universal cup 3 stage 20 : Kunming

Created: 5/5/2025 22:58
Updated: 5/5/2025 23:03
Exported: 5/5/2025 23:03

🦠 Bài A. Phần mềm diệt virus (Antivirus)

⏰ Giới hạn thời gian: 2 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Hyrule đang bước vào mùa cúm.

Vương quốc này có **n** thành phố, được kết nối với nhau bằng **m** con đường **có hướng**. **Thủ đô** là **thành phố số 1**, và từ **mọi thành phố khác đều có đường đến thủ đô**.

Mùa cúm kéo dài **q ngày**. Vào **giữa trưa** của ngày thứ `i`, virus sẽ bắt đầu lan truyền từ **thành phố** a_i.
Virus lan rất nhanh, nên **mọi thành phố có thể đi được từ** a_i **(theo hướng các cạnh) sẽ ngay lập tức bị lây nhiễm**.

Nếu virus **đến được thủ đô**, sẽ gây ra tổn thất kinh tế trị giá b_i vào ngày thứ `i`.

Để ngăn virus, mỗi đêm (bao gồm **đêm trước ngày 1**), Auru có thể triển khai **1 bộ lọc virus** tại **một thành phố** bất kỳ với **chi phí** c_i.

- Thành phố có bộ lọc sẽ **không bị nhiễm virus**.
- Virus **không thể truyền qua** thành phố đó.
- Nếu virus bắt đầu từ một thành phố đang được lọc, nó **biến mất ngay lập tức**.

⚠️ **Chỉ được duy trì 1 bộ lọc tại 1 thời điểm**, nên nếu triển khai bộ lọc mới ở thành phố khác thì bộ lọc cũ sẽ mất hiệu lực.

Yêu cầu:
Với mỗi ngày `i = 1, 2, ..., q`, hãy tính **giá trị nhỏ nhất của tổng chi phí**:

$$\text{Chi phí tổn thất kinh tế} + \text{Chi phí triển khai bộ lọc}$$

(khi chỉ xét các virus từ ngày 1 đến ngày `i`).

📥 Input

Nhiều test, dòng đầu là số bộ test **T** ($1 \le T \le 10^4$)

Mỗi test gồm:

- 1 dòng gồm **n, m, q**
- **m dòng** tiếp theo là các cạnh có hướng `u_i v_i` (đi từ `u_i` đến `v_i`)
- 1 dòng gồm **n số nguyên** `c_1, c_2, ..., c_n`: chi phí gắn bộ lọc tại từng thành phố
- **q dòng**, mỗi dòng gồm `a_i, b_i`: nơi virus bắt đầu ngày `i`, và tổn thất nếu thủ đô bị nhiễm ngày đó

Ràng buộc:

- $2 \le n \le 10^5, n-1 \le m \le 2 \times 10^5, 1 \le q \le 10^5$
- Tổng n, m, q trên tất cả test **không vượt quá** $10^5, 2 \times 10^5, 10^5$

📤 Output

Với mỗi test, in ra **q số**, mỗi số là chi phí nhỏ nhất sau ngày thứ `i`.

✨ Tóm tắt ý tưởng bài toán

- Ta có một đồ thị có hướng (DAG), từ mỗi đỉnh a_i, cần xác định xem có **đường nào dẫn tới thủ đô (đỉnh 1)** không.
- Nếu có, và **không có bộ lọc** chặn lại, thì phải **chịu tổn thất** b_i.
- Mỗi đêm được gắn 1 bộ lọc (một đỉnh), bộ lọc tồn tại **tại đúng 1 đỉnh duy nhất**, các đỉnh còn lại virus lan bình thường.
- Mỗi ngày có thể chọn một bộ lọc **khác**, nhưng chỉ được **1 cái tại một thời điểm**.

💡 Định hướng lời giải:

- Tiền xử lý đồ thị **ngược lại** (bắt đầu từ thủ đô, BFS/ngược lại) để biết những đỉnh nào **có thể dẫn tới thủ đô** (những đỉnh nguy hiểm).
- Với mỗi virus khởi phát ở a_i, nếu nó có đường tới thủ đô, thì ta cần xem có thể **chặn nó bằng cách đặt bộ lọc ở 1 đỉnh nào đó trong đường đi hay không**.
- Duy trì danh sách các đỉnh có thể "chặn" nhiều virus với chi phí tốt nhất.

→ Bài toán kết hợp giữa:

- Đồ thị (BFS ngược để tìm ảnh hưởng virus đến thủ đô)
- Cập nhật động (mỗi ngày chọn lọc tối ưu)
- Tối ưu hóa trên chi phí

Bạn muốn mình viết lại đề này bằng LaTeX luôn không?

🧩 Bài B. Dãy ngoặc (Brackets)

⏰ Giới hạn thời gian: 3 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Betty có một **xâu ngoặc** `s` dài `n`, gồm **8 loại dấu ngoặc**:

- `()`
- `[]`
- `{}`
- `<>`

Cô ấy có **m đoạn con** của `s`, mỗi đoạn t_i được tạo bằng cách **lấy từ chỉ số** l_i **đến** r_i trong `s`, tức là:

$$t_i = s[l_i..r_i]$$

Betty muốn **tạo ra nhiều nhất các cặp (a_i, b_i)** sao cho:

1. Mỗi đoạn t_i **chỉ xuất hiện trong tối đa 1 cặp** (không trùng).
2. Khi **nối** t_{a_i} **và** t_{b_i}, ta được **một dãy ngoặc hợp lệ**.

