DISEÑO DE ALGORITMOS Y CIRCUITOS CUÁNTICOS

DESIGN OF QUANTUM ALGORITHMS AND CIRCUITS

Jonathan José Jiménez Jiménez, INGENIERÍA INFORMÁTICA.
Amalia Regueira Fernández, INGENIERÍA INFORMÁTICA.
Jairo Joel Sánchez Vélez, INGENIERÍA DEL SOFTWARE.

Trabajo de fin de Grado en Ingeniería Informática.
Facultad de Informática. Universidad Complutense de Madrid.

Tutores:
Alberto Antonio del Barrio García.
Guillermo Botella Juan.

Índice

Índice de figuras

1. Resumen.

Últimamente se están haciendo muchos avances en el campo de la computación cuántica, algunas grandes empresas como Google, IBM o Microsoft han comenzado la carrera por el control de la computación cuántica poniendo a disposición de los usuarios las herramientas para desarrollar circuitos y algoritmos como Cirq o Qiskit. Es por esta razón que en este trabajo se explican los conceptos básicos y la implementación de algoritmos cuánticos, en concreto, el algoritmo de Shor, el algoritmo de Deutsch-Jozsa, el algoritmo Bernstein-Vazirani, y el algoritmo de Grover.

Palabras clave:

Computación cuántica, qubit, algoritmos cuánticos, algoritmo de Shor, algoritmo de Deutsch-Jozsa, algoritmo Bernstein-Vazirani, algoritmo de Grover.

2. Abstract.

Much progress has recently been made in the field of quantum computing. Some big companies, such as Google, IBM, or Microsoft, have started the fight to control the quantum computer market and have provided users with the tools to develop schemes and algorithms such as Cirq or Qiskit. For this reason, this project explains the basic concepts and applications of quantum algorithms, as the Shor algorithm, the Deutsch-Jozsa algorithm, Bernstein-Vazirani algorithm and Grover algorithm.

Keywords:

Quantum computing, qubit, quantum algorithms, Shor algorithm, Deutsch-Jozsa algorithm, Bernstein-Vazirani algorithm, Grover algorithm.

3. Introducción.

En la década de los 80, Richard Feyman presenta cómo un sistema cuántico puede ser utilizado para aumentar el rendimiento computacional clásico [1]. Un ordenador cuántico, es una maquina que funciona según las leyes de la mecánica cuántica como la superposición y el entrelazamiento. Gracias a este tipo de funcionamiento, es capaz de llevar a cabo múltiples tareas simultáneamente.

Desde entonces, se han producido avances en este campo, aunque la mayoría solo teóricamente. Estos avances vaticinan aplicaciones prometedoras para los ordenadores cuánticos.

En la actualidad, la computación cuántica se asemeja con el nacimiento de los ordenadores clásicos que funcionaban con transistores y eran tan grandes como una habitación. Se está progresando lentamente, pero todavía se encuentran problemas difíciles de solucionar como el aumento de qubits, la necesidad de llevar al sistema a temperaturas cercanas al cero absoluto, la lectura de información o la decoherencia cuántica, es decir, los estados cuánticos solo se mantienen durante un corto periodo de tiempo [2].

3.1. Objetivos.

Los objetivos en este trabajo son:

- Entender y explicar los conceptos básicos de la computación cuántica.
- Familiarizarse con el lenguaje de programación de Qiskit, usado para las simulaciones de este trabajo.
- Implementar los algoritmos de Deutsch-Jozsa, Bernstein-Vazirani, Shor y Grover. Usando entornos de simulación y computadores cuánticos reales.

4. Conceptos básicos.

4.1. Qubit.

Un qubit (quantum bit) es a la computación cuántica lo que el bit a la clásica, es decir, es su unidad mínima de información. Mientras que el bit

clásico solo puede tener valor 0 o 1, el qubit puede estar en el estado 0, 1 o en una combinación de ambos.

Para representar un qubit tenemos diversas formas de hacerlo, como ya se verá más adelante.

4.1.1. Visualización como vector dentro de una esfera de radio 1. (Esfera de Bloch):

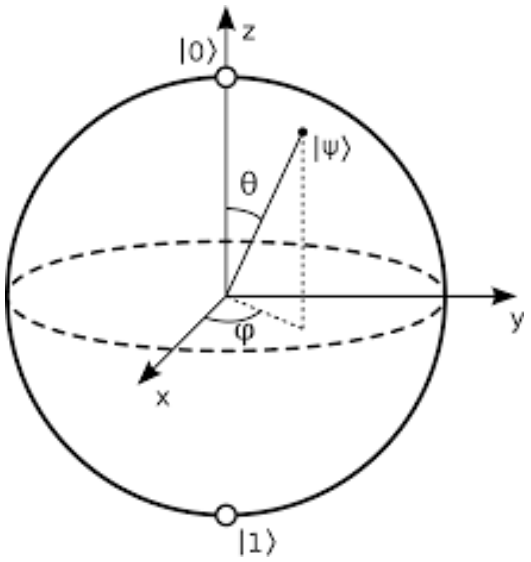

Figura 1: Esfera de Bloch.

De esta forma tenemos el vector (1,0) apuntando al polo norte de la esfera, que representa el estado $|0\rangle$, equivalente al bit 0. Y el vector (0,1) apuntado al polo sur, que representa el estado $|1\rangle$, equivalente al bit 1. De esta forma más visual, se trabaja rotando el vector a través de las tres dimensiones (figura 1).

4.1.2. Notación matemática o vectores ket:

El estado cuántico se puede expresar como la combinación lineal de todos los posibles estados que se podrían formar con el mismo número de bits.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,.$$

Siendo $\alpha^2 + \beta^2 = 1$, donde α^2 es la probabilidad de que el qubit valga 0 y β^2 es la probabilidad de que valga 1.

Cambiando los valores de α y β, el qubit puede tener infinitos valores, que son vectores unitarios dentro del espacio vectorial en los números complejos.

4.1.3. Notación matricial:

Otra posible representación se puede formar creando un matriz compuesta por una columna y 2^n filas. Todas las filas tendrán el valor 0, excepto la fila m con valor 1 para representar el estado m.

Por ejemplo, para representar el valor 0, la fila 0 ha de tener el valor 1. Y todas las demás filas, el valor 0.

$$\alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\end{pmatrix}.$$

4.2. Superposición.

La **superposición** es el estado de un objeto cuando posee simultáneamente dos o más valores, de forma que no se puede conocer con exactitud en que estado se encuentra. Para entenderlo de una forma sencilla imaginemos una moneda girando, mientras esté girando la moneda tiene dos posibles valores coexistiendo, cara y cruz. Una vez la moneda se detiene, podemos observar uno de los dos valores a los que ha colapsado la superposición.

La superposición cuántica se asemeja a la superposición de ondas, cuando las amplitudes de dos ondas llegan juntas, en fase, sus intensidades se intensifican, pero cuando llegan opuestas, se cancelan. Mediante el experimento de la doble rendija [3], al hacer pasar un electrón por cada rendija se debería obtener dos líneas, correspondientes a cada rendija. Pero lo que se obtiene es un patrón similar al de hacer pasar una onda, por tanto el electrón pasa por las dos rendijas a la vez.

Cuando se hace una medición de varios estados en superposición se produce el fenómeno de colapso, es decir, como su propio nombre indica 'colapsa' en un único estado aleatoriamente. Debido a esto se pierde la propiedad de superposición. Cabe mencionar que, debido a la naturaleza de la superposición es imposible conocer el valor de la medida con exactitud.

Veamos un ejemplo sobre cómo se aplica la superposición a un qubit:

$$|x\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}.$$

Así, el qubit $|x\rangle$ tiene los valores 1 y 0 al mismo tiempo. Y la misma probabilidad de medir el valor 1 o el 0.

4.3. Paralelismo cuántico.

El **paralelismo cuántico** se describe como la capacidad para evaluar una función $f(x)$ sobre varios valores de forma simultánea gracias a la superposición.

El concepto de paralelismo cuántico es una característica fundamental en el desarrollo e implementación de algunos algoritmos cuánticos, en particular en el algoritmo de Deutsch-Jozsa [16], que se desarrollará más adelante, y que se vale de esta característica cuántica para mejorar el comportamiento respecto al mismo problema tratado en computación clásica.

Supongamos el estado de dos qubits $|xy\rangle$, con $x = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$, en superposición e $y = |0\rangle$. También supongamos una transformación unitaria $U_f = |x, y \oplus f(x)\rangle$ de la forma de la figura 2.

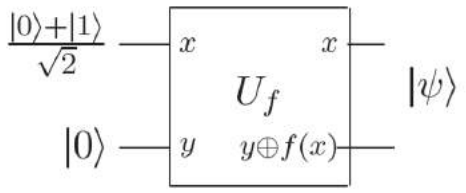

Figura 2: Circuito donde se produce paralelismo cuántico.

Aplicando la transformación tenemos:

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}.$$

Observamos que evaluamos la función a los dos posibles valores simultáneamente, y gracias a que el qubit y de $|xy\rangle$ vale cero, el valor resultante será $0 \oplus f(x) = f(x)$.

4.4. Entrelazamiento cuántico.

El **entrelazamiento cuántico**, es un fenómeno físico que se produce cuando dos o más partículas, interactúan de forma que el estado cuántico de cada una de ellas no se puede describir independientemente de las otras. Esto provoca que cuando una de estas partículas que se encuentra en entrelazamiento sufre una variación, las demás partículas sufran una variación similar.

Aunque nuestros qubits se encuentren en un estado de superposición, medir uno nos dirá el estado de los demás y colapsara su superposición.

Para entenderlo mejor, tomemos el siguiente ejemplo de un estado en entrelazamiento:

$$\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle).$$

Si medimos el primer qubit, se obtiene el valor 0 o 1 con una probabilidad del 50 % en ambos casos. Asumamos que se ha obtenido el valor 0, en consecuencia el valor del segundo qubit es 1. En el otro caso, si el valor obtenido es 1, entonces el valor del segundo qubit es 0.

Gracias a la superposición y el entrelazamiento, el qubit contiene mucha más información que el bit, por lo que si tenemos n bits, solo tenemos almacenado un número entero de entre los 2^n números posibles. Pero para n qubits, podemos formar infinitos estados, cualquier combinación lineal de los 2^n estados base.

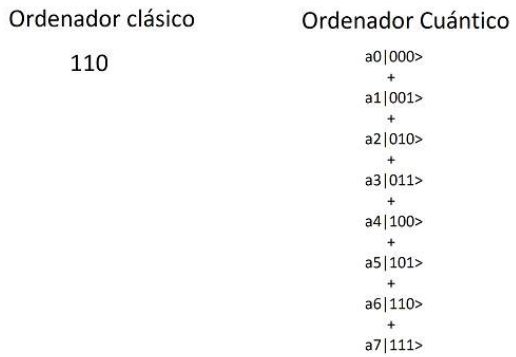

Figura 3: Cantidad de información con 3 bits y 3 qubits.

4.5. Puertas cuánticas.

Los ordenadores clásicos, están compuestos por puertas lógicas como AND, OR y NOT. Estas puertas están conectadas por cables que van de la salida de una a la entrada de otra. En cambio, los ordenadores cuánticos están compuestos por puertas cuánticas, que realizan cambios en las probabilidades de obtener un estado base. Además, las puertas cuánticas son reversibles a diferencia de la mayoría de las puertas lógicas clásicas.

Las puertas cuánticas son representadas mediante una matriz unitaria, es decir, una matriz cuya inversa es igual a su conjugada transpuesta. Seguidamente, vamos a explicar algunas de las más utilizadas.

Puerta Hadamard:

Hadamard es una puerta esencial en computación cuántica y en el diseño de algoritmos cuánticos, ya que con esta puerta pasamos un qubit en base computacional a un qubit en superposición, como una combinación lineal de $|0\rangle$ y $|1\rangle$. Al aplicar la puerta Hadamard a cualquiera de los estados base, devuelve una superposición equiprobable. Esto es equivalente a hacer una rotación de π radianes sobre el eje X y otra rotación de $\pi/2$ radianes sobre el eje Y en la esfera de Bloch.

Representación matricial:

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}.$$

Aplicada al estado $|0\rangle$

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

Aplicada al estado $|1\rangle$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Los estados $|+\rangle$ y $|-\rangle$, representan una forma compacta de aplicar Hadamard a los estados base $|0\rangle$ y $|1\rangle$ respectivamente.

La representación matricial de amplitudes de $|+\rangle$ es:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}.$$

La representación matricial de amplitudes de $|-\rangle$ es:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}.$$

Observemos el efecto de aplicar Hadamard a un estado en superposición: $|-\rangle$, $H|-\rangle$:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}=\frac{1}{2}\begin{bmatrix}0\\2\end{bmatrix}=\begin{bmatrix}0\\1\end{bmatrix}=|1\rangle\,.$$

Ahora a $|+\rangle$, $H|+\rangle$:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}=\frac{1}{2}\begin{bmatrix}2\\0\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}=|0\rangle\,.$$

A continuación, presentamos una serie de operaciones que serán usadas y mencionadas a lo lardo de este texto. Cabe destacar que la puerta inversa de Hadamard es la propia Hadamard, así por ejemplo si aplicamos Hadamard al estado $|+\rangle$, como acabamos de comprobar, obtenemos:

$$H|+\rangle=|0\rangle\,.$$

De la misma forma si aplicamos Hadamard a $|-\rangle$, obtenemos:

$$H|-\rangle=|1\rangle\,.$$

Al procedimiento de aplicar puertas Hadamard en paralelo a varios qubits lo llamaremos:

$$H^{\otimes n}, \tag{1}$$

lo podemos generalizar para n qubits, inicializados a $|0\rangle$, de la siguiente forma y es lo que se conoce como **Transformada de Hadamard**:

$$\frac{1}{\sqrt{2^n}}\sum_x|x\rangle\,. \tag{2}$$

Así para $n = 2$, donde hay 4 posibles estados, aplicando la fórmula (2) de modo que con $H^{\otimes 2}$, con $\otimes$ como producto tensorial obtenemos un estado resultante con la combinación de todos los posibles valores:

$$\frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}.$$

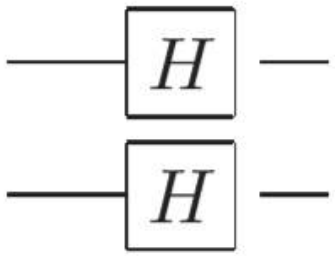

Figura 4: Circuito para dos puertas Hadamard en paralelo $H^{\otimes 2}$.

Estas operaciones, nos serán se utilidad en las siguientes secciones.

Puerta NOT o puerta X:
Esta puerta intercambia las probabilidades sobre un qubit.

$$a\,|0\rangle + b\,|1\rangle \to b\,|0\rangle + a\,|1\rangle\,.$$

Representación matricial:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Puerta U_1:
Esta puerta realiza una rotación en el eje Z de la esfera de Bloch, de un ángulo expresado en radianes pasado como parámetro.

$$U_1(z) = \begin{pmatrix} 1 & 0 \\ 0 & e^{iz} \end{pmatrix}$$

Puerta U_2:

Esta puerta recibe dos ángulos en radianes como parámetros, el primero para una rotación en el eje X y el segundo para para una rotación en el eje Z.

$$U_2(x,z) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{iz} \\ e^{ix} & e^{iz+ix} \end{pmatrix}$$

Puerta U_3:

Esta puerta recibe tres ángulos en radianes como parámetros, el primero para una rotación en el eje Y, el segundo para para una rotación en el eje X y el tercero para una rotación del eje Z.

$$U_1(y,x,z) = \begin{pmatrix} \cos(y/2) & -e^{iz}\sin(y/2) \\ e^{ix}\sin(y/2) & e^{iz+ix}\cos(y/2) \end{pmatrix}$$

Puerta UC_1: Equivale a la puerta $UROT$ que Qiskit, la usaremos más adelante en la transformada cuántica de Fourier. Realiza la rotación indicada en la puerta U_1 en el qubit objetivo si el qubit de control está a 1.

$$CU_1(z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{iz} \end{pmatrix}$$

Puerta SWAP:

Esta puerta intercambia dos qubits.

$$|01\rangle \rightarrow |10\rangle$$

Representación matricial:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Puerta CNOT:

Las puertas controladas son un tipo de puertas que se caracterizan por tener uno o varios qubits de control y otro objetivo. Cuando el qubit de control vale 0, la puerta no se activa y no cambia el estado del objetivo. Cuando vale 1, entonces aplica las operaciones sobre el objetivo.

La puerta CNOT aplica al qubit objetivo la operación NOT, si el qubit de control tiene el valor $|1\rangle$. Y no hará nada en caso de que el valor del control valga $|0\rangle$.

Veamos cómo actúa la puerta CNOT con el primer qubit como control y el segundo como objetivo.

$$|00\rangle \rightarrow |00\rangle$$

$$|01\rangle \rightarrow |01\rangle$$

$$|10\rangle \rightarrow |11\rangle$$

$$|11\rangle \rightarrow |10\rangle$$

Representación matricial:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Puerta CCNOT o Toffoli:

Actúa de forma similar a la puerta CNOT, solo que en este caso hay dos controles, ambos han de tener el valor $|1\rangle$ para que se aplique la operación NOT sobre el qubit objetivo.

Veamos cómo actúa la puerta CCNOT con el primer y segundo qubit como control y el tercero como objetivo.

$$|000\rangle \rightarrow |000\rangle$$

$$|001\rangle \rightarrow |001\rangle$$

$$|010\rangle \rightarrow |010\rangle$$

$$|011\rangle \rightarrow |011\rangle$$

$$|100\rangle \rightarrow |100\rangle$$

$$|101\rangle \rightarrow |101\rangle$$

$$|110\rangle \rightarrow |111\rangle$$

$$|111\rangle \rightarrow |110\rangle$$

Representación matricial:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Puerta CSWAP:

De forma similar a todas las otras puertas controladas, en función del qubit de control, esta puerta aplica la operación SWAP sobre dos qubits.

Veamos cómo actúa la puerta CSWAP con el primer qubit como control e intercambia el segundo y tercer qubit.

$$|000\rangle \rightarrow |000\rangle$$

$$|001\rangle \rightarrow |001\rangle$$

$$|010\rangle \rightarrow |010\rangle$$

$$|011\rangle \rightarrow |011\rangle$$

$$|100\rangle \rightarrow |100\rangle$$

$$|101\rangle \rightarrow |110\rangle$$

$$|110\rangle \rightarrow |101\rangle$$

$$|111\rangle \rightarrow |111\rangle$$

Representación matricial:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4.6. Phase Kickback

Con la puerta Hadamard y la puerta CNOT, podemos conseguir resultados bastantes útiles, que emplearemos más adelante en la sección de algoritmos cuánticos, más concretamente en el algoritmo de Bernstein-Vazirani.

Primero, vamos a observar los estados resultantes de aplicar la puerta CNOT en estados en superposición. Observemos el estado resultante de aplicar CNOT a dos qubits en superposición de la forma:

$$H^{\otimes 2} |00\rangle = |{+}{+}\rangle = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}.$$

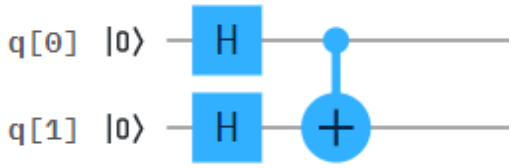

Figura 5: Circuito resultante de aplicar CNOT al estado $|{+}{+}\rangle$

La matriz de amplitudes resultante de la Figura 5 es :

$$\frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix}.$$

Con lo que observamos que la aplicación de CNOT que intercambia las amplitudes de $|01\rangle$ y $|11\rangle$, no varía al estado $|{+}{+}\rangle$, el cual su matriz de amplitudes también se corresponde con la de figura 5. Además si aplicamos otra vez Hadamard a ambos qubits, para después proceder a medir, obtenemos el estado inicial $|00\rangle$. Visualmente los dos qubits de la figura 5 nos quedan como en la figura 6.

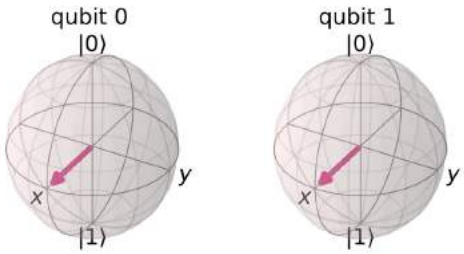

Figura 6: Estado de los qubits $|{+}{+}\rangle$.

Con el estado $|-\rangle$ como objetivo, obtenemos un resultado más interesante, ya que conseguiremos añadir una fase al qubit de control. Ahora aplicamos CNOT al estado de la forma:

$$H\,|{-}{+}\rangle = \frac{|00\rangle + |01\rangle - |10\rangle - |11\rangle}{2}.$$

:

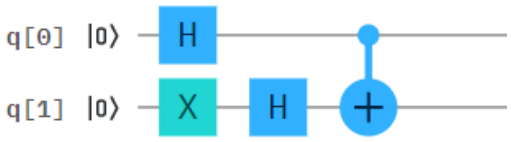

Figura 7: CNOT al estado $|-+\rangle$.

La matriz de amplitudes resultante de la Figura 7 es :

$$\frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}.$$

Visualmente en la esfera de Bloch, los qubits nos quedan como en la figura 8.

Si ahora, aplicamos Hadamard al circuito de la figura 7, nos queda el circuito de la figura 9, el cual midiendo colapsa al estado $|11\rangle$.

