Input	Output
`6` `1 4 2 1 7 6`	`1 4 1 2 -1 4`

Ý tưởng

- Với mỗi vị trí, cần tìm sinh viên cao hơn gần nhất bên trái và bên phải. Dùng stack một chiều để xây dựng các mảng `L` và `R`.
- Kết hợp hai kết quả: chọn chỉ số gần hơn; nếu chỉ có một phía thì dùng phía đó.

Các bước

1. Viết hàm `func(a, idx)` dùng stack giảm dần để tìm chỉ số người cao hơn gần nhất.
2. Chạy `func` cho chiều trái→phải và phải→trái.
3. Với mỗi vị trí:
    - Nếu cả hai phía tồn tại: so sánh khoảng cách, chọn nhỏ hơn.
    - Nếu chỉ một phía: dùng giá trị đó; nếu không có: in `-1`.

```
from queue import LifoQueue

def func(a, idx):
    S = LifoQueue()
    S.put((2e9, -1))
    L = [0]*len(a)
    for i, x in zip(idx, a):
        while x >= S.queue[-1][0]:
            S.get()
        L[i] = S.queue[-1][1]
        S.put((x, i))
```

```
    return L

if __name__ == "__main__":
    n = int(input())
    a = list(map(int, input().split()))

    L = func(a, range(n))
    R = func(a[::-1], range(n-1, -1, -1))

    for i in range(n):
        if L[i] > -1 and R[i] > -1:
            print(L[i] if i - L[i] <= R[i] - i else R[i], end=" ")
        else:
            print(L[i] + R[i] + 1, end=" ")
```

---

2. Queue - Hàng đợi

Ví dụ: Bịt mắt bắt dê - Josephus

Trong trò chơi, có `n` người đứng vòng tròn và loại dần: bắt đầu từ người 1, đếm đến `k` thì người đó ra khỏi vòng. Việc đếm tiếp tục từ người kế tiếp cho tới khi còn lại 1 người. Hãy xác định chỉ số người cuối cùng.

Input

- Một dòng chứa hai số nguyên dương `n` và `k`.

Output

- Chỉ số người cuối cùng còn lại trong vòng.

Ví dụ

Input	Output
`13 3`	`13`
`13 4`	`5`
`200 10`	`163`

Ý tưởng

- Mô phỏng bài toán Josephus bằng queue hoặc cấu trúc vòng: liên tục xoay `k-1` người ra cuối và loại người thứ `k`.
- Lặp cho đến khi chỉ còn một người.

Các bước

1. Đọc `n, k`, khởi tạo queue chứa `1..n`.
2. Trong khi queue còn hơn 1 phần tử:
    - Di chuyển `k-1` phần tử từ đầu ra cuối.
    - Loại bỏ phần tử tiếp theo.
3. In phần tử còn lại.

Cách 1: Dùng `queue.Queue`

```
import queue

n, k = map(int, input().split())
Q = queue.Queue()
for i in range(1, n+1):
    Q.put(i)
while Q.qsize() > 1:
    for i in range(k-1):
        Q.put(Q.get())
    Q.get()
print(Q.get())
```

Cách 2: Dùng danh sách liên kết vòng

```
class Node:
    def __init__(self, e=0, N=None):
        self.elem = e
        self.next = N

def add(T, x):
    if T is None:
        T = Node(x, None)
        T.next = T
    else:
```

```
        T.next = Node(x, T.next)
        T = T.next
    return T

def remove(T):
    T.next = T.next.next
    return T

if __name__ == '__main__':
    n, k = map(int, input().split())
    T = None
    for i in range(1, n+1):
        T = add(T, i)
    for _ in range(1, n):
        for i in range(1, k):
            T = T.next
        remove(T)
    print(T.elem)
```

Ví dụ: Nhóm bạn

Trong buổi gala tổng kết, ban tổ chức muốn sắp những sinh viên quen nhau trực tiếp hoặc qua các mối quan hệ bắc cầu vào cùng một khu vực. Cho danh sách `m` cặp quen biết giữa `n` sinh viên được đánh số từ 1 đến `n`, hãy xác định:

- Có bao nhiêu khu vực - thành phần liên thông cần bố trí.
- Khu vực đông nhất có bao nhiêu sinh viên.

