Concurso Estructuras de Datos 1 2025/2026

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - Stack Overflow	Pilas
B - A veces, César	Strings
C - Chuches Gratis	Conjuntos
D - Artículo en Revisión	Ad-hoc
E - Envíos Prioritarios	Colas
F - Star Gazer	Matemáticas

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - Stack Overflow	10	4.63 MB
B - A veces. César	4	2 MB
C - Chuches Gratis	6	72 MB
D - Artículo en Revisión	8	200 B
E - Envíos Prioritarios	9	6.3 MB
F - Star Gazer	5	31 KB
- **Total**	42	(+-) 85 MB

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - Stack Overflow	a.merinero.2025	44
B - A veces, César	i.gomezm.2025	35
C - Chuches Gratis	j.camacho.2022	6
D - Artículo en Revisión	j.camacho.2022	13
E - Envíos Prioritarios	-	-
F - Star Gazer	-	-

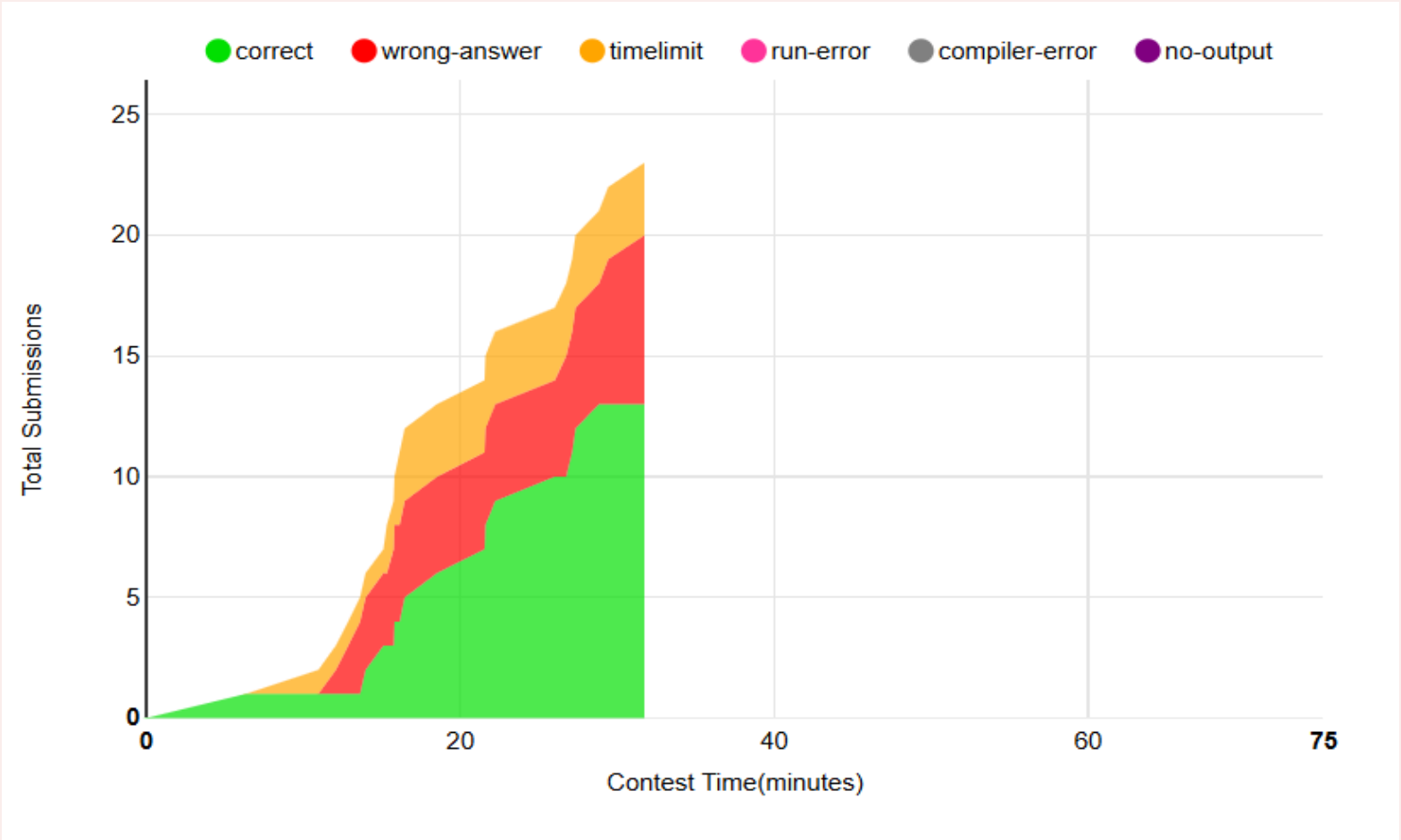
*Antes de congelar el marcador.

