eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Trabajo Fin de Grado
Grado en Física

Simulación de algunos algoritmos de computación cuántica

Autor:
Jon Lander Vallejo Dorado
Director:
Jesús Echevarría Ecenarro

Leioa, 27 de Febrero de 2017

Índice

1. Introducción y Objetivos

La clave del progreso en el manejo de la información tal y como la conocemos con los ordenadores clásicos viene de la reducción cada vez mayor del tamaño de los transistores. Sin embargo, los transistores están acercándose a unos tamaños que parecen sugerir que el límite de dicha reducción está cerca.

Al menos, esto podría haberle parecido a Paul Benioff, que en 1981 propuso el utilizar las leyes de la mecánica cuántica en el ámbito de la computación digital, de manera que se sustituirían los bits, cuyos valores 0 o 1 estarían controlados por el paso (o no) de corrientes, por qubits, sistemas cuánticos que además podrían estar un estado de superposición de ambos autoestados.

Desde entonces, grandes avances se han ido haciendo en este campo, sobre todo desde un punto de vista teórico. Estos avances predicen unas aplicaciones sorprendentes para los ordenadores cuánticos, aunque aún parece relativamente lejano el llegar a construir un ordenador cuántico lo suficientemente estable y potente como para poder obtener algunos de estos resultados.

Aun así, en este trabajo pretendemos explicar que consecuencias conllevaría la construcción de ordenadores cuánticos con el suficiente número de qubits. Para ello intentaremos entender, explicar y simular el comportamiento de un ordenador cuántico desde el principio, (preparación de los qubits,) hasta el final (obtención de los resultados).

Por otro lado, los ordenadores cuánticos se pueden construir bajo muchos paradigmas, pero en este trabajo nos centraremos en tipo de ordenador cuántico concreto, *the quantum gate array computer.*

Este trabajo está basado en esencia en el artículo titulado "Undergraduate computational physics projects on quantum computing, American Journal of Physics **83**, 688 (2015); doi: 10.1119/1.4922296", en donde se propone la realización de un conjunto de proyectos de complejidad creciente que ilustran las principales características de un ordenador cuántico. En estos proyectos se trata de escribir una serie de programas que simulan virtualmente (en un ordenador clásico) el funcionamiento de un ordenador cuántico. Los algoritmos concretos para la realización de los programas no se suministran en dicho artículo (únicamente hay algunas sugerencias), sino que son originales propios.

Finalmente, cabe destacar como objetivo secundario el familiarizarse con el lenguaje de programación Mathematica, el cual ha sido utilizado para llevar a cabo todas las simulaciones de este trabajo. El trabajo, a su vez, ha sido construido de manera que sea fácilmente entendible sin necesidad de saber programar con Mathematica, atendiendo más al pseudocódigo que al código. No obstante, en el apéndice final se incluye una explicación de cada una de las funciones utilizadas a lo largo del trabajo en orden de aparición, por si también se quiere entender el código paso a paso.

2. Base Teórica

Este apartado pretende explicar los conceptos teóricos fundamentales necesarios para el correcto seguimiento del desarrollo computacional. Será importante tener clara la notación utilizada tanto para el N-Qubit Register como para las puertas cuánticas. La esfera de Bloch será una buena ayuda visual para poder seguir paso a paso los procesos más simples sin necesidad de calcular las operaciones.

2.1. N-Qubit Register

El bit es la unidad más pequeña de información clásica, con dos posibles valores, 0 y 1. En un ordenador clásico, se puede utilizar un pequeño condensador con diferentes valores de carga para representar el 0 y el 1. En un ordenador cuántico, el bit se sustituye por un qubit, que es almacenado por un sistema cuántico con dos vectores de base en su espacio de Hilbert. Un buen ejemplo de sistema podría ser una partícula de spin ½, como un electrón, en la que solo el spin pueda cambiar. El estado general para un sistema de una partícula de spin ½ es:

$$|\Psi\rangle = a|\uparrow\rangle + b|\downarrow\rangle$$

Donde $|\uparrow\rangle$ y $|\downarrow\rangle$ son los estados con $S_z = \pm\hbar/2$. Las amplitudes complejas a y b obedecen la condición de normalización $|a|^2 + |b|^2 = 1$. Los estados de S_z son usados para representar 0 y 1: $|0\rangle = |\uparrow\rangle$; $|1\rangle = |\downarrow\rangle$

Un N-qubit register representa N de estos sistemas de dos niveles, considerados todos juntos como un solo sistema cuántico. Por las normas de la mecánica cuántica podemos describir un sistema de N partículas con un solo estado cuántico $|\Psi\rangle$. Por ejemplo, para un 3-qubit register tenemos una base de $2^N = 2^3 = 8$ estados:

$$|000\rangle = |\uparrow\rangle|\uparrow\rangle|\uparrow\rangle$$
$$|001\rangle = |\uparrow\rangle|\uparrow\rangle|\downarrow\rangle$$
$$|010\rangle = |\uparrow\rangle|\downarrow\rangle|\uparrow\rangle$$
$$\vdots$$
$$|111\rangle = |\downarrow\rangle|\downarrow\rangle|\downarrow\rangle$$

En el estado base $|001\rangle$, los qubits 1 y 2 tienen los valores $S_z = +\hbar/2$, mientras que el qubit 3 tiene $S_z = -\hbar/2$. El estado más general para un 3-qubit register será una superposición de los ocho estados de la base:

$$|\Psi\rangle = a|000\rangle + b|001\rangle + \cdots + g|110\rangle + h|111\rangle$$

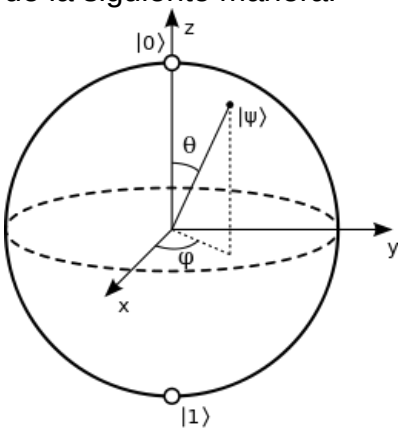
Donde las amplitudes complejas satisfacen la condición de normalización $|a|^2 + |b|^2 + \cdots + |h|^2 = 1$. El estado cuántico se puede representar matemáticamente como un vector columna con 8 valores complejos tal que:

$$|\Psi\rangle = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

2.2. La esfera de Bloch

Geométricamente la esfera de Bloch puede ser representada por una esfera de radio unidad en $\mathbf{R}^3$. En esta representación, cada punto de la superficie de la esfera corresponde unívocamente a un estado puro del espacio de Hilbert de dimensión compleja 2, que caracteriza a un sistema cuántico de dos niveles.

De este modo, un qubit puede ser perfectamente representado geométricamente por esta esfera. El grado de libertad sobre la asignación de estados sobre la superficie permite construir la esfera de la siguiente manera:

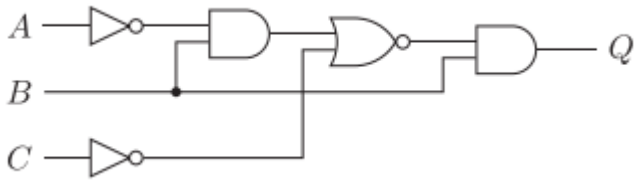


En la esfera el punto correspondiente al "polo norte" representaría el estado $|0\rangle$, mientras que el "polo sur" al estado $1\rangle$. Por tanto, en estos puntos el estado de la partícula estaría totalmente definido y cuanto más nos acerquemos al ecuador más equiprobables serán ambos estados.

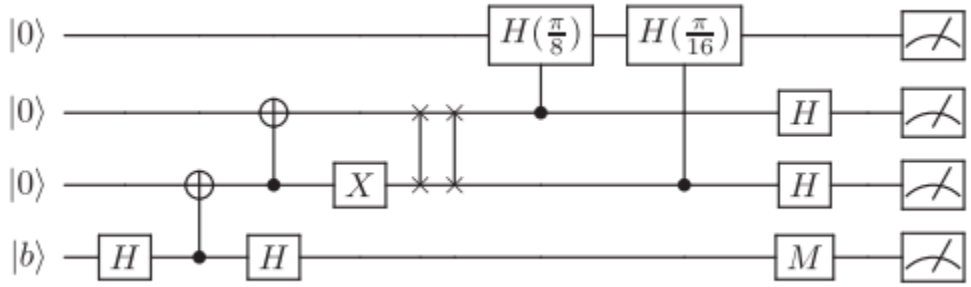
Por otro lado, un movimiento a lo largo del ecuador (o cualquier otro paralelo) mantendrá las probabilidades inalteradas, pero modificará el desfase φ entre los estados.

2.3. Puertas cuánticas

Los ordenadores clásicos están formados por puertas lógicas con nombres como AND, OR, y NOT, conectadas por cables que llevan los estados de los bits clásicos de la salida de una de las puertas a la entrada de la siguiente, tal y como podemos ver en la siguiente imagen:



Las puertas cuánticas se disponen en un esquema parecido, recogen las señales desde la izquierda y devuelven respuestas hacia la derecha. Podemos hacer la comparación con la siguiente imagen:



Sin embargo, hay un par de diferencias fundamentales:

- Cada línea horizontal representa a un qubit a lo largo del tiempo, no la disposición de unos cables en el espacio. Por tanto, las puertas cuánticas son una serie de operaciones llevadas a cabo una detrás de otra en un determinado orden
- Las operaciones llevadas a cabo por las puertas cuánticas no deben hacer que el estado cuántico del sistema colapse tal y como sucedería tras una medición, por ejemplo.

Un sistema cuántico que no ha sido medido o perturbado de alguna otra manera deberá obecer la ecuación de Schrödinger,

$$i\hbar \frac{d|\Psi\rangle}{dt} = \widehat{\mathcal{H}}|\Psi\rangle$$

En el ordenador cuántico el Hamiltoniano $\widehat{\mathcal{H}}$ debe variar con el tiempo para que la ecuación se ajuste a las puertas cuánticas que deseemos utilizar. Aplicar la ecuación de Schrödinger sobre un periodo de tiempo es siempre equivalente a aplicarle un operdador unitario a $|\Psi\rangle$. Por tanto, toda puerta cuántica deberá llevarse a cabo aplicando algún operador unitario al vector estado, tal que,

$$|\Psi\rangle \Leftarrow \widehat{U}|\Psi\rangle$$

Donde $\widehat{U}^{\dagger}\widehat{U} = \widehat{U}\widehat{U}^{\dagger} = \hat{I}$(Operador Identidad)

Principales puertas cuánticas:

Puerta de Hadamard: Es una de las puertas cuánticas más comunes y que más utilizaremos a lo largo de este trabajo. Actúa sobre 1 único qubit y se puede expresar matemáticamente mediante la siguiente matriz unitaria:

$$\widehat{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Para una interpretación más visual se puede utilizar la esfera de Bloch; aplicar la puerta de Hadamard a un estado es equivalente a una rotación de π radianes sobre el eje X seguida de una rotación de $\pi/2$ radianes sobre el eje Y.

Haciendo la operación se puede ver fácilmente que $\widehat{H}^{\dagger}\widehat{H} = \widehat{H}\widehat{H} = \hat{I}$ y que por tanto $\widehat{H}$ es unitaria. Pero lo que hace especial a esta puerta es el siguiente resultado:

$$\widehat{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$\widehat{H}|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Es decir, aplicar la puerta de Hadamard a cualquiera de los estados de la base nos devuelve una superposición equiprobable de ambos estados.
Análogamente, podemos calcular como del estado de superposición podemos llegar a los estados de la base:

$$\widehat{H}\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |0\rangle \quad , \quad \widehat{H}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = 1\rangle$$

Puertas de desplazamiento de fase: Éstas en realidad representan un conjunto de puertas y no una concreta. También actúan sobre un solo qubit y desfasan los dos estados de la base sin cambiar la probabilidad de medir cada estado, de modo que su representación matricial es:

$$\hat{R}_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

Esto se hace obvio si la aplicamos a un estado arbitrario $|\Psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$, ya que obtenemos $\hat{R}_\theta|\Psi\rangle = \begin{bmatrix} a \\ e^{i\theta} b \end{bmatrix}$. Es muy importante señalar que, aunque este desfase no pueda ser medido directamente, cambiarán las probabilidades de los resultados si después se aplican otras puertas cuánticas. Este caso se estudiará más en profundidad en el desarrollo computacional mediante ejemplos.

Como la probabilidad de medir cada estado no cambia, el ángulo polar se mantiene constante sobre la esfera de Bloch, de modo que es equivalente a un giro de θ sobre el eje Z.

El símbolo para esta puerta es una caja con el valor del desfase θ dentro de ella.

