

universal 3 stage 23 : Hong Kong

Created: 5/3/2025 00:00
Updated: 5/3/2025 00:04
Exported: 5/3/2025 00:03

Bài A. Tính Đối Xứng Tổng Quát

 **Giới hạn thời gian:** 2 giây


 **Giới hạn bộ nhớ:** 256 MB

Đề bài (dịch tiếng Việt)

Cho một dãy số nguyên $S = [s_1, s_2, \dots, s_m]$ gồm m phần tử. Dãy này được gọi là **đối xứng k** nếu với mọi chỉ số i từ 1 đến m , ta có:

$$|s_i - s_{m-i+1}| \leq k$$

Bạn được cho một dãy số nguyên $A = [a_1, a_2, \dots, a_n]$ gồm n phần tử. Nhiệm vụ của bạn là:

 Với mỗi vị trí làm **tâm** trong dãy A , hãy tìm **độ dài lớn nhất** của dãy con liên tiếp đối xứng k có tâm tại vị trí đó.

Tâm có thể:

- Là một chỉ số nguyên i (ví dụ: 1, 2, 3...), tức là dãy đối xứng có độ dài lẻ.
- Là một chỉ số nửa nguyên $i + 0.5$, tức là dãy đối xứng có độ dài chẵn.

 Một đoạn có chỉ số từ l đến r được xem là có **tâm tại** $\frac{l+r}{2}$.

Input

- Dòng đầu tiên: hai số nguyên n và k
($2 \leq n \leq 2 \times 10^5, 0 \leq k \leq 10^3$)
- Dòng thứ hai: n số nguyên a_1, a_2, \dots, a_n
($1 \leq a_i \leq 10^3$)

Output

- Dòng đầu tiên: n số nguyên. Số thứ i là độ dài lớn nhất của dãy con liên tiếp đối xứng k , có **tâm tại vị trí i** .
- Dòng thứ hai: $n - 1$ số nguyên. Số thứ i là độ dài lớn nhất của dãy con liên tiếp đối xứng k , có **tâm tại vị trí $i + 0.5$** .

👉 Nếu **không tìm được** dãy con thỏa mãn, in ra `0`.

Ví dụ

Input

```
5 0
1 2 1 2 1
```

Output

```
1 3 5 3 1
0 0 0 0
```

Input

```
5 1
```

```
1 2 1 3 1
```

Output

```
1 3 5 3 1
2 2 0 0
```

✨ Tóm tắt

- Kiểm tra các đoạn con đối xứng quanh từng tâm i và $i + 0.5$.
- Một dãy con là đối xứng k nếu phần tử đối xứng hai bên tâm có hiệu tuyệt đối nhỏ hơn hoặc bằng k .
- Cần tìm **độ dài lớn nhất** cho mỗi tâm có thể.

Bạn muốn mình code bài này luôn không?

📖 Bài B. Tiêu Diệt Kẻ Địch

🕒 **Giới hạn thời gian:** 3 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

🔍 Đề bài (dịch tiếng Việt)

Bạn phải tiêu diệt **n kẻ địch**. Mỗi kẻ địch thứ i có:

- Máu: a_i

- Giáp: b_i

Bạn có thể **tấn công nhiều lần**. Mỗi đòn tấn công bạn chọn gây sát thương x lên **tất cả kẻ địch**, trong đó $1 \leq x \leq k$.

💰 Mỗi đòn đánh sát thương x sẽ tốn chi phí là c_x .

Cách kẻ địch nhận sát thương:

Khi một kẻ địch nhận sát thương y :

- Nếu **giáp còn**, thì giáp sẽ bị giảm y điểm (không ảnh hưởng đến máu).
 - Nếu **giáp đã hết**, máu mới bắt đầu bị giảm.
 - Kẻ địch được xem là **bị tiêu diệt** nếu máu ≤ 0 ở bất kỳ thời điểm nào.
-

Nhiệm vụ của bạn:

1. **Tìm chi phí nhỏ nhất** để tiêu diệt toàn bộ kẻ địch.
2. **Đếm số chiến lược tấn công khác nhau** (dãy các đòn đánh) có chi phí nhỏ nhất đó (modulo 998244353).

Một **chiến lược tấn công** là một dãy các đòn đánh x_1, x_2, \dots, x_m với $1 \leq x_i \leq k$.

Hai chiến lược được xem là **khác nhau** nếu dãy đòn đánh khác nhau.

