universal cup 3 stage 19 : Shenyang

Created: 5/14/2025 20:58
Updated: 5/14/2025 21:13
Exported: 5/14/2025 21:15

🇻🇳 Bài A. An Toàn Là Trên Hết (Safety First)

⏱️ **Giới hạn thời gian:** 3 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📜 Đề bài
Một cái thang gồm **n đoạn**, mỗi đoạn có độ dài là một số nguyên dương và các đoạn này **không tăng dần** từ trái qua phải:

$$d_1 \geq d_2 \geq \cdots \geq d_n$$

Cậu bé Little Q muốn dựng chiếc thang sao cho **chiều cao của thang đúng bằng** m, trong đó **chiều cao** được định nghĩa là **hiệu độ cao giữa điểm cao nhất và thấp nhất của chiếc thang** sau khi được dựng thẳng đứng.

⚠️ Tuy nhiên, vì lý do an toàn, chiếc thang phải **ổn định**, tức là phải thỏa mãn:

- Đầu dưới của đoạn **đầu tiên** phải chạm đất;
- Với mỗi $i = 1$ đến $n - 1$, thì đầu trên của đoạn thứ i phải được **ghép nối vào đầu trên hoặc đầu dưới** của đoạn thứ $i + 1$.

🎯 **Yêu cầu**: Với mỗi bộ giá trị n, m, hãy tính **số cách khác nhau** để dựng thang sao cho:

- Có đúng n đoạn;
- Chiều cao là đúng bằng m;
- Các đoạn ghép nối **ổn định và đúng quy tắc như trên**;

- Hai chiếc thang được coi là khác nhau nếu:
  - Có **bất kỳ đoạn nào có độ dài khác nhau**, hoặc
  - Có **cách ghép nối khác nhau** giữa hai đoạn liên tiếp.

Vì số lượng cách dựng có thể rất lớn, hãy in ra kết quả **modulo** 998244353.

🧾 Input

- Dòng đầu: Số lượng test T ($1 \le T \le 10^5$)
- Mỗi test gồm 1 dòng với hai số nguyên n, m ($1 \le n, m \le 2000$)

📤 Output

- Với mỗi test, in ra một dòng là **số cách khác nhau** để dựng thang thỏa mãn yêu cầu, **modulo** 998244353.

📌 Ví dụ

Input

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

Output

```
1
4
10
```

📚 Giải thích

- **Test 1:** Chỉ có duy nhất 1 đoạn dài 3 và nó ghép vào đất, thỏa yêu cầu.
- **Test 2:** Có 4 cách khác nhau để ghép 2 đoạn sao cho chiều cao bằng 3.
- **Test 3:** Có 10 cách khác nhau để ghép 3 đoạn sao cho tổng chiều cao bằng 3.

🧠 Tóm tắt ý tưởng bài toán

- Tìm **số cách chia m thành tổng các đoạn** không tăng dần, với đúng **n đoạn**.
- Mỗi đoạn có thể ghép **vào đầu trên hoặc đầu dưới** của đoạn sau.
- Do đó, bài toán là một bài toán **đếm tổ hợp có ràng buộc thứ tự và cấu trúc cây ghép**.
- Cần sử dụng **lập trình động (DP)** hoặc **quy hoạch tổ hợp** với tiền xử lý để trả lời nhanh cho nhiều test.

🇻🇳 Bài B. Bảng Pha Màu Ma Thuật (Magical Palette)

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

🎨 Đề bài

Thỏ trắng có một **bảng màu ma thuật** dạng **lưới** gồm n hàng và m cột. Trước khi bắt đầu pha màu, thỏ trắng sẽ:

- Nhỏ một loại sắc tố vào bên **trái mỗi hàng**, gọi là $a_1, a_2, \ldots, a_n$
- Nhỏ một loại sắc tố vào bên **trên mỗi cột**, gọi là $b_1, b_2, \ldots, b_m$

Có tổng cộng $n \times m$ loại sắc tố khác nhau, được đánh số từ 0 đến $nm - 1$.

Mỗi ô tại hàng i, cột j sẽ có màu được tính bởi công thức:

$$c_{i,j} = (a_i \cdot b_j) \bmod (n \cdot m)$$

🐰 Thỏ trắng **mong muốn tất cả các ô trong bảng** có màu **khác nhau**. Nhiệm vụ của bạn là kiểm tra điều này **có khả thi không**, và nếu có, hãy đưa ra một cách chọn a và b phù hợp.

🧾 Input

- Dòng đầu chứa số nguyên T — số test ($1 \le T \le 10^4$)
- Với mỗi test:
  - Một dòng gồm hai số nguyên n và m ($1 \le n, m \le 10^6$, và $n \cdot m \le 10^6$)

📌 Tổng số ô của tất cả test cộng lại không vượt quá 10^6

📤 Output

- Với mỗi test:
  - Nếu **không tồn tại cách** chọn a, b phù hợp → in `No`

- Nếu **tồn tại cách**, in 3 dòng:
  1. `Yes`
  2. Dãy $a_1, a_2, \ldots, a_n$ ($0 \le a_i < n \cdot m$)
  3. Dãy $b_1, b_2, \ldots, b_m$ ($0 \le b_j < n \cdot m$)

📌 Ví dụ

Input

```
2
2 3
2 2
```

Output

```
Yes
1 2
1 3 5
No
```

🔍 Giải thích test 1:

- Ma trận kết quả là:

$$\begin{bmatrix} 1\cdot 1 & 1\cdot 3 & 1\cdot 5 \\ 2\cdot 1 & 2\cdot 3 & 2\cdot 5 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 6 & 10 \end{bmatrix} \bmod 6 = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 0 & 4 \end{bmatrix}$$

→ Tất cả 6 giá trị đều **khác nhau** ⇒ hợp lệ.

