SIROCCO CUP 2026 / KC1

Không chuyên - Đề 1

Thứ Hai, 02/11/2026 · Online

180 phút · 4 bài · 400 điểm

Mô phỏng, số học và trạng thái

Bài	Tên bài	Tag
A	Hộ vệ	Số học, Kiểm tra nguyên tố
B	Bảng đen	Tìm kiếm nhị phân, Toán học, Gom nhóm
C	Mua kẹo	Hàng đợi, Mô phỏng, Phân tích khấu hao
D	Vực Sâu	Quy hoạch động, Mô phỏng trạng thái

Problemset trực tuyến
https://codeforces.com/gym/728947/problems

KC1-A / KHÁM PHÁ LỜI GIẢI

A. Hộ vệ

Số học | Kiểm tra nguyên tố

ĐIỂM MẤU CHỐT

Chỉ cần kiểm tra hai điều kiện độc lập: hiệu bằng 2 và cả hai số đều nguyên tố.

Từ quan sát đến thuật toán

Số nguyên tố phải lớn hơn 1. Nếu x là hợp số thì $x = u \times v$ với $2 \le u \le v$; khi đó $u^2 \le x$. Vì vậy, chỉ cần tìm một ước từ 2 đến căn bậc hai của x. Không tìm được ước nào thì x là nguyên tố.

Với cặp a < b, trước hết kiểm tra b - a = 2. Nếu điều kiện này sai, trả lời N ngay; nếu đúng, kiểm tra tính nguyên tố của cả a và b. Giới hạn mỗi số không quá 10^6 và chỉ tối đa 10 cặp khiến chia thử hoàn toàn đủ nhanh.

Ví dụ để tự kiểm tra

Cặp (3,5) đạt vì 3 và 5 đều nguyên tố. Cặp (2,3) không đạt dù cả hai đều nguyên tố, vì hiệu chỉ bằng 1. Cặp (9,11) cũng không đạt: hiệu bằng 2 nhưng 9 là hợp số.

KC1-A / CHỨNG MINH VÀ CÀI ĐẶT

Hộ vệ

Vì sao thuật toán đúng?

Nếu thuật toán trả lời Y, hai kiểm tra nguyên tố đều đúng và hiệu bằng 2, nên cặp thỏa đúng định nghĩa. Ngược lại, một cặp nguyên tố song sinh có hiệu bằng 2 và không số nào có ước từ 2 đến căn của nó; thuật toán chắc chắn trả lời Y. Hai chiều này bao quát mọi đầu vào.

Mã giả

```
NGUYEN_TO(x):
    nếu x < 2: trả về sai
    d = 2
    trong khi d <= x / d:
        nếu x chia hết cho d: trả về sai
        d = d + 1
    trả về đúng

CHO MỖI CẶP (a,b):
    nếu b-a = 2 và NGUYEN_TO(a) và NGUYEN_TO(b):
        in Y
    ngược lại: in N
```

Độ phức tạp

O(tvM) thời gian, O(1) bộ nhớ phụ, với M = max(b).

Các lỗi dễ gặp

- Không coi 1 là số nguyên tố; xử lý x < 2 trước vòng lặp.
- Phải kiểm tra cả hai số. Điều kiện hiệu bằng 2 chưa đủ.
- Điều kiện d ≤ x/d tránh tràn khi mở rộng giới hạn; trong bài hiện tại d^2 vẫn an toàn.

KC1-A / LỜI GIẢI TỪNG SUBTASK

Hộ vệ

Subtask	Điểm	Giới hạn bổ sung
1	20	$b \le 100$
2	30	$b \le 10^4$
3	50	Không có giới hạn bổ sung

Subtask 1 - 20 điểm

Thử mọi ước d từ 2 đến x-1 để kiểm tra từng số x. Xử lý x<2 là không nguyên tố. Sau đó kiểm tra đồng thời a,b nguyên tố và b-a=2. Với số nhỏ, duyệt thẳng giúp bám sát định nghĩa mà chưa cần tối ưu.