Input

- Dòng đầu: hai số nguyên dương `n m`.
- `m` dòng tiếp theo: mỗi dòng chứa hai số `u v` khác nhau, biểu thị quan hệ quen biết giữa `u` và `v`.

Output

- Dòng 1: số khu vực cần bố trí.
- Dòng 2: số sinh viên trong khu vực đông nhất.

Ví dụ

Input	Output

`10 6`
`8 9`
`3 5`
`4 5`
`3 4`
`1 7`
`2 7`
	`5`
	`3`

Ý tưởng

- Mỗi khu vực tương ứng một thành phần liên thông trong đồ thị quan hệ.
- Sử dụng BFS/DFS để đếm số thành phần và kích thước lớn nhất.

Các bước

1. Xây dựng danh sách kề cho đồ thị quen biết.
2. Khởi tạo mảng `visited`.
3. Với mỗi đỉnh chưa thăm, chạy BFS/DFS:
    - Tăng số nhóm.
    - Cập nhật kích thước lớn nhất.
4. In số nhóm và kích thước lớn nhất.

```
from queue import Queue

class Friend:
    def nhap(self):
        self.n, self.m = map(int, input().split())
        self.A = [[] for _ in range(self.n + 2)]
        for _ in range(self.m):
            x, y = map(int, input().split())
            self.A[x].append(y)
            self.A[y].append(x)
        self.visited = [0] * (self.n + 5)
        self.max_size = 0
        self.groups = 0

    def bfs(self, start, q):
        q.put(start)
```

```
        size = 1
        self.visited[start] = 1
        while q.qsize():
            u = q.get()
            for v in self.A[u]:
                if self.visited[v] == 0:
                    self.visited[v] = 1
                    size += 1
                    q.put(v)
        return size

    def sol(self):
        self.nhap()
        q = Queue()
        for i in range(1, self.n + 1):
            if self.visited[i] == 0:
                self.groups += 1
                self.max_size = max(self.max_size, self.bfs(i, q))
        print(self.groups)
        print(self.max_size)

if __name__ == "__main__":
    Friend().sol()
```

Ví dụ: Phản ứng hóa học - Y hệt bài Nhóm bạn

Nasus có `n` hóa chất, một số cặp trong đó phản ứng với nhau. Anh ấy muốn đổ lần lượt từng hóa chất vào ống nghiệm sao cho độ nguy hiểm cuối cùng là lớn nhất. Độ nguy hiểm bắt đầu từ 1; mỗi lần thêm một hóa chất nếu trong ống đã có chất phản ứng với nó thì độ nguy hiểm nhân đôi, ngược lại giữ nguyên. Hãy tìm độ nguy hiểm cực đại có thể đạt sau khi đổ tất cả hóa chất theo thứ tự tối ưu.

Input

- Dòng đầu: hai số nguyên `n m` (`1 ≤ n ≤ 50`, `0 ≤ m ≤ n(n-1)/2`).
- `m` dòng tiếp theo: mỗi dòng hai số `a b` (`1 ≤ a < b ≤ n`) biểu thị hai chất phản ứng với nhau.

Output

- Một số nguyên: độ nguy hiểm lớn nhất.

Ví dụ

Input	Output
`1 0`	`1`

`3 2`
`1 2`
`2 3`
	`4`

Ý tưởng

- Các chất phản ứng tạo thành các nhóm liên thông. Nếu nhóm có `size` chất, độ nguy hiểm góp phần `2^(size-1)`.
- Nhân tất cả các nhóm lại để được kết quả cuối.