Estadísticas*

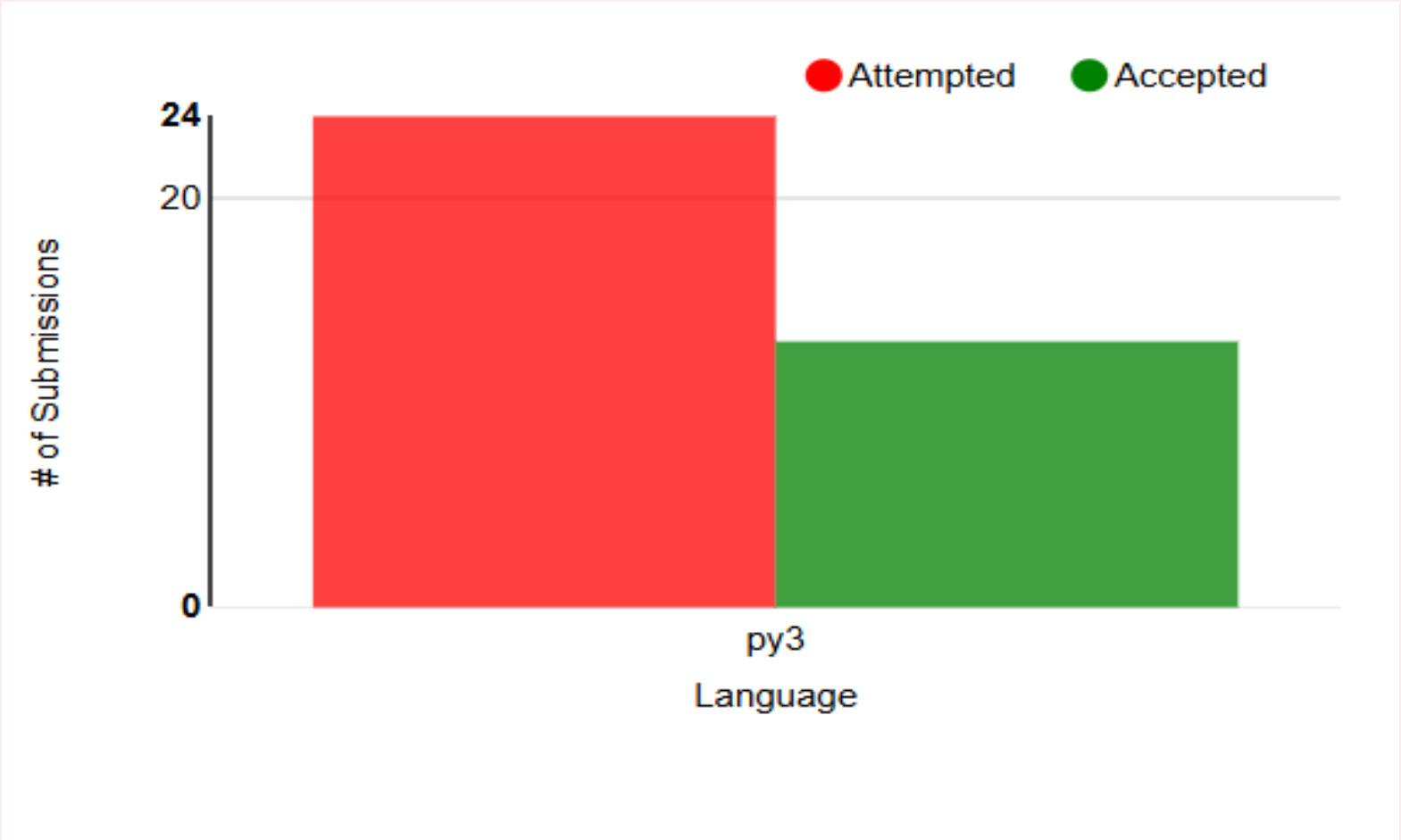
Problema	Válidos	Envíos	% éxito
A - Stack Overflow	1	7	14 %
B - A veces, César	2	6	33 %
C - Chuches Gratis	8	17	47 %
D - Artículo en Revisión	14	15	93 %
E - Envíos Prioritarios	0	1	0 %
F - Star Gazer	0	5	0 %