Độ phức tạp: O(tM) thời gian, O(1) bộ nhớ; M=max(b)≤100.

```
prime(x) = x>=2 và không có d trong [2,x-1] chia hết x
in Y khi b-a=2 và prime(a) và prime(b)
```

Subtask 2 - 30 điểm

Chỉ thử ước tới căn bậc hai của mỗi số. Nếu x là hợp số thì ít nhất một ước không vượt √x. Mỗi số chỉ cần thử tối đa 100 ứng viên; vẫn giữ phép kiểm tra hiệu bằng 2.

Độ phức tạp: O(t√M) thời gian, O(1) bộ nhớ; $M \le 10^4$.

```
thử d từ 2 khi d<=x/d; gặp x%d=0 thì kết luận hợp số
đồng thời kiểm tra b-a=2
```

Subtask 3 - 50 điểm

Dùng nguyên lời giải chia thử tới căn bậc hai ở trên. Khi $b \le 10^6$, mỗi số chỉ cần khoảng 1000 lần thử ước và $t \le 10$, nên không cần đổi sang sàng. Mã giả lời giải đầy đủ đã bao quát cả ba nhóm.

Độ phức tạp: O(t√M) thời gian, O(1) bộ nhớ; $M \le 10^6$.

KC1-B / KHÁM PHÁ LỜI GIẢI

B. Bảng đen

Tìm kiếm nhị phân | Toán học | Gom nhóm

ĐIỂM MẤU CHỐT

Đừng dựng dãy dài: tính tổng trọn từng nhóm, rồi chỉ đếm phần cần lấy trong nhóm cuối.

Từ quan sát đến thuật toán

Nhóm thứ k có k số, mỗi số bằng k. Nó đóng góp k^2 vào tổng và k vào số lượng phần tử. Sau k nhóm, tổng giá trị là $S(k) = k(k+1)(2k+1)/6$; tổng số phần tử là $k(k+1)/2$.

Tìm k nhỏ nhất sao cho $S(k) \geq x$. Hàm S tăng nghiêm ngặt, nên tìm được k bằng nhị phân. Khi đó phải lấy trọn k-1 nhóm đầu và lấy thêm một phần của nhóm k.

Đặt $r = x - S(k-1)$, luôn có $1 \leq r \leq k^2$. Mỗi phần tử tiếp theo đóng góp k, nên cần thêm ceil(r/k) phần tử. Đáp án là $k(k-1)/2 + \text{ceil}(r/k)$. Chia nguyên có thể viết $\text{ceil}(r/k) = (r+k-1)/k$.

Ví dụ để tự kiểm tra

Với x = 6: S(1)=1 < 6 và S(2)=5 < 6, còn S(3)=14. Lấy 3 phần tử của hai nhóm đầu: 1,2,2, tổng 5. Cần thêm ceil((6-5)/3)=1 phần tử là 3. Đáp án bằng 4. Với x=14, lấy đúng 6 phần tử, không được nhảy sang nhóm 4.

MỖI KHỐI k CÓ k PHẦN TỬ

1 | 2 2 | 3 3 3 | 4 4 4 4

Tổng sau các khối: 1, 5, 14, 30. Đừng nhầm tổng với số phần tử.

KC1-B / CHỨNG MINH VÀ CÀI ĐẶT

Bảng đen

Vì sao thuật toán đúng?

Tính tối tiểu của k cho S(k-1) < x ≤ S(k), nên mọi tiền tố đủ tổng đều đi vào nhóm k và không cần sang nhóm k+1. Trong nhóm k, j phần tử thêm cho tổng S(k-1)+jk. Điều kiện tổng đạt x tương đương j ≥ r/k; số nguyên nhỏ nhất thỏa là ceil(r/k). Công thức vì thế đếm đúng tiền tố ngắn nhất.

Mã giả

```
S(k) = k*(k+1)*(2*k+1)/6  // dùng số nguyên 128 bit

lo = 1; hi = 1
trong khi S(hi) < x: hi = 2*hi
trong khi lo < hi:
    mid = lo + (hi-lo)/2
    nếu S(mid) >= x: hi = mid
    ngược lại: lo = mid+1
k = lo
r = x-S(k-1)
đáp án = k*(k-1)/2 + (r+k-1)/k
```

Độ phức tạp

O(log x) thời gian cho mỗi truy vấn, O(1) bộ nhớ phụ.

