Resolución de Tarea 10 - Algoritmos Cuánticos (Fecha: 12 de Diciembre de 2025)

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 - Diseño de Algoritmos I
Septiembre - Diciembre 2025
Estudiante: Junior Miguel Lara Torres (17-10303)

Tarea 10 (9 puntos)

Indice

Pregunta 1

Se realizan 6 simulaciones para N = 35 con un máximo de K = 10 iteraciones.

La simulación del Algoritmo de Shor para la factorización de $N = 35$ se basa en encontrar el **orden** r (el período) de una base aleatoria x módulo N. Si el orden r es par y $x^{r/2} \not\equiv \pm 1 \pmod{N}$, se puede calcular un factor no trivial mediante el máximo común divisor.

A continuación se detalla el proceso, iteración por iteración, para las 6 simulaciones. La probabilidad de éxito del algoritmo se **amplifica** al repetir el proceso hasta $k = 10$ veces, aunque, como se muestra, la factorización a menudo se encuentra rápidamente.

- Simulación 1

Paso	Detalle del Proceso
Iteración 1:	Se selecciona la base aleatoria $\mathbf{x = 26}$.
Búsqueda de orden r:	Se calcula el orden r de 26 mod 35, resultando $\mathbf{r = 6}$.
Verificación de r:	El orden $r = 6$ es **par**. Se calcula $x^{r/2} \pmod{35} = 26^3 \pmod{35} = \mathbf{6}$.
Prueba de no-trivialidad:	Dado que $6 \not\equiv 34 \pmod{35}$, el resultado es **no trivial**.
Factorización:	Se calcula $\gcd(35, 6+1) = \mathbf{7}$ y $\gcd(35, 6-1) = \mathbf{5}$.
Conclusión:	El algoritmo encuentra un factor no trivial ($\mathbf{7 \times 5}$) en **1** iteración.

- Simulación 2

Paso	Detalle del Proceso
Iteración 1:	Se selecciona la base aleatoria $\mathbf{x = 8}$.
Búsqueda de orden r:	Se calcula el orden r de 8 mod 35, resultando $\mathbf{r = 4}$.
Verificación de r:	El orden $r = 4$ es **par**. Se calcula $x^{r/2} \pmod{35} = 8^2 \pmod{35} = 64 \pmod{35} = \mathbf{29}$.

Paso	Detalle del Proceso
Prueba de no-trivialidad:	Dado que $29 \not\equiv 34 \pmod{35}$, el resultado es **no trivial**.
Factorización:	Se calcula $\gcd(35, 29+1) = \mathbf{5}$ y $\gcd(35, 29-1) = \mathbf{7}$.
Conclusión:	El algoritmo encuentra un factor no trivial ($\mathbf{5} \times \mathbf{7}$) en **1** iteración.

- Simulación 3

Paso	Detalle del Proceso
Iteración 1:	Se selecciona la base aleatoria $\mathbf{x = 17}$.
Búsqueda de orden r:	Se calcula el orden r de 17 mod 35, resultando $\mathbf{r = 12}$.
Verificación de r:	El orden $r = 12$ es **par**. Se calcula $x^{r/2} \pmod{35} = 17^6 \pmod{35} = \mathbf{29}$.
Prueba de no-trivialidad:	Dado que $29 \not\equiv 34 \pmod{35}$, el resultado es **no trivial**.
Factorización:	El algoritmo procede a calcular los factores, obteniendo $\mathbf{5} \times \mathbf{7}$.
Conclusión:	El algoritmo encuentra un factor no trivial en **1** iteración.

- Simulación 4

Paso	Detalle del Proceso
Iteración 1:	Se selecciona la base aleatoria $\mathbf{x = 19}$.
Búsqueda de orden r:	Se calcula el orden r de 19 mod 35, resultando $\mathbf{r = 6}$.
Verificación de r:	El orden $r = 6$ es **par**. Se calcula $x^{r/2} \pmod{35} = 19^3 \pmod{35} = \mathbf{34}$.
Prueba de no-trivialidad:	Dado que $34 \equiv N - 1 \pmod{35}$, este es un **resultado trivial** que no garantiza un factor. **El algoritmo continúa.**

Paso	Detalle del Proceso
Iteración 2:	Se selecciona la base $\mathbf{x = 32}$.
Búsqueda de orden r:	Se calcula el orden r de 32 mod 35, resultando $\mathbf{r = 12}$.
Verificación:	El orden $r = 12$ es **par**. Se calcula $x^{r/2} \pmod{35} = 32^6 \pmod{35} = \mathbf{29}$.
Factorización:	Dado que $29 \not\equiv 34 \pmod{35}$, se encuentran los factores $\mathbf{5} \times \mathbf{7}$.
Conclusión:	El algoritmo encuentra un factor no trivial en **2** iteraciones.