Input

- Dòng đầu tiên: số lượng test T ($1 \leq T \leq 1000$)

- Mỗi test gồm:
 - Dòng 1: hai số nguyên n, m — số kẻ địch và giới hạn tối đa cho a_i, b_i
 - Dòng 2: n số nguyên a_1, a_2, \dots, a_n — máu từng kẻ địch
 - Dòng 3: n số nguyên b_1, b_2, \dots, b_n — giáp từng kẻ địch
 - Dòng 4: số nguyên k — số sát thương tối đa có thể dùng trong một đòn
 - Dòng 5: k số nguyên c_1, c_2, \dots, c_k — chi phí khi đánh với sát thương i

 Đảm bảo:

- Tổng n của tất cả test $\leq 5 \cdot 10^5$
- Tổng m của tất cả test $\leq 10^4$

Output

Với mỗi test, in ra một dòng gồm hai số nguyên:

- **Chi phí nhỏ nhất** để tiêu diệt toàn bộ kẻ địch
- **Số lượng chiến lược khác nhau** đạt được chi phí này, **modulo** 998244353

Tóm tắt

- Mỗi đòn đánh gây sát thương đồng thời lên **tất cả kẻ địch**.
- Sát thương tiêu hao **giáp trước, không xuyên qua** nếu giáp chưa hết.
- Cần tìm cách kết hợp các đòn đánh $x \in [1..k]$ để làm máu mọi kẻ địch ≤ 0 , với tổng chi phí nhỏ nhất.

- Đồng thời đếm có bao nhiêu **cách kết hợp khác nhau** (theo thứ tự) để đạt được chi phí đó.
-

Bạn muốn mình giải thích hướng thuật toán hay bắt đầu viết code mẫu cho bài này?

Bài C. Câu chuyện của Hoàng đế Bie

 **Giới hạn thời gian:** 1 giây

 **Giới hạn bộ nhớ:** 1024 MB

Đề bài (dịch tiếng Việt)

Hoàng đế **Bie** muốn tạo ra một mảng số nguyên A gồm n phần tử.

Ông ta bắt đầu bằng cách chọn **một vị trí ban đầu** p (với $1 \leq p \leq n$), và gán một số nguyên dương tùy ý mà ông thích vào phần tử A_p .

Sau đó, ông khởi tạo hai biến: $l = r = p$ (đánh dấu phạm vi hiện tại của mảng đã được xây dựng).

Trong $n - 1$ **bước tiếp theo**, ông thực hiện các thao tác sau:

Hai thao tác có thể thực hiện:

1. **Mở rộng sang trái (Left Expand):**

- Chỉ thực hiện được nếu $l > 1$.
- Chọn một số k sao cho $0 \leq k < A_r$

- Gán giá trị $A_r - k$ cho phần tử bên trái: A_{l-1}
- Giảm chỉ số trái: $l = l - 1$

2. Mở rộng sang phải (Right Expand):

- Chỉ thực hiện được nếu $r < n$.
- Chọn một số k sao cho $0 \leq k < A_l$
- Gán giá trị $A_l - k$ cho phần tử bên phải: A_{r+1}
- Tăng chỉ số phải: $r = r + 1$

⚠ Lưu ý:

- Một phần tử mới được tạo ra luôn **nhỏ hơn hoặc bằng** phần tử đã biết ở đầu bên kia.
- Khi mở rộng sang trái thì dùng giá trị hiện tại ở bên phải; mở rộng sang phải thì dùng giá trị hiện tại ở bên trái.

🕒 Sau nhiều năm, Hoàng đế Bie **chỉ nhớ mảng A cuối cùng, không nhớ vị trí khởi đầu p** nữa.

👉 Hãy giúp ông xác định **những chỉ số nào có thể là vị trí ban đầu p**.



Input

- Dòng đầu tiên: số lượng test T ($1 \leq T \leq 10^5$)



- Với mỗi test:
 - Dòng 1: số nguyên n — độ dài mảng
 - Dòng 2: n số nguyên A_1, A_2, \dots, A_n — mảng đã được xây dựng

Đảm bảo:

- Tổng tất cả n của các test $\leq 5 \cdot 10^5$
- Luôn có ít nhất một vị trí khởi đầu hợp lệ

Output

- Với mỗi test, in ra **các chỉ số p hợp lệ** (từ nhỏ đến lớn), cách nhau bởi dấu cách.

Ví dụ

Input

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

Output

```
1
2
1 2 3
```

✨ Tóm tắt

- Có hai thao tác mở rộng dần sang trái/phải, mỗi lần mở rộng thêm một phần tử mới.
- Mỗi phần tử mới luôn \leq phần tử đầu bên kia (theo quy tắc $A_{mới} = A_{đầubênkia} - k$)
- Tìm tất cả các chỉ số p sao cho nếu bắt đầu từ đó, ta có thể **khôi phục đúng mảng A** theo quy tắc mở rộng trên.

Bạn muốn mình viết code kiểm tra vị trí p hợp lệ không?

📖 Bài D. Bậc Thầy Song Toàn VI

🕒 **Giới hạn thời gian:** 2.5 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

🧠 Đề bài (dịch tiếng Việt)

Viktor đang tấn công Piltover để chiếm quyền kiểm soát các cổng Hexgate bằng công nghệ Hextech.

Thành phố Piltover có **n ngã tư**, được nối với nhau bằng **$n - 1$ con đường hai chiều**, sao cho **bất kỳ hai ngã tư nào cũng có thể đi đến nhau**.

📌 Tức là hệ thống này tạo thành một **cây**.

