UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Aplicaciones de la Mecánica Cuántica
Proyecto final:
Algoritmo de Shor

Dr. Carlos Luna Criado

Nombre:	Matricula:
Giovanni Gamaliel López Padilla	1837522
Ivan Arturo Pla Guzman	1837515

1 de noviembre de 2020

Índice

1. Resumen

En este trabajo se planteará el algoritmo desarrollado por Peter Shor, este algoritmo fue creado para realizar la factorización de un número N en dos números primos a partir de principios cuánticos haciendo uso de estados cuánticos de qubits, compuertas lógicas cuánticas, transformadas de Fourier cuántica, la periodicidad de una función modular y la estimación de la fase de un qubit. Se comparó los tiempos de procesamiento para el número 21 con el algoritmo de criba general de cuerpo de números, el algoritmo de Shor ejecutado en una computadora clásica y el algoritmo de Shor ejecutado en una computadora cuántica, dando así que el algoritmo más eficiente es el de Shor ejecutado en una computadora cuántica, esto debido a su estabilidad en el tiempo de calculo y que es el más rápido de los algoritmos propuestos.
Palabras clave: Algoritmo de Shor, Transformada de Fourier Cuántica, Python, Qiskit, complejidad algorítmica.

2. Introducción

2.1. Historia de la factorización de números

En el área de la matemáticas, la factorización es una técnica que consiste en la descomposición en factores de una expresión algebraica en forma de un producto. El teorema fundamental de la aritmética cubre la factorización de números enteros, este teorema pertenece a la teoría de números. El teorema de factorización afirma lo siguiente:

Teorema 1. *Cada entero positivo tiene una única descomposición en números primos.*

El teorema fue demostrado por primera vez por Euclides, aunque la primer demostración completa apareció en las *Disquisitiones Arithmeticae* de Carl Friedrich Gauss.
El análisis numérico es el estudio de algoritmos, el cual consiste en un conjunto de instrucciones o reglas ordenadas y finitas que permiten realizar una actividad mediante pasos sucesivos con el proposito de resolver problemas matemáticos dentro de la matemática continua, esto da una aproximación numérica. Hay problemas sencillos de calcular como una raiz cuadrada, no tienen soluciones exactas o son costosas en tiempo para obtenerlas.Para obtener una aproximación útil, se debe conocer la presición de los resultados y cuantos recursos se necesita para lograrlo, estos conceptos cuantificados son conocidos como *presición, convergencia, estabilidad y complejidad.*

2.2. Complejidad algorítmica

La complejidad algorítmica, informalmente, es una medida que permite a los programadores conocer la cantidad de recursos que necesita un algoritmo para resolver un problema en función de su tamaño. El objetivo es comparar la eficiencia de algoritmos a la hora de resolver un problema conocido [1]. Para realizar una clasificación en la complejidad de un algoritmo se usa normalmente la notación asintótica, la cual es que por el número de operaciones básicas ejecutadas por un algirtmo se obtiene una función. Cada una se ve denotada por $T(N)$, donde N es

el número de elementos numéricos dentro del algoritmo. Para valores pequeños de N, las constantes que acompañan a los términos de la función T pueden influir de manera significativa al coste total y con ello obtener conclusiones erróneas respecto a la eficiencia del algoritmo. En cambio este tipo de análisis se realiza con números grandes de N. Dependiendo del análisis que se realice, podemos encontrar diferentes tipos de notaciones. La notación más usada es la llamada O grande y se denota por O.

Definición 1. *Se dice que un algoritmo F tiene una complejidad $O(G(N))$ si existen dos constantes C y N_0 para las que se cumpla $|F(N)| < C \cdot G(N)$ para todo $N > N_0$*

En otras palabras, que el algoritmo F tiene una complejidad $O(G)$ si el número de operaciones necesarias es constante para un número grande de N. Un ejemplo de esta clasificación puede observarse en la siguiente tabla: algorit

Notación	Nombre	Ejemplo de algoritmo
$O(1)$	Constante	Acceso a un elemento de un vector
$O(logN)$	Logarítmica	Búsqueda binaria
$O(N)$	Lineal	Búsqueda secuencial
$O(NlogN)$	Lineal-Logarítmica	Algoritmo de ordenamiento *quicksort*
$O(N^2)$	Cuadrática	Algoritmo de ordenamiento simple
$O(N^3)$	Cúbica	Multiplicación de matrices
$O(2^N)$	Exponencial	Partición de conjuntos

Tabla 1: Ejemplos de algoritmos numéricos con su clasificación O de complejidad

La mayor parte de los algoritmos de factorización elementales son de proposito general, es decir, permiten descomponer cualquier número introducido, la diferencia entre algoritmos es el tiempo que se toman para encontrar la factorización del número dado. El problema de factorizar enteros de tiempo polinómico no ha sido resulto en computación clásica. Esto puede ser de gran ayuda al avance en el ambito de la criptografía, ya que muchos sistemas criptográficos dependen de la imposibilidad de ser resueltos en un tiempo corto. La complejidad de este problema se encuentra en el núcleo de varios sistemas criptográficos importantes. Un algoritmo veloz para la factorización de enteros significaría que el algoritmo de clave pública RSA es inseguro. Si un número grande, de b bits es el producto de dos primos de aproximadamente el mismo tamaño, no existe algoritmo conocido capaz de factorizarlo en tiempo polinómico. Esto significa que ningún algoritmo conocido puede factorizarlo en tiempo O(b^K), para cualquier constante k. Aunque, existen algoritmos que son más rápidos que O(a^b) para cualquier a mayor que 1. En otras palabras, los mejores algoritmos son súper-polinomiales, pero subexponenciales. En particular, el mejor tiempo asintótico de ejecución lo contiene el algoritmo de *criba general del cuerpo de números [2,3] (CGCN)*, que para un número n es:

$$O\left(exp\left[cn^{1/3}\,(logn)^{2/3}\right]\right) \tag{1}$$

Para una computadora ordinaria, la CGCN es el mejor algoritmo conocido para números grandes.

