

# universal cup 3 stage 17 : Jinan

Created: 5/14/2025 21:00

Updated: 5/14/2025 21:15

Exported: 5/14/2025 21:16

## 🌟 Bài A: Kẻ Ngốc (The Fool)

🕒 **Giới hạn thời gian:** 1 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

““Nơi này mang hương vị của nhiều cảm xúc.  
Hào hứng, giận dữ, vui vẻ và... mận? Mận là một cảm xúc à?”  
— **Neeko**, Tắc kè hoa tò mò”

### 📖 Mô tả:

Neeko đến từ một bộ tộc Vastaya cổ xưa, có khả năng biến hình và hòa nhập vào đám đông bằng cách sao chép vẻ ngoài của người khác, thậm chí cảm nhận được cảm xúc để phân biệt bạn – thù. Trong một nhiệm vụ, Neeko đã thâm nhập vào một **lưới các ô vuông**, mỗi ô là một chuỗi ký tự có độ dài **k**.

Lưới có kích thước **n × m**, và **mọi ô đều giống nhau** ngoại trừ **duy nhất một ô** — đó chính là ô của **Neeko**, khác biệt với phần còn lại ít nhất **một ký tự**.

👉 Nhiệm vụ của bạn là **tìm tọa độ của ô Neeko đang đứng**.

### 📥 Input:

- Dòng đầu tiên gồm 3 số nguyên:  $n, m, k$

— với:

$$2 \leq n, m \leq 200,$$

$$1 \leq k \leq 10.$$

- Tiếp theo là  **$n$  dòng**, mỗi dòng chứa đúng  **$m \cdot k$  ký tự** liên tục, không có khoảng trắng hay ngắt dòng.

Mỗi dòng đại diện cho một hàng trong lưới.

Trong đó, chuỗi con từ vị trí  $(j-1) \cdot k + 1$  đến  $j \cdot k$  trong dòng thứ  $i$  tương ứng với ô  $(i, j)$ .

 **Đảm bảo đầu vào hợp lệ và có duy nhất một lời giải.**

## Output:

- In ra 2 số nguyên  $r$  và  $c$  là **chỉ số dòng** và **cột** của ô mà Neeko đang đứng (ô duy nhất khác biệt).

## Ví dụ:

### Input:

```
3 5 3
QWQQWQQWQQWQQWQ
QWQQWQQWQQWQQWQ
QWQQWQQWQQWQQWQ
```

### Output:

## 👉 Tóm tắt:

- Lưới  $n \times m$ , mỗi ô là chuỗi độ dài  $k$ .
- Có đúng **một ô khác biệt**, những ô còn lại đều giống nhau.
- Tìm tọa độ  $(r, c)$  của ô đó.

## 📄 Bài B: Ảo Thuật Gia (The Magician)

🕒 **Giới hạn thời gian:** 1 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

““Và bây giờ, tấm màn được kéo lên.”

— “????”

## 📖 Mô tả bài toán:

Bạn được cấp một số **lá bài Tarot đặc biệt** (The Lovers, Death, The Star, The Moon, The Sun, The World) và một tập hợp các **lá bài Tây thông thường** trong tay. Mỗi lá Tarot có **năng lực biến đổi** bộ bài thường theo những cách khác nhau.

Bạn cần **sử dụng các lá bài Tarot (mỗi lá dùng tối đa 1 lần)** để tạo ra **nhiều bộ “Flush” nhất có thể** từ những lá bài thường đang có.

---

## ★ Định nghĩa “Flush”:

Một **Flush** là một **bộ gồm 5 lá bài** có cùng **chất (suit)**.

- Các chất có thể là: ♦ (Diamonds), ♣ (Clubs), ♥ (Hearts), ♠ (Spades)
- Trong đó, **Wild Card (⊗)** có thể đóng vai bất kỳ chất nào.
- Các lá dùng cho một Flush sẽ **bị loại khỏi tay** và **không được tái sử dụng**.

---

## 🎴 Các lá Tarot và năng lực:

Lá Tarot	Năng lực
The Star	Đổi tối đa 3 lá bài sang ♦ (Diamonds)
The Moon	Đổi tối đa 3 lá bài sang ♣ (Clubs)
The Sun	Đổi tối đa 3 lá bài sang ♥ (Hearts)
The World	Đổi tối đa 3 lá bài sang ♠ (Spades)
The Lovers	Đổi 1 lá bài thường thành ⊗ (Wild Card)
Death	Chọn đúng 2 lá, <b>thay một lá bằng bản sao hoàn chỉnh</b> của lá còn lại (bao gồm cả chất, số và trạng thái ⊗ nếu có)

## 📝 Lưu ý quan trọng với Wild Card:

- Khi một lá bài đã trở thành ⊗ thì **không thể bị thay đổi lại** bởi các lá khác.

- Nhưng nếu dùng **Death để biến 1 lá ☸ thành bản sao của 1 lá bình thường**, thì lá mới **không còn là Wild nữa**.
- 



## Input:

- Dòng đầu là số test: ``T`` ( $1 \leq T \leq 13$ )
- Với mỗi test:
  - Dòng 1: ``n`` (số lá bài đang cầm,  $1 \leq n \leq 52$ )
  - Dòng 2: ``n`` lá bài, mỗi lá là chuỗi 2 ký tự: [số][chất], ví dụ: ``4H`` (4♥), ``TS`` (10♠)
  - Dòng 3: 6 số nguyên ``t1 t2 t3 t4 t5 t6`` (mỗi số là 0 hoặc 1)
    - t1: The Star
    - t2: The Moon
    - t3: The Sun
    - t4: The World
    - t5: The Lovers
    - t6: Death

⚠ Đảm bảo: tổng tất cả số lá bài trong các test  $\leq 104$ .