Các bước

1. Dùng BFS/DFS để xác định kích thước từng thành phần liên thông.
2. Với mỗi kích thước `size`, nhân `res *= 2^(size-1)`.
3. In `res`.

```
from queue import Queue

class Friend:
    def nhap(self):
        self.n, self.m = map(int, input().split())
        self.A = [[] for _ in range(self.n + 2)]
        for _ in range(self.m):
            x, y = map(int, input().split())
            self.A[x].append(y)
            self.A[y].append(x)
        self.visited = [0] * (self.n + 5)
        self.res = 1

    def bfs(self, start, q):
        q.put(start)
        size = 1
        self.visited[start] = 1
```

```
        while q.qsize():
            u = q.get()
            for v in self.A[u]:
                if self.visited[v] == 0:
                    self.visited[v] = 1
                    size += 1
                    q.put(v)
        return size

    def sol(self):
        self.nhap()
        q = Queue()
        for i in range(1, self.n + 1):
            if self.visited[i] == 0:
                size = self.bfs(i, q)
                self.res *= 2 ** (size - 1)
        print(self.res)

if __name__ == "__main__":
    Friend().sol()
```

Ví dụ: Tìm đường đi ngắn nhất trong mê cung

Cho mê cung dạng ma trận `n x m` với giá trị `0` $có thể đi$ và `1` $chướng ngại$. Mèo Toto đứng tại ô `(sx, sy)` và chuột ở `(fx, fy)` (chuột luôn nằm trên ô `0`). Mỗi bước mèo chỉ được di chuyển 4 hướng $lên, xuống, trái, phải$ sang ô có giá trị `0`. Hãy tìm số bước ngắn nhất để mèo tới chuột, hoặc `-1` nếu không thể.

Input

- Dòng 1: hai số `n m`.
- `n` dòng tiếp theo: mỗi dòng `m` số `0/1`.
- Dòng cuối: 4 số `sx sy fx fy`

Output

- Một số nguyên: độ dài đường đi ngắn nhất; nếu không tồn tại, in `-1`.

Ví dụ

Input	Output

`3 3`
`0 1 0`
`0 0 0`
`1 0 0`
`1 1 3 3`
	`4`

`3 3`
`0 1 0`
`0 1 1`
`1 1 0`
`1 1 3 3`
	`-1`

Ý tưởng

- Mô hình hóa mê cung là đồ thị lưới; dùng BFS từ điểm xuất phát để tìm khoảng cách ngắn nhất đến điểm đích.
- Đánh dấu đã thăm để tránh vòng lặp.

Các bước

1. Bọc thêm viền 1 để tránh kiểm tra biên nhiều lần $tuỳtriểnkhai$.
2. Khởi tạo queue với điểm xuất phát, đánh dấu bước 1.
3. Trong khi queue chưa rỗng:
    - Pop ô hiện tại, thử 4 hướng.
    - Nếu ô kế hợp lệ (`0`), đánh dấu bước mới và push vào queue.
4. Sau BFS, kết quả là giá trị ở ô đích, hoặc `-1` nếu chưa được thăm.

```
from queue import Queue

n, m = map(int, input().split())
grid = [[1] * (m + 2)]
for _ in range(n):
    row = list(map(int, input().split()))
    row.append(1)
    grid.append([1] + row)
grid.append([1] * (m + 2))

q = Queue()
```

```
sx, sy, fx, fy = map(int, input().split())
q.put((sx, sy))
grid[sx][sy] = 1

while q.qsize() and grid[fx][fy] == 0:
    x, y = q.get()
    for dx, dy in [(-1, 0), (1, 0), (0, 1), (0, -1)]:
        if grid[x + dx][y + dy] == 0:
            grid[x + dx][y + dy] = grid[x][y] + 1
            q.put((x + dx, y + dy))

print(grid[fx][fy] - 1)
```

Ví dụ: Trinh thám

Amas quan sát hàng quân đỏ gồm `n` người, mỗi lần ống nhòm chỉ nhìn được `k` người liên tiếp. Khi trượt khung nhìn từ trái sang phải - mỗi lần dịch 1 vị trí, hãy báo cáo chiều cao lớn nhất trong khung ở mỗi vị trí.

Input

- Dòng 1: hai số nguyên `n k`.
- Dòng 2: `n` số nguyên là chiều cao các quân đỏ.

Output

- Một dòng gồm `n - k + 1` số nguyên: chiều cao tối đa trong từng khung nhìn.