*Antes de congelar el marcador.

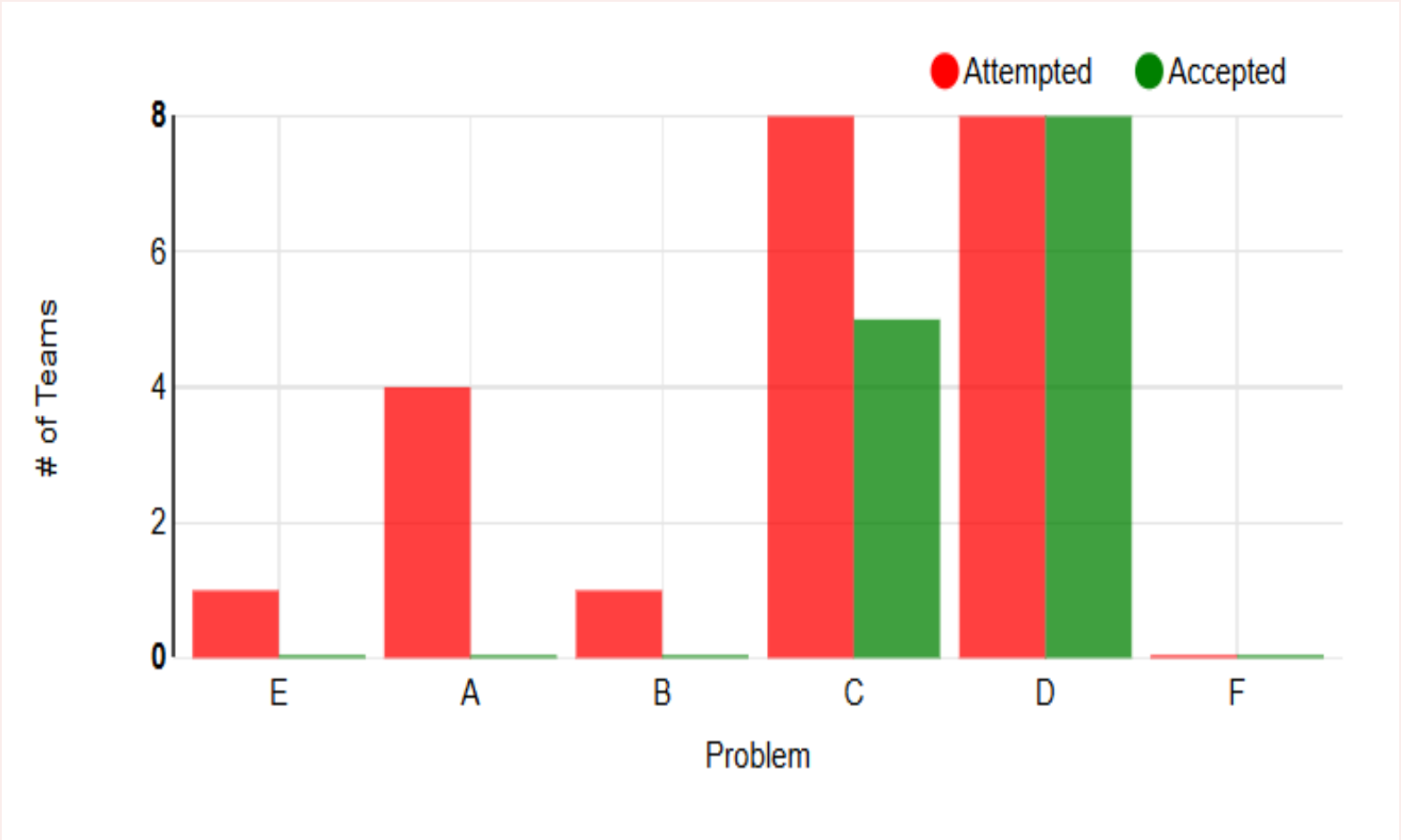
Estadísticas varias



Estadísticas varias



Estadísticas varias



A. Stack Overflow

Envíos	Válidos	% éxito
7	1	14 %

A. Stack Overflow

¿Qué es realmente un **Stack Overflow**?

No es solo una web; es un error de memoria. Para entenderlo, debemos simular la **Call Stack** (pila de llamadas):

- Cada vez que llamamos a una función, se guarda en la pila.
- Cada vez que una función termina (`RETURN`), se saca de la pila.

A. Stack Overflow

¿Qué es realmente un **Stack Overflow**?

No es solo una web; es un error de memoria. Para entenderlo, debemos simular la **Call Stack** (pila de llamadas):

- Cada vez que llamamos a una función, se guarda en la pila.
- Cada vez que una función termina (`RETURN`), se saca de la pila.