2.3. Origen de la computación cuántica

Al inicio de la decada de 1960, Rolf Landauer se cuestionaba acerca de si las leyes físicas determinaban límites a los procesos de computo. En concreto se interesó por el origen del calor disipado por los ordenadores. Uno de los problemas actuales de los ordenadores de alta velocidad es el calor que se produce durante su funcionamiento. Cada vez se fabrican más microchips más pequeños en el cual se administra una mayor cantidad de transistores, sin embargo, existe un límite en el cual podemos ir disminuyendo el tamaño de cada transistor debido a que los canales donde circulan los electrones pueden escaparse por el fenómeno del efecto tunel, es por ello que la computación digital clásica no debe estar muy lejos del límite.
Las ideas esenciales de la computación cuántica surgieron en los primeros años de la decada 1980, esto de la mente de Paul Beinoff, quien hacia operar computadoras clásicas con operaciones cuánticas. Entre 1981 y 1982, Richard Feymann propusó el uso de fenónemos cuánticos para realizar cálculos computacionales exponiendo la idea que, dada la naturaleza de la mecánica cuántica algunos procesos se realizarian en un menor periodo de tiempo a comparación de una computadora clásica. En 1985, David Deutsch describió el primer computador cuántico universal, este ordenador era capaz de simular cualquier otro computador cuántico por medio de la tesis de Church-Turing ampliado, el cual es un enunciado formalmente indemostrable, no obstante, tiene una aceptación universal. Las tesis de Church y Turing son las siguientes:

Tesis 1 (de Church)**.** *La clase de las funciones que pueden ser calculadas mediante un algoritmo coincide con la clase de las funciones recursivas.*

Tesis 2 (de Turing)**.** *La clase de las funciones que pueden ser calculadas mediante un método definido coincide con la clase de las funciones calculables mediante una Máquina de Turing.*

En base a estos enunciados, surgió la idea de que un computador cuántico podría ejecutar diferentes algoritmos cuánticos. A lo largo de la decada de 1990, la teoría acerca de la computación cuántica empezó a tener sus primeras aplicaciones en forma de algoritmos cuánticos, el primero en realizar estos aportes Dan Simon demostrando la ventaja de usar un computador cuántico frente a uno tradicional al comparar el modelo de probabilidad clásica con el modelo cuántico. Sus ideas sirvieron como base para el desarrollo de algoritmos de auténtico interés práctico. En el año 1993, Charles Benett descubre el tele-transporte cuántico, que abre una nueva vía de investigación dirigida hasta las comunicaciones.
Entre 1994 y 1995 Peter Shor definió el algortimo que permite calcular los factores primos de números a un tiempo menor a cualquier computadora clásica. En el año 2000, IBM diseña un computador cuántico de 5 qubits capaz de ejecutar un algoritmo de búsqueda de orden que forma parte del algoritmo de Shor. El algoritmo de Shor tiene una complejidad algorítmica [4]

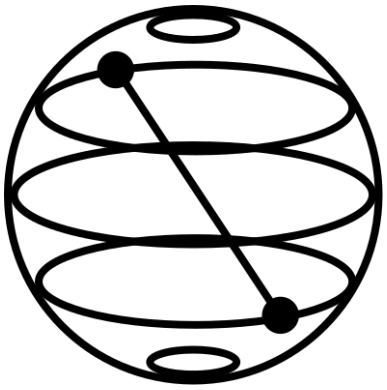
$$O(n^2(logn)(log(logn))) \tag{2}$$

En 2001, nuevamente, IBM en conjunto con la Universidad de Standford, consiguen ejecutar por primera vez el algoritmo de Shor en el primer computador cuántico de 7-qubits, en el experimento se calcularon los factores primos de 15, dando como resultado 3 y 5 utilizando para ello 1018 moléculas, cada una de ellas con 7 átomos.

2.3.1. Qiskit - IBM

En el año 2017, IBM lanzó su primer version de la libreria Qiskit, la cual, es un framework de libre acceso para computación cuántica. El framework te provee de herramientas para la creación y manipulación de programas cuánticos y correrlos en los prototipos de dispositivos cuánticos llamados *IBM Q Experience*. Este framework esta desarrollado en el lenguaje *Python*.

(a)



(b)

```
In [7]: from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
        from qiskit.tools.visualization import circuit_drawer
        import numpy as np

        qr = QuantumRegister(2)
        cr = ClassicalRegister(2)
        qp = QuantumCircuit(qr,cr)

        qp.rx( np.pi/2,qr[0])
        qp.cx(qr[0],qr[1])

        qp.measure(qr,cr)

        circuit_drawer(qp)
```

Out[7]:

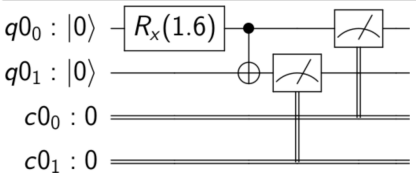


Figura 1: (a) Logo de Qiskit, (b) Ejemplo de un circuito cuántico implementado dentro de Jupyter Notebook de Python.

3. Objetivo

- Desarrollar el algoritmo de Shor para la factorización de un número dado usando la librería Qiskit en el lenguaje python.
- Calcular la diferencia de tiempo de procesamiento para un número N entre el algoritmo de criba general de cuerpo de números usando una computadora clásica, el algoritmo de Shor usando una computadora clásica y el algortimo de Shor usando una computadora cuántica.