---



## Output:

Với mỗi test, in ra **số Flush lớn nhất có thể chơi được**.

---



## Ví dụ:

### Input:

```
4
5
2H 3H 4H 5H 6D
1 1 1 1 0 0
5
2S 3S 4D 5C 6D
0 0 1 0 1 1
5
2S 3S 4D 5C 6D
0 0 1 0 1 0
13
AS 2S 3S 4S 5H 6H 7H 8H 9H TH JH QH KH
0 0 0 0 0 1
```

### Output:

```
1
1
0
2
```

## Tóm tắt ý tưởng bài toán:

- Từ các lá bài đang có, đếm số lá thuộc từng chất.
- Dùng các lá bài Tarot để chuyển đổi hoặc tạo  $\otimes$  giúp tăng số Flush có thể tạo.
- Mỗi lần tạo được một Flush (5 lá cùng chất hoặc có  $\otimes$  phụ trợ), loại bỏ 5 lá đó và tiếp tục.

- Cần chiến lược hợp lý để chọn cách dùng các Tarot sao cho tổng số Flush là **tối đa**.
- Bài toán mang tính **tối ưu tổ hợp + greedy + thử tất cả cấu hình dùng bài Tarot**.

## Bài C: Hoàng Hậu (The Empress)

 **Giới hạn thời gian:** 2 giây

 **Giới hạn bộ nhớ:** 1024 MB

---

### Mô tả bài toán:

Capoo đã phát minh ra một ngôn ngữ lập trình thú vị có tên là **Push-Pop**. Đây là một ngôn ngữ được thông dịch, bắt đầu với một **stack trống (vô hạn dung lượng)** và đọc dòng lệnh đầu tiên.

Chỉ có **hai loại lệnh** trong ngôn ngữ:

---

### Cấu trúc lệnh:

1. ``POP a GOTO x; PUSH b GOTO y``

- Nếu phần tử trên cùng của stack là ``a``, thì **pop** nó ra khỏi stack và chuyển tới lệnh thứ ``x``.
- Nếu không phải ``a``, thì **push b** vào stack và chuyển tới lệnh thứ ``y``.

2. ``HALT; PUSH b GOTO y``

- Nếu **stack rỗng**, thì **dừng chương trình (HALT)** sau khi chạy dòng này.
- Nếu **stack không rỗng**, thì **push b** vào stack và chuyển tới dòng thứ ``y``.

---

## Yêu cầu:

Bạn cần **thiết kế một chương trình gồm tối đa 64 lệnh**, sao cho **chạy đúng**  $k$  dòng lệnh thì **dừng lại (HALT)**.

- Mỗi lệnh được đánh số từ 1 đến  $n$ .
- Mỗi dòng cần in đúng định dạng một trong hai dạng lệnh ở trên.
- Mỗi giá trị  $a$ ,  $b \leq 128$ .
- Giá trị  $x$ ,  $y$  phải nằm trong  $[1, n]$ .
- $k$  là một **số lẻ**, và  $1 \leq k \leq 2^{31} - 1$ .

 **Đảm bảo rằng luôn tồn tại lời giải cho mọi  $k$  được cho.**

---

## Input:

- Một số nguyên  $k$  (số dòng lệnh cần chạy trước khi HALT), **luôn lẻ**
- 

## Output:

- Dòng đầu:  $n$  — số lượng dòng lệnh trong chương trình ( $1 \leq n \leq 64$ )
  - $n$  dòng tiếp theo: mỗi dòng là một lệnh Push-Pop đúng cú pháp
- 

## Ví dụ:



### Input:

1

### Output:

1  
HALT; PUSH 1 GOTO 1

### Input:

5

### Output:

5  
POP 1 GOTO 2; PUSH 1 GOTO 2  
HALT; PUSH 1 GOTO 3  
POP 1 GOTO 4; PUSH 2 GOTO 4  
POP 1 GOTO 2; PUSH 2 GOTO 4  
HALT; PUSH 99 GOTO 4

### 👉 Tóm tắt ý tưởng thuật toán:

- Bạn cần viết một chương trình chạy **chính xác** `k` **bước** (mỗi bước là một lệnh).
- Vì `k` là **số lẻ**, số lệnh sẽ là kiểu cây nhị phân chạy theo stack: mỗi PUSH tăng số bước, mỗi POP kiểm tra điều kiện.

- Mục tiêu là **thiết kế một cấu trúc vòng lặp (hoặc dạng nhị phân)** sao cho sau đúng `k` bước thì đến được dòng `HALT`.

🔧 Bài toán thực chất là một **bài toán xây dựng (constructive)** — bạn chỉ cần **in ra chương trình hợp lệ** chạy đúng `k` bước.

---

Bạn muốn mình giải hoặc sinh lời giải cho `k` cụ thể nào không?

## 👑 Bài D: Hoàng Đế (The Emperor)

🕒 **Giới hạn thời gian:** 2 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

---

### 📖 Mô tả bài toán:

Capoo đã phát minh ra một ngôn ngữ lập trình thú vị mang tên **Push-Pop**. Đây là một ngôn ngữ được thông dịch. Bộ thông dịch bắt đầu với **một ngăn xếp rỗng (stack) có dung lượng vô hạn**, và bắt đầu thực thi từ dòng lệnh đầu tiên.