Các lỗi dễ gặp

- Không dùng căn bậc ba bằng số thực để quyết định k; sai số tại S(k) rất dễ gây lệch 1.
- Ép kiểu __int128 trước phép nhân đầu tiên. Tích trung gian có thể vượt long long dù x chỉ tới 10^{18}.
- Phân biệt tổng giá trị S(k) với số phần tử k(k+1)/2.

KC1-B / LỜI GIẢI TỪNG SUBTASK

Bảng đen

Subtask	Điểm	Giới hạn bổ sung
1	20	Tổng x trong file ≤ 2×10^5
2	30	Mọi x ≤ 10^9
3	50	Không có giới hạn bổ sung

Subtask 1 - 20 điểm

Sinh lần lượt 1 số 1, 2 số 2, 3 số 3,... và cộng vào tổng cho đến khi đạt x. Mỗi phần tử ít nhất bằng 1, nên số phần tử đã sinh không vượt x. Tổng số bước qua toàn bộ truy vấn vì vậy không quá tổng x.

Độ phức tạp: O(Σx) thời gian, O(1) bộ nhớ.

```
sum=0; count=0
cho k=1,2,...: lặp tối đa k lần:
    sum+=k; count++; nếu sum>=x: trả count
```

Subtask 2 - 30 điểm

Duyệt theo khối thay vì theo từng phần tử. Khối k đóng góp k^2. Nếu tổng sau khối vẫn nhỏ hơn x, lấy trọn khối; khi khối đầu tiên đủ tổng xuất hiện, lấy thêm ceil((x-sum)/k) phần tử rồi dừng. Với x≤10^9, chỉ cần ít hơn 1500 khối cho mỗi truy vấn.

Độ phức tạp: O(tX^(1/3)) thời gian, O(1) bộ nhớ; X=max(x).

```
sum=0; count=0; k=1
trong khi sum+k*k<x: sum+=k*k; count+=k; k++
trả count + (x-sum+k-1)/k
```

Subtask 3 - 50 điểm

Khi x tới 10^{18}, không duyệt tất cả khối cho mỗi truy vấn. Nhị phân k nhỏ nhất có S(k)≥x rồi dùng công thức k(k-1)/2+ceil((x-S(k-1))/k). Tính tích trong S bằng số nguyên 128 bit trước khi chia cho 6.

Độ phức tạp: O(t log X) thời gian, O(1) bộ nhớ.

KC1-C / KHÁM PHÁ LỜI GIẢI

C. Mua kẹo

Hàng đợi | Mô phỏng | Phân tích khấu hao

ĐIỂM MẤU CHỐT

Lưu cả một lô kẹo bằng một cặp (số lượng, giá), không lưu từng viên.

Từ quan sát đến thuật toán

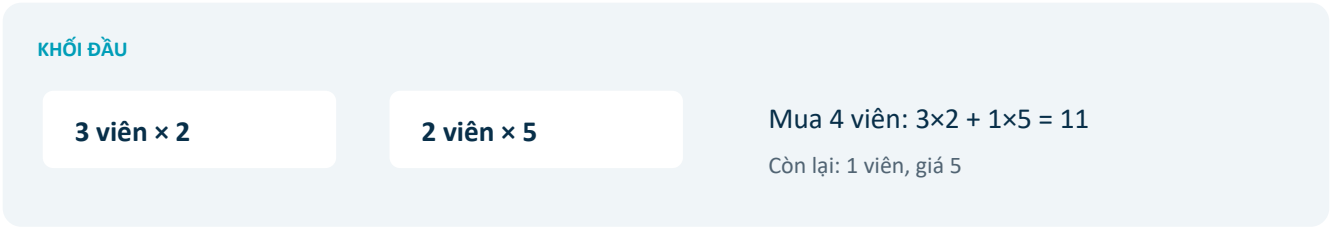
Kẹo nạp trước được bán trước: đây là thứ tự FIFO. Một lần nạp 10^9 viên cùng giá không cần 10^9 phần tử; chỉ cần một khối (số lượng, giá) trong hàng đợi.

