universal cup 3 stage 22 : Zhangzhou

Created: 5/3/2025 23:10
Updated: 5/3/2025 23:14
Exported: 5/4/2025 22:24

🔠 Bài A. A + B = C

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Cho ba số nguyên dương p_A, p_B, p_C. Bobo thách bạn tìm ba **xâu nhị phân vô hạn** A, B, C với **chu kỳ lần lượt là** p_A, p_B, p_C, sao cho:

$$A \oplus B = C$$

(Hoặc xác định rằng điều đó **không thể thực hiện được**.)

👉 Chú thích:

- Một xâu nhị phân vô hạn có chu kỳ p là một xâu lặp vô hạn với đoạn đầu dài p lặp đi lặp lại mãi mãi.
- Phép XOR giữa hai xâu nhị phân s và t được định nghĩa là:

$$(s \oplus t)_i = s_i \oplus t_i \quad (\text{XOR tại từng vị trí})$$

🔢 Input:

- Dòng đầu là số nguyên T — số lượng test ($1 \le T \le 10^4$)
- Với mỗi test, gồm 1 dòng chứa 3 số nguyên p_A, p_B, p_C ($1 \le p_A, p_B, p_C \le 10^6$)

🔒 **Ràng buộc đặc biệt:**
Tổng của tất cả các giá trị $\max(p_A, p_B, p_C)$ trong toàn bộ các test **không vượt quá 10^6**.

🧾 Output:

- Nếu **không tồn tại** ba xâu như yêu cầu, in `"NO"`
- Nếu **có thể**, in `"YES"` và sau đó in 3 dòng chứa:
  - Dòng 1: xâu nhị phân độ dài p_A
  - Dòng 2: xâu nhị phân độ dài p_B
  - Dòng 3: xâu nhị phân độ dài p_C

✅ In `"YES"` hoặc `"NO"` không phân biệt chữ hoa/thường (`yes`, `Yes`, `yEs` đều hợp lệ).

📌 Ví dụ:

Input:

```
2
2 3 6
2 3 5
```

Output:

```
YES
01
011
```

```
001110
NO
```

✍️ Tóm tắt đề bài:

- Cho ba số nguyên là độ dài chu kỳ của ba xâu nhị phân vô hạn.
- Kiểm tra xem có tồn tại ba xâu có các chu kỳ tương ứng sao cho $A \oplus B = C$.
- Nếu có: in ra `"YES"` và ba xâu con đầu tiên của mỗi chu kỳ.
- Nếu không: in `"NO"`.

Bạn có muốn mình bắt tay luôn vào tìm lời giải cho bài này không?

□ Bài B. Rolling Stones

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo đang chơi một trò chơi có tên **Rolling Stones** trên một **bàn tam giác đều** gồm n hàng (với $n \geq 2$). Bàn gồm tổng cộng n^2 ô, và mỗi ô được đánh số từ 1 đến 4.

Bobo có một **viên đá hình tứ diện đều (tetrahedron)**, mỗi mặt mang số từ 1 đến 4. Viên đá bắt đầu tại ô đầu tiên của hàng đầu tiên với **hướng ban đầu** như sau:

- Mặt số **1** hướng **trái**
- Mặt số **2** hướng **xuống hàng tiếp theo**

- Mặt số **3** hướng **phải**
- Mặt số **4** hướng **xuống dưới (mặt đáy)**

🎯 Mục tiêu:

Chuyển viên đá đến **ô đích** (được chỉ định), tuân theo các quy tắc sau:

1. **Số khớp:** Khi viên đá nằm trên một ô, thì **số ở đáy của viên đá phải khớp với số trên ô đó**
2. **Không đi lại:** Mỗi ô chỉ được đi qua **một lần duy nhất**, bao gồm cả ô bắt đầu và ô kết thúc
3. Viên đá **di chuyển bằng cách lăn theo cạnh**, đến một ô lân cận (hàng kế bên hoặc cạnh bên)

🔢 Input:

- Dòng đầu: số nguyên n ($2 \le n \le 100$)
- Tiếp theo là n dòng, dòng thứ i có $2i - 1$ số $a_{i,1}, a_{i,2}, ..., a_{i,2i-1}$ ($1 \le a_{i,j} \le 4$) là giá trị của các ô theo hàng.
    - Đảm bảo rằng $a_{1,1} = 4$ (ô đầu tiên phải khớp đáy ban đầu)
- Dòng cuối: hai số nguyên x, y — chỉ ra ô đích ở hàng thứ x, cột thứ y

---

🧾 Output:

- Nếu **không thể** đến ô đích theo quy tắc ⇒ in `-1`
- Nếu **có thể**, in ra **số bước lăn tối thiểu** để đến ô đích

---

🧪 Ví dụ:

Input 1:

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

Output 1:

```
6
```

Input 2:

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

Output 2:

```
-1
```

🧠 Tóm tắt đề bài:

- Cho một bàn tam giác đều với các ô có số 1–4.
- Viên đá tứ diện bắt đầu ở ô (1, 1), có hướng mặt được cho.
- Mỗi bước lăn sẽ thay đổi hướng các mặt viên đá.
- Chỉ được lăn đến ô có số trùng với mặt đáy hiện tại.
- Mỗi ô chỉ đi một lần.
- Yêu cầu: đến được ô đích chỉ định trong **ít bước nhất**, hoặc in `-1` nếu không thể.

Bạn có muốn mình triển khai luôn hướng giải cụ thể và code cho bài này không?

🧮 Bài C. Middle Point

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo đang khám phá một tập hợp các **điểm nguyên** (lattice points) trên mặt phẳng 2D.

Ban đầu, tập hợp S gồm 4 điểm tạo thành hình chữ nhật:

$$S = \{(0, 0), (A, 0), (0, B), (A, B)\}$$

🎯 Mục tiêu:

Bobo muốn **tạo thêm điểm** (X, Y) bằng cách thực hiện **các phép trung điểm** như sau:

- Mỗi bước, **chọn hai điểm** $P, Q \in S$ sao cho **trung điểm** $\frac{P+Q}{2}$ vẫn là điểm nguyên ⇒ thêm điểm đó vào S.

🧾 Input:

- Dòng 1: hai số nguyên A, B ($0 \le A, B \le 10^9$)
- Dòng 2: hai số nguyên X, Y ($0 \le X \le A$, $0 \le Y \le B$)

📤 Output:

- Nếu **không thể tạo ra** điểm (X, Y), in `-1`
- Nếu **có thể**, in ra:
  - Số bước k thực hiện
  - k dòng, mỗi dòng gồm 4 số nguyên U_i, V_i, S_i, T_i biểu thị hai điểm $P = (U_i, V_i)$, $Q = (S_i, T_i)$ được dùng để sinh ra điểm trung bình

⚠️ Nhiều đáp án đúng có thể tồn tại, chỉ cần in ra một đáp án hợp lệ.

🧪 Ví dụ:

Input:

```
2 2
1 1
```

Output:

```
1
0 0 2 2
```

Input:

```
2024 2024
2023 2023
```

Output:

```
-1
```

✍️ Tóm tắt đề bài:

- Tập ban đầu gồm 4 đỉnh của hình chữ nhật: (0, 0), (A, 0), (0, B), (A, B)
- Mỗi bước bạn được quyền lấy trung điểm của 2 điểm trong tập **nếu trung điểm là số nguyên**
- Mục tiêu: tạo được điểm (X, Y) từ các phép trung điểm trên
- Xuất ra số bước nhỏ nhất và chi tiết từng bước (hoặc -1 nếu không thể)

Bạn có muốn mình bắt đầu trình bày cách giải hoặc code mẫu cho bài này không?

🎮 Bài D. Guessing Game

⏱️ **Giới hạn thời gian:** 2 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Alice và Bobo chơi một trò chơi đoán số. Ở vòng thứ k, bạn sẽ viết một **cặp số nguyên** (a_k, b_k) lên bảng.

Sau đó, **bạn sẽ bí mật chọn một chỉ số** i với $1 \leq i \leq k$, rồi:

- Gửi giá trị a_i cho **Alice**
- Gửi giá trị b_i cho **Bobo**

🧠 Luật chơi:

- **Alice bắt đầu trước.** Mỗi lượt, họ có thể:
  - **Đoán được số của người kia** và thắng
  - Hoặc **nói "chưa biết"** để nhường lượt
- Cả hai đều rất thông minh và trung thực:
  - Nếu biết chắc chắn thì sẽ nói ra ngay
  - Nếu chưa chắc chắn thì sẽ không đoán

❓ Yêu cầu:

Với mỗi vòng thứ k, bạn cần in ra:

- A_k: Số lượng giá trị $i \in [1, k]$ mà **Alice thắng**
- B_k: Số lượng giá trị $i \in [1, k]$ mà **Bobo thắng**

🔢 Input:

- Dòng đầu tiên: số nguyên q ($1 \le q \le 10^6$) — số cặp truy vấn
- q dòng tiếp theo: mỗi dòng chứa một cặp số nguyên (a_k, b_k), với $1 \le a_k, b_k \le 10^5$

Đảm bảo rằng các cặp (a_k, b_k) đều **khác nhau**.