Ví dụ

Input	Output
`8 3` `4 7 2 5 6 3 9 1`	`7 7 6 6 9 9`

Ý tưởng

- Dùng deque để lưu chỉ số phần tử theo thứ tự giảm dần.
- Đảm bảo đầu deque luôn là phần tử lớn nhất còn trong cửa sổ.

Các bước

1. Duyệt từng vị trí `i`:
   - Loại bỏ các chỉ số ở cuối deque có giá trị ≤ `a[i]`.
   - Thêm `i` vào cuối deque.
2. Nếu chỉ số đầu deque nằm ngoài cửa sổ hiện tại (`<= i-k`), pop khỏi đầu.
3. Khi `i ≥ k-1`, ghi nhận `a[dq[0]]` vào kết quả.

```
from collections import deque

n, k = map(int, input().split())
a = list(map(int, input().split()))

dq = deque()
res = []

for i in range(n):
    while dq and a[dq[-1]] <= a[i]:
        dq.pop()
    dq.append(i)

    if dq[0] <= i - k:
        dq.popleft()

    if i >= k - 1:
        res.append(a[dq[0]])

print(*res)
```

Ví dụ: Làm bóng tuyết

Alice mỗi ngày làm một quả cầu tuyết có khối lượng `v_i`. Ngày thứ `i` có nhiệt độ `t_i`, mọi quả cầu còn tồn tại đều tan đi `t_i` đơn vị khối lượng - quả nào tan hết sẽ biến mất. Hãy cho biết tổng khối lượng tuyết tan trong từng ngày.

Input

- Dòng 1: số ngày `n`.
- Dòng 2: `n` số nguyên `v_i`.

- Dòng 3: `n` số nguyên `t_i`.

Output

- Một dòng gồm `n` số nguyên: lượng tuyết tan mỗi ngày.

Ví dụ

Input	Output

`3`
`10 10 5`
`5 7 2`
	`5 12 4`

Ý tưởng

- Theo dõi các quả cầu còn tồn tại bằng hàng đợi ưu tiên chứa "khối lượng còn lại + offset".
- Mỗi ngày giảm tất cả quả cầu đang tồn tại một lượng `t_i`; những quả cầu tan hết được lấy ra khỏi PQ.

Các bước

1. Duyệt từng ngày:
   - Đẩy `v_i + offset` vào priority queue $min - heap$.
2. Trong khi phần tử nhỏ nhất sau khi trừ offset ≤ `t_i`, pop ra và cộng phần khối lượng tan.
3. Phần khối lượng còn lại tan trong ngày = `t_i * số quả còn lại trong PQ`.
4. Cộng offset bằng `t_i`, in tổng tan của ngày.

```
from queue import PriorityQueue

n = int(input())
V = list(map(int, input().split()))
T = list(map(int, input().split()))

pq = PriorityQueue()
offset = 0

for v, t in zip(V, T):
    pq.put(v + offset)
```

```
    melted = 0
    while pq.qsize() and pq.queue[0] - offset <= t:
        melted += pq.get() - offset
    melted += t * pq.qsize()
    print(melted, end=" ")
    offset += t
```

Ví dụ: Sắp xếp có điều kiện - theo phần dư 3

Cho dãy `n` số nguyên không âm. Hãy sắp xếp chúng theo quy tắc: tất cả số chia hết cho 3 $dư 0$ đứng trước và tăng dần, tiếp theo tới những số dư 1 - tăng dần, cuối cùng là các số dư 2 - tăng dần.

Input

- Dòng 1: `n` (`1 ≤ n ≤ 200`).
- Dòng 2: `n` số nguyên không âm.

Output

- Dãy sau khi sắp theo quy tắc, các số cách nhau một khoảng trắng.

Ví dụ

Input	Output
`12` `4 7 2 8 4 8 3 2 4 9 3 6`	`3 3 6 9 4 4 4 7 2 2 8 8`

Ý tưởng

- Cần sắp xếp theo khóa `(x % 3, x)` với `x % 3` tăng từ 0 → 2.
- Có thể dùng priority queue hoặc sort với custom key.