Khi mua c viên, nhìn khối ở đầu. Lấy take = min(c, số lượng khối), cộng take × giá vào tiền, rồi trừ take khỏi cả c lẫn số lượng khối. Khối nào hết thì bỏ khỏi đầu. Tiếp tục đến khi c = 0.

Một lần mua có thể đi qua nhiều khối, nhưng mỗi khối chỉ được thêm một lần và bỏ một lần trong cả test. Những lần dừng giữa một khối được tính theo số truy vấn mua. Đây là lý do tổng thời gian là tuyến tính, không phải q^2.

Ví dụ để tự kiểm tra

Nạp (3 viên, giá 2), rồi (2 viên, giá 5). Mua 4 viên: lấy hết 3 viên đầu, trả 3×2; lấy thêm 1 viên của khối sau, trả 1×5. Tổng bằng 11. Hàng đợi còn (1 viên, giá 5); lần mua tiếp trả 5.



KC1-C / CHỨNG MINH VÀ CÀI ĐẶT

Mua kẹo

Vì sao thuật toán đúng?

Bất biến: hàng đợi mô tả chính xác các viên chưa bán, theo thứ tự chúng sẽ được lấy ra. Thêm một khối ở cuối bảo toàn thứ tự nạp. Khi mua, khối đầu chứa đúng các viên phải bán trước; lấy tối đa lượng còn thiếu rồi chuyển sang khối sau là bắt buộc theo FIFO. Mỗi lần cập nhật vì vậy bán đúng các viên và cộng đúng giá.

Mã giả

```
Q = hàng đợi rỗng
VỚI MỖI SỰ KIỆN:
    nếu là nạp (x,a):
        thêm (x,a) vào cuối Q
    nếu là mua c:
        tiền = 0
        trong khi c > 0:
            take = min(c, Q.đầu.số_lượng)
            tiền += take * Q.đầu.giá
            c -= take
            Q.đầu.số_lượng -= take
            nếu Q.đầu.số_lượng = 0: bỏ đầu Q
        in tiền
```

Độ phức tạp

O(q) thời gian khấu hao và O(q) bộ nhớ cho một test.

Các lỗi dễ gặp

- Dùng số nguyên 64 bit cho số tiền và ép kiểu trước take×giá. Tiền một lần mua có thể tới 10^{18}.
- Không bỏ một khối nếu chỉ bán một phần của nó.
- Xóa hàng đợi khi chuyển sang test mới. Dữ liệu bảo đảm đủ kẹo nên không cần tự đặt quy tắc khi thiếu hàng.

KC1-C / LỜI GIẢI TỪNG SUBTASK

Mua kẹo

Subtask	Điểm	Giới hạn bổ sung
1	40	Tổng số viên nạp trong file ≤ 2×10^5
2	60	Không có giới hạn bổ sung

Subtask 1 - 40 điểm

Lưu giá của từng viên vào một hàng đợi. Nạp x viên giá a thì đẩy a đúng x lần; mua c viên thì lấy c phần tử đầu và cộng giá. Viên nào nạp trước luôn được bán trước, đúng nguyên tắc FIFO.

Độ phức tạp: O(q+V) thời gian, O(V) bộ nhớ; V là tổng số viên nạp.

```
nạp (x,a): lặp x lần, đẩy a vào cuối Q
mua c: lặp c lần, cộng giá ở đầu Q rồi bỏ đầu
```

Subtask 2 - 60 điểm

Số lượng có thể tới 10^9 nên lưu theo khối (số lượng,giá). Khi mua, lấy tối đa từ khối đầu; chỉ bỏ khối khi đã hết. Mỗi khối được thêm và bỏ một lần, nên tổng công việc không phụ thuộc số viên.

Độ phức tạp: O(q) thời gian khấu hao, O(q) bộ nhớ. Tiền dùng 64 bit.

KC1-D / KHÁM PHÁ LỜI GIẢI

D. Vực Sâu

Quy hoạch động | Mô phỏng trạng thái

ĐIỂM MẤU CHỐT

Một hầm bắt buộc làm hai luồng hành trình đổi giếng cho nhau; phải cập nhật đồng thời.