De esta forma, hemos cambiado el qubit de control permaneciendo igual el qubit objetivo, $CNOT\,|{-}{+}\rangle = |{-}{-}\rangle$. Por lo tanto, la CNOT se puede

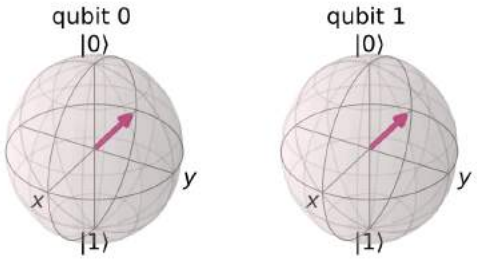

Figura 8: Estado de los qubits al aplicar CNOT sobre $|-+\rangle$.

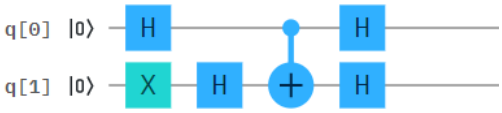

Figura 9: Phase kickback

aplicar en ambas direcciones. Esto es muy útil en el diseño de algoritmos cuánticos, posteriormente lo veremos con el algoritmo de Bernstein-Vazirani.

4.7. Transformada cuántica de Fourier,QFT.

La **transformada cuántica de Fourier** [10], es la implementación cuántica de la transformada discreta de Fourier [6]. Se utiliza con frecuencia en el diseño de algoritmos cuánticos, como por ejemplo, el algoritmo de Shor que mostraremos en la sección de algoritmos cuánticos. En esta sección, primero hacemos una breve presentación matemática, pero lo importante es la idea intuitiva que se observa sobre los ejemplos que mostramos.

4.7.1. Procedimiento QFT.

Dado un vector $(x_0, ..., x_{n-1})$, lo mapea en el vector $(y_0, ..., y_{n-1})$ de acuerdo con la siguiente formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=x}^{N-1} x_j e^{2\pi i \frac{jk}{N}}. \tag{3}$$

De la misma forma, dado un estado cuántico produce la siguiente trans-

formación:

$$\sum_{i=0}^{N-1} x_i \left|i\right\rangle \rightarrow \sum_{i=0}^{N-1} y_i \left|i\right\rangle .$$

Esta transformación solo afecta a las amplitudes de los qubits, por lo que la transformación, la podemos renombrar de la siguiente forma, siendo $w_N^{jk} = e^{2\pi i \frac{jk}{N}}$:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w_N^{jk} = \left|x\right\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} w_N^{jk} \left|y\right\rangle . \tag{4}$$

Estas fórmulas matemáticas, se traducen en una transformación desde las bases computaciones $\left|0\right\rangle \left|1\right\rangle$, que tienen su efecto sobre el eje X y que son como el sistema binario clásico que conocemos, hacia las bases de Fourier $\left|+\right\rangle$ $\left|-\right\rangle$ que tienen su efecto sobre el eje Z. Un ejemplo de esta transformación es la puerta Hadamard, que es la QFT sobre un único qubit y transforma los estados $\left|0\right\rangle$ y $\left|1\right\rangle$ en los estados $\left|+\right\rangle$ y $\left|-\right\rangle$ respectivamente. A modo de intuición, primero vamos a representar un determinado número en las bases computacionales, para posteriormente representarlo en las bases de Fourier.

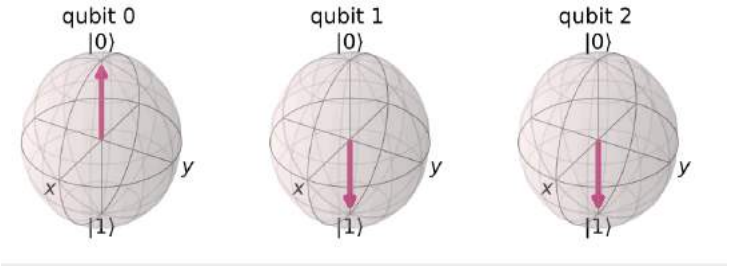

Figura 10: El número 6 representado en bases computacionales.

El resultado de aplicar QFT sobre 3 qubits de la forma $\left|110\right\rangle$, se observa en la figura 12. Observamos las rotaciones que se aplican sobre el eje Z, en el caso de este ejemplo en el que se quiere codificar el número 6 en las bases de Fourier, el qubit más a la izquierda se rota $\frac{6}{2^n}2\pi$, el de su derecha por el doble $\frac{12}{2^n}2\pi$ y así sucesivamente, en nuestro ejemplo obtenemos el circuito de la figura 11.

Observamos que primero se aplica la transformada de Hadamard sobre todos los qubits. De esta forma ya se pueden realizar las rotaciones sobre el eje Z. Como resultado de ejecutar el circuito de la figura 11, obtenemos los qubits en la forma de la figura 12.

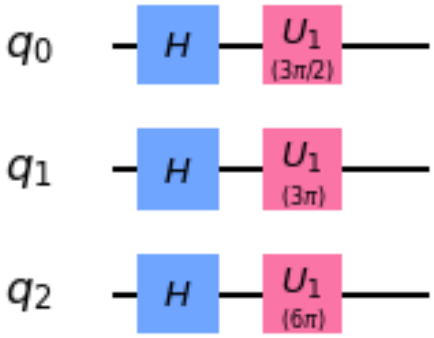

Figura 11: Circuito que implementa la QFT.

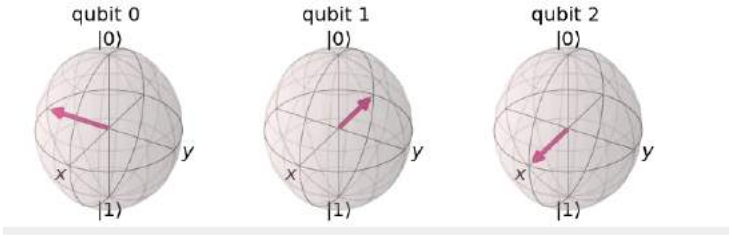

Figura 12: El número 6 representado en bases de Fourier.

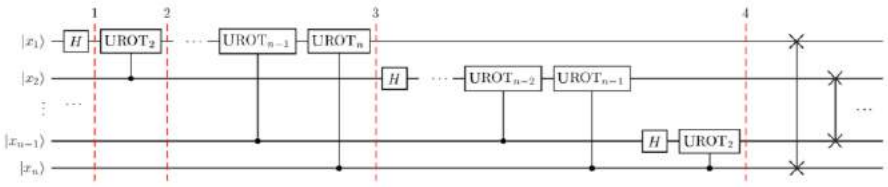

Figura 13: Circuito QFT para n qubits.

Para aplicar la QFT a un estado cuántico cualquiera, nos valdremos del circuito de la figura 13.

Para la implementación del circuito general, vamos a utilizar puertas Hadamard, una puerta que realiza rotaciones controladas sobre el eje Z, llamada U_1 controlada, y puertas SWAP que intercambian qubits. Como observamos en el circuito de la figura 13, la rotación mas pronunciada se produce sobre el qubit menos significativo, por eso es intercambiada por el qubit más significativo.

A modo de demostración, vamos a implementar el ejemplo de la figura 12, para ello, nos valdremos del circuito genérico de la figura 13, lo demostraremos en Qiskit.

Primero, mostramos el circuito resultante:

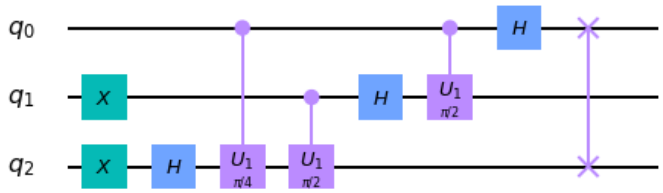

Figura 14: Circuito que aplica la QFT al estado $|110\rangle$.

Ejecutando el circuito de la figura 14, obtenemos un estado igual que el ejecutado en la figura 11, por lo tanto, usando el circuito genérico hemos implementado la idea intuitiva de las rotaciones, en las que se basa la QFT, expresada en la figura 11.

Finalmente, para medir en las bases computacionales, que son en las que se miden los resultados , aplicamos la **QFTInversa** que es la aplicación

inversa de la QFT.

5. Frameworks:

Después de investigar sobre computación cuántica, hemos visto que dos de los frameworks más utilizados para desarrollar circuitos son Cirq y Qiskit, por ello procederemos a compararlos.

5.1. IBM: Qiskit.

Qiskit [4] es un framework de código abierto para diseñar circuitos cuánticos y ejecutarlos en simuladores y computadores cuánticos reales creada por IBM y publicada en 2016. Está basado en Python, pero también tiene versiones para swift y javascript.

Qiskit está compuesto por cuatro componentes:

- Terra. Proporciona herramientas para crear circuitos cuánticos en código máquina.
- Aqua. Contiene una biblioteca de algoritmos cuánticos.
- Aer. Contiene simuladores que se almacenan localmente en el dispositivo del usuario.
- Ignis. Contiene herramientas que permiten la configuración de ruido en los simuladores.

5.2. Google: Cirq.

Cirq [15] es una herramienta de código abierto, desarrollada por Google AI Quantum Team y publicada en el International Workshop on Quantum Software and Quantum Machine Learning el 18 de Julio del 2018.

Esta herramienta nos permite manipular circuitos cuánticos, de una forma abstracta, tal que una persona sin conocimiento de física sería capaz de utilizarla. Estos algoritmos se ejecutan en máquinas Noisy Intermediate Scale Quantum (NISQ), es decir, máquinas de un número pequeño de qubits y sin corrección de errores causados por el ruido.

5.3. Comparativa.

Vamos a dividir esta comparativa en tres aspectos clave: facilidad de aprendizaje, estadísticas y herramientas que ofrece cada una de ellas.

En primer lugar, vamos a realizar un pequeño experimento en cada una de las dos herramientas, es decir, implementaremos un pequeño circuito para poder evaluar los tres aspectos mencionados anteriormente. Hemos decidido escoger el circuito que se ve en la figura 15.

Este circuito consta de una puerta de Hadamard aplicada en el primer qubit, seguido de dos puertas CNOT, una que tiene como control el primer qubit y como objetivo el segundo, y la otra tiene el segundo qubit como control y el tercero como objetivo. Para entender el circuito, se tiene que interpretar de izquierda a derecha, de modo que las puertas más a la izquierda son tratadas antes que las de la derecha. En este circuito, se espera que se obtenga como resultado el estado:

$$\frac{|000\rangle + |111\rangle}{\sqrt{2}}.$$

De tal manera que al medir, se obtenga el valor $|000\rangle$ con probabilidad del 50 %, o el valor $|111\rangle$ con idéntica probabilidad.

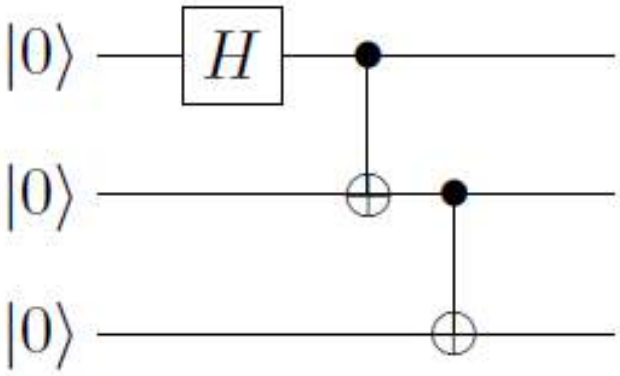

Figura 15: Circuito que analizar.

Una vez escogido el circuito que queremos implementar, nos dirigimos a abrir las herramientas.

En cuanto a Qiskit, para este experimento vamos a utilizar su plataforma online, IBM Quantum Experience, que nos permite tanto redactar el código a mano, como arrastrar las puertas desde una interfaz gráfica bastante intuitiva.

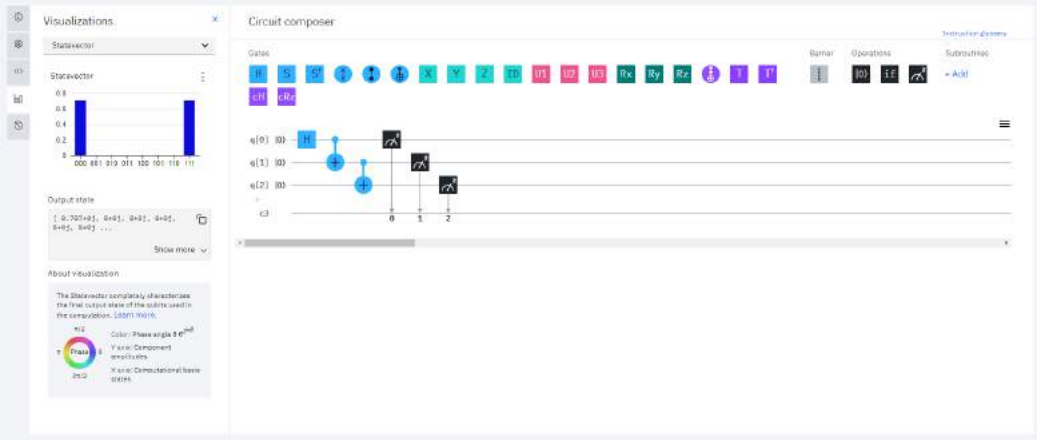

Figura 16: Interfaz del compositor de circuitos de IBM Quantum Experience.

Como podemos ver en la figura 16, la interfaz es bastante fácil de entender. La pantalla esta dividida en tres secciones:

- La de la derecha, nos muestra el circuito, los cuadrados y círculos de colores representan las distintas puertas cuánticas.
- A la izquierda, hay un pequeño panel de botones donde podemos ir a diferentes pantallas, entre ellas ver un esquema del código o esta previsualización de la salida.
- La sección del centro, en este caso nos muestra este histograma que nos ofrece bastante información, el eje horizontal nos muestra los estados de la base computacional. Para este caso solo hay dos estados posibles: el $|000\rangle$ y el $|111\rangle$. Por otro lado, el eje vertical mide la magnitud de las amplitudes asociadas con cada uno de los estados. Por último, otra de las cosas destacables de esta gráfica, es el mapa de colores que nos muestra más abajo, indicando la fase de la amplitud. Como las barras del histograma son azules, dicha fase será 0, sin embargo, si fueran amarillas sería π.

Sin embargo, Cirq no nos ofrece una interfaz similar, así que nos descargamos todo lo necesario para instalarlo en local y ejecutaremos el código desde la terminal del ordenador.

Una vez seguidos todos los pasos explicados en ambas documentaciones, al implementar el circuito en ambas herramientas, llegamos al punto que se ve en las figuras 17 y 18.

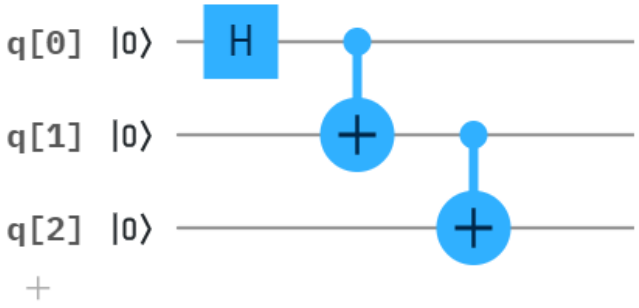

Figura 17: Circuito en Qiskit.

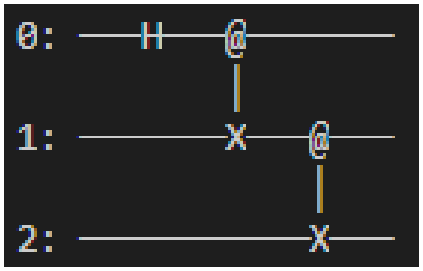

Figura 18: Circuito en Cirq.

En la figura 17, podemos ver cómo queda el circuito sobre la plataforma de Qiskit y en la figura 18, como se imprime en la terminal Cirq.

El siguiente paso es ejecutar el circuito en ambas plataformas. En esta parte observamos que Qiskit nos ofrece ejecutarlo en un ordenador cuántico real, de esta forma se pueden obtener datos reales y no solo los simulados.

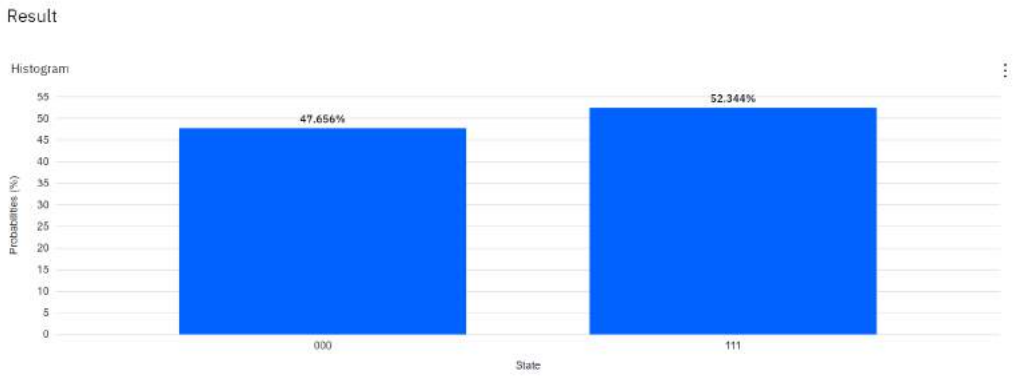

Figura 19: Resultados en Qiskit.

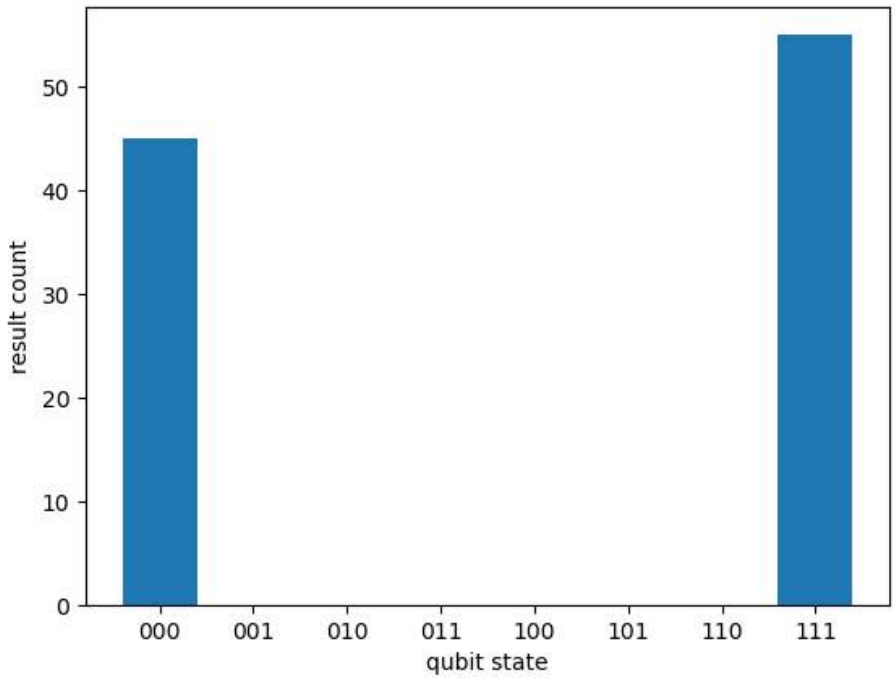

Figura 20: Resultados en Cirq.

Como podemos ver en las figuras 19 y 20, los resultados en ambas herramientas son muy similares tanto en forma como en resultado.

Para ambas figuras, el qubit más a la izquierda sería el qubit número 2 es el más significativo. Y el que se encuentra más a la derecha es el qubit 0 es el menos significativo.

Por otro lado, el eje Y nos muestra la probabilidad con la que se espera dicho resultado. En Cirq, El estado $|000\rangle$ se espera con una probabilidad del 45 % más o menos. Y para el estado $|111\rangle$ alrededor de un 55 %. En Qiskit, se puede ver la probabilidad exacta justo encima de la barra correspondiente ($47,656\,\%$ y $52,433\,\%$, respectivamente).

Para poder obtener estas probabilidades, el circuito se ejecuta una serie de veces, al que llamaremos shots, termino que usaremos más adelante en experimentos en entornos reales. En cada ejecución se produce un resultado y cuando se realizan todas, se obtienen los porcentajes de cada posible resultado en relación al número de ejecuciones.

Para poder comparar estadísticamente las dos aplicaciones, vamos a utilizar las métricas de Halstead [18]. Estas métricas se basan en contar en el código fuente operandos y operadores, y aplicando unas simples cuentas obtenemos la complejidad del programa.

Primero debemos definir una serie de variables:

n_1 y n_2 son los números de operadores y operandos diferentes respectivamente, n es el vocabulario de un programa.

N_1 y N_2 son los números de operadores y operandos totales respectivamente, y N es la longitud total del programa.

Una vez definidas estas variables y viendo el anexo H, podemos identificar la cantidad de operadores y operandos que hay para el código de Qiskit.

$$n_1 = 11$$

$$n_2 = 6$$

$$N_1 = 14$$

$$N_2 = 16$$

Por otra parte, viendo el Anexo I podemos hacer lo mismo para Cirq.

$$n_1 = 10$$

$$n_2 = 5$$

$$N_1 = 17$$

$$N_2 = 21$$

El vocabulario (n) antes mencionado se calcula a través de una sencilla formula:

$$n = n_1 + n_2$$

Siendo así para Qiskit $n = 17$ y para Cirq $n = 15$

La longitud observada y el volumen los podemos calcular de esta forma:

$$N = N_1 + N_2$$

$$V = Nlog_2 n$$

Por lo tanto, la longitud observada y el Volumen para Qiskit son $N = 30$ Y $V = 122{,}62$, análogamente, para Cirq son $N = 38$ y $V = 148{,}46$.