Puerta Pauli-(X/Y/Z): Equivalente a una rotación de π radianes sobre el eje X, Y o Z respectivamente. Sus representaciones matriciales son:

$$\hat{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} ; \hat{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} ; \hat{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Puerta CNOT: Esta puerta es esencial en la construcción de un ordenador cuántico. Se puede utilizar para entrelazar y desentrelazar estados. Además, cualquier circuito cuántico puede ser simulado usando una combinación de puertas CNOT y rotaciones de un solo qubit. Su representación matricial es:

$$\hat{C}_{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Como se puede deducir de la matriz, su aplicación hace que los estados $|0\rangle$ y $|1\rangle$ del qubit 2 se inviertan si y solo si el qubit 1 está en el estado $|1\rangle$

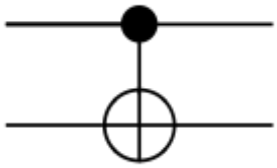
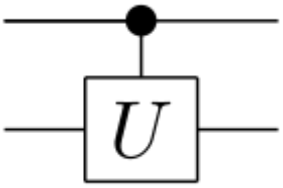
El término CNOT es una abreviación de “Controlled NOT”. NOT es la puerta lógica que cambia $|0\rangle$ por $|1\rangle$ y $|1\rangle$ por $|0\rangle$, mientras que el término Controlled hace referencia a que la aplicación o no de la puerta NOT está regulada por otro qubit.

De este mismo modo, podemos construir una puerta más general “Controlled $\hat{U}$”, donde $\hat{U}$ es una puerta que afecta a un único qubit.

$$\hat{C}_U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{0,0} & U_{0,1} \\ 0 & 0 & U_{1,0} & U_{1,1} \end{bmatrix}$$

El símbolo para esta puerta consiste en dibujar un punto negro en el qubit que se encarga de "controlar" y el símbolo correspondiente a $\hat{U}$ en el otro qubit unidos por una línea recta.

En la siguiente tabla podemos ver una serie de puertas con sus correspondientes símbolos.

Puerta cuántica	Símbolo
Hadamard	
CNOT	
Controlled U	

3. Desarrollo computacional

El objetivo de este apartado es construir virtualmente un ordenador cuántico, es decir, diseñaremos una serie de programas que harán que el ordenador simule el comportamiento de uno cuántico. Para ello utilizaremos el lenguaje de programación Mathematica.

El tipo de ordenador cuántico que simularemos consta de principalmente tres partes. La primera es la inicialización del estado cuántico, el cual suele ser siempre un estado de la base; la segunda es la aplicación de una serie de puertas cuánticas (en un determinado orden) al estado del sistema; y por último, la medición del sistema cuántico.

La primera y la última son las partes más sencillas y que se mantendrán prácticamente inalteradas para cualquier algoritmo que queramos ejecutar, por lo que empezaremos programando estas dos partes.

3.1. Inicialización y medición cuántica

Inicializar un autoestado del sistema será tan sencillo como asignar 0 a todos los valores del N-Qubit Register menos 1, que valdrá 1. En concreto, la mayoría de los programas de este trabajo se inicializarán en el estado en que todos los qubits están completamente definidos y valen 0. El N-Qubit Register correspondiente a este estado será aquel con valor 1 en la primera posición y valor 0 en el resto de las 2^N posiciones.

Por ejemplo, para N=3 tenemos:

$$|\Psi\rangle = |000\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Para asignar este valor en Mathematica podemos escribirlo directamente

$\psi = \{1,0,0,0,0,0,0,0\}$

O podemos hacerlo de forma más general para cualquier n=número de qubits

$n = 3; \psi = \text{Table}[\text{If}[i == 1,1,0], \{i, 2\text{^}n\}]$

Sobra decir que si no quisiéramos inicializarlo en este autoestado solo habría que cambiar $i == 1$ por $i == p$ donde p es la posición correspondiente al autoestado deseado. Para calcular la posición de este autoestado tenemos que pasar el número de binario a decimal, y entonces sumarle 1. Este 1 se añade para corregir el desfase debido a la asignación del estado $|000\rangle$ a la posición 1, en vez de 0.
También cabe señalar que esta corrección es necesaria porque Mathematica empieza a contar la posición de las listas desde el 1. Sin embargo, en otros lenguajes de programación como Python o C++, en donde se empieza a contar desde el 0, esta corrección no sería necesaria.

Una vez medimos un sistema cuántico, éste colapsa en uno de los estados de la base y la probabilidad de que colapse a cada uno de los autoestados vendrá dado por el cuadrado del módulo de la amplitud de probabilidad correspondiente a cada estado.
Por ejemplo, para $n = 3$, la probabilidad de que el spin de los 2 primeros qubits den $S_z = -\hbar/2$ y el tercero dé $S_z = \hbar/2$ es:

$$P(|110\rangle) = |\langle 110|\Psi\rangle|^2 = |g|^2$$

Por tanto, deberemos construir un programa que nos devuelva aleatoriamente uno de los estados de la base, pero, por supuesto, esta aleatoriedad deberá estar ajustada a la distribución de probabilidad dada por las amplitudes $a, b, c, \dots, h$ tal y como hemos explicado anteriormente.

El siguiente programa cumple con todos estos requisitos:

$$\psi = \text{Abs}[\psi]\text{^}2;$$
$$x = \text{RandomReal}[];$$
$$\text{Do}[\psi[[i]] += \psi[[i-1]], \{i, 2, \text{Length}[\psi]\}];$$
$$m = \text{Catch}[\text{Do}[\text{If}[\psi[[i]] - x > 0, \text{Throw}[i]], \{i, \text{Length}[\psi]\}]];$$
$$\psi = \text{Array}[\text{If}[\# == m, 1, 0]\&, \text{Length}[\psi]];$$
$$\text{Print}["|", \text{IntegerString}[\text{FirstPosition}[\psi, 1][[1]] - 1, 2, n], " > "]$$

A continuación, explicamos el pseudocódigo línea a línea:

- Línea 1: Calculamos el cuadrado del módulo de cada una de las amplitudes, es decir, las probabilidades de cada estado.

- Línea 2: Producimos un número aleatorio comprendido entre el 0 y el 1. Obviamente, este valor deberá ser fijado para que no se recalcule a lo largo del programa.

- Línea 3: Sustituimos cada probabilidad por el sumatorio de las anteriores más la actual. Esto dividirá la unidad en una serie de rangos numéricos, cuya anchura vendrá dada por la probabilidad de cada estado propio. De esta manera, la probabilidad de que el número aleatorio producido en la línea anterior caiga en cada uno de estos rangos coincidirá con la probabilidad del autoestado correspondiente.

- Línea 4: Calculamos cual el estado correspondiente a ese rango numérico en el que ha caído el número aleatorio.

- Línea 5: Construimos el vector correspondiente al estado calculado en la línea anterior

- Línea 6: Devuelve el estado en la notación de Dirac. Importante señalar que de nuevo tenemos que tener en cuenta el desplazamiento debido a la notación de Mathematica. En este caso deberemos restar 1 al número decimal (Es el mismo proceso explicado previamente, pero a la contra).

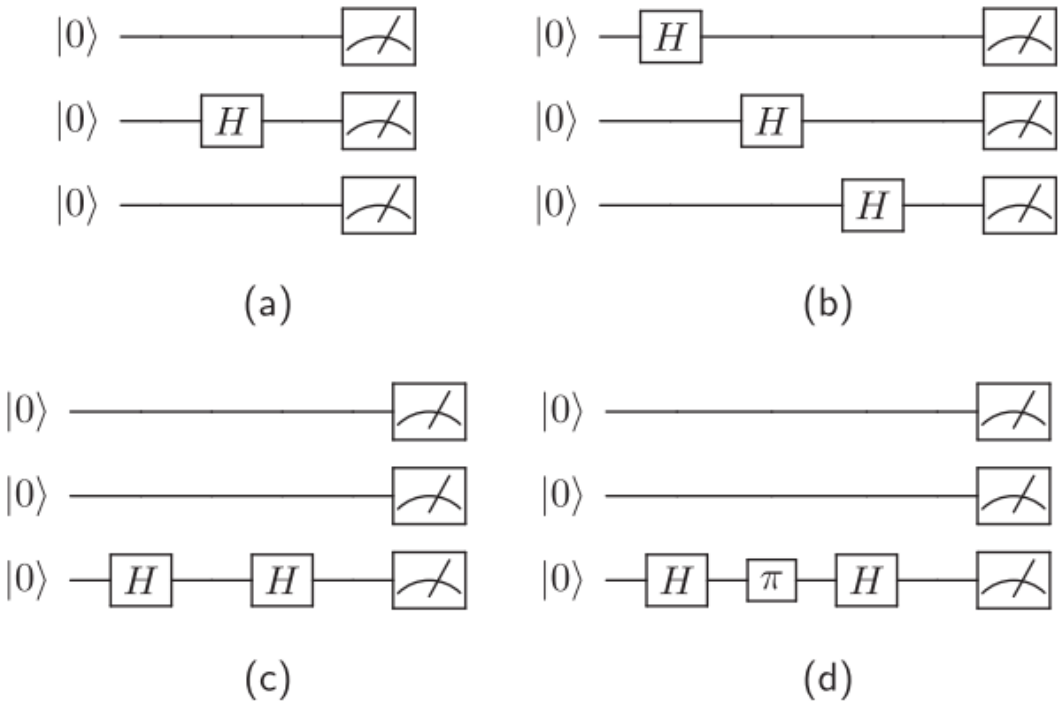
3.2. Aplicación matemática de las puertas cuánticas

En el apartado anterior hemos diseñado tanto la primera como la última parte del ordenador cuántico, por lo que solo queda estructurar la aplicación de las puertas cuánticas. El problema es que aquí es donde reside toda la información específica de cada algoritmo, por lo que no tiene sentido dar una explicación general para la construcción de esta parte.

Por tanto, optaremos por utilizar unos ejemplos sencillos para explicar el funcionamiento de esta parte, así como para explicar las principales características de algunas puertas cuánticas y los efectos producidos de combinarlas de diferentes maneras.

Para empezar, solo utilizaremos puertas de Hadamard y de desplazamiento de fase. Las combinaremos de 4 maneras diferentes para posteriormente estudiar y comparar los resultados obtenidos en cada caso.

Los cuatro casos son los siguientes:



Lo primero en lo que tenemos que fijarnos es en que tenemos 3 qubits, por lo que la dimensión de las matrices será $2^3 \times 2^3 = 8 \times 8$. Sin embargo, las matrices tal y como las hemos visto antes eran de 2×2 ya que estaban en un espacio de un solo qubit. Para deducir la forma de estas nuevas matrices a partir de las conocidas de 2×2 tendremos que utilizar álgebra tensorial.

$$\hat{H}^{(1)} = \hat{H} \otimes \hat{I} \otimes \hat{I}$$

Donde el superíndice indica a que qubit se está aplicando la matriz Hadamard. El cálculo consiste en multiplicar tensorialmente el operador $\hat{H}$ por operadores $\hat{I}$(Identidad) hasta completar el espacio (en este caso 2 veces). También es importante tener en cuenta el orden, ya que según a que qubit queramos aplicar el operador, éste irá en una posición u otra. De este modo, las operaciones para $\hat{H}^{(2)}$ y $\hat{H}^{(3)}$ son:

$$\hat{H}^{(2)} = \hat{I} \otimes \hat{H} \otimes \hat{I}$$

$$\hat{H}^{(3)} = \hat{I} \otimes \hat{I} \otimes \hat{H}$$

El cálculo matemático de este producto será explicado más en detalle en posteriores programas, ya que por ahora solo buscamos dejar claros los conceptos más fundamentales.

Por tanto, ahora simplemente daremos el resultado directo de estas operaciones:

$$\widehat{H}^{(1)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\widehat{H}^{(2)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

$$\widehat{H}^{(3)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

De forma análoga se puede expandir cualquier operador a cualquier espacio. Y así podemos deducir el valor de la puerta de desplazamiento de fase para un ángulo arbitrario θ. La única matriz que nos hace falta calcular para poder hacer las simulaciones es la que se aplica al qubit 3, y es de esta forma:

$$\hat{R}_\theta{}^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\theta} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$$

Con esto ya tenemos todas las matrices necesarias por lo que pasamos a explicar el código y los resultados.

```
h1 = 1/Sqrt[2] {{1,0,0,0,1,0,0,0}, {0,1,0,0,0,1,0,0}, {0,0,1,0,0,0,1,0}, {0,0,0,1,0,0,0,1},
{1,0,0,0, −1,0,0,0}, {0,1,0,0,0, −1,0,0}, {0,0,1,0,0,0, −1,0}, {0,0,0,1,0,0,0, −1}};
h2 = 1/Sqrt[2] {{1,0,1,0,0,0,0,0}, {0,1,0,1,0,0,0,0}, {1,0, −1,0,0,0,0,0}, {0,1,0, −1,0,0,0,0},
                {0,0,0,0,1,0,1,0}, {0,0,0,0,0,1,0,1}, {0,0,0,0,1,0, −1,0}, {0,0,0,0,0,1,0, −1}};
h3 = 1/Sqrt[2] {{1,1,0,0,0,0,0,0}, {1, −1,0,0,0,0,0,0}, {0,0,1,1,0,0,0,0}, {0,0,1, −1,0,0,0,0},
                {0,0,0,0,1,1,0,0}, {0,0,0,0,1, −1,0,0}, {0,0,0,0,0,0,1,1}, {0,0,0,0,0,0,1, −1}};
r3 = DiagonalMatrix[{1, −1,1, −1,1, −1,1, −1}];
Counts[Table[
        ψ = {1,0,0,0,0,0,0,0};
        (a) ψ = h2.ψ
        (b) ψ = h3.h2.h1.ψ
        (c) ψ = h3.h3.ψ
        (d) ψ = h3.r3.h3.ψ
        ψ = Abs[ψ]^2; x = RandomReal[]; Do[ψ[[i]]+= ψ[[i − 1]], {i, 2, Length[ψ]}];
        m = Catch[Do[If[ψ[[i]] − x > 0, Throw[i]], {i, Length[ψ]}]];
        ψ = Array[If[# == m, 1,0]&, Length[ψ]];
        IntegerString[FirstPosition[ψ, 1][[1]] − 1,2,3],100000]]
```

El código es bastante simple pero no sobra destacar un par de detalles importantes:

- Las matrices deben escribirse en orden inverso a como se ven en las figuras. La línea temporal en las figuras avanza de izquierda a derecha por lo que en el caso (b) la aplicación matemática será: $|\Psi\rangle \Leftarrow \hat{H}^{(3)}\hat{H}^{(2)}\hat{H}^{(1)}|\Psi\rangle$.
- Hemos añadido las funciones Counts y Table para realizar un conteo sobre 100000 outputs (para cada caso). Los resultados están en la siguiente tabla:

Valor de la medición	Número de mediciones obtenidas			
	Caso (a)	Caso (b)	Caso (c)	Caso (d)
000	49821	12363	100000	0
001	0	12465	0	100000
010	50179	12504	0	0
011	0	12600	0	0
100	0	12462	0	0
101	0	12700	0	0
110	0	12326	0	0
111	0	12580	0	0

Análisis de los resultados:

Caso (a): Vemos que los únicos dos resultados que han aparecido son $|000\rangle$ y $|010\rangle$. Como ya vimos, cuando el operador $\hat{H}$ es aplicado a un estado puro obtenemos una distribución equiprobable de ambos estados. Como solo se lo aplicamos al qubit 2, éste es el único que se ve alterado.