Từ quan sát đến thuật toán

Xét các hầm từ trên xuống theo đúng thứ tự đầu vào. Gọi f[u] là giá trị lớn nhất của một hành trình đang ở giếng u ngay trước độ sâu đang xử lý. Khởi tạo f[u]=0 cho mọi giếng, vì có thể xuất phát ở bất kỳ miệng giếng nào.

Với hầm (u,v,w), mọi hành trình đang ở u bắt buộc đi sang v và nhận w; mọi hành trình đang ở v bắt buộc đi sang u và nhận w. Vì thế new_f[u]=old_f[v]+w và new_f[v]=old_f[u]+w. Các giếng khác không thay đổi.

Không lấy max với trạng thái cũ tại u hay v: đứng yên để bỏ qua hầm là hành động không được phép. Sau tất cả hầm, đáp án là max f[u]. Trọng số không âm cũng cho phép tiếp tục xuống mà không làm mất giá trị nếu đã có thể dừng trước đó.

Ví dụ để tự kiểm tra

Ban đầu f=(0,0,0). Qua hầm (1,2,5): f=(5,5,0). Qua (2,3,7): f=(5,7,12). Qua (1,3,4): f=(16,7,9). Đáp án là 16; xuất phát ở giếng 1, lần lượt đi 1 → 2 → 3 → 1 và nhận 5+7+4.

KC1-D / CHỨNG MINH VÀ CÀI ĐẶT

Vực Sâu

Vì sao thuật toán đúng?

Giả sử f đúng trước một hầm. Những hành trình đến u sau hầm chính xác là những hành trình ở v trước hầm, với cùng khoản thưởng w; lấy lớn nhất cho new_f[u]=old_f[v]+w. Lập luận đối xứng cho v; giếng khác không bị tác động. Quy nạp qua toàn bộ hầm cho f cuối cùng đúng, và lấy max trên mọi giếng xét đủ mọi điểm kết thúc.

Mã giả

```
f[1..n] = 0
VỚI CÁC HẦM (u,v,w) THEO THỨ TỰ ĐẦU VÀO:
    old_u = f[u]
    old_v = f[v]
    f[u] = old_v+w
    f[v] = old_u+w
đáp án = max(f[1..n])
```

Độ phức tạp

O(n+m) thời gian và O(n) bộ nhớ.

Các lỗi dễ gặp

- Lưu cả old_u và old_v trước khi gán. Gán f[u] rồi dùng f[u] để gán f[v] sẽ tính cùng hầm hai lần.
- Không xây đồ thị vô hướng rồi chạy đường đi dài nhất; thứ tự độ sâu và việc bắt buộc đi xuyên hầm là phần quyết định mô hình.
- m có thể bằng 0; khi đó đáp án bằng 0. Dùng long long cho tổng.

KC1-D / LỜI GIẢI TỪNG SUBTASK

Vực Sâu

Subtask	Điểm	Giới hạn bổ sung
1	40	n,m ≤ 2000
2	60	Không có giới hạn bổ sung

Subtask 1 - 40 điểm

Thử từng giếng làm điểm xuất phát. Với một điểm xuất phát, quét các hầm từ trên xuống: nếu đang ở một đầu hầm thì bắt buộc chuyển sang đầu kia và cộng w; nếu không thì bỏ qua. Lấy tổng lớn nhất của mọi điểm xuất phát.

Độ phức tạp: O(nm+n) thời gian, O(m) bộ nhớ để lưu hầm.

```
cho start=1..n: pos=start; value=0
    quét (u,v,w): nếu pos=u hoặc pos=v:
        pos=đầu còn lại; value+=w
lấy max của các value
```

Subtask 2 - 60 điểm

Mô phỏng đồng thời các hành trình bằng một trạng thái cho mỗi giếng. Hầm (u,v,w) biến hai giá trị thành old_f[v]+w và old_f[u]+w. Các giếng khác giữ nguyên. Phải lấy cả hai giá trị cũ trước khi cập nhật.

Độ phức tạp: O(n+m) thời gian, O(n) bộ nhớ; có thể đọc hầm và cập nhật ngay.