

Capítulo

1

Introdução à Programação de Computadores Quânticos

Renato Portugal, Franklin L. Marquezino

Resumo

A computação quântica é um novo paradigma computacional que explora o comportamento quântico para realizar um processamento mais eficiente do que o processamento baseado na lógica binária clássica. Este tutorial apresenta as bases da computação quântica de forma didática com foco no modelo de circuitos quânticos, usa os algoritmos de Grover e Bernstein-Vazirani como exemplos, descreve como implementar algoritmos quânticos e como programar computadores quânticos da IBM usando a interface gráfica IBM Q Experience e o kit de programação quântica Qiskit. Como pré-requisito, é necessário que o leitor tenha conhecimento básico de Álgebra Linear e Python.

Abstract

Quantum computation is a new computational paradigm that exploits the quantum behavior in order to perform a kind of processing that is more efficient than the processing based on classical binary logic. This tutorial presents the foundations of quantum computation in a didactic way with a focus on the quantum circuit model, uses the Grover and Bernstein-Vazirani algorithms as an examples, describes how to implement quantum algorithms, and how to program IBM quantum computers using the graphical interface IBM Q Experience and the quantum kit Qiskit. As a prerequisite, it is necessary that the reader has basic knowledge of linear algebra and Python.

Sumário

1	A base da computação clássica	3
2	Origens e novidades da computação quântica	4
3	Qubit, portas lógicas e circuitos quânticos	5
3.1	Breve revisão de álgebra linear na notação de Dirac	5
3.2	Qubit	7
3.3	Portas lógicas quânticas de 1 qubit	9
3.4	Estados quânticos de 2 ou mais qubits	15
3.5	Portas lógicas quânticas de 2 qubits	19
3.6	Portas lógicas quânticas de 3 ou mais qubits	25
4	Paralelismo quântico e o modelo padrão da computação quântica	29
5	Algoritmo de Bernstein-Vazirani	29
5.1	Análise do algoritmo de Bernstein-Vazirani	31
5.2	Circuito do oráculo U_f de Bernstein-Vazirani	32
5.3	Implementação do algoritmo de Bernstein-Vazirani no <i>composer</i> da IBM	33
6	Algoritmo de Grover	34
6.1	Formulação do problema	34
6.2	Como implementar uma função em um computador quântico?	35
6.3	Descrição do algoritmo de Grover	36
6.4	Circuito (não-econômico) do algoritmo de Grover	36
6.5	Implementação do algoritmo de Grover no <i>composer</i> da IBM	38
6.6	Análise do algoritmo de Grover	40
6.7	Implementação do algoritmo de Grover usando um circuito econômico . .	42
7	Programando os computadores quânticos da IBM	45
7.1	Qasm	45
7.2	Qiskit	46
7.2.1	Escrevendo um programa quântico básico	48
7.2.2	Executando o programa quântico	51
7.3	Implementação do algoritmo de Grover	54
	Referências	59

1. A base da computação clássica

A máquina de Turing é uma descrição abstrata de um modelo computacional útil para determinar o que um computador pode calcular. A máquina tem um fita infinita dividida em células onde podemos gravar 0, 1 ou deixar em branco. Ela tem um cabeçote que pode ler uma célula, mudar o valor e depois se mover para uma das células vizinhas obedecendo a um programa. Este programa consiste de instruções básicas que determinam o movimento do cabeçote e a escrita na fita condicionado ao que foi lido nas últimas células visitadas e ao estado da máquina. O número de estados possíveis deve ser finito. O programa é executado recursivamente até que um comando de parada condicional é encontrado. Se a máquina de Turing não para, não há um resultado da computação. Existem problemas computacionais para os quais não é possível fazer uma máquina de Turing que pare em um tempo finito, como por exemplo o famoso problema da parada [12, 36, 40].

O modelo de circuitos lógicos se baseia em *portas lógicas*, que são dispositivos que implementam funções booleanas básicas, como AND, OR e NOT. Este modelo tem diversas vantagens do ponto de vista de implementação prática e desenvolvimento de computadores e, em especial, na análise do processamento computacional e cálculo da eficiência de um algoritmo. Um circuito lógico de n bits de entrada e 1 bit de saída é equivalente a uma tabela verdade de 2^n linhas. Uma vez que cada linha pode ter duas saídas (0 ou 1), um chip de n bits de um computador clássico pode calcular 2^{2^n} funções booleanas diferentes. Combinando funções booleanas, podemos construir algoritmos com saídas de mais de 1 bit. Uma função booleana genérica de n variáveis pode ser colocada na forma normal disjuntiva (FND) e pode ser convertida em um circuito lógico usando as portas AND, OR e NOT. Uma vez que esta decomposição é útil no contexto da implementação de algoritmos quânticos, vamos dar um exemplo de como obter a FND de uma tabela verdade específica. Fica evidente como se obtém o caso geral.

Seja $f(a, b, c)$ uma função booleana de 3 bits definida pela seguinte tabela verdade:

a	b	c	$f(a, b, c)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Para obter a FND, vamos focar nas saídas iguais a 1. Como há 2 saídas iguais a 1, vamos escrever uma expressão booleana que é disjunção de 2 cláusulas conjuntivas com todas as variáveis, isto é,

$$(a \wedge b \wedge c) \vee (\bar{a} \wedge \bar{b} \wedge c).$$

A primeira cláusula se refere a linha de entrada 001 e saída 1. Para que a cláusula tenha saída 1, devemos negar as variáveis a e b , da seguinte forma, $(\bar{a} \wedge \bar{b} \wedge c)$. A segunda cláusula se refere a linha de entrada 110 e saída 1. Para que a cláusula tenha saída 1,

devemos negar a variável c , da seguinte forma, $(a \wedge b \wedge \bar{c})$. A FND da tabela acima é

$$f(a,b,c) = (\bar{a} \wedge \bar{b} \wedge c) \vee (a \wedge b \wedge \bar{c}).$$

Como este procedimento pode ser feito para uma tabela verdade genérica, concluímos que $\{\wedge, \vee, \neg\}$ é um conjunto universal. O circuito lógico desta expressão booleana está mostrado na figura 1.

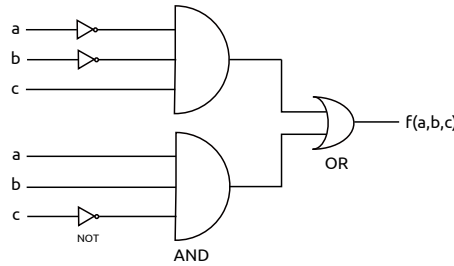


Figura 1. O circuito lógico da função booleana $f(a,b,c)$.

A entrada fica à esquerda do circuito e é representada pelos bits a , b e c . À medida que a informação se propaga para a direita, a porta NOT atua em 1 bit convertendo 0 em 1 e vice-versa. Cada porta AND e OR atua em mais de 1 bit gerando uma única saída à direita, reduzindo sistematicamente de n de bits até 1 bit de saída (0 ou 1), que é o valor de $f(a,b,c)$. A redução sistemática do número de bits requer que $(n - 1)$ bits de informação sejam apagados e, pelas leis da termodinâmica, este processo gera calor por ser um processo irreversível [5, 22].

Entre algumas referências importantes no contexto do modelo de circuitos está a dissertação de mestrado de Claude Shannon [34] e as referências [8, 9, 41].

Nas próximas seções, entramos no contexto quântico. Para adiantar o tema um pouco e frisar algumas diferenças, a entrada e a saída de um circuito quântico têm o mesmo tamanho e o processamento é reversível. Para obter um circuito quântico que implementa uma tabela verdade, devemos usar portas lógicas reversíveis [39]. Como veremos adiante, esta tarefa pode ser realizada através da *porta Toffoli generalizada*.

2. Origens e novidades da computação quântica

Um momento marcante no início da computação quântica foi a palestra *Simulating Physics with Computers* de Richard Feynman na conferência *First Conference on the Physics of Computation* no MIT em 1981, quando ele incentivou os pesquisadores a buscarem um computador que usasse os princípios da mecânica quântica para realização de cálculos, em vez da lógica binária clássica. O principal argumento de Feynman se baseou no fato de que a simulação clássica de sistemas quânticos cresce exponencialmente em função do tamanho do sistema. A questão colocada foi que um computador cujo processamento e memória explorem os fenômenos quânticos poderia simular eficientemente sistemas quânticos. Ficou claro que existia uma difícil tarefa a ser realizada, pois todo o edifício da computação clássica teria que ser refeito a partir de uma nova base.

O primeiro andar ficou pronto em 1985 quando David Deutsch da Universidade de Oxford descreveu formalmente a primeira máquina de Turing universal quântica, que pode simular qualquer outro computador quântico [14]. Sabemos que a máquina de Turing clássica não é adequada para a implementação e o desenvolvimento de algoritmos complexos. O modelo de circuitos e portas lógicas é mais adequado e serve de orientação no momento do desenvolvimento de chips dos processadores. Esta observação também é válida no contexto quântico. O modelo de circuitos e portas lógicas quânticas foi desenvolvido a partir de 1985 [15] e culminou no aparecimento do algoritmo de Shor para fatoração de números inteiros em tempo polinomial em 1994 [28, 31, 35].

A construção de computadores quânticos enfrenta enormes problemas de engenharia para evitar erros quando o número de qubits aumenta [1]. Diversos laboratórios de mecânica quântica construíram computadores quânticos universais com poucos qubits. No entanto, somente a partir de 2017 e 2018, empresas como a IBM, Google e Microsoft anunciaram a construção de computadores quânticos universais com dezenas de qubits. Em 2019, a IBM anunciou um computador quântico comercial de 20 qubits para ser usado em nuvem. Infelizmente, ainda não está claro se os erros estão sob controle. A curto prazo acredita-se que computadores universais baseados em portas lógicas com poucas centenas de qubits com ruídos estarão disponíveis [32]. Entre as principais aplicações no escopo destas máquinas podemos citar a simulação de sistemas quânticos, a simulação de química quântica, a solução de alguns problemas de otimização, a programação linear, a aprendizagem profunda quântica e a teoria de jogos quânticos [32]. Por exemplo, para simular o comportamento de uma molécula de cafeína, o computador clássico deve ter da ordem de 10^{48} bits e o computador quântico da ordem de 160 qubits¹. Infelizmente, a área de algoritmos quânticos requer uma quantidade maior de qubits lógicos estáveis e com pouco ruído (mais de 1000 qubits já com correção de erros e tolerância a falhas). Isto deve demorar várias décadas para ser atingido. Por outro lado, as primeiras demonstrações de supremacia quântica devem ocorrer em breve [10].

3. Qubit, portas lógicas e circuitos quânticos

Antes de definir o conceito de *qubit* (*quantum bit*), vamos fazer uma breve revisão de álgebra linear. Existem diversas notações para mostrar que uma variável v é um vetor, por exemplo, \vec{v} . Na computação quântica, a notação mais usada é a *notação de Dirac*: $|v\rangle$. Uma sequência de vetores é denotada por $|v_0\rangle, |v_1\rangle, |v_2\rangle$ e assim por diante. Na computação quântica, é muito frequente abusar desta notação e denotar a mesma sequência por $|0\rangle, |1\rangle, |2\rangle$ e assim por diante. Referências adicionais que podem ser usadas para esta seção são [33] e [44].

3.1. Breve revisão de álgebra linear na notação de Dirac

A base de um espaço vetorial de duas dimensões é constituída de dois vetores. Na notação de Dirac, a *base canônica* é denotada por $\{|0\rangle, |1\rangle\}$, onde $|0\rangle$ e $|1\rangle$ são vetores bidimensionais ortogonais, isto é, eles têm duas entradas ou componentes e o produto interno entre

¹Veja a apresentação do prof. Ulisses Mello da IBM no Youtube.

$|0\rangle$ e $|1\rangle$ é 0. Estes vetores são dados por

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ e } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Na álgebra linear, esta base é chamada de *base canônica*. Na computação quântica, ela é denominada *base computacional*.

Note que $|0\rangle$ não é o vetor nulo, mas sim o primeiro vetor da base canônica. O vetor nulo é definido pelo vetor com todas as entradas nulas. No caso bidimensional ele é dado por

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

sem nenhuma denominação especial na notação de Dirac.

Um vetor genérico do espaço vetorial bidimensional é obtido através de uma *combinação linear* dos vetores da base e denotado por

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

onde α e β são números complexos. Estes números são as entradas do vetor $|\psi\rangle$, como pode ser visto pela notação matricial

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

O *vetor dual* ao vetor $|\psi\rangle$ é denotado por $\langle\psi|$ e é obtido transpondo o vetor $|\psi\rangle$ e conjugando cada entrada. Usando a equação anterior, obtemos

$$\langle\psi| = (\alpha^* \quad \beta^*),$$

que pode ser escrito como

$$\langle\psi| = \alpha^*\langle 0| + \beta^*\langle 1|,$$

onde

$$\langle 0| = (1 \quad 0) \text{ e } \langle 1| = (0 \quad 1).$$

Podemos ver o vetor dual $\langle\psi|$ como uma matriz 1×2 e o vetor $|\psi\rangle$ como uma matriz 2×1 . Suponha que $|\psi_1\rangle$ e $|\psi_2\rangle$ sejam dois vetores bidimensionais dados por

$$|\psi_1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \text{ e } |\psi_2\rangle = \begin{pmatrix} \gamma \\ \delta \end{pmatrix}.$$

O *produto interno* de $|\psi_1\rangle$ e $|\psi_2\rangle$ é um número complexo denotado por $\langle\psi_1|\psi_2\rangle$ e definido pelo produto matricial do vetor dual $\langle\psi_1|$ pelo vetor $|\psi_2\rangle$, da seguinte forma:

$$\langle\psi_1|\psi_2\rangle = (\alpha^* \quad \beta^*) \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \alpha^*\gamma + \beta^*\delta.$$

Na notação de Dirac, o cálculo do produto interno é feito da seguinte maneira:

$$\langle\psi_1|\psi_2\rangle = (\alpha^*\langle 0| + \beta^*\langle 1|) \cdot (\gamma|0\rangle + \delta|1\rangle) = \alpha^*\gamma\langle 0|0\rangle + \beta^*\delta\langle 1|1\rangle = \alpha^*\gamma + \beta^*\delta.$$

A *norma* do vetor $|\psi_1\rangle$ é denotada por $\| |\psi_1\rangle \|$ e definida como

$$\| |\psi_1\rangle \| = \sqrt{\langle \psi_1 | \psi_1 \rangle} = \sqrt{|\alpha|^2 + |\beta|^2},$$

onde $|\alpha|$ é o *valor absoluto* de α , isto é

$$|\alpha| = \sqrt{\alpha \cdot \alpha^*}.$$

Se $\alpha = a + bi$, onde i é a unidade imaginária ($i = \sqrt{-1}$), a é a parte real e b a parte imaginária, então

$$|\alpha| = \sqrt{(a + bi) \cdot (a - bi)} = \sqrt{a^2 + b^2}.$$

Em espaços vetoriais reais, o produto interno é chamado de *produto escalar* e é dado por

$$\langle \psi_1 | \psi_2 \rangle = \| |\psi_1\rangle \| \| |\psi_2\rangle \| \cos \theta,$$

onde θ é o ângulo formado pelos vetores $|\psi_1\rangle$ e $|\psi_2\rangle$.

Usando estas definições, podemos mostrar que a base de vetores $\{|0\rangle, |1\rangle\}$ é ortonormal, isto é, os vetores $|0\rangle$ e $|1\rangle$ são ortogonais e têm norma 1, ou seja

$$\langle 0|0\rangle = 1, \quad \langle 0|1\rangle = 0, \quad \langle 1|0\rangle = 0, \quad \langle 1|1\rangle = 1.$$

Uma forma algébrica de denotar ortonormalidade e de compactar as quatro últimas equações em uma única é

$$\langle k | \ell \rangle = \delta_{k\ell},$$

onde k e ℓ são bits e $\delta_{k\ell}$ é chamado de *delta de Kronecker* e definido como

$$\delta_{k\ell} = \begin{cases} 1, & \text{se } k = \ell, \\ 0, & \text{se } k \neq \ell. \end{cases}$$

Para maior aprofundamento na área de álgebra linear, sugerimos as seguintes referências: [2, 38, 23]. Para uma compilação de resultados relevantes para a computação quântica, sugerimos o apêndice A do livro [30] e a seção 2.1 do livro [28].

3.2. Qubit

A unidade básica de memória de um computador clássico é o bit, que pode assumir os valores 0 ou 1. Usualmente, o bit é implementado usando o potencial elétrico, seguindo a convenção de que potencial elétrico nulo ou baixo representa o bit 0 e potencial elétrico alto representa o bit 1. No final de um cálculo, é necessário medir o potencial elétrico para determinar se a saída é o bit 0 ou o bit 1.

A unidade básica de memória de um computador quântico é o qubit. No final da computação, uma medição do qubit retorna o valor 0 ou 1 da mesma forma que o bit clássico. A diferença aparece durante a computação, pois o qubit admite a coexistência simultânea dos valores 0 e 1. Durante a computação, isto é, antes da medição, o *estado* de um qubit é representado por um vetor bidimensional de norma 1 e os estados de um qubit correspondentes aos valores 0 e 1 são $|0\rangle$ e $|1\rangle$. O termo *estado* pode sempre ser

identificado com “um vetor de norma 1” e pode ser pensado como o “valor” do qubit antes da medição. A coexistência quântica é representada matematicamente por uma soma de vetores ortogonais da seguinte forma

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

onde α e β são números complexos que obedecem ao vínculo

$$|\alpha|^2 + |\beta|^2 = 1.$$

Isto é, o estado do qubit é o vetor $|\psi\rangle$ de norma 1 com as entradas α e β . Os números complexos α e β são chamados de *amplitudes* do estado $|\psi\rangle$.

A coexistência dos bits 0 e 1 não pode ser implementada em um sistema físico clássico, pois não é possível ter potencial elétrico baixo e alto ao mesmo tempo, como todo mundo sabe. Na mecânica quântica, por mais incrível que pareça, é possível ter um sistema quântico (usualmente microscópico) com potencial elétrico baixo e alto ao mesmo tempo. Esta coexistência só pode ser mantida se o sistema quântico estiver isolado do meio ambiente macroscópico ao redor. Quando medimos o sistema quântico para determinar o valor do potencial elétrico, o aparelho de medição inevitavelmente influencia irreversivelmente no valor do potencial elétrico gerando um resultado estocástico, que é potencial elétrico baixo ou alto, similar ao bit clássico. Ou seja, a coexistência só é mantida quando ninguém estiver observando. Note que a mecânica quântica é uma *teoria científica*, isto é, suas leis e resultados podem ser testados de forma objetiva em laboratório. Além disso, leis e afirmações desnecessárias são sumariamente descartadas. Portanto, a afirmação “a coexistência só é mantida quando ninguém estiver observando” tem consequências práticas e é uma afirmação testada e re-testada por mais de 100 anos em milhares de laboratórios de mecânica quântica no mundo. Por outro lado, teorias alternativas mais palatáveis classicamente, que poderiam ajudar na compreensão ou visualização da superposição quântica, foram descartadas por testes experimentais.

Do ponto de vista computacional, podemos ter um qubit em superposição e usar este fato em um circuito. Por exemplo, o circuito

$$|\psi\rangle \longrightarrow \boxed{\diagup} \Longrightarrow 0 \text{ ou } 1$$

indica que o “valor” inicial do qubit é $|\psi\rangle$ e esta informação foi transmitida inalterada da esquerda para a direita até que uma medição do potencial elétrico foi feita, como indicado pelo *medidor* (o visor de um voltímetro). O resultado da medição é 0 ou 1. A informação clássica é mostrada pela linha dupla. Se o estado do qubit for $|0\rangle$, uma medição resulta necessariamente em 0 e se o estado do qubit for $|1\rangle$, uma medição resulta necessariamente em 1. No caso geral, se o estado do qubit for $\alpha|0\rangle + \beta|1\rangle$, uma medição resulta em 0 com probabilidade $|\alpha|^2$ e em 1 com probabilidade $|\beta|^2$, como mostrado no circuito

$$\alpha|0\rangle + \beta|1\rangle \longrightarrow \boxed{\diagup} \Longrightarrow \begin{cases} 0, \text{ com probabilidade } |\alpha|^2, \\ 1, \text{ com probabilidade } |\beta|^2. \end{cases}$$

A saída pode ser vista com um histograma da distribuição de probabilidades. É importante repetir o fato de que α e β são chamados de *amplitudes* do estado $\alpha|0\rangle + \beta|1\rangle$ e são

números que podem ser negativos e ter parte imaginária. Por outro lado, $|\alpha|^2$ e $|\beta|^2$ são números reais positivos no intervalo $[0, 1]$ e são chamados de probabilidades. Confundir amplitude com probabilidade gera erros imperdoáveis.