4. Marco teórico

4.1. Estados cuánticos y el término Qubit

En física cuántica, el estado cuántico es cualquier estado posible en el que puede estar un sistema mecánico cuántico. Un estado cuántico completamente especificado puede describirse mediante un vector de estado, una función de onda o un conjunto completo de números cuánticos para un sistema específico. Un estado cuántico parcialmente conocido, como un conjunto con algunos números cuánticos fijos, puede ser descrito por un operador de densidad.

Una mezcla de estados cuánticos es nuevamente un estado cuántico. Los estados cuánticos que no se pueden escribir como una mezcla de otros estados se denominan estados cuánticos puros, mientras que todos los demás estados se denominan estados cuánticos mixtos. Un estado cuántico puro se puede representar

mediante un rayo en un espacio de Hilbert sobre los números complejos, mientras que los estados mixtos se representan mediante matrices de densidad, que son operadores semidefinitos positivos que actúan en el espacio de Hilbert.

Un bit cuántico, qbit o qubit es la unidad básica de información cuántica, es la versión cuántica del clásico bit binario.

Los dos estados básicos de un qbit son $|0\rangle$ y $|1\rangle$, que corresponden al 0 y 1 del bit clásico. Pero además, el qbit puede encontrarse en un estado de superposición cuántica combinación de esos dos estados ($\alpha|0\rangle + \beta|1\rangle$). En esto es significativamente distinto al estado de un bit clásico, que puede tomar solamente los valores 0 o 1.

Los algoritmos cuánticos que operan sobre estados de superposición realizan simultáneamente las operaciones sobre todas las combinaciones de las entradas. Por ejemplo, los dos qbits

$$\frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle + |1\rangle) = \frac{1}{2}(|0\rangle|0\rangle + |0\rangle|1\rangle + |1\rangle|0\rangle + |1\rangle|1\rangle)$$

representan simultáneamente las combinaciones 00, 01, 10 y 11. En este "paralelismo cuántico"se cifra la potencia del cómputo cuántico.

Una característica importante que distingue al qbit del bit clásico es que múltiples qbits pueden presentarse en un estado de entrelazamiento cuántico. En el estado no entrelazado

$$\frac{1}{2}\left(|0\rangle|0\rangle + |0\rangle|1\rangle + |1\rangle|0\rangle + |1\rangle\,|1\rangle\right)$$

pueden darse las cuatro posibilidades: que la medida del primer cúbit dé 0 o 1 y que la medida del segundo cúbit dé 0 o 1. Esto es posible porque los dos cúbits de la combinación son separables (factorizables), pues la expresión anterior puede escribirse como el producto

$$(|0\rangle + |1\rangle) \times (|0\rangle + |1\rangle)$$

Por convención existe que los estados cuánticos descritos por números binarios pueden ser escritos como un número en base decimal, como por ejemplo:

$$|1100\rangle \rightarrow |12\rangle$$

4.2. Compuertas lógicas clásicas y cuánticas

El álgebra de Boole es la teoría matemática que se aplica en una lógica combinatoria. Las variables booleanas son símbolos utilizados para representar acciones lógicas, las cuales solo pueden dar como resultado dos valores, 0 y 1. Las operaciones booleanas son posibles a través de los operadores binarios como negación, suma y resta, es por ello que estas operaciones son utiles para realizar operaciones anteriormente mencionadas.

Las operaciones booleanas no necesariamente tienen una transformación inversa, es por ello que se pierde información del estado antes de aplicar una transformación booleana. En el caso cuántico son conceptualmente análogas a las clásicas, éstas reciben un registro de qubits en un determinado estado y aplican una operación sobre el mismo para transformarlo, la aplicación de estas compuertas cuánticas no colapsa los estados y se permite realizar una serie de operaciones sobre el mismo estado del qubit. La representación de estas compuertas son de matrices cuadradas y unitarias. Además, a diferencia de la mayoría de compuertas lógicas clásicas, las compuertas cúanticas sí contemplan la existencia de su transformación inversa [5, 6].

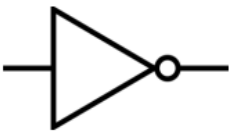


Figura 2: Representación gráfica de la compuerta NOT.

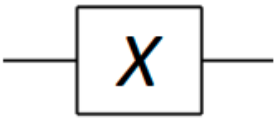


Figura 3: Representación gráfica de la compuerta cuántica NOT.

A	NOT(A)
0	1
1	0

Tabla 2: Compuerta clásica NOT.

A	NOT(A)
$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$
$\|\psi\rangle$	$\hat{X}\|\psi\rangle$

Tabla 3: Compuerta cuántica NOT.

4.2.1. Compuerta de Hadamard

La compuerta de Hadamard opera únicamente sobre un qubit, y realiza la operación de un cambio de base, de pasar de la base $\{|0\rangle, |1\rangle\}$ a la base $\{|+\rangle, |-\rangle\}$

$$|0\rangle \underset{\hat{H}}{\rightarrow} \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv |+\rangle$$
$$|1\rangle \underset{\hat{H}}{\rightarrow} \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv |-\rangle$$

donde:

$$\hat{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Esta compuerta es usada comunmente para generar superposiciones a partir de los estados base computacional, esta se puede observar que crea una rotación de π alrededor del eje XZ.

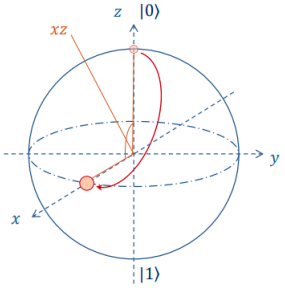


Figura 4: Representación gráfica del efecto de la transformación de Hadamard aplicada a un qubit.