🧠 Tóm tắt ý tưởng bài toán

- Bạn cần tạo ma trận C sao cho mọi giá trị $c_{i,j} = a_i \cdot b_j \mod nm$ là **khác nhau**
- Nếu tồn tại a và b thoả mãn điều đó, in ra chúng
- Nếu không, in `"No"`

✳️ Gợi ý thuật toán nhanh:

- Chỉ khi **n** và **m** là **nguyên tố cùng nhau** (gcd(n, m) = 1) thì có thể tìm được a, b thoả mãn để tất cả tích modulo đều khác nhau. Đây là tính chất cơ bản của **nhóm nhân modulo**.

🇻🇳 Bài C. Sự kiện Khủng hoảng: Thiên thạch (Crisis Event: Meteorite)

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

🦊 **Đề bài**
Sulfox – chú cáo fennec đang chơi trò "Anomaly Collapse", một game chiến thuật theo lượt trên lưới một chiều. Trong chương mới, một sự kiện khủng hoảng **thiên thạch** xuất hiện.

Giả sử có **một chiến trường một chiều** gồm **n ô**, đánh số từ 1 đến n.
Ban đầu, **một số ô có nhân vật** đứng trên đó. Nhiệm vụ của bạn là **đảm bảo tất cả các nhân vật sống sót sau m vòng thiên thạch**.

🪂 Mô tả mỗi vòng

- **Đầu mỗi vòng i (từ 1 đến m)**: tại **ô j**, có `a[i][j]` thiên thạch rơi xuống (có thể là 0).
- Nếu tại ô j có thiên thạch rơi xuống **vào đầu vòng**, và tại đó có nhân vật → **nhân vật sẽ chết ngay lập tức**.
- Sau khi thiên thạch rơi xong, đến **giai đoạn hành động**:
  - Nhân vật còn sống có thể **di chuyển sang ô kề cạnh** bao nhiêu lần tùy ý.
  - **Không thể đi vào ô có thiên thạch**, trừ khi phá vỡ chúng.
    - Để vào ô có `x` thiên thạch, cần tiêu tốn **x điểm** để phá.

🎯 Yêu cầu

Với mỗi test:

- Cho biết ban đầu các nhân vật ở đâu
- Và dữ liệu thiên thạch rơi trong `m` vòng

Bạn cần tính:

- **Tổng số điểm tối thiểu** cần dùng để **tất cả nhân vật sống sót qua toàn bộ m vòng**
- Nếu **không thể**, in ra `-1`.

🧾 Input

- Dòng đầu: số test `T` ($1 \le T \le 10^4$)
- Với mỗi test:

- Dòng 1: hai số nguyên `n` và `m` ($1 \le n, m \le 10^6$), với tổng `n × m` của mọi test $\le 10^6$
- Dòng 2: dãy `c[1..n]` (0 hoặc 1): vị trí ban đầu của nhân vật
- Tiếp theo `m` dòng, mỗi dòng `n` số: `a[i][j]` là số thiên thạch rơi ở vòng i tại ô j

---

📤 Output

- Với mỗi test, in ra:
  - `-1` nếu không thể đảm bảo sự sống sót cho toàn bộ nhân vật
  - Hoặc **số điểm tối thiểu** cần dùng để dọn thiên thạch

📌 Ví dụ

Input

```
2
3 4
1 0 1
0 1 0
2 0 0
0 0 3
4 5 0
1 1
1
1000
```

Output

```
4
-1
```

🧠 Tóm tắt ý tưởng bài toán

- Tại mỗi vòng:
  - Cập nhật lại bản đồ thiên thạch tích luỹ.
  - Kiểm tra xem nhân vật nào **đang ở ô bị rơi thiên thạch → chết**
  - Cho mỗi nhân vật còn sống:
    - Tìm cách **di chuyển** sang ô kề để **tránh thiên thạch vòng sau**
    - Tính toán số **điểm phá thiên thạch tối thiểu** để mở đường đi
- Bài toán là dạng **BFS nhiều nguồn trên đường thẳng**, mỗi lần mô phỏng một vòng, và dồn điểm khi cần mở đường.

Bạn cần:

- Tối ưu BFS bằng hàng đợi hai đầu (0-1 BFS)
- Mỗi lần, chỉ xử lý số lượng ô có thể đi tới, và giữ chi phí nhỏ nhất.

🇻🇳 Bài D. Trò chơi Tích vô hướng (Dot Product Game)

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

🎮 **Đề bài**
Alice và Bob chơi một trò chơi trên hai **hoán vị** $A = [a_1, a_2, ..., a_n]$ và $B = [b_1, b_2, ..., b_n]$ (hoán vị của 1 đến n).
Hai người **luân phiên** chơi, bắt đầu từ **Alice**.

- Alice **chỉ** được hoán đổi hai phần tử trong dãy **A**
- Bob **chỉ** được hoán đổi hai phần tử trong dãy **B**

Mỗi lượt chơi:

- Họ được phép **hoán đổi hai phần tử tùy chọn**, **miễn là tích vô hướng giữa A và B tăng lên**:

$$\sum_{i=1}^{n} a_i \cdot b_i$$

- Ai **không còn nước đi nào** hợp lệ sẽ **thua cuộc**

📌 Vì Alice và Bob đều chơi **tối ưu**, nên người thắng cuộc có thể xác định **ngay từ đầu**.