O estado de um qubit pode ser caracterizado por dois ângulos θ e φ da seguinte maneira:

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle,$$

onde $0 \leq \theta \leq \pi$ e $0 \leq \varphi < 2\pi$. Esta notação mostra que existe uma correspondência biunívoca entre o conjunto de estados de 1 qubit e pontos na superfície de uma esfera de raio 1, chamada *esfera de Bloch*. Os ângulos θ e φ são ângulos esféricos que descrevem a posição do estado $|\psi\rangle$, como mostrado na figura 2. O ponto na esfera de Bloch é descrito por um vetor tridimensional de entradas reais dado por

$$\begin{pmatrix} \sin \theta \cos \varphi \\ \sin \theta \sin \varphi \\ \cos \theta \end{pmatrix}.$$

A figura 2 mostra a posição dos estados $|0\rangle$ e $|1\rangle$, cujos ângulos esféricos são $(\theta, \varphi) = (0, 0)$ e $(\theta, \varphi) = (\pi, 0)$.

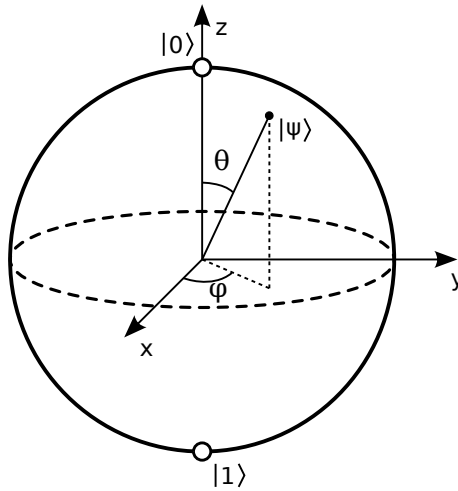


Figura 2. Esfera de Bloch (Fonte: Wikipedia)

3.3. Portas lógicas quânticas de 1 qubit

Uma porta lógica de 1 qubit é uma matriz quadrada² bidimensional unitária. Uma matriz de duas dimensões U é unitária se a multiplicação de U por um vetor de norma 1 arbitrário resulta em um vetor de norma 1. Mais formalmente, seja $|\psi'\rangle = U|\psi\rangle$, onde $|\psi\rangle$ é um vetor bidimensional de norma 1. Se U for uma matriz unitária, então $|\psi'\rangle$ tem norma 1. Por exemplo, a matriz de Hadamard

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

²É muito comum o uso do termo *operador* no lugar de matriz quadrada.

é unitária. Portanto, a multiplicação de H pelos vetores da base tem que dar vetores de norma 1. De fato,

$$H|0\rangle = H\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

Note que o vetor resultante tem norma 1. Vamos denotar este vetor por $|+\rangle$, isto é

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

A multiplicação de H por $|1\rangle$ resulta no vetor $|-\rangle$ definido como

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle,$$

que também tem norma 1. Estes cálculos são importantes para saber a saída da porta H . Se a entrada for $|0\rangle$, a saída será $|+\rangle$. Se a entrada for $|1\rangle$, a saída será $|-\rangle$. Se a entrada for uma superposição de $|0\rangle$ e $|1\rangle$, a saída será a mesma superposição de $|+\rangle$ e $|-\rangle$.

Verificar que a multiplicação de H pelos vetores de uma base ortonormal resulta em vetores de norma 1 não é suficiente para mostrar que H é uma matriz unitária. Além disso, é necessário mostrar que os vetores resultantes são ortogonais, isto é, mostrar que $\langle - | + \rangle = 0$.

Um circuito quântico é uma forma pictórica de descrever um algoritmo quântico. O qubit de entrada fica à direita e a informação flui inalterada da esquerda para a direita até encontrar uma porta lógica. A porta lógica recebe a entrada pela esquerda, processa a informação, e ela sai pela direita e continua seu fluxo. O processamento da porta é feito pela multiplicação da matriz unitária que representa a porta pelo vetor que representa o estado do qubit na entrada. Por exemplo, a expressão $|+\rangle = H|0\rangle$ é representada pelo seguinte circuito:

$$|0\rangle \longrightarrow \boxed{H} \longrightarrow |+\rangle.$$

A entrada é o vetor $|0\rangle$, que é transportado inalterado pelo fio até a porta H , que age ou atua na entrada e a transforma em $|+\rangle$, que é transportado até a saída à direita. A ação ou atuação de uma porta é calculada pelo produto matricial da matriz que representa a porta pelo vetor de entrada. Portanto, o resultado da computação é o vetor $|+\rangle$. Se ao final da computação fizermos uma medição, a representação é

$$|0\rangle \longrightarrow \boxed{H} \longrightarrow \boxed{\text{medição}} \longrightarrow \begin{cases} 0, & \text{com probabilidade } \frac{1}{2}, \\ 1, & \text{com probabilidade } \frac{1}{2}. \end{cases}$$

A medição de 1 qubit no estado $|+\rangle$ resulta em 0 com probabilidade 1/2 ou 1 com a mesma probabilidade. A figura 3 mostra o histograma da distribuição de probabilidades gerado no Qiskit com duas iterações.

Neste ponto já podemos usar o simulador de circuitos quânticos (*composer*) dos computadores quânticos da IBM. Após abrir a página eletrônica da IBM³, o usuário deve

³<https://quantum-computing.ibm.com/>

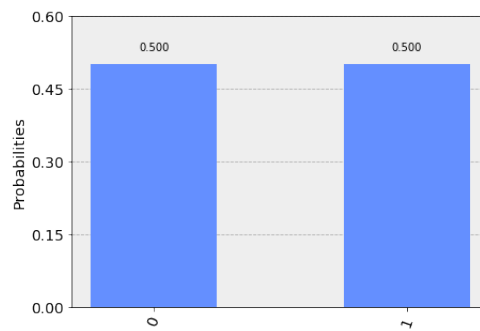


Figura 3. Histograma da distribuição de probabilidades da porta H

se registrar para usar este simulador. Depois de cadastrar uma senha e abrir a página principal, o usuário de clicar em `Create a circuit →`. O *composer* da IBM é muito simples, pois podemos pegar as portas lógicas disponibilizadas e arrastar para o circuito. Vamos nos restringir a um uso bem básico no momento. A figura 4 mostra o circuito da porta H já pronto no *composer*. Este circuito pode ser facilmente feito, pegando a porta H no conjunto de portas e arrastando para o primeiro qubit do circuito. Em seguida devemos arrastar o medidor e soltá-lo depois da porta H . A seta do medidor indica que a saída foi direcionada para um registrador clássico.

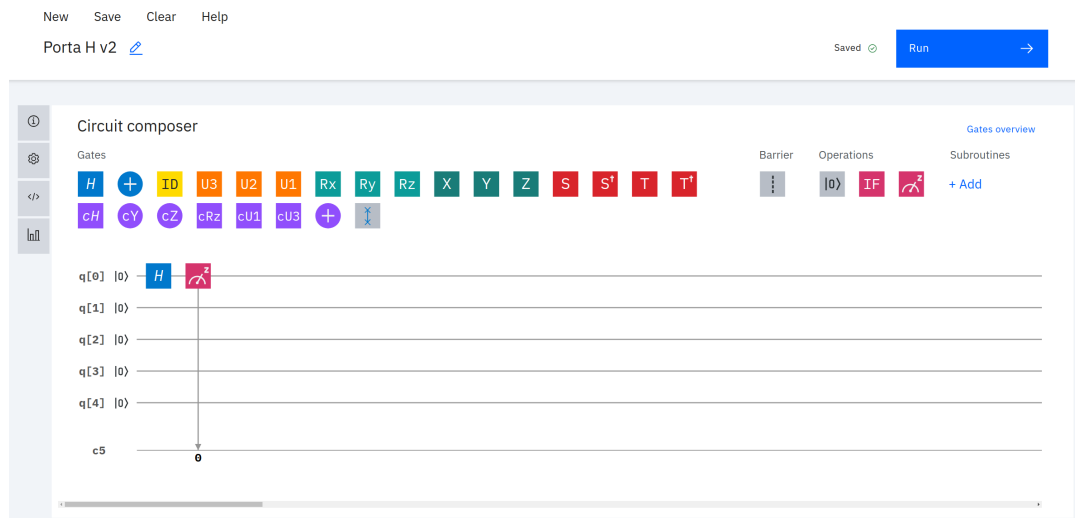


Figura 4. Exemplo de circuito usando a porta H e um medidor (Reprint Courtesy of IBM Corporation ©)

Depois do circuito pronto, temos duas opções: (1) rodar o programa no computador quântico clicando em `Run` e selecionando *ibmq_ourense* (*ibmqx2*, *ibmq_vigo* e outros *backends* também servem) ou (2) simular o programa clicando em `Run` e selecionando o *ibmqasm_simulator*. A opção `Run` só fica ativa após o circuito ser salvo. Para que o resultado saia rápido, é melhor usar a segunda opção. A saída da execução está mostrada na figura 5. O resultado 00000 foi obtido em 50.977% das execuções e o resultado 00001 foi obtido em 49.023%. Os quatro primeiros bits devem ser descartados, pois se referem aos qubits que não foram usados. A saída usa a ordem dos bits menos significativos à

direita, como é usual na computação clássica. O *composer* roda o experimento diversas vezes (até 8192) e mostra o histograma da distribuição de probabilidades. No caso da porta H , o resultado mais provável é 50% cada, porém resultados próximos de 50% têm probabilidades não-desprezíveis e ocorrem com frequência. Para obter resultados mais próximos de 50%, temos que aumentar o número de repetições.

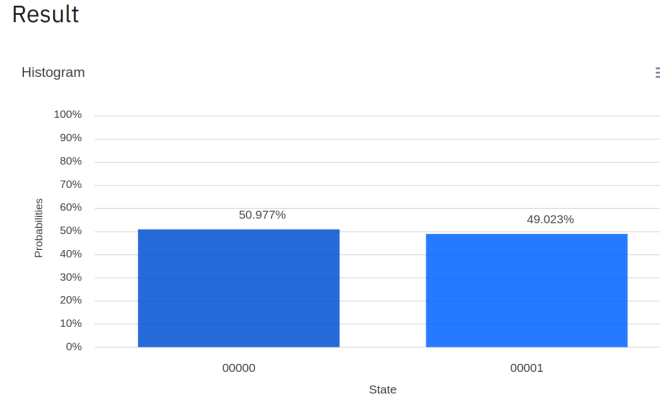


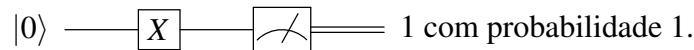
Figura 5. Saída do circuito usando a porta H e um medidor (Reprint Courtesy of IBM Corporation ©)

Se o usuário optar pela opção **Run** e selecionar *ibmq_ourense*, a tarefa entra na fila e pode demorar muito. O tamanho da fila pode ser visto na coluna à direita da página principal. Além disso, o usuário gasta unidades (que é logo recuperada). A saída do computador quântico usualmente é diferente do simulador, pois os erros degradam os resultados. Aumentar o número de repetições não garante que a distribuição de probabilidades tenda para a distribuição correta. É importante usar o simulador antes para testar a corretude do circuito com algumas casas decimais.

Um exemplo mais simples do que o anterior é a porta X dada pela matriz

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

X é a porta NOT quântica, pois $|1\rangle = X|0\rangle$ e $|0\rangle = X|1\rangle$. Podemos verificar estas equações através da multiplicação matricial de X com $|0\rangle$ e $|1\rangle$. De forma mais compacta, podemos escrever $|j \oplus 1\rangle = X|j\rangle$, onde \oplus é a operação XOR ou soma módulo 2. Por causa disto, a porta X também é representada como \oplus . Um circuito usando a porta X é



A simulação no *composer* deverá resultar em uma única saída com probabilidade 1. Aqui vai uma dica sobre o *composer*: Se você quiser apagar uma porta colocada no circuito, clique na porta e depois clique no \times . Se não quiser aproveitar o circuito anterior, faça um novo.

Agora podemos fazer um experimento mais complexo. Como gerar uma superposição tal que as amplitudes do estado $\alpha|0\rangle + \beta|1\rangle$ sejam diferentes? Por exemplo,

como gerar o estado $\alpha|0\rangle + \beta|1\rangle$ tal que $|\alpha|^2 = 25\%$ e $|\beta|^2 = 75\%$? A resposta é usar a porta U_3 . A expressão algébrica desta porta é

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos \frac{\theta}{2} & -e^{i\lambda} \sin \frac{\theta}{2} \\ e^{i\phi} \sin \frac{\theta}{2} & e^{i(\lambda+\phi)} \cos \frac{\theta}{2} \end{bmatrix}.$$

Portanto, aplicando a porta $U_3(\theta, 0, 0)$ no estado $|0\rangle$, obtemos

$$U_3(\theta, 0, 0)|0\rangle = \cos \frac{\theta}{2}|0\rangle + \sin \frac{\theta}{2}|1\rangle.$$

Devemos escolher $\theta = 2\pi/3$, pois $\sin^2(\pi/3) = 3/4$. Para digitar os ângulos no *composer*, clique na porta U_3 e depois clique no lápis. Note que podemos usar π como entrada. Por exemplo, $\theta = 2\pi/3$ pode ser digitado como “2*pi/3”. Os outros ângulos devem ser zero.

U_3 é uma porta coringa, pois ela substitui todas as outras portas de 1 qubit. Infelizmente, esta porta é perfeita demais para ser verdade. Por exemplo, poderíamos a princípio escolher θ tão pequeno quanto desejarmos a um custo de uma única porta. No mesmo contexto, o *composer* disponibiliza duas outras portas de 1 qubit usando ângulos arbitrários, que são

$$U_1(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}, \quad U_2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{bmatrix}.$$

Note que $U_1(\lambda) = U_3(0, 0, \lambda)$ e $U_2(\phi, \lambda) = U_3(\pi/2, \phi, \lambda)$.

Para a determinação da complexidade computacional de um algoritmo, temos que nos restringir a um conjunto finito de portas de 1 qubit. Algumas das portas de 1 qubit disponíveis no *composer* são

$$\text{ID} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

conhecidas como *matrizes de Pauli*,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, \quad T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}$$

conhecidas como porta Hadamard, porta fase, porta fase conjugada, porta $\pi/8$ ou porta T e sua conjugada. O símbolo \dagger é a notação da matriz transposta conjugada (transponha a matriz e conjugue cada entrada). Os números complexos $e^{\pm i\pi/4}$ são iguais a

$$e^{\pm i\pi/4} = \frac{1 \pm i}{\sqrt{2}}.$$

Todo circuito quântico sem os medidores tem uma forma algébrica equivalente. Por exemplo, se A , B e C são portas de 1 qubit, o circuito

$$|0\rangle \text{ — } \boxed{A} \text{ — } \boxed{B} \text{ — } \boxed{C} \text{ — } |\psi\rangle$$

tem a seguinte expressão algébrica equivalente

$$|\psi\rangle = C \cdot B \cdot A \cdot |0\rangle,$$

onde a operação \cdot é o produto matricial, que usualmente é omitido. Note que a expressão algébrica equivalente ao circuito tem a ordem inversa das portas. Portanto, o circuito acima também pode ser mostrado como

$$|0\rangle \longrightarrow \boxed{CBA} \longrightarrow |\psi\rangle,$$

onde CBA é uma matriz 2×2 . Por exemplo, os seguintes circuitos são equivalentes:

$$\longrightarrow \boxed{H} \longrightarrow \boxed{X} \longrightarrow \boxed{H} \longrightarrow = \longrightarrow \boxed{Z} \longrightarrow ,$$

pois $Z = HXH$. A expressão algébrica equivalente pode ser usada para prever a saída do circuito.

Exercício 1. Mostre que $|\ell \oplus 1\rangle = X|\ell\rangle$, $(-1)^\ell |\ell\rangle = Z|\ell\rangle$, $(-1)^\ell i |\ell \oplus 1\rangle = Y|\ell\rangle$ onde ℓ é um bit.

Exercício 2. Calcule $H \cdot |0\rangle$, $Z \cdot H \cdot |0\rangle$, $S \cdot H \cdot |0\rangle$, $Z \cdot S \cdot H \cdot |0\rangle$. Ache os pontos na esfera de Bloch que correspondem a estes vetores.

Exercício 3. Usando o o simulador do composer da IBM, obtenha a saída do seguinte circuito:

$$|0\rangle \longrightarrow \boxed{X} \longrightarrow \boxed{\text{medidor}} =$$

Mostre que o resultado está correto.

Exercício 4. Usando o simulador do composer da IBM, obtenha a saída do seguinte circuito:

$$|0\rangle \longrightarrow \boxed{X} \longrightarrow \boxed{H} \longrightarrow \boxed{\text{medidor}} =$$

O resultado deveria dar necessariamente 0 com probabilidade 0.5 e 1 com probabilidade 0.5?

Exercício 5. PARTE 1. Usando o simulador do composer da IBM, tente mostrar que os seguintes circuitos não são equivalentes:

$$\longrightarrow \boxed{H} \longrightarrow \neq \longrightarrow \boxed{H} \longrightarrow \boxed{Z} \longrightarrow .$$

Atenção: você deve usar apenas resultados de simulações (não faça cálculos). Você concluiu que os circuitos não são equivalentes? Veja nota de rodapé⁴.

PARTE 2. Calcule algebricamente a saída de ambos circuitos sem o medidor e mostre que as distribuições de probabilidades são iguais.

⁴Se você concluiu que os circuitos são equivalentes ou não são equivalentes em qualquer caso a sua conclusão está errada, pois teoricamente as distribuições de probabilidades são iguais e não dá para concluir nada a partir dos resultados do composer.

PARTE 3. Coloque uma porta H no final de ambos circuitos da seguinte forma:

$$\text{---} \boxed{H} \text{---} \boxed{H} \text{---} \neq \text{---} \boxed{H} \text{---} \boxed{Z} \text{---} \boxed{H} \text{---} ,$$

e conclua que os circuitos originais não são equivalentes usando o *composer* (sem fazer cálculos).

Exercício 6. Mostre algebricamente que a saída do circuito

$$|0\rangle \text{---} \boxed{H} \text{---} \boxed{T} \text{---} \boxed{H} \text{---}$$

é

$$\frac{\sqrt{2}+1+i}{2\sqrt{2}}|0\rangle + \frac{\sqrt{2}-1-i}{2\sqrt{2}}|1\rangle.$$

Compare a previsão teórica com o resultado do *composer* da IBM.

Exercício 7. a) Obtenha a porta H a partir de $U_2(\phi, \lambda)$ e $U_3(\theta, \phi, \lambda)$.

b) Obtenha as portas X , Y e Z a partir de $U_3(\theta, \phi, \lambda)$.

c) Obtenha as portas S , S^\dagger , T e T^\dagger a partir de $U_1(\lambda)$.

3.4. Estados quânticos de 2 ou mais qubits

O estado de 2 qubits é descrito por um vetor de norma 1 que pertence a um espaço vetorial de 4 dimensões, pois após a medição dos qubits podemos ter 4 resultados: 00, 01, 10, 11. O primeiro bit se refere ao primeiro qubit e o segundo bit ao segundo qubit, ou seja, adotamos a convenção inversa de representar o bit mais significativo à esquerda⁵. Seguindo as leis da mecânica quântica, devemos indexar a base canônica com os possíveis resultados da seguinte forma:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

No caso clássico, podemos ter dois bits com valores 00 ou 01 ou 10 ou 11 de forma exclusiva. No caso quântico, o estado de 2 qubits é uma superposição da forma

$$|\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle,$$

onde a_0 , a_1 , a_2 e a_3 são números complexos. A medição dos qubits resulta em 00 ou 01 ou 10 ou 11 de forma exclusiva e estocástica. Isto é, não há como saber qual será o resultado da medição mesmo conhecendo $|\psi\rangle$. Porém, se conhecemos $|\psi\rangle$ então sabemos

⁵O gráfico de probabilidades da saída do *composer* da IBM adota a ordem usual (bits menos significativos à direita).

quais são as probabilidades:

$$\begin{aligned}\text{prob}(00) &= |a_0|^2, \\ \text{prob}(01) &= |a_1|^2, \\ \text{prob}(10) &= |a_2|^2, \\ \text{prob}(11) &= |a_3|^2.\end{aligned}$$

Se não conhecemos o estado $|\psi\rangle$ de 2 qubits, uma única medição não permite a determinação de $|\psi\rangle$, isto é, a determinação dos coeficientes a_0, a_1, a_2 e a_3 . Existe um teorema importante na computação quântica conhecido como *teorema da não-clonagem* [16, 29, 43], que é formulado da seguinte maneira:

Teorema. *Não é possível criar uma cópia idêntica de um estado quântico desconhecido.*

Este teorema restringe severamente qualquer possibilidade de determinação de $|\psi\rangle$ através de medições. No entanto, se pudermos gerar $|\psi\rangle$ novamente, por exemplo, através de um circuito, podemos repetir o processo todo diversas vezes e obter uma aproximação para $\text{prob}(00)$, $\text{prob}(01)$, $\text{prob}(10)$ e $\text{prob}(11)$. Por exemplo, repetindo 1000 vezes, podemos determinar estas probabilidades com 2 dígitos precisos. Infelizmente, ainda assim não podemos determinar $|\psi\rangle$ exatamente, pois conhecer $|a_0|^2$ não permite determinar a_0 exatamente. Pode parecer que isto não merece importância—falso. Vamos dar um exemplo crítico. Suponha que

$$\text{prob}(00) = \frac{1}{2}, \quad \text{prob}(01) = 0, \quad \text{prob}(10) = 0, \quad \text{prob}(11) = \frac{1}{2}.$$

Temos pelos menos duas possibilidades para $|\psi\rangle$:

$$|\psi_1\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad \text{e} \quad |\psi_2\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}.$$

Note que $|\psi_1\rangle$ e $|\psi_2\rangle$ são ortogonais. Isto mostra que podemos cometer um erro sério e dificulta a verificação da equivalência de circuitos usando implementações no *composer*.