4.2.2. Compuerta de desplazamiento de fase

La compuerta de desplazamiento de fase aplica unicamente a un qubit de tal forma que al estado $|0\rangle$ lo deja intacto, pero al estado $|1\rangle$, lo lleva a $e^{i\theta}|0\rangle$. Por lo que la probabilidad de medir cualquier estado no cambia al aplicar esta compuerta, sin embargo, sí modifican la fase del estado cuántico.

$$|0\rangle \underset{\hat{R}(\phi)}{\rightarrow} |0\rangle$$
$$|1\rangle \underset{\hat{R}(\phi)}{\rightarrow} e^{i\phi}|1\rangle$$

donde:

$$\hat{R}(\phi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

por lo que ϕ es el desplazamiento. Algunos casos comunes de estas puertas son la puerta $\frac{\pi}{8}$ donde $\phi = \frac{\pi}{4}$, la puerta de fase donde $\phi = \frac{\pi}{2}$ y la puerta de Pauli-Z donde $\phi = \pi$.

4.2.3. Compuerta SWAP

La compuerta SWAP es una operación de dos qubit. Expresado en estados base $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. La compuerta SWAP intercambia el estado de los dos qubits involucrados en la operación. Esta representada por la matriz:

$$U_{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

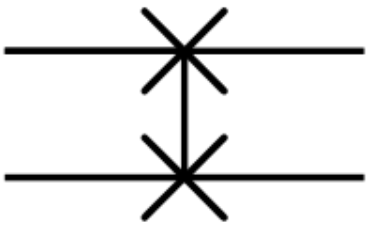


Figura 5: Representación gráfica de la compuerta SWAP.

4.2.4. Compuertas Controladas

Las puertas controladas operan sobre 2 qubits o más, de los cuales uno o más controlan la operación. Por ejemplo, la puerta NOT controlada (o CNOT) opera sobre 2 qubits, y realiza la operación NOT en el segundo qubit solo cuando el primer qubit es $|1\rangle$, y en otro caso lo deja intacto. Se representa por la matriz

$$U_{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

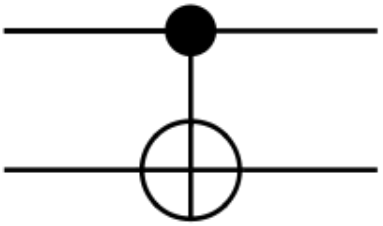


Figura 6: Representación gráfica de la compuerta NOT controlada.

4.3. Algoritmo de Shor

La razones por las cuales el algoritmo de Shor es más eficiente (ecuación 2) que el algortimo de criba general del cuerpo de números (ecuación 1) son las siguientes:

- Transformada de Fourier Cuántica.
- Periodo de la función $a^x \bmod N$.

4.3.1. Transformada de Fourier Cuántica (QFT)

La Transformada de Fourier Cuántica es una forma efectiva para realizar un cambio de base, este lo realiza de la base computacional $\{|0\rangle, |1\rangle\}$ hacia la base de Fourier $\{|\tilde{0}\rangle, |\tilde{1}\rangle\}$.
Si se tienen n qubits, entonces existiran 2^n bases de estados. Definiendo a N como $N \equiv 2^n$, entonces:

$$\begin{aligned} |\tilde{x}\rangle &\equiv QFT\,|x\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}} |y\rangle \end{aligned}$$

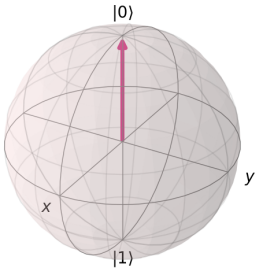
Para el caso de un quibit [7,8] , se tendria lo siguiente:

$$\begin{aligned} |\tilde{0}\rangle &= \frac{1}{\sqrt{2}} \sum_{y=0}^{1} e^{\frac{2\pi i (0) y}{2}} |y\rangle \\ &= \frac{1}{\sqrt{2}} \sum_{y=0}^{1} |y\rangle \\ &= \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle\right) \end{aligned}$$

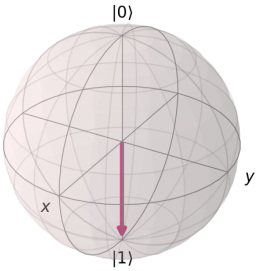
$$\begin{aligned} |\tilde{1}\rangle &= \frac{1}{\sqrt{2}} \sum_{y=0}^{1} e^{\frac{2\pi i (1) y}{2}} |y\rangle \\ &= \frac{1}{\sqrt{2}} \left[e^{\frac{2\pi i (0)}{2}} |0\rangle + e^{\frac{2\pi i 1}{2}} |1\rangle \right] \\ &= \frac{1}{\sqrt{2}} \left[|0\rangle - |1\rangle\right]. \end{aligned}$$

Con lo cual observamos que el estado $|1\rangle$ al final queda multiplicado por un factor diferente a $1/\sqrt{2}$, extendiendo esta transformación para k qubits, se tiene lo siguiente:

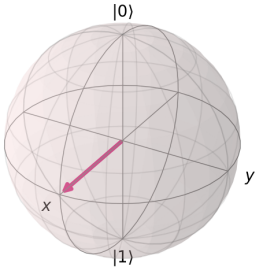
$$
\begin{aligned}
|\tilde{x}\rangle &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}} |y\rangle \,. \\
&= \frac{1}{\sqrt{N}} \sum_{y_1=0}^{1} \sum_{y_2=0}^{1} \cdots \sum_{y_k=0}^{1} exp\left[\frac{2\pi i x}{N} \sum_{k=1}^{n} y_k 2^{n-k}\right] |y_k\rangle \,. \\
&= \frac{1}{\sqrt{N}} \sum_{y_1, y_2, \cdots, y_k} \bigotimes_{k=1}^{n} e^{\frac{2\pi i x}{N} y_k} |y_k\rangle \,. \\
&= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} \left[|0\rangle + e^{\frac{2\pi i x}{N} 2^{n-k}} |1\rangle\right] . \\
&= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{N} \left[|0\rangle + e^{2\pi i x 2^{-k}} |1\rangle\right] .
\end{aligned}
$$



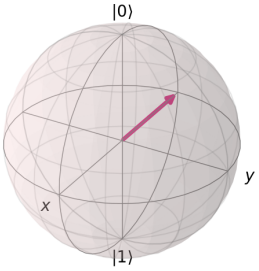
(a) Qubit con el estado cuántico $|0\rangle$ representado en la esfera de Bloch.



(b) Qubit con el estado cuántico $|1\rangle$ representado en la esfera de Bloch.



(c) Qubit con el estado cuántico $|\tilde{0}\rangle$ representado en la esfera de Bloch.



(d) Qubit con el estado cuántico $|\tilde{1}\rangle$ representado en la esfera de Bloch.

Figura 7: Representación en la esfera de Bloch de los estados bases computacionales cuánticos y las bases de Fourier.

Para obtener la transformada de Fourier en base a los circuitos cuánticos, tenemos que realizar una serie de operaciones con la compuerta de Hadamard y la compuerta de rotaciones, en donde la fase de cada rotación es $exp(2\pi i/2^k)$ [9]. La

transformada de Fourier Cuática por si sola no nos ayuda en mucho, es por ello que se propuso un proceso en el cual se pueda medir en que fase se encuentra el estado cuántico, esto lo lleva el proeso de estimación de fase cuántico (QPE).

4.3.2. Estimación de Fase Cuántica (QPE).

Al tener aplicada la transformada de Fourier Cuántica al estado de los qubits este contiene información en sus eigenvalores, ya que tienen la forma de $e^{i\theta}$ y esto es que cada eigenvector forma una base ortonormal [10], es por ello que se tiene lo siguiente:

$$\mathcal{U}\left|\psi\right\rangle = e^{i\theta_\psi}\left|\psi\right\rangle.$$

Y el proposito de este proceso es obtener la fase θ_ψ dado del estado $|\psi\rangle$. Este mismo proceso se puede realizar usando el periodo de una función modular, la cual es $f(x) = a^x mod N$, la cual nos puede dar información acerca de la periodicidad de los residuos en una disivisón de valores.

4.3.3. Periodo de la función a^xmod N

Se tiene la función

$$f(x) = a^x mod N \tag{3}$$

donde a y N son enteros positivos tal que $a < N$ y que no tienen ningún factor en común. El periodo (o el orden r), es el valor más pequeño diferente de cero tal que:

$$a^r mod N = 1 \tag{4}$$

Usando como ejemplos $a = 3$ y $N = 35$, se tiene el siguiente periodo:

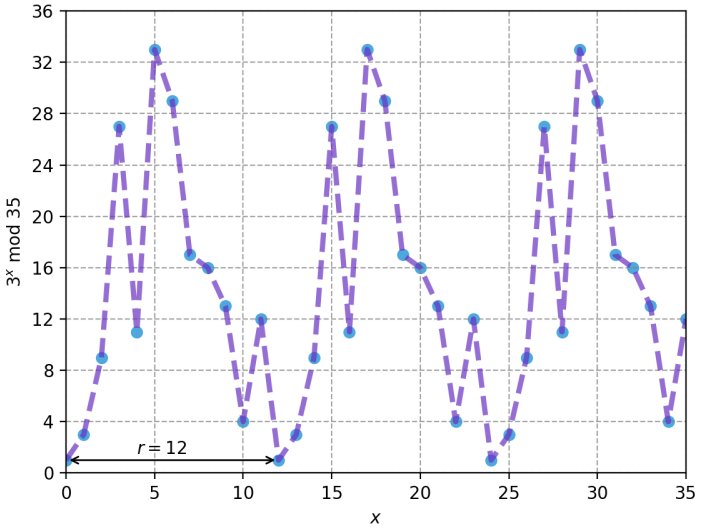


Figura 8: Periodo de la función 3 para visualizar la condición 4 usando el código 3

Para realizar esta operación dentro del Algoritmo de Shor se propusó un operador unitario tal que:

$$\mathcal{U}\left|y\right\rangle \equiv \left|aymod N\right\rangle \tag{5}$$

Para visualizar el funcionamiento de este operador se usará el ejemplo de $a = 3$ y $N = 35$, trabajando con los eigenestados de U empezando con el estado $|1\rangle$ podemos visualizar que la transformación sucesiva del operador U estariamos multiplicando el estado por $amodN$, por lo tanto, después de r transformaciones regresaremos al estado $|1\rangle$.

$$\begin{aligned}
\mathcal{U}|1\rangle &= |3\rangle \\
\mathcal{U}^2|1\rangle &= |9\rangle \\
\mathcal{U}^3|1\rangle &= |12\rangle \\
&\vdots \\
\mathcal{U}^{r-1}|1\rangle &= |12\rangle \\
\mathcal{U}^{r}|1\rangle &= |1\rangle
\end{aligned}$$

Utilizando una superposición de estos estados, obtendremos lo siguiente:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left| a^k mod N \right\rangle . \tag{6}$$

Realizando esa serie para $a = 3$ y $N = 35$, encontramos que el estado descrito en 6 es invariante bajo la transformación de U.