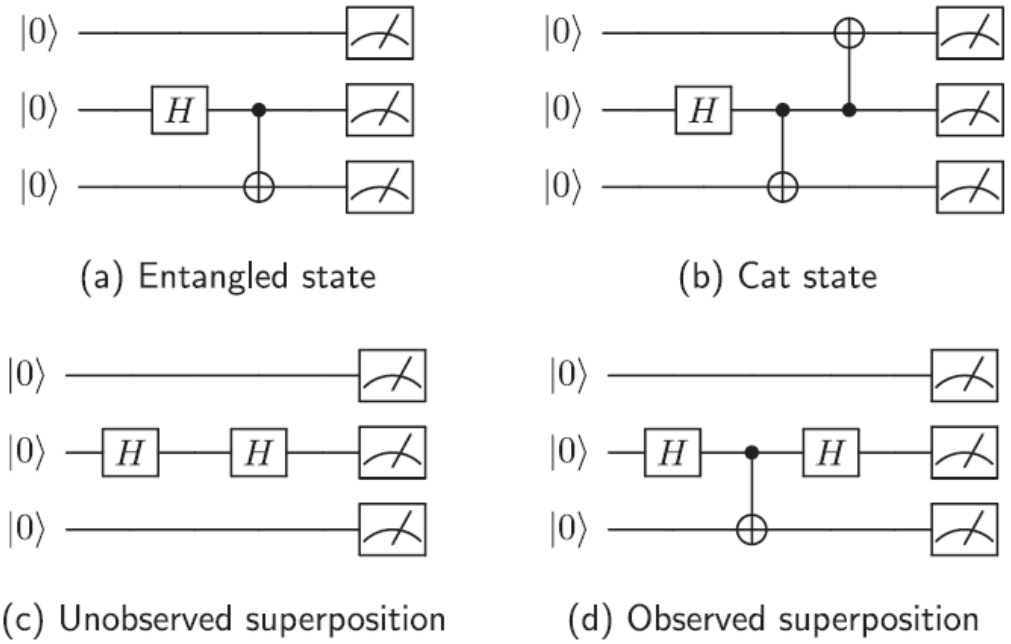
Caso (b): Esta vez, le aplicamos el operador $\hat{H}$ a todos los qubits y de esto obtenemos una distribución equiprobable de todos los posibles resultados.

Caso (c): Esta propiedad también la estudiamos anteriormente, $\hat{H}$ puede tanto "separar" un estado puro en 2 como volver a "juntar" estos dos estados en 1. De este modo las matrices se "anulan" entre ellas. Matemáticamente también podemos ver como $\hat{H} \,.\, \hat{H} = \hat{I}$

Caso (d): Aquí podemos ver perfectamente la importancia de las puertas de desplazamiento de fase. El desfasar los estados tras aplicar $\widehat{H}$, causa que al volver a aplicar $\widehat{H}$ (para "juntar" de vuelta estos estados) obtengamos $|1\rangle$ en vez de $|0\rangle$. Como esto solo se aplica al qubit 3 obtenemos $|001\rangle$ en vez de $|000\rangle$.

A continuación, veremos otros 4 casos básicos, pero esta vez añadiremos una nueva puerta, la puerta CNOT. El procedimiento es completamente análogo al de los casos anteriores así que está vez iremos un poco más directos con los resultados.

Los 4 nuevos casos son:



(a) Entangled state (b) Cat state

(c) Unobserved superposition (d) Observed superposition

Como podemos ver en las figuras, las únicas nuevas matrices que necesitaremos son:

$$\hat{c}_{NOT}{}^{(2,3)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad \hat{c}_{NOT}{}^{(2,1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Donde el primer superíndice nos especifica cual es el qubit que controla y el segundo cual es el controlado.

El código que hemos utilizado para introducir éstas matrices en Mathematica es:

```
cnot23 = DiagonalMatrix[{1,1,0,0,1,1,0,0}] + DiagonalMatrix[{0,0,1,0,0,0,1}, 1]
         + DiagonalMatrix[{0,0,1,0,0,0,1}, -1];
cnot21 = DiagonalMatrix[{1,1,0,0,1,1,0,0}] + DiagonalMatrix[{0,0,1,1},4]
         + DiagonalMatrix[{0,0,1,1}, -4]
```

Y el código específico para cada conjunto de operaciones es:

(a) $\psi = \text{cnot23}.\text{h2}.\psi$
(b) $\psi = \text{cnot21}.\text{cnot23}.\text{h2}.\psi$
(c) $\psi = \text{h2}.\text{h2}.\psi$
(d) $\psi = \text{h2}.\text{cnot23}.\text{h2}.\psi$
(El resto de código es igual al de los casos anteriores)

Y los resultados obtenidos se recogen en la siguiente tabla al igual que en los ejemplos anteriores:

Valor de la medición	Número de mediciones obtenidas			
	Caso (a)	Caso (b)	Caso (c)	Caso (d)
000	50184	49838	100000	25051
001	0	0	0	25008
010	0	0	0	24846
011	49816	0	0	25095
100	0	0	0	0
101	0	0	0	0
110	0	0	0	0
111	0	50162	0	0

Análisis de los resultados:

Caso (a): Hemos creado un estado de entrelazamiento cuántico entre los qubits 2 y 3. Primero, el operador $\hat{H}$ pone al qubit 2 en un estado de superposición de los estados $|0\rangle$ y $|1\rangle$, entonces, tras la aplicación de ${\hat{c}_{NOT}}^{(2,3)}$ el qubit 3 pasará al estado $|1\rangle$ si y solo si qubit 2 también estaba en $|1\rangle$. Es decir, el qubit 3 también entra en un estado de superposición y los resultados varían aleatoriamente entre $|000\rangle$ y $|011\rangle$.

Caso (b): Parecido al caso anterior, pero añadiéndole una ${\hat{c}_{NOT}}^{(2,1)}$ al final, lo cual hace que el qubit 1 también entre en un estado de superposición de $|0\rangle$ y $|1\rangle$. Es decir, los resultados varían aleatoriamente entre $|000\rangle$ y $|111\rangle$. El estado obtenido de estas operaciones se llama “estado del gato” en honor al famoso gato de Schrödinger. Este estado hace referencia al hecho de que el sistema se encuentra en una superposición de autoestados completamente opuesta.

Caso (c): Esta operación es equivalente a una anterior. El primer $\hat{H}$ pone al qubit 2 en un estado de superposición mientras que el segundo $\hat{H}$ vuelve a ponerlo en un estado puro. Por tanto, el resultado siempre es $|000\rangle$.

Caso (d): Modificamos el caso anterior introduciendo una ${\hat{c}_{NOT}}^{(2,3)}$ entre las dos $\hat{H}$. El ${\hat{c}_{NOT}}^{(2,3)}$ le afecta al qubit 2 como si hubiera sido medido ya que necesita “conocer” su valor. Por tanto, el estado del qubit 2 colapsa y al volver a aplicarle el operador $\hat{H}$, este pasa de nuevo a un estado de superposición. Finalmente, los resultados obtenidos son aleatorios entre los estados $|000\rangle$, $|001\rangle$, $|010\rangle$ y $|011\rangle$.

Resultados a destacar

Hemos podido comprobar directamente como el operador $\hat{H}$ hace que un determinado qubit pase de un estado puro, $|0\rangle$ o $|1\rangle$, a un estado de superposición equiprobable de ambos estados y viceversa. Asimismo, si aplicamos este operador a todos los qubits conseguiremos que el sistema se encuentre en un estado de superposición equiprobable de todos los estados propios del sistema cuántico. Este resultado es muy importante, se utilizará en los siguientes algoritmos y es lo que permitirá que sean tan eficientes.

También hemos comprobado como puertas cuánticas que por sí solas no afectaban a la distribución de probabilidad de algún determinado qubit (desplazamiento de fase, CNOT…) podían afectarle si se combinaban con otras puertas. Más adelante veremos aplicaciones realmente útiles de esta propiedad.

3.3. Algoritmos de computación cuántica

Como ya hemos mencionado en la introducción la importancia de los ordenadores cuánticos reside en la impresionante eficiencia de algunos de sus algoritmos. Ahora veremos cuantitativamente su eficiencia y analizaremos el origen y las consecuencias de esta eficiencia.

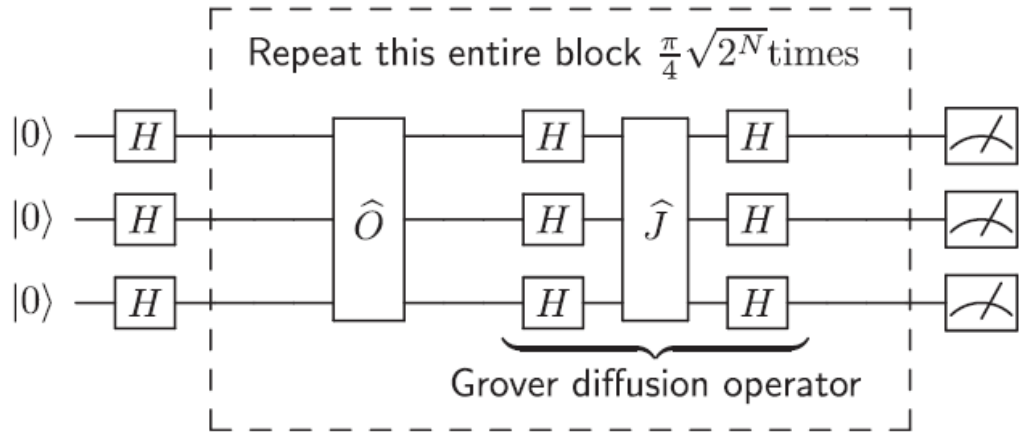
Algoritmo de búsqueda de Grover

Considera una base de datos clásica como puede ser una agenda con D nombres, cada uno de ellos seguidos de un número de teléfono. Para encontrar un dado número de teléfono, ¿cuántas entradas en la agenda tienen que ser comprobadas antes de encontrar el nombre correspondiente? En el mejor de los casos nos valdrá con mirar una sola entrada, pero en el peor de los casos tendremos que revisar las D entradas para encontrar justo el que nosotros queremos. De media, el número de revisiones necesarias para encontrar el nombre correspondiente será $D/2$.

Desde un punto de visto más computacional; tenemos una función lógica de N variables Booleanas, T(True) o F(False), y la función solamente será T para una combinación específica de inputs. Para encontrar esta combinación de entre las $D = 2^N$ posibilidades para los inputs, uno tiene que ir probando cada una de esas D posibilidades hasta encontrar la que nos devuelva el valor T. Obviamente, de media siguen siendo $D/2$ ensayos.

Usando el algoritmo de búsqueda de Grover tenemos una reducción de las veces que tiene que ser consultada la base de datos de media de $D/2$ a $(\pi/4)\sqrt{D}$. Esto implica un aumento de la eficiencia enorme para bases de datos suficientemente grandes. Y aquí está el truco: La base de datos o función lógica debe ponerse en la forma de oráculo cuántico. Un oráculo clásico devuelve una respuesta de 1bit (SÍ o NO) en respuesta a una pregunta como "¿Es el teléfono 666 666 666 la entrada número 125 de la agenda?" o "¿La función lógica vale T para los inputs TFTTFTFFT?". Un oráculo cuántico debe aceptar una superposición cuántica de preguntas, y devolver la correspondiente superposición cuántica de respuestas.

Usando la superposición, es posible consultarle al oráculo todas las posibles preguntas simultáneamente. Puede parecer que entonces solo es necesario consultar una vez al oráculo cuántico, pero medir los qubits hace que el estado cuántico colapse, haciendo imposible determinar cuál era el estado de superposición previo a la medida. No obstante, Grover mostró la manera de obtener la información deseada del oráculo con muchos menos intentos que los necesitados por un algoritmo clásico.