🛡️ Cơ chế chiến đấu

- Viktor sử dụng **tia Hextech** gây **sát thương a** trên **đường đi ngắn nhất giữa hai ngã tư**.

- Jayce dùng **lá chắn Hextech** để bảo vệ các ngã tư, mỗi lá chắn có **máu b** .
-

💥 Cơ chế tấn công:

- Khi **một lá chắn bị tấn công**, nó có thể **kích hoạt quá tải (overload)** để **giảm sát thương còn một nửa**.
 - Ví dụ: nếu nhận 3 sát thương, overload \rightarrow chỉ còn 1.5 sát thương.
 - **! Không được kích hoạt overload hai lần liên tiếp** cho cùng một lá chắn.
-

📋 Các sự kiện (tổng cộng q sự kiện)

Có hai loại:

1. Tấn công: ` $A \times y \ z`$

👉 Gây sát thương z đến tất cả ngã tư nằm **trên đường ngắn nhất từ x đến y** .

2. Truy vết: ` $D \times h`$

👉 Tại ngã tư x , có một lá chắn **chưa bị phá**, Jayce nhớ rằng **máu ban đầu là h** .

\rightarrow Nhiệm vụ của bạn: **Tính số lần tấn công lớn nhất** mà lá chắn có thể đã chịu được (kết hợp overload xen kẽ để tối ưu), **và vẫn sống sót đến hiện tại**.

📝 Input

- Dòng đầu: hai số nguyên n, q ($1 \leq n \leq 5 \cdot 10^5$, $1 \leq q \leq 3 \cdot 10^5$)
- $n - 1$ dòng tiếp theo: mỗi dòng chứa 2 số u, v — hai ngã tư nối với nhau
- q dòng tiếp theo: mỗi dòng mô tả sự kiện dạng:

- $\texttt{A } x \ y \ z$ — Tấn công từ x đến y với sát thương z
- $\texttt{D } x \ h$ — Truy vết lá chắn ở nút x , máu ban đầu h

Giá trị ràng buộc thêm:

- $1 \leq z \leq 370$
- $1 \leq a_i \leq 10^9$

Output

- Với mỗi sự kiện **truy vết** $\texttt{D } x \ h$, in ra một dòng — **số lần tấn công tối đa** mà lá chắn tại nút x có thể đã hứng chịu, để máu vẫn > 0 .

Ví dụ

Input

```
5 7
1 2
1 3
2 4
2 5
A 1 4 2
A 3 5 2
D 1 4
D 2 3
D 2 1
A 5 5 10
D 5 100
```

Output

2
1
0
2

✨ Tóm tắt

- Ghi lại **số lần và giá trị tấn công** cho mỗi nút trên cây.
- Với mỗi nút x , ta cần tính xem một máu ban đầu h có thể chịu được bao nhiêu lần tấn công (với các sát thương cụ thể đã xảy ra tại x) mà:
 - Sử dụng overload **một cách tối ưu** (không hai lần liên tiếp).
- Trả về số lần tấn công tối đa **từ lịch sử tại nút đó**, để máu còn > 0 .

Bạn có muốn mình phân tích hướng giải cụ thể và đưa ra đoạn code C++/Python cho bài này không?

📖 Bài E. Bao Lồi Lõm (Concave Hull)

🕒 **Giới hạn thời gian:** 3 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

🎲 Đề bài (dịch tiếng Việt)

Bạn được cho **n điểm** trên mặt phẳng tọa độ. Mỗi điểm có tọa độ (x_i, y_i) .

Một **Concave Hull** là **một đa giác đơn** (tức là không tự cắt nhau), thoả mãn các điều kiện:

1. Tập đỉnh của đa giác là **một tập con không rỗng** của n điểm đã cho.
 2. **Tất cả n điểm** đều nằm **bên trong hoặc trên biên của đa giác**.
 3. **Chính xác một góc trong đa giác là góc lõm** (lớn hơn π); tất cả các góc còn lại đều là **góc lồi** (nhỏ hơn π).
-

Nhiệm vụ của bạn

Tính:

- **Gấp đôi tổng diện tích của tất cả các Concave Hull** (thoả mãn điều kiện trên)
 - Kết quả in ra **modulo** $10^9 + 7$
-

Input

- Dòng đầu: số nguyên n ($3 \leq n \leq 2000$) — số điểm.
- n dòng tiếp theo, mỗi dòng chứa hai số nguyên x_i, y_i ($0 \leq x_i, y_i \leq 10^9$) — tọa độ điểm thứ i .