Neste ponto, a seguinte questão é relevante: Suponha que o estado $|\psi\rangle$ de 2 qubits é conhecido, é possível determinar o estado de cada qubit? A resposta é “depende de $|\psi\rangle$ ”. Se $|\psi\rangle$ for um dos estados da base computacional então sabemos o estado de cada qubit. Por exemplo, suponha que $|\psi\rangle = |10\rangle$. Temos que fatorar $|\psi\rangle$ da seguinte forma:

$$|10\rangle = |1\rangle|0\rangle = |1\rangle \otimes |0\rangle,$$

onde \otimes é chamado de *produto de Kronecker*. Quando a fatoração é bem sucedida, sabemos o estado de cada qubit. Neste caso, o primeiro qubit está no estado $|1\rangle$ e o segundo no estado $|0\rangle$. Quando escrevemos $|1\rangle|0\rangle$, o produto de Kronecker fica subentendido.

O produto de Kronecker⁶ de dois vetores ou duas matrizes é definido da seguinte

⁶Existe uma formulação mais abstrata do produto de Kronecker que é denominada *produto tensorial*. Neste curso, usamos estes termos como sinônimos. A notação $A \otimes B$ é lida da forma “A tensor B”, no entanto os termos *tensor* e *produto tensorial* não têm nenhuma relação com tensores (generalização de matrizes) ou produto de tensores.

forma. Seja A uma matriz $m \times n$ e B uma matriz $p \times q$. Então

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ & \ddots & \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

A dimensão da matriz resultante é $mp \times nq$. O produto de Kronecker dos vetores bidimensionais $|1\rangle$ e $|0\rangle$ é calculado interpretando estes vetores como matrizes 2×1 e é dado por

$$|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle.$$

Note que o produto de Kronecker não é comutativo. Por exemplo, $|1\rangle \otimes |0\rangle \neq |0\rangle \otimes |1\rangle$.

As leis da mecânica quântica afirmam que se o estado do primeiro qubit for $|\psi_1\rangle$ e o estado do segundo qubit for $|\psi_2\rangle$ então o estado $|\psi\rangle$ do sistema composto é

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle.$$

Sempre podemos obter o estado do sistema composto quando conhecemos os estados das partes. Porém, a direção inversa não é possível em geral. Por exemplo, suponha que o estado de 2 qubits seja

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Queremos encontrar estados de 1 qubit $|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle$ e $|\psi_2\rangle = \gamma|0\rangle + \delta|1\rangle$ tal que

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle).$$

Expandindo o lado direito e igualando os coeficientes, obtemos o seguinte sistema de equações:

$$\begin{aligned} \alpha\gamma &= \frac{1}{\sqrt{2}}, \\ \alpha\delta &= 0, \\ \beta\gamma &= 0, \\ \beta\delta &= \frac{1}{\sqrt{2}}. \end{aligned}$$

Como este sistema de equações não admite nenhuma solução, o estado $|\psi\rangle$ de 2 qubits não pode ser escrito como o produto de Kronecker de estados de 1 qubit. Aqui vai mais uma estranheza quântica:

Fato. *Um sistema quântico composto pode estar em um estado bem definido enquanto que as partes do sistema não têm nenhum estado definido.*

Um estado quântico de um sistema composto que não pode ser fatorado usando o produto de Kronecker é chamado *estado emaranhado*. Os estados emaranhados são muito importantes na computação quântica, pois sem eles o poder computacional do computador quântico seria o mesmo do computador clássico. No entanto, é importante frisar que a presença de um estado emaranhado em um algoritmo quântico para resolver um problema não garante que este algoritmo é mais eficiente do que um algoritmo clássico para o mesmo problema.

Podemos generalizar a discussão desta seção para n qubits, onde $n \geq 1$. A base computacional é constituída por vetores de 2^n entradas

$$|0 \cdots 00\rangle = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad |0 \cdots 01\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \cdots, \quad |1 \cdots 11\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Note que o número de cadeias de bits é 2^n . Cada cadeia pode ser escrita na notação decimal como

$$|0 \cdots 00\rangle = |0\rangle, \quad |0 \cdots 01\rangle = |1\rangle, \quad |0 \cdots 10\rangle = |2\rangle, \quad \dots \quad |1 \cdots 11\rangle = |2^n - 1\rangle.$$

Um estado genérico pertence a um espaço vetorial de 2^n dimensões, e, na notação decimal, ele é escrito como

$$|\psi\rangle = a_0 |0\rangle + a_1 |1\rangle + a_2 |2\rangle + \cdots + a_{2^n-1} |2^n - 1\rangle,$$

onde

$$|a_0|^2 + |a_1|^2 + |a_2|^2 + \cdots + |a_{2^n-1}|^2 = 1.$$

Após uma medição dos n qubits, obtemos como resultado uma cadeia de n bits de forma estocástica. O resultado é a cadeia $0 \cdots 00$ com probabilidade $|a_0|^2$ ou é a cadeia $0 \cdots 01$ com probabilidade $|a_1|^2$ e assim por diante.

Um vetor da base computacional de n qubits pode ser escrito como produto de Kronecker de n vetores de 1 qubit. Por exemplo, no caso de $n = 3$ qubits, podemos obter o segundo vetor da base computacional através do produto de Kronecker da seguinte maneira:

$$|0\rangle \otimes |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |001\rangle.$$

Na notação decimal, $|0\rangle$ pode ser confundido com o estado de 1 qubit. Para evitar a confusão, temos que saber qual é o número de qubits em questão. Por exemplo, se $|0\rangle$ se refere ao estado de 3 qubits na notação decimal, então a descrição deve ser convertida para $|000\rangle$.

Exercício 8. Mostre que $|0\rangle \otimes |1\rangle \neq |1\rangle \otimes |0\rangle$ (\otimes não é comutativo) e $(|0\rangle \otimes |1\rangle) \otimes |1\rangle = |0\rangle \otimes (|1\rangle \otimes |1\rangle)$. (\otimes é associativo)

Exercício 9. Por definição, $M^{\otimes n} = M \otimes \cdots \otimes M$ (n termos), onde M é uma matriz de qualquer dimensão. Calcule $I_2^{\otimes 3}$, $X^{\otimes 3}$, $I_2 \otimes X$, $X \otimes I_2$, $I_2 \otimes Z$, $Z \otimes I_2$.

Exercício 10. Diga se é verdadeiro ou falso: a) $\langle 01|10\rangle = 1$, b) $\langle 101|111\rangle = 1$, c) $\langle +|- \rangle = 1$, d) $\langle 0|+\rangle = 0$, e) $\langle 10|++\rangle = 1/2$, f) $\langle ++|+-\rangle = 0$, g) $\langle 111|--+\rangle = -1/2$.

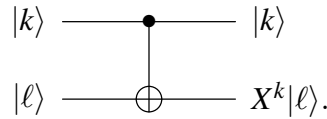
Exercício 11. Seja $*$ o produto de números, \cdot o produto matricial e \otimes o produto de Kronecker. Diga se é verdadeiro ou falso ou inconsistente: a) $\langle 01|10\rangle = \langle 0|1\rangle * \langle 1|0\rangle$, b) $|0\rangle\langle 1| = |0\rangle * \langle 1|$, c) $|0\rangle\langle 1| = |0\rangle \cdot \langle 1|$, d) $|0\rangle\langle 1| = |0\rangle \otimes \langle 1|$, e) $|+\rangle\langle -| = |- \rangle\langle +|$, f) $(|0\rangle\langle 1|) \cdot |1\rangle = |0\rangle (\langle 1|1\rangle)$, g) $H \cdot (|0\rangle\langle 1|) = |+\rangle \cdot \langle 1|$.

3.5. Portas lógicas quânticas de 2 qubits

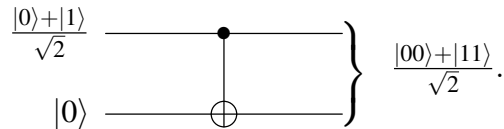
A porta de 2 qubits mais importante é a porta CNOT ou porta NOT controlada, também representada por $C(X)$ ou cX . Ela é definida pela expressão

$$\text{CNOT} |k\rangle |\ell\rangle = |k\rangle X^k |\ell\rangle,$$

e é representada pelo circuito



No circuito acima, $|k\rangle$ e $|\ell\rangle$ são estados da base computacional de 1 qubit. O estado do primeiro qubit não muda após a aplicação da porta CNOT. O estado do segundo qubit só muda se o valor de k for 1. Neste caso a saída é $X|\ell\rangle = |\ell \oplus 1\rangle$. Se $k = 0$ então $X^0 = I_2$ e $I_2|\ell\rangle = |\ell\rangle$, onde I_2 é a matriz identidade 2×2 . A definição que acabamos de mostrar descreve a atuação da porta CNOT na base computacional de 2 qubits. Na álgebra linear, esta definição está completa, pois para saber a atuação da porta CNOT em um vetor que é uma superposição de vetores da base computacional usamos a linearidade desta porta. Por exemplo, no circuito abaixo a entrada do primeiro qubit está em superposição:



Como calcular a saída? A melhor maneira de determinar a saída é usar um cálculo algébrico. Após usar a distributividade do produto de Kronecker, a entrada pode se escrita como

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle = \frac{1}{\sqrt{2}} |0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \otimes |0\rangle = \frac{|00\rangle + |10\rangle}{\sqrt{2}}.$$

Para calcular a atuação da porta CNOT em uma soma de vetores, usamos a linearidade do produto matricial (denotado por \cdot), isto é,

$$\text{CNOT} \cdot \left(\frac{|00\rangle + |10\rangle}{\sqrt{2}} \right) = \frac{1}{\sqrt{2}} \text{CNOT} \cdot |00\rangle + \frac{1}{\sqrt{2}} \text{CNOT} \cdot |10\rangle.$$

As notações $\text{CNOT} \cdot |00\rangle$ ou $\text{CNOT}|00\rangle$ denotam a multiplicação da matriz CNOT pelo vetor $|00\rangle$. A matriz CNOT é

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

Podemos confirmar que a saída é

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

Como este resultado é um estado emaranhado, não podemos fatorá-lo e portanto não podemos escrever uma saída para cada qubit. Após a medição, a saída é 00 com probabilidade 1/2 ou 11 com probabilidade 1/2.

Exercício 12. Mostre que $\text{CNOT}|k\rangle|\ell\rangle = |k\rangle|\ell \oplus k\rangle$ onde k e ℓ são bits.

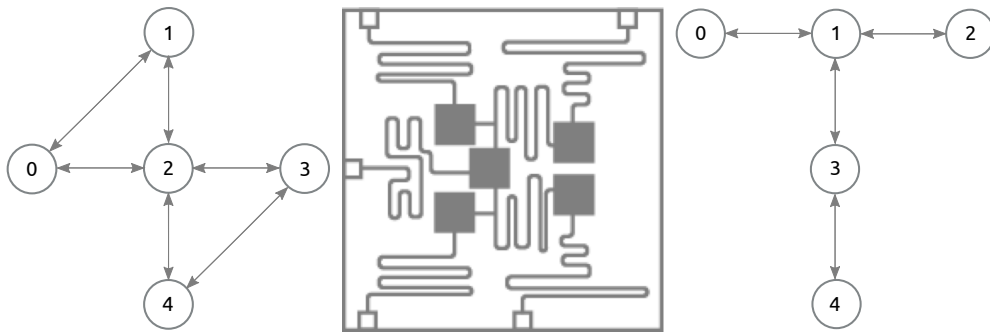


Figura 6. Mapa de conectividade do chip do IBM Q 5 Yorktown [ibmqx2] (esquerda), IBM Q Ourense [ibmq_ourense] (direita) e o desenho de chip físico (centro) com cerca de 5mm

Para gerar o circuito acima no *composer* da IBM, vamos usar o *backend ibmqx2*, cujo mapa de conectividade está mostrado no lado direito da figura 6. O usuário pode selecionar o *backend* no *composer* da IBM após salvar uma nova simulação e clicar na opção **Run**. O mapa descreve como os qubits interagem e deve ser usado no momento de fazer uma porta CNOT controlada por uma questão de eficiência. Se o usuário fizer uma porta CNOT com controle no qubit 0 e alvo no qubit 4, o compilador vai usar portas CNOTs auxiliares que aumentam o circuito compilado e deterioram o resultado.

Como os qubits estão inicialmente no estado $|0\rangle$, temos que usar uma porta H no primeiro qubit para gerar a superposição $(|0\rangle + |1\rangle)/2$. Portanto, devemos implementar a expressão

$$\text{CNOT} \cdot (H \otimes I)$$

como mostrado na figura 7. Note que no *composer*, o qubit $q[0]$ representa o qubit mais a direita, isto é, a ordem é $|q[4]q[3]q[2]q[1]q[0]\rangle$. Portanto, a aplicação de $(H \otimes I)$ em dois qubits quer dizer que H foi aplicada a $|q[1]\rangle$ e I foi aplicado a $|q[0]\rangle$ consistente com a figura 7.

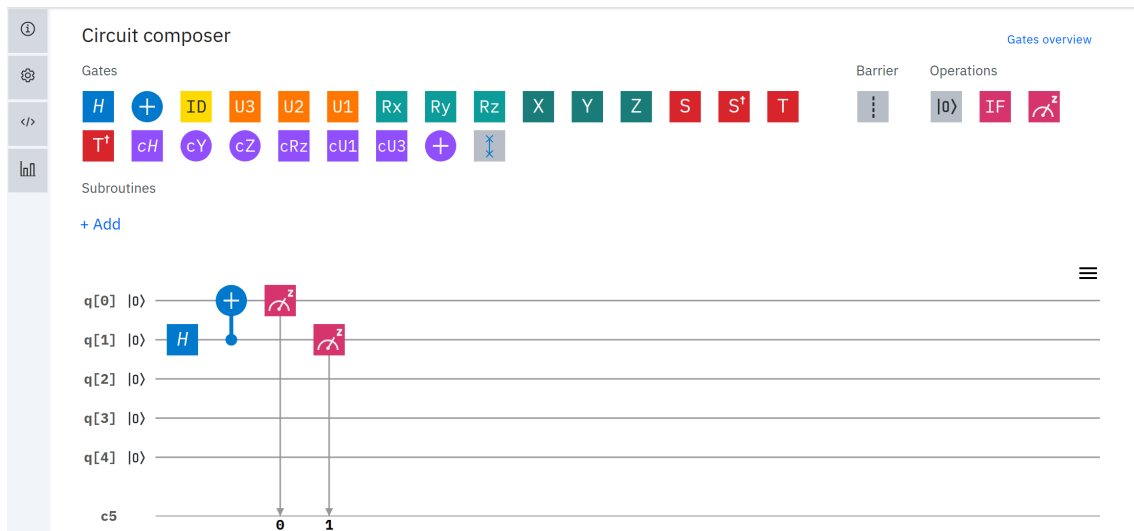


Figura 7. Circuito usando as portas H e CNOT (Reprint Courtesy of IBM Corporation ©)

O controle da porta CNOT é o qubit $q[1]$ e o alvo é o qubit $q[0]$. Para fazer CNOT arrastamos a porta \oplus de cor azul até a segunda coluna e depois editamos a porta clicando no lápis e trocamos a ordem dos qubits clicando nas conexões dos qubits (círculos da esquerda dentro do modo de edição da porta). No final do algoritmo, devemos colocar as portas de medição representadas pelo visor do voltímetro. Automaticamente é gerado um programa *Qasm* (Quantum assembly language) que pode ser visto clicando em \langle / \rangle no lado esquerdo da tela. Em diversas situações é melhor lidar com o programa *Qasm* em vez da interface gráfica, por exemplo, quando queremos replicar uma parte do circuito.

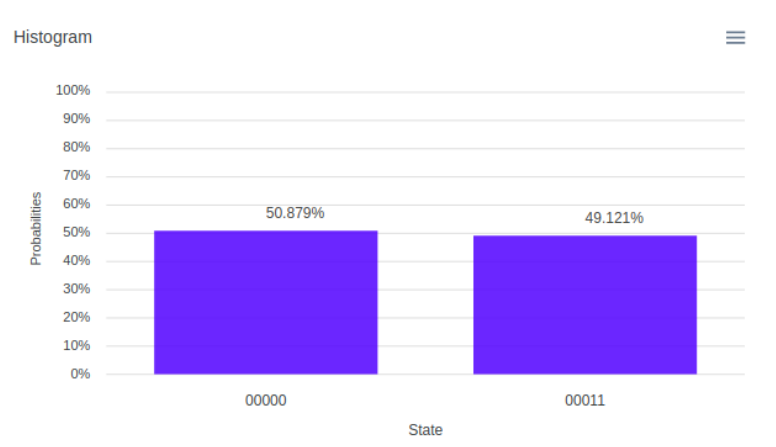
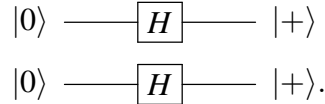


Figura 8. Saída do circuito usando as portas H e CNOT no *backend ibmqx2* (Reprint Courtesy of IBM Corporation ©)

A saída de uma simulação com 1024 repetições está mostrada na figura 8. Como vimos antes, o resultado é um histograma da distribuição de probabilidades que é gerado por 1024 repetições neste caso. A figura mostra que o resultado $q[4]q[3]q[2]q[1]q[0]=00011$ foi obtido com probabilidade 0.49121 (503 vezes em 1024) e o resultado $q[4]q[3]q[2]q[1]q[0]=00000$ foi obtido com probabilidade 0.50879 (521 ve-

zes em 1024). No nosso caso, devemos descartar os qubits q[2], q[3] e q[4]. Portanto, o resultado mostra que o algoritmo retornou 00 com probabilidade 0.51 e 11 com probabilidade 0.49 aproximadamente, que são resultados próximos de 1/2 como era esperado em função do cálculo analítico.