En la figura se muestra el circuito cuántico para el algoritmo de búsqueda de Grover. El tamaño de la base de datos que vamos a manejar es $D = 2^N$, donde N es el número de qubits. El diagrama está dibujado para $N = 3 \Longrightarrow D = 8$. Además de las puertas de Hadamard (que vienen representadas con una H en el diagrama) que ya hemos explicado anteriormente; necesitaremos un operador $\hat{O}$, que será el oráculo cuántico, y un operador especial $\hat{J}$, los cuales explicaremos a continuación.

El oráculo es donde se contiene la información de cuál es la respuesta correcta y funciona de la siguiente manera:

- Si al oráculo se le da cualquiera de las $D - 1$ preguntas incorrectas devuelve el input sin alterar. Por ejemplo, si la pregunta correcta es 110, entonces 100 no lo será y por tanto, $\hat{O}|100\rangle = |100\rangle$.
- Del mismo modo, si se le hace la pregunta adecuada, el oráculo devolverá el input multiplicado por -1. Siquiendo con el ejemplo anterior, tenemos: $\hat{O}|110\rangle = -|110\rangle$

Por lo tanto, el oráculo cuántico es como la matriz identidad, menos por un factor de -1 para el elemento diagonal correspondiente a la pregunta correcta. Cuando la pregunta adecuada es 110, la matriz queda de la forma:

$$\hat{O} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Es inmediato comprobar que el oráculo $\hat{O}$ es unitario, que si multiplicamos la matriz por su adjunto obtenemos la matriz identidad, y por tanto es un operador permitido. Por otro lado, el operador $\hat{J}$ es idéntico al oráculo excepto porque el factor -1 va siempre en el primer término diagonal, tal que:

$$\hat{J} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Tal y como hemos mencionado antes, la clave de la eficiencia de este método reside en la utilización de la superposición cuántica para darle al oráculo todas las preguntas simultáneamente. Este estado se consigue aplicando el operador $\hat{H}$ a todos los qubits. Entonces el oráculo cambia el signo de la amplitud de probabilidad de la pregunta correcta. A priori, podríamos pensar que este signo (una diferencia de fase en la amplitud de probabilidad) no afecta en nada a las probabilidades (que dependen del valor absoluto de la amplitud de probabilidad) , sin embargo, ya hemos visto que esto no es verdad. De hecho, El "operador de difusión de Grover" está diseñado matemáticamente para convertir esta diferencia de fase, la cual no se puede medir, en una diferencia de magnitud que se mostrará cuando el qubit sea medido.

A. <u>Programación y resultados para N=3</u>

Las nuevas matrices las introducimos mediante el siguiente código:

$o = \text{DiagonalMatrix}[\{1,1,1,1,1,-1,1\}]; j = \text{DiagonalMatrix}[\{-1,1,1,1,1,1,1,1\}]$

Y aplicamos las puertas del algoritmo mediante:

$\psi = \text{h1.h2.h3.}\psi; \text{Do}[\psi = \text{h1.h2.h3.}j.\text{h1.h2.h3.}o.\psi, \text{Round}[\text{Pi}/4 * \text{Sqrt}[8]]]$

No obstante, sucede que nunca hay una certeza del 100% de que el algoritmo encuentre la respuesta correcta, y cuanto menor es el número de qubits mayor es la probabilidad de fallar.

Además, puede parecer lógico pensar que para corregir esta incertidumbre será suficiente con aumentar el número de iteraciones, pero esto no solo haría al algoritmo menos eficiente, sino que también lo haría más impreciso.

En número óptimo de iteraciones es el número entero más cercano a $\sqrt{2^3}\pi/4$, que para abreviar denominaremos ItOp. En este caso el conteo lo hemos hecho sobre 10 000

Valor de la medición	Número de mediciones obtenidas		
	ItOp	ItOp-1	ItOp+1
000	83	307	928
001	84	312	962
010	76	294	972
011	98	310	970
100	76	342	983
101	76	358	938
110	9423	7787	3320
111	84	290	927

Por lo tanto, queda claro que no tiene ningún sentido aumentar el número de iteraciones, de hecho, cuando ha habido el mayor número de iteraciones hemos obtenido el peor resultado de todos.

La probabilidad teórica de obtener la respuesta correcta es de $121/128 \approx 0.9453$, lo cual supone un error que muy poca gente estaría dispuesta a aceptar a la hora de buscar un número de teléfono en su agenda, por ejemplo.
Veamos cual es esta probabilidad tras aumentar el número de qubits.

B. <u>Incrementando el número de qubits</u>

Como ya mencionamos, para construir las matrices utilizadas en los apartados anteriores, para n(número de qubits)$= 3$, utilizamos álgebra tensorial. En este apartado explicaremos en detalle la forma de hallar estas matrices para cualquier n.
La dimensión de estas matrices será de $2^n \times 2^n$ así que usaremos una notación más compacta que se ajustará mejor a nuestro problema. Siendo p y q dígitos binarios, que podrán valer 0 o 1, una matriz de 2×2, $\widehat{M}$, puede ser representada de la forma $M_{p,q}$, de modo que

$$\widehat{M} = \begin{bmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{bmatrix}$$

Bajo esta notación la delta de Kronecker es equivalente a la matriz identidad ya que

$$\delta_{pq} = \begin{bmatrix} \delta_{00} & \delta_{01} \\ \delta_{10} & \delta_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{p,q}$$

En el caso de $n = 3$, siendo $[pqr]$ un número binario de 3 dígitos, con esta notación el producto tensorial se puede reescribir de la siguiente manera

$$\widehat{H}^{(2)} = \hat{I} \otimes \widehat{H} \otimes \hat{I} \Leftrightarrow H^{(2)}_{[pqr],[p'q'r']} = \delta_{pp'} H_{q,q'} \delta_{rr'}$$

La matriz de Hadamard opera sobre los índices q, q' que se corresponden con el qubit 2, mientras que las deltas de Kronecker actúan como operador identidad para el resto de qubits.
Veamos un ejemplo para entenderlo mejor.
Si los índices de las matrices se empiezan a contar desde 0, tenemos que el elemento de la séptima fila y quinta columna de $\widehat{H}^{(2)}$ es $H^{(2)}_{6,4}$, y se computa así:

$$H^{(2)}_{6,4} = H^{(2)}_{[110],[100]} = \delta_{11} H_{1,0} \delta_{00} = H_{1,0} = \frac{1}{\sqrt{2}}$$

Pero Mathematica empieza a contar desde 1 por lo que habrá que tener en cuenta este corrimiento a la hora de escribir el código.
El algoritmo de Grover no requiere el cálculo de más matrices pero el de Shor, que veremos en el siguiente apartado, sí. Si queremos calcular la forma de una puerta de dos qubits en un espacio de 3 qubits, por ejemplo, una puerta CNOT en la que el qubit 2 controla al qubit 1 tendremos que aplicar:

$$C^{(2,1)}_{NOT[pqr],[p'q'r']} = C_{NOT[qp],[q'p']} \delta_{rr}$$

Al igual que antes, tenemos una delta de Kronecker por cada por cada qubit que no está siendo afectado por la puerta. En este caso, como el qubit 2 es el qubit que controla, se usan primero los índices q, q' para la matriz C_{NOT}. Por ejemplo:
Para el elemento de la séptima fila y tercera columna de $\hat{C}^{(2,1)}_{NOT}$, que es $C^{(2,1)}_{NOT\ 6,2}$,

$$C^{(2,1)}_{NOT\ 6,2} = C^{(2,1)}_{NOT[110],[010]} = C_{NOT[11][10]} \delta_{00} = C_{NOT\ 3,2} = 1$$

Si utilizamos directamente estas fórmulas para computar las matrices el código nos podría quedar de la siguiente forma:

```
ans = 7; n = 4; h = 1 / Sqrt[2] {{1,1},{1,-1}};
o = DiagonalMatrix[Table[If[i == ans, -1,1],{i, 2^n }]];
j = DiagonalMatrix[Table[If[i == 1, -1,1],{i, 2^n }]];
H = Table[sol = 1;
  Do[
    If[k == m,
      sol = sol * h[[IntegerDigits[i, 2, n][[k]] + 1, IntegerDigits[j, 2, n][[k]] + 1]],
      sol = sol * IdentityMatrix[2][[IntegerDigits[i, 2, n][[k]] + 1,
                IntegerDigits[j, 2, n][[k]] + 1]]],
    {k, n}]; sol,
  {m, 1, n}, {i, 0,2^n - 1}, {j, 0,2^n - 1}]
```

Sin embargo, este código aún tiene un problema. Consume muchos más recursos de los que podría, y debido al crecimiento exponencial del tamaño de las matrices, necesitamos un código más eficiente.

Para aprovecharnos de la gran cantidad de 0 que hay en casi todas las matrices que utilizamos utilizaremos las SparseArray, que es una manera de computar matrices y vectores guardando solo la información de los términos no nulos.

De esta manera, tendríamos esta nueva versión del código, que obtendría exactamente los mismos resultados, pero de una manera mucho más eficiente:

```
ans = 7; n = 4; h = 1 / (Sqrt[2]){{1,1},{1,-1}};
j = SparseArray[{Band[{2,2}] → 1, {1,1} → -1}, {2^n, 2^n }];
o = SparseArray[{Band[{1,1}] → Table[If[i == ans, -1,1],{i, 2^n }]}, {2^n, 2^n }];
H = SparseArray[{k_, i_, j_}/;
      Drop[IntegerDigits[i - 1,2, n], {k}] == Drop[IntegerDigits[j - 1,2, n], {k}]

      :→ h[[IntegerDigits[i - 1,2, n][[k]] + 1, IntegerDigits[j - 1,2, n][[k]] + 1]],

    {n, 2^n, 2^n}]
```

Este nuevo código es mucho más compacto y eficiente. Para mostrar esto de una manera más cuantitativa haremos uso de la función Timing, la cual nos dice el tiempo que ha requerido el ordenador para realizar las operaciones.

En la siguiente tabla recogemos los datos de los tiempos requeridos por los dos programas para diferentes valores de n(número de qubits):

Número de qubits	Tiempo requerido	
	Con Arrays	Con SparseArrays
4	0.03125s	0. (Valor demasiado pequeño)
5	0.125s	0.03125s
6	0.59375s	0.109375s
7	3.20313s	0.40625s
8	16.7813s	1.78125s
9	1min 22.1563s	8.07813s
10	6min 46.297s	36.5s
11	33min 23.28s	2min 53.531s

Los resultados son claros, el ahorro temporal al utilizar las SparseArrays es enorme, por tanto, las utilizaremos a partir de ahora en todos los programas. Esto será especialmente útil para el algoritmo de Shor, para el cual el mínimo número de qubits utilizables es 7.
Después de haber analizado la optimización del programa, ahora estudiaremos su precisión. Podríamos hacerlo por fuerza bruta como en los apartados anteriores, pero viendo los recursos que exigen estos programas lo más inteligente será analizar el cuadrado del módulo de las amplitudes de probabilidad tras realizar todas las operaciones.
Recogemos el resultado de este análisis en las siguientes tablas:

Número de qubits	Probabilidad de medir: El estado correcto	Cualquiera de los otros estados
4	$\frac{63\,001}{65\,536}$	$\frac{169}{65\,536}$
5	$\frac{536\,431\,921}{536\,870\,912}$	$\frac{14161}{536\,870\,912}$
6	$\frac{17\,952\,891\,602\,536\,801}{18\,014\,398\,509\,481\,984}$	$\frac{976\,300\,110\,241}{18\,014\,398\,509\,481\,984}$
7	$\frac{156\,519\,773\,016\,413\,457\,418\,027\,622\,209}{158\,456\,325\,028\,528\,675\,187\,087\,900\,672}$	$\frac{15\,248\,441\,040\,277\,305\,268\,191\,169}{158\,456\,325\,028\,528\,675\,187\,087\,900\,672}$

Número de qubits	Probabilidad de medir: El estado correcto	Cualquiera de los otros estados
4	0.961318	0.00257873
5	0.999182	0,0000263769
6	0.996586	0.000541955
7	0.987779	0.0000962312
8	0.986186	0.0000541716
9	0.995791	8.2364e-6

Estos resultados son muy interesantes, ya que no vemos un aumento o una disminución constante de la fiabilidad según va aumentando el número de qubits. Por la limitada potencia de un ordenador convencional no se ha podido obtener un muestreo muy grande, sin embargo, es suficiente para intuir un que la probabilidad de obtener el estado correcto tiende al alza, pero con fluctuaciones.
Podemos ver como en la primera tabla el denominar es sistemáticamente cada vez más grande (y aquí podemos intuir la tendencia al alza), no obstante, el numerador no siempre se mantiene tan ajustado como uno podría pensar en un principio (y es en este término donde ocurren las fluctuaciones).
Matemáticamente, estas fluctuaciones vienen del término para las iteraciones $(\pi/4)\sqrt{2^n}$, el cual no es un número entero y por tanto es necesario redondear.

C. <u>Conclusiones y puntualizaciones finales</u>

Al principio, el algoritmo ha parecido no llegar a la altura de las expectativas. Aunque conseguía ahorrarnos unas cuantas iteraciones respecto al algoritmo clásico no era capaz de darnos la respuesta correcta con la frecuencia suficiente como para poder considerar al programa fiable.
Sin embargo, al aumentar el número de qubits hemos podido apreciar un aumento palpable de la fiabilidad del programa. Quizás, debido al pequeño tamaño del muestreo, los resultados pueden no ser muy convincentes, pero se puede demostrar

matemáticamente como la probabilidad de obtener una respuesta incorrecta tiende a 0 para un valor de n lo suficientemente grande.
Así que, no cabe ninguna duda de que el algoritmo de Grover puede llegar a desbancar por completo a su análogo clásico.
También es importante señalar la importancia de utilizar SparseArrays debido a la enorme exigencia computacional de estos programas.