🔍 **Định nghĩa dãy ngoặc hợp lệ**:

- Rỗng (`""`) là hợp lệ.
- Nếu `x` là hợp lệ, thì `LxR` cũng hợp lệ với `L, R` là cùng loại (ví dụ: `(` và `)`).

- Nếu `x = y₁ + y₂`, và cả `y₁, y₂` đều hợp lệ, thì `x` cũng hợp lệ.

📥 Input

Nhiều test, dòng đầu là số test **T** ($1 \le T \le 10^4$).

Mỗi test gồm:

- Một dòng chứa hai số nguyên `n, m` ($1 \le n, m \le 5 \times 10^5$).
- Một dòng chứa chuỗi `s` độ dài `n` chỉ gồm 8 dấu ngoặc `"()[]{}<>"`.
- `m dòng`, mỗi dòng gồm l_i, r_i ($1 \le l_i \le r_i \le n$) mô tả đoạn t_i.

Ràng buộc:

- Tổng `n` của tất cả test ≤ 5×10^5
- Tổng `m` của tất cả test ≤ 5×10^5

📤 Output

Với mỗi test, in ra một số nguyên — **số cặp tối đa (a_i, b_i)** có thể ghép nối tạo thành dãy ngoặc hợp lệ.

✨ Tóm tắt ý tưởng bài toán

- Có m đoạn con, cần **chọn các cặp không trùng** và **nối chúng lại thành dãy ngoặc hợp lệ**.

- Mỗi cặp `(a, b)` ứng với nối $t_a + t_b$, cần kiểm tra xem nó có hợp lệ không.

💡 Ý tưởng xử lý:

- Với mỗi t_i, tính **số lượng dư (chênh lệch)** của từng loại dấu mở/đóng, ví dụ:
  - Dấu `(`: dư mở `+1`, đóng `-1`, không được âm khi duyệt từ trái sang phải.
- Với mỗi đoạn, lưu dạng **signature** biểu diễn "cân bằng" của đoạn đó.
- Đếm số lượng đoạn có signature tương thích để ghép thành đoạn hợp lệ.
- Ghép các đoạn có signature đối lập với nhau (ví dụ: một đoạn mở, một đoạn đóng tương ứng).

→ Bài toán cần xử lý nhiều đoạn, nên cần tổ chức tính toán signature hiệu quả và đếm nhanh số lượng tương ứng.

Bạn muốn mình viết lại đề này bằng LaTeX không?

□ Bài C. Đồng xu vàng (Coin)

⏰ **Giới hạn thời gian: 2 giây**

🧠 **Giới hạn bộ nhớ: 1024 MB**

📝 Đề bài (dịch tiếng Việt)

Chính phủ quyết định cắt giảm ngân sách hải quân và **phân phát vàng trực tiếp cho cướp biển**.
Kết quả là **số cướp biển bị loại bỏ còn nhiều hơn cả hải quân từng làm được**.

Các cướp biển vừa cướp được **một đồng vàng khổng lồ**!
Họ muốn chọn người giữ đồng xu vàng bằng **một trò chơi sinh tồn** như sau:

- Ban đầu có `n` tên cướp biển xếp thành hàng theo thứ tự từ 1 đến n.
- Họ tiến hành loại bỏ theo quy tắc:
  - Bắt đầu từ vị trí **1**, cứ **cách `k` tên**, loại bỏ một tên.
  - Tức là loại các tên ở vị trí: `1, 1 + k, 1 + 2k, ..., 1 + ([n/k] - 1)k`.
- Sau mỗi vòng loại, ta **còn lại một số tên cướp**, và **tiếp tục quy trình trên** với dãy còn lại.
- Lặp lại cho đến khi **chỉ còn một tên duy nhất**, và **tên đó nhận được đồng vàng**.

Charlie là tên cướp thông minh nhất.
Anh ta muốn biết: **nên đứng ở vị trí ban đầu nào để sống sót cuối cùng**?

📩 Input

Nhiều test.
Dòng đầu tiên là số lượng test **T** ($1 \le T \le 100$)

Mỗi test gồm:

- Một dòng chứa hai số nguyên `n, k`
  ($2 \le n, k \le 10^{12}$)

Ràng buộc:
Tổng các `n` và `k` trong toàn bộ các test không vượt quá 10^{12}

📤 Output

Với mỗi test, in ra vị trí ban đầu (đánh số từ 1) của tên cướp **sống sót cuối cùng**.

📌 Ví dụ

Input	Output
6 2	4
8 3	8
10000 2	8192
1919810 114514	1919805

✨ Tóm tắt đề bài

- Một bài toán **loại bỏ tuần hoàn** (giống bài toán Josephus).
- Mỗi vòng: bắt đầu từ 1, loại mọi người cách `k` bước.
- Lặp lại đến khi còn 1 người → tìm vị trí ban đầu của người đó.

💡 Ý tưởng xử lý:

- Bài toán cần thuật toán **O(log n)** hoặc gần như **O(1)** để giải cho `n` rất lớn.
- Không thể mô phỏng từng vòng → cần quy hoạch động hoặc công thức dạng **Josephus với bước nhảy k**.

Bạn có muốn mình trình bày bài này bằng LaTeX không?

□ Bài D. Búp bê lồng nhau (Dolls)

⏰ **Giới hạn thời gian: 1 giây**

🧠 **Giới hạn bộ nhớ: 1024 MB**

📝 Đề bài (dịch tiếng Việt)

David vừa nhận được `n` **búp bê Nga lồng nhau**, mỗi con có **kích thước khác nhau**.
Anh ấy xếp chúng thành một hàng từ trái sang phải, vị trí `i` chứa búp bê có kích thước a_i.