🔁 Mở rộng sang n ván chơi

Để trò chơi thú vị hơn, ta chơi **n ván**, với biến đổi giữa các ván như sau:

- **Ván 1**: Dùng hoán vị ban đầu A và B
- Với mỗi ván $k = 2$ đến n:
  - Dùng kết quả ván $k - 1$
  - Sau đó, thực hiện **biến đổi** theo lệnh:
    - $t_k \in \{A, B\}$: biến đổi trên dãy A hoặc B
    - Dịch **trái** đoạn từ vị trí l_k đến r_k đúng d_k lần

⏩ *Dịch trái đoạn [l..r] d lần nghĩa là lấy đoạn đó xoay sang trái d bước.*

🧾 Input

- Dòng đầu: số test T ($1 \le T \le 10^5$)
- Mỗi test:
  - Dòng 1: số nguyên n ($1 \le n \le 5 \times 10^5$)
  - Dòng 2: dãy A — hoán vị 1 đến n
  - Dòng 3: dãy B
  - Dòng tiếp theo: $n - 1$ dòng, mỗi dòng gồm:
    - `ti li ri di`: loại dãy cần dịch (A hoặc B), chỉ số bắt đầu, kết thúc và số lần dịch trái

📌 Tổng tất cả n trên mọi test $\le 5 \times 10^5$

📤 Output

- Với mỗi test: in ra một **xâu độ dài n**, ký tự thứ i là:
  - `A` nếu Alice thắng ván thứ i
  - `B` nếu Bob thắng

📌 Ví dụ

Input

```
3
3
1 2 3
1 2 3
A 1 1 1
B 1 1 1
3
1 2 3
2 1 3
A 1 2 1
B 2 2 1
3
1 2 3
2 1 3
A 1 3 1
B 1 2 1
```

Output

```
BBB
ABB
AAB
```

🧠 Tóm tắt ý tưởng

🎯 Mục tiêu:

Tại mỗi ván, xác định xem **ai thắng** nếu họ đều chơi tối ưu.

✅ Điều kiện thắng:

- **Alice thắng** ⟺ Dãy A và B **không đồng hướng**
  (nghĩa là **A không sắp xếp tăng cùng chiều với B**)
- Nếu A và B có **cùng thứ tự tăng** (cùng hướng hoặc đảo ngược), thì:

- Không thể đổi để tăng tích vô hướng
- Người đầu (Alice) **không có nước đi → Bob thắng**

💡 **Nhận xét then chốt:**

- Nếu A = B (cùng chiều), hoặc A = reversed(B) → tích đang **tối đa hoặc tối thiểu** → Không ai có nước đi hợp lệ
- Nếu A và B **khác chiều**, thì luôn tồn tại một cặp chỉ số swap giúp tăng tích

✳️ **Cách làm:**

- Với mỗi game:
  - So sánh **thứ tự tương đối** của A và B (dựa vào **inverse permutation**)
  - Nếu A và B **đồng thứ tự** → Bob thắng (`B`)
  - Ngược lại → Alice thắng (`A`)
- Với mỗi lần dịch:
  - **Update đoạn** tương ứng trong A hoặc B (bằng xoay trái d bước)

Nếu bạn cần giải thuật và code cụ thể, mình có thể cung cấp.

🇻🇳 Bài E. Thắp Sáng Lưới (Light Up the Grid)

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

🧩 **Đề bài**

Little Q đang thử nghiệm trò chơi puzzle với lưới **2 × 2**, mỗi ô có trạng thái **bật (1)** hoặc **tắt (0)**.

Anh ấy có thể thực hiện **4 loại thao tác**, với chi phí tương ứng:

Hành động	Miêu tả	Chi phí
Toggle ô (i, j)	Đảo trạng thái 1 ô cụ thể	a_0
Toggle 1 hàng	Đảo trạng thái cả 1 hàng (2 ô)	a_1
Toggle 1 cột	Đảo trạng thái cả 1 cột (2 ô)	a_2
Toggle toàn bộ	Đảo trạng thái toàn bộ 4 ô	a_3

⚠️ **Vấn đề**: Do lỗi màn hình, **Little Q không biết trạng thái ban đầu**, nhưng **biết trước m trạng thái ban đầu có thể xảy ra**.

Anh ta **muốn tìm 1 chuỗi thao tác** (bất kể trạng thái ban đầu là gì), **sau khi thực hiện xong, mọi trạng thái đều trở thành "bật toàn bộ (1111)"**.

> *"Yêu cầu: Tìm **chi phí tối thiểu** cho chuỗi thao tác như vậy."*

🧾 Input

- Dòng 1: T, a_0, a_1, a_2, a_3
  - T: số test ($1 \le T \le 10^4$)
  - a_0, a_1, a_2, a_3: chi phí của 4 loại thao tác ($1 \le \le 10^6$)
- Với mỗi test:

- Dòng đầu: m — số trạng thái ban đầu có thể (1 ≤ m ≤ 16)
- Tiếp theo là m lưới (2 dòng mỗi lưới, mỗi dòng gồm 2 ký tự 0 hoặc 1). Các lưới **khác nhau**

📤 Output

- Với mỗi test, in ra **chi phí tối thiểu** cần thiết để đảm bảo **mọi trạng thái ban đầu đều trở thành lưới toàn bật**.