Ngôn ngữ này chỉ có **hai loại câu lệnh**:

---



### Các loại câu lệnh:

1. `POP a GOTO x; PUSH b GOTO y`

- Nếu phần tử trên cùng của stack là `a`, thì:
  - Thực hiện thao tác **pop** (loại bỏ phần tử đó).

- Sau đó **chuyển luồng điều khiển** đến dòng lệnh thứ ``x``.
- Nếu không phải ``a``, thì:
  - **Push** ``b`` vào stack.
  - Sau đó **chuyển tới dòng lệnh thứ** ``y``.

## 2. ``HALT; PUSH b GOTO y``

- Nếu **stack rỗng**, chương trình sẽ **kết thúc (HALT)** ngay sau khi thực hiện dòng lệnh đó.
- Nếu **stack không rỗng**, thì:
  - **Push** ``b`` vào stack.
  - Sau đó **chuyển tới dòng lệnh thứ** ``y``.

## **Yêu cầu:**

Capoo muốn nâng cấp trình thông dịch đơn giản để xử lý các chương trình dài hơn.

Bạn được cung cấp một chương trình gồm tối đa **1024 dòng lệnh**.

Nhiệm vụ của bạn là **mô phỏng chương trình đó** và trả về **số bước thực thi** trước khi chương trình dừng.

- Nếu chương trình **không bao giờ dừng**, in ra ``-1``.
- Nếu chương trình có dừng, in ra **số bước thực thi (số lệnh đã chạy), modulo** ``998244353``.

## **Input:**

- Dòng đầu tiên: một số nguyên  $n$  ( $1 \leq n \leq 1024$ ) — số dòng lệnh trong chương trình.
- Tiếp theo  $n$  dòng, mỗi dòng là **một câu lệnh hợp lệ** theo cú pháp Push-Pop.

Mỗi tham số trong lệnh đảm bảo:

- $1 \leq a, b \leq 1024$
- $1 \leq x, y \leq n$



## Output:

- Nếu chương trình **không bao giờ dừng**, in ra  $-1$ .
- Ngược lại, in ra **số lệnh đã thực thi trước khi dừng**, lấy **modulo 998244353**.



## Ví dụ:

### Input 1:

```
1  
HALT; PUSH 1 GOTO 1
```

### Output 1:

```
1
```



### Input 2:

```
5
POP 1 GOTO 2; PUSH 1 GOTO 2
HALT; PUSH 1 GOTO 3
POP 1 GOTO 4; PUSH 2 GOTO 4
POP 1 GOTO 2; PUSH 2 GOTO 4
HALT; PUSH 99 GOTO 4
```

### Output 2:

5

### Input 3:

```
1
POP 1 GOTO 1; PUSH 1 GOTO 1
```

### Output 3:

-1

## 📖 Bài E: Cổ Xe Chiến Xa (The Chariot)

🕒 **Giới hạn thời gian:** 2.077 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

### 📖 Mô tả bài toán:

Trong một thế giới hỗn loạn như **Night City**, phương tiện di chuyển an toàn nhất là **taxi AI của Delamain** — đắt đỏ nhưng hiệu quả.

Bạn cần đi **D mét** qua thành phố và biết rằng giá cước **taxi Delamain** như sau:

---

### **Giá cước taxi:**

- **A eurodollars** là **giá cơ bản** cho **X mét đầu tiên**.
  - **Sau đó, Y mét tiếp theo** có giá **B eurodollars / mét**.
  - **Sau X + Y mét**, giá chuyển sang **C eurodollars / mét**.
- 

### **Tùy chọn đặc biệt:**

Tại **bất kỳ thời điểm nào**, bạn có thể **dừng lại và gọi xe mới**.

➡ Khi làm vậy, bạn **trả tiền** cho chuyến xe trước đó, rồi **khởi đầu hành trình mới với giá cơ bản**.

Bạn có thể lặp lại hành động này **bất kỳ số lần nào**, thậm chí là sau mỗi mét di chuyển.

---

### **Nhiệm vụ:**

Tính **chi phí nhỏ nhất** (số eurodollars) để đi **D mét**, sử dụng các taxi Delamain và có thể **chia nhỏ hành trình bằng cách gọi xe mới**.

---



## Input:

- Dòng đầu tiên chứa số nguyên  $T$  ( $1 \leq T \leq 2077$ ) — số lượng test.
- Mỗi test gồm 6 số nguyên:  
 $A, B, C, X, Y, D$ 
  - $A$ : Giá cơ bản (áp dụng cho  $X$  mét đầu).
  - $B$ : Giá mỗi mét cho  $Y$  mét tiếp theo.
  - $C$ : Giá mỗi mét sau  $X + Y$  mét.
  - $D$ : Quãng đường cần đi.

*“Các giá trị này  $< 102077$ , không có số 0 đứng đầu.*

*Tổng số chữ số của mỗi loại  $A, B, C, X, Y, D$  trên tất cả các test đều không vượt quá **8311 (thập phân).**”*



## Output:

- Với mỗi test, in ra **một số nguyên** — chi phí **nhỏ nhất** (không có số 0 đứng đầu).