Con el volumen, queremos tomar una medida más precisa de la dificultad de comprender el código, teniendo en cuenta la longitud del codigo (N) y su vocabulario (n). Para este ejemplo, Cirq es el que tiene un volumen mayor, lo que significa que es más complejo.

En cuanto a las herramientas que ofrecen, tanto Qiskit como Cirq son muy similares. Sin embargo, hay que mencionar que esta última se encuentra en fase alpha, lo que significa que aún está en desarrollo e implica que se pueden producir cambios o eliminar partes de la API al hacer nuevos lanzamientos.

Por otro lado, Qiskit tiene un repertorio de herramientas más amplio, por ejemplo el elemento barrera no se encuentra todavía disponible en Cirq. También influye que Qiskit fue creado mucho antes, por lo que ha tenido más tiempo para avanzar y mejorar.

En resumen, después de evaluar estos tres factores, facilidad de aprendizaje, complejidad y herramientas que ofrece, nos decantamos por el framework de IBM, Qiskit.

6. Algoritmos cuánticos:

6.1. Algoritmo de Deutsch-Jozsa.

Para una primera aproximación a los algoritmos cuánticos, presentamos el algoritmo de **Deutsch-Jozsa** [10] [16]. Este algoritmo evalúa si una función f

: $f(x_n...x_1, x_0)$, cuya entrada son ceros y unos, y su salida es cero o uno, tiene la propiedad de ser constante o balanceada. Se ha asegurado previamente que esta función f, tiene esta propiedad. Una función es constante si devuelve siempre cero o uno para cualquier entrada. Por el contrario, es balanceada si sus salidas son la mitad uno y la otra mitad cero.

Tomemos por ejemplo una función que tiene como entrada números en binario de la forma $f(x_2, x_1, x_0)$, con N=2^3 posibles entradas. La función tendrá el valor 0, si $x_0 = 1$ y el valor 1, si $x_0 = 0$. En el mejor de los casos encontramos dos valores distintos en dos comprobaciones, $f(1, 0, 1) = 0$ y $f(0, 0, 0) = 1$. Sabiendo así que la función es balanceada. Sin embargo, en el caso peor, haríamos $2^{3-1} + 1 = 5$ comprobaciones pare tener la certeza de que tipo de función se trata.

Resumiendo, si hay 2^n posibles entradas, habría que evaluar la función $\frac{2^n}{2} + 1$ veces para tener la certeza de qué tipo de función se trata.

El algoritmo de Deutsch-Jozsa mejora exponencialmente su rendimiento para este problema específico. Como acabamos de mencionar, en computación clásica, en el caso peor para un problema de 2^3 posibles entradas, sería necesario evaluar $\frac{8}{2} + 1 = 5$ veces la función para tener la certeza de que tipo de función se trata. Pero en computación cuántica, sería necesario solo una evaluación para conocer el resultado del problema, por lo tanto, no cabe duda que para este problema la computación cuántica es mejor.

Antes de generalizar el problema a n qubits, trataremos el problema de Deutsch, que es una implementación para un único qubit. Este algoritmo fue el primero cuyo comportamiento, en coste computacional, se mostró mejor que los algoritmos diseñados en computación clásica.

Este algoritmo, hace uso de la superposición para formar un único estado, que contiene todas las posibles salidas para una entrada determinada. Para este desarrollo, necesitaremos la U_f **query function** y puertas Hadamard. La query function admite varias implementaciones, su ecuación práctica viene dada como sigue:

$$|x\rangle = (-1)^{f(x)} |x\rangle . \tag{5}$$

Vamos a observar el efecto que tiene la ecuación (5), en un estado en superposición $|+\rangle$ y estudiar sus implicaciones. Dado el estado en superposición:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

Al aplicarle U_f, definida en la ecuación (5) a $|\psi_1\rangle$ nos queda:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle).$$

Dependiendo de la query function, el valor del qubit será positivo o negativo. Es en este punto donde procedemos a suponer los posibles valores de la query function.

Supongamos $f(0) \neq f(1)$ entonces:

$$|\psi_3\rangle = \frac{\pm 1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Si por el contrario $f(0) = f(1)$:

$$|\psi_4\rangle = \frac{\pm 1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

Ahora aplicamos Hadamard a $|\psi_3\rangle$ y nos queda:

$$|\psi_5\rangle = \pm|1\rangle.$$

También aplicamos Hadamard a $|\psi_4\rangle$ y nos queda:

$$|\psi_6\rangle = \pm|0\rangle.$$

Recordemos el efecto que tiene aplicar Hadamard a un estado en superposición H $|-\rangle$:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}0\\ 2\end{bmatrix} = \begin{bmatrix}0\\ 1\end{bmatrix} = |1\rangle.$$

Para el desarrollo y explicación de este algoritmo, definimos la **query function** de forma general U_f como sigue:

$$U_f = |x, y \oplus f(x)\rangle. \tag{6}$$

La función f, hace la transformación como lo indica la ecuación (6), se trata de una función booleana. Como requisito, la función a determinar constante o balanceada debe ser implementada como la ecuación (6).

Hay que tener en cuenta, que dada la naturaleza de la computación cuántica, al medir un estado en superposición, colapsará a un estado de la forma

$|x, f(x)\rangle$, lo que nos quiere decir, que este algoritmo no nos mostrará los valores de las evaluaciones f(0) o f (1). Sino que nos dará una información global, en este caso nos proporcionará si $f(0) = f(1)$ o $f(0) \neq f(1)$, más precisamente, como mencionamos antes, nos dirá si una función dada es constante o balanceada.

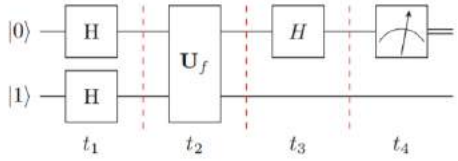

Figura 21: Problema de Deutsch 1 qubit ($|0\rangle\,|1\rangle$).

6.1.1. Procedimiento al problema de Deutsch.

El problema de Deutsch resuelve el problema para un único qubit. En la figura 21 observamos su circuito genérico.

Algoritmo del problema de Deutsch.

1. Aplicamos la puerta Hadamard al estado de entrada, $|01\rangle$. De esta forma ponemos el estado como producto de dos superposiciones.

2. Aplicamos la función U_f al estado anterior.

3. Aplicamos Hadamard al primer qubit.

4. Procedemos a medir, a continuación en el ejemplo explicamos como interpretar esta medición.

Ejemplo:

Vamos a desarrollar el ejemplo de la figura 21. Aplicaremos la **query function**, en este caso de la forma de la ecuación (6), pero en la práctica, como veremos en secciones siguientes, esta ecuación se corresponderá con una puerta CNOT de la forma $C_{1,2}$, con el primer qubit como control y el segundo como objetivo. En rojo hemos resaltado el segundo qubit, que a efectos prácticos solo sirve para entender mejor el funcionamiento de este algoritmo y poder implementar la query function de la forma de la ecuación (6). Estará presente también en la generalización para n qubits, lo que se conoce propiamente como algoritmo de **Deutsch-Jozsa**.

1. Inicializamos el primer qubit a $|0\rangle$ y el segundo a $|1\rangle$. De esta forma nos quedamos en el estado:

$$|\phi_1\rangle = |0\rangle\,|1\rangle.$$

2. Aplicamos Hadamard a ambos qubits. De este modo tenemos el estado que sigue, denotamos que aparecen todos los posibles valores que toma la función:

$$|\phi_2\rangle = \frac{1}{2}(|0\rangle\,|0\rangle - |0\rangle\,|1\rangle + |1\rangle\,|0\rangle - |1\rangle\,|1\rangle) = |+\rangle\,|-\rangle.$$

3. Aplicamos la función U_f a ambos qubits. La función admite varias implementaciones, vamos a desgranarla para entender su aplicación.

   Tenemos en nuestro ejemplo (x, y), con $x = |+\rangle$ e $y = |-\rangle$, de forma que aplicando nos queda $(x, y \oplus f(x))$, con ello si $f(x) = 0$, obtenemos el mismo (x, y). Sin embargo, si $f(x) = 1$, el signo del segundo qubit y cambia. Usando la generalización de este comportamiento tomando (5), nos queda $(-1)^{f(x)}\,|x\rangle\,(|-\rangle)$. En nuestro caso tenemos:

$$\pm\,|-\rangle\,(|-\rangle).$$

   Visto en otra notación, los qubits nos queda:

$$|\phi_3\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

4. Aplicamos Hadamard al primer qubit o también a ambos:

$$|\phi_4\rangle = |1\rangle\,|1\rangle.$$

5. Medimos el primer qubit:

   Como se observa, el resultado de medir el primer qubit colapsa al estado $|1\rangle$, por lo tanto obtenemos un 1 como resultado. Lo que interpretamos como que las funciones f(0) y f(1) son distintas, por lo tanto se trata de una función balanceada. Intuitivamente y observando el estado $|\phi_3\rangle$ resultante después de aplicar la U_f, observamos que en el primer qubit en superposición, los signos de $|0\rangle$ y $|1\rangle$ son diferentes. Por lo tanto queda claro que su evaluación es diferente.

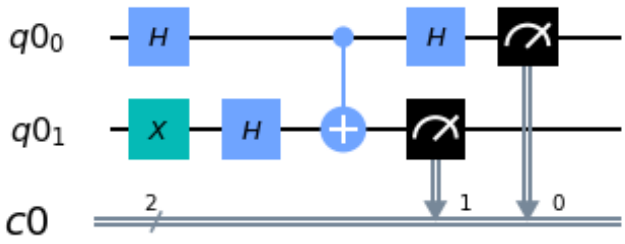

Figura 22: Algoritmo de Deutsch en Qiskit ($|0\rangle\,|1\rangle$).

Con una única evaluación hemos obtenido el resultado esperado. En computación clásica, por el contrario para este ejemplo hubieran sido necesarias dos evaluaciones, una para evaluar $f(0)$ y otra para evaluar $f(1)$.

6.1.2. Implementación del algoritmo de Deutsch.

Vamos a implementar en Qiskit el ejemplo de la figura 21. Observamos el circuito en Qiskit en la figura 22:

Resultados en un entorno ideal.

Para la interpretación del resultado solo nos interesa la información del primer qubit. Observamos en la figura 23 que en ambos valores aparece el 1 en el primer qubit, por lo tanto se trata de una función balanceada.

Resultados en un simulador con ruido.

A través de Qiskit, podemos hacer una ejecución en un simulador con ruido. En este simulador obtenemos los resultados de la figura 24, en la cual obtenemos resultados similares al entorno ideal.

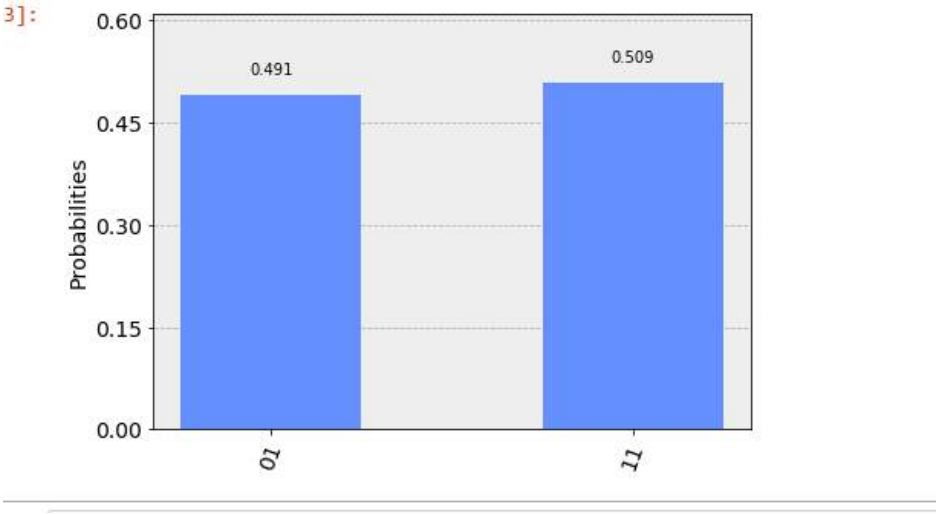

Figura 23: Histograma resultante de la ejecución ideal del circuito de la figura 22.

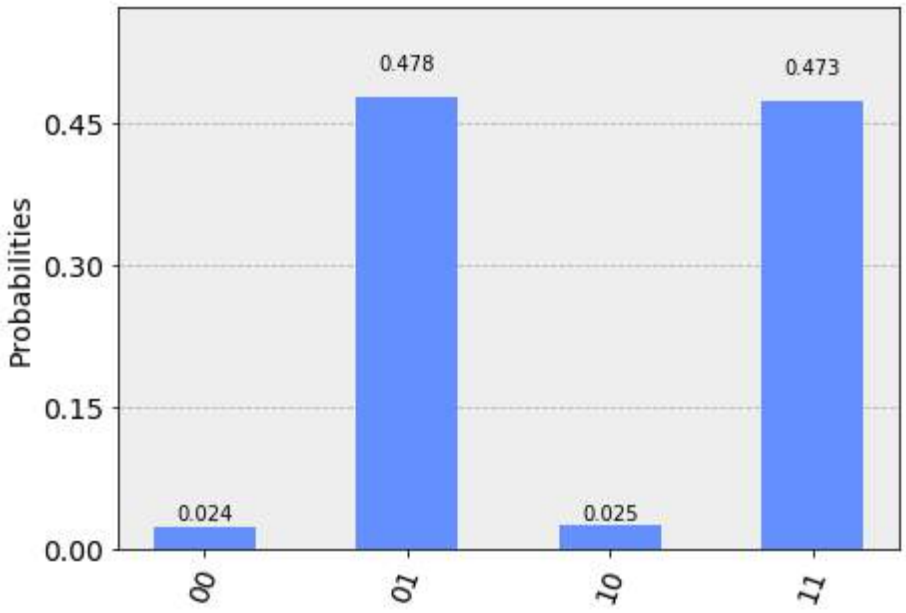

Figura 24: Histograma resultante de la ejecución en el simulador con ruido del circuito de la figura 22.

Resultado en un entorno real cuántico.

Al ejecutar el código sobre el computador cuántico **ibmq london**, observamos los resultados de la figura 25.

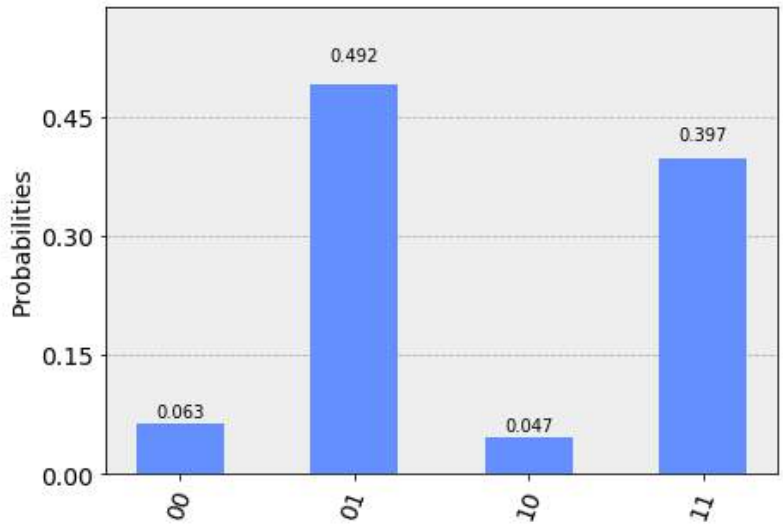

Figura 25: Histograma resultante de la ejecución en entorno real del circuito de la figura 22.

En un entorno real, aparecen dos probabilidades casi despreciables. Esto se debe a la existencia de pequeños errores cuánticos, debido a la tecnología en computación cuántica de hoy en día y en la que se trabaja para minimizar estos pequeños errores. Pero en términos generales, obtenemos el mismo resultado que en el simulador con ruido.

6.1.3. Implementación del algoritmo de Deutsch-Jozsa para n qubits.

Por ejemplo, si queremos aplicar el algoritmo a una función de la forma $f(1,0,0)$, necesitamos generalizar el problema de Deutsch a n qubits.

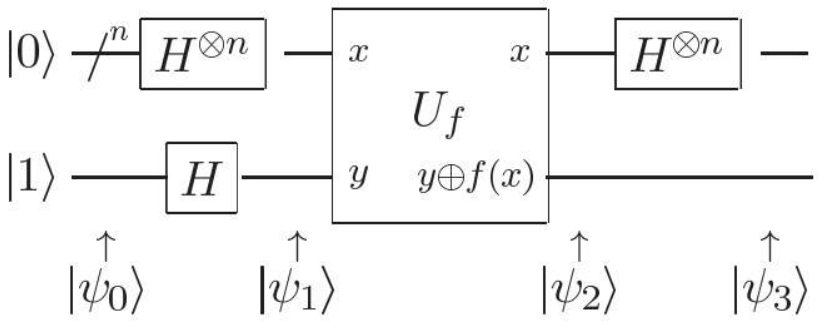

Figura 26: Circuito de Deutsch-Jozsa para n qubits.

A continuación, guiados por el circuito para n qubits de la figura 26, procedemos a explicar un ejemplo para 3 qubits. Cabe recordar, que el algoritmo de Deustsch-Jozsa solo distingue entre funciones constantes o balanceadas. Previamente se aseguró que f tiene esta propiedad, por lo tanto no podemos esperar otro resultado.

Ejemplo del algoritmo de Deutsch-Jozsa en Qiskit para 3 qubits.

Para esta implementación que llamaremos ADJ_3, tenemos que definir una función para la que el algoritmo se encarga de determinar si es constante o balanceada. Para este ejemplo definimos la siguiente:

$$f(x) = x_0 \oplus x_1 x_2. \tag{7}$$

A continuación, necesitamos especificar una **query function** como la ecuación (5). Para poder desarrollar el algoritmo, elegimos la siguiente especificación:

$$Z_0 |x\rangle = (-1)^{x_0} |x\rangle \, CZ_{1,2} = |x\rangle = (-1)^{x_1 x_2} |x\rangle \, . \tag{8}$$

Ahora con estas dos ecuaciones, vamos a desarrollar el algoritmo paso a paso.

El primer paso es poner el estado $|000\rangle$ en superposición, para ello aplicamos **la transformada de Hadamard**, obteniendo:

$$ADJ_3 = \frac{1}{2\sqrt{2}} |000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle \,. \quad (9)$$

El estado ADJ_3 en forma de matriz de amplitudes es:

$$\frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} .$$

En probabilidades tenemos para cada estado:

$$\left| \frac{1}{2\sqrt{2}} \right|^2 = 0{,}125.$$

Si lo multiplicamos por ocho obtenemos **1**.

Ahora aplicamos la query function, que en nuestro caso se corresponde con la ecuación (8). De este modo aplicamos $CZ_{1,2}$ y Z_0 a los qubits, teniendo en cuenta que el qubit menos significativo se coloca más a la derecha, y fijándonos en (9). Al aplicarlo, el efecto que se obtiene es cambiar el signo de algunas amplitudes, obteniendo la siguiente matriz de amplitudes:

$$\frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} . \quad (10)$$

Como paso final y antes de proceder a medir, como lo indica el algoritmo, aplicamos Hadamard a cada qubit. De forma que $H^{\otimes 3}$ será aplicado a la matriz (10).

$$H^{\otimes 3} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} =$$

$$\frac{1}{2\sqrt{2}}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1\\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1\\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1\\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1\\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1\end{bmatrix}. \tag{11}$$

Multiplicando las matrices (11) y (10), obtenemos la siguiente matriz de amplitudes:

$$\frac{1}{8}\begin{bmatrix}0\\ 4\\ 0\\ 4\\ 0\\ 4\\ 0\\ -4\end{bmatrix}. \tag{12}$$

Observamos que los estados de salida son los siguientes:
$|001\rangle$, $|011\rangle$, $|101\rangle$ y $|111\rangle$, cuya probabilidad para cada uno es la siguiente:

$$\frac{1}{2} = 0{,}5. \tag{13}$$

La probabilidad final del algoritmo para cada estado es la siguiente:

$$\left|\frac{1}{2}\right|^2 = 0{,}25. \tag{14}$$

En definitiva, se trata de una función balanceada, pues en la matriz (10) observamos que tras aplicar la query function los signos son diferentes. Por el contrario, si fuera constante sus signos serían iguales. En la conclusión de este algoritmo explicamos qué condiciones se tienen que dar para que una función sea constante.

Implementación.
Observamos el circuito desplegado en Qiskit en la figura 27.

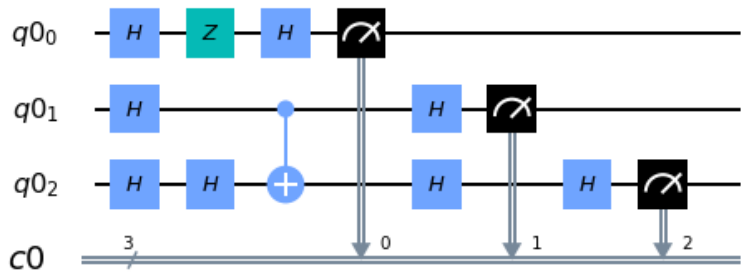

Figura 27: Circuito ADJ-Z 3 qubits balanceado.

Observamos en la figura 28, un resultado prácticamente igual al resultante de operar con matrices, los resultados $|001\rangle$, $|011\rangle$, $|101\rangle$ y $|111\rangle$ con probabilidad de 0,25.