📌 Ví dụ

Input

```
2 1000 100 10 1
4
10
00
01
00
00
10
00
01
1
11
11
```

Output

```
1121
```

```
2
```

🧠 Tóm tắt ý tưởng

✅ Quan sát

- Lưới chỉ có **4 ô** → chỉ có **16 trạng thái** (từ `0000` đến `1111`)
- Có **10 thao tác có thể áp dụng** (tất cả các thao tác toggle đơn, hàng, cột, toàn bộ)

✳️ Ý tưởng giải

- Với mỗi thao tác, tạo **một mặt nạ bitmask** (biến đổi 4 bit tương ứng lưới 2×2)
- Xét mọi **tổ hợp thao tác** (tối đa $2^{10} = 1024$)
  - Áp dụng tổ hợp này vào **mọi trạng thái ban đầu**
  - Nếu kết quả luôn là `1111` → ghi nhận chi phí
- Trả về chi phí nhỏ nhất thoả mãn yêu cầu

⚙️ Kỹ thuật

- Encode lưới 2×2 thành 4-bit số nguyên (giả sử theo thứ tự: `(0,0) → bit 0`, `(0,1) → bit 1`, `(1,0) → bit 2`, `(1,1) → bit 3`)
- Với mỗi thao tác → tạo 1 bitmask (ví dụ: toggle hàng 0 → bitmask `0011`)
- Với mỗi tổ hợp thao tác → xor tất cả bitmask → tạo ra **mặt nạ biến đổi**
- Dùng XOR để áp dụng mặt nạ vào trạng thái ban đầu

Nếu bạn muốn mình code giải chi tiết bằng Python hoặc C++ cho bài này, hãy nói nhé.

Dưới đây là bản dịch tiếng Việt và tóm tắt bài toán:

🇻🇳 Bài F. Thắp Sáng Siêu Lập Phương (Light Up the Hypercube)

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

🧩 Đề bài

Trong năm 2040, AI tuỳ chỉnh của Sulfox đã tạo ra một trò chơi mới trên **siêu lập phương n chiều** (hypercube). Trong đó:

- Siêu lập phương n chiều có 2^n **đỉnh (vertex)**, mỗi đỉnh có một **bóng đèn** có thể **bật** hoặc **tắt**.
- Mỗi lần **toggle**, đèn sẽ đổi trạng thái từ bật ↔ tắt.

Có 2^n loại thao tác toggle, được đánh số từ 0 đến $2^n - 1$. Với mỗi thao tác i:

- **Chi phí thao tác** là a_i
- i được viết ở dạng **nhị phân** $b_n b_{n-1} \dots b_1$
  - Với mỗi bit $b_j = 1$: thao tác này **kéo dài theo chiều j**
  - Với mỗi bit $b_j = 0$: thao tác này **cố định chiều j** (tức là nằm trên mặt phẳng có chiều j = const)

- Như vậy thao tác số i sẽ toggle tất cả các đỉnh trong một **k-face** (mặt phụ k chiều), với $k = \text{số bit 1 trong } i$

🔦 Ví dụ:

- $i = 0$ (toàn bit 0): toggle **1 đỉnh duy nhất**
- $i = 1$ (000...001): toggle **2 đỉnh trên cạnh** theo chiều 1
- $i = 2^n - 1$: toggle **tất cả các đỉnh** trong hypercube

🎯 Nhiệm vụ

Vì AI không cho Sulfox nhìn thấy trạng thái ban đầu, và có thể là **bất kỳ** trong số 2^{2^n} trạng thái bật/tắt của 2^n đèn,
anh ấy muốn tìm **một chuỗi thao tác**, sao cho:

- **Bất kể trạng thái ban đầu là gì**, sau khi áp dụng chuỗi thao tác đó, **tất cả đèn đều bật**
- Tính **tổng chi phí nhỏ nhất** của chuỗi thao tác như vậy

Do đáp án có thể rất lớn, hãy in ra **modulo** $998\,244\,353$.

🧾 Input

- Dòng đầu: Số lượng test T ($1 \le T \le 10^4$)
- Với mỗi test:
  - Dòng 1: Số nguyên n ($2 \le n \le 20$)

- Dòng 2: Dãy $a_0, a_1, ..., a_{2^n-1}$ — chi phí của các thao tác ($1 \le$ mỗi phần tử $\le 10^6$)

⛳ Đảm bảo: Tổng 2^n qua tất cả test không vượt quá 2^{20}

📤 Output

- Với mỗi test, in ra **chi phí tối thiểu**, modulo $998\,244\,353$

📌 Ví dụ

Input

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

Output

```
25
65666
```

✅ **Tóm tắt**:

- Mỗi test là một siêu lập phương n chiều.

- Có 2^n thao tác, mỗi thao tác định nghĩa một mặt (k-face) → lật toàn bộ các đỉnh trong mặt đó.
- Mục tiêu: Tìm chuỗi thao tác có tổng chi phí nhỏ nhất sao cho **bất kỳ trạng thái nào**, sau khi áp dụng chuỗi đó → trạng thái **tất cả đèn bật** (`1 1 1 ... 1`)

Kết quả yêu cầu là chi phí tối thiểu, mod 998244353.

Dưới đây là bản **dịch tiếng Việt** và **tóm tắt** của bài toán:

🇻🇳 Bài G. Đoán Đa Giác (Guess the Polygon)

⏱️ **Giới hạn thời gian:** 1 giây
💾 **Giới hạn bộ nhớ:** 1024 MB
📡 **Bài toán tương tác (interactive problem)**

🧩 Đề bài

Bạn được cho một **đa giác đơn** gồm **n đỉnh** (không giao nhau, không có đỉnh trùng, các cạnh liên tiếp không thẳng hàng).
Tuy nhiên, các **đỉnh bị xáo trộn** và bạn **không biết thứ tự nối các đỉnh**.