Các bước

1. Đọc `n` và danh sách.
2. Đưa các phần tử vào priority queue với toán tử `<` so sánh theo `(value % 3, value)`.
3. Pop lần lượt để tạo dãy đã sắp.

4. In kết quả.

```
from queue import PriorityQueue

class Item:
    def __init__(self, value):
        self.value = value
    def __lt__(self, other):
        if self.value % 3 == other.value % 3:
            return self.value < other.value
        return self.value % 3 < other.value % 3

if __name__ == '__main__':
    n = int(input())
    arr = list(map(int, input().split()))
    pq = PriorityQueue()
    for x in arr:
        pq.put(Item(x))
    res = []
    while pq.qsize():
        res.append(str(pq.get().value))
    print(" ".join(res))
```

Ví dụ: Lắp ghép ống nước

Có `n` đoạn ống với độ dài `a1, a2, …, an`. Máy nâng mỗi lần chỉ nối tối đa `k` đoạn lại thành một đoạn mới và chi phí chính là tổng độ dài của đoạn sau khi nối. Sau mỗi lần nối, đoạn mới được đưa trở lại để tiếp tục nối cho đến khi chỉ còn một đoạn duy nhất. Hãy tìm tổng chi phí nhỏ nhất.

Input

- Dòng 1: hai số nguyên `n k` (`1 < k < n ≤ 10^5`).
- Dòng 2: `n` số nguyên dương $\leq 3 * 10^4$ là độ dài các đoạn ống.

Output

- Một số nguyên: tổng chi phí tối thiểu.

Ví dụ

Input	Output
`3 2` `8 4 6`	`28`
`6 3` `8 2 1 5 2 3`	`39`

Ý tưởng

- Đây là tổng chi phí nối Huffman tổng quát: mỗi lần lấy `k` đoạn ngắn nhất ghép lại.
- Priority queue giúp luôn truy xuất nhanh các đoạn ngắn nhất.

Các bước

1. Đưa tất cả độ dài vào min-heap.
2. Trong khi PQ còn hơn 1 phần tử:
    - Pop tối đa `k` phần tử nhỏ nhất, cộng vào `t`.
    - Push `t` trở lại, cộng `t` vào `res`.
3. In `res`.

```
from queue import PriorityQueue

pq = PriorityQueue()
n, k = map(int, input().split())
for x in map(int, input().split()):
    pq.put(x)

res = 0
while pq.qsize() > 1:
    t = 0
    cnt = min(k, pq.qsize())
    for _ in range(cnt):
        t += pq.get()
    pq.put(t)
    res += t

print(res)
```

Ví dụ: Giao hàng

Titi có `n` món hàng, mỗi món cần giao trước một thời hạn `t_i` và sẽ nhận thưởng `v_i` nếu giao đúng hạn - mỗi đơn vị thời gian chỉ giao được một món. Hãy chọn thứ tự giao hàng để tổng tiền thưởng tối đa.

Input

- Dòng 1: số nguyên `n`.
- `n` dòng tiếp theo: mỗi dòng gồm `t_i` và `v_i`.

Output

- Một số nguyên: tổng thưởng lớn nhất.

Ví dụ

Input	Output

`6`
`3 5`
`3 7`
`1 3`
`2 4`
`2 2`
`4 1`
	`17`

Ý tưởng

- Bài toán chọn công việc có deadline: duyệt thời gian từ lớn xuống 1, mỗi lúc chọn công việc có thưởng cao nhất còn chưa làm.
- Dùng max-heap - thông qua lưu số âm để lấy thưởng lớn nhất tại mỗi thời điểm.