📤 Output:

- In ra q dòng, dòng thứ k chứa hai số nguyên A_k, B_k

🧪 Ví dụ:

Input:

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

Output:

```
1 0
0 2
1 2
0 0
```

✍️ Tóm tắt đề bài:

- Bạn có một chuỗi các cặp (a_k, b_k)
- Với mỗi k, xét tất cả chỉ số $i \in [1, k]$
  - Nếu Alice biết chắc chắn b_i dựa trên a_i, và nói ra được trước ⇒ cô ấy thắng
  - Nếu không, nhưng Bobo biết chắc chắn a_i dựa trên b_i ⇒ Bobo thắng

---

💡 Gợi ý hướng giải:

- Với mỗi vòng k, lưu lại các cặp đã thấy từ 1 đến k
- Dùng hai **hashmap**:
  - `` `mpA[a]` `` = tập các b từng đi kèm với a
  - `` `mpB[b]` `` = tập các a từng đi kèm với b
- Khi số lượng giá trị ứng với a chỉ có đúng một b ⇒ Alice có thể đoán được b nếu nhận được a
- Tương tự cho Bobo

Bạn muốn mình triển khai luôn lời giải chi tiết hoặc code Python/C++ cho bài này chứ?

🔁 Bài E. Permutation Routing

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo được cho một cây $T = (V, E)$ gồm n đỉnh. Mỗi đỉnh i ban đầu chứa một số p_i, và p là một hoán vị của các số từ 1 đến n, nghĩa là:

$$\{p_1, p_2, ..., p_n\} = \{1, 2, ..., n\}$$

🔁 Phép biến đổi:

Mỗi phép biến đổi, Bobo:

- Chọn một **matching** $M \subseteq E$ (tập các cạnh không có đỉnh chung)
- Với mỗi cạnh $(u, v) \in M$, thực hiện hoán đổi $p_u \leftrightarrow p_v$

🎯 Mục tiêu:

Thực hiện nhiều nhất **3n phép biến đổi**, sao cho cuối cùng:

$$p_i = i \quad \text{với mọi } 1 \leq i \leq n$$

🔢 Input:

- Dòng đầu là số lượng test case T
- Với mỗi test:
  - Dòng 1: số đỉnh n ($1 \le n \le 1000$)
  - Dòng 2: hoán vị ban đầu $p_1, ..., p_n$
  - $n - 1$ dòng tiếp theo: các cạnh của cây

✅ Đảm bảo tổng của n^2 trên tất cả test ≤ 10^6

📩 Output:

Với mỗi test case:

- Dòng đầu tiên: số thao tác m ($0 \le m \le 3n$)
- Sau đó m dòng:
  - Mỗi dòng bắt đầu bằng số k_i: số cạnh trong matching
  - Tiếp theo là k_i số: chỉ số của các cạnh trong matching đó (đánh số từ 1)

🧪 Ví dụ:

Input:

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

```
2 4
1 5
```

Output:

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

✍️ Tóm tắt đề bài:

- Cho một cây, mỗi đỉnh có một số hoán vị ban đầu
- Mỗi lần chọn các cạnh không chung đỉnh (matching) và swap giá trị ở hai đầu mỗi cạnh
- Mục tiêu: đưa mỗi giá trị về đúng vị trí của nó (tức là $p_i = i$)
- In ra tối đa 3n phép swap để đạt được điều này

Bạn có muốn mình trình bày luôn **thuật toán** và **code mẫu** để giải bài này không?

🔁 Bài F. Infinite Loop

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo bị mắc kẹt trong một **vòng lặp thời gian vô hạn**!

Mỗi ngày có đúng k giờ. Mỗi ngày, có đúng n công việc xuất hiện:

- Công việc thứ i trong ngày đến vào **đầu giờ thứ** a_i và mất b_i giờ liên tục để hoàn thành.
- Các a_i **tăng dần** (đảm bảo theo thứ tự tăng).

🧠 Quy tắc làm việc của Bobo:

- Bắt đầu từ ngày đầu tiên, Bobo **không có công việc nào**.
- **Bất kỳ lúc nào có công việc chưa hoàn thành**, Bobo sẽ làm việc theo quy tắc:
  - **Chọn công việc đến sớm nhất chưa xong**, làm đến khi xong.