## Ví dụ:

### Input:

```
5
160 27 41 3 12 3
160 27 41 3 12 4
160 27 41 3 12 99
1 999 999 1 99 999
999 999 1 1 99 999999999999999999
```



## Output:

```
160
187
3226
999
10000000000099799
```

## 👉 Tóm tắt yêu cầu bài toán:

- Mỗi hành trình có thể được **chia nhỏ** bằng cách gọi nhiều xe.
- Cần **tính toán chiến lược chia đoạn tối ưu**, sao cho tổng chi phí **nhỏ nhất** khi đi  $d$  mét.
- Có thể là:
  - Gọi xe mới sau mỗi  $X$  mét (chỉ trả  $A$ ).
  - Hoặc kết hợp đi dài hơn (thêm chi phí  $B, C$ ) nếu rẻ hơn.
- Tính toán **tham lam + tối ưu hóa cắt đoạn** để tìm chi phí nhỏ nhất.

## 🧙 Bài F: Ẩn Sĩ (The Hermit)

🕒 **Giới hạn thời gian:** 2 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

“Đôi mắt ta thấy Đạo từ trong cõi vắng.”  
— **Xu Dog, nhà giả kim lòng danh**”



---

## Mô tả bài toán:

Xu Dog phát hiện ra rằng việc **loại bỏ tập chất một cách tinh tế** sẽ tăng cường bản chất tâm linh của đan dược. Những tập chất này hóa ra lại là **vấn đề toán học**!

Ông đưa ra một bài toán cho bạn như sau:

---

## Bài toán con cần tính:

Với hai số nguyên **đương**  $n \leq m$ , xét **tất cả các tập con kích thước**  $n$  của tập hợp  $\{1, 2, \dots, m\}$ .

Với **mỗi tập con**, cần tìm:

*“Tập con lớn nhất có thể chọn (sau khi loại bớt vài phần tử) sao cho:”*

- *“Giá trị nhỏ nhất  $\neq$  Ước chung lớn nhất (GCD) của tập con đó.”*

Nếu **không tồn tại tập con không rỗng** nào thỏa mãn, thì giá trị trả về là **0**.

---

## Định nghĩa GCD:

GCD (Greatest Common Divisor): Là **ước số chung lớn nhất** của tất cả phần tử trong tập.

Ví dụ:

- $\text{GCD}(\{6, 9, 15\}) = 3$
- $\text{GCD}(\{5, 7, 9\}) = 1$



## Input:

- Một dòng gồm hai số nguyên:  $m, n$   
— Với  $1 \leq n \leq m \leq 10^5$



## Output:

- Một số nguyên: **Tổng giá trị lớn nhất có thể giữ lại sau khi loại bỏ**, xét trên **tất cả các tập con kích thước  $n$ , modulo 998244353**



## Ví dụ:

### Input:

4 3

### Xét tất cả các tập con size 3 của $\{1, 2, 3, 4\}$ :

- $\{1, 2, 3\} \rightarrow$  loại 1  $\rightarrow$  còn  $\{2, 3\} \rightarrow \text{GCD} = 1, \text{min} = 2 \neq 1 \rightarrow \checkmark \rightarrow$  giữ được 2 phần tử
- $\{1, 2, 4\} \rightarrow$  không thể loại phần tử nào sao cho  $\text{min} \neq \text{GCD} \rightarrow \times \rightarrow 0$
- $\{1, 3, 4\} \rightarrow$  loại 1  $\rightarrow$  còn  $\{3, 4\} \rightarrow \text{GCD} = 1, \text{min} = 3 \neq 1 \rightarrow \checkmark \rightarrow$  giữ được 2 phần tử
- $\{2, 3, 4\} \rightarrow$  giữ nguyên  $\rightarrow \text{GCD} = 1, \text{min} = 2 \rightarrow \checkmark \rightarrow$  giữ được 3 phần tử

### Output:

7



---

**Input:**

11 4

**Output:**

1187

---

**👉 Tóm tắt yêu cầu bài toán:**

- Duyệt **tất cả tập con size**  $n$  từ  $\{1, 2, \dots, m\}$ .
- Với mỗi tập, tìm tập con con lớn nhất thỏa điều kiện:
  - **GCD  $\neq$  Min**
- Lấy tổng kết quả trên tất cả các tập con, **modulo 998244353**.

⚠️ Vì số lượng tổ hợp có thể lên đến  $C(m, n)$  nên cần **ý tưởng tối ưu mạnh** (không được brute-force).

**🎡 Bài G: Bánh Xe May Rủi (The Wheel of Fortune)**

🕒 **Giới hạn thời gian:** 7 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

---

## 📖 Mô tả bài toán:

Trong hành trình khám phá vũ trụ, **The Hierophant** và **The High Priestess** sử dụng **một bánh xe hình đa giác lồi** để nghiên cứu tác động của trọng lực lên các quá trình ngẫu nhiên. Mỗi lần quay bánh xe, **giải thưởng phụ thuộc vào tam giác** được tạo bởi tâm quay nối với 2 đỉnh liên kề.

---

## 🧩 Chi tiết về bánh xe:

- Bánh xe là **đa giác lồi có  $n$  đỉnh**, định nghĩa bởi danh sách các đỉnh  $(x_i, y_i)$  **theo chiều ngược kim đồng hồ**.
  - Bánh xe được chia thành  $n$  **tam giác vùng**, mỗi tam giác tạo bởi:
    - Tâm quay  $o$**
    - Đỉnh  $i$  và đỉnh  $(i \bmod n) + 1$**
- 

## ⚠️ Vấn đề lệch tâm:

Thông thường, nếu tâm quay đúng là trọng tâm (centroid), thì mỗi vùng sẽ có xác suất tỉ lệ với **góc**. Tuy nhiên, ở đây **tâm quay không đặt tại trọng tâm**, khiến kết quả luôn lệch về hướng từ tâm quay đến trọng tâm.