 **Đảm bảo:**

- **Không có hai điểm trùng nhau**
 - **Không có ba điểm nào thẳng hàng**
-

Output

- Một số nguyên duy nhất — $2 \times \text{tổng diện tích của các Concave Hull}$, mod $10^9 + 7$

Ví dụ

Input

```
4
0 0
2 0
1 2
1 1
```

Output

```
8
```

Giải thích ví dụ

- Có **3 Concave Hulls** thoả mãn điều kiện.
- Diện tích của 3 hình lần lượt là: 1.5, 1.5, 1 → Tổng = 4
- Nhân 2 lên: $4 \times 2 = 8$

Tóm tắt

- Tìm tất cả các **đa giác đơn** tạo từ tập con các điểm đã cho sao cho:
 - Chứa hết tất cả điểm
 - Có đúng 1 góc lõm
 - Tính diện tích từng hình, cộng lại, rồi nhân đôi.
-

Bài này có yêu cầu **tối ưu thuật toán hình học** rất cao — bạn có muốn mình trình bày chiến lược giải chi tiết và code mẫu không?

Bài F. Trò Chơi Tiền Bạc 2 (Money Game 2)

 **Giới hạn thời gian:** 4 giây

 **Giới hạn bộ nhớ:** 1024 MB

Đề bài (dịch tiếng Việt)

Putata và Budada tổ chức một trò chơi gồm **n người chơi** ngồi thành **vòng tròn**.

- Người chơi được đánh số từ 0 đến $n - 1$.
 - Mỗi người có **số tiền gửi ban đầu là a_i** (số nguyên).
 - Mỗi người có **hai người hàng xóm** là:
 - Bên trái: $(i - 1) \bmod n$
 - Bên phải: $(i + 1) \bmod n$
-

Luật chơi (mỗi vòng)

- Chọn một người chơi **x chưa từng được chọn** trước đó.
 - Người này sẽ **cho đi một nửa số tiền của mình** (*làm tròn xuống*) cho **một người hàng xóm (bên trái hoặc bên phải)*.
 - **Mỗi người chỉ được chia tiền một lần duy nhất.**
-

Yêu cầu

Với mỗi người chơi i , tính giá trị $f(i)$ — **số tiền tối đa mà người đó có thể nhận được**, sau một số lượng vòng chơi nào đó (*hoặc không có vòng nào*).

 Các giá trị $f(i)$ là độc lập nhau, tức là:

- Với mỗi i , bạn cần xét cách tối ưu để **người chơi số i nhận nhiều tiền nhất**, không cần đồng thời tối ưu cho người khác.
-

Input

- Dòng đầu: số nguyên T ($1 \leq T \leq 5 \cdot 10^5$) — số test case.
- Mỗi test gồm:
 - Dòng 1: số nguyên n — số người chơi.
 - Dòng 2: n số nguyên a_0, a_1, \dots, a_{n-1} — số tiền ban đầu của từng người.

 **Tổng tất cả n trong các test không vượt quá $5 \cdot 10^5$.**

Output

- Với mỗi test case, in một dòng gồm n số: $f(0), f(1), \dots, f(n-1)$

Ví dụ

Input

```
3
5
2 1 4 3 5
5
2 1 3 1 2
1
1000000000
```

Output

```
6 5 7 8 8
4 4 5 4 4
1000000000
```

✨ Tóm tắt

- Mỗi người chơi chỉ được **chia tiền đúng 1 lần** cho 1 hàng xóm.
- Với mỗi người chơi i , muốn tìm cách bố trí sao cho **tổng tiền của i đạt giá trị lớn nhất có thể**:
 - Giữ lại tiền gốc của mình.
 - Nhận tối đa 1 lần từ **trái** (nếu người trái chưa chia cho ai khác).
 - Nhận tối đa 1 lần từ **phải**.

👉 Với người chơi i :

$$f(i) = a_i + \max \left(\left\lfloor \frac{a_{(i-1+n)\%n}}{2} \right\rfloor, \left\lfloor \frac{a_{(i+1)\%n}}{2} \right\rfloor \right)$$

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

📖 Bài G. Yelkrab

🕒 **Giới hạn thời gian:** 2 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

🧠 Đề bài (dịch tiếng Việt)

Một kỳ thi lập trình đặc biệt dành cho... **heo** (Piggy) sắp diễn ra!

Trong kỳ thi này có **n track thi đấu**, track thứ k yêu cầu **đúng k thí sinh mỗi đội**.

Bạn có **n chú heo**, mỗi chú heo có một cái tên s_i (chữ thường, không rỗng).

Mỗi **đội thi** phải gồm đúng k chú heo có tên từ danh sách ban đầu.

🌟 Cách tính điểm:

- **Điểm của một đội:** độ dài **prefix chung dài nhất** giữa các tên heo trong đội.
- **Điểm của trường:** tổng điểm của các đội.

🚫 Mỗi heo chỉ **tham gia tối đa 1 đội**.

