UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

SEGURIDAD EN COMPUTACIÓN

TEMA:

Problemas matemáticos en Criptografía

Docente:

Mg. Rolando Jesús Cárdenas Talavera

Semestre:

VIII

Presentado por:

Philco Puma, Josue Samuel

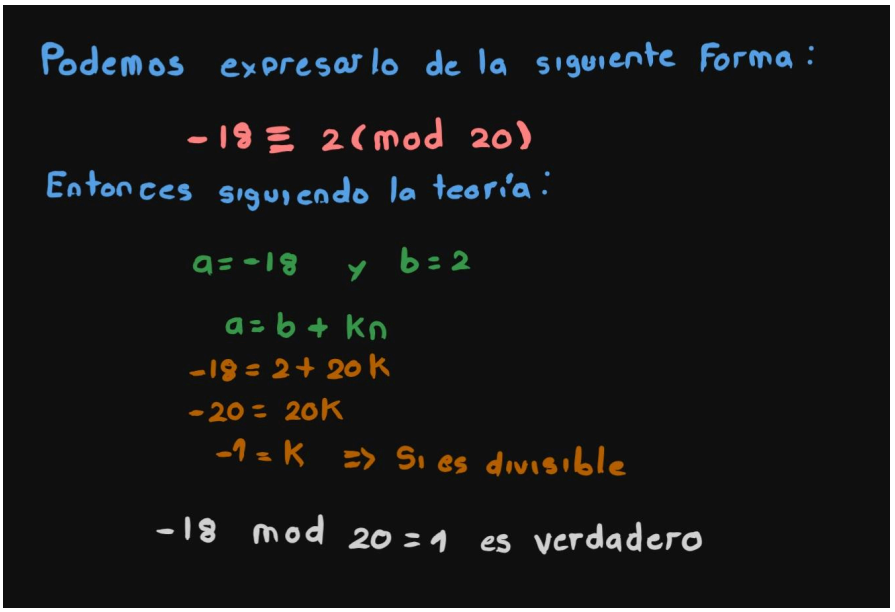
Arequipa - Perú

2025

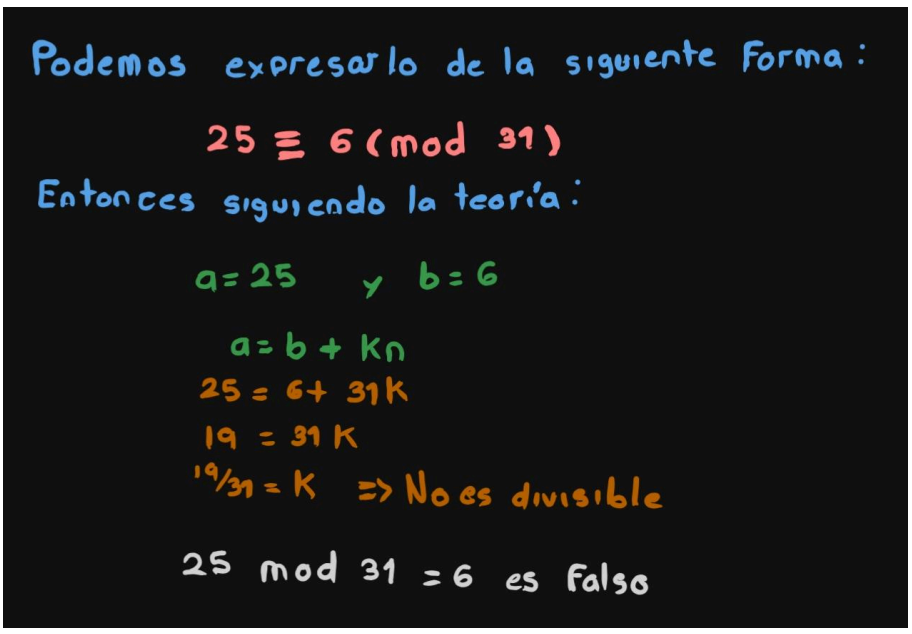
1. **Hallar la congruencia correcta (demostrar) α**

   En las congruencias, se tiene que las denotaciones $a \equiv b\,(mod\,n)$ que es llamada también **Aritmética modular**, significa que a y b deben dejar el mismo al momento de dividirse por n, o que de forma equivalente su diferencia es divisible por n: $a = b + kn$ para algún $k \in Z$. Entonces con esta teoría pasemos a demostrar la congruencia correcta de los siguientes 4 ejemplos:

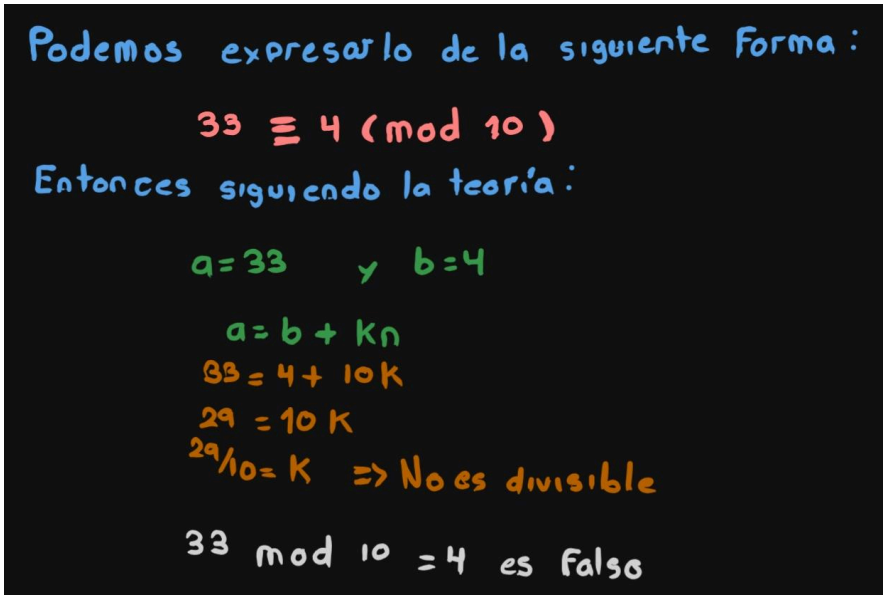
   a) $-\ 18\ mod\ 20\ =\ 2$ (Verdadero)



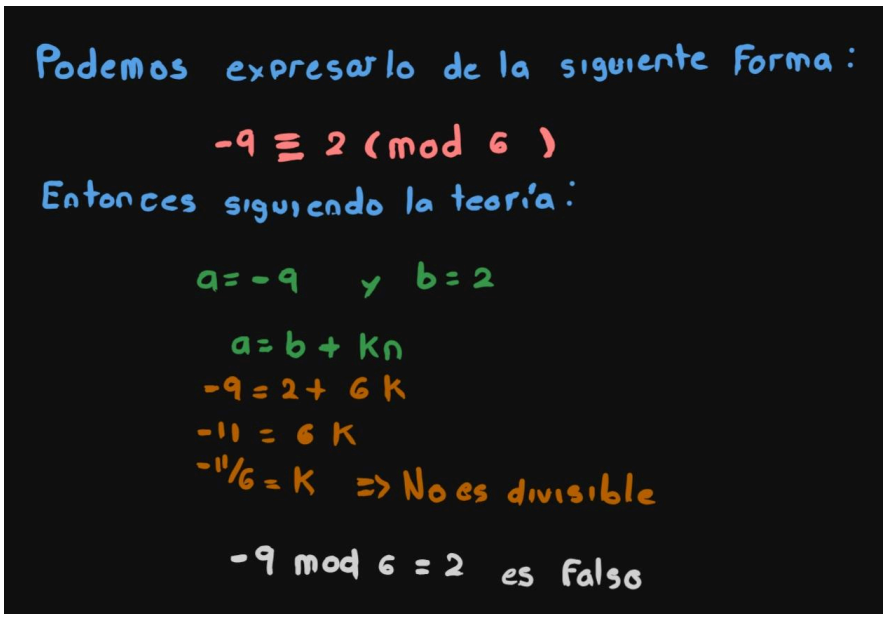
   b) $25\ mod\ 31\ =\ 6$ (Falso)



   c) $33\ mod\ 10\ =\ 4$ (Falso)



d) $-9 \; mod \; 6 = 2$ (Falso)

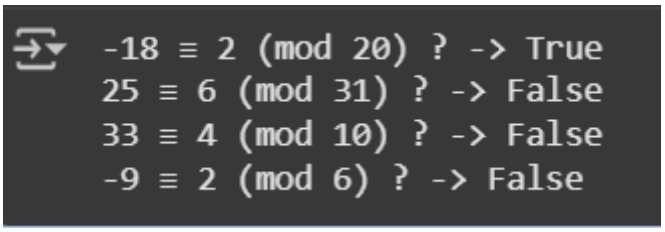


Entonces, si lo llevamos a código en Python quedaría de la siguiente forma:

```python
def are_congruents(a, b, n):
  return a % n == b % n

list_cases = [
    (-18, 2, 20),
    (25, 6, 31),
    (33, 4, 10),
    (-9, 2, 6)
```

```
]

for a, b, n in list_cases:
  print(f"{a} ≡ {b} (mod {n}) ? -> {are_congruents(a, b, n)}")
```