Algoritmo de Shor

A. Importancia del algoritmo

En criptografía clásica, el algoritmo RSA (Rivest, Shamir, Adleman) es un sistema criptográfico de clave pública que se utiliza tanto para cifrar como para firmar digitalmente. La idea del algoritmo es la siguiente:
Supongamos que Bob quiere enviar a Alicia un mensaje secreto que solo ella pueda leer. Alicia envía a Bob una caja con una cerradura abierta, de la que solo Alicia tiene la llave. Bob recibe la caja, escribe el mensaje, lo pone en la caja y la cierra con su cerradura (ahora Bob no puede leer el mensaje). Bob envía la caja a Alicia y ella la abre con su llave. En este ejemplo, la caja con la cerradura es la "clave pública" de Alicia, y la llave de la cerradura es su "clave privada".

Técnicamente, Bob envía a Alicia un «mensaje llano» M en forma de un número m menor que otro número n, mediante un protocolo reversible conocido como *padding scheme* (patrón de relleno). A continuación, genera el "mensaje cifrado" c mediante la siguiente operación:

$$c \equiv m^e \pmod{n}$$

donde e es la clave pública de Alicia.
Ahora Alicia descifra el mensaje en clave c mediante la operación inversa dada por

$$m \equiv c^d \pmod{n}$$

donde d es la clave privada que solo Alicia conoce.

Para la generación de las claves, el algoritmo consta de los siguientes pasos:

1. Cada usuario elige dos números primos distintos p y q. (Por motivos de seguridad, estos números deben escogerse de forma aleatoria y deben tener una longitud en bits parecida. Se pueden hallar primos fácilmente mediante un test de primalidad.)
2. Se calcula $n = pq$. (n se usa como el módulo para ambas claves, pública y privada)
3. Se calcula $\varphi(n) = (p-1)(q-1)$, donde φ es la función φ de Euler.
$$\varphi(n) = |\{n \in \mathbb{N} | n \leq m \wedge mcd(m,n) = 1\}|$$
4. Se escoge un número positivo e menor que $\varphi(n)$, que sea coprimo con $\varphi(n)$. (e se da a conocer como el exponente de la clave pública)
5. Se determina un d (mediante aritmética modular) que satisfaga la congruencia $e \cdot d \equiv 1 \pmod{\varphi(n)}$, es decir, que d sea el multiplicador modular inverso de $e \mod \varphi(n)$
(Esto suele calcularse mediante el algoritmo de Euclides extendido)

La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Si un número grande, de b bits es el producto de dos primos de aproximadamente el mismo tamaño, no existe algoritmo conocido capaz de factorizarlo en tiempo polinómico. Esto significa que ningún algoritmo conocido puede factorizarlo en tiempo $O(b^k)$, para cualquier constante *k*. Aunque, existen algoritmos que son más rápidos que $O(a^b)$ para cualquier a mayor que 1.

En otras palabras, los mejores algoritmos son súper-polinomiales, pero sub-exponenciales. En particular, el mejor tiempo asintótico de ejecución es el del algoritmo de criba general del cuerpo de números (CGCN), que es:

$$O\left(\exp\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}}(\log(b))^{\frac{2}{3}}\right)\right)$$

Para una computadora ordinaria, la CGCN es el mejor algoritmo conocido para números grandes. Para una computadora cuántica, en cambio, Peter Shor desbubrió en 1994 un algoritmo que lo resuelve en tiempo polinómico. Esto tendría implicaciones importantes en criptografía, para descomponer un número n el algoritmo de Shor solo tarda un tiempo $O((\log(n))^3)$.

B. <u>Algoritmo de Shor paso a paso</u>

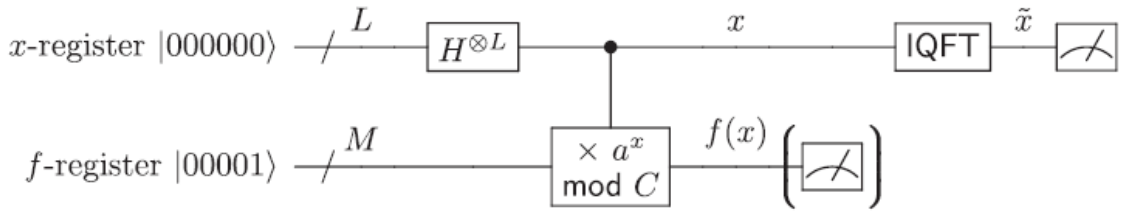
Dado un entero compuesto C, los siguientes pasos hallan factores no triviales (que no sean 1 o C) del número C:

1. Comprobar que el número C es impar y no es una potencia de algún entero más pequeño. Si C es par o una potencia, entonces ya tenemos un factor de C y hemos acabado.
2. Escoger cualquier entero a en el rango $1 < a < C$.
3. Encontrar $gdc(a, C)$. Si por suerte el máximo común divisor es mayor que 1, entonces tenemos un factor de C y de nuevo ya hemos acabado.
4. Encontrar el entero más pequeño $p > 1$ que cumpla $a^p \equiv 1(mod\ C)$
5. Si p es impar, o si p es par y cumple $a^{p/2} \equiv -1(mod\ C)$, entonces volver al paso 2 y coger un a diferente.
6. Los números $P_{\pm} = gdc(a^{\frac{p}{2}} \pm 1, C)$ son factores no triviales de C.

Por ejemplo, si tenemos $C = 15$

1. 15 es un número impar y no es una potencia de un entero más pequeño, así que seguimos.
2. Arbitrariamente elegimos $a = 7$.
3. $gdc(7{,}15) = 1$, así que seguimos.
4. Buscamos p empezando por 2:
   - $7^2 = 49$ y $49(mod\ 15) = 4 \neq 1$
   - $7^3 = 343$ y $343(mod\ 15) = 13 \neq 1$
   - $7^4 = 2401$ y $2401(mod\ 15) = 1$, asi que $p = 4$
5. $p = 4$ es par, y $7^{4/2} = 49$. Como $49 \not\equiv -1(mod\ 15)$, no es necesario volver al paso 2 para elegir otro a.
6. $P_+ = \gcd(50{,}15) = 5$ y $P_- = \gcd(48{,}15) = 3$ son los buscados factores de 15.

Un ordenador clásico podría determinar rápidamente si C es una potencia; o cual es el máximo común divisor de dos números grandes, utilizando el algoritmo de Euclides. Por tanto, el único paso para el que es necesario un ordenador cuántico es el paso 4. Este paso se denomina “period-finding” y se representa de forma general para un número $N = L + M$ de qubits mediante el siguiente diagrama:



A este paso se le llama así porque la función $f(x) = a^x (mod\ C)$ es una función periódica con periodo p (justo el número que buscamos). Por ejemplo, si tenemos $C = 15$ y elegimos $a = 7$ obtenemos $p = 4$, y como cabe de esperar, si desarrollamos la función $f(x)$ vemos que el periodo coincide con el resultado:
$f(0) = 7^0\ (mod\ 15) = 1$
$f(1) = 7^1\ (mod\ 15) = 7$
$f(2) = 7^2\ (mod\ 15) = 4$
$f(3) = 7^3\ (mod\ 15) = 13$
$f(4) = 7^4\ (mod\ 15) = 1$
$f(5) = 7^5\ (mod\ 15) = 7$
$f(6) = 7^6\ (mod\ 15) = 4$

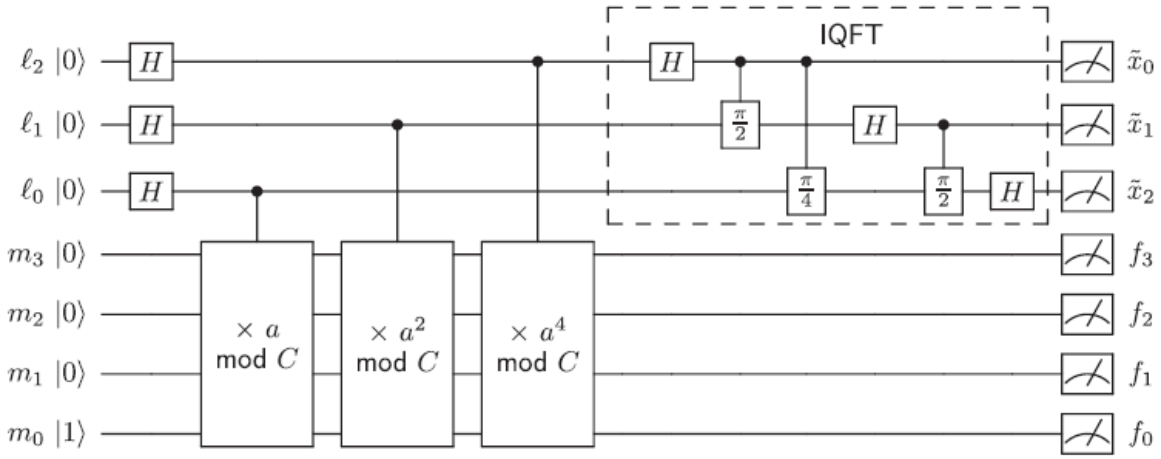
Como vemos en el circuito cuántico el registro de los qubits se divide en dos partes; el "$x - register$" con L qubits inicializados en el estado $|0 \dots 0\rangle$, y el "$f - register$" con M qubits inicializados en el estado $|0 \dots 01\rangle$. Los pasos en el cálculo del circuito son los siguientes:

1. Aplicar la puerta de Hadamard a cada uno de los L-qubits en el $x - register$. Esto pone el $x - register$ en un estado de superposición equiprobable de todos los posibles 2^L valores de x.
2. En función del valor del $x - register$, multiplicar el $f - register$ por $a^x (mod\ C)$. Así se queda el valor de $f(x)$ guardado en el $f - register$.
3. Medir el $f - register$. En realidad, no importa si el $f - register$ es medido ahora, después cuando se mida el $x - register$, o nunca, en el diagrama se ha elegido este orden por arbitrariedad.
4. Realizar la transformada de Fourier cuántica (IQFT) sobre el $x - register$. Como la transformada de Fourier de una función tiene picos en las frecuencias presentes en la función, la IQFT hará posible encontrar el periodo (1/frecuencia) de $f(x)$. La IQFT será explicada más en detalle en posteriores apartados.
5. Medir el output $\tilde{x}$ de la IQFT. Shor demostró que el valor medido $\tilde{x}/2^L$ es aproximadamente igual a s/p, donde s es un entero desconocido. Esto se utiliza para encontrar p. Por ejemplo, si $\frac{\tilde{x}}{2^L} = 0.32 \approx \frac{1}{3} = \frac{2}{6} = \frac{3}{9} \cdots$ entonces se estima que p debe ser alguno de los denominadores $3, 6, 9$ … Estos valores son los que se comprueban para ver cuál es el que satisface la relación que buscábamos $a^p \equiv 1 (mod\ C)$, y por tanto será el verdadero periodo p.

C. <u>Algoritmo de Shor con 7 qubits</u>

Los números más pequeños que satisfacen el paso 1 del algoritmo (enteros impares no primos y que no son potencias de otros enteros) son $C = 15{,}21{,}33{,}35{,}39$ … Ya a día de hoy, se ha utilizado el algoritmo se Shor en ordenadores cuánticos reales (no simulados) para factorizar los números 15 y 21. Vandersypen $et\ al.$ han construido ordenadores cuánticos con $N = 7$ usando spins nucleares como qubits. Usaron el algoritmo de Shor para factorizar $C = 15$ usando $L = 3$ y $M = 4$.

Este es el circuito cuántico para el algoritmo de Shor con $N = 7$ qubits, basado en el diseño de Vandersypen $et\ al$.



El algoritmo tiene 3 tipos de puertas cuánticas:

- Puertas de Hadamard: Ya se han explicado y utilizado para el algoritmo de Grover. No es necesaria ninguna modificación del código para esta parte.
- Puertas de desplazamiento de fase controladas: Son un caso particular de una "Controlled-U", y para construirlas utilizaremos las propiedades del álgebra tensorial tal y como explicamos en apartados anteriores. El código para la construcción de estas matrices es:

  $\text{rmat[x\_]} \coloneqq \{\{1,0,0,0\},\{0,1,0,0\},\{0,0,1,0\},\{0,0,0,\text{Exp}[I * x]\}\};$
  $\text{cr[x\_, l\_, k\_]} \coloneqq \text{SparseArray}[\{\text{i\_, j\_}\}/;$
  $(a = \text{IntegerDigits}[i - 1,2,n]; a[[l]] = 0; a[[k]] = 0;$
  $b = \text{IntegerDigits}[j - 1,2,n]; b[[l]] = 0; b[[k]] = 0;$
  $a == b) :\to \text{rmat}[x]$
  $[[\text{FromDigits}[\{\text{IntegerDigits}[i - 1,2,n][[l]], \text{IntegerDigits}[i - 1,2,n][[k]]\},2] + 1,$
  $\text{FromDigits}[\{\text{IntegerDigits}[j - 1,2,n][[l]], \text{IntegerDigits}[j - 1,2,n][[k]]\},2] + 1]],$
  $\{2\text{^}n, 2\text{^}n\}]$

- Puertas $a^x(mod\ C)$: Hay 3 puertas cuánticas usando los bits del $x - register$ l_2, l_1, l_0 para controlar los bits del $f - register$ m_3, m_2, m_1, m_0. La puerta controlada por el bit l_k multiplica condicionalmente el $f - register$ por $a^{2^k}(mod\ 15)$. Juntas, las tres puertas multiplican el $f - register$ por $a^{l_0+2l_1+4l_2}(mod\ 15) = a^x(mod\ 15) = f(x)$, como se requiere para el paso 2 del "period-finding".