Các bước

1. Gom các công việc theo deadline vào mảng `jobs[deadline]`.
2. Duyệt `time` từ `MAX_T` xuống 1:
    - Đưa các công việc deadline = `time` vào PQ.
    - Nếu PQ không rỗng, pop phần thưởng cao nhất và cộng vào đáp án.
3. In tổng thưởng.

```
from queue import PriorityQueue

MAX_T = 10**5
jobs = [[] for _ in range(MAX_T + 1)]
n = int(input())
for _ in range(n):
    t, v = map(int, input().split())
    jobs[t].append(v)

pq = PriorityQueue()
ans = 0
for time in range(MAX_T, 0, -1):
    for reward in jobs[time]:
        pq.put(-reward)  # max-heap via negatives
    if pq.qsize():
        ans -= pq.get()

print(ans)
```

Ví dụ: Thuật toán mã hóa Huffman

Cho một xâu gồm các chữ cái in hoa tiếng Anh. Hãy tính tổng số bit sau khi mã hóa xâu theo thuật toán Huffman - mã tiền tố tối ưu.

Input

- Một xâu có độ dài không quá `10^5` ký tự (chỉ gồm `A` – `Z`).

Output

- Một số nguyên: số bit cần để mã hóa Huffman.

Ví dụ

Input	Output
`GIAOTHONGVANTAI`	`44`

Ý tưởng

- Huffman: mỗi lần lấy hai nút có tần suất nhỏ nhất ghép thành nút mới, tổng chi phí là tổng tần suất các ghép.

Các bước

1. Đếm tần suất từng ký tự, đẩy vào min-heap.
2. Trong khi PQ còn hơn 1 phần tử:
   - Pop 2 phần tử nhỏ nhất `x, y`.
   - Cộng `x + y` vào `res`, push `x + y` lại PQ.
3. In `res`.

```
from queue import PriorityQueue

s = input().strip()
freq = {}
for c in s:
    freq[c] = freq.get(c, 0) + 1

pq = PriorityQueue()
for f in freq.values():
    pq.put(f)

res = 0
while pq.qsize() > 1:
    x = pq.get()
    y = pq.get()
    total = x + y
    res += total
    pq.put(total)

print(res)
```

Ví dụ: Lại là / Dãy con liên tục khác biệt

Cho dãy `n` số nguyên không âm. Tìm độ dài tối đa của một dãy con liên tiếp trong đó không có hai phần tử giống nhau.

Input

- Dòng 1: `n`.

- Dòng 2: `n` số nguyên không âm ≤ 32767.

Output

- Độ dài lớn nhất của dãy con liên tiếp thỏa điều kiện.

Ví dụ

Input	Output
`12` `4 7 2 8 4 8 3 2 4 9 3 6`	`5`

Ý tưởng

- Sliding window: dùng map lưu vị trí cuối của từng giá trị. Nếu gặp phần tử trùng trong cửa sổ, dời biên trái.

Các bước

1. Khởi tạo `D = {}`, `L = 0`, `res = 0`.
2. Duyệt từng phần tử $index1-based$:
   - Nếu `x` đã thấy và `D[x] ≥ L`, cập nhật `L = D[x]`.
   - Cập nhật `D[x] = i`, `res = max(res, i - L)`.
3. In `res`.

```
D = {}
n = int(input())
a = list(map(int, input().split()))
L = 0
res = 0
for i, x in enumerate(a, 1):
    if x in D and D[x] >= L:
        L = D[x]
    D[x] = i
    res = max(res, i - L)
print(res)
```

Ví dụ: Dãy con liên tục có tổng cho trước

Nhập dãy `n` số nguyên và giá trị `T`. Hãy tìm độ dài lớn nhất của một dãy con liên tiếp có tổng bằng `T`.

Input

- Dòng 1: hai số `n` và `T` (`|T| ≤ 10^9`).
- Dòng 2: `n` số nguyên có trị tuyệt đối ≤ `10^9`.

Output

- Độ dài lớn nhất của dãy con liên tiếp có tổng `T`; nếu không tồn tại, in `0`.

Ví dụ

Input	Output
`10 9` `2 6 0 3 -3 5 4 0 9 8`	`6`

Ý tưởng

- Dùng prefix sum và dictionary lưu vị trí đầu tiên của mỗi tổng. Khi `s - T` xuất hiện trước đó, đoạn giữa có tổng `T`.