O próximo exemplo é mais simples do que o anterior. Considere o seguinte circuito sem a medição ao final:



Como calcular a saída? Usualmente, a forma algébrica é a mais simples. A forma algébrica do circuito acima é $(H \otimes H)|00\rangle$. O cálculo é executado da seguinte forma:

$$(H \otimes H)|00\rangle = (H \otimes H) \cdot (|0\rangle \otimes |0\rangle) = (H|0\rangle) \otimes (H|0\rangle) = |+\rangle \otimes |+\rangle.$$

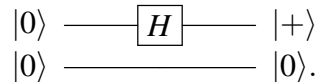
Na segunda igualdade, usamos a seguinte propriedade do produto de Kronecker:

$$(A \otimes B) \cdot (|\psi_1\rangle \otimes |\psi_2\rangle) = (A|\psi_1\rangle) \otimes (B|\psi_2\rangle),$$

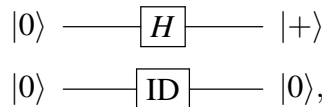
que é válida para quaisquer matrizes A e B e vetores $|\psi_1\rangle$ e $|\psi_2\rangle$ desde que as dimensões dos vetores sejam compatíveis com as dimensões correspondentes das matrizes. Se A é uma matriz $k \times k$ e B uma matriz $\ell \times \ell$ então $|\psi_1\rangle$ tem que ter k entradas e $|\psi_2\rangle$ tem que ter ℓ entradas. Portanto a saída do circuito é

$$|+\rangle \otimes |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{|0\rangle + |1\rangle + |2\rangle + |3\rangle}{2}.$$

Vamos ver mais um exemplo simples. Considere o seguinte circuito sem a medição ao final:



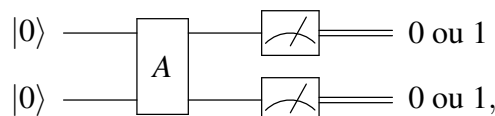
Como calcular a saída usando a expressão algébrica equivalente? A dica é usar o seguinte circuito equivalente:



onde ID é a matriz identidade de duas dimensões também representada como I_2 . O cálculo algébrico é feito da seguinte forma:

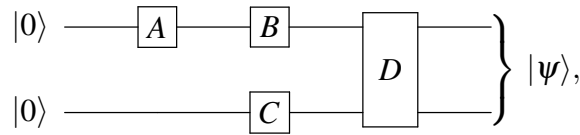
$$(H \otimes I_2)|00\rangle = (H|0\rangle) \otimes (I_2|0\rangle) = |+\rangle \otimes |0\rangle = \frac{|00\rangle + |10\rangle}{\sqrt{2}}.$$

O circuito de 2 qubits mais geral possível é



onde A é uma matriz unitária 4×4 . Relembrando, a matriz A é unitária se a ação de A em um vetor de norma 1 genérico resulta em um vetor de norma 1. Existem várias maneiras algébricas de verificar se A é uma matriz unitária: (1) As colunas de A tem que ser vetores de norma 1 ortogonais entre si, ou (2) a multiplicação de A pela sua transposta conjugada tem que dar a matriz identidade, isto é, $A \cdot A^\dagger = I$.

Quando convertemos um circuito quântico na sua expressão algébrica equivalente, devemos usar o produto de Kronecker para portas na mesma coluna e o produto matricial para portas na mesma linha ou em sequência, no entanto, devemos inverter a ordem das portas no segundo caso. Por exemplo, a expressão algébrica equivalente ao circuito



onde A , B e C são portas de 1 qubit e D é uma porta de 2 qubits, é

$$|\psi\rangle = D \cdot (B \otimes C) \cdot (A \otimes I_2) \cdot |00\rangle.$$

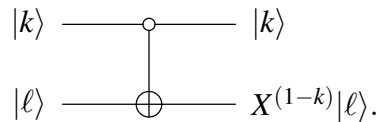
Podemos simplificar um pouco esta expressão e escrever

$$|\psi\rangle = D(BA \otimes C)|00\rangle.$$

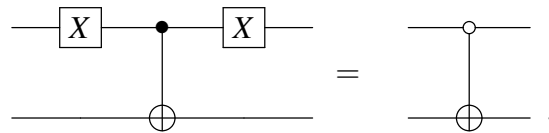
A porta CNOT é tão importante que vamos descrever uma variante chamada porta CNOT ativada pelo 0. Ela é definida pela expressão

$$|k\rangle|\ell\rangle \longrightarrow |k\rangle X^{(1-k)}|\ell\rangle,$$

e é representada pelo circuito



Note que o qubit de controle é denotado pelo círculo vazio indicando que o controle só é ativado se o qubit está no estado $|0\rangle$. Esta porta pode ser obtida a partir da porta CNOT usual por conjugação pela porta $(X \otimes I_2)$, como mostra a seguinte equivalência de circuitos:



Esta porta é representada por uma matriz em blocos da forma

$$\begin{bmatrix} X & \\ & I_2 \end{bmatrix}.$$

Exercício 13. Calcule $(X^{\otimes 3}) \cdot |010\rangle$ e compare com $(X \cdot |0\rangle) \otimes (X \cdot |1\rangle) \otimes (X \cdot |0\rangle)$ (pense num vetor como uma matriz 2×1) e expresse o resultado na base computacional usando a notação de Dirac (não use o formato matricial nas respostas finais). Por que as respostas são iguais? Qual cálculo é mais simples?

Exercício 14. Calcule $H^{\otimes 2}$. Calcule $(H^{\otimes 2}) \cdot |11\rangle$ e compare com $(H \cdot |1\rangle)^{\otimes 2}$. Por que as respostas são iguais?

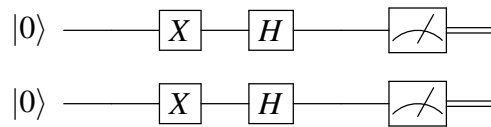
Exercício 15. Considere os seguintes estados de 2 qubits:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle), \quad |\psi_2\rangle = \frac{1}{\sqrt{3}}(|00\rangle + |01\rangle) + \frac{1}{\sqrt{6}}(|10\rangle + |11\rangle), \quad |\psi_3\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle).$$

PARTE 1: Mostre que $|\psi_1\rangle$ e $|\psi_2\rangle$ não são emaranhados e $|\psi_3\rangle$ é emaranhado.

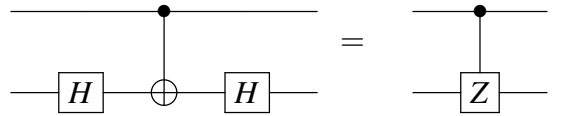
PARTE 2: Implemente no composer da IBM os circuitos que geram os estados $|\psi_1\rangle$, $|\psi_2\rangle$ e $|\psi_3\rangle$.

Exercício 16. Implemente o seguinte circuito no composer da IBM:



Use os resultados do exercício 14 para mostrar que a distribuição de probabilidades gerada pelo composer é igual ou próxima do esperado segundo os cálculos analíticos.

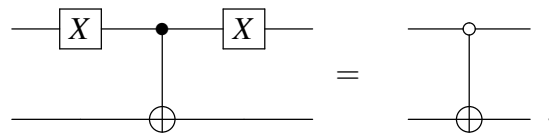
Exercício 17. Mostre a equivalência algébrica dos seguintes circuitos



Exercício 18. Sabendo que a porta NOT controlada pelo 0 é representada por uma matriz em blocos da forma

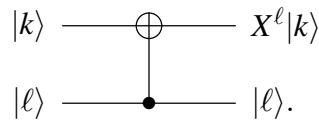
$$\begin{bmatrix} X & \\ & I_2 \end{bmatrix},$$

mostre a equivalência algébrica dos seguintes circuitos

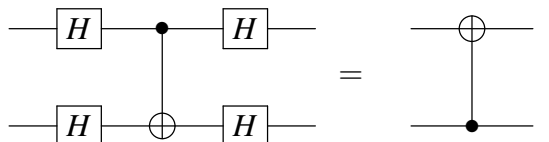


usando matrizes em blocos. [Dica: converta a expressão $(X \otimes I_2) \cdot CNOT \cdot (X \otimes I_2)$ para o formato matricial usando matrizes em blocos e faça a multiplicação.]

Exercício 19. Obtenha a representação matricial da porta NOT com os qubits invertidos, mostrada no seguinte circuito:



Exercício 20. Mostre a equivalência algébrica dos seguintes circuitos



Exercício 21. Seja U uma porta de 1 qubit. Mostre que a porta U controlada de 2 qubits é uma matriz diagonal em blocos dada por

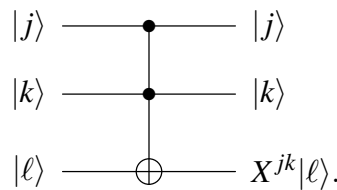
$$C(U) = \begin{bmatrix} I_2 & \\ & U \end{bmatrix}.$$

3.6. Portas lógicas quânticas de 3 ou mais qubits

A porta de 3 qubits mais importante é a porta Toffoli designada como CCNOT ou $C^2(X)$, definida pela expressão

$$C^2(X) |j\rangle |k\rangle |\ell\rangle = |j\rangle |k\rangle X^{jk} |\ell\rangle,$$

e representada pelo circuito

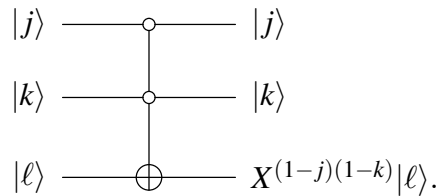


Esta porta tem 2 controles e 1 alvo. A porta X atua no alvo se, e somente se, os valores dos bits de controle forem ambos iguais a 1. Se um dos bits de controles for 0, o valor de terceiro qubit não se altera. Esta porta pode ser vista como a versão quântica da porta clássica AND, pois se $\ell = 0$, a saída do terceiro qubit é $(j \text{ AND } k)$. A saída do terceiro qubit também pode ser escrito como $(j \text{ AND } k) \oplus \ell$. A matriz da porta Toffoli é uma matriz diagonal em blocos dada por

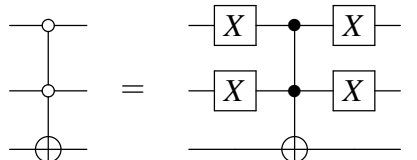
$$C^2(X) = \begin{bmatrix} I_2 & & & \\ & I_2 & & \\ & & I_2 & \\ & & & X \end{bmatrix}.$$

A porta Toffoli está disponível no *composer* da IBM; basta arrastar a porta \oplus de cor lilás para o circuito e escolher a posição (coluna e altura).

Existem variantes da porta Toffoli ativadas pelo 0. Por exemplo, o circuito



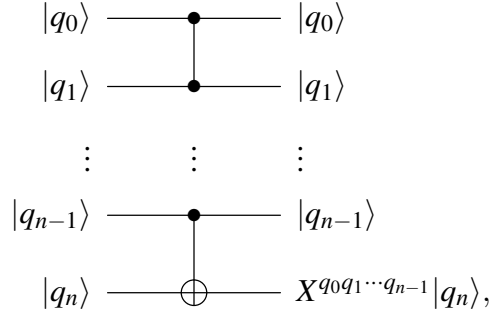
implementa uma porta que inverte o valor do terceiro qubit se, e somente se, os dois primeiros qubits estão no estado $|0\rangle$. Esta porta pode ser implementada no *composer* da IBM usando uma porta Toffoli usual e a porta X , como mostra a equivalência de circuitos a seguir:



A porta Toffoli generalizada $C^n(X)$ é uma porta de $(n + 1)$ qubits com n qubits de controle e um alvo. Ela é definida pela expressão

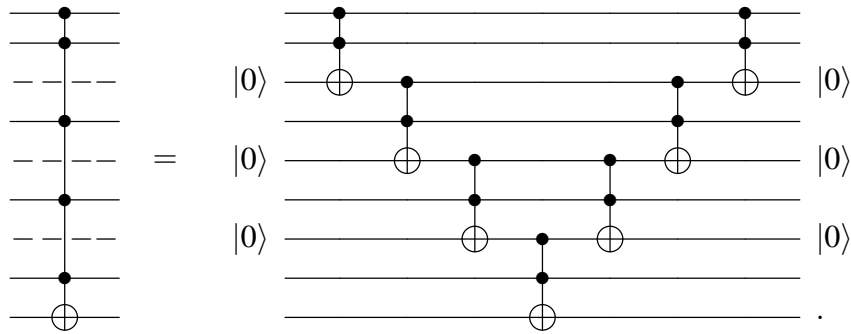
$$C^n(X)|q_0\rangle|q_1\rangle\cdots|q_{n-1}\rangle|q_n\rangle = |q_0\rangle|q_1\rangle\cdots|q_{n-1}\rangle X^{q_0q_1\cdots q_{n-1}}|q_n\rangle,$$

e é representada pelo circuito



onde $q_0q_1\cdots q_{n-1}$ é o produto dos bits q_0, q_1, \dots, q_{n-1} . Portanto, o estado do qubit alvo inverte se, e somente se, todos os qubits de controle estiverem no estado $|1\rangle$. O estado de cada qubit de controle não muda quando descrevemos esta porta atuando na base computacional. A atuação em um vetor genérico é obtida escrevendo o vetor na base computacional e usando linearidade.

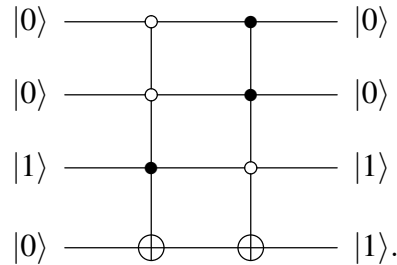
A maneira mais simples de decompor a porta Toffoli generalizada em termos de portas Toffoli usuais é usando $(n - 2)$ qubits de rascunho chamados *ancillas*. Os qubits ancillas são introduzidos de maneira intercalada com os qubits de controle, sendo que o primeiro qubit ancilla deve ser inserido entre o segundo e terceiro qubits. A melhor maneira de explicar a decomposição é mostrar um exemplo. Considere a porta $C^5(X)$, cuja decomposição requer 3 *ancillas*, como mostra a seguinte equivalência de circuitos:



A porta Toffoli generalizada também pode ser ativada pelo 0. Neste caso, o qubit de controle é mostrado com um círculo vazio e pode ser obtido através porta Toffoli generalizada ativada pelo 1 com os controles conjugados por portas X . Para n qubits, temos 2^n portas Toffoli generalizadas que podem implementar qualquer função booleana de n bits, como descrevemos no próximo parágrafo.

Agora vamos mostrar como obter o circuito quântico de uma tabela verdade. Para mostrar que as portas Toffoli generalizadas ativadas pelo 0 e 1 podem ser usadas para implementar uma função booleana em um computador quântico, vamos tomar a função booleana $f(a, b, c)$ descrita na seção 1 como exemplo. Fica evidente como se obtém o

caso geral. Como f tem 3 bits de entrada, vamos usar portas Toffoli generalizadas de 3 controles. A saída da função f é a saída de uma medição do qubit alvo. Como a função f tem 2 cláusulas, vamos usar 2 portas Toffoli generalizadas. A primeira porta deve ser ativada pela entrada $|001\rangle$ e a segunda pela entrada $|110\rangle$. O seguinte circuito implementa a função f :



Note que a entrada é do tipo $|a, b, c\rangle|0\rangle$ e a saída é $|a, b, c\rangle|f(a, b, c)\rangle$. Isto mostra que o computador quântico pode calcular qualquer função booleana de n bits usando uma porta Toffoli generalizada de $(n + 1)$ bits para cada saída 1 da tabela verdade associada. É claro que não faz sentido construir um máquina muito mais cara para executar algoritmos clássicos. No entanto, a implementação que acabamos de descrever pode ser usada em entradas superpostas, algo não permitido no computador clássico. Infelizmente, esta técnica de construção de circuito quânticos para cálculo de tabelas verdades não é eficiente, pois o número de portas Toffoli generalizadas é exponencial em função do número de qubits no pior caso.

Exercício 22. *Implemente um circuito no ibmqx2 da tabela verdade*

$q[0]$	$q[1]$	$q[2]$
$ 0\rangle$	$ 0\rangle$	$ 1\rangle$
$ 0\rangle$	$ 1\rangle$	$ 0\rangle$
$ 1\rangle$	$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$ 1\rangle$	$ 1\rangle$

Use 3 qubits onde a entrada do $q[2]$ está sempre no estado $|0\rangle$ e ignore os qubits $q[3]$ e $q[4]$. Usando portas X em $q[0]$ e $q[1]$, é possível gerar as 4 entradas da tabela verdade. Na saída coloque apenas um medidor no $q[2]$ e ignore as saídas de $q[0]$ e $q[1]$. [Dica: implemente a porta Toffoli com controles em $q[0]$ e $q[1]$ e com alvo em $q[2]$ no ibmqx2 e mostre que você obtêm a tabela verdade

$q[0]$	$q[1]$	$q[2]$
$ 0\rangle$	$ 0\rangle$	$ 0\rangle$
$ 0\rangle$	$ 1\rangle$	$ 0\rangle$
$ 1\rangle$	$ 0\rangle$	$ 0\rangle$
$ 1\rangle$	$ 1\rangle$	$ 1\rangle$

No editor qasm, implemente usando copy-paste uma segunda porta Toffoli ativada pelo zero. Estas duas portas em sequência implementam a tabela verdade desejada, segundo a lógica da decomposição de expressões booleanas na FND.]

Exercício 23. *Mostre que basta uma porta X para construir o circuito de uma tabela verdade de uma tautologia de n bits (todas as saídas são 1) quando a entrada é $|0\rangle^{\otimes(n+1)}$.*

Exercício 24. PARTE 1. Gere um estado $|GHZ\rangle$ de 3 qubits

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

tanto no `ibmqx2` como no `ibmq_ourense` da forma mais simples possível (dois qubits devem ser ignorados) seguindo o mapa de conectividade destes computadores. Após confirmar a corretude destes circuitos através simulações, compare a saída do simulador com as saídas dos computadores quânticos `ibmqx2` e `ibmq_ourense`. Quais são as fidelidades dos resultados? [Dica: use uma porta H e vários $CNOT$ s.]

PARTE 2. Gere um estado $|GHZ\rangle$ de 4 qubits

$$\frac{|0000\rangle + |1111\rangle}{\sqrt{2}}$$

tanto no `ibmqx2` como no `ibmq_ourense` da forma mais simples possível (um qubit deve ser ignorado). Compare a saída do simulador com as saídas dos computadores quânticos `ibmqx2` e `ibmq_ourense`. Quais são as fidelidades dos resultados?

PARTE 3. Mostre que o circuito da figura 9 gera um estado $|GHZ\rangle$ de 5 qubits

$$\frac{|00000\rangle + |11111\rangle}{\sqrt{2}}$$

que pode ser implementado no `ibmqx2`. Obtenha o circuito que gera o mesmo estado no `ibmq_ourense`. Compare a saída do simulador com as saídas dos computadores quânticos `ibmqx2` e `ibmq_ourense`. Quais são as fidelidades dos resultados?

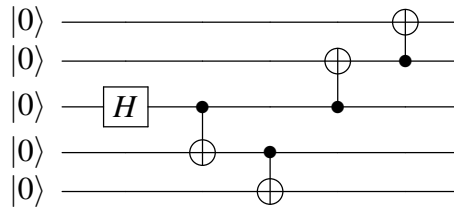


Figura 9. Circuito para gerar um estado $|GHZ\rangle$ de 5 qubits seguindo o mapa de conexão do `ibmqx2`

Exercício 25. (Muito difícil) O estado $|W\rangle$ de n qubits é definido como

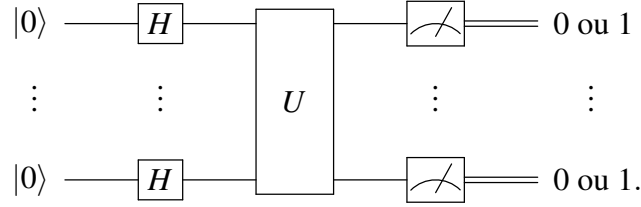
$$|W\rangle = \frac{1}{\sqrt{n}}(|100\dots 0\rangle + |010\dots 0\rangle + |001\dots 0\rangle + \dots + |000\dots 1\rangle).$$

Faça um circuito que gera o estado $|W\rangle$ de 3 qubits

$$|W\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle).$$

4. Paralelismo quântico e o modelo padrão da computação quântica

O modelo padrão da computação quântica é descrito pelo circuito



O estado inicial de cada qubit é $|0\rangle$. Depois, a porta Hadamard H é aplicada a cada qubit, preparando para o *paralelismo quântico*. Na sequência, a matriz unitária U é aplicada em todos os qubits. Finalmente é realizada a medição de todos os qubits retornando uma cadeia de bits.

O paralelismo quântico é a execução simultânea de um algoritmo com mais do que uma entrada. Para entender este conceito, vamos focar apenas na matriz U sem considerar as portas H . A matriz U e as medições do modelo padrão podem ser interpretados como um algoritmo randomizado no sentido clássico. Se x é uma cadeia de n bits e U é uma matriz $2^n \times 2^n$, então $U|x\rangle$ é uma superposição de 2^n vetores da base computacional. Após a medição, uma cadeia y de n bits é retornada. Isto é, se a entrada de U é x , a saída após a medição é y . O “algoritmo” U foi executado para uma única entrada x . Agora vamos colocar as portas H de volta.

O primeiro passo do modelo padrão é executar o seguinte cálculo: $(H|0\rangle) \otimes \dots \otimes (H|0\rangle)$, isto é

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \otimes \dots \otimes \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right).$$

Após expandir este produto, obtemos

$$\frac{1}{\sqrt{2^n}} (|0\dots 0\rangle + |0\dots 01\rangle + |0\dots 10\rangle + \dots + |1\dots 1\rangle).$$

Todos as possíveis cadeias de n bits aparecem na soma acima. Ou seja, todas as entradas possíveis de um algoritmo clássico com n bits estão representadas na soma acima. O próximo passo do modelo padrão é aplicar a matriz U . Pela linearidade da álgebra linear, a matriz U deve ser aplicada a cada termo da soma acima simultaneamente. Portanto, é possível realizar 2^n cálculos simultâneos no computador quântico de n bits. Note porém que os resultados destes cálculos estão misturados em uma nova soma e, após a medição, o resultado é uma única cadeia de n bits.