La construcción de estas últimas matrices es bastante más complicada que la de las demás, por eso, antes de escribir el código explicaremos unos conceptos fundamentales que deberán quedar claros para su correcta comprensión.
Son matrices de permutación, lo cual significa que por cada una de las 128 columnas que tiene solo una de las filas valdrá 1 y el resto valdrá 0. (Aquí podemos apreciar de nuevo el acierto que supone utilizar las SparseArrays.) Por tanto, el problema a resolver para construir estas matrices será encontrar cual es la fila no nula que le corresponde a cada columna.

Empezaremos explicando el pseudocódigo para la matriz controlada por l_0. Esto se ejecutará para cada columna $k = 1 \dots 128$:

1. Se escribe el número decimal $k-1$ (debido a que empezamos a contar por 1 en vez de 0) como un número binario de 7 digitos tal que $k-1 = l_2 l_1 l_0 m_3 m_2 m_1 m_0$. Cada dígito se corresponderá con un determinado qubit tal y como vemos a la izquierda en el diagrama del circuito cuántico.
2. Como ya vimos al explicar las "Controlled-U", éstas solo actúan si el qubit que controla vale 1, si no, funcionan como la matriz identidad. Por tanto, si $l_0 = 0$, entonces $j = k$.
3. Si $l_0 = 1$, expresamos el número binario de cuatro dígitos $m_3 m_2 m_1 m_0$ como un número decimal f. Si $f \geq 15$ entonces $j = k$. Pero, si $f < 15$ entonces calcularemos un nuevo valor $f' = A_0 f (mod\ 15)$ donde $A_0 = a^{2^0}$. Entonces, la fila j que buscamos vendrá dada por $j = l_2 l_1 l_0 m'_3 m'_2 m'_1 m'_0$. De este modo, si $l_0 = 1$ entonces el $f - register$ es multiplicado por $A_0 (mod\ 15)$.

Las otras dos matrices se construyen exactamente igual, pero cambiando l_0 por l_1 y l_2, y A_0 por A_1 y A_2; donde $A_n = a^{2^n} (mod\ 15)$.
Finalmente, el código nos queda de la siguiente forma:

```
n = 7; a = 7; A = Table[Mod[a^(2^(i - 1) ),15], {i, 3}];
f[k_ ] := FromDigits[Drop[IntegerDigits[k, 2, n],3],2];
matfx =
  Table[
    SparseArray[
      {j_, k_}/; And[IntegerDigits[k - 1,2, n][[3 + 1 - i]] == 1,
        FromDigits[Drop[IntegerDigits[k - 1,2, n],3],2] < 15,
        j == FromDigits[Join[Drop[IntegerDigits[k - 1,2, n], -4],
          IntegerDigits[Mod[f[k - 1] * A[[i]],15],2,4]],2] + 1] → 1,
      {2^n, 2^n}] +
    SparseArray[
      {j_, k_}/; And[IntegerDigits[k - 1,2, n][[3 + 1 - i]] == 0||
        FromDigits[Drop[IntegerDigits[k - 1,2, n], 3], 2] ≥ 15,
       j == k] → 1,
      {2^n, 2^n}],
  {i, 3}]
```

<u>Cómo funciona el algoritmo desde el punto de vista de la física cuántica</u>

Una vez registrada la información de $f(x) = a^x (mod\ 15)$ en el $f - register$ ya no se vuelven a usar estos qubits para nada. Puede parecer que, por tanto, esta información no se está utilizando. Ya estudiamos una situación parecida antes. En ese caso usábamos una CNOT para que el qubit 2 controlara al qubit 1, y vimos como el utilizar esta puerta hacía que los efectos sobre el qubit 2 cambiaran. De la misma manera, el valor del $x - register$ cambia por el hecho de ser utilizado para controlar al $f - register$. Este proceso funciona de la siguiente manera:

- Debido a la aplicación de las puertas de Hadamard, todos los posibles valores de x son igual de probables, y debido a las puertas $a^x (mod\ 15)$ por cada valor de x el $f - register$ contiene uno de los p posibles valores de $f(x)$. Por ejemplo, para $a = 7$ tenemos que $p = 4$ y estos cuatro posibles valores de $f(x)$ son 1,7,4,13. En este punto $|\Psi\rangle$ es una superposición equiprobable de muchos términos de modo que si medimos el $f - register$ el resultado será 1,4,7 o 13, mientras que si

medimos el $x - register$, el resultado puede ser cualquier número entero (lo suficientemente pequeño como para que el $x - register$ lo pueda contener).

- Ahora, supongamos que medimos el $f - register$ y obtenemos 7. De acuerdo con los principios de la física cuántica, una medición causa que el estado $|\Psi\rangle$ colapse para que esté de acuerdo con la medición. Por tanto, tras medir que $f = 7$ solo los estados de x que se correspondan con $f(x) = 7$ se mantendrán en la superposición.
- Es decir, todos los posibles valores del $x - register$ dejan de ser igual de probables y obtenemos, en cambio, una función con picos de probabilidad en estos valores de x específicos. Y como la función $f(x)$ es periódica con periodo $p = 4$, estos picos estarán separados por dicho periodo.
- La IQFT toma la transformada de Fourier de esta función. Como la función tiene periodo 4, la transformada de Fourier contendrá la frecuencia $\omega = 1/4$ y sus armónicos $\omega = 2/4$ y $3/4$. En una transformada de Fourier discreta las frecuencias son de la forma $\omega = \tilde{x}/2^L$. Así, al medir el $x - register$, la probabilidad tendrá picos en $\tilde{x}/2^L = s/4 = s/p$ para todos los enteros armónicos s, el cuál es el resultado que buscamos.

Algoritmo de Shor: El programa

Una vez todas las matrices necesarias están computadas, el diseño del programa es bastante sencillo. Primero, no inicializamos $|\Psi\rangle$ con el primer autoestado, sino con el segundo (que se corresponde con el estado $|0000001\rangle$). En Mathematica lo podemos escribir así:

$\psi = \text{Array}[\text{If}[\# == 2,1,0]\&,2\text{^}n]$

Entonces, aplicamos las puertas cuánticas en el mismo orden que aparecen en el circuito cuántico, resultándonos el siguiente código:

$\text{Do}[\psi = H[i].\psi,\{i,3\}];\text{Do}[\psi = \text{matfx}[[i]].\psi,\{i,3\}];$
$\psi = H[3].\text{cr}[\text{Pi / 2,2,3}].H[2].\text{cr}[\text{Pi / 4,1,3}].\text{cr}[\text{Pi / 2,1,2}].H[1].\psi$

Por último, para la medición cuántica tendremos que cambiar ligeramente en código. Hasta ahora el resultado devuelto ha sido el estado del sistema tras la medición, es decir, un número binario de 7 dígitos (uno por cada qubit). Sin embargo, ahora el valor que nos interesa es $\tilde{x}/2^L$, donde $\tilde{x}$ es un numero decimal dado por el número binario de 3 dígitos $\tilde{x}_2\tilde{x}_1\tilde{x}_0$. Para realizar esta corrección será suficiente con eliminar la última línea del código y sustituirla por:

$\text{FromDigits}[\text{Reverse}[\text{Drop}[\text{IntegerDigits}[\text{FirstPosition}[\psi,1][[1]]\text{-}1,2,n],\text{-}4]],2]/8//\text{N}$

Así, ya tenemos el programa completado. Para finalizar este apartado recogeremos los valores obtenidos para diferentes a.

$\tilde{x}/2^L$	Número de mediciones obtenidas			
	$a = 7$	$a = 8$	$a = 11$	$a = 4$
0	24	19	48	42
0.125	0	0	0	0
0.25	18	20	0	0
0.375	0	0	0	0
0.5	32	27	52	58
0.625	0	0	0	0
0.75	26	34	0	0
0.875	0	0	0	0

Por un lado, para $a = 7$ y $a = 8$ tenemos 4 diferentes resultados, los cuales se corresponden con $s/p = s/4 = 0/4, 1/4, 2/4, 3/4$. Así,

para $a = 7$ tenemos que
$P_+ = \gcd\left(a^{\frac{p}{2}} + 1,\ 15\right) = \gcd(50, 15) = 5$ y $P_- = \gcd\left(a^{\frac{p}{2}} - 1,\ 15\right) = \gcd(48, 15) = 3$,

y para $a = 8$
$P_+ = \gcd\left(a^{\frac{p}{2}} + 1,\ 15\right) = \gcd(65, 15) = 5$ y $P_- = \gcd\left(a^{\frac{p}{2}} - 1,\ 15\right) = \gcd(63, 15) = 5$
tal y como predice la teoría.

Por otro lado, para $a = 11$ y $a = 14$ solo tenemos 2 diferentes resultados, pero como $s/p = s/2 = 0/2, 1/2$,

para $a = 11$ tenemos que
$P_+ = \gcd\left(a^{\frac{p}{2}} + 1,\ 15\right) = \gcd(12, 15) = 3$ y $P_- = \gcd\left(a^{\frac{p}{2}} - 1,\ 15\right) = \gcd(10, 15) = 5$,

y para $a = 4$
$P_+ = \gcd\left(a^{\frac{p}{2}} + 1,\ 15\right) = \gcd(5, 15) = 5$ y $P_- = \gcd\left(a^{\frac{p}{2}} - 1,\ 15\right) = \gcd(3, 15) = 3$
de nuevo obtenemos el resultado esperado.

D. <u>Algoritmo de Shor con N qubits</u>

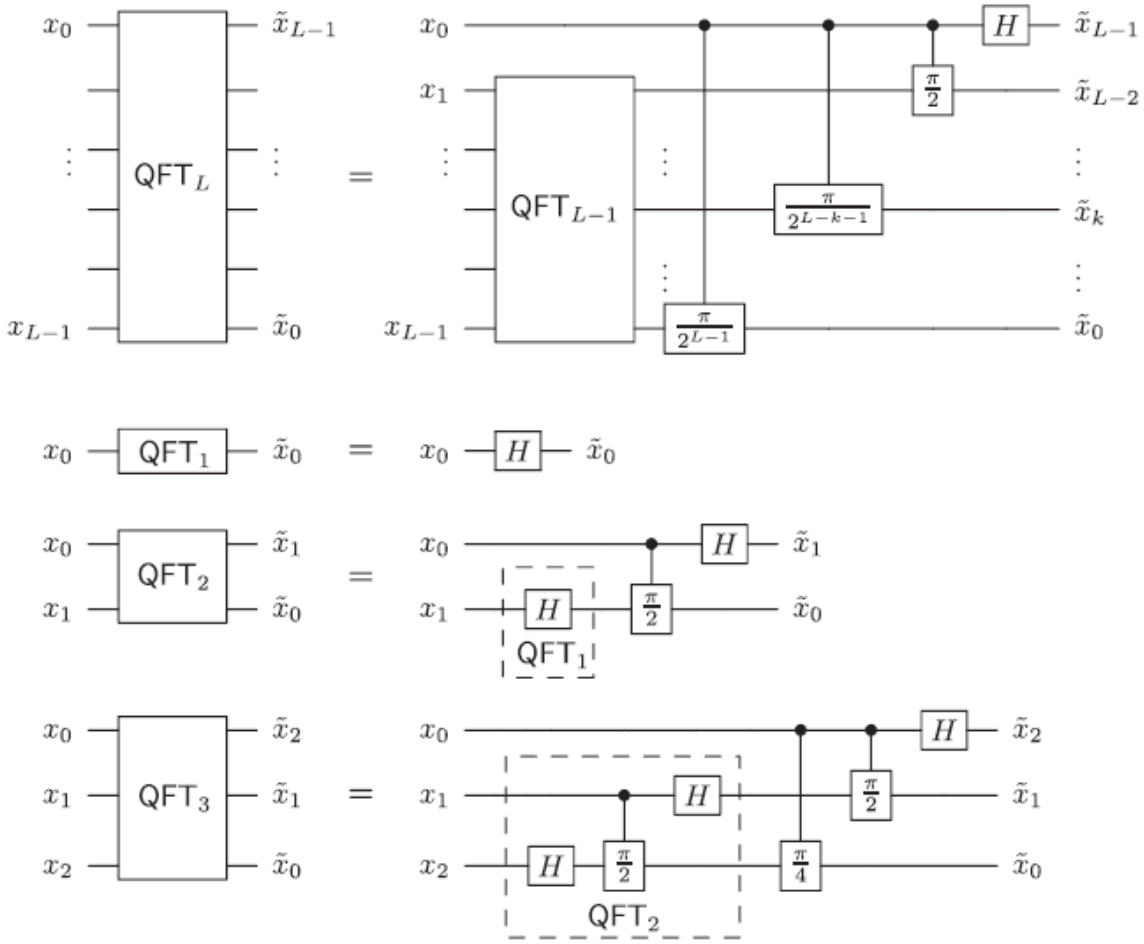
<u>Generalización de las puertas $a^x(mod\ C)$:</u>