Các bước

1. Khởi tạo `D = {0:0}`, `s = 0`, `res = 0`.
2. Duyệt từng phần tử:
   - Cập nhật `s += x`.
   - Nếu `s - T` đã có trong `D`, cập nhật `res = max(res, i - D[s - T])`.
   - Nếu `s` chưa có, thêm `D[s] = i`.
3. In `res`.

```
D = {0: 0}
n, T = map(int, input().split())
a = list(map(int, input().split()))
s = 0
res = 0
for i, x in enumerate(a, 1):
    s += x
```

```
    L = D.get(s - T, i)
    res = max(res, i - L)
    if s not in D:
        D[s] = i
print(res)
```

Ví dụ: Cổ động viên cân bằng

Cho một chuỗi gồm các ký tự `X` cổ vũ đai xanh và `D` cổ vũ đai đỏ. Hãy tìm độ dài lớn nhất của một đoạn con liên tiếp mà số ký tự `X` bằng số ký tự `D`.

Input

- Một chuỗi độ dài ≤ `10^6`, gồm các ký tự `X` hoặc `D`.

Output

- Độ dài tối đa của đoạn con cân bằng (số `X` = số `D`). Nếu không có, in `0`.

Ví dụ

Input	Output
`XXDXDXX`	`4`

Ý tưởng

- Quy đổi `X` thành `+1`, `D` thành `-1`. Đoạn con cân bằng khi tổng tiền tố ở đầu và cuối bằng nhau.

Các bước

1. Chuyển chuỗi thành mảng `arr` gồm `1` và `-1`.
2. Khởi tạo `prefix = {0:0}`, `curr = 0`, `best = 0`.
3. Duyệt từng phần tử:
    - Cộng vào `curr`.
    - Nếu `curr` đã gặp trước đó, cập nhật `best` với độ dài tương ứng; nếu chưa, lưu vị trí vào `prefix`.
4. In `best`.

```
s = input().strip()
arr = [1 if c == 'X' else -1 for c in s]

prefix = {0: 0}
curr = 0
best = 0

for i, val in enumerate(arr, 1):
    curr += val
    prev = prefix.get(curr, i)
    best = max(best, i - prev)
    if curr not in prefix:
        prefix[curr] = i

print(best)
```

Ví dụ: Số gần may mắn

Một số nguyên dương được gọi là “gần may mắn” nếu nó chia hết cho một số may mắn, trong đó số may mắn là số chỉ gồm các chữ số `6` hoặc `8`. Cho số `n`, hãy kiểm tra xem `n` có phải số gần may mắn hay không.

Input

- Một dòng chứa số nguyên dương `n`.

Output

- In `YES` nếu `n` là số gần may mắn, ngược lại in `NO`.

Ví dụ

Input	Output
`12`	`YES`

Ý tưởng

- Một số gần may mắn nếu chia hết cho ít nhất một số toàn chữ số `6` hoặc `8`.

- Sinh trước danh sách các số may mắn nhỏ tới 3 chữ số là đủ cho ràng buộc gốc rồi thử chia.

Các bước

1. Hàm `is_lucky` kiểm tra một số chỉ gồm chữ số `6` / `8`.
2. Sinh danh sách số may mắn bằng cách duyệt phạm vi nhỏ $ví dụ 1..1000$.
3. Với mỗi số may mắn `lucky`, kiểm tra `n % lucky == 0`.
4. Nếu có, in `YES`, ngược lại `NO`.

```
def is_lucky(num):
    return all(ch in '68' for ch in str(num))

n = int(input())

lucky_numbers = []
for i in range(1, 1001):
    if is_lucky(i):
        lucky_numbers.append(i)

for lucky in lucky_numbers:
    if n % lucky == 0:
        print("YES")
        break
else:
    print("NO")
```

Ví dụ: Truy vấn max của đoạn con liên tiếp

Cho dãy `n` số nguyên và `m` truy vấn. Mỗi truy vấn cung cấp hai chỉ số `L, R` $1-based$, yêu cầu tìm giá trị lớn nhất trong đoạn con `a_L … a_R`.

Input

- Dòng 1: hai số nguyên `n m`.
- Dòng 2: `n` số nguyên (`|a_i| ≤ 10^9`).
- `m` dòng tiếp: mỗi dòng gồm hai số `L R`.