Khi một vị trí chứa nhiều búp bê (sau khi gộp), ta định nghĩa:

- l_i: kích thước nhỏ nhất trong vị trí `i`
- r_i: kích thước lớn nhất trong vị trí `i`

Hai vị trí **liền kề** `i` và `i+1` có thể được **gộp lại** nếu:

- $r_i < l_{i+1}$ **hoặc** $r_{i+1} < l_i$

Sau khi gộp:

- Tất cả búp bê từ vị trí `i` và `i+1` gộp thành một tập hợp mới
- Đặt tập hợp mới tại vị trí `i`
- Các phần tử sau đó dồn về bên trái (giảm số lượng vị trí đi 1)

✅ **Mục tiêu**: Thực hiện tối đa số lần **gộp hợp lệ**, theo chiến lược tối ưu.

📥 Input

Nhiều test.
Dòng đầu là số test `T` ($1 \le T \le 10^4$)

Mỗi test gồm:

- Một dòng chứa số nguyên `n` ($1 \le n \le 10^5$): số lượng búp bê
- Một dòng gồm `n` số nguyên khác nhau `a₁, a₂, ..., aₙ` ($1 \le a_i \le n$)

Ràng buộc:

- Tổng tất cả `n` trong mọi test **không vượt quá 10^5**

📤 Output

Với mỗi test, in ra một số nguyên — **số lượng gộp tối đa có thể thực hiện được**.

📌 Ví dụ

Input	Output
4 2 1 4 3	3
4 1 4 2 3	3
4 3 1 4 2	2

Input	Output
5 1 3 5 2 4	3
5 1 4 2 5 3	3
5 2 5 3 1 4	3
6 1 3 6 5 2 4	4
6 2 5 1 3 6 4	4

✨ Tóm tắt ý tưởng bài toán

- Mỗi lần gộp ta **giảm số lượng phần tử đi 1**, nên tối đa có thể gộp `n-1` lần.
- Nhưng phải **thoả điều kiện hợp lệ** giữa hai cụm:
  - Hai cụm `A`, `B` chỉ gộp được nếu **toàn bộ A nhỏ hơn toàn bộ B** hoặc ngược lại.

💡 **Chiến lược hiệu quả**:

- Duyệt từ trái qua phải, chia thành **các phân đoạn tăng liên tiếp**:
  - Trong mỗi đoạn tăng này, không thể gộp các phần tử với nhau.
  - Chỉ có thể gộp các đoạn **khác nhau**.
- Kết luận: **Số lượng lần gộp tối đa = n - số đoạn tăng liên tiếp**

📌 Tức là:

```
ans = n - số đoạn tăng liên tiếp
```

Bạn có muốn mình trình bày bài này bằng LaTeX không?

🌳 Bài E. Truy tìm trọng số (Extracting Weights)

⏰ **Giới hạn thời gian: 1 giây**

🧠 **Giới hạn bộ nhớ: 1024 MB**

📌 **Dạng: Bài toán tương tác (interactive)**

📝 Đề bài (dịch tiếng Việt)

Emily có một **cây gồm `n` đỉnh**, đỉnh 1 là **gốc**, mỗi đỉnh `i` có một **trọng số** w_i với $w_1 = 0$.
Emily muốn bạn **đoán trọng số** của tất cả các đỉnh.

Bạn có thể thực hiện **tối đa `n` truy vấn**. Mỗi truy vấn là một cặp (u_i, v_i):

- Nếu đường đi đơn giản từ u_i đến v_i có đúng `k` **cạnh**, bạn sẽ nhận được **XOR của tất cả các trọng số trên đường đi này** (bao gồm u_i và v_i).
- Nếu **không có đúng `k` cạnh**, bạn nhận được `-1`.

⛔ Emily **sẽ không trả lời từng truy vấn ngay lập tức**. Bạn phải gửi tất cả truy vấn cùng lúc và **nhận lại toàn bộ kết quả một lượt**.

🎯 Mục tiêu

- Nếu bạn **không thể** xác định toàn bộ trọng số trong **≤ n truy vấn**, hãy in `No` và thoát.

- Nếu có thể, in `Yes`, gửi `q` truy vấn, nhận lại kết quả, và **đoán đúng toàn bộ trọng số từ** w_2 **đến** w_n (vì $w_1 = 0$ đã biết).

📥 Input

- Một test duy nhất.
- Dòng đầu: `n k`
- $n - 1$ dòng tiếp: các cạnh x_i y_i tạo thành cây.
- Đảm bảo $0 \le w_i < 2^{30}$.

🔄 Giao tiếp tương tác

1. Trước tiên:
   - In `Yes` hoặc `No` để nói có thể giải được hay không.
2. Nếu là `Yes`:
   - In truy vấn:

     ```
     ? q u₁ v₁ u₂ v₂ ... u_q v_q
     ```

   - Nhận lại `q` số nguyên: đáp án từng truy vấn.
3. Sau đó:
   - In đáp án cuối cùng:

```
! w2 w3 ... wn
```

🛑 Lưu ý: Truy vấn là **đồng loạt một lần**, không được hỏi từng cái.

📌 Ví dụ

Input:

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

Output:

```
YES
? 3 1 2 2 3 2 4
! 1 2 3
```

✨ Tóm tắt ý tưởng bài toán

- Bạn được phép hỏi những **cặp `(u, v)` sao cho khoảng cách đúng `k` cạnh**, từ đó nhận về XOR của tất cả trọng số trên đường đi từ **`u → v`**.

🎯 Mục tiêu: Dùng tối đa **`n`** truy vấn để **khôi phục trọng số `w2` đến `wn`** (biết **`w1 = 0`**).