En la figura 29, observamos los vectores de estado correspondientes al resultado obtenido en el resultado (13). El estado $|111\rangle$ se observa en amarillo porque es un valor negativo, que se corresponde al -4 de la matriz (12). Esto se debe a que la fase de la amplitud de este estado es π y las amplitudes del resto de estados están en fase cero. También se puede ver como el estado $|111\rangle$, se encuentra rotado 180 grados respecto a los otros estados en la esfera de Bloch.

Ejecución en un entorno ideal (figura 28).

El resultado esperado para la función $x_1 \bigoplus x_2x_3$, debería ser 1, o lo que es lo mismo, balanceada. Ya que la mitad de sus salidas son ceros, y la otra mitad son unos, cómo muestra el cuadro 1.

El resultado en el entorno ideal figura 28, es similar al modelo matricial como observamos en el resultado (14). En el resultado (13) observamos el valor resultante de la matriz (12), que se corresponde con la figura 29, y que representan las probabilidades de los estados resultantes, antes de ajustar los valores al modelo probabilístico cuántico igual a 1.

Si se hubiera tratado de una función constante, hubiésemos obtenido una medición del estado $|0..,0\rangle$ del 100 %. Esto es debido, a que todas las amplitudes de los estados hubieran tenido el mismo signo. Por lo tanto multiplicando

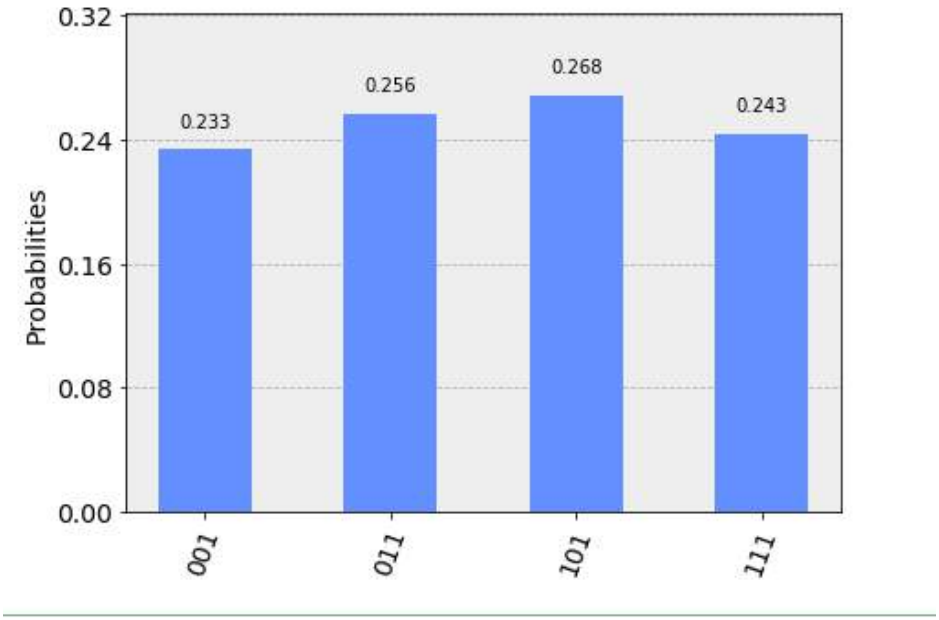

Figura 28: Resultado de la ejecución en entorno ideal.

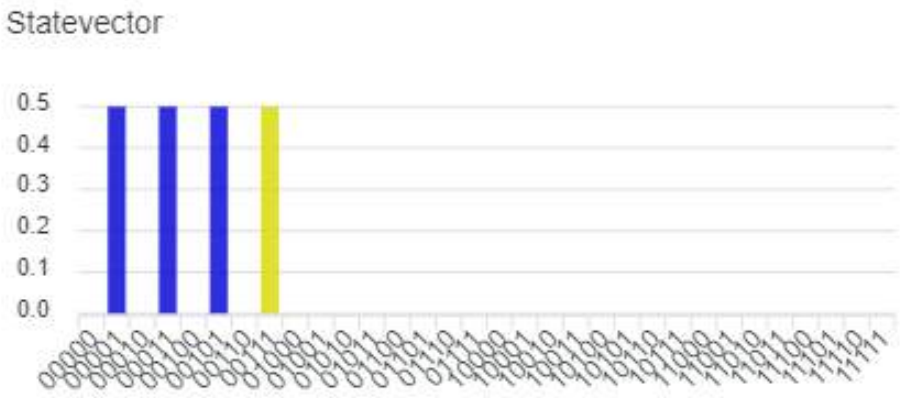

Figura 29: Resultado de la ejecución en entorno ideal sin ajuste de probabilidades a 1.

x_1	x_2	x_2	$x_1 \bigoplus x_2x_3$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Cuadro 1: Función 7.

la matriz (11), cuya primera fila son todos unos, con el vector de amplitudes del mismo signo, se observan claramente estos resultados.

Ejecución en un simulador con ruido (figura 30).

En el simulador con ruido, se obtienen los mismos resultados en términos generales. Aunque aparecen nuevos posibles resultados, se consideran despreciables.

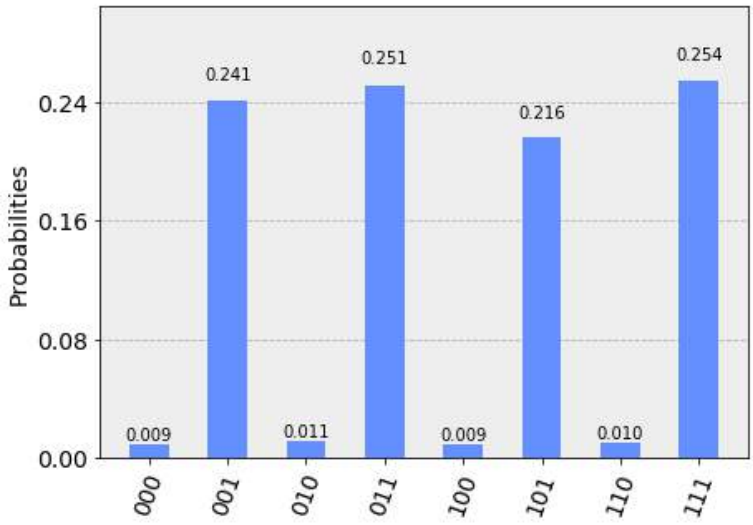

Figura 30: Resultado de la ejecución en un simulador con ruido.

Ejecución en un entorno real cuántico ibmq london (figura 31). El computador ibmq london trabaja con cinco qubits. En este computador podemos observar resultados que difieren un poco con relación a los resultados obtenidos en el entorno ideal y modelo matricial, debido al ruido y a pequeños errores cuánticos. Pero son parecidos a los del simulador con ruido. Cabe recalcar sobre todo que, algunos estados que en entorno ideal no tenían probabilidades de aparecer, en este experimento aparecen todos de forma casi despreciable.

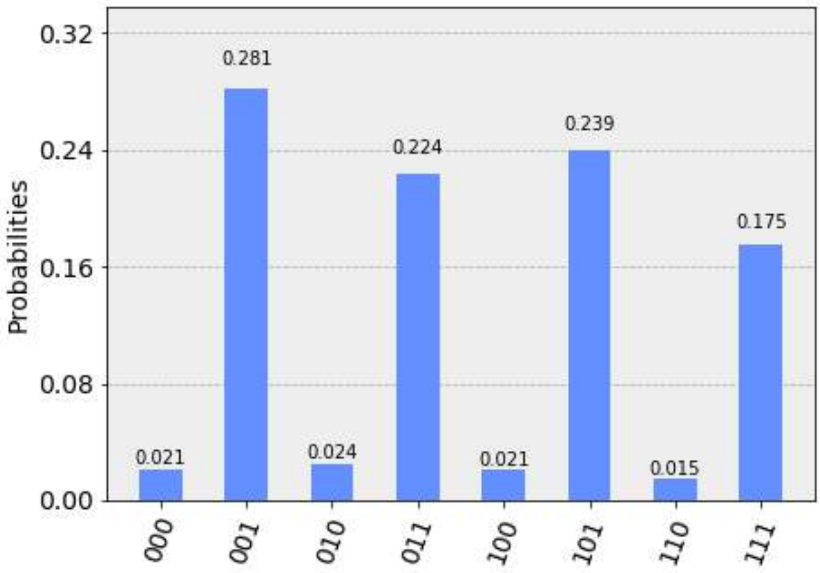

Figura 31: Resultado de la ejecución en un entorno real cuántico.

6.1.4. Conclusiones sobre el algoritmo de Deutsch-Jozsa.

El algoritmo de Deutsch-Jozsa representa el **hola mundo** de la computación cuántica, pero con todos los conceptos cuánticos que esto supone. Para este problema concreto, ya muestra una mejora exponencial en cuanto al coste computacional. Así mismo, introduce conceptos y elementos fundamentales para el diseño de algoritmos cuánticos, como la superposición, que es lo que aporta la aceleración computacional para este problema, o la puerta Hadamard, la cual marca el inicio y final de varios algoritmos. Este algoritmo supuso dar el salto del mundo clásico al mundo cuántico. Sin embargo, com-

prendimos que el entorno clásico va a acompañar y va a colaborar junto con la computación cuántica, ya que las mediciones de los resultados se siguen codificando en binario.

En definitiva, este algoritmo nos hizo comprender el enorme potencial de la computación cuántica y el gran avance que supondría desarrollar algoritmos cuánticos más útiles.

6.2. Algoritmo de Bernstein-Vazirani.

El algoritmo de **Bernstein-Vazirani** [17], es una extensión del algoritmo de Deutsch-Jozsa, pero en este caso resuelve un problema un tanto más complejo. Dada una función f, cuya entrada son cadenas de ceros y unos, y cuya salida es cero o uno de la forma:

$$f(x_n, x_{n-1}, ..., x_0) \rightarrow \{0, 1\}$$

Y un número secreto **S** en binario. El algoritmo de Bernstein-Vazirani adivina este número.

En términos matemáticos, la función f realiza el producto bit a bit módulo 2 de la entrada con el número **S**. Con $f(x)$, donde $x = x_n, x_{n-1}, ..., x_0$, de la forma:

$$f(x) = (S \cdot x) mod 2.$$

El operador "$\cdot$" es el producto escalar bit a bit módulo dos, es una forma de implementar la solución de forma clásica. Por ejemplo con $x = 00001$ de entrada y un número **S** de cinco bits de tamaño. La función devuelve el valor del bit x_0 de **S**. También se usa para determinar los valores de los signos de los qubits [17] en la solución en computación cuántica como veremos explicado más adelante con un ejemplo.

Otra forma de implementar este problema en computación clásica, es con una operación **AND** entre la cadena de entrada y el número que se pretende adivinar.

De este modo, el problema se resuelve de la siguiente forma:

Dado un número **S= 10011** que se pretende adivinar. Realizaríamos 5 comprobaciones, aplicando una mascara con todo ceros y un uno en el bit i de la comprobación i, de la siguiente forma:

$$AND_1 \frac{10011}{00001} = 00001$$

$$AND_2 \frac{10011}{00010} = 00010$$
$$AND_3 \frac{10011}{00100} = 00000$$
$$AND_5 \frac{10011}{01000} = 00000$$
$$AND_5 \frac{10011}{10000} = 10000$$

Lo que significa que, si tenemos un número de n bits, necesitaríamos n comprobaciones. Ya que la entrada de la función son ceros y un único uno. Teniendo así un coste $O(N)$, siendo N el número de comprobaciones. En computación cuántica, este problema se resuelve al igual que el algoritmo de Deutsch-Jozsa, en una única comprobación que explicaremos a continuación.

¿Cómo funciona el algoritmo de Bernstein-Vazirani?

Este algoritmo funciona gracias a la **phase kickback** explicada anteriormente en este texto. En el esquema de este algoritmo (figura 32), podemos ver el esquema de phase kickback, donde observamos que el objetivo está a $|-\rangle$.

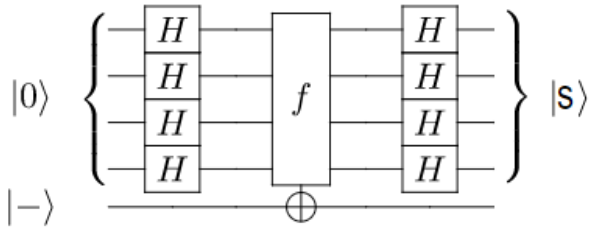

Figura 32: Esquema circuito algoritmo de Bernstein-Vazirani

6.2.1. Procedimiento.

Como mencionamos anteriormente, el problema se resuelve con una única comprobación siguiendo el siguiente algoritmo:

1. Iniciamos poniendo a cero el registro de entrada $|0\rangle^{\otimes n}$ y el qubit de salida lo ponemos a $|-\rangle$.
2. Aplicamos la transformada de Hadamard $H^{\otimes n}$ al registro de entrada.
3. Aplicamos phase kickback a los qubits.
4. Aplicamos la transformada de Hadamard $H^{\otimes n}$ al registro de salida.
5. Procedemos a medir, obteniendo el número **S**.

Para comprender mejor el algoritmo, vamos a desarrollar un ejemplo paso a paso. Como número a adivinar elegimos $\mathbf{S} = |11\rangle$, ya que sobre este número se ven más claras las operaciones que vamos a realizar. Previamente aclararemos y mostraremos la operación que vamos a realizar sobre las amplitudes de los qubits y que representará a la phase kickback. Esta operación es el producto escalar bit a bit módulo dos, de la forma:

$$f_s(x) = S \cdot x =$$
$$(x_n s_n + ... + x_0 s_0) mod 2.$$

El número x representa el estado del qubit, y S el número a adivinar. De esta forma en nuestras aplicaciones, los siguientes casos nos quedarán:

$$01 \cdot 11 = 0 \cdot 1 + 1 \cdot 1 = 1 mod 2.$$
$$11 \cdot 11 = 1 \cdot 1 + 1 \cdot 1 = 0 mod 2.$$

Partimos del estado:

$$|\psi_0\rangle = |00\rangle .$$

Como primer paso aplicamos la trasformada de Hadamard:

$$|\psi_1\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle).$$

Aplicamos el oráculo o query function de la forma $|x\rangle = (-1)^{f_s(x)} |x\rangle$:

$$|\psi_1\rangle = \frac{1}{2}((-1)^{00\cdot 11}|00\rangle + (-1)^{01\cdot 11}|01\rangle + (-1)^{10\cdot 11}|10\rangle + (-1)^{11\cdot 11}|11\rangle)$$

$$|\psi_1\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle).$$

Finalmente, aplicamos la transformada de Hadamard para invertir el estado de superposición $|\psi_1\rangle$. De modo que obtenemos:

$$|\psi\rangle = |11\rangle\,.$$

Como observamos, hemos encontrado el número oculto **S=11**.

6.2.2. Implementación del algoritmo genérico de Bernstein-Vazirani.

La dificultad de este algoritmo reside en la construcción de la función $f_s(x)$ sin tener conocimiento del número **S**. En este experimento, para construir la función $f_s(x)$, se utilizan puertas CNOT cuyos controles son los qubits correspondientes a los bits con valor 1 en el número **S**, y tienen como objetivo el registro salida.

Si en la posición correspondiente al número a adivinar hay un uno, phase kickback hará una rotación de fase pasando de $|-+\rangle$ a $|--\rangle$, cuya medición colapsará a $|11\rangle$ obteniendo el 1 deseado. En caso de que no haya un uno, se aplicará Hadamard a $|0\rangle$ resultando el estado $|+\rangle$, al cual también se le aplicada Hadamard volviendo al $|0\rangle$, por lo tanto colapsando al 0. En la figura 33 se observa este esquema de phase kickback con el último qubit en el estado $|-\rangle$.

Resultados en un entorno ideal.

Procedemos a ejecutar el algoritmo con un único shot en un entorno ideal, (figura 34). Como podemos ver, obtenemos el número **S** con probabilidad del 100 %.

Resultados en un simulador con ruido (figura 35).

Observamos que con ruido la probabilidad de encontrar el número deseado, es la más alta. Aparecen otras probabilidades pero en su mayoría son despreciables en comparación con la esperada.

Resultados en un entorno real ibmq-16-melbourne.

Procedemos a ejecutar el algoritmo en un entorno real **ibmq-16-melbourne** de 15 qubits con **shots=1024**. Observamos que en este caso, la probabilidad sigue siendo alta, pero no es la más alta. Esto se debe al mayor ruido existente en un entorno real.

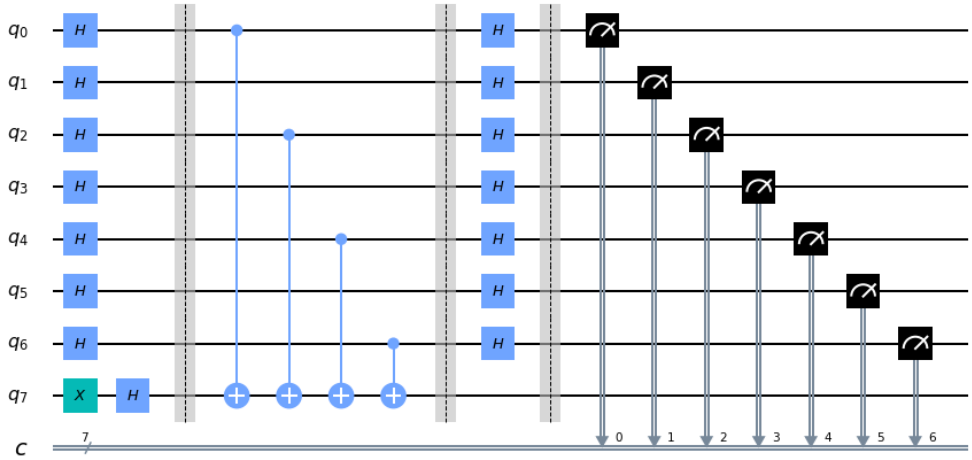

Figura 33: Circuito en Qiskit para **S='1010101'**

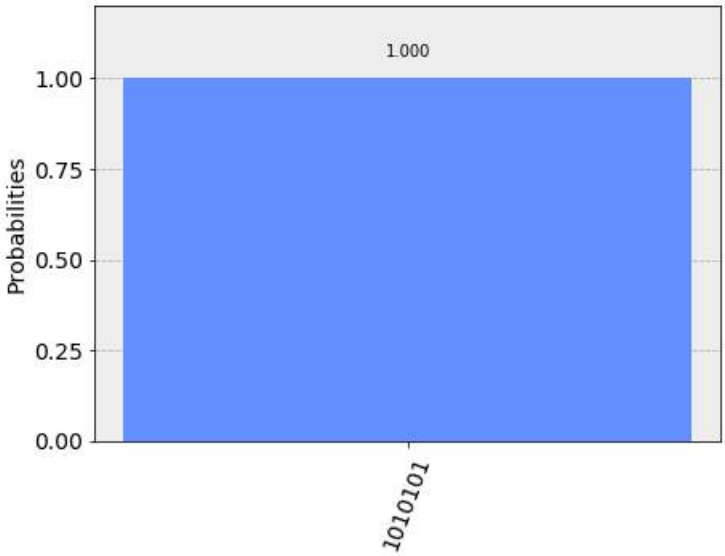

Figura 34: Resultado en entorno ideal con una comprobación **S='1010101'**

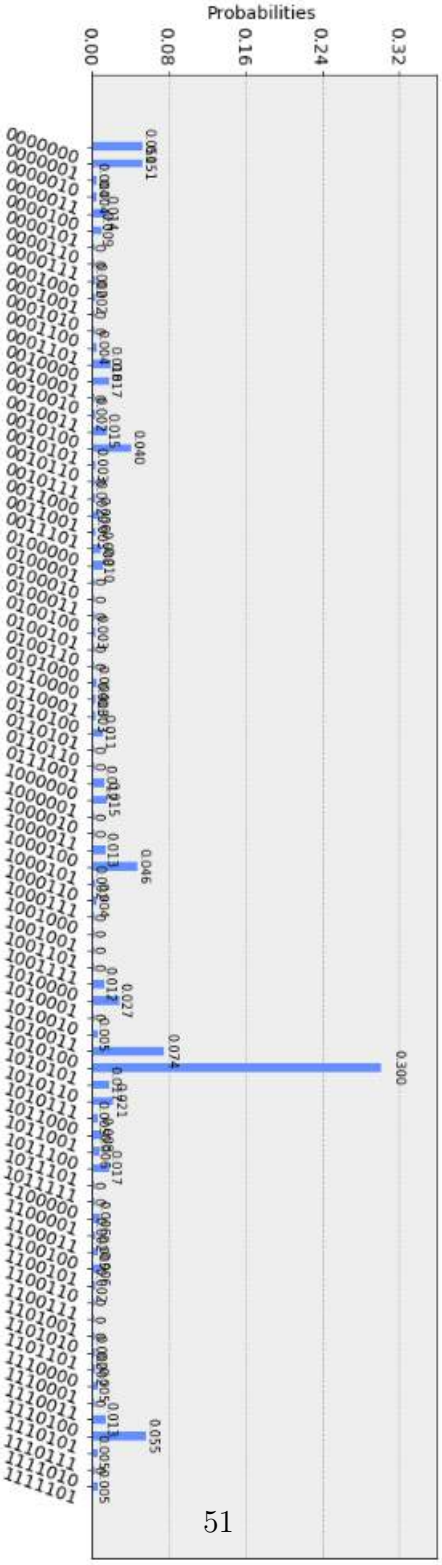

Figura 35: Resultado en un simulador con ruido **S='1010101'**

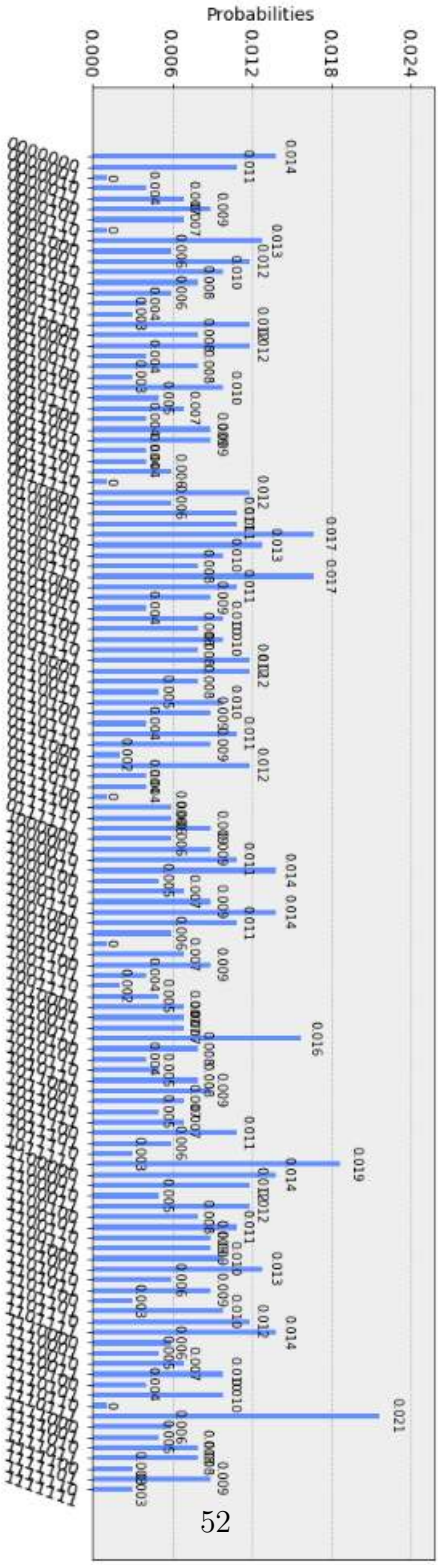

Figura 36: Resultado en un entorno real **S='1010101'**

6.2.3. Conclusiones sobre el algoritmo de Bernstein-Vazirani.

Con el algoritmo de Deutsch-Jozsa, introdujimos la computación cuántica y comprobamos la potencia de la superposición cuántica. Ahora con este algoritmo damos un salto en cuanto a la utilidad, implementado un algoritmo más aplicativo y útil como lo es este, en el cual comprobamos las utilidades que podemos conseguir al combinar diferentes puertas cuánticas. De esta forma encontramos la forma de aplicar CNOT en ambas direcciones con phase kickback, y la enorme usabilidad que tiene este concepto en el diseño de circuitos cuánticos. También experimentamos en este caso, una aceleración exponencial en cuanto al tiempo de ejecución, aplicando phase kickback.