❓ Nhiệm vụ của bạn:

Cho q truy vấn. Truy vấn thứ i cho biết:

- x_i: ngày mà công việc được tạo ra
- y_i: chỉ số công việc trong ngày đó

Yêu cầu: xác định chính xác **ngày** và **giờ** mà công việc đó hoàn thành.

🔢 Input:

- Dòng đầu: 3 số nguyên n, k, q
  - n: số công việc mỗi ngày ($1 \le n \le 10^5$)
  - k: số giờ mỗi ngày ($1 \le k \le 10^8$)
  - q: số truy vấn ($1 \le q \le 10^5$)
- n dòng tiếp theo: mỗi dòng a_i, b_i: giờ bắt đầu và thời gian làm của công việc thứ i trong ngày
  ($1 \le a_i, b_i \le k$, $a_1 < a_2 < \cdots < a_n$)
- q dòng truy vấn: mỗi dòng chứa x_i, y_i
  - ($1 \le x_i \le 5 \times 10^5$, $1 \le y_i \le n$)

📤 Output:

- Với mỗi truy vấn, in ra 2 số:
  - d_i: ngày kết thúc công việc
  - h_i: giờ trong ngày đó (bắt đầu từ 1)

🧪 Ví dụ:

Input:

```
2 5 6
1 1
4 3
1 1
1 2
2 1
2 2
```

```
3 1
3 2
```

Output:

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

✍️ Tóm tắt đề bài:

- Có một **chuỗi vô hạn các ngày**, mỗi ngày lặp lại n công việc giống nhau theo cùng lịch (a_i, b_i).
- Bobo xử lý tuần tự theo thứ tự công việc đến (trong tổng timeline).
- Với mỗi truy vấn (ngày, công việc thứ y), tìm thời điểm công việc đó **hoàn thành** (ngày, giờ).

Bạn muốn mình bắt đầu trình bày hướng giải hoặc code chi tiết cho bài này chứ?

➗ Bài G. Same Sum

⏱️ **Giới hạn thời gian:** 2 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Cho dãy số nguyên $a_1, a_2, \ldots, a_n$ gồm n phần tử.

Bạn cần xử lý q truy vấn. Có 2 loại truy vấn:

1. **Cộng đoạn (loại 1):**

   ```
   1 L R v
   ```

   Với mọi $i \in [L, R]$, cập nhật:

   $$a_i \leftarrow a_i + v$$

2. **Truy vấn kiểm tra (loại 2):**

   ```
   2 L R
   ```

   Với $R - L + 1$ **chẵn**, kiểm tra xem đoạn $a_L, a_{L+1}, ..., a_R$ có thể chia thành các **cặp không trùng nhau** mà **tổng của mỗi cặp đều bằng nhau** hay không?

🔢 Input:

- Dòng đầu tiên: hai số nguyên n, q ($1 \le n, q \le 2 \cdot 10^5$)
- Dòng tiếp: n số nguyên $a_1, ..., a_n$ ($0 \le a_i \le 2 \cdot 10^5$)
- q dòng tiếp theo: mỗi dòng là một truy vấn như trên

📤 Output:

- Với mỗi truy vấn loại 2, in `YES` nếu có thể chia được thành các cặp có tổng bằng nhau, ngược lại in `NO`.

✅ Có thể in in hoa/thường bất kỳ (`YES`, `Yes`, `yes` đều hợp lệ)

🧪 Ví dụ:

Input:

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

Output:

```
YES
NO
YES
```

✍️ Tóm tắt đề bài:

- Dãy số nguyên kích thước lớn, có cập nhật đoạn và kiểm tra đoạn.
- Với truy vấn loại 2:
  - Kiểm tra xem đoạn có thể chia thành các **cặp** (x_i, y_i) mà $x_i + y_i = S$ như nhau không?

Bạn có muốn mình bắt tay vào **thiết kế thuật toán tối ưu** và **triển khai code mẫu** cho bài này không?

🧩 Bài H. The Witness

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo đang chơi trò chơi **The Witness**, một trò giải đố dựa trên việc nối các điểm để tách các ô màu đen và trắng.

Bạn được cung cấp một **lưới chữ nhật** $n \times m$, trong đó mỗi ô có màu **đen (B)** hoặc **trắng (W)**.

🧠 Yêu cầu:

Tìm một **đường đi đơn** từ **điểm bắt đầu** đến **điểm kết thúc** trên **lưới các đỉnh** (không phải các ô), sao cho:

1. Mỗi bước di chuyển là qua cạnh của lưới (lên, xuống, trái, phải giữa các đỉnh kề nhau).
2. Đường đi là **đơn** (không đi qua lại một đỉnh).
3. Đường đi chia lưới thành **nhiều miền kín** sao cho mỗi miền chỉ chứa **một màu duy nhất**.