💡 **Ý tưởng giải quyết:**

1. Duyệt BFS hoặc DFS từ gốc để biết khoảng cách của mọi đỉnh đến đỉnh 1 (gốc).
2. Gộp thành từng lớp (level) các đỉnh có khoảng cách `d` từ root.
3. Với `k` cố định, muốn lấy thông tin từ đỉnh `u`, chỉ được hỏi cặp `(u, v)` nếu `dist(u, v) == k`.
4. Với mỗi đỉnh `v`, nếu có một đỉnh `u` cách `k` cạnh, bạn có thể dùng truy vấn `(u, v)` để lấy `w[u] ⊕ ... ⊕ w[v]`.

📈 Nếu bạn có thể xây dựng được **tập đủ thông tin** để suy ra toàn bộ `w[i]`, thì in `YES` và tiếp tục.

Ngược lại (ví dụ: cây là chuỗi dài, `k` quá nhỏ để kết nối đủ các đỉnh), thì **in** `NO`.

Bạn có muốn mình viết LaTeX lại bài này không?

🌸 Bài F. Hoa số học (Flowers)

⏰ Giới hạn thời gian: 3 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Frank yêu những con số đẹp.
Khi đang tưới hoa trong vườn, anh tưởng tượng nếu mỗi bông hoa **mọc ra số** thì sẽ thật tuyệt.
Thế là anh đã nghĩ ra khái niệm **“hoa số học” (number flower)** — một loại **đồ thị vô hướng đặc biệt** thỏa mãn 3 điều kiện sau:

🌼 Định nghĩa hoa số học (number flower)

Cho đồ thị vô hướng **liên thông** có `n` đỉnh, được đánh số từ `1` đến `n`. Gọi đồ thị đó là một **hoa số học** nếu:

1. Đồ thị có **chính xác `n - 1` cạnh**
   (→ tức là là một cây).

2. **Tất cả các đỉnh chỉ có bậc ≤ 2**, **ngoại trừ đỉnh 1**.
   (→ tức là các đường là nhánh (dây) nối về gốc 1, gốc có thể có nhiều nhánh).

3. Các đỉnh nối trực tiếp với **đỉnh 1** được gọi là **đỉnh chính** (**key nodes**):

   - Tất cả các **key node phải là các số nguyên tố cùng nhau (pairwise coprime)**.

   - Mỗi đỉnh không chính (không phải đỉnh 1 hay key node) sẽ thuộc về một nhánh của một key node sao cho:

     - Giá trị của đỉnh đó là **bội số** của đỉnh key gần nhất.

     - Trên đường đi đơn từ đỉnh đó tới key node, các nhãn phải **giảm dần**.

🎯 Yêu cầu

Cho số nguyên `n` và số nguyên tố `p`.
Hãy **tính số lượng hoa số học khác nhau** với `n` đỉnh, **modulo** `p`.

Hai đồ thị được coi là giống nhau nếu chúng có cùng danh sách cạnh `(u, v)`.

📥 Input

Một dòng chứa hai số nguyên:

- `n` — số đỉnh ($1 \le n \le 10^{10}$)
- `p` — một số nguyên tố lớn hơn 10^8 ($10^8 < p < 10^9$)

📤 Output

In ra số lượng **hoa số học** khác nhau có `n` đỉnh, **modulo** `p`.

📌 Ví dụ

Input	Output
`5 998244353`	`1`
`10 998244353`	`4`
`10000000000 998244353`	`889033323`

✨ Tóm tắt ý tưởng bài toán

- Do đồ thị là **cây**, nên chỉ có `n-1` **cạnh**.
- Đỉnh 1 có thể có nhiều nhánh, mỗi nhánh là một dây (path) không chia nhánh (bậc ≤ 2).
- **Mỗi dây** phải xuất phát từ một **đỉnh chính** — đỉnh nối trực tiếp với đỉnh 1, và:
  - Các đỉnh trong dây phải giảm dần nhãn và là **bội số của đỉnh chính**.

💡 Bài toán trở thành:

- Chọn tập các **key nodes** từ `2` đến `n`, sao cho các nhãn của chúng là **cùng nhau từng đôi một (pairwise coprime)**.
- Với mỗi tập như vậy, đếm số cách thêm các đỉnh còn lại vào các nhánh của các key node sao cho thỏa mãn điều kiện **giảm dần và là bội số**.
- Tổng số đỉnh luôn là `n`.

✳️ Tổng quát hóa:

- Gọi `f(n)` là số lượng cách chia `n-1` đỉnh còn lại vào các cây con của các key node, với điều kiện ràng buộc số học.
- Phải xử lý bài toán trong `O(log n)` hoặc `O(sqrt n)` do `n` cực lớn (tới 10^{10}).

📚 Đây là bài toán thuộc dạng **đếm cấu trúc số học đặc biệt**, có thể liên quan đến:

- **Hàm Euler (phi)**
- **Đếm tập con pairwise coprime**
- **Đệ quy theo số đỉnh còn lại**

Bạn có muốn mình viết lời giải bằng code Python hoặc C++ cho bài này không?

🤝 Bài G. Ước chung lớn nhất (GCD)

⏰ Giới hạn thời gian: 3 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Cho hai số nguyên `a` và `b`. Ở mỗi lượt, bạn có thể chọn **một trong hai thao tác**:

- Nếu `a > 0`: thay `a ← a - gcd(a, b)`
- Nếu `b > 0`: thay `b ← b - gcd(a, b)`

Grace muốn biết: **cần tối thiểu bao nhiêu lượt** để biến **cả `a` và `b` về 0**.