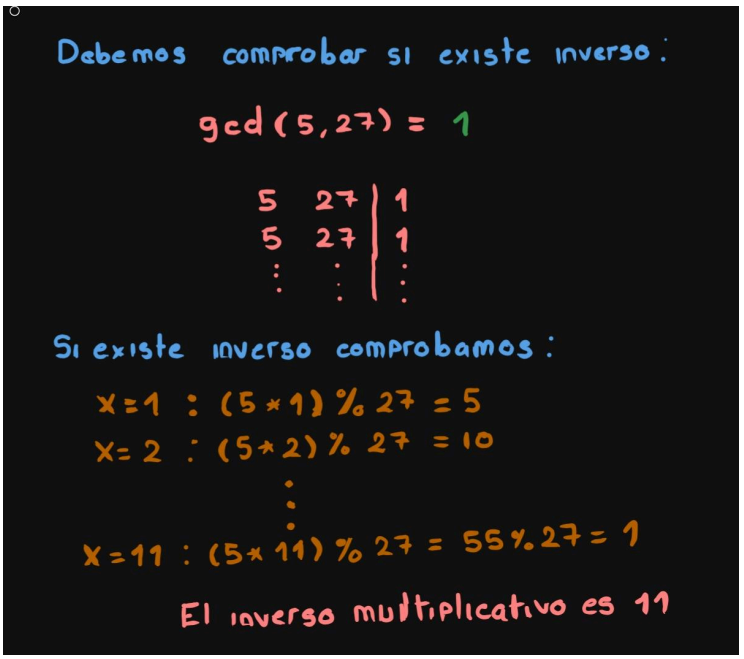
Lo que hace es seguir la teoría de $a \equiv b\ (mod\ n)$ donde es $a\ mod\ n = b\ mod\ n$, entonces la salida que nos da es la siguiente:



2. Hallar el inverso multiplicativo de 5 en módulo 27

El **Inverso Multiplicativo** de x módulo n es y tal que $xy = 1\ (mod\ n)$. Este inverso existe si y sólo si $gcd(x, n) = 1$, es decir, que x y n deben ser coprimos.

Entonces hallemos el inverso multiplicativo del ejercicio y marcar la alternativa correcta:



a) 10
b) 7
c) 4
d) 11

En código quedaría de la siguiente forma:

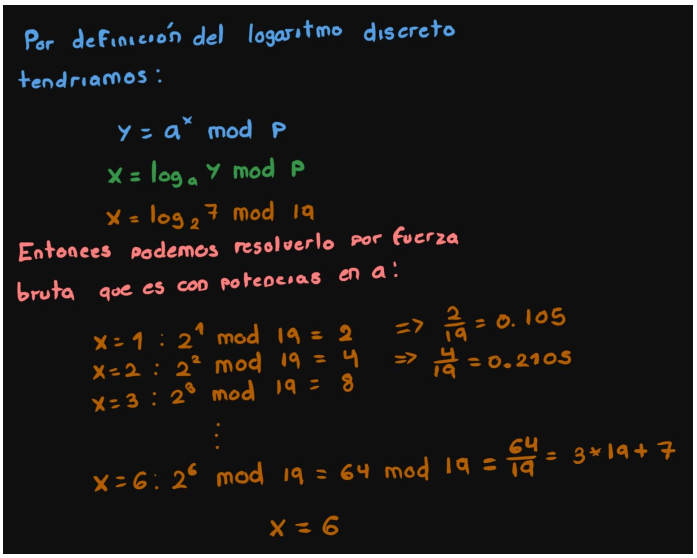
```python
def inverse_multiplicative(a, n):
  for x in range(1, n):
    if (a * x) % n == 1:
      return x
  return None

print(inverse_multiplicative(5, 27))
```

Lo que va a hacer es recorrer a todos lo números posibles hasta $n - 1$ chequeando si $(a \cdot x)\ mod\ n = 1$, si encuentra retorna el valor encontrado, sino dará None.

3. **El valor de x en la expresión** $2^x\ mod\ 19 = 7$

Calcular $y = a^x\ mod\ p$ es un problema de **logaritmo discreto**, es sencillamente calcularlo ya que crece linealmente con el exponente *x*, pero el problema inverso que es hallar *x* conociendo los valores de *a*, *y* y *p*, para resolverlo se pueden hacer en potencias de *a* en módulos pequeños.