- Simulación 5

Paso	Detalle del Proceso
Iteración 1:	Se selecciona la base aleatoria $\mathbf{x = 19}$.
Búsqueda de orden r:	Se calcula el orden r de 19 mod 35, resultando $\mathbf{r = 6}$.
Verificación de r:	El orden $r = 6$ es **par**. Se calcula $x^{r/2} \pmod{35} = 19^3 \pmod{35} = \mathbf{34}$.
Prueba de no-trivialidad:	Dado que $34 \equiv N - 1 \pmod{35}$, este es un **resultado trivial**. **El algoritmo continúa.**

Paso	Detalle del Proceso
Iteración 2:	Se selecciona la base $\mathbf{x = 5}$.
Chequeo Inicial:	Antes de buscar el orden r, se evalúa si $\gcd(x, N) = \gcd(5, 35)$.
Factorización:	El resultado $\gcd(5, 35) = \mathbf{5}$ es un factor no trivial (ya que $1 < 5 < 35$), por lo que se detiene la búsqueda.
Conclusión:	El algoritmo encuentra un factor no trivial en **2** iteraciones, gracias a la prueba inicial de **máximo común divisor**.

- Simulación 6

Paso	Detalle del Proceso
Iteración 1:	Se selecciona la base aleatoria $\mathbf{x = 11}$.
Búsqueda de orden r:	Se calcula el orden r de 11 mod 35, resultando $\mathbf{r = 3}$.
Verificación de r:	El orden $r = 3$ es **impar**. Dado que el algoritmo requiere que r sea par para continuar, el resultado es inválido. **El algoritmo continúa.**

Paso	Detalle del Proceso
Iteración 2:	Se selecciona la base $\mathbf{x = 17}$.
Búsqueda de orden r:	Se calcula el orden r de 17 mod 35, resultando $\mathbf{r = 12}$.
Verificación:	El orden $r = 12$ es **par**. Se calcula $x^{r/2} \pmod{35} = 17^6 \pmod{35} = \mathbf{29}$.
Factorización:	Dado que $29 \not\equiv 34 \pmod{35}$, se encuentran los factores $\mathbf{5} \times \mathbf{7}$.
Conclusión:	El algoritmo encuentra un factor no trivial en **2** iteraciones.

Implementación en Python

El archivo funcional se encuentra en shor.py

```python
import random
import math

# El algoritmo de Shor depende de la exponenciación modular
```

```python
# Utilizamos la función integrada de Python para eficiencia y precisión
# pow(a, b, m) calcula (a^b) mod m
# math.gcd calcula el máximo común divisor

def find_order(x, N):
    """
    Encuentra el orden r de x módulo N.
    (Simula la función de búsqueda de periodo de QFT de forma clásica).
    """
    if math.gcd(x, N) != 1:
        return 0  # Caso trivial: ya encontramos un factor

    r = 1
    val = x
    while val != 1:
        val = pow(x, r, N)
        if val == 1:
            break
        r += 1
    return r

def shor_simulation(N, k_max=10):

    print(f"--- Simulación del Algoritmo de Shor para N = {N} ---")

    for i in range(1, k_max + 1):
        # 1. Seleccionar una base aleatoria x
        x = random.randint(2, N - 1)

        # 2. Verificar la coprimalidad
        d_init = math.gcd(x, N)
        if d_init != 1:
            print(f"\nIteración {i}/{k_max}: Base x = {x}. GCD(x, N) = {d_init}.")
            if 1 < d_init < N:
                print(f"Éxito: ¡Factor no trivial encontrado en el paso inicial! Factor = {d_i
                return True, i
            # Si d_init = N, el test es inútil; si d_init=1, procedemos

        # 3. Encontrar el orden r (período)
        r = find_order(x, N)

        print(f"\nIteración {i}/{k_max}: Base x = {x}. Orden r = {r}.")

        # 4. Verificar condiciones de factorización: r debe ser par
        if r == 0 or r % 2 != 0:
            print(f"Resultado: Orden r={r} no es par/válido. Continuar.")
            continue

        # 5. Verificar condición de no-trivialidad de la raíz
        r_half = r // 2

        # Calculamos x^(r/2) mod N
        x_r_half_mod_N = pow(x, r_half, N)
        print(f"Verificación: x^(r/2) mod N = {x_r_half_mod_N}")

        if x_r_half_mod_N == N - 1:
            print(f"Resultado: x^(r/2) = N-1 ({N-1}). Raíz trivial. Continuar.")
            continue

```

alt text

Figure 1: alt text

```
        # 6. Calcular factores no triviales
        d1 = math.gcd(N, x_r_half_mod_N + 1)
        d2 = math.gcd(N, x_r_half_mod_N - 1)

        if (1 < d1 < N) or (1 < d2 < N):
            factor = d1 if 1 < d1 < N else d2
            print(f"Éxito: ¡Factor no trivial encontrado! {N} = {factor} * {N // factor}"
            return True, i
        else:
            print("Resultado: Factor trivial obtenido. Continuar.")

    print(f"\n--- Simulación Finalizada ---")
    return False, k_max
```

Pregunta 2

Por mi súbalo a la NAZA, ya es cuestión de usted si permite mi atrevimiento xD. Memardo aquí.