$$\begin{aligned}
|u_0\rangle &= \frac{1}{\sqrt{12}} \left(|1\rangle + |3\rangle + |9\rangle + \cdots + |4\rangle + |12\rangle \right) \\
U|u_0\rangle &= \frac{1}{\sqrt{12}} \left(U|1\rangle + U|3\rangle + U|9\rangle + \cdots + U|4\rangle + U|12\rangle \right) \\
&= \frac{1}{\sqrt{12}} \left(|3\rangle + |9\rangle + |27\rangle + \cdots + |12\rangle + |1\rangle \right) \\
&= |u_0\rangle .
\end{aligned}$$

Lo cual obtenemos que el eigenvalor es 1. Un eigenestado más general es aquel el cual contenga una fase diferente para cada base de estados. Especificamente, veremos el caso donde cada fase del k-esimo es proporcional a k, de tal manera que:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} \left| a^k mod N \right\rangle \tag{7}$$

$$U|u_1\rangle = e^{\frac{2\pi i}{r}} |u_1\rangle . \tag{8}$$

Teniendo así un único eigenestado para cada valor entero en s donde $0 \leq s \leq r-1$. Esto es muy conveniente, ya que realizando la suma de los eigenestados, las diferencias de fases serán canceladas, obteniendo así el estado $|1\rangle$.

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle .$$

Siendo así que el estado $|1\rangle$ es una superposición de estos eigenestados, lo cual indica que podemos realizar el algoritmo de estimación de fase (QPE) [11, 12] al operador U usando el estado $|1\rangle$, para así obtener la medición de la fase:

$$\phi = \frac{s}{r}$$

Realizando del algoritmo de fracciones continuas sobre ϕ podemos encontrar el valor de r. El circuito cuántico que realiza esta tarea es el siguiente:

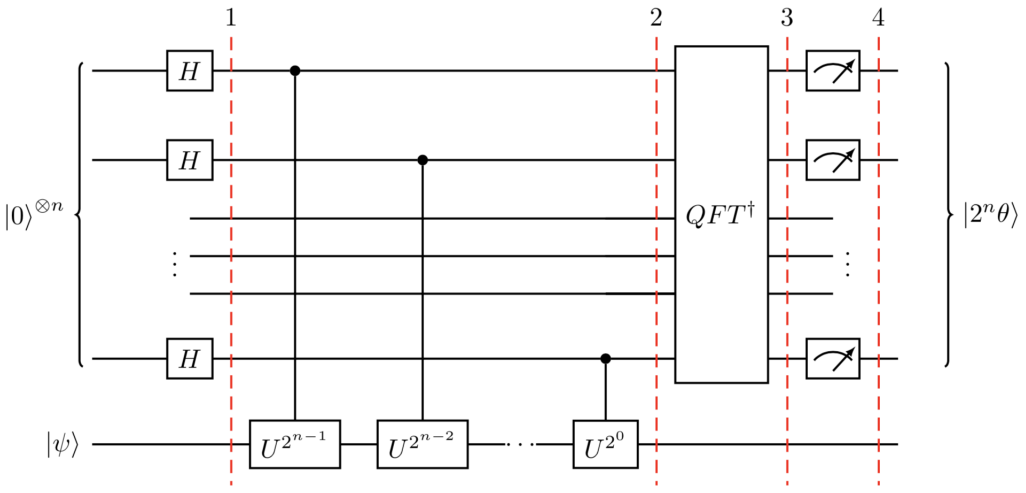


Figura 9: Circuito cuántico que realiza el algortimo QPE (Quantum Phase Estimation) sobre una serie de qubits.

En la figura 9 se logra apreciar que existen varios pasos en el algoritmo los cuales son los siguientes:

1. Los estados iniciales $|0\rangle^{\otimes n}$ se les aplica la compuerta Hadamard a cada uno de ellos para lograr una superposición de ellos mismos.
2. A cada estado se le aplica la compuerta de desplazamiento de fase para así obtener que se acerquen al estado en donde existe una periodicidad en la ecuación 4.
3. Se aplica una transformada de Fourier inversa para obtener el estado cuántico después de haber sido rotados.
4. Se realiza la medición de estos estados para obtener una estadistica y así llevar a conclusones acerca de las fases de los estados cuánticos.

4.4. Implementación con Qiskit

Para mostrar de forma detallada se trabajara con el problema cuando $a = 7$ y $N = 15$. Con esto, realizamos el operador U aplicado al estado $|y\rangle$ de la siguiente manera:

$$U|y\rangle = |aymod15\rangle \tag{9}$$

Para crear las operaciones acumulativas U^x, se creará una repetición del circuito x veces, por lo que la función *c_amod15* regresará la compuerta U al valor de a

y la añadirá al circuito cuántico con la función *modular_exponentiation*, repetida x veces como lo siguiente:

[]:
```
    def c_amod15(a, x):
        if a not in [2,7,8,11,13]:
            raise ValueError("'a' debe ser 2,7,8,11 o 13")
        U = QuantumCircuit(4)
        for iteration in range(x):
            if a in [2,13]:
                U.swap(0,1)
                U.swap(1,2)
                U.swap(2,3)
            if a in [7,8]:
                U.swap(2,3)
                U.swap(1,2)
                U.swap(0,1)
            if a == 11:
                U.swap(1,3)
                U.swap(0,2)
            if a in [7,11,13]:
                for q in range(4):
                    U.x(q)
        U = U.to_gate()
        U.name = "%i^%i mod 15" % (a, x)
        c_U = U.control()
        return c_U
    def modular_exponentiation(given_circuit, n, m, a):
        for x in range(n):
            exponent = 2**x
            given_circuit.append(a_x_mod15(a, exponent),
                                 [x] + list(range(n, n+m)))
```