6.3. Algoritmo de Shor.

Actualmente los cifrados de clave publica, como RSA, se basan en el producto de números primos de gran tamaño, del orden de 10^{300} y en aumento en función de la capacidad de cálculo de los ordenadores.

Para un número N muy grande, en la actualidad no se conoce un algoritmo que haga la descomposición eficientemente. Debido a esta dificultad algunos algoritmos de cifrado, utilizan este problema como núcleo en su cifrado.

El mejor algoritmo de factorización conocido, la criba general del cuerpo de números [11] tiene un orden de:

$$O(e^{(lnN)^{\frac{1}{3}}(lnlnN)^{\frac{2}{3}}}),$$

esto significa, que para superarlo una sola vez, para un número de un tamaño similar a 10^{15}, se tardaría aproximadamente 20330 años.

Mientras que el algoritmo de Shor [7], resuelve el problema en un tiempo del orden de $O((logN)^3)$, y en espacio $O(logN)$. Siendo N el número que se desea factorizar. De este modo, el tiempo necesario tan sólo sería de aproximadamente 1 hora.

El problema de factorización se plantea de una manera muy sencilla, dado un número impar no primo, ¿es posible encontrar dos factores primos p y q, tal que, $N = p * q$?

Para ello, primero convierte el problema de descomposición en factores al de encontrar el orden de una función $f(x) = a^x modN$. Y a continuación utilizar un algoritmo cuántico para encontrar el periodo de esta función.

6.3.1. Procedimiento: Parte clásica.

1. Escoger un número aleatorio a, tal que 1<a<N.

2. Calcular el Máximo Común Divisor MCD (a, N):

   Si $MCD(a, N) \neq 1$ devolvemos este resultado.

   Si no, utilizar el algoritmo cuántico para encontrar el periodo r más pequeño. De forma que se cumpla $f(x + r) = f(x)$ y r perteneciente a los enteros.

3. Si r es impar, volver al paso 1.

   Si no, calcular $MCD(a^{r/2} + 1)$ y $MCD(a^{r/2} - 1)$.

4. Si los dos resultados obtenidos son 1 o N, volver al paso 1.

   Si no, devolver el resultado diferente a 1 y N. Y calcular el factor faltante haciendo la división.

6.3.2. Procedimiento: Parte cuántica.

1. Crear un circuito con dos registros (registro de periodo y registro de cálculo) de tamaño $L = log_2 N$ qubits cada uno. Aproximando L al siguiente número entero.

2. Al primer registro, se aplica la puerta Hadamard quedando así en superposición.

3. Construir la función f(x) como función cuántica y aplicarla al estado anterior.

4. Aplicar la transformada IQFT al primer registro.

5. Realizar una medición sobre el primer registro.

6. Convertir la fracción $\frac{2}{2^L}$ en una fracción irreducible. Obtener el denominador r', que es un candidato a r.

7. Comprobar si $f(x + r') = f(x)$:

   Si se cumple la condición, termina.

   Si no, obtener más r' usando múltiplos de r' o valores cercanos de a.

Ejemplo: Factorizar N=33.

De forma clásica y utilizando el método mas simple, empezaríamos a hacer divisiones con los números menores $\sqrt{33} \simeq 6$, hasta tener una división con resto 0. Si no obtenemos una división exacta con ningún valor, entonces el número N es primo.

Para este ejemplo, obtenemos rápidamente un factor de N, el 3. Pero en el caso peor, se tendrían que hacer $\sqrt{N}$ comprobaciones.

Para el método de Shor tendríamos los siguientes pasos:

1. Escoger un número a, entre 1 y 33. En este ejemplo elegimos a=7.
2. $MCD(33, 7) = 1$, podemos continuar.
3. Buscamos el periodo r de la función f(x): (Cuadro 2).

x	$7^x mod 33$
0	1
1	7
2	16
3	13
4	25
5	10
6	4
7	28
8	31
9	19
10	1

Cuadro 2: Función modular.

4. Periodo r=10, r es par.
5. Se procede a calcular $MCD(7^{\frac{10}{2}} + 1, 33) = 11$ y $MCD(7^{\frac{10}{2}} - 1, 33) = 3$.

En el mejor caso, en el paso 2 encontraríamos un factor. Pero normalmente no tendremos tanta suerte. El coste de este algoritmo, se encuentra en el paso 3 que mencionamos anteriormente.

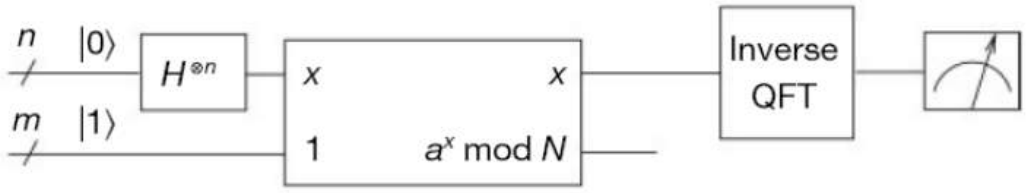

Figura 37: Diseño de Vandersypen et al.

6.3.3. Diseño de Vandersypen et al.

El diseño de la figura 37 [8], se compone por tres tipos de puertas cuánticas:

- Puertas Hadamard, que hacen que el registro periodo se encuentre en superposición.
- Puertas controladas de multiplicación modular, que sirven en conjunto para crear la función $f(x) = a^x mod N$. Para cada parte $a^{2*i} mod N$, siendo i el índice del qubit que activa la puerta.
- Puerta IQFT, sirve para salir del estado de superposición y poder observar el resultado.

6.3.4. Algoritmo de Shor según la propuesta de Kitaev.

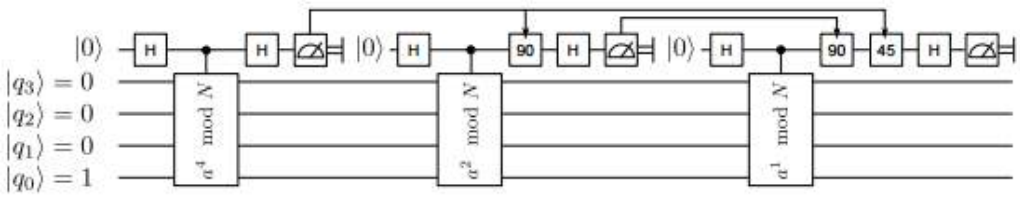

Figura 38: Diseño de Kitaev.

Según Kitaev [9], si solo el resultado de la IQFT es de interés, los n qubits de la IQFT pueden reemplazarse por un solo qubit. Este enfoque requiere la lectura de este qubit y posteriormente, reiniciar el estado de éste. Debido a esto la QFT de Kitaev se denominará KQFT, que reemplaza una QFT

que actúa sobre N qubits, por un QFT semiclásico, que actúa repetidamente sobre un único qubit. Podemos ver el diseño del circuito en la figura 38 [12].

Por otro lado, el estado $|1\rangle$ (en notación decimal), está sujeto a una multiplicación modular condicionada por el qubit más significativo del registro de periodo. Por lo tanto, como mucho habrá dos posibles resultados, 1 o $a^{2^k} mod N$, así que solo es necesario crear una puerta condicionada que realice la operación de convertir $|1\rangle$ a $|a^{2^k} mod N\rangle$, a ser posible de la manera más eficiente.

Las multiplicaciones siguientes, se pueden reemplazar de manera similar. Teniendo en cuenta todos los posibles resultados que se obtendrían en la multiplicación anterior. Algo que se vuelve una tarea hercúlea, ya que el número de estados a considerar aumenta exponencialmente. En consecuencia, se debería intentar hacer multiplicadores modulares completos, en la medida que sea posible.

Finalmente, el último multiplicador solo tiene que crear el número correcto de correlaciones entre el registro de periodo y el registro de cálculo.

6.3.5. Implementación del algoritmo de Shor en Qiskit.

Ejemplo N=21,a=5 (apéndice I).

Utilizamos el diseño de Vandersypen et al para este ejemplo. Necesitamos para empezar, dos registros de $log_2(21) = 5$ qubits cada uno, haciendo un total de 10 qubits.

Al primer registro, inicializado a cero, se aplica una puerta Hadamard. Y al segundo registro, al que llamaremos registro cálculo, lo inicializaremos a uno.

Como paso previo para el registro de cálculo, veamos qué multiplicaciones son necesarias para crear la correcta correlación entre los dos registros (cuadro 3).

Por lo tanto, en el registro de cálculo necesitaremos multiplicar por 5, 4 o 16:

Para conseguir esto, se opta por:

- Una puerta incompleta multiplicadora modular por cuatro (figura 39).
- Una puerta incompleta multiplicadora modular por dieciséis (figura 40).
- Y otra puerta que hace pasar del estado $|1\rangle$ al estado $|5\rangle$.

x	$5^x mod 21$
1	5
2	4
4	16
8	4
16	16

Cuadro 3: Función 5 modular 21.

Una puerta incompleta, es una puerta que para todos los posibles valores de entrada esperados, se comporta como una puerta completa, pero no devuelve los valores correctos de los valores de entrada inesperados. Por ejemplo, consideremos una puerta que hace la operación de raíz cuadrada. Una puerta incompleta haría las raíces perfectas como $\sqrt{9} = 3$ sin ningún problema, pero si se intenta hacer $\sqrt{7}$ te da como resultado 3. Por el contrario, una puerta completa devuelve el resultado más aproximado $\sqrt{7} = 2,64$

Por último, se aplicará la QFT al registro periodo y tomar una medición (figura 41).

En la figura 42, se puede ver el circuito completo.

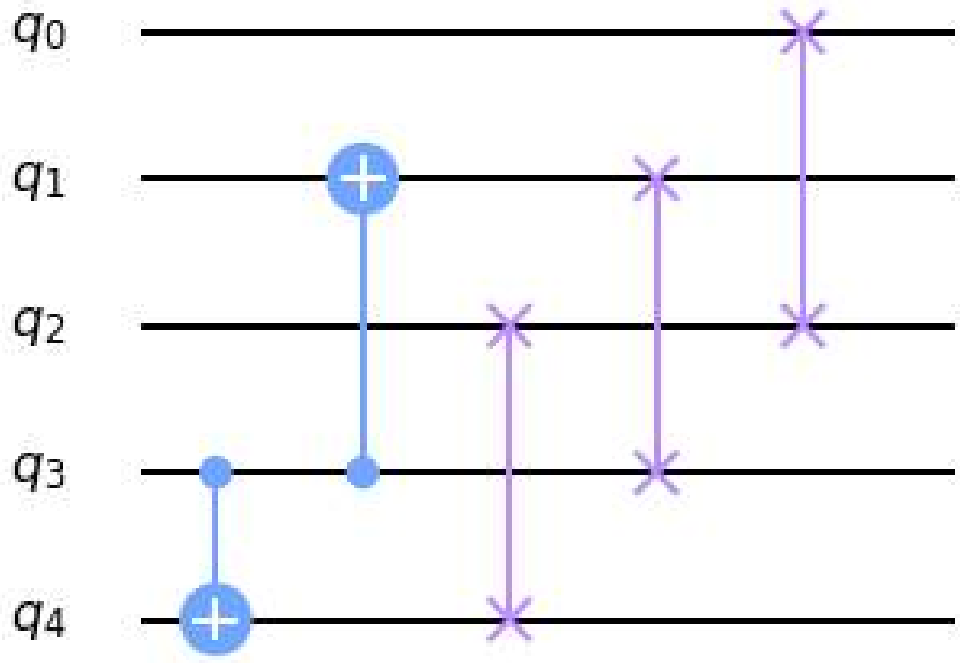

Figura 39: Multiplicador modular por 4

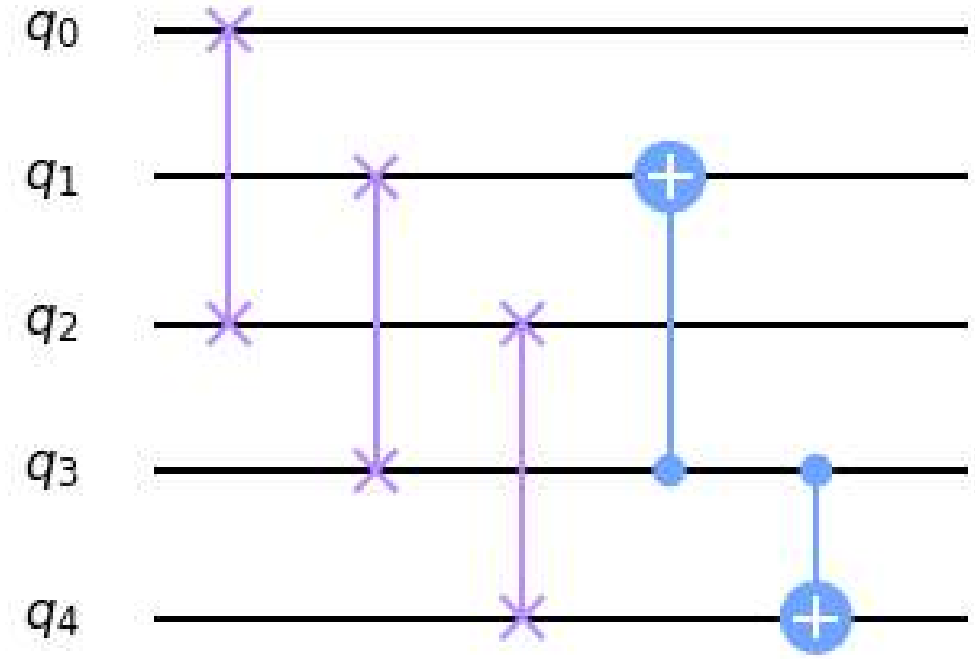

Figura 40: Multiplicador modular por 16.

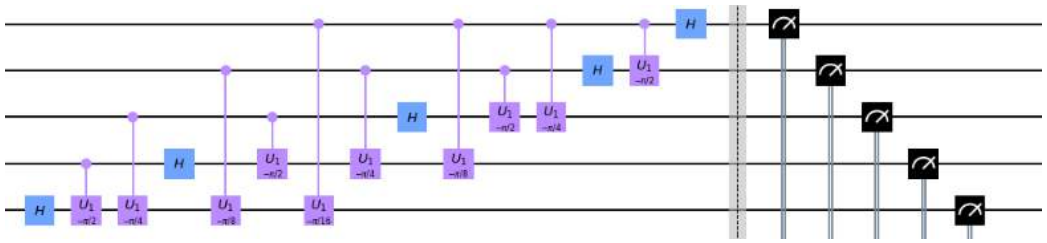

Figura 41: QFT sobre 5 qubits.

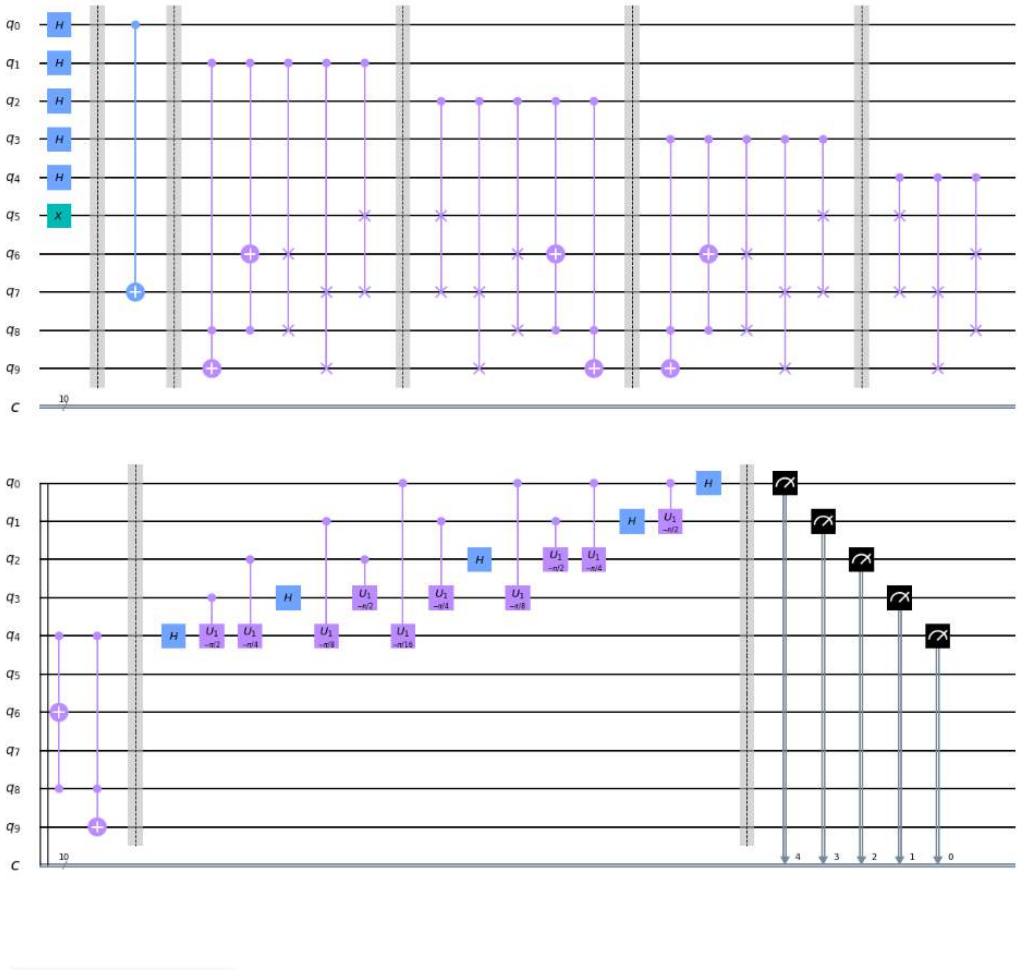

Figura 42: Circuito completo.