📥 Input:

- Dòng 1: hai số nguyên n, m ($1 \le n, m \le 40$) — số hàng và cột
- n dòng tiếp theo: mỗi dòng là một chuỗi độ dài m gồm các ký tự `'B'` hoặc `'W'` — màu của ô
- Dòng tiếp theo: bốn số nguyên sx, sy, ex, ey — tọa độ hai đỉnh bắt đầu và kết thúc
  - Các đỉnh nằm **trên viền ngoài** của lưới
  - Đảm bảo điểm bắt đầu và kết thúc khác nhau

📤 Output:

- Nếu **không tồn tại** đường đi thỏa mãn ⇒ in `NO`
- Nếu **có**, in:
  - `YES`
  - Một số nguyên $\ell \ge 2$ — độ dài đường đi
  - ℓ dòng tiếp theo: tọa độ các đỉnh trong đường đi

✅ Có thể in hoa/thường tùy ý: `yes`, `Yes`, `YES` đều hợp lệ.

📐 Mô tả lưới và đỉnh:

- Lưới có $n \times m$ **ô vuông** → chia lưới thành $(n+1) \times (m+1)$ **đỉnh**
- Mỗi đỉnh có thể đi đến các đỉnh kề bằng cách di chuyển qua cạnh của lưới

---

🧪 Ví dụ:

Input:

```
3 3
BBB
BWB
WWW
3 0 0 3
```

Output:

```
YES
9
3 0
2 0
2 1
1 1
1 2
2 2
2 3
1 3
0 3
```

Input:

```
2 2
WB
BW
0 0 2 2
```

Output:

```
NO
```

✍️ Tóm tắt đề bài:

- Bạn cần tìm một đường đi đơn từ điểm A đến B, đi qua các đỉnh của lưới.
- Đường đi này sẽ tạo ra các vùng (region) khép kín — và yêu cầu là **mỗi vùng chỉ chứa một màu duy nhất**.
- Tức là đường đi phải **phân tách đen và trắng** hoàn toàn.

Bạn có muốn mình giải thích hướng tiếp cận (ví dụ: chuyển grid thành đồ thị đỉnh-cạnh, BFS/DFS, xử lý kiểm tra tách vùng màu) hoặc viết code luôn không?

🤝 Bài I. Best Friend, Worst Enemy

⏱️ **Thời gian:** 1 giây
💾 **Bộ nhớ: 32MB**

📄 Đề bài (dịch tiếng Việt):

Bobo đang phân tích một nhóm gồm n người. Mỗi người i có hai thuộc tính nguyên (x_i, y_i), **đảm bảo tất cả người đều có thuộc tính khác nhau**.

Với mỗi cặp người $(i, j), i \neq j$, Bobo định nghĩa:

- **Friend index**:

$$\text{Friend}(i,j) = \max(|x_i - x_j|,\ |y_i - y_j|)$$

- **Enemy index**:

$$\text{Enemy}(i,j) = |x_i - x_j| + |y_i - y_j|$$

🎯 Mục tiêu:

Gọi người j là:

- **Best Friend của i** nếu: không có ai khác $k \neq i$ sao cho Friend(i, k) < Friend(i, j)
- **Worst Enemy của i** nếu: không có ai khác $k \neq i$ sao cho Enemy(i, k) > Enemy(i, j)

💡 Tức là j là người **gần nhất theo max-distance** và **xa nhất theo manhattan-distance** trong số các người khác (theo nghĩa đảo chiều).

❓ Nhiệm vụ:

Với mỗi $t \in [1, n]$, in ra số lượng **cặp có thứ tự** (i, j) thoả mãn:

- $1 \leq i, j \leq t$
- $i \neq j$
- j là **cả best friend và worst enemy của i**, **chỉ xét trong nhóm người từ 1 đến t**

🔢 Input:

- Dòng 1: số nguyên n ($2 \le n \le 4 \times 10^5$)
- n dòng tiếp theo: mỗi dòng gồm 2 số nguyên $x_i,\ y_i$ ($1 \le x_i, y_i \le 10^7$)

📤 Output:

- n dòng, dòng thứ t chứa số lượng cặp (i, j) thỏa mãn yêu cầu, khi xét **chỉ** người từ 1 đến t

---

🧪 Ví dụ:

Input:

```
2
1 5
1 10
```

Output:

```
0
2
```

↪ Có 2 cặp thoả mãn: (1,2) và (2,1)

✍️ Tóm tắt đề bài:

- Mỗi người có tọa độ (x, y), tính khoảng cách tới người khác theo 2 kiểu:

    - **Max distance** (chebyshev distance)
    - **Manhattan distance**
- Tìm người **gần nhất theo max** và **xa nhất theo manhattan**
- Đếm các cặp (i, j) sao cho j đồng thời là best friend và worst enemy của i (tức là người đạt min của max-distance và max của manhattan-distance)

🧠 Gợi ý hướng giải:

- Với mỗi i, để xác định:
    - **Best friend** = người có **min** giá trị `` `max(|xi - xj|, |yi - yj|)` ``
    - **Worst enemy** = người có **max** giá trị `` `|xi - xj| + |yi - yj|` ``

Vì giới hạn là 400.000, ta không thể so sánh mọi cặp $O(n^2)$ ⇒ cần dùng **cấu trúc dữ liệu tối ưu hóa**.

Bạn muốn mình bắt tay vào trình bày **thuật toán tối ưu chi tiết** và **cài code mẫu** cho bài này không?

🏆 Bài J. Balance in All Things

⏱️ **Thời gian:** 6 giây
💾 **Bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo tham gia một giải đấu kỳ lạ gồm 2^n người chơi (đánh số từ 1 đến 2^n).
Ban đầu, **tất cả người chơi đều có điểm là 0**.

Giải đấu gồm k vòng đấu. Trong mỗi vòng:

- Các người chơi được **ghép đôi**, đấu 1-1.

🧠 Quy tắc tính điểm sau mỗi trận:

Giả sử hai người chơi có điểm là a và b:

- Nếu $a > b$: người có **điểm cao hơn bị trừ 1 điểm**, người có điểm thấp hơn **cộng 1 điểm**
- Nếu $a = b$: người có **số thứ tự nhỏ hơn thắng**, **+1 điểm**, người kia **−1 điểm**

🔒 **Giới hạn đặc biệt:**
Tại **mọi thời điểm**, điểm của mỗi người chơi phải nằm trong đoạn **[−3, 3]**

❓ Yêu cầu:

Tính số **cách hợp lệ để sắp xếp các trận đấu** trong k vòng, sao cho sau mỗi trận:

- Điểm người chơi được cập nhật như trên
- Không ai vượt quá giới hạn điểm

🔢 Input:

- 1 dòng gồm 3 số nguyên:

- n: số mũ ⇒ có 2^n người chơi ($1 \le n \le 400$)
- k: số vòng ($1 \le k \le 20$)
- P: số nguyên tố để lấy mod ($10^8 \le P \le 10^9 + 9$)

📤 Output:

- In ra số cách hợp lệ modulo P

🧪 Ví dụ:

Input:

```
3 1 1000000007
```

Output:

```
15
```

🧠 Ý tưởng thuật toán (tổng quan):

- Đây là một bài **đếm tổ hợp với ràng buộc trạng thái**
- Có thể quy về **bài toán đếm trạng thái** với:
  - Dãy điểm người chơi (trạng thái)
  - Tại mỗi vòng: chọn cách ghép các người chơi → tạo trạng thái mới hợp lệ

- Dùng **quy hoạch động (DP)** theo:
  - Số vòng hiện tại
  - Cấu hình điểm của các người chơi (rút gọn dưới dạng đếm số lượng người có điểm là -3, -2, ..., $+3$)

Bạn có muốn mình viết chi tiết phần **thuật toán**, cách **mã hóa trạng thái**, và sau đó **triển khai code tối ưu** cho bài này không?

🥔 Bài K. Brotato

⏱️ **Thời gian:** 1.5 giây
💾 **Bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo đang chơi game **Brotato**, có tất cả **n màn chơi**. Mỗi màn:

- Có xác suất **thất bại** là p
- Có xác suất **thành công** là $1 - p$

Luật chơi bình thường:

- Nếu Bobo **thất bại**, anh ta phải **chơi lại từ level 1**

Tuy nhiên, Bobo có **k vật phẩm đặc biệt** cho phép:

- Nếu thua, **dùng 1 vật phẩm để chơi lại ngay tại màn đó** (thay vì từ đầu)

❓ Nhiệm vụ:

Tính **số lần chơi kỳ vọng tối thiểu** để Bobo **hoàn thành cả n màn** nếu chơi tối ưu với k vật phẩm.