Để tránh kết quả bị **cố định và nhàm chán**, ta sẽ **đặt một viên nam châm nhỏ** (khối lượng  $w$ ) tại **một điểm ngẫu nhiên trong bánh xe**, làm **dịch chuyển trọng tâm**, và từ đó kết quả quay sẽ bị lệch theo hướng mới.

---

## 🎯 Yêu cầu:

Tính **xác suất** mỗi vùng **tam giác i** (giữa đỉnh  $i$  và  $i+1$ ) là vùng chiến thắng sau khi:

- Đặt một **nam châm khối lượng**  $w$  tại **vị trí ngẫu nhiên trong bánh xe** (phân bố đều)
- Và kết quả được xác định bởi **đường thẳng từ tâm quay**  $o$  **đến trọng tâm mới**



### Input:

- Dòng đầu:  $n$  (số đỉnh của đa giác),  $w$  (khối lượng của nam châm)
- $n$  dòng tiếp theo: tọa độ các đỉnh  $x_i, y_i$
- Dòng cuối: tọa độ tâm quay  $ox, oy$



### Output:

- $n$  dòng, mỗi dòng là một số thực  $p_i$ : xác suất tam giác vùng  $i$  là vùng chiến thắng
- **Đảm bảo sai số tuyệt đối hoặc tương đối  $\leq 10^{-6}$**



### Ví dụ:

Input:

```
5 5
1 0
```




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

### Output:

```
0.313777778
0.235555556
0.107555556
0.107555556
0.235555556
```

### Tóm tắt bài toán:

- Cho đa giác lồi chia thành  $n$  vùng tam giác nối với tâm quay.
- Nam châm được đặt ngẫu nhiên trong đa giác → thay đổi trọng tâm → thay đổi hướng xác định vùng thẳng.
- Yêu cầu: tính xác suất **mỗi vùng** trở thành vùng thẳng dựa trên phân bố ngẫu nhiên của trọng tâm mới.
- Cần kiến thức hình học, xác suất, và kỹ thuật mô phỏng hoặc tích phân 2D theo diện tích.

 Bài toán này là một dạng **mô phỏng xác suất hình học trên đa giác lồi có trọng tâm thay đổi**.

### Bài H: Sức Mạnh Làm Tròn (Strength)

 **Giới hạn thời gian:** 3 giây

 **Giới hạn bộ nhớ:** 1024 MB

---

## 📖 Mô tả bài toán:

Trong đời sống hàng ngày, con người thường **làm tròn các con số** để giao tiếp dễ dàng hơn.

Ví dụ:

- Số **98** được làm tròn thành **100**
- Hoặc **145 → 150 → 200**

Tuy nhiên, nếu **làm tròn nhiều lần liên tiếp**, kết quả có thể rất xa so với giá trị ban đầu.

Ví dụ: **2000** có thể đến từ **2001, 1999, 1888**, thậm chí là **11451** nếu làm tròn nhiều bước.

---

## 🔄 Quy tắc làm tròn "tích cực" (aggressive rounding):

Tại **mỗi bước**, chọn một chữ số bất kỳ trong biểu diễn thập phân:

- Gọi số đó là  $x_i$ , ở vị trí  $10^{i-1}$  (chữ số hàng đơn vị là  $i=1$ )
- Nếu  $x_i < 5$ , trừ đi  $x_i * 10^{i-1}$
- Nếu  $x_i \geq 5$ , cộng thêm  $(10 - x_i) * 10^{i-1}$

💡 Có thể thực hiện **bất kỳ số lần (kể cả 0 lần)** và với **bất kỳ chữ số nào**.

---

## 🎯 Nhiệm vụ:

Với một số  $x$ , đếm xem trong khoảng  $[0, z]$  có **bao nhiêu số có thể làm tròn thành  $x$**  (bằng cách áp dụng chuỗi các bước làm tròn tích cực như trên).

---

### Input:

- Dòng đầu tiên: Số test  $T$  ( $1 \leq T \leq 10^5$ )
  - Với mỗi test: hai số nguyên  $x, z$  ( $0 \leq x, z \leq 10^{18}$ )
- 

### Output:

- Với mỗi test, in ra một dòng chứa **số lượng số  $y$  trong  $[0, z]$**  mà sau nhiều lần làm tròn tích cực sẽ trở thành  $x$ .
- 

### Ví dụ:

#### Input:

```
5
0 2147483646
10 100
671232353 1232363
123001006660996 3122507962333010
100019990010301090 44519984489341188
```

#### Output:



2147483647

55

0

1919810

114514

## 👉 Tóm tắt yêu cầu bài toán:

- Cho  $x$  và  $z$
- Tìm **bao nhiêu số**  $y \in [0, z]$  có thể **làm tròn tích cực về**  $x$
- Làm tròn tích cực nghĩa là tại mỗi chữ số:
  - Nếu  $< 5$ : trừ về 0
  - Nếu  $\geq 5$ : cộng về 10
- Có thể làm việc này nhiều lần với các chữ số khác nhau

⚠ Bài toán yêu cầu xử lý số rất lớn (tới  $10^{18}$ ), nên cần tư duy quy về **truy vết làm tròn ngược**, kết hợp **duyet các trạng thái hợp lệ**, hoặc xây dựng **cây truy vết ngược lại từ**  $x$ .