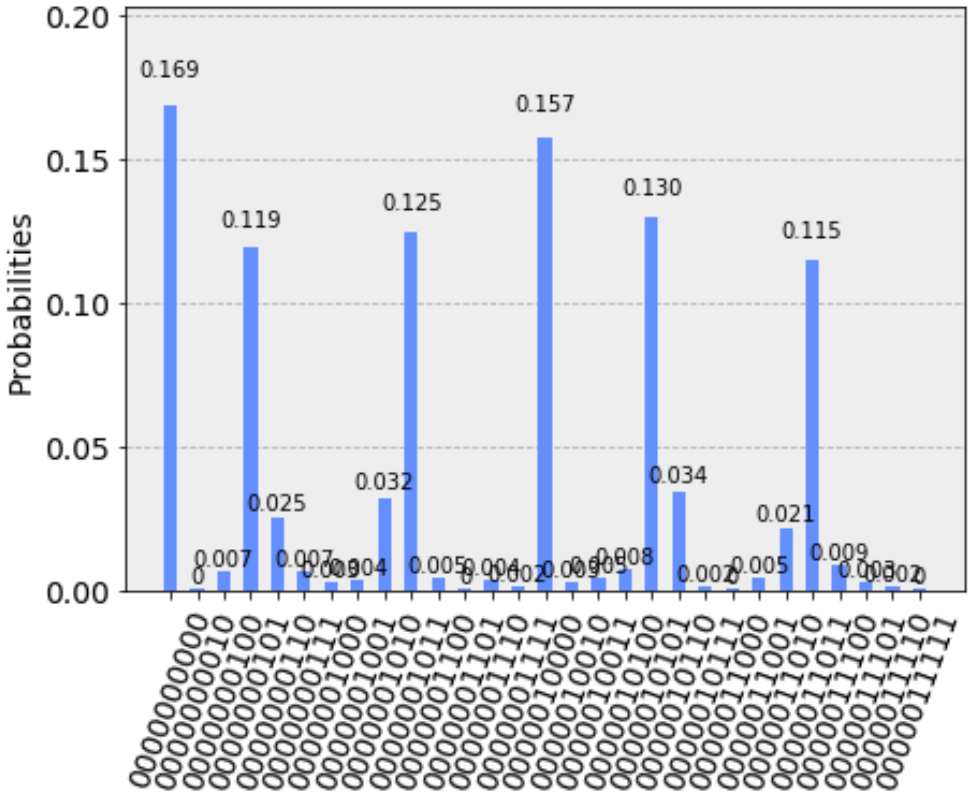

Figura 43: Resultado ideal.

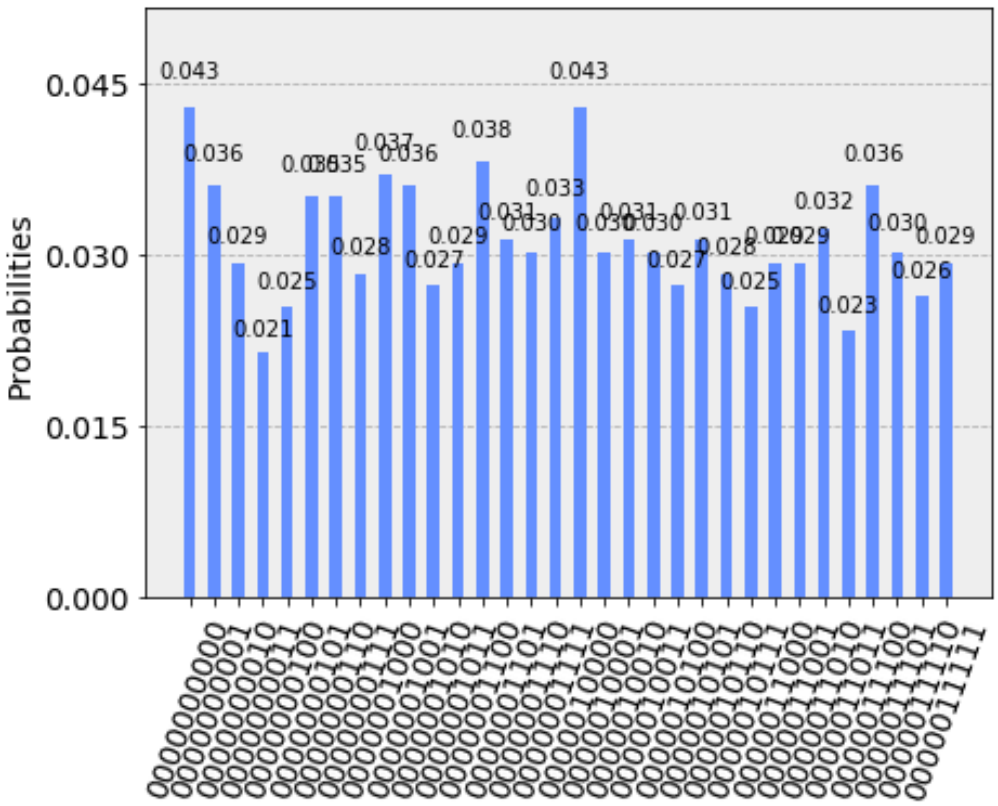

Figura 44: Resultado con ruido.

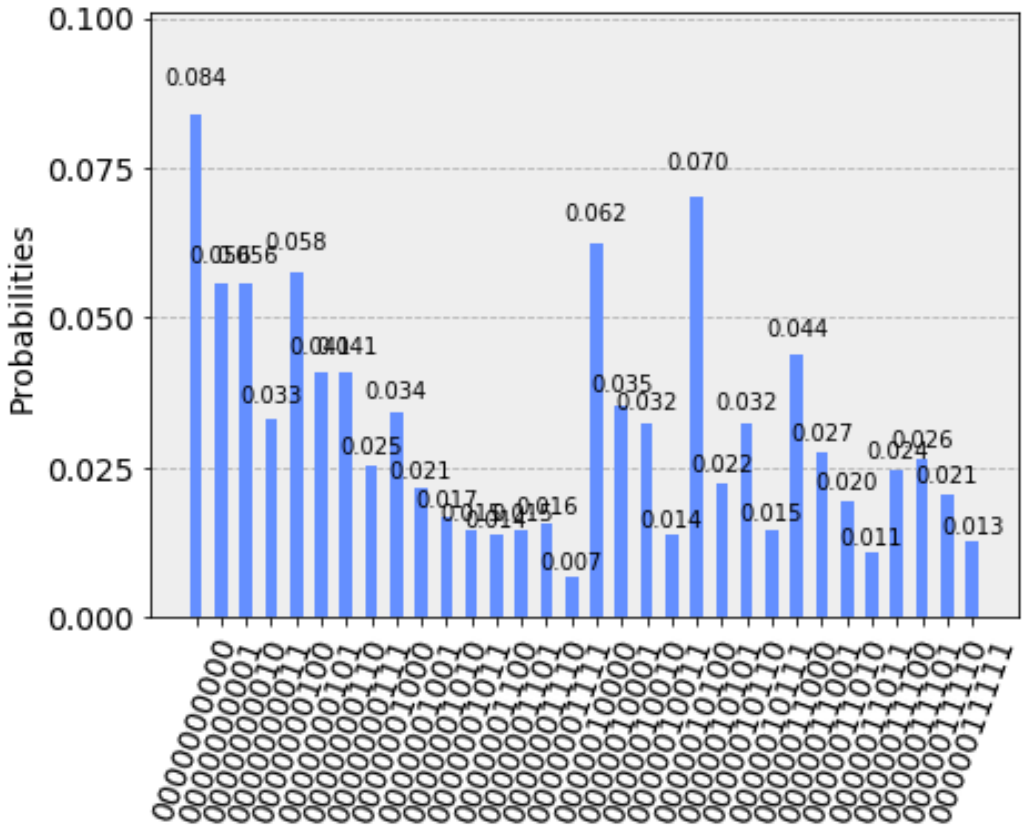

Figura 45: Resultado en un ordenador cuántico real.

Y	Probabilidad	$\frac{Y}{2^L} = \frac{Y}{32}$	r'
0	0.169	$\frac{0}{32}$	32
5	0.119	$\frac{5}{32}$	32
11	0.125	$\frac{11}{32}$	32
16	0.157	$\frac{1}{2}$	2
21	0.130	$\frac{21}{32}$	32
27	0.115	$\frac{27}{32}$	32

Cuadro 4: Probabilidades de la figura 43.

En las figuras 43, 44 y 45 se muestra la ejecución de este ejemplo en tres entornos distintos: un simulador ideal, un simulador con ruido y el ordenador cuántico real ibmq-16-melbourne respectivamente.

Resultados en un simulador ideal mostrados en la figura 43:

Cogiendo las estadísticas de la figura 43, fijándonos en las barras que sobresalen sobre las demás, se obtiene el cuadro 4. Los únicos candidatos a r, son 2 y 32. Ahora comprobamos si son unos resultados validos.

$f(x) \neq f(x+2)$ y $f(x) \neq f(x+32)$ no son válidos como periodos. Así que hay que buscar otros a partir de sus múltiplos.

$f(x) = f(x+6)$ y $f(x) = f(x+96)$ sí son válidos.

Ya que, $5^{x+6} \mod 21 = (5^6 \mod 21) * (5^x \mod 21) = 1 * (5^x \mod 21)$.

Y $5^{x+96} \mod 21 = (5^{96} \mod 21) * (5^x \mod 21) = 1 * (5^x \mod 21)$.

Como son pares seguimos con el último paso.

$MCD(5^{\frac{6}{2}+1}, 21) = 21$ y $MCD(5^{\frac{6}{2}-1}, 21) = 1$, por lo tanto, hay que volver al paso 1, elegir un nuevo número a.

En el simulador con ruido (figura 44). Se puede apreciar cómo las probabilidades se encuentran más uniformemente repartidas, haciendo más difícil obtener resultados fiables. Esto es debido a la gran cantidad de puertas utilizadas que van acumulado interferencias.

En el entorno real de ibmq-16-melbourne (figura 45). Ocurre el mismo efecto que en el simulador con ruido.

Ejemplo N=21,a=8 (apéndice D).

Con el diseño de Kitaev, podemos reducir el número de qubits necesarios para implementar el algoritmo a 6 qubits, 4 menos de los que necesitaríamos con el diseño de Vandersypen et al.

Como se puede ver en la figura 46, al elegir bien los números como es este el caso, N=21 y a=8, en los bloques con contorno rojo no hay ninguna puerta debido a que en esos bloques se debería multiplicar por $8^{2^i} mod 21 = 1$, es decir, no hacer nada.

La única puerta necesaria, es la puerta que hace pasar del estado 1 al estado 8.

Los resultados en el simulador ideal (figura 47). Dan como solución 0 y 16, con similar probabilidad del 50 %.

Al hacer la fracción irreducible $\frac{16}{32} = \frac{1}{2}$, se obtiene como candidato a periodo r=2, que es par. Se comprueba que la función $f(x) = 8^x mod 21$ tiene como periodo 2, al comprobar que $f(x) = f(x+2)$.

Como último paso, obtenemos los factores al realizar el Máximo Común Denominador $MCD(8^{\frac{2}{2}} + 1, 21) = MCD(9, 21) = 3$ y $MCD(8^{\frac{2}{2}} - 1, 21) = MCD(7, 21) = 7$. El resultado esperado $3 * 7 = 21$ es igual al obtenido.

En el simulador con ruido (figura 48), aparecen más resultados, pero con una probabilidad bastante inferior e insignificante. Básicamente obtenemos los mismos resultados que en el simulador ideal, 0 y 16 aunque con una probabilidad menor.

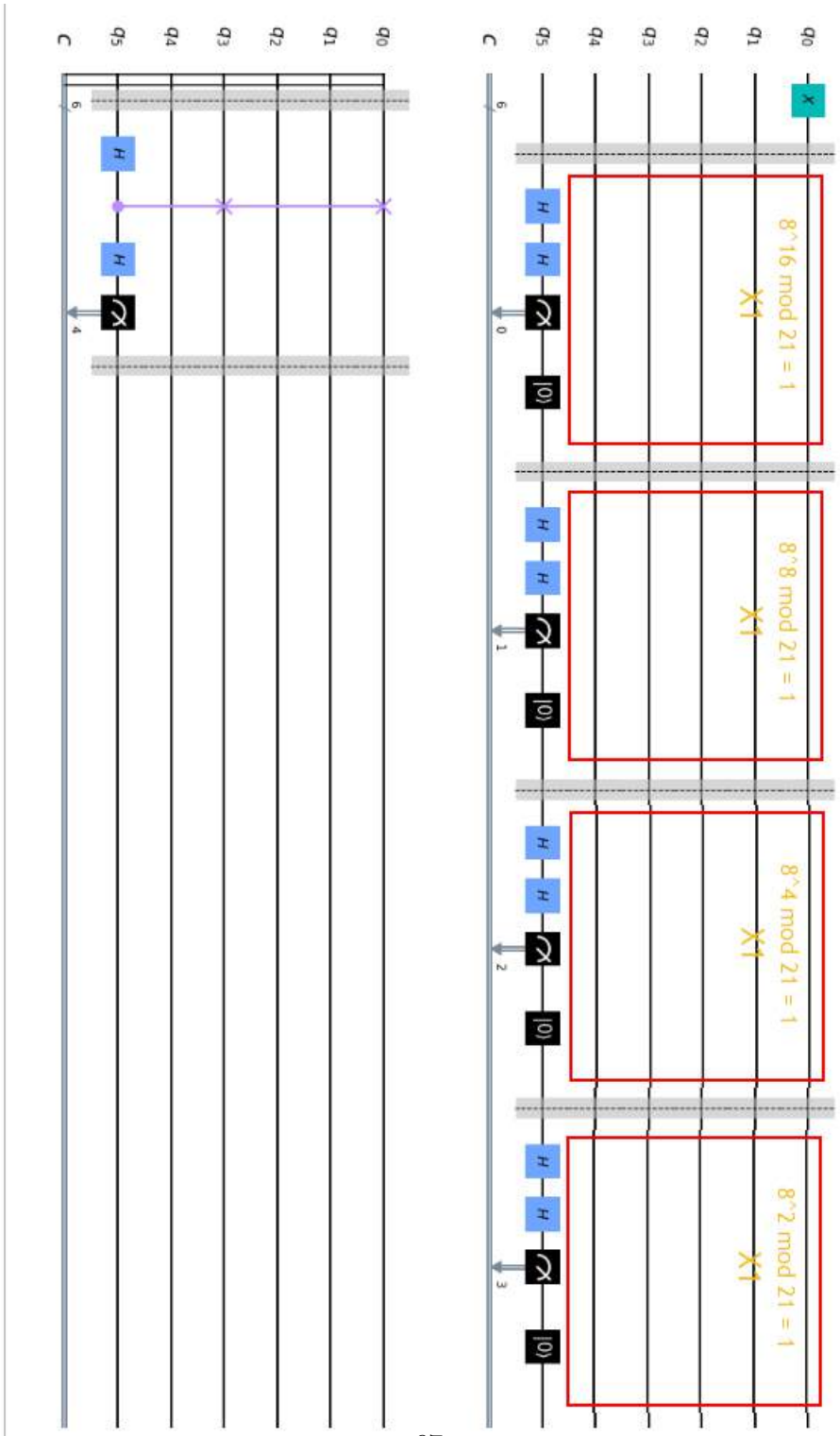

Figura 46: Implementación basada en el diseño de Kitaev.

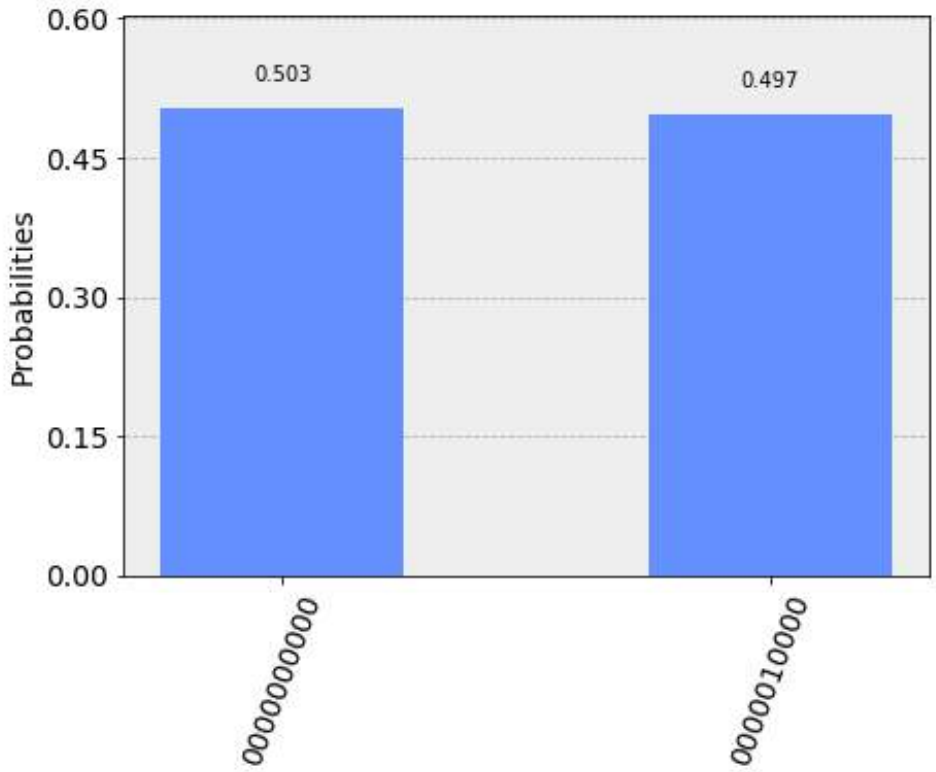

Figura 47: Resultado ideal del modelo de Kitaev.

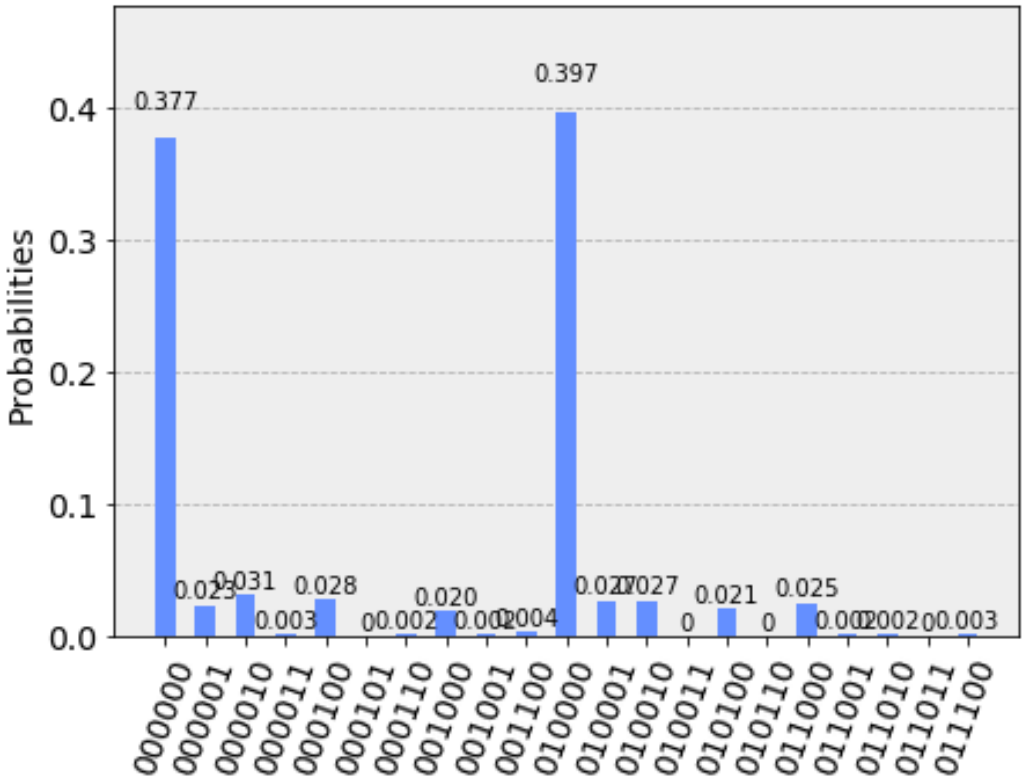

Figura 48: Resultado en el simulador con ruido.

6.3.6. Reflexiones sobre el algoritmo de Shor.

Una función recibe por parámetro un número 'N', se calcula un número 'a' coprimo a 'N' y con un bucle iterativo se van guardando los resultados de la ecuación $a^i mod N$ en una lista, incrementando la i hasta que se intente guardar el mismo número. De esta forma el tamaño de la lista es el periodo de la función. Haciendo esto, el algoritmo de Shor es bastante efectivo, pero se pierde la velocidad que proporciona la superposición que brinda la computación cuántica.

Además, el algoritmo de Shor está sujeto a cierta aleatoriedad, que puede hacer que se obtengan resultados no deseados. Como consecuencia de que se trate de un algoritmo probabilístico.

A día de hoy, a la pregunta ¿Shor puede romper el cifrado RSA? La respuesta es no. Debido a que el número de qubits está limitado y que cuantos más qubits se añaden, más ruido existe y más imprecisos son los resultados obtenidos. Además, la construcción de las puertas multiplicadoras modulares agrava este problema.

Por todo esto, la respuesta es no. Pero con los últimos progresos en computación cuántica, tal vez en futuro no muy lejano la respuesta cambie, al igual que el método de cifrado.

6.4. Algoritmo de Grover.

En la búsqueda de un elemento dentro de un conjunto desordenado de tamaño N, tendríamos que ir uno por uno hasta encontrar el elemento deseado. Esto conlleva que la tarea de búsqueda tenga un coste lineal O(N). Por lo tanto, si suponemos un conjunto de 500 elementos y tardásemos un segundo en analizar si un elemento es el correcto para la búsqueda. En el peor de los casos se tardaría 500 segundos. Y en la mayoría de los casos se tardaría una de media 250 segundos.

Con el algoritmo de Grover [13] [14], la tarea de búsqueda se vería reducida su ejecución. Para el mismo supuesto anterior, se tardaría $\sqrt{500} \simeq 23$ segundos. Es decir, tendría un coste de $O(\sqrt{N})$, gracias a la propiedad de la superposición que ofrecen los ordenadores cuánticos.