Output

- Với mỗi truy vấn, in ra giá trị lớn nhất trong đoạn `[L, R]`.

Ví dụ

Input	Output

`6 3`
`4 7 2 8 1 -6`
`1 5`
`2 3`
`3 6`
	`8`
	`7`
	`8`

Ý tưởng

- Xây segment tree để lưu giá trị lớn nhất trên mỗi đoạn con. Truy vấn `[L, R]` chia đôi đoạn đến khi trùng khớp các nút.

Các bước

1. Xây cây `Node(1, n+1)` và gọi `add` để đưa từng phần tử vào.
2. Hàm `add` cập nhật max từ lá lên gốc.
3. Hàm `get` kiểm tra xem `[L, R]` nằm hoàn toàn trong con trái/phải hay phải tách đôi.
4. Với mỗi truy vấn, in `get(root, L, R+1)`.

```
class Node:
    def __init__(self, u, v):
        self.A = u
        self.B = v
        self.elem = -10**18
        if u + 1 == v:
            self.left = None
            self.right = None
        else:
            mid = (u + v) // 2
            self.left = Node(u, mid)
            self.right = Node(mid, v)

def add(T, pos, val):
    T.elem = max(T.elem, val)
    if T.left is None:
```

```
        return
    if pos < T.left.B:
        add(T.left, pos, val)
    else:
        add(T.right, pos, val)

def get(T, L, R):
    if T.A == L and T.B == R:
        return T.elem
    if R <= T.left.B:
        return get(T.left, L, R)
    if L >= T.right.A:
        return get(T.right, L, R)
    return max(get(T.left, L, T.left.B), get(T.right, T.right.A, R))

if __name__ == "__main__":
    n, m = map(int, input().split())
    arr = list(map(int, input().split()))
    root = Node(1, n + 1)
    for idx, value in enumerate(arr, 1):
        add(root, idx, value)
    for _ in range(m):
        L, R = map(int, input().split())
        print(get(root, L, R + 1))
```

Ví dụ: Segment Tree

Ý tưởng

- Cây phân đoạn chia đoạn `[1, n]` thành các đoạn con, mỗi nút lưu giá trị max của đoạn của nó.
- Truy vấn max `[L, R]` bằng cách đi xuống các nút trùng khớp hoặc tách đôi đoạn.

Các bước

1. Xây cây `Node(1, n+1)`; mỗi nút lưu đoạn `[A, B)` và con trái/phải.
2. Hàm `add(T, pos, val)` cập nhật max trên đường từ lá `pos` lên gốc.
3. Hàm `get(T, L, R)`:
    - Nếu `[L, R)` trùng đoạn nút, trả `T.elem`.

- Nếu nằm hẳn trong trái/phải, gọi đệ quy tương ứng.
- Nếu cắt đôi, lấy max của hai phần.

4. Sau khi xây cây từ mảng ban đầu, trả lời từng truy vấn.

```
class Node:
    def __init__(self, u, v):
        self.A = u
        self.B = v
        self.elem = -10**18
        if u + 1 == v:
            self.left = None
            self.right = None
        else:
            mid = (u + v) // 2
            self.left = Node(u, mid)
            self.right = Node(mid, v)

def add(T, pos, val):
    T.elem = max(T.elem, val)
    if T.left is None:
        return
    if pos < T.left.B:
        add(T.left, pos, val)
    else:
        add(T.right, pos, val)

def get(T, L, R):
    if T.A == L and T.B == R:
        return T.elem
    if R <= T.left.B:
        return get(T.left, L, R)
    if L >= T.right.A:
        return get(T.right, L, R)
    return max(get(T.left, L, T.left.B), get(T.right, T.right.A, R))

if __name__ == "__main__":
    n, m = map(int, input().split())
    arr = list(map(int, input().split()))
    root = Node(1, n + 1)
    for idx, value in enumerate(arr, 1):
```

```
            add(root, idx, value)
        for _ in range(m):
            L, R = map(int, input().split())
            print(get(root, L, R + 1))
```

-