## 🌹 Bài I: Người Treo Cổ (The Hanged Man)

🕒 **Giới hạn thời gian:** 3 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

## 📖 Mô tả bài toán:

Claudette Morel — một nhà thực vật học — trong quá trình nghiên cứu hoa hồng, bị gai đâm vào tay. Để ngăn ngừa điều đó lặp lại, cô muốn **loại bỏ gai**, hay nói cách khác là **biến cây hoa hồng thành "thornless graph"** (đồ thị không có gai).

---

## Định nghĩa "thornless graph":

Một **đồ thị đơn** (không có vòng lặp, không có cạnh trùng lặp) được gọi là **thornless** nếu:

- **Mỗi cạnh** thuộc **duy nhất một chu trình đơn** (simple cycle).

 **Chu trình đơn** là chu trình mà **không có đỉnh nào lặp lại**, trừ đỉnh đầu và cuối trùng nhau.

---

## Nhiệm vụ:

Bạn được cho một **cây** gồm  $n$  đỉnh (có  $n - 1$  cạnh).

Bạn được phép **thêm một số cạnh mới** (không lặp, không tự nối), sao cho:

- Đồ thị kết quả là **thornless**.
- 

## Input:

- Dòng đầu:  $T$  — số lượng test ( $1 \leq T \leq 10^5$ )
- Với mỗi test:
  - Dòng đầu tiên:  $n$  — số đỉnh của cây ( $2 \leq n \leq 3 \times 10^5$ )
  - $n - 1$  dòng tiếp theo: mỗi dòng 2 số  $u, v$  — cạnh ban đầu

🔴 Tổng số đỉnh  $n$  của tất cả test cộng lại không vượt quá  $3 \times 10^5$ .

---

## Output:

Với mỗi test:

- Nếu **không thể** biến cây thành **thornless**, in  $-1$ .
- Nếu **có thể**, in:
  - Dòng đầu: số cạnh  $k$  được thêm vào.
  - $k$  dòng tiếp theo: mỗi dòng là một cạnh  $x \ y$  được thêm vào ( $1 \leq x, y \leq n$ )

✅ Có thể in **bất kỳ lời giải hợp lệ** nào.

---

## Ví dụ:

Input:

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

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

### Output:

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

### 👉 Tóm tắt bài toán:

- Bạn bắt đầu với **một cây** (đồ thị liên thông, không chu trình).
- Cần **thêm cạnh** sao cho mỗi cạnh thuộc **một và chỉ một chu trình đơn**.
- Nếu không làm được, in `-1``.
- Nếu làm được, in danh sách cạnh thêm vào.

🔍 Bài toán yêu cầu bạn phải **hiểu tính chất cấu trúc của đồ thị thornless**, và thiết kế cách **thêm cạnh thông minh** để mỗi cạnh nằm gọn trong đúng **1 chu trình đơn duy nhất**.

### 🌱 Bài J: Tiết Chế (Temperance)

🕒 **Giới hạn thời gian:** 2 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

---

## Mô tả bài toán:

Chen Loong — một nông dân lão luyện — mô tả nông trại của mình bằng **hệ tọa độ 3 chiều**, mỗi cây được xem như **một điểm  $(x, y, z)$** .

---

## Định nghĩa mật độ của cây:

Với mỗi cây  $A_i = (x_i, y_i, z_i)$ , ta định nghĩa **mật độ** của cây là:

- Gọi:
  - $a$  là số cây khác có cùng **tọa độ x** với  $A_i$
  - $b$  là số cây khác có cùng **tọa độ y**
  - $c$  là số cây khác có cùng **tọa độ z**
- Khi đó:

$$\text{mật độ của } A_i = \max(a, b, c)$$

---

## Yêu cầu:

Với mỗi test, bạn cần xác định:

Với từng giá trị  $k = 0$  đến  $n - 1$ , **ít nhất phải xóa bao nhiêu cây** để tất cả cây còn lại đều có **mật độ  $\geq k$** .

 **Lưu ý:** Khi xóa cây, mật độ của các cây khác **có thể thay đổi**!

---

## Input:

- Dòng đầu: `T` — số test ( $1 \leq T \leq 2 \times 10^4$ )
- Với mỗi test:
  - Dòng đầu tiên: `n` (số cây,  $1 \leq n \leq 10^5$ )
  - `n` dòng tiếp theo: tọa độ các cây: `x_i y_i z_i` ( $1 \leq x, y, z \leq 10^5$ )

 **Tổng số cây** của tất cả test  $\leq 2 \times 10^5$



## Output:

- Với mỗi test: in `n` số nguyên trên 1 dòng:  
Số cây cần xóa để thỏa mãn điều kiện **mật độ**  $\geq k$ , với từng `k = 0, 1, ..., n-1`



## Ví dụ:

### Input:

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

### Output:

0 0 2 5 5

0 3 3

## 👉 Tóm tắt yêu cầu bài toán:

- Với mỗi test, bạn cần xác định:
  - Với mỗi ngưỡng mật độ  $k$  từ 0 đến  $n-1$ :
    - Cần **xóa ít nhất bao nhiêu cây** để tất cả cây còn lại có mật độ  $\geq k$
- Mật độ được tính dựa trên **số cây khác có cùng tọa độ x, y, hoặc z**
- Phải xử lý **nhều test**, mỗi test có thể lên tới  $10^5$  cây → yêu cầu thuật toán **tối ưu và thông minh**.