Yêu cầu

Với mỗi test, bạn cần tính:

$$\text{Với mỗi } 1 \leq i \leq n, \text{ tính } \bigoplus_{j=1}^i (f(i, j) \times j)$$

Trong đó:

- $f(i, j)$: điểm cao nhất trường có thể đạt được nếu chỉ dùng **i heo đầu tiên**, lập thành các **đội size đúng j**.
 - \oplus : phép XOR giữa các giá trị $f(i, j) \times j$
-

Input

- Dòng đầu: số nguyên T — số test case.
- Mỗi test gồm:
 - Một dòng: số nguyên n
 - n dòng tiếp theo: mỗi dòng là chuỗi tên s_i

 Ràng buộc:

- Tổng n của tất cả test $\leq 5 \cdot 10^5$
 - Tổng độ dài tất cả tên $\leq 10^6$
-

Output

- Với mỗi test, in ra n số nguyên, cách nhau bởi khoảng trắng

- Số thứ i : là kết quả XOR từ $j = 1$ đến $j = i$ của $f(i, j) \times j$

Ví dụ

Input

```
2
5
aa
ab
ab
ac
d
1
aaaaa
```

Output

```
2 6 1 9 8
5
```

Tóm tắt

- Mỗi track yêu cầu đội có size = số track.
- Cần chia các heo thành các nhóm đúng size và không trùng nhau.
- Mỗi nhóm có điểm là prefix chung dài nhất của tên các heo trong nhóm.
- Với i heo đầu tiên, tìm tất cả chia nhóm size j tối đa và tính tổng điểm tốt nhất có thể.

- XOR tất cả $f(i, j) \times j$ với $1 \leq j \leq i$
-

Bạn có muốn mình phân tích hướng giải thuật chi tiết và viết code C++ hoặc Python mẫu cho bài này không?

Bài H. Mạt chược (Mah-jong)

 **Giới hạn thời gian:** 3 giây

 **Giới hạn bộ nhớ:** 1024 MB

Đề bài (dịch tiếng Việt)

Gọi một **multiset** (tập có thể lặp phần tử) s gồm các số nguyên dương là:

- **Pong** nếu $s = \{x, x, x\}$ với một số nguyên dương x
- **Chow** nếu $s = \{x, x + 1, x + 2\}$ với một số nguyên dương x

Một multiset được gọi là **Mahjong** nếu nó có thể được **chia** thành **vài tập Pong và vài tập Chow** (có thể không có tập nào).

Nhiệm vụ

Bạn được cho một mảng số nguyên a_1, a_2, \dots, a_n với $1 \leq a_i \leq 8$.

Hãy đếm số lượng **đoạn con liên tiếp** $[l, r]$ (với $1 \leq l \leq r \leq n$) sao cho tập các phần tử từ a_l đến a_r tạo thành một **Mahjong**.



Input

- Dòng đầu: số nguyên T — số lượng test.
- Với mỗi test:
 - Dòng đầu: số nguyên n
 - Dòng thứ hai: n số nguyên a_1, \dots, a_n , với $1 \leq a_i \leq 8$

✓ Ràng buộc:

- Tổng tất cả n trong các test $\leq 10^5$



Output

- Với mỗi test, in ra một dòng duy nhất — số đoạn con tạo thành Mahjong.



Ví dụ

Input

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



Output

```
2
5
1
3
2
```

✨ Tóm tắt

- Một đoạn là **Mahjong** nếu tập con các phần tử từ a_l đến a_r **có thể chia** thành nhiều **Pong** hoặc **Chow**.
- Mỗi tập con có độ dài chia hết cho 3 (vì mỗi tổ hợp là 3 phần tử).
- Vì $a_i \in [1..8]$, ta có thể dùng **quy hoạch động (DP)** trên tần số xuất hiện từ $1 \rightarrow 8$ để thử tất cả các cách chia.

Bạn muốn mình giải thích thuật toán chi tiết và viết code Python/C++ cho bài này chứ?

📖 Bài I. Ma Meilleure Ennemie

🕒 **Giới hạn thời gian:** 1.5 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

🧠 Đề bài (dịch tiếng Việt)

Ekko đang bị phân thân vì thiết bị Z-Drive khiến có tới **n Ekko** trong vũ trụ hiện tại. Heimerdinger muốn gửi họ về đúng **m vũ trụ** có sẵn, theo một thuật toán phân chia đặc biệt.

Quy trình phân chia vũ trụ

1. Ban đầu tất cả **n Ekko** trong **1 nhóm**.
 2. Heimerdinger chọn một số nguyên **x** từ 1 đến m , gọi là **số tốt** nếu:
 - Với mọi Ekko i trong nhóm, thì Ekko $(i + x) \bmod n$ cũng phải ở trong nhóm.
 3. Với số tốt x , chia nhóm thành các **nhóm tạm thời** sao cho:
 - Nếu Ekko i trong nhóm tạm, thì $(i + x) \bmod n$ cũng ở trong đó.
 - Cách chia phải là cách cho **số nhóm tạm thời lớn nhất có thể**.
 4. Nếu nhóm chỉ chia được ra **1 nhóm duy nhất**, thì:
 - Gửi tất cả Ekko trong nhóm về **vũ trụ x** .
 5. Ngược lại, tiếp tục lặp lại bước trên **với từng nhóm tạm thời mới**.
-

Yêu cầu

Tính xem có **bao nhiêu cách khác nhau** để phân các Ekko về các vũ trụ, **theo quy trình trên, modulo 998244353**.