📌 **Nhiệm vụ** của bạn là **tính diện tích của đa giác đó**, dưới dạng phân số tối giản $\frac{u}{v}$

🔁 Cách tương tác

Bạn **không được biết đa giác**, nhưng bạn có thể thực hiện **tối đa** $n-2$ **truy vấn**.
Mỗi truy vấn là một dòng:

```
? p q
```

Tương đương với hỏi:
➡️ Với đường thẳng $x = \frac{p}{q}$, **tổng độ dài đoạn** nằm bên trong hoặc trên đa giác là bao nhiêu?

Hệ thống sẽ trả về:

```
r s
```

Trong đó $\frac{r}{s}$ là tổng độ dài theo phương y của các đoạn nằm trong đa giác tại đường thẳng $x = \frac{p}{q}$

🟩 Sau khi đã đủ thông tin, bạn có thể **đưa ra câu trả lời cuối cùng**:

```
! u v
```

Trong đó $\frac{u}{v}$ là diện tích đa giác (tối giản: gcd(u, v) = 1)

🧾 Input

- Số lượng test: T ($1 \le T \le 1000$)
- Với mỗi test:
  - Một số nguyên n ($3 \le n \le 1000$): số đỉnh
  - Sau đó là n dòng: mỗi dòng gồm hai số nguyên x, y: tọa độ của 1 đỉnh (đã bị xáo trộn)

📌 Đảm bảo tổng tất cả n của mọi test ≤ 1000

📤 Output

Với mỗi test, sau khi đủ truy vấn, bạn in:

```
! u v
```

Lưu ý:

- **Không in thêm dòng thừa**
- Nhớ **flush** output sau mỗi truy vấn hoặc kết quả:
  - `` `sys.stdout.flush()` `` trong Python
  - `` `fflush(stdout)` `` hoặc `` `cout.flush()` `` trong C++

📌 Ví dụ

Input

```
2
4
0 0
1 3
1 1
3 0
2 1
3
0 0
999 1000
1000 999
1999 999000
```

Output

```
? 1 1
! 3 1
? 1 1
! 1999 2
```

🧠 Tóm tắt

- Mỗi truy vấn bạn chọn một đường thẳng $x = p/q$
- Kết quả trả về là tổng chiều dài các đoạn cắt qua đa giác tại đường thẳng đó
- Diện tích đa giác bằng **tích phân theo phương** x của chiều dài đoạn cắt — tức là:

$$\text{Area} = \int_{x=0}^{x=1000} \text{length}(x)\, dx$$

=> bạn có thể dùng **phép lấy mẫu (sampling)** tại các giá trị x để ước lượng diện tích.

Bạn đã sẵn sàng để viết lời giải tương tác rồi!

Dưới đây là bản dịch tiếng Việt và tóm tắt bài toán:

🇻🇳 Bài H. Bản Đồ Hướng Dẫn (Guide Map)

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

🧭 Đề bài

Byteland có **n thành phố**, giữa **mỗi cặp thành phố** đều có **đường hai chiều**.

Tuy nhiên, **chỉ có (n - 2) đường có phong cảnh** (gọi là "scenery roads").
Và **nếu xây thêm đúng 1 đường nữa có phong cảnh**, bạn có thể đi từ **bất kỳ thành phố nào tới thành phố khác chỉ qua các đường có phong cảnh**.

🏞️ Little Q bắt đầu từ thành phố 1, cầm một **bản đồ hướng dẫn** (guide map) — trên đó **chỉ hiện một số đường** trong Byteland (có thể có hoặc không là đường có phong cảnh).

Little Q thực hiện hành trình như sau:

- **Bắt đầu ở thành phố 1 (được đánh dấu là đã thăm)**
- Tại mỗi thành phố u, nếu tồn tại đường trên bản đồ đến một **thành phố chưa thăm** v → Q sẽ chọn thành phố v **nhỏ nhất** và đi tới đó
- Nếu **không còn đường đi tới thành phố chưa thăm**, Q sẽ **quay ngược lại** đường đã đi trước đó
- Hành trình kết thúc khi Q trở lại thành phố 1 và không còn đường nào để đi

---

🎯 Yêu cầu

Tìm **số bản đồ hướng dẫn khác nhau**, sao cho:

- Q **đi qua tất cả các đường có phong cảnh**
- Q **đi qua không quá 1 đường không có phong cảnh**

Hai bản đồ hướng dẫn khác nhau nếu có ít nhất một đường chỉ xuất hiện trong một trong hai bản đồ.

Trả lời **modulo** 998244353

🧾 Input

- Dòng đầu: n ($2 \le n \le 2 \times 10^5$): số thành phố
- $n-2$ dòng sau, mỗi dòng là một đường có phong cảnh: $u\,v$ ($1 \le u, v \le n$)

📤 Output

- In ra số lượng bản đồ hướng dẫn thỏa mãn điều kiện, **modulo** 998244353

📌 Ví dụ

Input

```
4
1 4
2 3
```

Output

```
6
```

🧠 Tóm tắt ý tưởng bài toán

- Có n thành phố → tổng số cạnh để tạo thành cây là $n - 1$
- Đề bài cho $n - 2$ cạnh có phong cảnh → **thiếu 1 cạnh để thành cây**

→ Nếu ta **xây thêm 1 cạnh mới bất kỳ**, thì tập các cạnh phong cảnh sẽ tạo thành một **đồ thị liên thông** (cây hoặc đồ thị chứa đúng 1 chu trình)

→ Yêu cầu là: từ các cạnh phong cảnh + **chọn thêm tối đa 1 cạnh không phong cảnh**
→ **Q phải đi hết tất cả các cạnh có phong cảnh, và đi qua không quá 1 cạnh khác**

→ Bản đồ hướng dẫn chính là một **cây DFS duy nhất** chứa tất cả cạnh phong cảnh + 1 cạnh tự do.