El problema: Nuestra memoria es finita. Tenemos un límite N de llamadas simultáneas. Si alcanzamos ese número, el programa explota.

A. Stack Overflow

¿Qué es realmente un **Stack Overflow**?

No es solo una web; es un error de memoria. Para entenderlo, debemos simular la **Call Stack** (pila de llamadas):

- Cada vez que llamamos a una función, se guarda en la pila.
- Cada vez que una función termina (`RETURN`), se saca de la pila.

El problema: Nuestra memoria es finita. Tenemos un límite N de llamadas simultáneas. Si alcanzamos ese número, el programa explota.

Por lo tanto, **¡la estructura de datos a utilizar es una pila!**

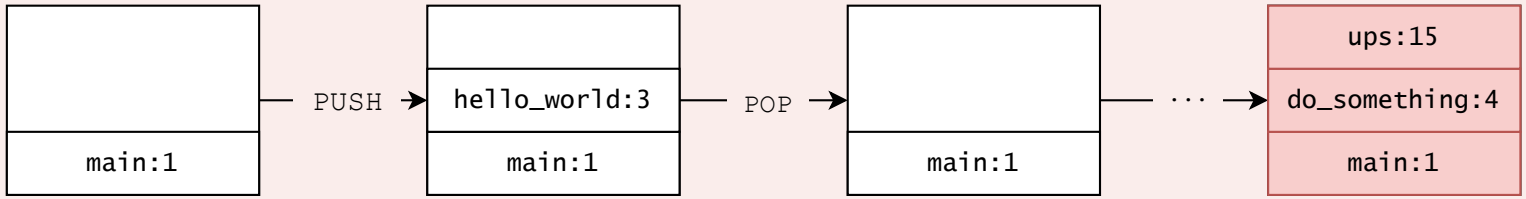
A. Stack Overflow

Entrada del problema:

- N: Capacidad máxima de la pila.
- M: Número total de eventos a procesar.

Eventos:

1. `nombre_funcion:linea` → Operación **PUSH** (Añadir a la pila).
2. `RETURN` → Operación **POP** (Quitar el último elemento).