Bài toán mang tính chất **đếm tần suất theo tọa độ và cấu trúc dữ liệu động**, kết hợp với kỹ thuật **greedy / sắp xếp** để kiểm soát thay đổi mật độ khi xóa cây.

## 😈 Bài K: Con Quỷ (The Devil)

🕒 **Giới hạn thời gian:** 7 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

## 📖 Mô tả bài toán:

Bạn có một danh sách gồm `n` cụm từ (phrase), mỗi cụm từ gồm các **từ (word)** viết bằng chữ cái **hoa và thường**.

Bạn cần tạo **viết tắt (abbreviation)** cho từng cụm từ sao cho:

1. Mỗi cụm từ được viết tắt bằng **việc chọn một tiền tố không rỗng** từ **mỗi từ** trong cụm từ đó.
2. Sau đó **nối các tiền tố** lại theo đúng thứ tự → ra viết tắt.
3. **Mỗi viết tắt phải là duy nhất.**
4. Mục tiêu là **tổng độ dài viết tắt của tất cả các cụm từ là nhỏ nhất có thể.**

---

## Chi tiết luật tạo viết tắt:

Ví dụ:

Cụm từ: ``China Collegiate Programming Contest``

Một số viết tắt hợp lệ:

- ``ChCoPrCo`` → dùng tiền tố ``Ch``, ``Co``, ``Pr``, ``Co``
- ``CCPContest`` → dùng ``C``, ``C``, ``P``, ``Contest``

⚠ Một cụm từ có thể có nhiều viết tắt khác nhau tùy vào cách chọn tiền tố.

---

## Yêu cầu:

Tạo ra viết tắt cho `n` cụm từ, sao cho:

- Tất cả viết tắt **khác nhau**
- Tổng độ dài các viết tắt là **ít nhất**



- Nếu **không thể**, in ``no solution``
- 

## Input:

- Dòng đầu: số nguyên ``n`` ( $1 \leq n \leq 128$ )
- ``n`` dòng tiếp theo: mỗi dòng là một cụm từ ( $\leq 128$  từ), mỗi từ chỉ gồm chữ cái ``a-z``, ``A-Z``.

Không có hai cụm từ nào giống hệt nhau.

---

## Output:

- Nếu **không tồn tại cách viết tắt hợp lệ**, in ra:

```
no solution
```

- Ngược lại, in ``n`` dòng – dòng thứ ``i`` là viết tắt cho cụm từ thứ ``i`` trong input.

! Nếu có nhiều cách hợp lệ, in ra **bất kỳ** một cách.

---

## Ví dụ:

### Input 1:

```
5
automated teller machine
active teller machine
active trouble maker
```

always telling misinformation  
American Teller Machinery

### Output 1 (một khả năng):

atm  
atma  
actm  
atem  
ATM

### Input 2:

3  
A AA  
AA A  
A A A

### Output 2:

no solution

### 👉 Tóm tắt bài toán:

- Tạo **viết tắt tối ưu độ dài** cho `n` cụm từ.
- Mỗi viết tắt: ghép các **tiền tố không rỗng** từ từng từ.
- Đảm bảo **không trùng viết tắt**.

- Nếu không thể → in `no solution`.

---

📌 Bài toán yêu cầu kỹ thuật **tìm tổ hợp tiền tố duy nhất cho từng cụm từ** sao cho tổng độ dài là nhỏ nhất — tương đương với **bài toán quy hoạch tổ hợp + kiểm tra trùng + tối ưu độ dài**.

## Bài L: Tòa Tháp (The Tower)

🕒 **Giới hạn thời gian:** 3 giây

💾 **Giới hạn bộ nhớ:** 1024 MB

---

### 📖 Mô tả bài toán:

**Houraisan Kaguya** là một công chúa NEET thích xem video trên một nền tảng giống như Bilibili — gọi là **Mikufans**. Trong video, người dùng có thể để lại **tin nhắn chạy ngang gọi là danmaku**.

Chúng ta chỉ quan tâm đến **top danmaku** — tức là tin nhắn hiển thị **ở dòng trên cùng** của video, mỗi dòng chứa đúng **1 tin nhắn**.

---

### 🧩 Các loại sự kiện:

Trong quá trình phát video, có **n sự kiện** ( $1 \leq n \leq 5 \times 10^5$ ), mỗi sự kiện là:

1. `1 k`

- Một **người dùng mới** gửi **k dòng tin nhắn top danmaku**.

- ID của người dùng này là **số nguyên dương nhỏ nhất chưa từng được dùng**.
- Các tin nhắn sẽ được **gán vào các dòng trống thấp nhất hiện có**, liên tiếp từ trên xuống.

## 2. `2 u`

- **Toàn bộ tin nhắn** của người dùng `u` sẽ bị xóa khỏi màn hình.
- Những dòng chứa tin nhắn đó trở thành **dòng trống**.
- ID `u` chắc chắn hợp lệ và chưa từng bị xóa trước đó.

## 3. `3 l`

- Kaguya muốn biết **ai là người gửi tin nhắn ở dòng thứ `l` ( $1 \leq l \leq 10^9$ )**.
- Nếu dòng đó trống, in ra `0`.