Từ đây có thể dùng **tổ hợp + cấu trúc đồ thị** để đếm số cách chọn thêm 1 cạnh và tạo bản đồ hợp lệ.

Nếu bạn cần mình giải thích hướng tư duy chi tiết và công thức, cứ nhắn nhé.

Dưới đây là bản dịch tiếng Việt và tóm tắt bài toán:

🇻🇳 Bài I. Cây Đang Lớn (Growing Tree)

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

🌱 Đề bài

Little Q trồng một cái cây. Ban đầu cây chỉ có **1 nút gốc** là nút số 1. Mỗi sáng, mỗi **nút lá** u sẽ mọc ra **2 nhánh mới** như sau:

- Một nhánh từ u đến $2u$, có độ dài a_{2u}
- Một nhánh từ u đến $2u+1$, có độ dài a_{2u+1}

Sau đó, nút u **không còn là lá**, và hai nút mới $2u$, $2u+1$ trở thành **lá mới**.

Cứ như vậy cây sẽ **mọc trong n ngày**, tạo thành:

- Tổng số nút: $2^{n+1}-1$
- Số nút lá: 2^n
- Tổng số **nhánh** (cạnh): $2^{n+1}-2$

🌳 **Chiều chiều**, Little Q có thể chọn **1 nhánh đã mọc** để **cắt tỉa**, tức là **đổi độ dài của nhánh đó thành số nguyên dương bất kỳ** (có thể lớn hơn hoặc nhỏ hơn độ dài ban đầu).
Hoặc anh ấy **không cắt gì cả**.

🎯 Mục tiêu

Sau khi cây mọc xong:

- Với **mỗi đường từ gốc đến lá**, tính tổng độ dài các nhánh trên đường đó
- ***Yêu cầu: các tổng độ dài này phải **khác nhau hoàn toàn**.*

Nhiệm vụ của bạn là:

- Tìm **số nhánh tối thiểu cần phải cắt tỉa**
- Hoặc in `-1` nếu **không thể** nào tạo ra đường đi khác nhau được

---

🧾 Input

- Dòng đầu: số test T ($1 \le T \le 1000$)
- Với mỗi test:
  - Dòng 1: số ngày n ($1 \le n \le 10$)
    ⚠️ Tổng số test với $n > 7$ **không vượt quá 10**
  - Dòng 2: $2^{n+1} - 2$ số nguyên $a_2, a_3, ..., a_{2^{n+1}-1}$: độ dài các nhánh

---

📤 Output

- Với mỗi test, in ra:
  - Số nhánh **ít nhất cần phải cắt tỉa** để đảm bảo đường từ gốc tới mỗi lá có tổng độ dài **khác nhau**
  - Hoặc `-1` nếu **không thể**

📌 Ví dụ

Input

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

Output

```
1
2
-1
```

🧠 Tóm tắt ý tưởng

- Với n, có 2^n đường từ gốc tới lá
- Mỗi đường là duy nhất vì cây nhị phân hoàn chỉnh
- Mỗi đường có tổng độ dài (sum of weights)
- Nếu tồn tại **hai đường có cùng tổng**, phải **sửa ít nhất 1 cạnh trong cây** để phá sự trùng lặp

💡 Tối ưu: dùng **DFS + HashMap** để lưu các tổng, và đếm số lần **phải chỉnh sửa để loại bỏ trùng lặp**, chọn cách sửa **ít nhánh nhất**

Bạn đã sẵn sàng để viết lời giải rồi! Nếu cần mình hỗ trợ chi tiết về thuật toán hay code, cứ nói nhé.

🇻🇳 Bài J. Hãy Khiến Họ Tin (Make Them Believe)

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

🏆 Đề bài

Bạn được cung cấp danh sách **8 đội tuyển** tham dự vòng loại trực tiếp của CKTG 2024, theo thứ tự từ trên xuống trong sơ đồ nhánh 8 đội.
Mỗi đội có:

- **Tên đội** (tối đa 3 ký tự, gồm chữ in hoa và số)
- **Chỉ số sức mạnh** t_i (tất cả khác nhau)

👉 Bạn cần **mô phỏng toàn bộ vòng loại trực tiếp (knockout)** với quy tắc:

- **Sức mạnh cao hơn sẽ thắng**
- Đội thắng tiến vào vòng tiếp theo
- Trận **Chung kết (Final)** là vòng 3 → xác định:
  - **Nhà vô địch (Champion)**: đội thắng
  - **Á quân (Runner-up)**: đội thua

🧾 Input

Gồm đúng 8 dòng, mỗi dòng:

```
Tên_đội  Chỉ_số_sức_mạnh
```

- Tên chỉ gồm chữ in hoa hoặc số, độ dài ≤ 3

- Tất cả tên và sức mạnh là **khác nhau**

📤 Output

Một dòng duy nhất:

```
A beats B
```

Trong đó:

- `A`: tên đội **vô địch**
- `B`: tên đội **á quân**

📌 Ví dụ

Input

```
LNG 55
WBG 65
HLE 70
BLG 75
TES 48
T1 80
GEN 60
FLY 50
```

Output

```
T1 beats BLG
```

🧠 Tóm tắt cách làm

- Mỗi trận đấu là so sánh **2 đội liên tiếp** theo nhánh
- Tiến hành 3 vòng:
  - **Tứ kết**: 4 trận → còn 4 đội
  - **Bán kết**: 2 trận → còn 2 đội
  - **Chung kết**: 1 trận → xác định đội vô địch và á quân
- Mỗi trận chỉ cần so sánh `sức mạnh` để chọn đội thắng

Bạn đã sẵn sàng code mô phỏng đơn giản này rồi! Nếu cần mình cung cấp code mẫu bằng Python hoặc C++, hãy nhắn nhé.