📥 Input

- Dòng đầu tiên: số lượng test **T** (1 ≤ T ≤ 1000)
- Mỗi test gồm 1 dòng với 2 số nguyên **a, b**
  - $1 \leq a \leq b \leq 10^{18}$
  - Tổng tất cả các `a` trên mọi test không vượt quá 10^4

📤 Output

Với mỗi test, in ra **số lượt tối thiểu** cần thiết để đưa cả `a` và `b` về 0.

📌 Ví dụ

Input	Output
3	
3 4	3
12 20	4
114 514	6

✨ Tóm tắt ý tưởng bài toán

- Ở mỗi bước, ta **trừ đi `gcd(a, b)` từ một trong hai số**.
- Bài toán yêu cầu số bước **ít nhất** để `a = b = 0`.

💡 Nhận xét:

- Vì `gcd(a, b)` chia hết cả `a` và `b`, có thể viết `a = x * g`, `b = y * g` với `g = gcd(a, b)`
  → Bài toán chuyển về: "bao nhiêu phép trừ `g` để `a, b` thành 0"

🚀 Chiến lược giải bài toán

Gọi `f(a, b)` là số bước tối thiểu để về 0.

Quy luật đơn giản hóa:

- Nếu `a = 0`: `f(a, b) = b // gcd(a, b)`
- Nếu `b = 0`: `f(a, b) = a // gcd(a, b)`

- Nếu cả `a, b > 0`: ta nên **trừ đi nhiều lần nhất có thể trên số lớn hơn**

Do `a ≤ 5000`, ta có thể **dựng bảng sẵn**:

```
def min_rounds(a, b):
    res = 0
    while a > 0 and b > 0:
        g = math.gcd(a, b)
        if a > b:
            res += a // g
            a %= g
        else:
            res += b // g
            b %= g
    res += (a + b) // math.gcd(a, b) if a + b > 0 else 0
    return res
```

Bạn muốn mình viết code hoàn chỉnh Python/C++ cho bài này không?

📡 Bài H. Quét chân trời (Horizon Scanning)

⏰ Giới hạn thời gian: 1 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Hana đang phát triển một hệ thống **radar** để giám sát các hòn đảo trong quần đảo mà cô quản lý.

- Có `n` **hòn đảo**, mỗi đảo nằm tại tọa độ `(xᵢ, yᵢ)` trên mặt phẳng.
- Radar được đặt tại **gốc tọa độ (0, 0)** và có thể **xoay theo góc** `θ`, với **góc quét là** `α`.

📌 Khi radar được xoay về góc `θ`, nó quét được vùng góc từ `θ - α/2` đến `θ + α/2` (tính theo **radian**).

🎯 Mục tiêu

Tìm giá trị **nhỏ nhất của góc quét** `α` sao cho **với mọi góc quay** `θ`, radar **luôn bao phủ được ít nhất** `k` **đảo**.

📥 Input

- Dòng đầu: Số lượng test **T** ($1 \le T \le 10^4$)
- Với mỗi test:
  - 1 dòng: `n k` ($1 \le k \le n \le 2 \times 10^5$)
  - `n` dòng tiếp theo: mỗi dòng là tọa độ (x_i, y_i) của 1 đảo ($|x_i|, |y_i| \le 10^9$, không có đảo nào trùng nhau hay nằm tại gốc)

🔒 Tổng `n` của tất cả test không vượt quá 2×10^5

📤 Output

Với mỗi test, in ra một số thực — **góc quét nhỏ nhất** `α` **(đơn vị radian)**, sao cho radar luôn quét được ít nhất `k` đảo tại mọi góc `θ`.

- Sai số tuyệt đối hoặc tương đối không vượt quá `1e-6`.

📌 Ví dụ

Input	Output
`1 1` `0 1`	`6.2831853072`

`4 2`
`-1 1`
`0 1`
`0 2`
`1 1`
	`3.1415926546`

✨ Tóm tắt ý tưởng bài toán

> *"Bài toán hình học trên mặt phẳng với radar **xoay tròn toàn bộ 360°**, cần tìm **góc nhỏ nhất** `α` sao cho **tại mọi vị trí**, vùng quét của radar luôn chứa **ít nhất** `k` **đảo**."*

💡 Chiến lược giải bài

1. **Chuyển các đảo sang góc** so với gốc tọa độ bằng `atan2(yᵢ, xᵢ)` → tạo ra danh sách `angle[]` có `n` phần tử trong [0, 2π).
2. **Sort lại các góc tăng dần**. Để xử lý trường hợp radar quét qua biên `0`/`2π`, ta **nối thêm** mỗi góc `θ` + `2π` → mảng có độ dài `2n`.
3. Với mỗi góc `θᵢ`, ta dùng **tìm nhị phân** để xác định **khoảng nhỏ nhất** `α` sao cho đoạn `[θᵢ, θᵢ + α]` chứa ≥ `k` góc.
4. **Tối ưu hóa**:

- **Binary search** trên giá trị `α` trong khoảng `[0, 2π]`
- Với mỗi `α`, kiểm tra xem **mọi** `θ` đều có thể chứa ít nhất `k` điểm không → dùng **cửa sổ trượt**

Bạn có muốn mình viết luôn lời giải bằng code Python không?

🎒 Bài I. Items

⏰ Giới hạn thời gian: 8 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Ivan có `n` **loại vật phẩm**, mỗi loại có **số lượng vô hạn**.
Vật phẩm loại thứ `i` có **trọng lượng** w_i.

Ivan muốn chọn đúng `n` **món** (có thể trùng loại) sao cho **tổng trọng lượng đúng bằng** `m`.