Hai kết quả được coi là **khác nhau** nếu **tồn tại ít nhất một Ekko** được gửi về **hai vũ trụ khác nhau** trong hai cách.



Input

- Một dòng chứa 2 số nguyên n, m với $1 \leq n, m \leq 10^{18}$



Output

- Một số nguyên: **số lượng cách phân chia khác nhau**, mod 998244353



Ví dụ

Input

4 4

Output

6

✨ Tóm tắt thuật toán

Đây là một bài toán **phân tích nhóm và quay vòng** (modulo) với bài toán phân nhóm quay đều theo modulo n .

Ý tưởng chính:

- Mỗi số tốt x phải chia n , tức là $n \bmod x = 0$
- Với mỗi x chia hết n , số nhóm tạm thời sẽ là $\gcd(n, x)$



- Mỗi cách chia sẽ tạo ra cây phân rã, đệ quy theo nhóm con → tổ hợp chia và số gốc sinh.

📌 Cốt lõi là:

$$\text{Số kết quả khác nhau} = \sum_{d|n} \phi(d) \cdot m^{n/d} \mod 998244353$$

Trong đó:

- d là ước của n
- $\phi(d)$ là **Euler's totient function**
- $m^{n/d}$: số cách gán vũ trụ cho nhóm gốc

Bạn muốn mình cài thuật toán này bằng Python và giải ví dụ mẫu chứ?

📖 Bài J. Reconstruction - Tái dựng cây

🕒 **Giới hạn thời gian:** 3 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

🧠 Đề bài (dịch tiếng Việt)

Bạn được cho:

- Một cây ban đầu **T1** với **n đỉnh**.
- Một **mảng** `a[1..n]`, ban đầu bằng 0.
- Bạn sẽ thực hiện **n lần thao tác** để xây dựng cây mới **T2**, như sau:

Quy trình xây dựng cây T2 từ T1

1. **Chọn một đỉnh x chưa bị xóa khỏi T1.**
2. **Gán $a[x]$ là cha của x trong cây mới T2** (nếu $a[x] = \emptyset$ thì x là gốc).
3. **Với mọi đỉnh y còn kết nối với x trong T1**, ta gán $a[y] = x$. Tức là sau khi x bị xóa, tất cả hàng xóm của x sẽ chọn x làm cha.
4. **Xóa x và các cạnh nối với x ra khỏi T1.**

Lặp lại **n lần**, mỗi lần xóa một đỉnh \rightarrow bạn có được cây T2 thông qua mảng cha a .

Yêu cầu

Bạn được cho một cây khác **T** (gồm n đỉnh), **không biết gốc**, và bạn cần xác định:

- Với mỗi đỉnh $u \in [1..n]$, nếu gốc cây **T** là u , thì cây T có thể được tạo từ T1 thông qua quy trình trên không?

In ra một chuỗi nhị phân dài n :

- Ký tự thứ i là **'1'** nếu i là gốc hợp lệ \rightarrow cây T có thể tạo được từ T1.
 - Ngược lại là **'0'**.
-

Input

- Dòng 1: n — số đỉnh của cây
- $n - 1$ dòng tiếp theo: cạnh của cây **T1**
- $n - 1$ dòng sau đó: cạnh của cây **T**

✅ **Bảo đảm:** cả hai tập cạnh tạo thành **cây hợp lệ**.

Output

- Một dòng gồm chuỗi nhị phân dài n , ký tự thứ i là `'1'` nếu đỉnh i có thể là gốc của cây T hợp lệ tạo từ T1.
-

Ví dụ

Input

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

Output

```
001
```

Input

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

```
5 2
3 6
```

Output

```
010110
```

✨ Tóm tắt ý tưởng giải

Trực giác:

- Bạn có thể coi **quá trình xóa các đỉnh trong T1 là một phép duyệt ngược**, và **xây T2 theo thứ tự xóa**.
- Vì mỗi thao tác chọn đỉnh x rồi đặt $a[x]$ làm cha \rightarrow đỉnh nào xóa sau cùng sẽ là **gốc của T2**.

Ý tưởng chính:

- Gán thứ tự xóa (reverse)** sao cho **các đỉnh x bị xóa ngược theo cây T2**, bắt đầu từ lá đến gốc.
- Việc tạo được cây T2 từ T1 theo quy tắc chỉ hợp lệ nếu với **từng đỉnh**, khi bị xóa thì các hàng xóm còn lại phải đúng như cách nối trong T.

Thuật toán:

- Chọn thử từng đỉnh làm gốc của T ($n \leq 5e5$).
- Với mỗi đỉnh đó:
 - Duyệt cây T từ gốc (DFS/BFS), gán cha $par[u]$.
 - Duyệt cây T **từ dưới lên**, với mỗi đỉnh u , kiểm tra xem các hàng xóm v trong T1 đã được xét chưa, nếu chưa thì $a[v] = u$ hợp lệ không?