Esta parte es la más simple de modificar. Por cómo se construyó el código será suficiente con cambiar referencias específicas a los registros de los qubits (en este caso $L = 3$ y $M = 4$). Es decir, simplemente tendremos que declarar estas 2 nuevas variables y sustituirlas por dichos valores "3" y "4".
A su vez, el poder utilizar más qubits permitirá factorizar números más grandes, por tanto, se declarará una nueva variable más, c. De modo que el código queda:

```
c = 15; M = 4; L = 8; n = M + L; a = 7; A = Table[Mod[a^(2^(i - 1)), c], {i, L}];
f[k_] := FromDigits[Drop[IntegerDigits[k, 2, n], L],2];
matfx =
  Table[
    SparseArray[
      {j_, k_}/; And[IntegerDigits[k - 1,2, n][[L + 1 - i]] == 1,
        FromDigits[Drop[IntegerDigits[k - 1,2, n], L],2] < c,
        j == FromDigits[Join[Drop[IntegerDigits[k - 1,2, n], -M],
          IntegerDigits[Mod[f[k - 1] * A[[i]], c],2, M]],2] + 1] → 1,
      {2^n, 2^n}] +
    SparseArray[
      {j_, k_}/; And[IntegerDigits[k - 1,2, n][[L + 1 - i]] == 0||
        FromDigits[Drop[IntegerDigits[k - 1,2, n], L], 2] ≥ c,
       j == k] → 1,
      {2^n, 2^n}],
  {i, L}]
```

Generalización de la IQFT:

La IQFT que hemos utilizado es específica para $L = 3$ y para generalizarla a cualquier L nos serviremos de la siguiente imagen:



Esta imagen nos muestra cómo obtener de forma recursiva la QFT para cualquier L. Sin embargo, nosotros buscamos la IQFT, y para ello nos aprovecharemos de otra propiedad exclusiva de la computación cuántica; para obtener la función inversa invertiremos el orden en el que se aplicarán las puertas. Como comprobación, podemos ver como la QFT_3 invertida coincide con la IQFT utilizada en el algoritmo de Shor con $L = 3$.

Como el código encargado de crear cada una de estas puertas que forman la IQFT ya lo tenemos, nos faltará crear un código que nos genere una lista con las puertas necesarias en el orden correcto.

Para ello declararemos la siguiente función:

```
IQFT[l_] := (QFT = {H[l]};
   Do[Do[QFT = Join[QFT, {cr[Pi / (2^(i - 1)), l - k + 1, l - k + i]}], {i, k, 2, -1}];
      QFT = Join[QFT, {H[l - k + 1]}], {k, 2, l}];
   Reverse[QFT])
```

De modo que si por ejemplo ejecutamos $\mathrm{IQFT}[3]$ obtenemos:

$$\{H[1], \mathrm{cr}\left[\frac{\pi}{2}, 1,2\right], \mathrm{cr}\left[\frac{\pi}{4}, 1,3\right], H[2], \mathrm{cr}\left[\frac{\pi}{2}, 2,3\right], H[3]\}$$

Que de izquierda a derecha coincide justo con la IQFT utilizada para $L = 3$

Completando el programa

Aunque con esto sería suficiente, añadiremos un cambio más a la parte final del programa. Debido a los grandes recursos computacionales que requieren, no ejecutaremos estas simulaciones completamente. A su vez, nos quedaremos con la función de probabilidad obtenida de las operaciones y analizaremos los resultados que podríamos obtener si realizáramos la simulación hasta el final.
Así, tras declarar todas las funciones y matrices necesarias para este programa, solo nos quedaría aplicar el siguiente código:

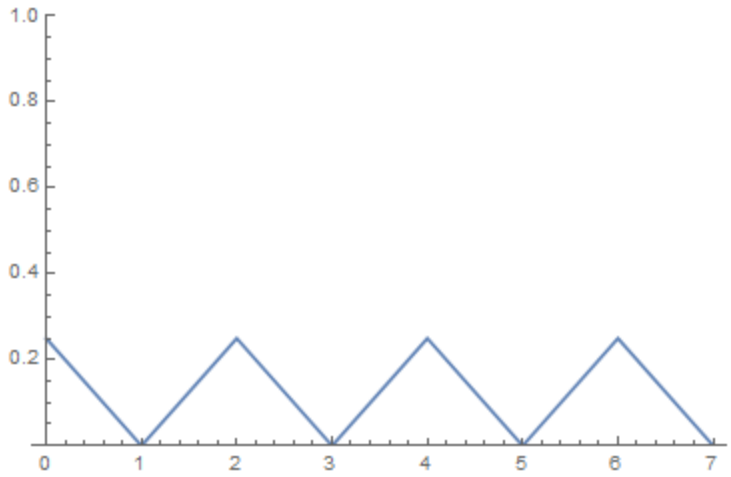
```
ψ = Array[If[# == 2,1,0]&,2^n];
Do[ψ = H[i].ψ,{i,L}]; Do[ψ = matfx[[i]].ψ,{i,L}]; operations = IQFT[L];
Do[ψ = operations[[i]].ψ,{i,Length[operations]}];
ψ = Abs[ψ]^2; plotdata = ψ;
ListLinePlot[
  Table[
    {i,
      Sum[
        plotdata[[
          FromDigits[Join[Reverse[IntegerDigits[i,2,L]],IntegerDigits[j,2,M]],2]+1]],
        {j,0,2^M-1}]},
    {i,0,2^L-1}],
  PlotRange→{0,1}]
```

El código nos devuelve una gráfica que nos especifica la probabilidad sobre 1 (eje Y) de medir cada estado $\tilde{x}$ (eje X).
Por supuesto, esto no tendría ningún sentido en un ordenador cuántico real, ya que no hay manera de acceder al estado del sistema antes de su medición y de su consecuente colapso. Por tanto, esta modificación se hará con fines más didácticos que prácticos. Además, esto nos permitirá un análisis más exacto de los resultados, ya que evitaremos los errores causados por las fluctuaciones inherentes a la aleatoriedad.

Usar una mayor cantidad de qubits nos permite factorizar números más grandes que 15. El número total de qubits es $N = L + M$. El $f - register$ guarda los valores binarios de $a^x(mod\ C)$, por lo que M debe satisfacer $2^M \geq C$. Además, para asegurarse tener una buena precisión a la hora de obtener el periodo p es aconsejable que L cumpla $2^L \geq C^2$. Aun así, podemos ver del apartado anterior que esta condición no siempre es necesaria; para $C = 15$ deberíamos tener como mínimo $L = 8$, pero con $L = 3$ ya hemos obtenido resultados precisos. Esto se debe a que "casualmente" los armónicos $s/p = s/4$ coincidían exactamente con las fracciones de los posibles resultados $s'/2^L = 2s/8 = s/4$.

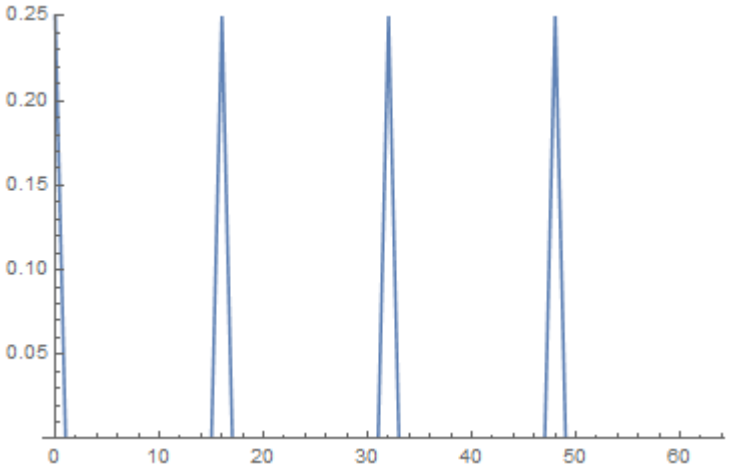
Resultados

Antes de aumentar el número de qubits, a modo de comprobación, empezaremos simulando un caso ya calculado anteriormente, Este se corresponde a $M = 4, L = 3$, $a = 7, c = 15$:



Podemos ver como la gráfica coincide perfectamente con los resultados obtenidos anteriormente. Las mediciones de $\tilde{x} = 1,3,5,7$ tienen una probabilidad 0 mientras que las otras 4 son igual de probables y nos permiten obtener $\tilde{x}/2^L = 0.0, 0.25, 0.5, 0{,}75 = s/4 = s/p$.

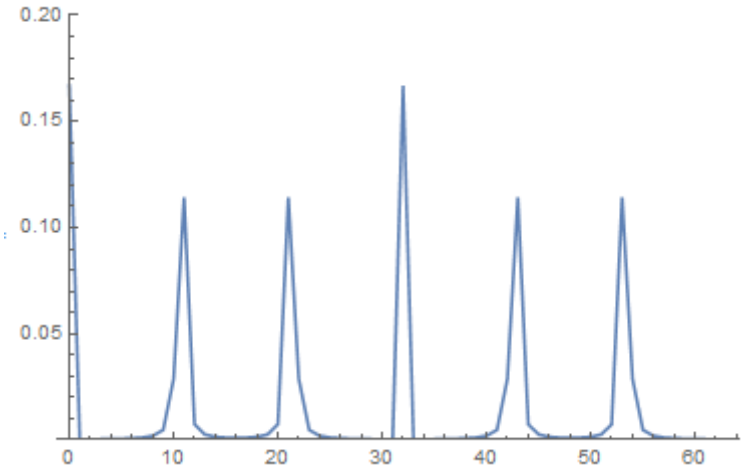
También podríamos utilizar un mayor número de qubits para factorizar el mismo número, $C = 15$. Aumentar M no tendría ningún sentido ya que su "único" papel es contener información y no nos aumentaría la precisión de los resultados. Sin embargo, veamos que pasa si aumentamos L.
$M = 4, L = 6, a = 7, c = 15$



En realidad, el resultado es el mismo, tenemos una probabilidad de 0.25 de obtener $\tilde{x} = 0, 16, 32\ o\ 48$. Lo cual nos encaja de nuevo con, $\tilde{x}/2^L = \tilde{x}/64 = 0.0, 0.25, 0.5, 0{,}75 = s/4 = s/p$. Sin embargo, debido al gran aumento de $\tilde{x}$ posibles totales, vemos como la función toma una forma mucho más picuda ("precisa"). Aunque en este caso no influya en el resultado final, es importante ya que para casos en los que s/p no coincida con $\tilde{x}/2^L$, aumentar L nos permitirá acercarnos más a los valores exactos de s/p.

Por último, aumentaremos aún más el número de qubits, pero en este caso para factorizar un número más grande, $C = 39$.

Para ello utilizamos $M = 6, L = 6, a = 10$.



En este caso vemos que tenemos 6 picos, pero algunos han perdido esa estructura simétrica que vimos anteriormente. Esto se debe a que ahora, algunos de los valores numéricos de $\tilde{x}/2^L$ no coinciden con los de s/p.
Los picos se sitúan justo sobre $\tilde{x} = 0,11,21,32,43,53$, por lo que el ordenador cuántico nos devolvería con mayor probabilidad $\tilde{x}/2^L = 0, 0.1719, 0.3281, 0.5, 0.6719, 0.8281 \approx \frac{0}{6}, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6} = s/p \Longrightarrow p = 6$

Por tanto:
$P_+ = \gcd\left(a^{\frac{p}{2}} + 1,\ C\right) = \gcd(1001, 39) = 13$ y $P_- = \gcd\left(a^{\frac{p}{2}} - 1,\ C\right) = \gcd(999, 39) = 3$,
Como cabía de esperar ya que $39 = 13{\times}3$

Cabe señalar que los picos más altos se corresponden con $\tilde{x} = 0, 32$, que son los únicos valores que coinciden exactamente con los armónicos $0/6$ y $3/6$ respectivamente.
A su vez, vemos que los estados $\tilde{x} = 11, 21, 43, 53$ son menos probables, pero, no son los únicos estados posibles de sus inmediaciones, es decir, aún es posible obtener valores como $\tilde{x} = 10, 12, 20, 22, 42, 44, 52, 54$ ... pero con mucha menos probabilidad.
Dicho de otro modo, los picos no son tan altos, pero son más anchos. Podríamos decir que la probabilidad de medir cada armónico es igual, aunque no siempre con la misma precisión. Aquí es donde se aprecia la importancia de aumentar el valor de L.

E. <u>Reflexiones finales</u>
Al igual que el algoritmo de Grover, está sujeto a cierta aleatoriedad, que puede hacer que obtengamos resultados indeseados.
Aun así, también hemos visto ciertos casos en los que el resultado medido coincide exactamente con el buscado. Más concretamente, siempre que busquemos un periodo p que cumpla $p = 2^n$ para cualquier n entero, podremos obtener los armónicos exactos.
Por otro lado, si p es de la forma $p = n'2^n$, donde $n' \neq 2$ y entero, tendremos una situación parecida a la última, con 2^n picos perfectamente centrados (altos y estrechos), mientras que los otros serán más bajitos y estrechos.
Así, Shor sigue prometiéndonos resolver un problema irresoluble clásicamente.