📥 Input

- Dòng đầu tiên: số lượng test **T** ($1 \le T \le 10^4$)
- Mỗi test gồm:
  - Một dòng chứa hai số nguyên `n` và `m` ($1 \le n \le 10^5$, $0 \le m \le n^2$)
  - Một dòng chứa `n` số nguyên $w_1, w_2, \ldots, w_n$ ($0 \le w_i \le n$)

📌 Tổng tất cả `n` qua các test ≤ 10^5

📤 Output

Với mỗi test, in `"Yes"` nếu có thể chọn đúng `n` món sao cho tổng trọng lượng bằng `m`, ngược lại in `"No"`.

(Chấp nhận mọi kiểu viết hoa: `"YES"`, `"Yes"`, `"yEs"`, v.v.)

📌 Ví dụ

Input	Output
4	
5 25	Yes
0 0 0 0 5	
5 11	No
4 4 4 5 5	
5 0	No
1 2 3 4 5	
5 25	No
0 1 2 3 4	

✨ Tóm tắt yêu cầu

- Có thể chọn **chính xác `n` món**, trùng loại được.

- Tổng trọng lượng đúng **bằng** `m`.
- Mỗi món có trọng lượng nằm trong `w[]`.

💡 Chiến lược giải nhanh

1. Tìm:
   - `min_w`: trọng lượng nhẹ nhất
   - `max_w`: trọng lượng nặng nhất
2. Gọi:
   - `min_total = n × min_w`
   - `max_total = n × max_w`

👉 Khi đó, nếu `m` **nằm ngoài đoạn** `[min_total, max_total]` → **in** `"No"`

✅ Điều kiện cần và đủ:

$$n \times \min(w) \leq m \leq n \times \max(w)$$

🧠 Giải thích

- Vì bạn phải chọn **đúng** `n` **món**, nên tổng trọng lượng tối thiểu là lấy `n` món nhẹ nhất (`min_w`).
- Tổng lớn nhất là `n` món nặng nhất (`max_w`).

- Nếu `m` không nằm trong đoạn đó → không thể đạt được.

Bạn có muốn mình viết code Python tối ưu cho bài này không?

🔄 Bài J. Just another Sorting Problem

⏰ Giới hạn thời gian: 1 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Jessica tổ chức một cuộc thi sắp xếp với 2 người chơi: **Alice** và **Bob**, thi đấu trên **1 hoán vị** dài `n`.

- Người chơi **luân phiên thực hiện thao tác**, người đi trước được xác định trước (Alice hoặc Bob).
- Trò chơi kết thúc ngay lập tức **khi mảng trở thành dãy tăng dần**, và **Alice sẽ chiến thắng**.
- Nếu điều này **không bao giờ xảy ra**, thì **Bob thắng**.

🧩 Quy tắc chơi

- **Alice**: Trong lượt của mình, được **hoán đổi hai phần tử bất kỳ** $i \neq j$.
- **Bob**: Trong lượt của mình, chỉ được **hoán đổi hai phần tử kề nhau** $i, i+1$.

📥 Input

- Số lượng test **T** ($1 \le T \le 10^4$)
- Mỗi test gồm:
  - Dòng 1: `n` và tên người chơi bắt đầu (`"Alice"` hoặc `"Bob"`)
  - Dòng 2: `n` số nguyên tạo thành hoán vị `p` ($1 \le p_i \le n$, không trùng lặp)

📌 Tổng độ dài các hoán vị trong toàn bộ test $\le 10^5$

📤 Output

- Với mỗi test, in `"Alice"` nếu Alice thắng, `"Bob"` nếu Bob thắng.

📌 Ví dụ

Input	Output
2 Alice 2 1	Alice
3 Bob 1 3 2	Bob
10 Bob 1 2 3 4 5 6 7 8 10 9	Bob

✨ Tóm tắt yêu cầu

- Cho một hoán vị `p`, và người chơi đi trước.
- Mỗi lượt, Alice được swap bất kỳ, Bob chỉ được swap kề nhau.
- Nếu mảng trở thành dãy tăng bất kỳ lúc nào, Alice thắng.
- Nếu không bao giờ được, Bob thắng.
- Cả hai chơi **tối ưu**.

💡 Ý tưởng giải bài

🎯 Nhận xét:

- Alice có khả năng "toàn năng" hơn Bob, vì có thể swap bất kỳ.
- **Alice có thể luôn sắp xếp xong trong 1 lượt** nếu đi trước.
  - Ví dụ: chỉ cần làm 1 hoán đổi bất kỳ hoặc nhiều swap để sắp xếp.

📊 Giải pháp:

- Nếu Alice đi trước → cô có thể **thắng ngay lập tức** bằng cách sắp xếp dãy thành dãy tăng → **Alice thắng**.
- Nếu **Bob đi trước**, thì:
  - **Nếu dãy đã gần hoàn chỉnh** (chỉ 1 cặp sai): Bob **có thể phá** nếu swap hợp lý.
  - Nhưng nếu Alice **chỉ cần 1 lượt để sắp xếp**, thì cô vẫn thắng ngay sau lượt đầu của Bob.

✅ Vậy nên cần tính **số nghịch thế** (inversion count).

🧠 Quy luật quyết định:

- Nếu số lượng phần tử sai vị trí là **chẵn** và **Alice đi trước** → Alice thắng
- Nếu số lượng phần tử sai vị trí là **lẻ** và **Bob đi trước** → Alice thắng
- Ngược lại → Bob thắng

Thực tế: Alice thắng nếu:

```
(inversion_count % 2 == 0) == (first_player == "Alice")
```

Hoặc cách đơn giản hơn: **đếm số phần tử sai vị trí**:

```
bad = sum(p[i] != i+1 for i in range(n))
```

→ Nếu `bad == 0`: đã sort xong (không xảy ra vì đề bài đảm bảo chưa sort)

→ Nếu `bad == 2`: chỉ cần swap 1 lần (Alice thắng nếu đi trước)

→ Nếu `bad >= 3`:

- Nếu `bad` lẻ và Alice đi trước → Alice thắng
- Nếu `bad` chẵn và Bob đi trước → Alice thắng
- Ngược lại → Bob thắng

Bạn có muốn mình viết code Python mẫu cho bài này không?