Dưới đây là bản dịch tiếng Việt và tóm tắt của **bài toán K - Fragile Pinball**:

🇻🇳 Bài K. Bóng Phản Xạ Mỏng Manh (Fragile Pinball)

⏱️ **Giới hạn thời gian:** 3 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📜 Đề bài

Bạn có một **đa giác lồi** gồm n đỉnh ($3 \le n \le 6$), được cho dưới dạng tọa độ 2D theo thứ tự **ngược chiều kim đồng hồ**.

Một **quả bóng nhỏ xíu** (xem như điểm) bắt đầu **từ bên trong hoặc trên cạnh** của đa giác, và di chuyển **thẳng với vận tốc không đổi**.

Bạn có thể:

- **Bật một cạnh** tại **một thời điểm bất kỳ**, nhưng **chỉ một lần duy nhất cho mỗi cạnh**.
- Nếu bóng **đang nằm chính xác trên cạnh được bật**, bóng sẽ **phản xạ** theo quy tắc vật lý: **góc tới = góc phản xạ**.

Lưu ý:

- Nếu bóng **đang nằm trên hai cạnh cùng lúc** (ví dụ ở đỉnh), bạn **chỉ có thể bật từng cạnh một lần**, nhưng có thể bật liên tiếp (2 phản xạ liên tiếp).
- Tổng số **phản xạ tối đa** là k, nếu vượt quá bóng sẽ vỡ (không thể tiếp tục).

🎯 Nhiệm vụ

Tính **quãng đường dài nhất** mà quả bóng có thể di chuyển **trong đa giác lồi**, với giới hạn phản xạ là:

- $k = 0, 1, 2, ..., n$

🧾 Input

- Dòng 1: số nguyên n ($3 \le n \le 6$) — số đỉnh của đa giác
- n dòng tiếp theo: mỗi dòng chứa 2 số nguyên x_i, y_i (tọa độ đỉnh thứ i)

🔒 Đảm bảo:

- Các đỉnh là duy nhất
- Ba đỉnh bất kỳ không thẳng hàng
- Đa giác là **lồi**, theo thứ tự ngược chiều kim đồng hồ

📤 Output

- In ra $n + 1$ dòng
- Dòng thứ i: **quãng đường dài nhất bóng có thể đi**, nếu cho phép tối đa $i - 1$ lần phản xạ
- Kết quả dạng số thực, sai số tuyệt đối hoặc tương đối ≤ 10^{-6}

📌 Ví dụ

Input

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

Output

```
5.000000000000000000
8.000000000000000000
```

```
8.868185038797563409
12.210024810881955830
```

🧠 Tóm tắt yêu cầu

- Bóng đi **thẳng**, chỉ **đổi hướng khi phản xạ**
- Mỗi cạnh chỉ **cho phép phản xạ một lần**
- Tìm đường đi dài nhất khi giới hạn phản xạ là 0, 1, ..., n

Bài này yêu cầu xử lý hình học 2D kết hợp **truy vết tia phản xạ**, và duyệt theo số lần phản xạ (tối đa 6 → có thể brute-force kết hợp memo hóa).

Nếu bạn cần lời giải hoặc ý tưởng thuật toán chi tiết, mình có thể cung cấp thêm.

Dưới đây là bản dịch tiếng Việt và tóm tắt cho **bài toán L – The Grand Contest**:

🇻🇳 Bài L. Cuộc thi vĩ đại (The Grand Contest)

⏱️ **Giới hạn thời gian:** 3 giây
💾 **Giới hạn bộ nhớ:** 1024 MB

📘 Mô tả bài toán

Có **2 đội** (Team 1 và Team 2) tham gia một cuộc thi lập trình, với **n lượt nộp bài** trong toàn bộ kỳ thi.

- Mỗi lượt nộp bao gồm:
  - **a**: số đội (1 hoặc 2)
  - **b**: số bài toán ($1 \le b \le 10^9$)
  - **c**: thời gian nộp bài ($1 \le c \le 10^{12}$)
  - **d**: kết quả (0: sai, 1: đúng)

🧮 Tính điểm

- **Một bài được tính là "đã giải" nếu có nộp đúng**
- Thứ hạng đội được xác định theo:
  1. **Số bài giải được (càng nhiều càng tốt)**
  2. **Tổng thời gian** của các bài đã giải (càng ít càng tốt)
     - Mỗi bài đã giải có thời gian tính là:
       `thời gian nộp đúng đầu tiên + p × (số lần sai trước đó)`
- Nếu bằng điểm và bằng thời gian → **team 1 xếp trước**

🔁 Chức năng “xóa khoảng thời gian [L, R]”

Bạn được phép **xóa một khoảng thời gian [L, R]** sao cho:

- Những lượt nộp **trước L**: không thay đổi
- Những lượt nộp **trong [L, R]**: đổi thời gian nộp thành L

- Những lượt nộp **sau R**: giảm thời gian đi `(R - L)`

❗ **Mục tiêu**: tìm **khoảng [L, R] ngắn nhất** (tức là `R - L` nhỏ nhất) và **L nhỏ nhất** có thể **đảo ngược thứ hạng giữa hai đội**.