4. Conclusiones

Además de familiarizarnos con el lenguaje de programación Mathematica; mediante simulaciones, hemos podido ver directamente el funcionamiento de un ordenador cuántico (bajo el paradigma de "quantum gate array") y las consecuencias que conllevaría su futura construcción.

Primeramente, nos hemos servido de sencillos ejemplos para entender el comportamiento general de este tipo de ordenadores. Siempre empezamos inicializando los qubits, asignando a cada uno un valor determinado. Normalmente, empezaremos con los estados $|0 \dots 0\rangle$ o $|0 \dots 1\rangle$. Entonces, a este estado se le aplicarán las puertas cuánticas necesarias para llevar a cabo el algoritmo y finalmente se simulará una medición de los qubits, cuya aleatoriedad se deberá ajustar a las leyes de la mecánica cuántica.

Además, estos sencillos ejemplos también nos han servido para obtener una mejor comprensión de las características principales de algunas de las puertas cuánticas más importantes como $\hat{H}$, $\hat{R}_\theta$ y $\hat{C}_{NOT}$.
Por un lado, hemos visto la enorme importancia de la puerta $\hat{H}$, la cual nos permite obtener una superposición equiprobable de autoestados involucrando a tantos qubits como queramos.
Por otro lado, vimos un fenómeno muy interesante que no tiene análogo clásico. Esto es la manipulación de las probabilidades mediante puertas que por sí solas no pueden hacerlo, de modo que al combinarlas con otras puertas podremos obtener nuevos resultados. Concretamente, vimos como el simple hecho de introducir un desplazamiento de fase entre dos puertas $\hat{H}$ nos permitió cambiar el valor determinado (porque partía de un autoestado) de un cierto qubit y como al utilizar un qubit para controlar otro se veían afectados ambos qubits, en vez de solo el controlado como pasaría clásicamente.

Por último, utilizamos estos programas para simular algoritmos con aplicaciones prácticas reales, como son el algoritmo de Grover y el de Shor.
Mediante el algoritmo de Grover pudimos comprobar la importancia de las puertas $\hat{H}$ y su propiedad de colocar al sistema en un estado equiprobable de autoestados. Esto es lo que nos permite obtener un algoritmo de búsqueda tan eficiente, que siendo D el tamaño de la base de datos, nos encuentra la respuesta correcta con tan solo $\frac{\pi}{4}\sqrt{D}$ iteraciones en vez de las $D/2$ que se necesitarían clásicamente. Cierto es que tenía el inconveniente de no ser tan preciso como cabría de esperar para, por ejemplo, usos comerciales, pero analizamos como esto se debía a una cantidad insuficiente de qubits y no era un problema inherente al algoritmo.
Por ejemplo, en una base de datos con aproximadamente 10^6 entradas (la antigua guía telefónica de Madrid) supondría un ahorro de un factor

$$\frac{2^{20}/_2}{\frac{\pi}{4}\sqrt{2^{20}}} \approx 652$$

en el número de iteraciones con una probabilidad de acierto del 99.999976%.

Con el algoritmo de Shor entendimos la verdadera revolución que podría conllevar en el mundo de la criptografía la construcción de un ordenador cuántico lo suficientemente potente. El algoritmo de Shor nos permite factorizar números con una eficiencia tal, que clásicamente parece imposible poder llegar a acercarse jamás.
Por ejemplo, para descomponer un número de 1000 bits o 302 dígitos tendríamos un ahorro de un factor

$$\frac{O\left(\exp\left(\left(\frac{64}{9}1000\right)^{\frac{1}{3}}(\log(1000))^{\frac{2}{3}}\right)\right)}{O((\log(10^{302}))^3)} \approx 10^{10}$$

También vimos cómo podía haber problemas de precisión, pero que de nuevo estaban relacionados con bajo número de qubits.

Así, podemos concluir que el éxito en la construcción de los ordenadores cuánticos puede conllevar un auténtico salto tecnológico, tanto por las aplicaciones ya descubiertas como las que podrían faltar por descubrir.

5. Bibliografía

Este trabajo está principalmente inspirado por las explicaciones del siguiente artículo: D. Candela, Undergraduate computational physics projects on quantum computing, American Journal of Physics **83**, 688 (2015); doi: http://dx.doi.org/10.1119/1.4922296

Además de desarrollos, resultados y análisis propios, para hacer más completo este trabajo se han utilizado los siguientes enlaces de Wikipedia:
https://en.wikipedia.org/wiki/Bloch_sphere
https://en.wikipedia.org/wiki/Quantum_gate
https://en.wikipedia.org/wiki/Grover%27s_algorithm
https://en.wikipedia.org/wiki/RSA_(cryptosystem)
https://en.wikipedia.org/wiki/Shor%27s_algorithm

Por último, toda la información necesaria para aprender a manipular las funciones de Mathematica tal y como se ha hecho en este trabajo se ha extraído de la página web oficial de Wolfram Mathematica:
https://www.wolfram.com/mathematica/

Apéndice: Explicaciones sobre el código usado

Nociones básicas

Si se parte desde 0, para entender las nociones básicas de Mathematica se recomiendo leer el tutorial “Introducción rápida para programadores”. Si no, con éstas breves explicaciones se pretende explicar la notación básica suficiente del lenguaje para poder entender los programas de este trabajo.

Funciones: Mathematica posee miles de funciones incorporadas. Los argumentos se encierran entre corchetes "[]" y se separan mediante comas ",".

Asignaciones: Como es habitual para asignar un valor a una variable usamos el símbolo “=”.
$x = 3$ asigna el valor 3 a la variable x

Listas: Se abren y cierran mediante llaves “{}“ y los elementos que la componen se separan mediante comas “ , “.
$x = \{7,3,4,9\}$ asigna una lista a la variable x
Para acceder a partes concretas de una lista se utiliza el doble corchete “[[ ]]”
Evaluar $x[[2]]$ nos devolverá 3
Evaluar $x[[-2]]$ nos devolverá 4
Evaluar $x[[2;;3]]$ nos devolverá $\{3,4\}$

Símbolo “ ; ”: Lo usamos para separar diferentes operaciones
$x = 3; y = 4; z = 5$
Tras evaluar una sucesión de operaciones Mathematica solo devolverá la última
Evaluar $x = 3; x = x + 1; x + 6$ nos devolverá 10

Operaciones: Para realizar operaciones entre números se utilizan los símbolos habituales " + "(Suma), " − "(Resta), " ∗ "(Multiplicación), "/"(División) y "^"(Potencia). Para multiplicar matrices se utiliza el punto ". ". Al aplicar las anteriores operaciones a una lista la operación se hará para cada uno de los elementos
Evaluar $x = \{1,2,3,4\}; x = x$^2 nos devolverá $\{1,4,9,16\}$

Patrones: los patrones representas clases de expresiones. El constructo básico de patrón es " _ " y representa cualquier expresión. $x\_$ representa un patrón cuyo valor será nombrado x.
("/." significa “reemplazar en todas partes”)
$\{f[1], g[2], f[5], g[3]\}\ /.\ f[x\_] \text{ -> } x + 5$ devolverá $\{6, g[2], 10, g[3]\}$

Condicionales: "/;" permite introducir condiciones.
$\{f[1], g[2], f[5], g[3]\}/.\ f[x\_]/; x > 2 \text{ -> } x + 5$ devolverá $\{f[1], g[2], 10, g[3]\}$

Declaración de funciones: En Mathematica las declaraciones de funciones son simplemente asignaciones que proporcionan reglas de transformación para patrones.
$f[x\_, y\_] := x + y;\ f[4, a]$ devolverá $4 + a$

Funciones puras: Se indican mediante la terminación & y su primer argumento mediante #.
Evaluar $(\# + 1)\ \&[50]$ devolvería 51

Lista de funciones utilizadas

En el siguiente listado se explicarán cada una de las funciones utilizadas en este programa por orden de aparición. A su vez, las funciones están separadas en bloques según el contexto en el que aparecieron por primera vez.

Para simular la inicialización y medición cuántica

Abs$[z]$: devuelve el valor absoluto del número real o complejo z.

RandomReal[]: devuelve un número "aleatorio" comprendido entre el 0 y el 1.

Do:
Do$[expr, \{i, i_{max}\}]$: Evalúa $expr$ tomando valores sucesivos de i desde 1 hasta i_{max}.
Do$[expr, \{i, i_{min}, i_{max}\}]$: Como antes, pero los valores de i van desde i_{min} hasta i_{max}.
Do$[expr, \{i, i_{min}, i_{max}\}, \{j, j_{min}, j_{max}\}, \ldots]$: Evalúa todas las posibles combinaciones de los índices $i, j, \ldots$

Length$[expr]$: Devuelve el número de elementos en $expr$.

Catch y Throw:
Catch$[expr]$ devuelve el argumento del primer $Throw$ generado en la evaluación de $expr$. Throw$[value]$ detiene la evaluación y devuelve $value$ como valor del primer $Catch$ que lo encierra.

If$[condition, t, f]$: Si $condition$ se evalúa como $True$ devuelve t, si se evalúa como $False$ devuelve f.

Array$[f, n]$: Genera una lista de longitud n, con elementos $f[i]$(i va de 1 hasta n).

Print$[expr]$: Imprime $expr$ en la pantalla

IntegerString$[n, b, len]$: Pasa el número n de base decimal a base b y lo devuelve como una cadena de caracteres de longitud len. Ajusta el tamaño del número a len añadiendo ceros a la izquierda.

FirstPosition$[expr, pattern]$: Devuelve la posición del primer elemento en $expr$ que coincide con $pattern$.

Para simular los primeros circuitos cuánticos completos

Sqrt$[z]$: Devuelve la raíz cuadrada de z.

DiagonalMatrix:
DiagonalMatrix$[list]$: Devuelve una matriz con los elementos de $list$ en la diagonal principal, y 0 en el resto de posiciones.
DiagonalMatrix$[list, k]$: Devuelve una matriz con los elementos de $list$ en la k-ésima diagonal, y 0 en el resto de posiciones.

Counts$[list]$: Devuelve una asociación en la que se asocia cada elemento distinto de $list$ al número de veces que aparece.
Por ejemplo, Counts$[\{a, b, c, a\}]$ devuelve $<|a \to 2, b \to 1, c \to 1|>$

Table:
Table$[expr, n]$: Genera una lista de n copias de $expr$
Table$[expr, \{i, i_{max}\}]$: Genera una lista con los valores de $expr$ cuando i va desde 1 hasta i_{max}.

Table$[expr, \{i, i_{min}, i_{max}\}, \{j, j_{min}, j_{max}\}, \dots]$: Devuelve una lista anidada. La lista asociada a i es la más externa.
Por ejemplo, Table$[10i + j, \{i, 4\}, \{j, 3\}]$ devuelve:
$\{\{11,12,13\}, \{21,22,23\}, \{31,32,33\}, \{41,42,43\}\}$

Para simular el algoritmo de Grover

Round$[x]$: Devuelve el entero más cercano a x.

IntegerDigits$[n, b, len]$: Pasa el número n de base decimal a base b y lo devuelve como una lista de longitud len. Ajusta el tamaño del número a len añadiendo ceros a la izquierda.

IdentityMatrix$[n]$: Devuelve la matriz identidad de dimensión $n \times n$

SparseArray$[\{k_, i_, j_\}/; cond: \to value, \{dim_k, dim_i, dim_j\}]$: Asigna el valor $value$ a las posiciones $\{k, i, j\}$ que hacen que $cond$ se evalúe como $True$.
$k/i/j$ va desde 1 hasta $dim_{k/i/j}$.
Band$[\{i, j\}]$: Representa la secuencia de posiciones de la diagonal que empieza en $\{i, j\}$ en una SparseArray.

Drop:
Drop$[list, n]$: Devuelve $list$ sin sus primeros n elementos.
Drop$[list, -n]$: Devuelve $list$ sin sus últimos n elementos.
Drop$[list, \{n\}]$: Devuelve $list$ sin su n-ésimo elemento.

Para simular el algoritmo de Shor

Exp$[z]$: Devuelve la exponencial de z

FromDigits$[list, b]$: Construye un número en base decimal a partir de la lista de sus dígitos ($list$) en base b.

Mod$[m, n]$: Devuelve el resto de la división m/n.

And$[a, b, \dots]$: Es la función lógica AND. Devuelve $True$ si y solo si todos sus argumentos devuelven $True$. Si no devuelve $False$.

Join$[list_1, list_2, \dots]$: Concatena las listas $list_1, list_2, \dots$

Reverse$[expr]$: Invierte el orden de los elementos en $expr$.

Para crear las gráficas

ListLinePlot$[\{\{x_1, y_1\}, \{x_2, y_2\}, \dots\}]$: Crea una gráfica uniendo mediante líneas rectas el i-ésimo punto (con coordenadas $\{x_i, y_i\}$) con el anterior y el siguiente.

Sum$[f, \{i_{min}, i_{max}\}]$: Evalúa el sumatorio $\sum_{i_{min}}^{i_{max}} f$.
En las gráficas se utiliza para obtener la probabilidad de medir cada posible $\tilde{x}$, el cual es el sumatorio de las posibles combinaciones del $f - register$ para cada posible combinación del $x - register$.

PlotRange: Es una opción para funciones de gráficos que especifica el rango de las coordenadas de la gráfica.
En nuestro caso concreto, $PlotRange \to \{0,1\}$ establece que la coordenada y va desde 0 hasta 1.