🔢 Input:

- Dòng 1: hai số nguyên n, k
  ($1 \le n \le 10^5$, $0 \le k \le 10^9$)
- Dòng 2: một số thực p ($0 < p \le 0.5$) với **tối đa 4 chữ số thập phân**

✅ Đảm bảo: $n \cdot p \le 20$ — giúp tránh tràn số

📤 Output:

- In ra **kỳ vọng số lần chơi** (expected number of level attempts), với **độ chính xác đến 1e-9**

🧪 Ví dụ:

Input:

```
5 0
0.5
```

Output:

```
62.0000000000
```

Input:

```
5 1
0.5
```

Output:

```
47.0000000000
```

📐 Tóm tắt ý tưởng:

- Khi không có vật phẩm ⇒ nếu thất bại ở màn i, phải chơi lại từ đầu
- Khi có vật phẩm ⇒ ưu tiên dùng vật phẩm ở những level "rủi ro cao nhất"

💡 Hướng giải (tối ưu hóa bằng DP):

Gọi:

- `E[i][j]`: kỳ vọng số lượt chơi để hoàn thành từ level i đến n, nếu còn j vật phẩm

Ta có 2 lựa chọn:

1. **Dùng vật phẩm ở level** i (nếu còn vật phẩm):
    - Thành công: đi tiếp, xác suất $1 - p$
    - Thất bại: **vẫn ở level** i, dùng 1 vật phẩm ⇒ ở lại, mất thêm lượt

$$E[i][j] = 1 + (1-p) \cdot E[i+1][j-1] + p \cdot E[i][j-1]$$

2. **Không dùng vật phẩm**:
   - Thất bại ⇒ trở lại từ đầu
   - Thành công ⇒ tiếp tục

$$E[i][j] = 1 + (1-p) \cdot E[i+1][j] + p \cdot E[1][j]$$

✅ Tối ưu hóa thực tế:

- Không thể lưu bảng DP 2 chiều vì giới hạn bộ nhớ
- Thay vào đó: **chọn k level để dùng vật phẩm**, còn lại thì không
- Ta **chọn k level cuối cùng** để dùng vật phẩm (rủi ro lớn nhất khi quay lại từ gần cuối)

Bạn có muốn mình triển khai luôn code Python dùng chiến lược tối ưu hóa greedy + mô phỏng ngược để tính giá trị kỳ vọng không?

🧭 Bài L. Z-order Curve

⏱️ **Thời gian:** 1 giây
💾 **Bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Z-order curve là một dạng duyệt không gian hai chiều sử dụng **chuỗi Moser–de Bruijn** $(B_t)_{t\geq 0}$:

- Mỗi số trong chuỗi có dạng nhị phân với các bit **chỉ nằm ở vị trí chẵn** (0-indexed).
- Ví dụ: 0, 1, 4, 5, 16, 17, 20, 21...

👉 Mỗi số tự nhiên $z \geq 0$ có thể **duy nhất** viết được dưới dạng:

$$z = B_x + 2 \cdot B_y$$

Như vậy, có thể hình dung các số tự nhiên được **xếp trong một bảng vô hạn** theo thứ tự đường Z (Z-order curve).

🎯 Nhiệm vụ:

Cho một đoạn từ L đến R trên đường cong Z (theo thứ tự),
Hãy tìm giá trị **nhỏ nhất** l sao cho đoạn từ l đến $l + (R - L)$ có **dạng hình học** giống hệt đoạn $L \to R$.

- Lưu ý: hình dạng là **có hướng**, tức $1 \to 2$ không giống $3 \to 4$.

🔢 Input:

- Dòng đầu: số nguyên T — số lượng test (1 ≤ T ≤ 100)
- T dòng tiếp theo: mỗi dòng gồm hai số nguyên L, R (0 ≤ L < R ≤ 10^{18})

📤 Output:

- Với mỗi test, in ra số nguyên nhỏ nhất l

🧠 Hiểu kỹ đề:

- Chuỗi Z-order sắp xếp theo các số dạng:

$$z = B_x + 2 \cdot B_y$$

- **B_x** là số có các bit ở vị trí chẵn (bit 0, 2, 4,...)
- **B_y** là số có các bit ở vị trí chẵn, nhưng ta nhân thêm 2 ⇒ đẩy bit sang lẻ (bit 1, 3, 5,...)