5. Algoritmo de Bernstein-Vazirani

Como exemplo de algoritmo quântico mais eficiente do que algoritmos clássicos equivalentes, vamos descrever o algoritmo de Bernstein-Vazirani [7, 27, 33], que resolve o seguinte problema computacional: Seja $s = s_0\dots s_{n-1}$ uma cadeia de n bits desconhecida. Apesar de não conhecermos s , temos à nossa disposição uma função Booleana

$f : \{0, 1\}^n \longrightarrow \{0, 1\}$ definida por

$$f(x) = x \cdot s = x_0 s_0 + \dots + x_{n-1} s_{n-1} \mod 2,$$

onde x_0, \dots, x_{n-1} são os bits de x . Nosso objetivo é encontrar s chamando a função $f(x)$ o menor número de vezes para qualquer x no domínio da função. Em computação quântica, esta função é chamada *oráculo*, pois é como se um oráculo revelasse o valor de $f(x)$ sem revelar s , e $f(x)$ pode ser usado no algoritmo para determinar s . Quando construímos o circuito do algoritmo, a parte relacionada a f é implementada por uma outra pessoa (oráculo), pois desconhecemos s . É importante entender isto, caso contrário, ficamos com a impressão que precisamos saber a resposta para encontrar a resposta, o que é um absurdo.

Na versão clássica deste problema, temos que consultar o *oráculo clássico* n vezes. De fato, escolhemos $x = 10\dots 0$ e perguntamos ao oráculo qual é o valor de $f(10\dots 0)$. A resposta é s_0 . A seguir escolhemos $x = 010\dots 0$ e perguntamos ao oráculo qual é o valor de $f(010\dots 0)$. A resposta é s_1 . A última consulta é $f(0\dots 01)$, cuja resposta é s_{n-1} . Isto mostra que precisamos consultar o oráculo n vezes e não tem como reduzir este número sem introduzir um erro na resposta. No caso quântico, vamos consultar o *oráculo quântico* uma única vez, o que permite encontrar todos os bits de s , como descrito a seguir.

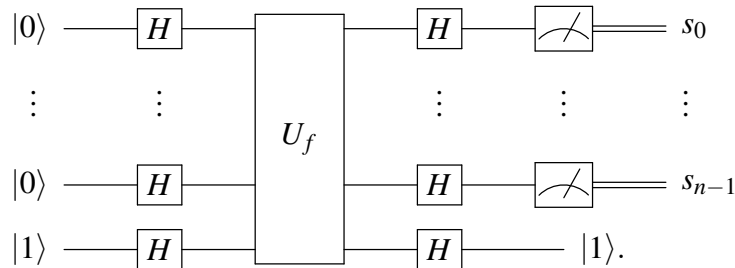
No caso quântico, a função $f(x) = x \cdot s$ é implementada pelo oráculo através de um operador unitário U_f de $n + 1$ qubits, que usualmente é definido como

$$U_f |x\rangle |j\rangle = |x\rangle |j \oplus f(x)\rangle,$$

onde \oplus é a operação XOR ou soma módulo 2. Este operador usa dois registradores⁷. O primeiro registrador tem n qubits e o segundo registrador tem 1 qubit. O operador U_f fica a nossa disposição e podemos usá-los tantas vezes quanto desejarmos. No entanto, ele é usado uma única vez no algoritmo de Bernstein-Vazirani.

Exercício 26. (a) Exiba explicitamente a matriz U_f para o caso $n = 2$ e $s = 01_{(2)}$ e mostre que o resultado é igual a $CNOT_{12} = I_2 \otimes CNOT$. (b) Mostre que U_f é uma matriz unitária no caso geral. [DICA: Mostre que $\langle x' | \langle j' | U_f^\dagger U_f | x \rangle | j \rangle = \delta_{xx'} \delta_{jj'}$.] (c) Mostre que U_f é uma matriz de permutação⁸ no caso geral.

O algoritmo de Bernstein-Vazirani está descrito no **Algoritmo 1** e o circuito é



⁷Um registrador quântico é um conjunto de qubits.

⁸Uma matriz de permutação é uma matriz quadrada unitária cujas entradas são 0 ou 1 tal que cada linha da matriz tenha exatamente uma entrada igual a 1.

Algoritmo 1: Algoritmo de Bernstein-Vazirani

Input: uma função Booleana $f : \{0, 1\}^n \rightarrow \{0, 1\}$ tal que $f(x) = x \cdot s$.

Output: com probabilidade igual a 1, retorna s .

- 1 Prepare o estado inicial $|0\rangle^{\otimes n}|1\rangle$;
 - 2 Aplique $H^{\otimes(n+1)}$;
 - 3 Aplique U_f ;
 - 4 Aplique $H^{\otimes(n+1)}$;
 - 5 Faça uma medição do primeiro registrador na base computacional.
-

5.1. Análise do algoritmo de Bernstein-Vazirani

Após o primeiro passo, o estado do computador quântico é

$$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle.$$

Após o segundo passo, o estado do computador quântico é

$$\begin{aligned} |\psi_1\rangle &= (H^{\otimes n}|0\rangle^{\otimes n}) \otimes (H|1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |-\rangle, \end{aligned}$$

onde $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$ e x está escrito na notação decimal. Após o terceiro passo, o estado do computador quântico é

$$\begin{aligned} |\psi_2\rangle &= U_f|\psi_1\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} U_f(|x\rangle \otimes |-\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{x \cdot s} |x\rangle \otimes |-\rangle. \end{aligned}$$

Exercício 27. Mostre que

$$U_f(|x\rangle \otimes |-\rangle) = (-1)^{x \cdot s} |x\rangle \otimes |-\rangle$$

para qualquer $x \in \{0, 1\}^n$. [DICA: Use a definição de $|-\rangle$ e faça o cálculo de $U_f(|x\rangle \otimes |-\rangle)$ assumindo que $x \cdot s = 0$ e simplifique o resultado. Depois faça o cálculo de $U_f(|x\rangle \otimes |-\rangle)$ assumindo que $x \cdot s = 1$ e simplifique o resultado. Depois junte estes resultados usando a notação $(-1)^{x \cdot s}$.]

Após o quarto passo, o estado do computador quântico é

$$\begin{aligned} |\psi_3\rangle &= H^{\otimes(n+1)}|\psi_2\rangle \\ &= H^{\otimes n} \left(\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{x \cdot s} |x\rangle \right) \otimes (H|-\rangle) \\ &= |s\rangle \otimes |1\rangle. \end{aligned}$$

Exercício 28. Mostre que

$$H^{\otimes n}|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{x \cdot s} |x\rangle$$

para qualquer $s \in \{0,1\}^n$. [DICA: Escreva s na notação binária $s_0 \dots s_{n-1}$ e mostre que $H^{\otimes n}|s\rangle = (H|s_0\rangle) \cdots (H|s_{n-1}\rangle)$. Em seguida mostre que $H|s_0\rangle = (1/\sqrt{2}) \sum_{x_0=0}^1 (-1)^{x_0 s_0} |x_0\rangle$ usando o fato que s_0 assume o valor 0 ou 1. Repita para s_1 e assim por diante e depois simplifique.] Usando $H^2 = I$, mostre que

$$|s\rangle = H^{\otimes n} \left(\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{x \cdot s} |x\rangle \right).$$

Após o quinto passo, os resultados das medições serão os bits s_0, \dots, s_{n-1} pois $|s\rangle = |s_0\rangle \otimes \cdots \otimes |s_{n-1}\rangle$.

5.2. Circuito do oráculo U_f de Bernstein-Vazirani

Como afirmamos anteriormente, quem implementa U_f é uma terceira pessoa, porém é conveniente saber como é feito. Vamos fazer um exemplo com $n = 4$ e $s = 1011$ que será suficiente para entender o caso geral. Neste caso particular, a função f é

$$f(x) = x \cdot s = x_0 + x_2 + x_3 \pmod{2}$$

e a ação do operador U_f em um termo genérico da base computacional $|x\rangle|j\rangle$ é

$$U_f|x_0x_1x_2x_3\rangle|j\rangle = |x_0x_1x_2x_3\rangle|j \oplus f(x)\rangle = |x_0x_1x_2x_3\rangle|j \oplus x_0 \oplus x_2 \oplus x_3\rangle.$$

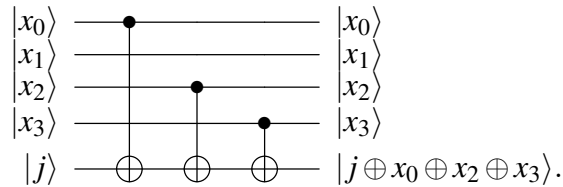
Pela definição da porta CNOT atuando nos qubits 0 e 4 (sem mostrar os qubits 1, 2 e 3), temos

$$\text{CNOT}_{04}|x_0\rangle|j\rangle = |x_0\rangle X^{x_0}|j\rangle = |x_0\rangle|j \oplus x_0\rangle.$$

Podemos ver que se usarmos CNOT_{04} , CNOT_{24} e CNOT_{34} , podemos gerar o resultado esperado para a porta U_f , isto é

$$U_f = \text{CNOT}_{34} \cdot \text{CNOT}_{24} \cdot \text{CNOT}_{04},$$

cujo circuito é

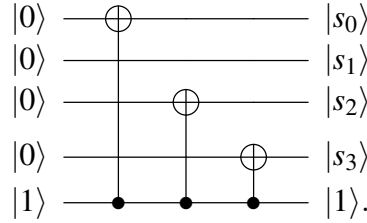


A partir deste exemplo, podemos generalizar para $s = s_0 \cdots s_{n-1}$ e verificar que

$$U_f = \text{CNOT}_{s'_0 n} \cdots \text{CNOT}_{s'_{m-1} n},$$

onde m é o peso de Hamming de s e s'_0, \dots, s'_{m-1} são os qubits correspondentes aos bits de s iguais a 1. Note que a ordem das portas CNOTs é irrelevante.

Exercício 29. (a) Faça o desenho do circuito completo do algoritmo de Bernstein-Vazirani usando apenas as portas H e $CNOT$ para $n = 4$ e $s = 1011$ exibindo a entrada e a saída do circuito. (b) Use o exercício 20 para mostrar que a versão simplificada do algoritmo completo sem a medição é



(c) Resolva a seguinte aparente contradição. O circuito simplificado do algoritmo de Bernstein-Vazirani não tem superposição nem emaranhamento. Portanto, um programador clássico também poderia determinar s consultando um oráculo clássico uma única vez.

5.3. Implementação do algoritmo de Bernstein-Vazirani no *composer* da IBM

Para implementar o algoritmo de Bernstein-Vazirani no *composer* da IBM, vamos usar o exemplo $n = 4$ e $s = 1011$ descrito na seção anterior.

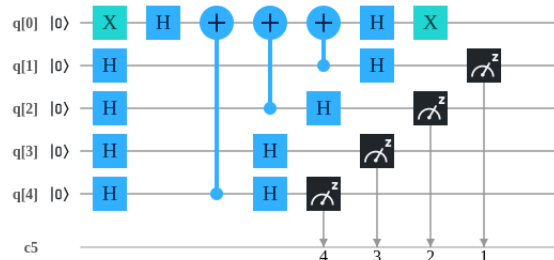


Figura 10. Circuito do algoritmo de Bernstein-Vazirani no *composer* da IBM para $n = 4$ e $s = 1011$. Note que é necessário inverter verticalmente o circuito

A figura 10 mostra o circuito do algoritmo de Bernstein-Vazirani para $n = 4$ e $s = 1011$. Note que é necessário inverter verticalmente o circuito, pois a ordem dos qubits nos computadores da IBM segue a notação $q[4]q[3]q[2]q[1]q[0]$, enquanto que neste texto e na maioria das referências usam a ordem $q[0]q[1]q[2]q[3]q[4]$. Se o usuário não inverter o circuito, a saída estará com os bits invertidos. A saída do *ibmq_vigo* após 1024 repetições está mostrada na figura 11. O programa *Qasm* deste circuito (sem as medições) está no rodapé⁹. Para replicar este experimento, abra um experimento novo, mude para o editor *Qasm*, copie a nota de rodapé e cole a partir da quarta linha. É necessário também adicionar quatro medidores nos qubits $q[1]$, $q[2]$, $q[3]$ e $q[4]$. Podemos ver que o resultado com a maior probabilidade foi $c[4]c[3]c[2]c[1] = 1011$ mostrando que s foi encontrado corretamente usando o oráculo U_f uma única vez. No entanto, quando rodamos este circuito no simulador obtemos a saída $c[4]c[3]c[2]c[1] = 1011$ com 100%

⁹x q[0]; h q[1]; h q[2]; h q[3]; h q[4]; h q[0]; cx q[4],q[0]; cx q[2],q[0]; h q[3]; h q[4]; cx q[1],q[0]; h q[2]; h q[0]; h q[1]; x q[0];

de probabilidade, que é o resultado correto. O resultado mostrado na figura 11 indicando que $s = 1011$ com probabilidade 65.430% está degradado devido aos erros inerentes do computador quântico da IBM.

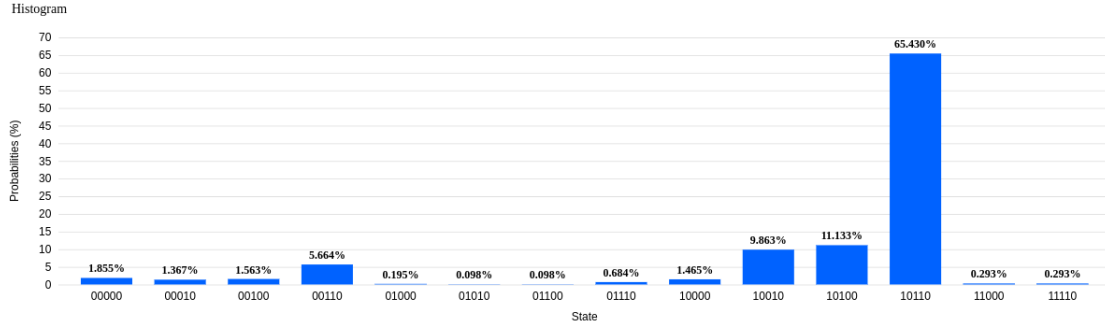


Figura 11. Histograma da distribuição de probabilidades do circuito da figura 10 gerado pela *ibmq_vigo*

6. Algoritmo de Grover

O algoritmo de Grover [17] é um algoritmo de busca em bancos de dados não-ordenados. Do ponto de vista prático é mais interessante ordenar o banco de dados e depois realizar a busca. Em função disto, vamos descrever nesta seção uma outra formulação do algoritmo de Grover que mostra de forma mais clara a sua importância [30]. Outras referências que apresentam o algoritmo de Grover são [3, 4, 6, 13, 18, 19, 20, 21, 24, 26, 27, 28, 37, 42].

6.1. Formulação do problema

Seja N uma potência de 2, isto é, $N = 2^n$ para algum número inteiro n . Suponha que $f : \{0, \dots, N-1\} \rightarrow \{0, 1\}$ é uma função booleana tal que $f(x) = 1$ se, e somente se, $x = x_0$ para algum valor fixo x_0 . Assim,

$$f(x) = \begin{cases} 1, & \text{se } x = x_0, \\ 0, & \text{caso contrário.} \end{cases}$$

Suponha que x_0 não é conhecido. Como podemos achar x_0 usando f ? Do ponto de vista computacional, queremos desenvolver um algoritmo que avalie f o número mínimo de vezes.

Classicamente, o algoritmo mais eficiente consulta f um número de vezes proporcional a N , isto é, a complexidade de consultas é $O(N)$. Como é feito na prática? Temos que pedir a uma pessoa a qual confiamos para gerar um número aleatório x_0 de n bits. Esta pessoa esconde x_0 e faz uma subrotina compilada da função f . Podemos usar a subrotina quantas vezes desejarmos, porém não podemos examinar a subrotina e tentar descobrir x_0 *hackeando* o código. O algoritmo clássico que resolve este problema é um iteração que consulta $f(j)$ para j indo de 0 a $2^n - 1$. Assim que $f(j)$ retorna o valor 1, o programa é interrompido e x_0 é encontrado.

Quanticamente, é possível melhorar a complexidade de consultas para $O(\sqrt{N})$. Como é feito na prática? Temos que pedir de novo a uma pessoa de confiança para gerar

um número aleatório x_0 de n bits. Esta pessoa, que usualmente é chamada de *oráculo*, implementa a função f através de uma matriz unitária F_{x_0} . Nós podemos usar F_{x_0} no computador quântico, porém não podemos ver os detalhes da implementação de F_{x_0} . Cada vez que usarmos F_{x_0} , adicionamos uma unidade na contagem. Podemos usar portas adicionais que obviamente não dependem de x_0 .

Temos um mesmo problema que pode ser resolvido por algoritmos que são executados de duas formas diferentes. No primeiro caso, um computador clássico de $O(n)$ bits é usado e a solução é obtida após $O(N)$ passos. No segundo caso, um computador quântico de $O(n)$ qubits é usado e a solução é obtida após $O(\sqrt{N})$ passos. Este ganho de complexidade justifica o investimento em um hardware quântico e no estudo de técnicas para desenvolver algoritmos quânticos, que necessariamente têm que explorar a superposição de estados. No caso do algoritmo de Grover, a matriz F_{x_0} atuando na superposição de estados avalia a função f em mais de uma entrada ao mesmo tempo com os mesmos recursos disponíveis. Esta tarefa só pode ser executada no computador clássico através de um número exponencial de processadores.

6.2. Como implementar uma função em um computador quântico?

O primeiro passo para desenvolver um algoritmo quântico que resolve o problema acima é a implementação da função f . Como f é uma função booleana cuja tabela verdade tem uma única linha com saída 1, f pode ser implementada com uma porta Toffoli generalizada ativada pelo x_0 , como descrito no final da subseção 3.6. Esta porta tem uma matriz unitária associada que denominamos F_{x_0} , que é definida pela sua atuação na base computacional como

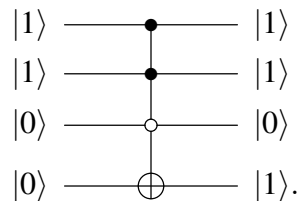
$$F_{x_0}|x\rangle|i\rangle = |x\rangle|i \oplus f(x)\rangle,$$

onde x é uma cadeia de n bits e i é um bit. O conjunto dos n primeiros qubits é chamado de primeiro registrador e o último qubit de segundo registrador. Note que se tomarmos $i = 0$, a equação acima se reduz à

$$F_{x_0}|x\rangle|0\rangle = \begin{cases} |x_0\rangle|1\rangle, & \text{se } x = x_0, \\ |x\rangle|0\rangle, & \text{caso contrário,} \end{cases}$$

que descreve o que a porta Toffoli generalizada faz quando ativada por x_0 (e somente por x_0). O resultado do cálculo de $f(x)$ é armazenado no segundo registrador enquanto que o valor do primeiro registrador fica inalterado.

Por exemplo, o circuito que implementa F_{x_0} no caso $N = 8$ e $x_0 = 6$, isto é, $f(110) = 1$ e $f(j) = 0$ se $j \neq 110$, é



Os três primeiros qubits formam o primeiro registrador e o quarto qubit é o segundo registrador. Note que o estado do segundo registrador só muda de $|0\rangle$ para $|1\rangle$ se a entrada

do primeiro registrador for $|110\rangle$, pois a cadeia de bits 110 ativa os 3 controles e qualquer outra cadeia de 3 bits não ativa.

No algoritmo de Grover, o segundo registrador está sempre no estado

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

Usando a linearidade, a atuação da matriz F_{x_0} é dada por

$$F_{x_0}|x\rangle|-\rangle = \begin{cases} -|x_0\rangle|-\rangle, & \text{se } x = x_0, \\ |x\rangle|-\rangle, & \text{caso contrário.} \end{cases}$$

Neste caso, a atuação da matriz F_{x_0} não altera o estado do segundo registrador.