¿Cómo funciona el algoritmo de Grover? Imaginemos una bolsa de canicas, entre las cuales se encuentra una única canica de color rojo, que es la que queremos encontrar. Primero, se construye una función que devuelve 1, si la canica es de color rojo. Y devuelve 0, en caso contrario. A

esta función de aquí en adelante, se le llamará función oráculo. Al pasar un estado en superposición por esta función, la canica roja queda marcada y es encontrada.

6.4.1. Procedimiento.

El primer paso para el algoritmo de Grover, es poner en superposición los qubits aplicando puertas Hadamard en todos ellos (figura 49). Obteniendo el siguiente estado al que llamaremos 's'.

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^N} |x\rangle \,.$$

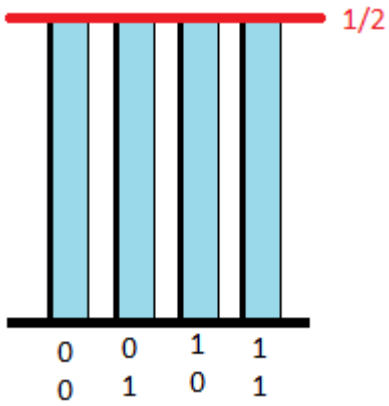

Figura 49: Hadamard sobre 2 qubits.

El segundo paso, es marcar el elemento a buscar. Para ello se construye una puerta cuántica, que hará de oráculo. Este oráculo invierte la probabilidad del estado que representa el elemento buscado $|w\rangle$. Haciendo que todos los elementos tengan la misma probabilidad, a excepción del elemento que se busca, que tiene probabilidad negativa (figura 50).

$$|\psi\rangle = |s\rangle - \frac{2}{\sqrt{N}} |w\rangle \,.$$

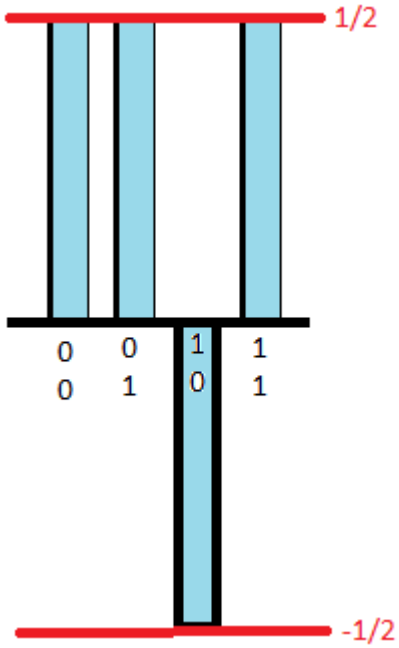

Figura 50: Aplicación del oráculo.

Posteriormente, se aplica lo que se llama operando de Grover, que realiza la simetría respecto a la media de las probabilidades. Lo que significa que todas las probabilidades van a bajar un poco, menos la que estaba invertida, que se ve amplificada (figura 51).

$$|\psi\rangle = \frac{1}{N\sqrt{N}} \left((N-4) \sum_{x \neq w} |x\rangle + (3N-4) |w\rangle \right).$$

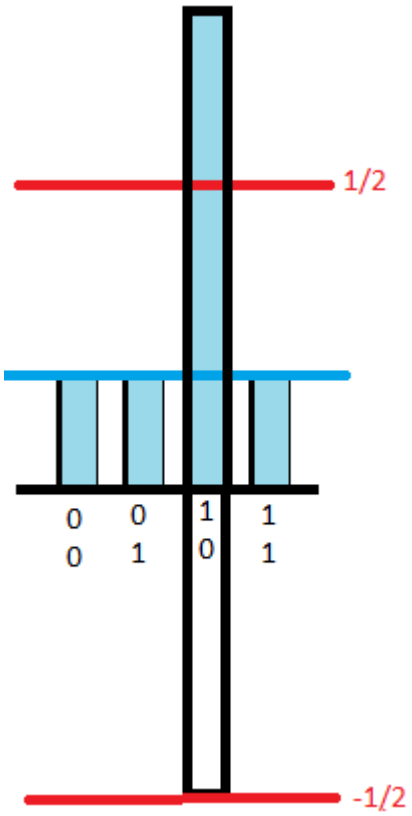

Figura 51: Resultado final.

En la figura 52, se puede ver el circuito general del algoritmo de Grover.

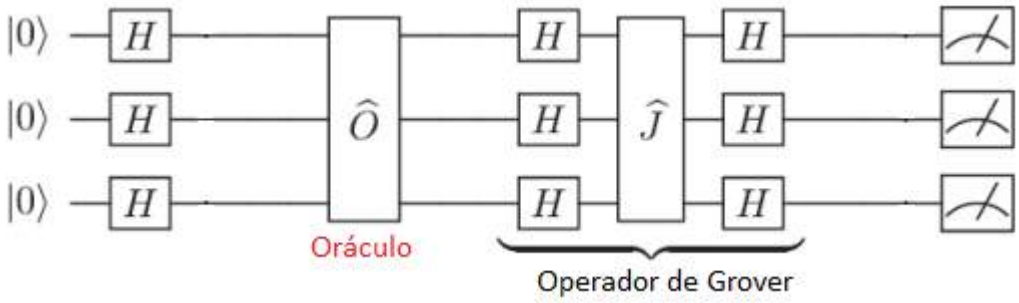

Figura 52: Circuito de Grover.

6.4.2. Implementación para 6 qubits (apéndice E).

En esta implementación, se tiene como objetivo buscar el elemento $|110010\rangle$. Para ello se construye la puerta oráculo, que invierte la probabilidad de este estado. Para ello, se descompone el estado buscado en grupos de dos qubits. En función de estos dos qubits, se añaden puertas X y puertas controladas Z para codificarlos. De este modo para buscar el elemento $|110010\rangle$, el oráculo quedaría como el la figura 53.

Además, se construye el operando de Grover que hace la inversión sobre la media (figura 54). Para la construcción del operando de Grover, se aplican puertas Hadamard y puertas Z en paralelo a todos los qubits. Después, se aplican puertas controladas Z, cuyo control son los qubits pares y objetivo los qubits impares. Finalmente, se aplica otra vez puertas Hadamard en paralelo a todos los qubits.

Se puede observar el circuito completo en la figura 55.

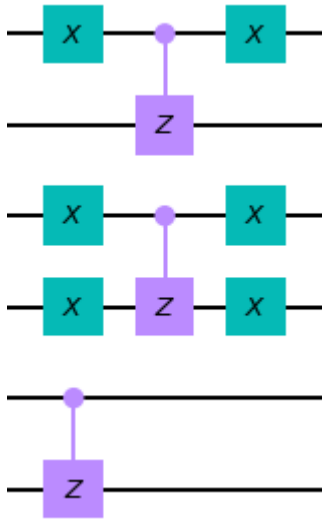

Figura 53: Oráculo para el estado $|110010\rangle$.

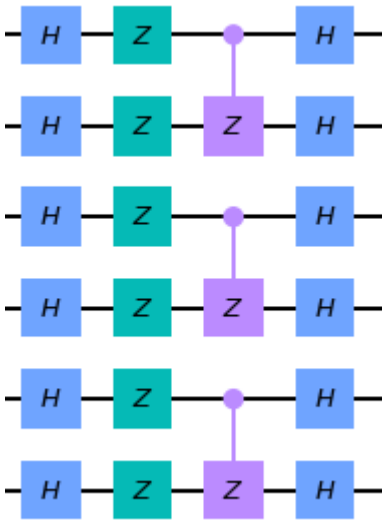

Figura 54: Inversión sobre la media.

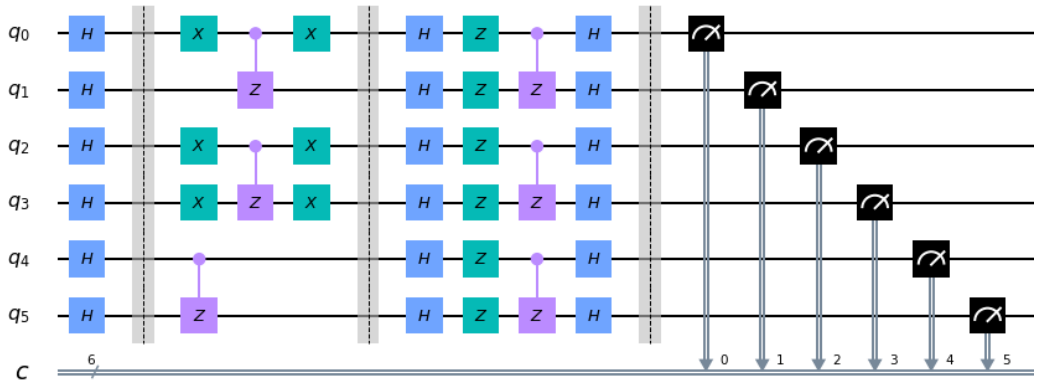

Figura 55: Circuito completo.

Ahora, se procede a la ejecución del ejemplo en el simulador ideal (figura 56), el simulador con ruido (figura 57), y el ordenador cuántico real ibmq-16-melbourne (figura 58).

En los tres entornos se obtiene el mismo resultado, el estado esperado 110010. En el entorno ideal solo se obtiene como resultado el estado $|110010\rangle$ con probabilidad 100 %, tal y como se debería esperar. No obstante, en el entorno con ruido y en el entorno real, aparecen resultados inesperados con una probabilidad muy por debajo a la del estado buscado $|110010\rangle$. Esto es debido a la existencia de ruido en estos dos entornos.

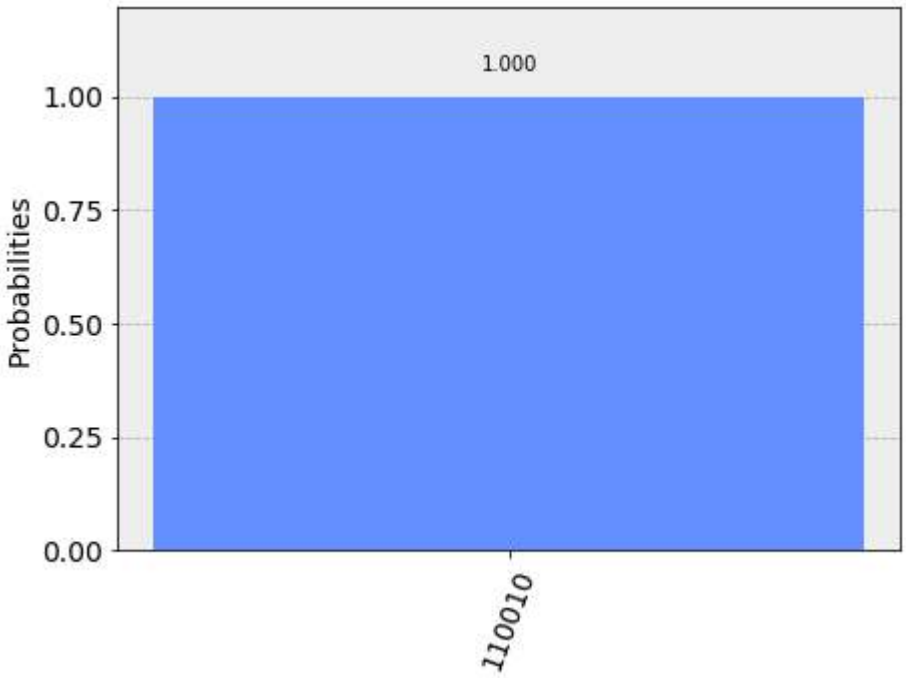

Figura 56: Resultados del simulador ideal.

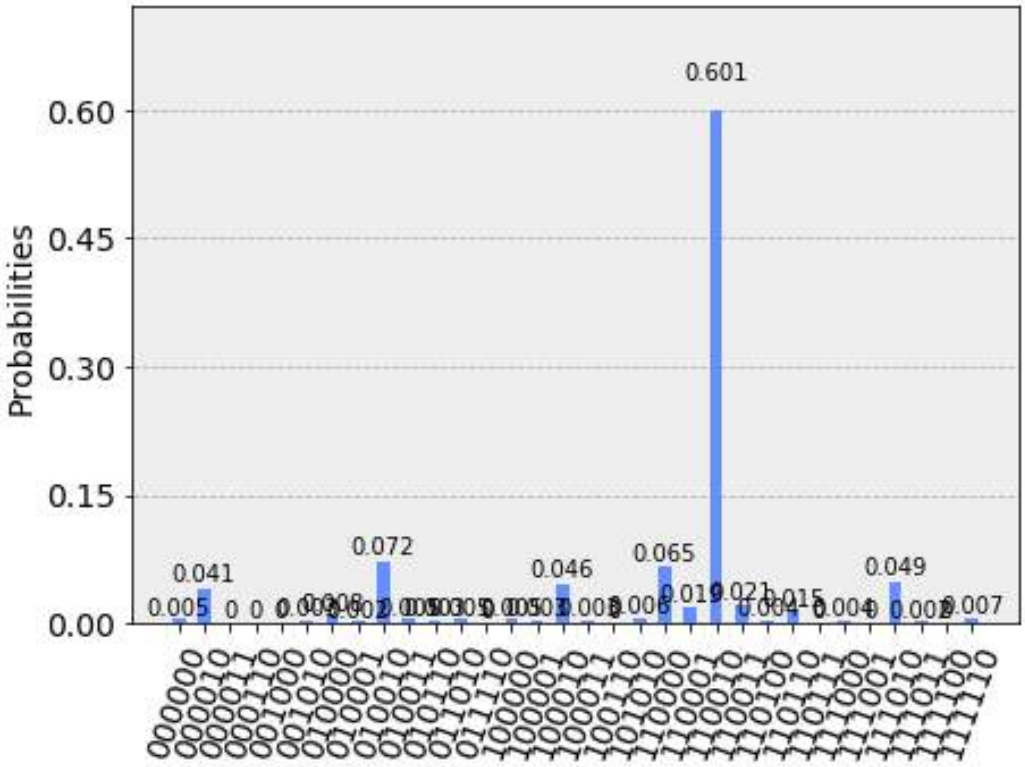

Figura 57: Resultados del simulador con ruido.

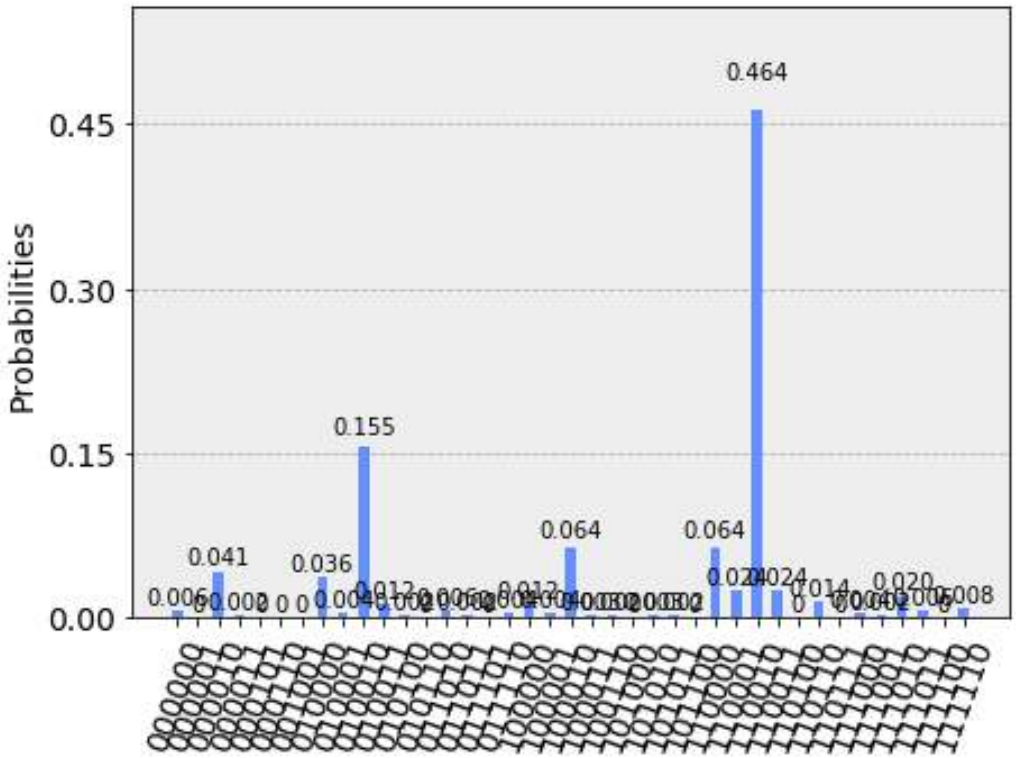

Figura 58: Resultados del ordenador cuántico real.

6.4.3. Conclusiones sobre el algoritmo de Grover.

Usando el algoritmo de Grover se obtienen buenos resultados, además de que se pueden mejorar los resultados repitiendo el oráculo y el operando de Grover. No obstante, para que funcione sobre una base de datos convencional, habría que precargar toda la base de datos en qubits, lo cual equivale a leerla entera.

El algoritmo de Grover, mejora notablemente la búsqueda en secuencia de datos desordenada, ya que evita la necesidad de ordenación previa.

Sin embargo, al tratarse de un algoritmo probabilístico, produce la respuesta correcta con una probabilidad de error. Aunque este error pueda reducirse por medio de iteraciones.

7. Conclusiones finales.

7.1. Castellano.

Con las implementaciones de los algoritmos de Deutsch-Jozsa, Bernstein-Vazirani y Grover, aprendimos los conceptos básicos de la computación cuántica y obtuvimos experiencia con las herramientas utilizadas, en especial con Qiskit. Obtuvimos una visión más específica de los avances que se han hecho en el campo de la computación cuántica y en qué punto se encuentra su desarrollo.

Por otro lado, en un primer momento, se planteó crear una implementación del algoritmo de Shor que pudiera factorizar un número cualquiera menor que 100 pasado como parámetro de una función, pero debido a la enorme complejidad de crear puertas cuánticas que pudiesen hacer la multiplicación modular, esta idea se descartó y se empezó el diseño de un algoritmo que pudiese factorizar un número fijo. Con esto en mente, se comenzó con el desarrollo e investigación dando como frutos los resultados mostrados y la comprensión de las limitaciones actuales de la computación cuántica.

Por último, mencionar que todo el código se encuentra en el repositorio de github, accesible a través de https://github.com/jairo84/Algoritmos-Cuanticos.git bajo licencia Crative Commons, para su estudio y análisis.

7.2. English

With the help of Deutsch-Jozsa, Bernstein-Vazirani and Grover algorithms, we learned the basic concept of quantum computing and experience with tools, especially with Qiskit. We obtained a more specific view of the progress of the use of quantum computing and where its development is located.

On the other hand, it was initially proposed to build an application of the Shor algorithm that could treat less than 100 as a single parameter parameter, but due to the great complexity of creating multiplier quantum gates. Due to this, this idea was ignored and the design of an algorithm that could factor a fixed number began. With this in mind, he began development and research, giving the presented results and understanding of the current limitations of quantum computing.

Lastly, mention that all the code is in the github repository accessible through https://github.com/jairo84/Algoritmos-Cuanticos.git licensed Crative Commons, for study and analysis.

8. Bibliografía.

Referencias

[1] Feynman, R. P. (1982). Simulating physics with computers. Int. J. Theor. Phys, 21(6/7).

[2] Cortese, J. A., Braje, T. M. (2018). Loading classical data into a quantum computer. arXiv preprint arXiv:1803.01958.

[3] Wolf, E. (2007). The influence of Young's interference experiment on the development of statistical optics. Progress in Optics, 50, 251-273.

[4] Documentación de Qiskit. https://qiskit.org/documentation/

[5] Coppersmith, D. (2002). An approximate Fourier transform useful in quantum factoring. arXiv preprint quant-ph/0201067.

[6] Micheli, G. D. (1994). Synthesis and optimization of digital circuits. McGraw-Hill Higher Education.

[7] Shor, P. W. (1994, November). Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th annual symposium on foundations of computer science (pp. 124-134). Ieee.

[8] Vandersypen, L. M., Steffen, M., Breyta, G., Yannoni, C. S., Sherwood, M. H., Chuang, I. L. (2001). Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature, 414(6866), 883-887.

[9] Kitaev, A. Y. (1995). Quantum measurements and the Abelian stabilizer problem. arXiv preprint quant-ph/9511026.

[10] Nielsen, M. A., Chuang, I. L. (2000). Quantum information and quantum computation. Cambridge: Cambridge University Press, 2(8), 23.

[11] Lenstra, A. K., Hendrik Jr, W. (1993). The development of the number field sieve (Vol. 1554). Springer Science Business Media.

[12] Monz, T., Nigg, D., Martinez, E. A., Brandl, M. F., Schindler, P., Rines, R., ... Blatt, R. (2016). Realization of a scalable Shor algorithm. Science, 351(6277), 1068-1070.

[13] Grover, L. K. (1996, July). A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (pp. 212-219).

[14] Figgatt, C., Maslov, D., Landsman, K. A., Linke, N. M., Debnath, S., Monroe, C. (2017). Complete 3-Qubit Grover search on a programmable quantum computer. Nature communications, 8(1), 1-9.

[15] Documentación de Cirq: https://cirq.readthedocs.io/en/stable/

[16] McMahon, D. (2007). Quantum computing explained. John Wiley Sons.

[17] Du, J., Shi, M., Zhou, X., Fan, Y., Ye, B., Han, R., Wu, J. (2001). Implementation of a quantum algorithm to solve the Bernstein-Vazirani parity problem without entanglement on an ensemble quantum computer. Physical Review A, 64(4), 042306.