Al algortimo también se le implemento la transformación inversa de Fourier cuántica, se le nombre como *qft_dagger*, esta función es la siguiente:

[]:
```
    def qft_dagger(n):
        qc = QuantumCircuit(n)
        for qubit in range(n//2):
            qc.swap(qubit, n-qubit-1)
        for j in range(n):
            for m in range(j):
                qc.cu1(-np.pi/float(2**(j-m)), m, j)
            qc.h(j)
        qc.name = "iqft"
        return qc
```

Ya con estas funciones preestablecidas, el algoritmo de Shor es construido de una manera amigable a la vista, este es el siguiente:

[]:
```
    def shor_program(n, m, a):
        # set up quantum circuit
        shor = QuantumCircuit(n+m, n)
```

```
        # initialize the qubits
        initialize_qubits(shor, n, m)
        shor.barrier()
        # apply modular exponentiation
        modular_exponentiation(shor, n, m, a)
        shor.barrier()
        # apply inverse QFT
        apply_iqft(shor, range(n))
        # measure the first n qubits
        shor.measure(range(n), range(n))
        return shor
    n = 4; m = 4; a = 7
    mycircuit = shor_program(n, m, a)
    mycircuit.draw(output='text')
```

Al termino de estas líneas de código tendremos la serie de posibles factores primos del número 15, por lo que con ayuda de un algoritmo clásico se podra comprobar si los cálculos antes realizados son correctos, este algoritmo es el siguiente:

[]:
```
    from math import gcd

    for measured_value in counts:
        measured_value_decimal = int(measured_value[::-1], 2)
        print(f"Medición {measured_value_decimal}")

        if measured_value_decimal % 2 != 0:
            print("Fallo. El número no es par")
            continue
        x = int((a ** (measured_value_decimal/2)) % 15)
        if (x + 1) % 15 == 0:
            print("Fallo. x + 1 = 0 (mod N) donde x = a^(r/2) (mod␣
↪N)")
            continue
        guesses = gcd(x + 1, 15), gcd(x - 1, 15)
        print(guesses)
```

Dando como resultado lo siguiente:

```
Medición 0
(1, 15)
Medición 8
(1, 15)
Medición 4
(5, 3)
Medición 12
(5, 3)
```

En donde se puede observar que no necesariamente el algoritmo cuántico encontrará la factorización que nosotros esperaríamos, si no que necesita ayuda de una computadora clásica para comprobar sus resultados. Este circuito esta hecho Especificamente para factorizar el número 15, el circuito cuántico que factoriza cualquier número N es el siguiente:

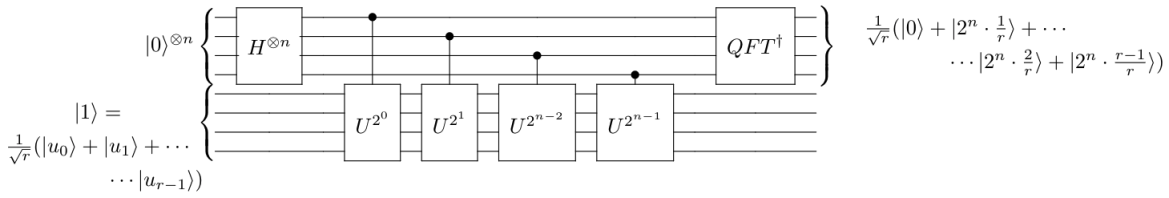


Figura 10: Algoritmo de Shor para la factorización de un número visto desde la estructura que propone Qiskit.

5. Resultados

Al ya tener definido el algortimo de Shor, se comenzo a desarrollar un programa que funcione para cualquier número dado usando la libreria de Qiskit (código 1). Este fue subido a la plataforma *IBM Q Experience* con un formato de Jupyter Notebook, ya que este es el que recibe para poder ejecutar el código. Se creo otro código el cual sigue los lineamientos del algoritmo de criba general del cuerpo de números. (codigo 5), los dos códigos generan archivos de texto que contienen el tiempo de procesamiento de cada algoritmo.

Algortimo	Número a factorizar	IBM Device	Lanzamientos	Repeticiones
CCGN	21	-	-	1000
Shor (Local)		Vigo	1000	
Shor (IBM Q)				

Tabla 4: Parámetros de entrada de cada algoritmo de factorización.

Usando los parámetros de entrada de la tabla 4 en cada algoritmo de los códigos 1 y 5 se obtuvieron los tiempos de procesamiento y al estar factorizando el mismo número repetidas veces se pueden obtener los tiempos promedios en los cuales se tarda cada algoritmo en su diferente situación. Estos resultados son los mostrados en la tabla 5.

Algoritmo	Promedio (ms)	σ (ms)
Clasico	0.056	0.0096
Shor	0.036	0.0153
Shor IBM	0.024	0.0102

Tabla 5: Promedio y desviación estandar de cada algoritmo de factotización ejecutados en una computadora clásica y cuántica.