- Cần cấu trúc dữ liệu để **mô phỏng xóa đỉnh** khỏi T1 theo thứ tự post-order duyệt cây T.

→ Tổng thời gian tối ưu: $O(n \log n)$

Bạn muốn mình cài thuật toán này bằng Python/C++ mẫu không?

Bài K. LR String

 **Giới hạn thời gian:** 2 giây

 **Giới hạn bộ nhớ:** 1024 MB

Đề bài (dịch tiếng Việt)

Bạn có một chuỗi ban đầu chỉ gồm các ký tự `'L'` và `'R'`, gọi là chuỗi gốc s .

Tuy nhiên, người em trai đã **thực hiện một số thao tác** trên chuỗi này, mỗi thao tác là:

- **Chọn một ký tự `'L'` (không phải ký tự đầu tiên) → xóa ký tự bên trái** của nó.
- **Chọn một ký tự `'R'` (không phải ký tự cuối cùng) → xóa ký tự bên phải** của nó.

Sau các thao tác đó, chuỗi gốc bị thay đổi thành một chuỗi mới.

Nhiệm vụ

Cho chuỗi gốc s , và một danh sách gồm q chuỗi nhỏ t_1, t_2, \dots, t_q , bạn cần xác định:

- Mỗi chuỗi t_i có thể là kết quả sau khi thực hiện một chuỗi thao tác (theo quy tắc trên) từ s hay không?



Input

- Dòng đầu: số nguyên T — số lượng test
- Mỗi test gồm:
 - 1 dòng chứa chuỗi s
 - 1 dòng chứa số nguyên q
 - q dòng tiếp theo: mỗi dòng là chuỗi t_i



Ràng buộc:

- Tổng độ dài các chuỗi s , tổng số chuỗi q , và tổng độ dài các chuỗi t_i **trên tất cả các test** không vượt quá 10^6



Output

- Với mỗi chuỗi t_i , in `"YES"` nếu có thể là kết quả của thao tác từ s , `"NO"` nếu không.



Ví dụ

Input

```
2
RRLRRLL
```



```
4
LLLLL
LLR
LRLR
R
RLLLLLL
3
LLLLL
RL
RRL
```

Output

```
NO
YES
NO
YES
YES
YES
NO
```

✨ Ý tưởng giải thuật

Đây là một **bài toán kiểm tra subsequence có điều kiện**.

Cách hiểu:

Ta có thể hiểu:

- ``L`` có thể **xóa bên trái nó** → tức là ``L`` có thể "bảo vệ" ký tự trước nó khỏi bị match vào kết quả.
- ``R`` có thể **xóa bên phải nó** → tức là ``R`` có thể "bảo vệ" ký tự sau nó khỏi bị match.

✅ Quan sát:

Một ký tự của chuỗi kết quả t **phải xuất hiện trong chuỗi gốc s theo đúng thứ tự**, nhưng:

- Một số ký tự của s **có thể bị xóa** do nằm **bên trái một** ``L'`` hoặc **bên phải một** ``R'``.

⇒ **Thuật toán hợp lý** là:

✓ **Duyệt từ cuối chuỗi s ngược lên**, lưu số ``L'`` và ``R'`` đã gặp để biết **vị trí nào bị xóa**.

Sau đó, từ chuỗi đã giữ lại, kiểm tra xem **chuỗi t** có phải là subsequence không.

Bạn có muốn mình cài thuật toán mẫu bằng Python hoặc C++ cho bài này không?

Bài L. Flipping Paths

 **Giới hạn thời gian:** 1 giây

 **Giới hạn bộ nhớ:** 1024 MB

Đề bài (dịch tiếng Việt)

Grammy có một tờ giấy hình chữ nhật kích thước $n \times m$, chia thành các ô vuông đơn vị.

Ban đầu, mỗi ô có màu **đen** (``W'``) hoặc **trắng** (``B'``).

Grammy muốn **tô cho toàn bộ bảng cùng màu** (tất cả ô đen hoặc tất cả ô trắng). Để làm điều đó, cô có thể thực hiện **tối đa 400 lần thao tác**, mỗi thao tác như sau:

- Đặt bút tại ô $(1, 1)$ (góc trên trái).
- Vẽ một đường đi đến ô (n, m) (góc dưới phải), chỉ đi **sang phải ('R')** hoặc **xuống dưới ('D')**.