6.3. Descrição do algoritmo de Grover

O algoritmo de Grover usa uma matriz adicional definida como

$$G = (2|d\rangle\langle d| - I_N) \otimes I_2,$$

onde

$$|d\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle.$$

A notação “ $|d\rangle\langle d|$ ” é chamada de *produto externo* entre o vetor $|d\rangle$ (matriz $N \times 1$) e o vetor dual $\langle d|$ (matriz $1 \times N$). O produto externo é equivalente ao produto matricial. A multiplicação de uma matriz $N \times 1$ por uma matriz $1 \times N$ resulta em uma matriz $N \times N$. Portanto, $|d\rangle\langle d|$ é uma matriz $N \times N$, dada por

$$|d\rangle\langle d| = \frac{1}{N} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

e a matriz G é o produto tensorial da matriz

$$(2|d\rangle\langle d| - I_N) = \frac{1}{N} \begin{bmatrix} (2-N) & 2 & \cdots & 2 \\ 2 & (2-N) & \cdots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ 2 & 2 & \cdots & (2-N) \end{bmatrix}$$

por I_2 . A matriz $(2|d\rangle\langle d| - I_N)$ é chamada matriz de Grover.

O algoritmo de Grover está descrito no **Algoritmo 2**.

6.4. Circuito (não-econômico) do algoritmo de Grover

Para obter um circuito do algoritmo de Grover, temos que fazer uma manipulação algébrica com a expressão da matriz de Grover $(2|d\rangle\langle d| - I_N)$. Note que

$$|d\rangle = H^{\otimes n}|0\rangle$$

Algoritmo 2: Algoritmo de Grover

Input: um inteiro N e uma função $f : \{0, \dots, N-1\} \rightarrow \{0, 1\}$ tal que $f(x) = 1$ somente para um ponto $x = x_0$ no domínio.

Output: com probabilidade igual ou maior que $1 - \frac{1}{N}$, retorna x_0 .

- 1 Prepare o estado inicial $|d\rangle|-\rangle$;
 - 2 Aplique $(GF_{x_0})^t$, onde $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$;
 - 3 Faça uma medição do primeiro registrador na base computacional.
-

onde $|0\rangle$ está na notação decimal e $H^{\otimes n} = H \otimes \dots \otimes H$. Transpondo a equação acima, obtemos

$$\langle d| = \langle 0|H^{\otimes n}.$$

Podemos verificar que $(H^{\otimes n}) \cdot (H^{\otimes n}) = (H \cdot H)^{\otimes n} = (I_2)^{\otimes n} = I_N$. Usando estes resultados, obtemos

$$(2|d\rangle\langle d| - I_N) = H^{\otimes n}(2|0\rangle\langle 0| - I_N)H^{\otimes n},$$

onde

$$(2|0\rangle\langle 0| - I_N) = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{bmatrix}.$$

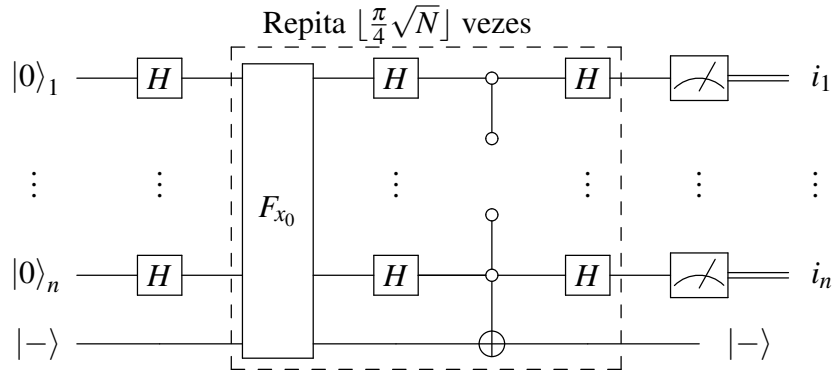
A matriz $(2|0\rangle\langle 0| - I_N)$ atua apenas no primeiro registrador. No entanto, é mais simples implementar esta matriz usando ambos os registradores. Vamos mostrar que ela é implementada por uma porta Toffoli generalizada ativada por n bits zero. De fato, a atuação da matriz $(2|0\rangle\langle 0| - I_N)$ em um estado $|x\rangle$ da base computacional do primeiro registrador, onde x é uma cadeia de n bits, é

$$(2|0\rangle\langle 0| - I_N)|x\rangle = \begin{cases} |0\rangle, & \text{se } x = 0, \\ -|x\rangle, & \text{caso contrário.} \end{cases}$$

Portanto, a atuação da matriz $(2|0\rangle\langle 0| - I_N)$ (no primeiro registrador) é igual à atuação da matriz $(-F_{x_0})$ (em ambos registradores) descrita na subseção 6.2 quando $x_0 = 0$ e quando o segundo registrador está no estado $|-\rangle$. O sinal de menos $-$ em $(-F_{x_0})$ não altera nem o resultado do algoritmo nem a probabilidade final. Ou seja, usar o operador G ou o operador $-G$ no algoritmo de Grover não altera o resultado.

Usando estes resultados algébricos discutidos acima, concluímos que um circuito

para implementar o algoritmo de Grover é

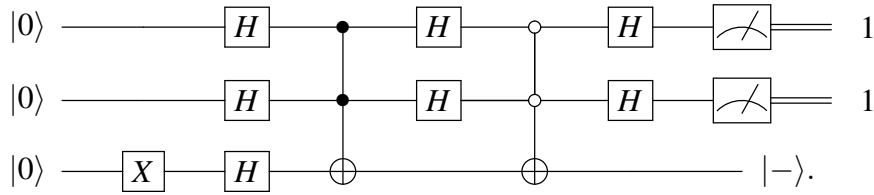


onde a saída i_1, \dots, i_n são os bits de x_0 , isto é $x_0 = (i_1 \dots i_n)_2$.

A construção deste circuito nesta seção está seguindo a descrição padrão do algoritmo de Grover, porém este circuito pode ser reduzido, como veremos adiante na descrição econômica.

6.5. Implementação do algoritmo de Grover no composer da IBM

Para implementar o algoritmo de Grover no composer da IBM, vamos usar o caso $N = 4$ e $x_0 = 11$ como exemplo. Particularizando o circuito para este caso, obtemos



Note que a caixa pontilhada na descrição geral do algoritmo não precisa de repetições, pois $\lfloor \pi\sqrt{N}/4 \rfloor = 1$ para $N = 4$. A saída esperada é $i_1 i_2 = 11$.

A figura 12 mostra o circuito do algoritmo de Grover para $N = 4$ e $x_0 = 3$. A saída do *ibmq_ourense* após 1024 repetições está mostrada na figura 13. O programa *Qasm* deste circuito (sem as medições) está no rodapé¹⁰. Para replicar este experimento, abra um experimento novo, mude para o editor *Qasm*, copie a nota de rodapé e cole a partir da quarta linha. É necessário também adicionar dois medidores nos qubits $q[1]$ e $q[2]$. Podemos ver que o resultado com a maior probabilidade foi $c[2]c[1] = 11$ mostrando que o valor de x_0 foi encontrado corretamente usando a matriz F_{x_0} uma única vez. No entanto, quando rodamos este circuito no simulador obtemos a saída $c[2]c[1] = 11$ com 100% de probabilidade. Este é de fato o resultado correto. O resultado mostrado na figura 13 indicando que $x_0 = 11$ com probabilidade 50.195% está degradado devido aos erros inerentes do computador quântico da IBM.

Para mudar o valor de x_0 devemos conjugar o controles da primeira porta Toffoli ($q[2]$ e $q[1]$) por X , isto é, colocar X no lugar das matrizes ID de maneira simétrica. Para

¹⁰ x q[0]; h q[1]; h q[2]; h q[0]; id q[1]; id q[2]; ccx q[2],q[1],q[0]; id q[1]; id q[2]; h q[1]; h q[2]; x q[1]; x q[2]; ccx q[2],q[1],q[0]; x q[1]; x q[2]; h q[1]; h q[2];

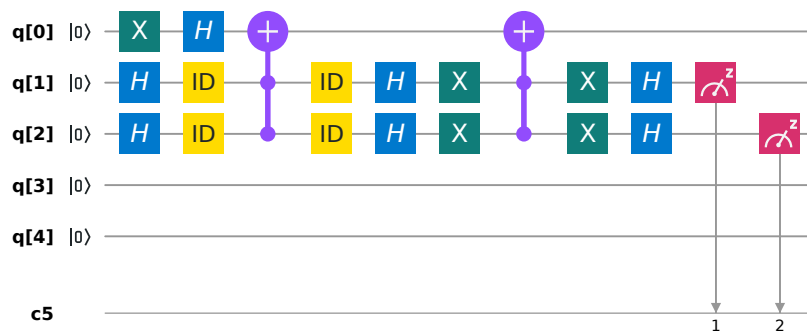


Figura 12. Circuito do algoritmo de Grover com $N = 4$ e $x_0 = 3$ (Reprint Courtesy of IBM Corporation ©)

Result

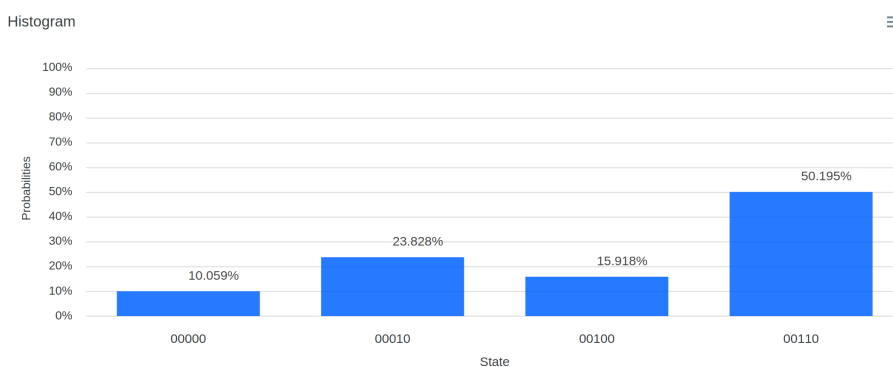
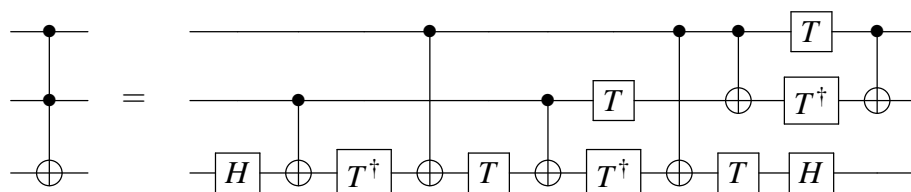


Figura 13. Saída q[2]q[1] do algoritmo de Grover com $N = 4$ e $x_0 = 3$ no *ibmq_ourense* (Reprint Courtesy of IBM Corporation ©)

$x_0 = 00$, devemos aplicar X tanto em $q[2]$ como em $q[1]$. Para $x_0 = 01$, devemos aplicar X em $q[1]$. Para $x_0 = 10$, devemos aplicar X em $q[2]$.

É importante saber que o circuito será compilado (*transpiled*) antes de ser enviado para o computador quântico. A implementação da porta Toffoli depende da sua decomposição em CNOTs e portas de 1 qubit (H , T), como descrito pela equivalência dos seguintes circuitos:



Existem outras formas de decompor a porta Toffoli e o compilador vai usar a decomposição mais conveniente.

É possível melhorar a fidelidade do resultado mostrado na figura 13 com o resultado obtido pelo simulador usando uma implementação mais econômica do algoritmo de Grover, como discutido adiante.

6.6. Análise do algoritmo de Grover

Por que o algoritmo de Grover funciona corretamente? Vamos responder esta pergunta usando uma interpretação geométrica de reflexões de vetores. O objetivo é achar x_0 , que é uma cadeia de n bits. Na computação quântica, o objetivo é colocar o estado do computador no estado $|x_0\rangle$, pois as medições de cada qubit retornam neste caso os bits de x_0 .

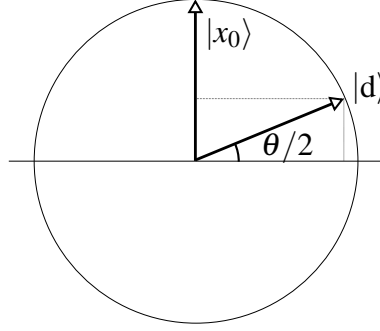


Figura 14. Descrição dos vetores $|x_0\rangle$ e $|d\rangle$

No início do algoritmo, o estado do computador é $|d\rangle$. Para N grande, $|d\rangle$ é quase ortogonal a $|x_0\rangle$. Uma medição neste ponto tem poucas chances de retornar x_0 . A figura 14 mostra a posição destes vetores e o ângulo $\theta/2$ que $|d\rangle$ faz com o eixo horizontal. Qualquer outra posição dos vetores desde que $|d\rangle$ seja quase ortogonal a $|x_0\rangle$ serve na análise. O ângulo θ é muito pequeno para N grande, e neste caso, $\theta/2$ é uma aproximação muito boa de $\sin(\theta/2)$. Além disso, o seno de um ângulo é igual ao cosseno do complemento, isto é,

$$\frac{\theta}{2} \approx \sin \frac{\theta}{2} = \cos \left(\frac{\pi}{2} - \frac{\theta}{2} \right).$$

Como $(\pi - \theta)/2$ é o ângulo entre $|x_0\rangle$ e $|d\rangle$, pela definição do produto interno, $\cos(\pi - \theta)/2$ é o produto interno de $|x_0\rangle$ e $|d\rangle$, cujo resultado é

$$\frac{\theta}{2} \simeq \sin \frac{\theta}{2} = \cos \left(\frac{\pi}{2} - \frac{\theta}{2} \right) = \langle x_0 | d \rangle = \frac{1}{\sqrt{N}}.$$

Portanto,

$$\theta \approx \frac{2}{\sqrt{N}}.$$

Vamos desconsiderar o estado do segundo registrador, pois ele permanece fixo em $|-\rangle$ durante todo o algoritmo. Por isso, vamos definir a matriz

$$f_{x_0}|x\rangle = \begin{cases} -|x_0\rangle, & \text{se } x = x_0, \\ |x\rangle, & \text{caso contrário,} \end{cases}$$

que é uma versão reduzida de F_{x_0} . Vamos usar f_{x_0} no lugar de F_{x_0} na análise do algoritmo. O primeiro passo é aplicar f_{x_0} em $|d\rangle$. A ação de f_{x_0} em $|d\rangle$ (escrito na base computacional) inverte o sinal da amplitude de $|x_0\rangle$ e não altera as outras amplitudes. A amplitude

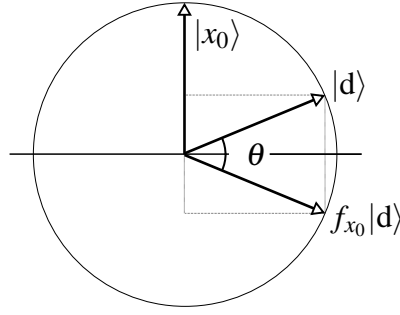


Figura 15. O vetor $f_{x_0}|d\rangle$ é a reflexão de $|d\rangle$ em relação ao eixo horizontal

de $|x_0\rangle$ é a linha pontilhada vertical da figura 14, que é invertida pela ação de f_{x_0} . Geometricamente, a ação de f_{x_0} é representada por uma reflexão de $|d\rangle$ em torno do eixo horizontal. O ângulo entre os vetores $|d\rangle$ e $(f_{x_0}|d\rangle)$ é θ , conforme mostra a figura 15.

O próximo passo é aplicar $(2|d\rangle\langle d| - I_N)$. Vamos chamar esta matriz de g , pois ela é uma versão reduzida com N dimensões de G , ou seja,

$$g = 2|d\rangle\langle d| - I_N.$$

Agora vamos mostrar que a ação da matriz g é uma reflexão em torno do eixo que passa pela origem na direção $|d\rangle$. A prova disto é feita em duas etapas. Primeiro, mostramos que a ação de g em $|d\rangle$ não muda a direção de $|d\rangle$. Segundo, mostramos que a ação de g em $|d^\perp\rangle$ inverte o sinal de $|d^\perp\rangle$, onde $|d^\perp\rangle$ é um vetor ortogonal a $|d\rangle$. A primeira etapa é consequência da conta

$$g|d\rangle = (2|d\rangle\langle d| - I_N)|d\rangle = 2|d\rangle\langle d|d\rangle - |d\rangle = |d\rangle,$$

pois $\langle d|d\rangle = 1$. A segunda etapa é consequência da conta

$$g|d^\perp\rangle = (2|d\rangle\langle d| - I_N)|d^\perp\rangle = 2|d\rangle\langle d|d^\perp\rangle - |d^\perp\rangle = -|d^\perp\rangle,$$

pois $\langle d|d^\perp\rangle = 0$.

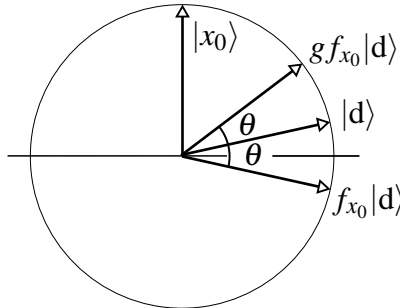


Figura 16. O vetor $(g f_{x_0}|d\rangle)$ é a reflexão de $(f_{x_0}|d\rangle)$ em torno de $|d\rangle$

A figura 16 mostra onde está o vetor $(g \cdot f_{x_0} \cdot |d\rangle)$. Isto mostra que a ação de $(G \cdot F_{x_0})$ rodou o estado inicial de θ graus em direção ao vetor $|x_0\rangle$. Como θ é um ângulo

pequeno, este resultado é modesto, porém promissor. É fácil verificar que a segunda ação de $(G \cdot F_{x_0})$ também roda de novo o estado do computador quântico de θ graus em direção ao vetor $|x_0\rangle$. Queremos saber quantas iterações r são necessárias tal que $r\theta = \pi/2$. O número de iterações é

$$r = \left\lfloor \frac{\pi}{2\theta} \right\rfloor = \left\lfloor \frac{\pi}{4} \sqrt{N} \right\rfloor.$$

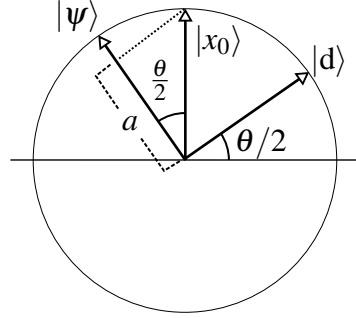


Figura 17. O vetor $|\psi\rangle$ é o estado final do computador quântico e a é a projeção de $|x_0\rangle$ no estado final. O ângulo entre $|\psi\rangle$ e $|x_0\rangle$ é menor ou igual a $\theta/2$

O final da análise é o cálculo da probabilidade de sucesso. Após r iterações, o estado do computador quântico sem considerar o segundo registrador é

$$|\psi\rangle = (g f_{x_0})^{\left\lfloor \frac{\pi}{4} \sqrt{N} \right\rfloor} |d\rangle.$$

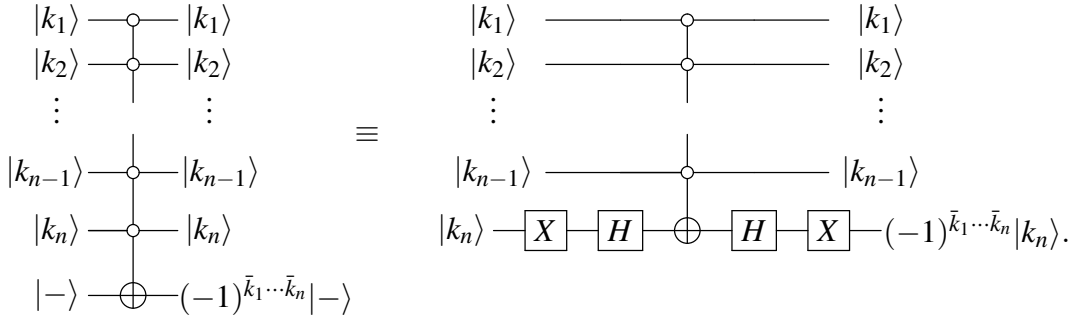
O vetor $|\psi\rangle$ é quase ortogonal a $|d\rangle$, como descrito na figura 17. O ângulo entre $|\psi\rangle$ e $|x_0\rangle$ é menor ou igual a $\theta/2$. A probabilidade de sucesso é maior ou igual do que o módulo ao quadrado da amplitude de $|x_0\rangle$ na decomposição de $|\psi\rangle$ na base computacional. Esta amplitude está mostrada na figura 17 através da letra a . A projeção de $|x_0\rangle$ no estado final é no máximo $\cos(\theta/2)$. Portanto, a probabilidade de sucesso $p = |a|^2$ satisfaz

$$p \geq \cos^2 \frac{\theta}{2} \geq 1 - \sin^2 \frac{\theta}{2} \geq 1 - \frac{1}{N}.$$

O caso $N = 4$ é especial, pois como $\sin(\theta/2) = 1/\sqrt{N}$, segue que $\theta = 60^\circ$. Com uma aplicação de $(g f_{x_0})$, o vetor $|d\rangle$ gira de 60° e coincide com o vetor $|x_0\rangle$. Neste caso, a probabilidade de sucesso é exatamente $p = 1$. Isto mostra que o resultado do *ibmqx2* na figura 12 está muito degradado enquanto que o resultado do *ibmq_ourense* é um pouco melhor.