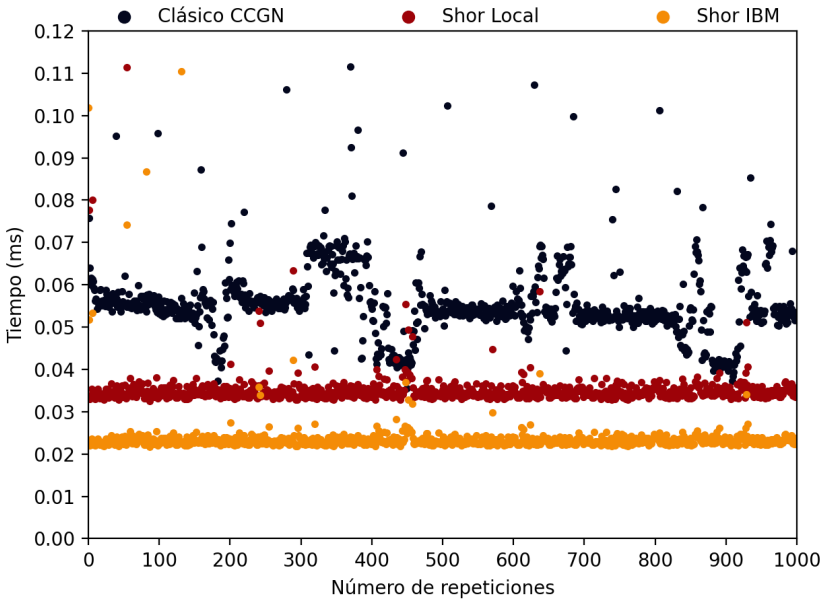


Figura 11: Comparación de los tiempos de procesamiento de los códigos 1 y 5, el código 1 fue ejecutado en una computadora clásica y en una computadora cuántica.

Con estos mismos resultados se generó la figura 11, en donde se muestra el tiempo de procesamiento de cada algoritmo en cada repetición que se realizó.

6. Conclusiones y discusión

Visualizando la figura 11 se aprecia como el algoritmo clásico es más inestable en cuanto al tiempo que tarda en lograr la factorización del número a comparación del algoritmo de Shor el cual mantiene tiempos semejenates durante cada repetición, esto puede ser debido a que el algoritmo de Shor maneja una serie de pasos como el cálculo del periodo de a función $a^x \text{mod } N$.
Lo más facil de ver es el tiempo medio que tarda cada algoritmo en obtener el resultado, esto es mostrado en la tabla 5, en este caso el algoritmo de Shor ejecutado en una computadora cuántica es el más rápido en promedio, las diferencias entre al algoritmo de Shor ejecutado de en una computadora clásica y ejecutado en una computadora cuántica son debido a que en la clásica se tiene que simular cada estado cuántico, en cambio en la cuántica este estado es preparado y medido experimentalmente.

Un problema que nos encontramos al momento de ejecutar los códigos del algoritmo de Shor (código 1) dentro del sistema de *IBM Q Experience* fue el número que podiamos introducir en el código, ya que el dispositivo que ellos tienen es de 5 qbits, por lo que el número más grande que se puede introducir al día de hoy es el 31, es por ello que aun no se puede comprobar experimentalmente la eficiencia de los algoritmos para números muy grandes, al menos de manera al publico en general pero con estos resultados podemos observar una mayor eficiencia y

estabilidad en los calculos que se realicen dentro de una computadora cuántica siguiendo conceptos y principios cuánticos.

Figura 12: Lista de dispositivos disponibles al público desde *IBM Q Experience*

7. Código

1. Shor_quantum.py
   Este algoritmo realiza la factorización de un número N en dos números primos, este código es capaz de correr en una computadora clásica y en el propotito de IBM Q Experience

2. Functions.py
   Esta serie de funciones son usadas en el programa *Shor_quantum.py*.

3. r_period.py
   Este código realiza el calculo del periodo r dados a y N, da como resultado la figura 8.

4. bloch_graphics.py
   Este código genera las figuras mostradas en 7 usando la representación de Qiskit.

5. Time_clasic.py
   Este código calcula la factorización de un número dado de manera clásica 1000 veces y va tomando el tiempo que tarda en cada uno guardandolo en un archivo de texto.

6. Time_graphics.py
   Este código crea la figura 11 a partir de los archivos de tiempo obtenidos con los códigos 1 y 5.

Referencias

[1] L. S. Cohen and T. Wendling, "Técnicas de diseño," *Técnicas de diseño*, pp. 15–18, 1998.

[2] Agrios, *Introducción a La Introducción a La Teoría de Números.*, vol. 3. 2003.

[3] D. E. I. D. E. Telecomunicación, "Números primos especiales y sus aplicaciones criptográficas," 2003.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[5] V. M. Bonillo, "Principios Fundamentales De Computación Cuántica," *univerasidad de A Coruña*, pp. 1–181, 2013.

[6] D. Koch, S. Patel, L. Wessing, and P. M. Alsing, "Fundamentals In Quantum Algorithms: A Tutorial Series Using Qiskit Continued," 2020.

[7] S. S. Zhou, T. Loke, J. A. Izaac, and J. B. Wang, "Quantum Fourier transform in computational basis," *Quantum Information Processing*, vol. 16, no. 3, pp. 1–19, 2017.

[8] S. J. Lomonaco and L. H. Kauffman, "A continuous variable Shor algorithm," pp. 97–108, 2005.

[9] L. M. Vandersypen, M. Breyta, G. Steffen, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.

[10] G. P. Berman, G. D. Doolen, G. V. López, and V. I. Tsifrinovich, "Nonresonant effects in the implementation of the quantum Shor algorithm," *Physical Review A - Atomic, Molecular, and Optical Physics*, vol. 61, no. 4, p. 7, 2000.

[11] A. Yimsiriwattana and S. J. Lomonaco Jr., "Distributed quantum computing: a distributed Shor algorithm," *Quantum Information and Computation II*, vol. 5436, p. 360, 2004.

[12] E. Gerjuoy, "Shor's factoring algorithm and modern cryptography. An illustration of the capabilities inherent in quantum computers," *American Journal of Physics*, vol. 73, no. 6, pp. 521–540, 2005.