A. Stack Overflow

```
pila = []
fallo = False

para cada linea en M:
    si no fallo:
        si linea == "RETURN":
            pila.pop()
        sino:
            pila.push(linea)
            si tamano(pila) > N:
                fallo = True
                mientras haya pila:
                    imprimir pila.pop()

si no fallo:
    imprimir "Exited with code 0"
```

B. A veces, César

Envíos	Válidos	% éxito
6	2	33 %

B. A veces, César

Tenemos que descifrar los mensajes que manda Julio César usando su cifrado característico. Este cifrado consiste en desplazar cada letra un número fijo de posiciones en el alfabeto, pero este número no siempre es el mismo.

- En cada caso de prueba se usa el mismo número para la rotación:
  - Se da la primera palabra cifrada $C1$.
  - Se da esa misma palabra decifrada, $D1$.
  - Por último, se da una palabra cifrada $C2$ usando el mismo número de rotaciones.

B. A veces, César

k	r	o	d
-3	-3	-3	-3
h	o	l	a

B. A veces, César

¡OJO! Nos dicen en el enunciado que se debe leer hasta final de fichero:

```
lines = []
for line in sys.stdin:
    if line.strip():
        lines.append(line.strip())
```

Dato: para parar la entrada en el IDE, se usa Ctrl+D.

B. A veces, César

Una vez hemos leído la entrada:

```
for i in range(0, len(lines), 3):
    C1 = lines[i]
    D1 = lines[i+1]
    C2 = lines[i+2]

    N = (ord(C1[0]) - ord(D1[0])) % 26

    descifrada = []
    for c in C2:
        nueva_letra = chr((ord(c) - ord('a') - N) % 26 + ord('a'))
        descifrada.append(nueva_letra)

    print("".join(descifrada))
```

C. Chuches Gratis

Envíos	Válidos	% éxito
17	8	47 %

C. Chuches Gratis

Queremos saber si la persona que pide chuches, las ha recibido ya antes.
No necesitamos saber cuándo ha venido exactamente, ni mantener a todas las personas en un orden específico $\rightarrow LIST$
Solo necesitamos saber si ha pedido ya o no ha pedido $\rightarrow SET$

C. Chuches Gratis

```
leer P
inicializar el set REPARTIDOS
para cada persona:
    leer NOMBRE *
    si NOMBRE está en REPARTIDOS
        imprimir "REPE"
    si no:
        imprimir "CHUCHE"
        añadir NOMBRE al set REPARTIDOS
```

*Usad `.strip()` en Python para que no es den problema posibles espacios / saltos de línea.

D. Artículo en Revisión

Envíos	Válidos	% éxito
15	14	93 %

D. Artículo en Revisión

En este problema, un artículo científico recibe evaluaciones de varios revisores. Cada revisor da un veredicto que puede ser “AC” (Aceptado), “Cm” (Cambios menores), “CM” (Cambios mayores) o “RJ” (Rechazado), ordenados del mejor al peor escenario.

Nuestra misión es leer el número de revisores y sus respuestas, y crear un programa que identifique y muestre por pantalla cuál ha sido el peor veredicto de todos los recibidos.

D. Artículo en Revisión

La solución debe procesar cada veredicto uno por uno dentro de un bucle. Llevamos un registro del peor veredicto visto hasta el momento; si el nuevo veredicto es peor que el guardado, lo actualizamos.

Como idea podemos asignar a cada posible veredicto un número del 1 al 4, siendo 1 el mejor veredicto y 4 el peor, y comparar directamente si el numero asignado es mayor o menor para saber entre dos veredictos cuál es el peor.

D. Artículo en Revisión

```
num = int(input())
verInt = 0
for i in range(num):
    newVer = str(input())

    if newVer == "AC": newVerInt = 1
    elif newVer == "Cm": newVerInt = 2
    elif newVer == "CM": newVerInt = 3
    elif newVer == "RJ": newVerInt = 4

    if newVerInt > veredictInt:
        verInt = newVerInt
        worst_ver = newVer

print(worst_ver)
```

E. Envíos Prioritarios

Envíos	Válidos	% éxito
1	0	0 %

E. Envíos Prioritarios

Amazon está pensando añadir un nuevo servicio llamado P2W que permite a los usuarios pagar para que sus envíos se salten posiciones en la cola de salida.