⟹ **Z-order** giống như ánh xạ:

- bit chẵn lấy từ x
- bit lẻ lấy từ y
  ⇒ Gộp lại thành số z

=> Đây là phép **interleave bit**:

```
interleave(x, y) = z
```

Ví dụ:

```
x = 10 (binary: 1010)
y = 5  (binary: 0101)
⟹ z = interleave(x, y) = 110110 (binary) = 54
```

💡 Ý tưởng lời giải:

Để so sánh đoạn từ L đến R, ta:

- **Tách mỗi số z** thành cặp (x, y)
- Với mỗi truy vấn:
  - Tạo danh sách các cặp (x, y) từ L đến R
  - Dịch đoạn đó đi khắp nơi, tìm giá trị l sao cho đoạn tương ứng có cùng vector bước `(dx, dy)`
  - So khớp `(x_i+1 - x_i, y_i+1 - y_i)` giữa các đoạn

✅ Vì $R - L \leq 64$ (vì p ≤ 0.5 và np ≤ 20), ta có thể duyệt **toàn bộ đoạn dài** một cách hiệu quả

Bạn muốn mình bắt tay vào **viết code Python hoàn chỉnh** cho bài này không?

🎲 Bài M. Rejection Sampling

⏱️ **Thời gian:** 1 giây
💾 **Bộ nhớ:** 1024 MB

📄 Đề bài (dịch tiếng Việt):

Bobo muốn tạo ra một tập con ngẫu nhiên $T \subset \{1, 2, ..., n\}$ gồm đúng k phần tử bằng thuật toán **rejection sampling** như sau:

1. Ban đầu $T \leftarrow \emptyset$
2. Với mỗi $i = 1 \rightarrow n$, thêm i vào T với xác suất p_i
3. Nếu $|T| = k$ thì giữ kết quả; nếu không thì lặp lại từ đầu

🔢 Yêu cầu:

Bạn được cho mảng trọng số $a_1, ..., a_n$. Bạn phải tìm xác suất $p_1, ..., p_n$ sao cho:

- Tổng $\sum p_i = k$
- Với mọi tập con S gồm k phần tử, xác suất thuật toán trả ra S là **tỉ lệ thuận** với $\prod_{i \in S} a_i$

✔️ Đảm bảo tồn tại duy nhất bộ xác suất $p_1, ..., p_n$

📥 Input:

- Dòng 1: n, k (2 ≤ n ≤ 10^5; 1 ≤ k ≤ n−1)
- Dòng 2: $a_1, ..., a_n$ (1 ≤ a_i ≤ 10^9)

📤 Output:

- n dòng, dòng thứ i là xác suất p_i
- Đáp án đúng nếu **sai số tuyệt đối của từng p_i ≤ 10^{-6}**

🧪 Ví dụ:

Input:

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

Output (gợi ý):

```
0.310035697652
0.473324044845
0.574114878920
0.642525378583
```

💡 Ý tưởng lời giải:

Mục tiêu: chọn các xác suất p_i sao cho:

- $\sum p_i = k$
- Xác suất chọn đúng tập S có độ dài k là **proportional** với $\prod_{i \in S} a_i$

🧠 **Công thức chính xác** được xây dựng như sau:

- Gọi $x_i = \frac{p_i}{1-p_i} \Rightarrow p_i = \frac{x_i}{1+x_i}$
- Bài toán trở thành tìm $x_1, ..., x_n$ sao cho:
  - $\sum_{|S|=k} \prod_{i \in S} x_i \propto \sum_{|S|=k} \prod_{i \in S} a_i$
- Điều đó dẫn đến việc tìm $x_i \propto a_i$, sau đó điều chỉnh để $\sum p_i = k$

✅ Phương pháp cụ thể:

1. Tìm $\lambda > 0$ sao cho:

$$p_i = \frac{\lambda a_i}{1 + \lambda a_i}$$

thỏa mãn:

$$\sum_{i=1}^{n} p_i = k$$

2. Đây là **phương trình đơn điệu**, có thể giải bằng **tìm kiếm nhị phân trên** λ

✅ Code Python (giải chính xác đến 1e-9):

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

def compute_sum(lambda_):
    return sum((lambda_ * ai) / (1 + lambda_ * ai) for ai in a)

lo, hi = 1e-15, 1e15
for _ in range(100):  # đủ cho độ chính xác 1e-9
    mid = (lo + hi) / 2
    if compute_sum(mid) < k:
        lo = mid
    else:
        hi = mid

lambda_ = (lo + hi) / 2
for ai in a:
    pi = (lambda_ * ai) / (1 + lambda_ * ai)
    print(pi)
```

Bạn có muốn mình chuyển đoạn code này sang **C++** hoặc giải thích chi tiết cách chứng minh công thức không?