Entonces resolvamos la expresión:



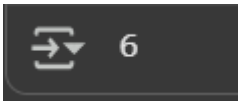
a) 6
b) 5
c) 4

d) 3

En código quedaría de la siguiente forma:

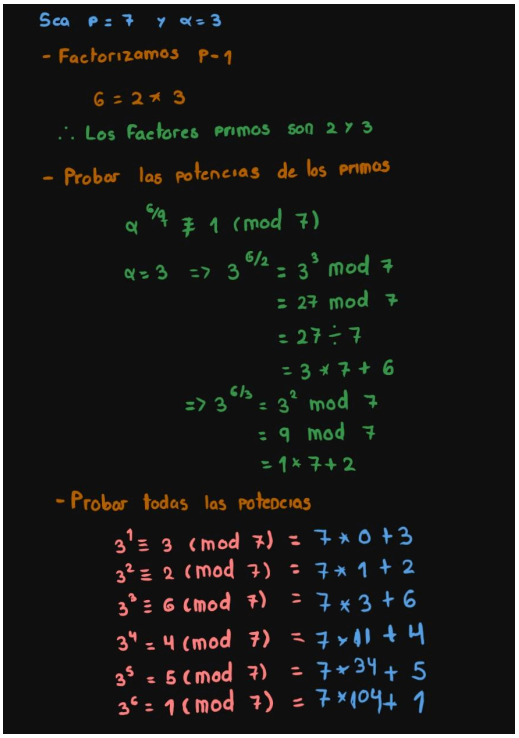
```python
def found_value(a, m):
  for x in range(1, m):
    if (a ** x) % m == 7:
      return x
  return None

print(found_value(2, 19))
```

Lo que va a hacer es recorrer por fuerza bruta todos los números posibles que van a cumplir con la condición del módulo, dando como resultado lo siguiente:



4. **Luego de investigar el concepto de raíz primitiva de un primo, responda: Si *α* es una raíz primitiva del primo *p*, es porque se cumple que:**

En raíces primitivas tenemos un *p* que es un número primo va a ser un entero *g* tal que las potencias de *g* módulo *p* generan todos los números enteros desde 1 hasta *p - 1*. Veamos un ejemplo matemático:

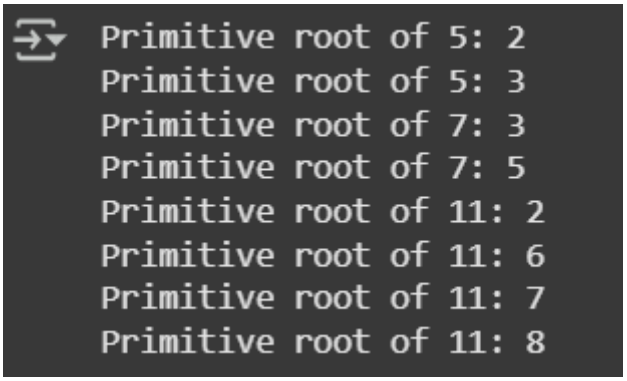


En código sería de la siguiente forma:

```python
def is_prime(p):
    if p < 2:
        return False
    for i in range(2, int(p ** 0.5) + 1):
        if p % i == 0:
            return False
    return True

def gcd(a, p):
    if p == 0:
        return a
    return gcd(p, a % p)

def is_primitive_root(a, p):
    if not is_prime(p) or gcd(a, p) != 1:
        return False

    phi = p - 1
    factors = []
    n = phi
    d = 2
    while d * d <= n:
        if n % d == 0:
            factors.append(d)
            while n % d == 0:
                n //= d
        d += 1
    if n > 1:
        factors.append(n)

    for q in factors:
        if pow(a, phi // q, p) == 1:
            return False
    return True

primes = [5, 7, 11]
for p in primes:
    for a in range(1, p):
        if is_primitive_root(a, p):
            print(f"Primitive root of {p}: {a}")
```

Este código primero verifica si p es un número primo ya que se aplica al grupo multiplicativo de los enteros módulo p. Luego se comprueba que α debe ser raíz primitiva si existe que $\boldsymbol{gcd(\alpha, p) = 1}$. Por último se factoriza en $p - 1$ para obtener

todos sus factores primos y aplica la condición de que si un valor es 1, entonces el orden de α es más pequeño que $p - 1$ y no cumple para ser raíz primitiva, caso contrario que ninguno sea 1, significa que el orden de α es exactamente igual a $p - 1$ y es raíz primitiva.



Entonces si vemos las alternativas dadas:

a) **Es número impar y genera todos los restos del primo *p***
b) **Es número par y genera todos los restos del primo *p***
c) **Es número par o impar y genera todos los restos del primo *p***
d) **Es un número que no genera todos los restos del primo *p***

Nuestra respuesta correcta para este ejercicio es que se va a cumplir que: **Es un número par o impar y genera todos los restos del primo *p*** (Letra **c**).

Referencia de raíz primitiva

[1] “Primitive root - algorithms for competitive programming”. Main Page - Algorithms for Competitive Programming. Accedido el 14 de septiembre de 2025. [En línea]. Disponible: https://cp-algorithms.com/algebra/primitive-root.html

[2] “Finding a primitive root of a prime number”. Mathematics Stack Exchange. Accedido el 14 de septiembre de 2025. [En línea]. Disponible: https://math.stackexchange.com/questions/124408/finding-a-primitive-root-of-a-prime-number#comment287482_124408