## Input:

- Dòng đầu: số nguyên `n` — số sự kiện.
- `n` dòng tiếp theo: mô tả sự kiện như trên (`1 k`, `2 u`, `3 l`).



## Output:

- Với mỗi truy vấn `3 l`, in ra:
  - ID người dùng nếu dòng `l` có tin nhắn.
  - `0` nếu dòng `l` đang trống.



---

## Ví dụ:

### Input:

```
7
1 2
1 4
3 3
2 1
3 2
1 4
3 7
```

### Output:

```
2
0
3
```

---

## Tóm tắt yêu cầu bài toán:

- Mô phỏng **hàng triệu dòng danmaku**, nhưng chỉ quản lý phần **có nội dung**.
- Cần **phân bổ đoạn dòng liên tục** cho từng user.
- Khi user bị xóa → xóa toàn bộ dòng họ chiếm giữ → các dòng trở nên trống.
- Truy vấn: dòng `1` hiện tại thuộc về ai?

## Gợi ý kỹ thuật (không phải giải):

- Dùng **segment tree động**, **treap**, hoặc **ordered map** để quản lý **đoạn dòng** được gán theo user.
- Dùng **interval tree** để tra xem dòng `l` thuộc đoạn nào (nếu có).
- Gán user ID tăng dần và lưu các đoạn `[start, end]` của từng user để xóa nhanh khi cần.

*“Bài toán là sự kết hợp giữa **quản lý đoạn động**, **gán ID liên tục**, và **truy vấn vị trí theo đoạn** trong không gian cực lớn ( $10^9$  dòng) nhưng **số lượng thao tác hữu hạn** ( $5 \times 10^5$ ).”*

## Bài M: Phán Xét (Judgement)

 **Giới hạn thời gian:** 1 giây

 **Giới hạn bộ nhớ:** 1024 MB

## Mô tả bài toán:

Trong một trò chơi một người chơi tên **Generalissimos**, bạn có một bảng lưới  $n \times m$  gồm các ô với 3 màu:

- `R`: đỏ
- `B`: xanh
- `.`: trắng

## 🌟 Luật tô màu:

- Ban đầu chỉ có:
  - Ô **(a, b)** màu **đỏ** (``R``)
  - Ô **(c, d)** màu **xanh** (``B``)
  - Các ô khác là **trắng** (``.``)
- Người chơi có thể thực hiện **vô số lần** thao tác sau:
  - Chọn một ô **không đặc biệt và hiện tại là trắng** (``.``): gọi là  $(x, y)$
  - Chọn **một ô kề cạnh theo cạnh** (tức là 4 hướng: trên, dưới, trái, phải) với  $(x, y)$ , **đang có màu** (``R`` hoặc ``B``)
  - Tô ô  $(x, y)$**  bằng màu của ô đã chọn ở bước 2
- Ô có thể được tô lại nhiều lần, màu mới **đè lên** màu cũ.

---

## ! Mục tiêu:

Bạn được cung cấp **bản đồ cuối cùng** của bảng sau một số thao tác tô.

Hãy xác định xem trạng thái này có **hợp lệ** hay không (tức là **có thể sinh ra bởi các thao tác hợp lệ**, không phải do “ăn gian”).

---

## 📥 Input:

- Dòng đầu: ``T`` — số lượng test ( $1 \leq T \leq 10^4$ )
- Với mỗi test:
  - Dòng đầu: ``n m`` — số hàng, số cột ( $1 \leq n, m \leq 500$ , đảm bảo ``n·m ≥ 2``)

- Dòng tiếp theo: `a b c d` — vị trí ban đầu của ô `R` và `B` ( $1 \leq a, c \leq n, 1 \leq b, d \leq m$ )
- `n` dòng tiếp theo: mỗi dòng có `m` ký tự `R`, `B` hoặc `.`

 Đảm bảo:

- `(a, b)` chứa ký tự `R`, `(c, d)` chứa `B`
- Không có hai ô đặc biệt trùng nhau
- Tổng số ô của tất cả test  $\leq 250,000$

## Output:

- Với mỗi test, in `YES` nếu cấu hình hợp lệ, ngược lại in `NO`

## Ví dụ:

**Input:**

```
4
3 3
1 1 1 2
RBB
RRR
BBR
6 6
1 1 6 6
RRRRRR
BBBBBR
BRRRBR
BRBBBR
```



BRRRRR  
BBBBBB  
5 5  
3 3 4 4  
BBR.B  
BBR.B  
RRR.B  
. . .BB  
BBBB.  
1 5  
1 1 1 3  
RBBBR

### Output:

YES  
YES  
NO  
NO

### 👉 Tóm tắt yêu cầu bài toán:

- Bắt đầu từ hai ô đặc biệt có màu.
- Từ hai ô đó, màu chỉ có thể **lan sang các ô trắng kề cạnh**.
- Cần kiểm tra xem trạng thái cuối có đúng là kết quả của các bước **lan màu dần dần từ hai ô đặc biệt**, mà **không tô bậy từ xa**.

### 💡 Hướng giải (gợi ý):

- Duyệt từ (a, b) theo BFS → đánh dấu được những ô có thể tô thành **`R`**
- Tương tự, duyệt từ (c, d) theo BFS → đánh dấu ô có thể tô thành **`B`**
- Cuối cùng:
  - Với mỗi ô có màu **`R`** hoặc **`B`**:
    - Nếu nó không thể được tô từ nguồn tương ứng → in **`NO`**
  - Nếu tất cả ô đều hợp lệ → in **`YES`**