6.7. Implementação do algoritmo de Grover usando um circuito econômico

O segundo registrador do algoritmo de Grover pode ser descartado e é possível fazer uma implementação mais econômica fugindo do modelo padrão descritos nos livros e seguindo publicações relacionadas com o Qiskit [25]. Primeiro vamos mostrar a equivalência dos seguintes circuitos:



Note que os circuitos não têm o mesmo número de qubits. Portanto, a equivalência só pode ser mostrada se eliminarmos o segundo registrador (último qubit) do primeiro circuito e antes disto movermos o sinal $(-1)^{\bar{k}_1 \dots \bar{k}_n}$ para o primeiro registrador. Isto é permitido pela seguinte propriedade do produto de Kronecker:

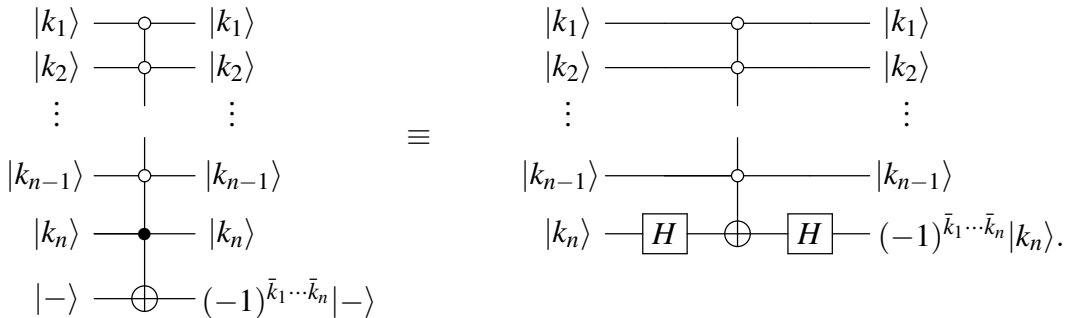
$$|v_1\rangle \otimes (a|v_2\rangle) = (a|v_1\rangle) \otimes |v_2\rangle = a(|v_1\rangle \otimes |v_2\rangle),$$

que é válida para quaisquer vetores $|v_1\rangle, |v_2\rangle$ e qualquer constante a . Portanto, a saída do primeiro circuito após a eliminação do segundo registrador é

$$(-1)^{\bar{k}_1 \dots \bar{k}_n} |k_1\rangle \otimes \dots \otimes |k_n\rangle.$$

No segundo circuito, se $|k_n\rangle = |0\rangle$, a primeira porta X gera o estado $|1\rangle$ e a primeira porta H cria (temporariamente) o estado $|-\rangle$, que inverte o sinal sob ação da porta Toffoli generalizada se os qubits de 0 a $n-1$ forem ativados. Na continuação, a segunda porta H desfaz o estado $|-\rangle$ convertendo para $|1\rangle$ e após a segunda porta X a saída é $(-|0\rangle)$ ou $|0\rangle$ dependendo se os $(n-1)$ primeiros qubits foram ativados ou não. Se $|k_n\rangle = 1$, a primeira porta X gera o estado $|0\rangle$ e a primeira porta H cria o estado $|+\rangle$, que não se altera sob a ação da porta Toffoli generalizada. Na continuação, a segunda porta H converte $|+\rangle$ em $|0\rangle$ e a saída é $|1\rangle$ independentemente se a porta Toffoli generalizada foi ativada ou não. O resultado é o mesmo em comparação com o resultado do primeiro circuito (após a eliminação do segundo registrador). Uma vez que vamos fazer uma medição do primeiro registrado apenas, isto completa a prova de que podemos substituir *sem nenhuma perda* o primeiro circuito pelo segundo no algoritmo de Grover. Esta substituição não é válida em outros algoritmos.

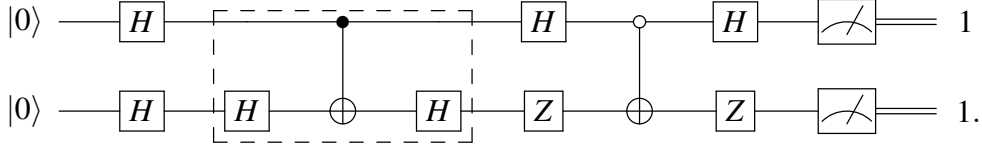
Se o n -ésimo qubit for ativado pelo 1, a equivalência dos circuitos fica descrita da seguinte maneira:



A verificação da equivalência é similar ao caso anterior. Portanto, a matriz F_{x_0} também pode ser reduzida de $(n+1)$ para n qubits introduzindo duas portas H de forma conjugada

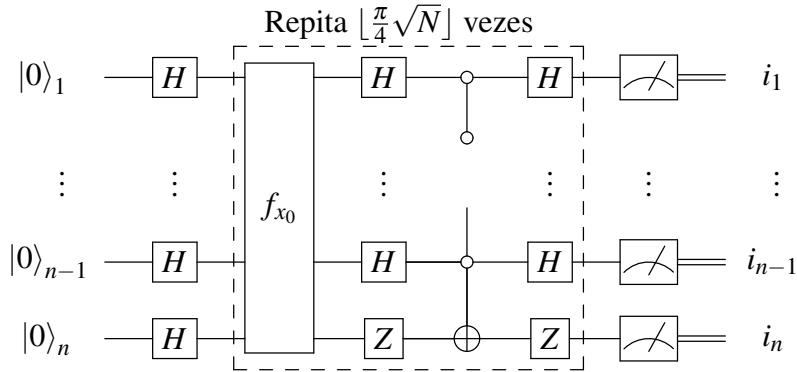
se o n -ésimo qubit for ativado pelo 1 e introduzindo duas portas H e duas portas X de forma conjugada se o n -ésimo qubit for ativado pelo 0. Os $(n-1)$ primeiros qubits podem ser ativados por 0 ou 1 e nada se altera para eles.

O circuito do algoritmo de Grover na forma econômica quando $N = 4$ e $x_0 = 11$ é

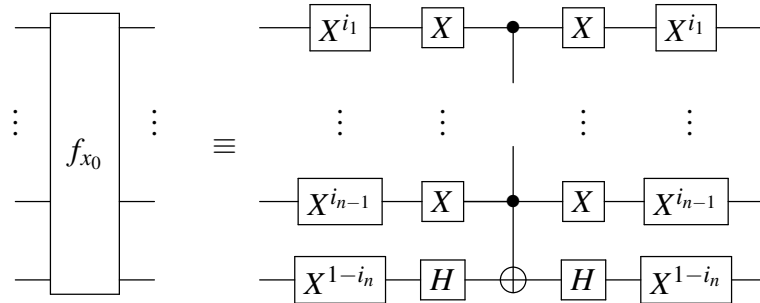


Para simplificar o circuito, substituímos as sequências de portas HXH no segundo qubit do circuito por Z . A parte do circuito dentro da caixa tracejada é escolhida pelo oráculo. O objetivo do algoritmo de Grover é desvendar o valor de x_0 consultando o oráculo, ou seja, usando a caixa tracejada, sem ver os detalhes da implementação. Temos que fingir que a caixa tracejada é uma caixa preta. No caso $N = 4$, existem 4 possíveis caixas tracejadas. Exibimos o caso $x_0 = 11$. É um bom exercício exibir os circuitos que o oráculo usa quando $x_0 = 00$, $x_0 = 01$ e $x_0 = 10$.

Para N genérico, o circuito do algoritmo de Grover na forma econômica com n qubits é



onde a matriz f_{x_0} é a forma reduzida de F_{x_0} e o circuito para f_{x_0} para $x_0 = (i_1 \dots i_n)_2$ genérico é



Na próxima seção, vamos mostrar como implementar o algoritmo de Grover de forma econômica usando o Qiskit.


Exercício 30. PARTE 1. Implemente o algoritmo de Grover na forma econômica no Qiskit quando $N = 4$, $x_0 = 00$, $x_0 = 01$ e $x_0 = 10$. **PARTE 2.** Implemente o algoritmo

de Grover na forma econômica no Qiskit quando $N = 8$ e $x_0 = 111$. PARTE 3. (Muito difícil) Implemente o algoritmo de Grover na forma econômica no Qiskit quando $N = 16$ e $x_0 = 1111$ e obtenha uma fidelidade de pelo menos 10% com o resultado correto.

7. Programando os computadores quânticos da IBM

Nesta seção, vamos mostrar como programar os computadores quânticos IBM. Já mostramos exemplos nas seções anteriores de uso dos computadores *ibmq_ourense* de Tenerife e *ibmqx2* de Yorktown, ambos de 5 qubits, usando o *composer* da página da IBM Q Experience¹¹. O *composer* da IBM é muito simples, pois podemos pegar as portas lógicas disponibilizadas e arrastar para o circuito. Porém, o *composer* não é útil para programas grandes. Neste caso, podemos usar a linguagem Qasm (*Quantum Assembly Language*) [11] cujos comandos podem ser editados por meio de um editor de textos usual. Além disto, as outras máquinas disponibilizadas pela IBM, como IBM Q 14 e 20, não possuem o *composer* e exigem a utilização um kit de desenvolvimento de software, chamado Qiskit¹², que é baseado na linguagem Python. Nas próximas seções, vamos descrever como usar e fazer programas através do Qiskit. Os comandos do Qiskit são baseados no modelo de circuitos e são muito semelhantes à linguagem Qasm. Em particular, vamos mostrar como como implementar de forma eficiente o algoritmo de Grover para 2 qubits, isto é, $N = 4$.

7.1. Qasm

Qasm é uma linguagem de baixo nível — do tipo *assembly* — desenvolvida pela IBM para representar os circuitos executáveis no IBM Q Experience. Programas em Qasm podem ser escritos de diversas formas. Sempre que um circuito é criado por meio do *composer* um código Qasm é gerado automaticamente. O código gerado pode ser visualizado e editado diretamente no browser, clicando-se em . Na figura 18, temos o mesmo circuito já apresentado na figura 4, porém dessa vez visualizando o código Qasm correspondente.

Toda linha em Qasm que inicie com duas barras (*//*) são ignoradas e, portanto, servem como comentário. A primeira linha diferente de comentário em qualquer código Qasm deve necessariamente ser o comando `OPENQASM` seguido pela versão do Qasm que está sendo utilizada. Note que todos os comandos na linguagem Qasm devem encerrar com ponto-e-vírgula.

O comando `include`, como o próprio nome já diz, permite incluir código de um arquivo externo. Normalmente incluímos pelos menos o arquivo `qelib1.inc`, pois este contém muitas definições úteis. Depois do `include`, outros comandos que são praticamente obrigatórios nos códigos Qasm são `qreg` e `creg`, utilizados respectivamente para declarar um registrador quântico e um clássico. Portanto, o comando `qreg q[5]` serve para declarar que nosso programa utiliza um registrador quântico de 5 qubits, e este registrador será referenciado no código por meio da variável `q`. Analogamente, o comando `creg c[5]` declara um registrador clássico, importante para armazenar o resultado da

¹¹<https://quantum-computing.ibm.com/>

¹²<https://www.doi.org/10.5281/zenodo.2562110>

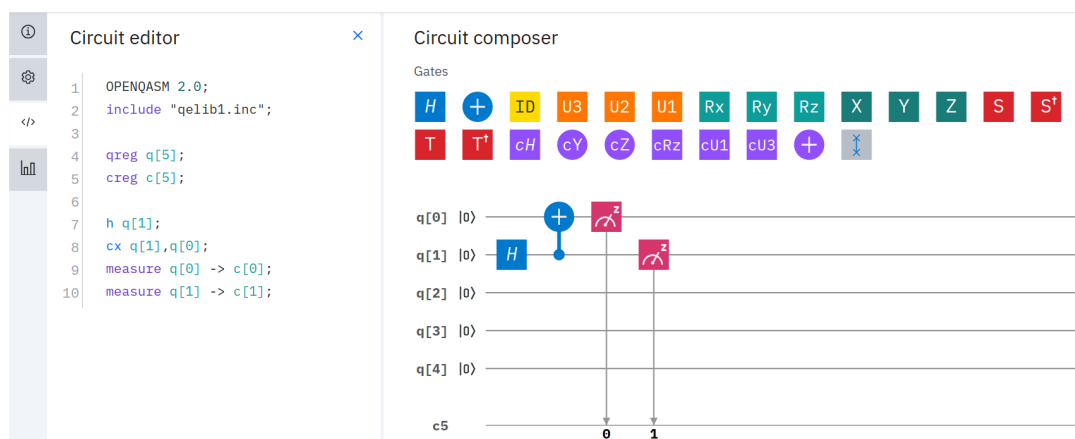


Figura 18. Circuito usando as portas H e CNOT, exibindo código-fonte Qasm equivalente ao circuito (Reprint Courtesy of IBM Corporation ©)

medição no final do algoritmo.

Incluir uma porta lógica por meio da linguagem Qasm é bastante intuitivo. Basta digitar o nome da porta e em seguida o qubit sobre o qual ela deve atuar. Por exemplo, o comando `h q[0]` significa que a porta de Hadamard deve ser aplicada no primeiro qubit ($q[0]$). Note que os qubits são numerados a partir de zero! Se o arquivo `qelib1.inc` foi incluído no início, então as principais portas — em particular, todas as portas disponíveis no *composer* — podem ser referenciadas diretamente: `h` é a porta de Hadamard (H), `x` é uma das portas de Pauli (X), e assim sucessivamente. Para incluir a porta T^\dagger utiliza-se o comando `tdg`, e para incluir a porta S^\dagger utiliza-se o comando `sdg`, onde `dg` se refere a *dagger*. No caso da porta CNOT, utiliza-se o comando `cx`, e nesse caso deve-se indicar tanto o qubit de controle como o qubit alvo — por exemplo, `cx q[0], q[1]` representa uma porta CNOT controlada pelo primeiro qubit e com alvo no segundo qubit.

Para efetuar uma medição, utiliza-se o comando `measure` seguido do registrador quântico a ser medido e do registrador clássico onde o resultado deve ser armazenado. Por exemplo, comando `measure q[0] -> c[0]` representa uma medição do primeiro qubit, com o resultado a ser armazenado no primeiro bit clássico.

Em vez de utilizar o *composer*, pode-se também editar um programa Qasm diretamente em outro editor de textos, e depois copiar o código-fonte para a página do IBM Q Experience na área correspondente para código Qasm. O circuito correspondente é mostrado, e então pode-se passar a editá-lo do modo visual, arrastando as portas quânticas. O método de editar o programa é bastante conveniente ao escrever circuitos grandes.

Finalmente, o código Qasm pode também ser gerado implicitamente usando linguagens de alto nível. A partir da próxima seção, veremos como o Qiskit pode ser utilizado para escrever programas quânticos usando linguagem Python.

7.2. Qiskit

Qiskit é um kit de desenvolvimento de software de código aberto, desenvolvido pela IBM visando facilitar a programação dos computadores quânticos já existentes. Ele pode ser obtido gratuitamente no site oficial do projeto (<https://qiskit.org>) e

pode ser usado em Linux, MacOS e Windows. Para instalar o Qiskit, é necessário em primeiro lugar ter o Python instalado na versão 3.5 ou mais recente. O Python normalmente já vem instalado no Linux e no MacOS. Usuários de Windows podem obter gratuitamente o Python no site oficial (<https://www.python.org>), ou por meio de outras distribuições voltadas para o ambiente científico. A melhor escolha é o Anaconda Python, que pode ser obtido gratuitamente em <https://www.anaconda.com/distribution>. A maneira mais fácil de instalar o Qiskit é primeiro instalar o Anaconda Python, que inclui o Jupyter Notebook, e depois instalar Qiskit.

Para instalar o Anaconda Python no Ubuntu, sugerimos seguir as instruções descritas no *site* <https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-20-04>, a princípio ignorando a seção “Setting up Anaconda Environments”. O Jupyter Notebook já vem instalado junto com o Anaconda Python e pode ser disparado com o comando `jupyter notebook`. A seguir, o comando `pip install qiskit` deve ser dado para instalar o Qiskit, e caso não haja erros, a instalação está completa. Vamos dar mais detalhes nos próximos parágrafos para quem tenha tido problemas na instalação.

A forma mais simples de instalar o Qiskit é usando o instalador de pacotes do Python, o pip (<https://pip.pypa.io>). O pip costuma vir instalado nas distribuições mais recentes do Python. Nesse caso, basta usar o comando `pip install qiskit`. Alguns pacotes adicionais de visualização e de algoritmos quânticos avançados só são instalados com o comando mais completo, `pip install qiskit[visualization] qiskit-aqua`. Às vezes, quando há duas versões diferentes do Python instaladas no mesmo computador, é necessário usar o comando `pip3` em vez do comando `pip`. Nesse caso, o comando seria `pip3 install qiskit`.

É recomendável ter instalado também o Jupyter Notebook, em especial durante o aprendizado. O Jupyter Notebook permite programar em Python diretamente no browser, permite a inclusão de textos explicativos com formatação e facilita a visualização dos resultados. Para instalar o Jupyter pode-se também usar o comando `pip install jupyter`. Para abrir o Jupyter, pode-se ir ao terminal de comando e digitar o comando `jupyter notebook`. Com isso, o Jupyter é aberto no browser padrão. Pode-se navegar pela estrutura de pastas do computador e abrir arquivos já existentes com a extensão `ipynb` (chamados de *notebooks*). Pode-se também criar novos notebooks clicando no botão *New* e em seguida escolhendo Python 3. Note que o Jupyter tem que ter o kernel Python 3, que pode ser instalado no Linux através de dois comandos consecutivos: 1) `python3 -m pip install ipykernel` e 2) `python3 -m ipykernel install --user`. Para mais detalhes, pode-se consultar o site oficial, <https://jupyter.org>.

A proposta dos desenvolvedores do Qiskit é permitir que qualquer pessoa com conhecimentos básicos de computação quântica e da linguagem de programação Python possa programar os computadores quânticos da IBM. Para executar os programas escritos com Qiskit diretamente nos computadores da IBM é necessário utilizar a *API Token* que é obtida no site do IBM Q Experience, clicando em *My account* e depois em *Advanced*. Além de permitir a execução diretamente em computadores quânticos reais, o Qiskit faci-

lita também a execução de experimentos em simuladores clássicos utilizando recursos de computação de alto desempenho.

O Qiskit é composto por quatro elementos — Terra, Aer, Ignis e Aqua —, dos quais utilizaremos os dois primeiros. O elemento Terra contém todos os fundamentos utilizados para escrever programas quânticos segundo o modelo de circuitos, e o elemento Aer possui recursos para simulação por meio de computação (clássica) de alto desempenho. Os elementos Ignis e Aqua contém recursos mais avançados e fogem do escopo deste curso.

7.2.1. Escrevendo um programa quântico básico

Para começar a utilizar o Qiskit Terra, vamos abrir a interface do Jupyter Notebook, como exemplificado na Figura 19. Devemos incluir as seguintes linhas no início do código para indicar ao Python quais comandos iremos utilizar:

```
from qiskit import QuantumCircuit
from qiskit import ClassicalRegister
from qiskit import QuantumRegister
```

Exemplos de *notebooks* relacionados com este tutorial podem ser baixados da conta *programaquantica*¹³ do GitHub.