El problema nos pide procesar un número N de eventos que representan la llegada de paquetes prioritarios, paquetes normales o la salida de camiones con una capacidad determinada.

Nuestro objetivo es imprimir los identificadores de los paquetes que se lleva cada camión, dando siempre prioridad a los envíos P2W y respetando el orden original de llegada de los clientes.

E. Envíos Prioritarios

Para solucionar esto de forma sencilla, mantenemos dos colas de espera separadas: una para los paquetes prioritarios y otra para los normales.

Cada vez que llega un paquete nuevo al centro de reparto, le asignamos su identificador numérico, empezando a contar desde el 0, y lo guardamos en su cola correspondiente.

Cuando un camión está a punto de salir, primero sacamos los paquetes de la cola prioritaria hasta llenar su capacidad máxima y, si aún queda espacio libre, continuamos rellenando el camión con los paquetes que esperan en la cola normal.

E. Envíos Prioritarios

```
n = int(input())
prior, desp, idx = deque(), deque(), 0
for _ in range(n):
    a = input().split()
    if a[0] == 'R':
        capacidad = int(a[1])
        r = []
        while prior and capacidad > 0:
            r.append(str(prior.popleft()))
            capacidad -= 1
        while desp and capacidad > 0:
            r.append(str(desp.popleft()))
            capacidad -= 1
        print(" ".join(r))
    elif a[0] == 'NOR':
        desp.append(idx); idx += 1
    elif a[0] == 'P2W':
        prior.append(idx); idx += 1
```

F. Star Gazer

Envíos	Válidos	% éxito
5	0	0 %

F. Star Gazer

En este problema Ricky quiere saber cuándo se volverán a alinear todos los cometas en el cielo para poder pedir un deseo. Se nos proporciona el número de cometas y el tiempo en meses que tarda cada uno en pasar.

El objetivo final es encontrar el tiempo mínimo en el que todos los cometas coincidirán, lo que matemáticamente se traduce en calcular el mínimo común múltiplo de todos los tiempos dados en la entrada.

F. Star Gazer

La forma general de resolver este problema es utilizando la relación matemática entre el Máximo Común Divisor (MCD) y el Mínimo Común Múltiplo (MCM).

$$a \cdot b = MCM(a,b) \cdot MCD(a,b)$$

Para calcular el Mínimo Común Múltiplo de dos números, multiplicamos ambos y dividimos el resultado entre su Máximo Común Divisor:

$$MCM(a,b) = \frac{a \cdot b}{MCD(a,b)}$$

Para calcular el MCD se utiliza el algoritmo de Euclides, que se puede programar de manera muy simple recursivamente:.

$$\begin{aligned} \text{Caso base:} \quad & MCD(a,0) = a \\ \text{Caso recursivo:} \quad & MCD(a,b) = MCD(b, a \,\%\, b) \end{aligned}$$

¡Esto es aconsejable llevarlo en el CHULETARIO a los concursos!

F. Star Gazer

Existen otras dos alternativas para resolverlo:

1. Si no te acuerdas de las propiedades matemáticas (y se te ha olvidado incluirlo en el chuletario) se podría precalcular los números primos menores a 40, ya que los límites del problema aseguran que el tiempo de los cometas siempre será menor a este número.
   Sabiendo esto, podemos ir factorizando cada número, guardando el exponente máximo de cada primo para luego multiplicarlos y obtener el mínimo común múltiplo (como en el colegio).
2. La segunda opción es utilizar la función nativa de Python que calcula directamente el mcm o el mcd, aunque es posible que en algunos concursos no dejen usar este tipo de funciones automáticas de las librerías.

F. Star Gazer

```
def calcular_mcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

data = iter(sys.stdin.read().split())
T = int(next(data))

for _ in range(T):
    N = int(next(data))
    mcm = int(next(data))
    for _ in range(N - 1):
        siguiente = int(next(data))
        # MCM(a, b) = (a * b) / MCD(a, b)
        mcm = (mcm * siguiente) // calcular_mcd(mcm, siguiente)
    print(mcm)
```