🔐 Bài K. Phục hồi khóa (Key Recovery)

⏰ **Giới hạn thời gian: 1 giây**

🧠 **Giới hạn bộ nhớ: 1024 MB**

⚠️ **Đây là bài tương tác (interactive)**

📝 Đề bài (dịch tiếng Việt)

Kevin không hài lòng với MBTI của mình, nên nhờ Doraemon tạo ra một **máy chuyển đổi MBTI thần kỳ**.

Máy có chức năng biến đổi MBTI của **8 người** cùng lúc, dựa trên MBTI ban đầu và một **khóa bí mật 32-bit** `k`.

Kevin phát hiện rằng MBTI sau khi biến đổi **không thực sự ngẫu nhiên**, mà tuân theo một quy tắc cụ thể gồm 3 khối cơ bản:

⚙️ 3 khối xử lý MBTI

1. `xor`**:**

- Đảo bit MBTI dựa trên khóa `k`
- Với mỗi `i = 0..7`:

$$\text{out}_i = \text{in}_i \oplus (\text{k}[4\text{i}{+}3,\ 4\text{i}])$$

(tức là lấy 4 bit liên tiếp từ `k`, dùng để XOR với in_i)

2. `perm`**:**

- Hoán vị cố định:

```
p = [7, 0, 5, 6, 11, 15, 4, 3, 1, 13, 12, 14, 8, 9, 10, 2]
```

- $\text{out}_i = p[\text{in}_i]$

3. `mix`:

- Pha trộn giữa 2 giá trị liên tiếp:
  - Với $in_i < 8$: $t_i = in_i \ll 1$
  - Với $in_i \geq 8$: $t_i = ((in_i \oplus 8) \ll 1) \oplus 3$
- $out_i = in_i \gg 1 \oplus t_{(i \gg 1) \oplus 1}$
  ($\ll$ và $\gg$ là dịch trái/phải vòng lặp 3 bit)

🔁 Chuỗi biến đổi

Mỗi nhóm 8 MBTI ban đầu sẽ trải qua **19 bước biến đổi** theo thứ tự:

```
(xor, perm, mix) × 6 → xor cuối cùng
```

→ Tổng cộng 19 khối xử lý

🎯 Mục tiêu của bài toán

Bạn **không biết giá trị của `k`**, nhưng có thể đưa input vào máy và quan sát output để **đoán ra khóa `k`**.

- Có thể chạy **tối đa 4096 lượt** với input 8 ký tự hexa (MBTI).
- Sau đó, cần in ra `! k` với `k` là khóa bạn đoán được (số nguyên **không âm < 2^{32}**).
- Nếu quá 4096 lần, hoặc sai format, chương trình bị chấm **"Wrong Answer"**.

📩 Input / Output

- Đưa input như sau:

```
? 04f37255
```

(gồm đúng 8 ký tự trong `{0..9, a..f}`, không cách)

- Nhận lại output từ hệ thống:

```
acf7e10b
```

- Sau cùng, nếu đoán được khóa `k`, in:

```
! 998244353
```

📌 Lưu ý quan trọng

- Input/output là **hex**, mỗi ký tự tương ứng 1 MBTI.
- Bạn cần **giải ngược toàn bộ quá trình 19 bước xử lý**, từ đầu ra suy ngược về khóa `k`.
- Máy luôn giữ `k` cố định, không thay đổi theo input bạn đưa vào.
- Có thể **test local** với file `interact.hpp` và ví dụ `local_test_example.cpp`.

✨ Tóm tắt

- Mỗi input là 8 MBTI (biểu diễn bằng hex: 0 → INFP, ..., f → ESTJ).
- Qua 19 bước (xor, perm, mix...) sẽ ra 8 MBTI mới.
- Dựa vào input → output, bạn cần đoán chính xác khóa `k` (32-bit).
- Có tối đa 4096 lần chạy máy → đoán `k` và in ra `! k`.

Bạn có muốn mình viết đoạn code mẫu Python để gửi input/nhận output không?

💥 Bài L. Cơ hội cuối cùng: Luồng tuyệt vọng (Last Chance: Threads of Despair)

⏰ Giới hạn thời gian: 2 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Fried-chicken đang chơi game Hearthstone. Trong một ván đấu căng thẳng, anh ấy chỉ còn lại **1 máu**, và để sống sót, anh **phải tiêu diệt toàn bộ lính** của đối thủ **Stewed-chicken**.

🎮 Luật chơi trong ván đấu

- Mỗi bên có một số **quái vật (minions)**, mỗi con có chỉ số **máu (Health)**.
- Đến lượt Fried-chicken:
  1. **Anh ấy thi triển phép “Threads of Despair”**:

- Bất cứ quái vật nào **chết** sẽ **phát nổ**, làm giảm máu của **tất cả các quái vật còn lại** đi **1 đơn vị**.
- Nếu vụ nổ tiếp tục khiến quái vật khác chết, thì **sẽ tiếp tục phát nổ dây chuyền**.
- 👉 Fried-chicken **không được tấn công** trong lúc nổ dây chuyền diễn ra.