3. Lật màu tất cả các ô đi qua (đen thành trắng, trắng thành đen).
 4. Xóa đường đi, và tiếp tục thao tác tiếp theo (nếu cần).
-

Nhiệm vụ

- Với mỗi test case, xác định có thể biến toàn bộ bảng về **một màu duy nhất** sau tối đa 400 thao tác không?
 - Nếu **không thể**, in `"NO"`.
 - Nếu **có thể**, in:
 - `"YES"`
 - Số thao tác đã dùng
 - Danh sách các đường đi từ $(1, 1) \rightarrow (n, m)$ bằng chuỗi ký tự `'R'` và `'D'`
-

Input

- Dòng đầu: số test case T
- Với mỗi test:
 - Dòng đầu: hai số nguyên n, m
 - n dòng tiếp theo: mỗi dòng có m ký tự `'W'` hoặc `'B'`

Ràng buộc:

- $1 \leq T \leq 500$
- $1 \leq n, m \leq 200, n \cdot m \geq 2$

- Tổng $\sum n \leq 1000$, tổng $\sum m \leq 1000$

Output

- Với mỗi test case, in:
 - `"NO"` nếu không thể
 - `"YES"` nếu có thể, tiếp theo:
 - Số thao tác k
 - k dòng mô tả các đường đi (chuỗi gồm `'R'` và `'D'`, dài đúng $n + m - 2$)

Ví dụ

Input

```
4
3 3
WBB
BWB
BBW
1 5
WWWWW
2 2
BB
BB
4 1
W
B
B
W
```

Output

YES

2

RRDD

DDRR

YES

0

YES

0

NO

✨ Tóm tắt cách giải

- Mỗi thao tác giống như "flip XOR" một đường đi trên bảng.
- Tổng số trạng thái có thể biểu diễn bằng vector nhị phân $n \times m$.
- Nếu số ô khác màu là chẵn (hoặc phù hợp kiểu parity), thường có thể đưa về một màu.
- Với bảng nhỏ (tổng ô ≤ 1000), bạn có thể đơn giản **đảo màu từng ô khác màu**, bằng cách chọn một đường đi có giao với ô đó.

Bạn có muốn mình cài thuật toán Python/C++ mẫu cho bài này không?

Bài M. Godzilla

 **Giới hạn thời gian:** 5 giây

 **Giới hạn bộ nhớ:** 1024 MB

Đề bài (dịch tiếng Việt)

Thành phố Bytetown là một lưới kích thước $n \times m$. Godzilla sẽ **ghé thăm từng ô đúng một lần**.

Ở mỗi ô (i, j) , Godzilla có thể:

- Không làm gì, hoặc
- Tiêu tốn năng lượng $e(i, j)$ để tấn công **một trong hai cách**:
 1. **Horizontal Atomic Breath**: tấn công toàn bộ hàng i
 2. **Vertical Atomic Breath**: tấn công toàn bộ cột j

▲ **Mỗi kiểu tấn công chỉ được thực hiện đúng một lần trên mỗi ô** (tức là: tại một ô, chỉ thực hiện tối đa 1 trong 2 kiểu tấn công).

Yêu cầu

Godzilla muốn mỗi ô được:

- Tấn công **bằng Horizontal đúng 1 lần**
- Tấn công **bằng Vertical đúng 1 lần**


 Nghĩa là: ta cần chọn đúng **1 ô mỗi hàng để dùng Horizontal**, và **1 ô mỗi cột để dùng Vertical**

→ Từ đó tạo nên một **tổ hợp gồm $n + m$ ô tấn công**, không trùng lặp.

Nhiệm vụ

Tìm cách chọn các ô tấn công **thoả mãn quy tắc** để:

- Tổng **sát thương** (cộng tất cả giá trị $d(i, j)$ tại ô được tấn công) là lớn nhất
- Đồng thời, tổng năng lượng tiêu hao $\bmod 4 = k$ với $k \in \{0, 1, 2, 3\}$

 In ra **4 dòng**, mỗi dòng là đáp án tối đa khi năng lượng tổng chia 4 dư 0, 1, 2, 3. Nếu không có cách nào thỏa mãn, in `-1`.

Input

- Dòng đầu: n, m ($2 \leq n, m \leq 75$)
- n dòng tiếp theo: ma trận d (sát thương)
- n dòng tiếp theo: ma trận e (năng lượng)

Ví dụ

Input

```
2 2
1 2
3 4
2 1
0 3
```

Output

```
-1
-1
10
-1
```

Input

```
3 3
1 2 3
4 5 6
7 8 9
2 1 0
0 3 2
1 2 1
```

Output

```
35
38
37
36
```

✨ Tóm tắt hướng giải

- Chọn một **ô trên mỗi hàng** để đánh horizontal → chọn n ô.
- Chọn một **ô trên mỗi cột** để đánh vertical → chọn m ô.
- Mỗi ô được phép được chọn tối đa một lần cho một trong hai kiểu tấn công → các chọn này **không giao nhau**.

🧠 Ý tưởng:

- Duyệt tất cả cách chọn n ô (mỗi hàng 1 ô) cho horizontal
- Với mỗi tổ hợp đó, loại bỏ các ô đã dùng → duyệt tất cả cách chọn 1 ô mỗi cột từ phần còn lại → vertical
- Dùng **dynamic programming với modulo 4** để lưu trạng thái theo năng lượng

- Trạng thái: $dp[mask][modulo] = max_damage$

Kỹ thuật:

- Tổng số cách chọn nhỏ hơn 2^{n+m} nếu ta gộp theo trạng thái → khả thi với $n + m \leq 150$
 - Tối ưu bằng bitmasking, memoization, pruning theo modulo
-

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