[18] Halstead, M. H. 1977. Elements of Software Science.

Apéndices:

A. ADJZ 3 qubits en Qiskit.

```
from qiskit import *
# Registro cuantico 3 qubits
qr=QuantumRegister(3)
# registro clasico donde almacenaremos los resultados
cr=ClassicalRegister(3)
circuit=QuantumCircuit(qr,cr)
# herramientas para la visuzaliacion
get_ipython().run_line_magic('matplotlib', 'inline')
circuit.draw()
# Transformada de Hadamard
circuit.h(qr[0])
circuit.h(qr[1])
circuit.h(qr[2])

circuit.draw()
circuit.draw(output='mpl')

# aplicamos la query function
circuit.z(qr[0])
circuit.h(qr[2])
circuit.cx(qr[1],qr[2])
circuit.h(qr[2])

circuit.draw(output='mpl')
# transformada de Hadamard
circuit.h(qr[0])
circuit.h(qr[1])
circuit.h(qr[2])

circuit.draw(output='mpl')
```

```
circuit.measure(qr,cr)

circuit.draw(output='mpl')
# Procedemos a ejecutar en entorno ideal
simulator=Aer.get_backend('qasm_simulator')
result=execute(circuit,backend=simulator).result()
# visualizacion resultado
from qiskit.tools.visualization import plot_histogram
plot_histogram(result.get_counts(circuit))

from qiskit.tools.visualization import plot_bloch_multivector
plot_bloch_multivector(result.get_counts(circuit))

from qiskit.tools.visualization import plot_gate_map

plot_gate_map(result.get_counts(circuit))
# ejecucion en entorno real

IBMQ.load_account()
provider=IBMQ.get_provider('ibm-q')

qcomp=provider.get_backend('ibmq_london')

job=execute(circuit,backend=qcomp)

from qiskit.tools.monitor import job_monitor

job_monitor(job)
result=job.result()
plot_histogram(result.get_counts(circuit))
```

B. ABV genérico.

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:
from qiskit import *
get_ipython().run_line_magic('matplotlib', 'inline')
from qiskit.tools.visualization import plot_histogram

# In[2]:
numero_secreto='1010101'
# In[4]:
circuit=QuantumCircuit(len(numero_secreto)+1,len(numero_secreto))

#aplicamos transformada de Hadamard al registro de entrada
circuit.h(range(len(numero_secreto)))

circuit.x(len(numero_secreto))
circuit.h(len(numero_secreto))

circuit.barrier()

for i, val in enumerate(reversed(numero_secreto)):
    if val=='1':
        circuit.cx(i,len(numero_secreto))


circuit.barrier()

circuit.h(range(len(numero_secreto)))

circuit.barrier()

circuit.measure(range(len(numero_secreto)),range(len(numero_secreto)))
# In[5]:
circuit.draw(output='mpl')
```

```
# In[6]:
simulador=Aer.get_backend('qasm_simulator')
res=execute(circuit,backend=simulador,shots=1).result()
plot_histogram(res.get_counts(circuit))
# In[8]:
IBMQ.load_account()
# In[9]:
provider=IBMQ.get_provider('ibm-q')
# In[10]:
qcomp=provider.get_backend('ibmq_16_melbourne')
# In[11]:
job=execute(circuit,backend=qcomp)
# In[12]:
from qiskit.tools.monitor import job_monitor
# In[13]:
job_monitor(job)
# In[14]:
resultado1=job.result()
# In[15]:
plot_histogram(resultado1.get_counts(circuit))
# In[16]:
job2=execute(circuit,backend=qcomp,shots=1)
# In[17]:
job_monitor(job2)
# In[18]:
resultado2=job2.result()
# In[19]:
plot_histogram(resultado2.get_counts(circuit))
# In[ ]:
```

C. Código Shor N=21,A=5 en Qiskit.

```
from qiskit import *;
from qiskit.tools.visualization import plot_histogram;
import math;
circuit = QuantumCircuit(10,10)
for i in range(0,5):
    circuit.h(i)
circuit.x(5)
circuit.barrier()
#si el qbit 0 esta a 1 se realiza 1X5
circuit.cx(0,7)
circuit.barrier()
# 5^2mod21 = X4
circuit.ccx(1,8,9)
circuit.ccx(1,8,6)
circuit.cswap(1,8,6)
circuit.cswap(1,7,9)
circuit.cswap(1,5,7)
circuit.barrier()
# 5^4mod21 = x16
circuit.cswap(2,5,7)
circuit.cswap(2,7,9)
circuit.cswap(2,8,6)
circuit.ccx(2,8,6)
circuit.ccx(2,8,9)
circuit.barrier()
# 5^8mod21 = X4
circuit.ccx(3,8,9)
circuit.ccx(3,8,6)
circuit.cswap(3,8,6)
circuit.cswap(3,7,9)
circuit.cswap(3,5,7)
circuit.barrier()
# 5^16mod21 = X16
circuit.cswap(4,5,7)
circuit.cswap(4,7,9)
```

```
circuit.cswap(4,8,6)
circuit.ccx(4,8,6)
circuit.ccx(4,8,9)
circuit.barrier()
for j in range(4,-1,-1):
    circuit.h(j)
    for k in range(j-1,-1,-1):
        circuit.cu1(-1*math.pi/float(2**(j-k)), k, j)
circuit.barrier()
for i in range(5):
    circuit.measure(i,4-i)
simulator = Aer.get_backend('qasm_simulator')
result = execute(circuit,backend=simulator).result()
plot_histogram(result.get_counts(circuit))
%matplotlib inline
circuit.draw(output='mpl')
```

D. Código Shor N=21,A=8, diseño Kitaev en Qiskit.

```
from qiskit import *
from qiskit.tools.visualization import plot_histogram;
import math;
q = QuantumRegister(6, 'q')
c = ClassicalRegister(6, 'c')
circuit = QuantumCircuit(q, c)
#inicializar el registro a 1
circuit.x(q[0])
circuit.barrier()
#aplicar a^16 mod 21
circuit.h(q[5])
#8^16 mod 21 = 1
circuit.h(q[5])
#medir el estado de q5
circuit.measure(q[5],c[0])
#reiniciar el cubit q[5] a |0>
circuit.reset(q[5])
circuit.barrier()
#aplicar a^8 mod 21
circuit.h(q[5])
#8^8 mod 21 = 1
#   modificar en funcion del estado anterior
if c[0] == 1:
    circuit.u1(math.pi/2,q[5])
circuit.h(q[5])
#medir el estado de q5
circuit.measure(q[5],c[1])
#reiniciar el cubit q[5] a |0>
circuit.reset(q[5])
circuit.barrier()
#aplicar a^4 mod 21
circuit.h(q[5])
#8^4 mod 21 = 1
#   modificar en funcion del estado anterior
```

```
if c[1] == 1:
    circuit.u1(math.pi/2,q[5])
if c[0] == 1:
    circuit.u1(math.pi/4,q[5])
circuit.h(q[5])
#medir el estado de q5
circuit.measure(q[5],c[2])
circuit.reset(q[5])
circuit.barrier()
#aplicar a^2 mod 21
circuit.h(q[5])
#aplicar 8^2 mod 21=1
#   modificar en funcion del estado anterior
if c[2] == 1:
    circuit.u1(math.pi/2,q[5])
if c[1] == 1:
    circuit.u1(math.pi/4,q[5])
if c[0] == 1:
    circuit.u1(math.pi/8,q[5])
circuit.h(q[5])
#medir el estado de q5
circuit.measure(q[5],c[3])
circuit.reset(q[5])
circuit.barrier()
#aplicar a mod 21
circuit.h(q[5])
#aplicar X8
circuit.cswap(5,0,3)

#   modificar en funcion del estado anterior
if c[3] == 1:
    circuit.u1(math.pi/2,q[5])
if c[2] == 1:
    circuit.u1(math.pi/4,q[5])
if c[1] == 1:
    circuit.u1(math.pi/8,q[5])
if c[0] == 1:
    circuit.u1(math.pi/16,q[5])
```

```
circuit.h(q[5])
#medir el estado de q5
circuit.measure(q[5],c[4])
circuit.barrier()
backend = Aer.get_backend('qasm_simulator')
sim_job = execute([circuit], backend)
sim_result = sim_job.result()
sim_data = sim_result.get_counts(circuit)
plot_histogram(sim_data)
circuit.draw(output='mpl')
```

E. Algoritmo de Grover con 6 qubits en Qiskit.

```
from qiskit import *
from qiskit.tools.visualization import plot_histogram;
import math;

q = QuantumRegister(6, 'q')
c = ClassicalRegister(6, 'c')
circuit = QuantumCircuit(q, c)

#inicializacion
for i in range(0,6):
    circuit.h(i)
circuit.barrier()

#oraculo |110010>
circuit.cz(4,5)
circuit.x(2)
circuit.x(3)
circuit.cz(2,3)
circuit.x(2)
circuit.x(3)
circuit.x(0)
circuit.cz(0,1)
circuit.x(0)
circuit.barrier()

#inversa sobre la media
for i in range(0,6):
    circuit.h(i)
    circuit.z(i)

circuit.cz(0,1)
circuit.cz(2,3)
circuit.cz(4,5)
```

```
for i in range(0,6):
    circuit.h(i)
circuit.barrier()

for i in range(0,6):
    circuit.measure(q[i],c[i])

%matplotlib inline
circuit.draw(output='mpl')
simulator = Aer.get_backend('qasm_simulator')
result = execute(circuit,backend=simulator).result()
plot_histogram(result.get_counts(circuit))
```

F. ADJZ 3 qubits en Cirq.

```
import cirq

qr = cirq.LineQubit.range(5)
c = cirq.Circuit()

c.append(cirq.H(qr[0]))
c.append(cirq.H(qr[1]))
c.append(cirq.H(qr[2]))

c.append(cirq.Z(qr[0]))
c.append(cirq.H(qr[2]))

c.append(cirq.H(qr[0]))
c.append(cirq.CX(qr[1], qr[2]))

c.append(cirq.measure(qr[0], key='x0'))

c.append(cirq.H(qr[1]))
c.append(cirq.H(qr[2]))

c.append(cirq.measure(qr[1], key='x1'))

c.append(cirq.H(qr[2]))

c.append(cirq.measure(qr[2], key='x2'))

result = cirq.Simulator().run(c, repetitions=1000)

print(c)

cirq.plot_state_histogram(result)
```

G. Código Shor N=21,A=5 en Cirq.

```
import cirq
import matplotlib.pyplot as plt
import numpy as np

qubits = cirq.LineQubit.range(10)
c =cirq.Circuit()

q_qft =[qubits[0], qubits[1], qubits[2], qubits[3], qubits[4]]
c.append(cirq.H(qubits[0]))
c.append(cirq.H(qubits[1]))
c.append(cirq.H(qubits[2]))
c.append(cirq.H(qubits[3]))
c.append(cirq.H(qubits[4]))
c.append(cirq.X(qubits[5]))

c.append(cirq.CX(qubits[0],qubits[7]))

c.append(cirq.CCX(qubits[1], qubits[8],qubits[9]))
c.append(cirq.CCX(qubits[8],qubits[1], qubits[6]))
c.append(cirq.CSWAP(qubits[1],qubits[6], qubits[8]))
c.append(cirq.CSWAP(qubits[1],qubits[7], qubits[9]))
c.append(cirq.CSWAP(qubits[1],qubits[5], qubits[7]))

c.append(cirq.CSWAP(qubits[2],qubits[5], qubits[7]))
c.append(cirq.CSWAP(qubits[2],qubits[7], qubits[9]))
c.append(cirq.CSWAP(qubits[2],qubits[6], qubits[8]))
c.append(cirq.CCX(qubits[2], qubits[6],qubits[8]))
c.append(cirq.CCX(qubits[2],qubits[8], qubits[9]))

c.append(cirq.CCX(qubits[3], qubits[8],qubits[9]))
c.append(cirq.CCX(qubits[6],qubits[3], qubits[8]))
c.append(cirq.CSWAP(qubits[3],qubits[6], qubits[8]))
c.append(cirq.CSWAP(qubits[3],qubits[7], qubits[9]))
c.append(cirq.CSWAP(qubits[3],qubits[5], qubits[7]))
```

```
c.append(cirq.CSWAP(qubits[4],qubits[5], qubits[7]))
c.append(cirq.CSWAP(qubits[4],qubits[7], qubits[9]))
c.append(cirq.CSWAP(qubits[4],qubits[6], qubits[8]))
c.append(cirq.CCX(qubits[4], qubits[6],qubits[8]))
c.append(cirq.CCX(qubits[4],qubits[8], qubits[9]))

c.append(cirq.QFT(*q_qft,without_reverse=True, inverse=True))

c.append(cirq.measure(qubits[0], key='x0'))
c.append(cirq.measure(qubits[1], key='x1'))
c.append(cirq.measure(qubits[2], key='x2'))
c.append(cirq.measure(qubits[3], key='x3'))
c.append(cirq.measure(qubits[4], key='x4'))


result = cirq.Simulator().run(c, repetitions=100)

print(c)

cirq.plot_state_histogram(result)
```

H. Circuito analizado en Qiskit.

```
%matplotlib inline
# Importing standard Qiskit libraries and configuring account
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
# Loading your IBM Q account(s)
provider = IBMQ.load_account()

circuit = QuantumCircuit(3, 3)

circuit.h(0)
circuit.cx(0,1)
circuit.cx(1,2)

circuit.measure(0,0)
circuit.measure(1,1)
circuit.measure(2,2)

circuit.draw()

backend = Aer.get_backend('qasm_simulator')

result = execute([circuit], backend).result()

plot_histogram(resut.get_counts(circuit))
```

I. Circuito analizado en Cirq.

```
import cirq

qubits = cirq.LineQubit.range(3)
c = cirq.Circuit()

c.append(cirq.H(qubits[0]))
c.append(cirq.CNOT(qubits[0],qubits[1]))
c.append(cirq.CNOT(qubits[1],qubits[2]))

c.append(cirq.measure(qubits[0], key='x0'))
c.append(cirq.measure(qubits[1], key='x1'))
c.append(cirq.measure(qubits[2], key='x2'))

result = cirq.Simulator().run(c, repetitions=100)

cirq.plot_state_histogram(result)
```

J. Aportaciones de Amalia Regueira Fernández.

Las aportaciones de esta alumna fueron:

- Estudio básico de los conceptos de computación cuántica, ya que en un principio no tenía conocimiento alguno de la misma.
- Estudio de las puertas cuánticas.
- Aprendizaje del framework Qiskit.
- Aprendizaje del framework Cirq.
- Aprendizaje del lenguaje Python, del cual solo había oído hablar pero nunca lo había utilizado.
- Estudio de la QFT.
- Aprendizaje del algoritmo Deutsch-Jozsa.
- Implementación y ejecución del algoritmo Deutsch-Jozsa en Cirq.
- Creación de este documento y de su correspondiente plantilla en Overleaf.
- Aprendizaje de LaTex, lenguaje que utilizamos para redactar la memoria y que tampoco había utilizado con anterioridad.
- Organización de las reuniones, tanto antes de la cuarentena como después.
- Redacción de la primera parte de los conceptos básicos.
- Redacción del apartado Google: Cirq.
- Estudio más profundo de la librería de Cirq en Python.
- Estudio sobre IBM Quantum Experience.
- Investigación sobre la realización de comparativas.
- Estudio de las métricas de Halstead.

- Realización y redacción de la comparativa.
- Aprendizaje del algoritmo de Shor.
- Ejecución del algoritmo de Shor en Qiskit.
- Implementación del algoritmo de Shor en Cirq.
- Aprendizaje del algoritmo de Bernstein-Vazirani.
- Corrección ortográfica de la memoria.
- Corrección sintáctica de la memoria.

K. Aportaciones de Jairo Joel Sánchez Vélez.

Las aportaciones de este alumno son:

- Adquisición de los conceptos básicos de la computación cuántica.
- Estudio de la creación de puertas cuánticas.
- Aprendizaje del lenguaje de programación Python.
- Aprendizaje del framework Qiskit.
- Aprendizaje del framework Cirq.
- Aprendizaje de LaTex.
- Redacción de la representación matricial de las puertas cuánticas.
- Redacción del apartado de superposición.
- Redacción del apartado de entrelazamiento.
- Aprendizaje del algoritmo de Deutsch-Jozsa.
- Aprendizaje del algoritmo de Bernstein-Vazirani.
- Correcciones de la sección algoritmo de Bernstein-Vazirani.
- Estudio del algoritmo de Shor.
- Redacción de la sección del algoritmo de Shor.
- Implementación del algoritmo de Shor con el diseño de Vandersypen et al en Qiskit.
- Colaboración en la implementación del algoritmo de Shor con el diseño de Vandersypen et al en Cirq.
- Implementación del algoritmo de Shor con el diseño de Kitaev en Qiskit.
- Creación de las puertas multiplicadores modulares usadas, así como la verificación del funcionamiento para todos los posibles resultados posibles.

- Ejecución en el entorno real de ibmq-16-melbourne del algoritmo de Shor.
- Estudio del algoritmo de Grover.
- Redacción de la sección del algoritmo de Grover.
- Implementación del algoritmo de Grover para 6 qubits.
- Ejecución en el entorno real de ibmq-16-melbourne del algoritmo de Grover.
- Comprobación del código generado.
- Configurar el simulador con ruido.
- Corrección ortográfica de la memoria.
- Corrección sintáctica de la memoria.
- Posicionar las imágenes en la memoria.
- Creación de un medio de comunicación.
- Creación de una carpeta compartida de Google Drive.
- Publicación del código generado en GitHub.

L. Aportaciones de Jonathan José Jiménez.

No tenía conocimiento en computación cuántica QC , por lo que mis primeras aportaciones fueron básicas, búsqueda de tutoriales, propiedades de de la QC, y sobre todo me centre en investigar y probar los frameworks que posteriormente utilizaríamos, sobre todo Qiskit e IBM Quantum Experience, también quirk, el cual me serviría de ayuda para visualmente conocer el estado de los qubits.

Una vez que ya tenía cierta base, se propuso el algoritmo se Shor, para lo que me puse a investigar y a analizar e intentar entender algunas implementaciones, pero algunas cosas no tenía claro, por lo que antes, me puse a investigar sobre la QFT y algún algoritmo sencillo, para asentar mejor los conocimientos y entender mejor el funcionamiento y como trabaja la QC. En ese momento, me centré en el paralelismo cuántico, y me di cuenta de su importancia, ya que es una de las propiedades de la QC que la hacen diferente a la computación tradicional, por ello pensé que era conveniente exponerlo, por lo que investigué el tema, y procedí a redactarlo y explicarlo, en su estudio me di cuenta que tenía relación con el algoritmo de Deutsch-Jozsa.

Por ello, propuse introducir el Algoritmo de Deutsch-Jozsa, e intentar detallar su funcionamiento a modo de introducción a los algoritmos en computación cuántica. Investigue en varios libros, en algunos usaban notaciones diferentes y las explicaciones eran matemáticas en la mayoría de los casos. Una vez con la idea ya comprendida y habiendo contrastado en varias fuentes, pensé que introducir primero el problema de Deutsch para posteriormente generalizar hacia el problema de Deutsch-Jozsa, sería la mejor forma de presentarlos. Al ser el primer algoritmo cuántico que presentamos, pensé que si exponíamos la representación matricial, a modo de comprensión de los resultados y operaciones, sería de ayuda, ya que investigué pero no encontré en ninguna fuente esta presentación, y a mi parecer es de ayuda en la comprensión de los resultados.

Posteriormente me centré en la QFT, que es fundamental en el algoritmo de Shor. Para ello investigué en varias fuentes y libros. En la mayoría de los libros la explicación era puramente matemática, hasta que en la documentación de Qiskit encontré ejemplos prácticos. Con lo que pensé que la mejor forma de exponer la QFT, era con ejemplos prácticos visuales. Debido a esto, investigué herramientas de visualización en Qiskit que nos aportaran esto. Finalmente encontré la adecuada, la cual nos mostraba los qubits como

vectores en la esfera de Bloch. Con esto procedí a implementar un ejemplo donde se mostrara el efecto de la QFT. También a modo de contraste, expliqué su analogía en las bases computacionales tradicionales.

Posteriormente, propusimos el algoritmo de Grover, por lo que me puse a investigarlo y a estudiarlo. Pero mi compañero Jairo ya tenía casi acabada la redacción , por lo que propuse el algoritmo de Bernstein-Vazirani. El cual estudié e investigué, y cuando lo entendí pasé a su redacción.

En resumen, las aportaciones del alumno son:

- Adquisición de los conceptos básicos de la computación cuántica.
- Estudio de la creación de puertas cuánticas.
- Aprendizaje del lenguaje de programación Python.
- Aprendizaje del framework Qiskit.
- Aprendizaje del framework Cirq.
- Aprendizaje de LaTex.
- Redacción paralelismo cuántico.
- Estudio y redacción *phaseKickback*.
- Redacción y explicación con ejemplo de la QFT.
- Estudio y redacción del problema de Deutsch
- Redacción de la sección del algoritmo de Deutsch-Jozsa.
- Implementación y ejecución del problema de Deutsch en entorno real.
- Implementación Algoritmo de Deutsch-Jozsa, y ejecución en entorno real en *ibmqlondon*.
- Estudio del algoritmo de Bernstein-Vazirani.
- Redacción de la sección algoritmo de Bernstein-Vazirani.
- Ejecución en el entorno real algoritmo Bernstein-Vazirani en ibmq-16-melbourne.

- Estudio del algoritmo de Shor.
- Estudio del algoritmo de Grover.