2. Sau khi kết thúc chuỗi nổ, **mỗi lính của Fried-chicken được tấn công một lần**, chọn bất kỳ lính nào của Stewed-chicken:
   - Khi một lính tấn công lính khác: **cả hai mất 1 máu**
   - Nếu máu ≤ 0, lính **chết ngay**

🎯 Nhiệm vụ

Hãy xác định: **Fried-chicken có thể tiêu diệt hết tất cả lính của Stewed-chicken không**, **bằng cách chọn thứ tự tấn công phù hợp**?

📩 Input

- Dòng đầu: số lượng test **T** ($1 \le T \le 5 \times 10^5$)
- Với mỗi test:
  - Dòng 1: `n m` — số lính của Fried-chicken và Stewed-chicken ($1 \le n, m \le 5 \times 10^5$)
  - Dòng 2: `n` số nguyên — máu các lính của Fried-chicken
  - Dòng 3: `m` số nguyên — máu các lính của Stewed-chicken

🔒 Tổng tất cả `n + m` trên mọi test ≤ 10^6

📤 Output

- Với mỗi test, in `Yes` nếu Fried-chicken có thể tiêu diệt hết lính đối thủ.
- Ngược lại, in `No`.

(chấp nhận mọi kiểu viết hoa: `YES`, `yes`, `Yes`, v.v.)

📌 Ví dụ

Input	Output
3	
3 2	
1 1 4	
2 6	Yes
3 2	
1 1 4	
2 7	No
2 1	
100 100	
2	Yes

✨ Tóm tắt bản chất bài toán

- Bạn được quyền chọn **mỗi lính tấn công một lần**, tổng cộng **n lượt đánh**.
- Mỗi cú đánh gây 1 sát thương lên cả người tấn công và người bị tấn công.
- **Bất kỳ lính nào chết** gây nổ, **gây -1 máu cho tất cả các lính còn lại** → có thể kích hoạt **hiệu ứng dây chuyền**.

📌 **Mục tiêu**: Tìm **thứ tự tấn công** sao cho sau khi nổ & tấn công, **tất cả lính của Stewed-chicken đều chết**.

🧠 Ý tưởng tổng quát

- Sắp xếp máu lính hai bên tăng dần.
- Dồn tấn công từ lính khỏe nhất của Fried-chicken vào lính yếu nhất của Stewed-chicken.
- Theo dõi tổng số máu cần tiêu diệt:
  Tổng máu lính địch ≤ số lượt đánh + tổng hiệu ứng nổ

Bạn có muốn mình viết code mẫu Python cho bài này không?

🧮 Bài M. Matrix Construction (Xây dựng ma trận)

⏰ Giới hạn thời gian: 1 giây

🧠 Giới hạn bộ nhớ: 1024 MB

📝 Đề bài (dịch tiếng Việt)

Mary rất thích xây ma trận!

Cô muốn điền một **hoán vị** từ `1` đến `n × m` vào một ma trận kích thước `n × m` sao cho:

> ***"Tổng của bất kỳ cặp phần tử liền kề nào cũng là duy nhất."***

📌 Cụ thể:

Với hai cặp tọa độ `(x₁, y₁)-(x₂, y₂)` và `(x₃, y₃)-(x₄, y₄)`, nếu:

- Cả hai cặp là **liền kề** (nghĩa là khoảng cách Manhattan = 1)
- Hai cặp không giống hệt nhau (không trùng cả điểm đầu và cuối)

Thì:

$$A_{x_1,y_1} + A_{x_2,y_2} \neq A_{x_3,y_3} + A_{x_4,y_4}$$

📩 Input

- Dòng đầu: số lượng test `T` ($1 \le T \le 10^4$)
- Mỗi test gồm 1 dòng: `n m` ($1 \le n, m \le 1000$) — kích thước ma trận

🔒 Tổng `n × m` trên mọi test ≤ 10^6

📤 Output

Với mỗi test:

- Dòng đầu: `"Yes"` nếu tồn tại ma trận hợp lệ, `"No"` nếu không
- Nếu `"Yes"`, in ra ma trận `n × m` chứa hoán vị từ `1 → n×m`

📌 Ví dụ

Input	Output
2	
1 1	yEs
	1
2 3	YES
	1 3 2
	6 5 4

✅ Ma trận thứ 2 hợp lệ vì không có cặp liền kề nào có tổng trùng nhau.

✨ Tóm tắt yêu cầu

- Điền các số từ `1` đến `n×m` sao cho **tổng 2 ô liền kề bất kỳ không trùng nhau**
- Có thể chọn cách điền tùy ý, chỉ cần thỏa điều kiện

💡 Ý tưởng giải nhanh

Nhận xét:

- Dễ gây xung đột tổng nếu điền tuần tự theo dòng (row-wise)
- Tuy nhiên, **điền theo zic-zac hoặc cách điền "chẵn-lẻ tách hàng/cột"** sẽ đảm bảo tổng các cặp liền kề khác nhau

✅ Chiến lược an toàn:

- Khi `min(n, m) == 1`, luôn trả lời `"Yes"` → chỉ 1 dòng/cột thì không có 2 cặp liền kề độc lập → **không vi phạm**
- Với `n ≥ 2, m ≥ 2`, bạn có thể:
  - Chia số thành 2 nhóm:
    - Nhóm 1: điền các số lẻ trước
    - Nhóm 2: điền số chẵn sau
  - Lý do: tổng hai số lẻ/chẵn luôn **chẵn**, còn tổng 1 lẻ + 1 chẵn = **lẻ** → dễ tránh trùng

---

Bạn có muốn mình viết code Python mẫu để sinh ma trận hợp lệ này không?