Nếu **không có cách nào** đảo ngược được thứ hạng → in `-1`

🧾 Input

- Dòng đầu: số test T ($1 \le T \le 4 \times 10^5$)
- Với mỗi test:
  - Dòng đầu: n, p — số lượt nộp, số phút phạt
  - Tiếp theo n dòng, mỗi dòng:
    - a, b, c, d: đội nào, bài nào, thời gian, kết quả (đúng/sai)

🔒 **Bảo đảm**:

- c **không giảm dần**
- Tổng n của mọi test ≤ 4×10^5

📤 Output

Với mỗi test:

- Nếu tồn tại: in ra `L R` — khoảng thời gian ngắn nhất làm đổi vị trí
- Nếu không tồn tại: in `-1`

📌 Ví dụ

Input

```
2
6 20
1 1 60 0
2 2 60 0
2 2 120 1
1 2 180 1
1 2 180 0
2 2 300 1
2 20
1 1 300 1
2 2 300 1
```

Output

```
120 160
-1
```

🧠 Tóm tắt hướng giải

- Với mỗi test:
  1. Mô phỏng tính thứ hạng gốc của hai đội
  2. Duyệt tất cả **cặp thời điểm (L, R)** ứng với các mốc `c_i`
     - Số lượng là $O(n^2)$ nhưng có thể tối ưu bằng kỹ thuật sliding window hoặc chỉ xét mốc quan trọng
  3. Với mỗi (L, R), cập nhật thời gian theo quy tắc → tính lại thứ hạng
  4. Nếu vị trí bị đảo → cập nhật câu trả lời

⏳ Bài yêu cầu kỹ thuật **mô phỏng thời gian** linh hoạt, cần tổ chức dữ liệu tốt và tối ưu hoá vòng lặp.

Nếu bạn cần lời giải cụ thể bằng Python hoặc hướng cài đặt chi tiết, mình có thể hỗ trợ thêm.

Dưới đây là bản **dịch tiếng Việt** và **tóm tắt bài toán M – Obliviate, Then Reincarnate**:

🇻🇳 Bài M. Quên Lãng và Tái Sinh

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

🏨 Mô tả bài toán

Bạn quản lý một **Khách sạn Hilbert Vũ trụ**, nơi có **vô hạn phòng**, được đánh số bằng mọi **số nguyên** $\mathbb{Z}$ (bao gồm cả âm, 0 và dương).

Khách sạn có:

- n **tầng**, mỗi phòng được gán vào tầng theo **chỉ số dư mod n**:
  → phòng x và y ở cùng tầng khi $x \equiv y \pmod{n}$

🧾 Di chuyển khách

Có m **lệnh di chuyển**, mỗi lệnh là cặp (a, b):

- Với mọi phòng $x \equiv a \pmod{n}$, nếu khách đang ở đó thì **họ sẽ chuyển sang** $x + b$
- Lệnh có thể áp dụng **vô hạn lần**, **bất kỳ thứ tự**, **áp dụng cho đúng tầng**

⚠️ Nhiều khách có thể **chồng lên nhau**, không vấn đề gì.

❓ Truy vấn

Có q truy vấn. Mỗi truy vấn cho một khách ban đầu ở phòng số x.
Hỏi rằng:

> *"**Khách đó có thể đi đến vô hạn phòng khác nhau không** nếu áp dụng các lệnh di chuyển một cách tự do?"*

Bạn cần in `Yes` nếu **có thể đi đến vô hạn phòng**, và `No` nếu **chỉ đến được hữu hạn phòng**.

🧾 Input

- Dòng đầu: n, m, q — số tầng, số lệnh, số truy vấn ($1 \le \le 5 \times 10^5$)
- m dòng sau: mỗi dòng a, b — lệnh di chuyển ($-10^9 \le a, b \le 10^9$)
- q dòng sau: mỗi dòng là số nguyên x — phòng ban đầu của khách

---

📤 Output

Với mỗi truy vấn, in:

- `Yes` nếu khách có thể đến vô hạn phòng khác nhau
- `No` nếu chỉ đến được số phòng hữu hạn

📌 Ví dụ

Input

```
3 2 3
1 1
-1 3
1
2
3
```

Output

```
Yes
Yes
No
```

🧠 Ý tưởng tóm tắt

🔁 Nhận định chính:

- Mỗi tầng $t \in \{0, 1, ..., n-1\}$ sẽ có tập các bước di chuyển b từ các lệnh có $a \equiv t \pmod{n}$
- Với mỗi tầng t, nếu **gcd của tập các b khác 0** → ta có thể cộng nhiều lần → sinh ra vô hạn vị trí

- Nếu **gcd = 0** (tức là không có $b \neq 0$) → chỉ đến được **một vài phòng hữu hạn**

✅ **Giải pháp:**

1. Với mỗi tầng t: lưu tập các b sao cho $a \equiv t \pmod{n}$
2. Tính **gcd** của các b cho tầng đó (nếu tập rỗng hoặc toàn số 0 → gcd = 0)
3. Với mỗi truy vấn phòng x:
   - Tính $t = x \mod n$
   - Nếu **gcd của tầng t > 0** → in `Yes`
   - Ngược lại → in `No`

Nếu bạn cần mình viết lời giải bằng Python hoặc tối ưu code, mình có thể hỗ trợ tiếp nhé!