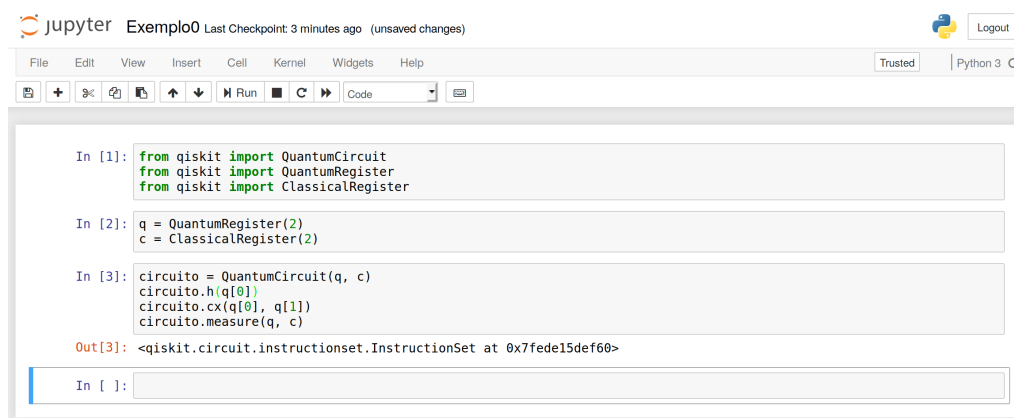


Figura 19. Exemplo de utilização do Jupyter Notebook

Os nomes dos módulos importados são bastante sugestivos. Evidentemente, para escrever um programa quântico no modelo de circuitos nosso primeiro passo deve ser definir um circuito! E para definir um circuito quântico, precisamos antes dizer quantos bits quânticos e clássicos ele deve ter. No final, claro, desejamos executar nosso circuito. Vejamos então a sequência de comandos típica para um programa básico no Qiskit. Digamos, por exemplo, que nosso circuito deva ter 2 qubits, como na figura 18. Nessa caso, devemos incluir a seguinte linha:

```
q = QuantumRegister(2)
```

¹³<https://github.com/programaquantica>

Agora, se também vamos precisar de 2 bits clássicos para ler o resultado final do circuito, então devemos incluir a seguinte linha:

```
c = ClassicalRegister(2)
```

Nos exemplos acima, os nomes `q` e `c` são apenas variáveis, e podem ser definidos livremente pelo usuário. Opcionalmente, é possível definir também um apelido para os registradores, para serem utilizados nas visualizações do Qiskit. Por exemplo:

```
q = QuantumRegister(2, 'qubit')
c = ClassicalRegister(2, 'bit')
```

Uma vez que tenhamos nossos registradores, já podemos definir nosso circuito quântico. O comando para isso é o seguinte:

```
circuito = QuantumCircuit(q, c)
```

No exemplo acima, o nome `circuito` é apenas uma variável, e pode ser definido livremente pelo usuário.

Nosso circuito quântico ainda está vazio. Precisamos incluir portas, medições etc. Para incluir uma porta quântica de Hadamard que atue no primeiro qubit, por exemplo, usamos o seguinte comando:

```
circuito.h(q[0])      # Hadamard
```

Outras portas de 1 qubits também estão disponíveis por meio de comandos semelhantes. Em particular, todas as portas elementares disponíveis no *composer* estão também disponíveis no Qiskit, com nomes autoexplicativos. Por exemplo, em vez da porta de Hadamard poderíamos aplicar uma das portas abaixo:

```
circuito.id(q[0])      # identidade
circuito.x(q[0])        # Pauli-X
circuito.y(q[0])        # Pauli-Y
circuito.z(q[0])        # Pauli-Z
circuito.s(q[0])        # S
circuito.sdg(q[0])      # transposto conjugado de S
circuito.t(q[0])        # T
circuito.tdg(q[0])      # transposto conjugado de T
```

Não podemos escrever algoritmos quânticos muito interessantes somente com portas atuando em 1 qubit. Precisamos pelo menos de uma porta que atue em 2 qubits, a saber, a porta CNOT. Uma porta CNOT controlada pelo primeiro qubit e com alvo no segundo qubit pode ser incluída no circuito com o seguinte comando do Qiskit:

```
circuito.cx(q[0], q[1])
```

Ao terminarmos de incluir as portas quânticas em nosso circuito, precisamos ainda efetuar uma medição. Foi por esse motivo que ao definirmos o circuito quântico tivemos de estabelecer não somente um registrador quântico mas também um registrador clássico. Agora, podemos indicar que desejamos efetuar uma medição em todos os três qubits do

registrador quântico e armazenar os resultados nos três bits do registrador clássico! Para isso, usamos o seguinte comando:

```
circuito.measure(q, c)
```

Se quiséssemos, poderíamos ter medido somente um qubit, deixando os demais intactos. Por exemplo, para medir somente o primeiro qubit do registrador quântico e armazenar o resultado no primeiro bit do registrador clássico, usaríamos o seguinte comando:

```
circuito.measure(q[0], c[0])
```

Nesse ponto, já temos um circuito básico porém completo — com portas quânticas e medições. Para termos certeza de que fizemos tudo corretamente, é interessante visualizar o circuito. Podemos facilmente obter essa visualização no Qiskit usando o seguinte comando¹⁴:

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

O resultado gerado pelo código acima corresponde à Figura 20. Em vez de `output='mpl'`, poderíamos ter `output='latex'`. O primeiro significa que a visualização será gerada usando internamente o Matplotlib, um pacote bastante completo para geração de gráficos em Python. Já o segundo significa que a visualização será gerada usando internamente o \LaTeX , um sistema de preparação de documentos muito usado em matemática, física, computação e áreas semelhantes. De todo modo, essa escolha faz pouca diferença para o usuário. Não é necessário conhecer comandos de Matplotlib ou de \LaTeX .

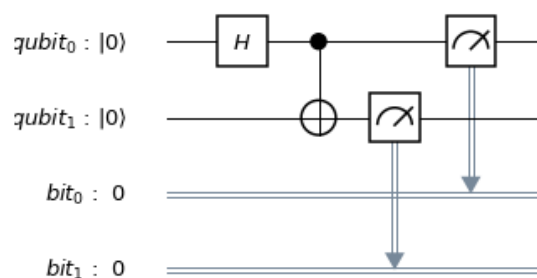


Figura 20. Visualização do circuito gerado pelo Qiskit

O Qiskit permite também obter diretamente o código-fonte Qasm para os circuitos quânticos gerados. Para isso, basta executar o seguinte comando:

```
codigo_qasm = circuito.qasm()
print(codigo_qasm)
```

¹⁴Caso esteja sendo usado o Jupyter Notebook, recomenda-se incluir antes uma linha com o comando `%matplotlib inline`, para indicar que a visualização deve ser exibida no próprio browser, junto com o código.

No caso do circuito que estamos considerando em nosso exemplo, o código-fonte Qasm gerado é o seguinte:

```
OPENQASM 2.0;
include "qelib1.inc";
qreg qubit[2];
creg bit[2];
h qubit[0];
cx qubit[0],qubit[1];
measure qubit[0] -> bit[0];
measure qubit[1] -> bit[1];
```

7.2.2. Executando o programa quântico

Agora que temos um programa quântico completo, podemos executá-lo em um simulador ou em um computador quântico real. Utilizando somente os comandos que aprendemos até aqui, a forma mais imediata de executar o programa seria gerar o código Qasm no Qiskit e em seguida copiá-lo no composer do IBM Q Experience. No entanto, veremos que ao utilizarmos o Qiskit, não precisamos mais do composer.

Para simular o circuito quântico, precisamos importar também o `BasicAer` e o `execute`. Também é útil importar uma ferramenta de visualização. Obtemos tudo isso por meio dos seguintes comandos:

```
from qiskit import BasicAer, execute
from qiskit.tools.visualization import *
```

Todos esses comandos `import` podem ficar agrupados numa única célula no início do notebook, juntamente com os que já havíamos introduzidos na seção anterior. O motivo de os estamos introduzindo gradativamente é puramente didático, para que o leitor compreenda em quais situações cada comando é necessário. Note que o Python também permite utilizar o asterisco para indicar que tudo o que houver em um determinado módulo deve ser importado.

Para simular o circuito, precisamos escolher um dos *backends* disponíveis no `BasicAer`. Para obter uma lista completa, usamos o comando `BasicAer.backends()`. Atualmente há três *backends* disponíveis no `BasicAer`: `qasm_simulator`, para simulação fiel ao funcionamento do computador quântico real; `statevector_simulator`, para simulação visando obter as amplitudes do estado final; e `unitary_simulator`, para obtenção da matriz unitária correspondente ao circuito.

Para executar o simulador no `qasm_simulator`, nosso circuito precisa ter pelo menos uma medição. Afinal, não se pode simular fielmente o funcionamento de um computador quântico se não houver alguma medição que gere resultado clássico. Para simular 1024 repetições do circuito em um computador quântico, por exemplo, usamos os seguintes comandos:

```
backend = BasicAer.get_backend('qasm_simulator')
```

```
job = execute(circuito, backend, shots=1024)
```

Agora todas as informações da simulação estão armazenadas no objeto `job`, mas o formato ainda é difícil para um humano ler. Podemos extrair o resultado e realizar uma contagem de todas as medições obtidas:

```
resultado = job.result()
contagem = resultado.get_counts()
```

O objeto `contagem` é um dicionário do Python — mas o leitor não precisa se preocupar com isso. Felizmente, o Qiskit já inclui algumas ferramentas de visualização muito úteis. Por exemplo, o comando `plot_histogram(contagem)` gera um histograma como da figura 21.

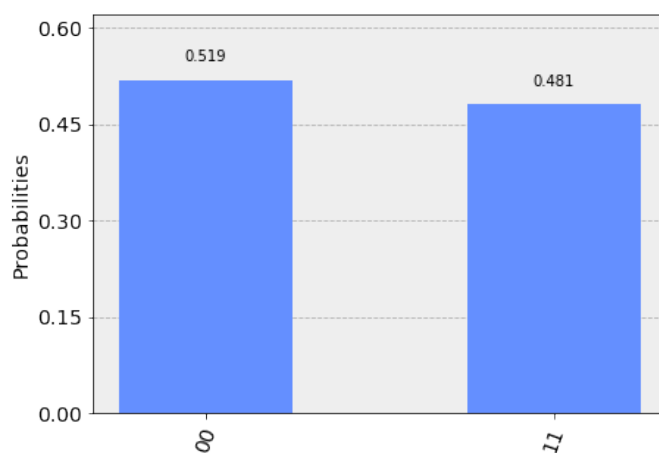


Figura 21. Visualização do resultado da simulação usando `qasm_simulator`

O backend `statevector_simulator` é utilizado de forma semelhante, por meio dos seguintes comandos:

```
backend = BasicAer.get_backend('statevector_simulator')
job = execute(circuito, backend)
resultado = job.result()
```

No entanto, ao ler o resultado estamos interessados em um vetor de estado, e não em uma contagem de resultados de medições. O comando que devemos utilizar aqui, portanto, é o seguinte:

```
estado = resultado.get_statevector()
```

Se imprimirmos o estado resultante com `print(resultado)`, veremos um *array* de números complexos. Este array pode ser manipulado de diversas formas dentro do Python. O Qiskit provê algumas visualizações úteis, como por exemplo o comando `plot_state_city(estado)`, que produz gráficos das partes real e imaginária da *matriz densidade* de um sistema, como na figura 22. Matrizes densidade são uma forma de representação dos estados de um sistema quântico.

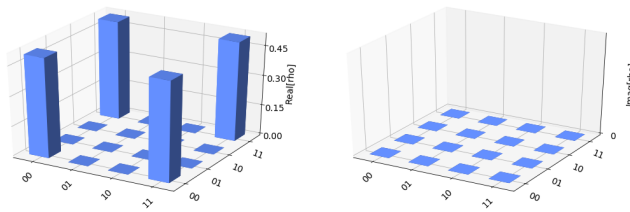


Figura 22. Visualização do resultado da simulação usando `statevector_simulator`

Para executar o simulador no `unitary_simulator`, nosso circuito não pode ter medições. Afinal, a medição não é uma operação unitária. Os comandos a serem executados nesse caso são os seguintes:

```
backend = BasicAer.get_backend('unitary_simulator')
job = execute(circuito, backend)
resultado = job.result()
```

O resultado dessa vez é uma matriz unitária, que pode ser extraída com o seguinte comando:

```
matriz = resultado.get_unitary()
```

A matriz resultante pode ser manipulada no próprio Python, usando pacotes como o *Numpy*.

Além dos backends de simulação mencionados acima, que são executados localmente no computador do usuário, o Qiskit também disponibiliza outros backends que direcionam os jobs para a própria IBM: a maioria deles direciona para computadores quânticos reais, e um direciona para ambiente de simulação de alto desempenho.

Para usar esses backends, é necessário acessar a conta no IBM Q Experience e copiar o API Token. Em seguida, deve-se substituir esse token no código abaixo:

```
from qiskit import IBMQ
IBMQ.save_account('Colar-Token-Aqui')
```

Isso fará com que um arquivo seja salvo no computador contendo informações necessárias para contactar os servidores da IBM. Basta executar uma vez em cada computador onde se queira usar o Qiskit. Depois disso, para acessar a conta IBM Q Experience a partir do Qiskit, será suficiente usar o comando

```
provedor = IBMQ.load_account()
```

Para obter uma listagem completa dos backends disponíveis, usa-se o comando `provedor.backends()`. Os backends disponíveis atualmente no IMBQ são `ibmq_ourense`, `ibmq_vigo` e `ibmqx2` de 5 qubits; `ibmq_16_melbourne` de 14 qubits; e `ibmq_qasm_simulator`, para executar remotamente no simulador clássico de alto desempenho, com suporte para até 32 qubits.

Por exemplo, para executar o circuito no IBM Q 5 Yorktown com 1024 repetições, fazemos o seguinte:

```
maquina = provedor.get_backend('ibmqx2')
job = execute(circuito, maquina, shots=1024)
```

Com isso, o job entra na fila para ser executado no computador quântico. Note que a espera pode demorar muitas horas, e o notebook precisa permanecer aberto e com conexão à Internet. Para saber a posição do job na fila em tempo real, pode-se usar o seguinte código:

```
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

Quando o job termina de executar, podemos extrair e visualizar o resultado com comandos muito semelhantes aos que já estamos acostumados:

```
resultado = job.result()
contagem = resultado.get_counts()
plot_histogram(contagem)
```

7.3. Implementação do algoritmo de Grover

Agora que já sabemos como implementar circuitos quânticos básicos no Qiskit e como executá-los em um simulador ou em um computador quântico da IBM, podemos estudar um exemplo mais sofisticado. O algoritmo de Grover é ideal para isso. Já estudamos esse importante algoritmo de busca na seção anterior, e já implementamos uma versão dele usando Qasm. No entanto, como os computadores quânticos da IBM não implementam diretamente portas com dois ou mais controles, tivemos de fazer uma decomposição do operador de Grover que foi muito custosa.

Nesta subseção, iremos implementar uma versão melhorada do algoritmo de Grover para uma lista com $N = 4$ elementos. Dessa vez usaremos uma decomposição mais econômica do operador de Grover. Os computadores quânticos atuais ainda acumulam muitos erros quando executam longas sequências de operações, então para conseguir bons resultados é importante reduzirmos tanto quanto possível o número de portas quânticas em nossos circuitos. Para reformular o circuito de Grover, vamos seguir o roteiro da subseção 6.7.

Primeiro, temos que definir os registradores e o circuito quântico. Precisamos de dois qubits no registrador quântico e dois no registrador clássico.

```
q = QuantumRegister(2, 'qubit')
c = ClassicalRegister(2, 'bit')
circuito = QuantumCircuit(q, c)
```

Agora, introduzimos no circuito as portas que vão colocar os qubits em superposição uniforme:

```
circuito.h(q[0])
circuito.h(q[1])
```

Para analisar a evolução parcial do algoritmo, podemos executar o circuito no backend `statevector_simulator` já nesse ponto, antes mesmo de ter o circuito completo. Fazemos isso com comandos que já estudamos na seção anterior:

```

backend = BasicAer.get_backend('statevector_simulator')
job = execute(circuito, backend)
estado = job.result().get_statevector()
print(estado)

```

O resultado é o vetor $[0.5+0.j \ 0.5+0.j \ 0.5+0.j \ 0.5+0.j]$, onde j é a notação do Python para a unidade imaginária. Portanto, o vetor corresponde ao estado $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$.

A próxima etapa é a implementação do oráculo. Para facilitar a visualização, sugerimos usar o comando `barrier` entre uma etapa e outra. Esses comandos podem ser removidos antes de executar o circuito no computador real.

Para construir o oráculo, vamos seguir o roteiro da subseção 6.7. Se quisermos executar o circuito no IBM Q 5 Yorktown, podemos usar a seguinte sequência de comandos:

```

circuito.h(q[1])
circuito.cx(q[0], q[1])
circuito.h(q[1])

```

Simulando novamente o circuito parcial no `statevector_simulator`, obtemos o vetor $[0.5+0.j \ 0.5+0.j \ 0.5+0.j \ -0.5+0.j]$, que corresponde ao estado $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$. Portanto, somente o elemento procurado teve sua fase invertida.

Se preferirmos executar o circuito no IBM Q 5 Tenerife, é recomendável inverter a posição dos qubits para respeitar o mapa de conectividade do computador. Agora vamos começar a construir o operador de Grover. Esse operador pode ser implementado com a seguinte sequência de comandos:

```

circuito.h(q[0])
circuito.z(q[1])

circuito.x(q[0])
circuito.cx(q[0], q[1])
circuito.x(q[0])

circuito.h(q[0])
circuito.z(q[1])

```

Simulando novamente o circuito parcial no `statevector_simulator`, obtemos o vetor $[0.+0.j \ 0.+0.j \ 0.+0.j \ -1.+0.j]$, que corresponde ao estado $-|11\rangle$. Portanto, temos exatamente o elemento que estávamos procurando. Para outros valores de N , o resultado não seria exato, mas teria uma amplitude suficientemente alta para o estado procurado.

Finalmente, podemos medir os dois qubits que armazenam o elemento procurado e salvar o resultado no registrador clássico:

```

circuito.measure(q, c)

```

Agora, para visualizar o circuito, usamos os comandos que já aprendemos. O resultado está na figura 23.

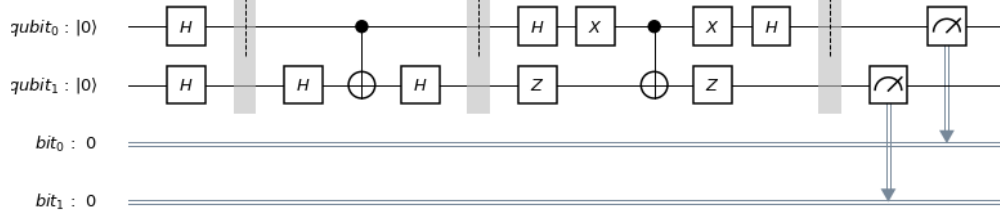


Figura 23. Circuito quântico para algoritmo de Grover implementado no Qiskit

O simulador `statevector_simulator` não funciona se houver uma medição, portanto não podemos mais executá-lo. No entanto, podemos executar o simulador `qasm_simulator`, que nos dará 100% de medições resultando no estado $|11\rangle$. Se quisermos executar o circuito no computador quântico da IBM, usamos os seguintes comandos:

```
job = execute(circuito, backend=maquina, shots=1024)
res = job.result()
contagem = res.get_counts()
```

Como o computador quântico real não é perfeito, não devemos esperar que 100% das medições sejam do estado procurado. O resultado obtido em nosso teste foi o seguinte: dentre 1024 rodadas, o estado $|00\rangle$ foi obtido 22 vezes, o estado $|01\rangle$ foi obtido 58 vezes, o estado $|10\rangle$ foi obtido 114 vezes, e o estado $|11\rangle$ foi obtido 830 vezes. Portanto, obtivemos o elemento procurado em 81% das vezes. Na figura 24 temos o histograma correspondente.

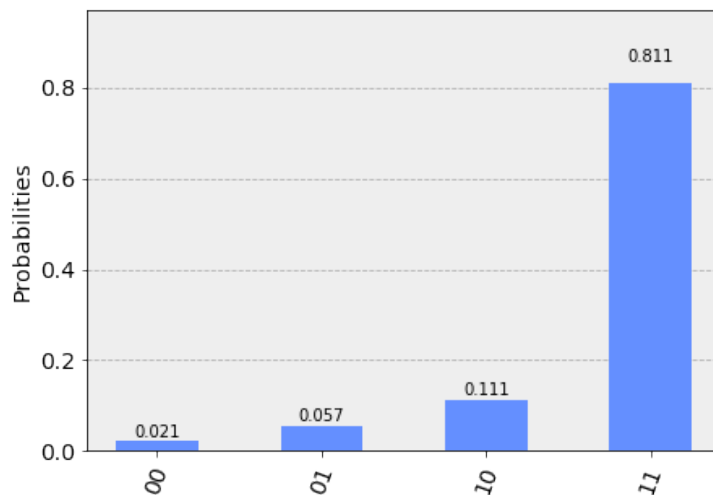


Figura 24. Resultado da execução do algoritmo de Grover no IBM Q 5 Yorktown

O leitor deve notar que o circuito dessa seção ficou bastante compacto, principalmente se comparado com a primeira versão do algoritmo de Grover que apresentamos.

Isso só foi possível pois no final da seção anterior nos dedicamos a reescrever o circuito da forma mais eficiente possível. Essa busca pelo melhor circuito possível é, atualmente, a parte mais desafiadora na programação de computadores quânticos. Convém ressaltar que o Qiskit permite incluir portas bastante sofisticadas no circuito quântico, e já cuida das decomposições automaticamente. Se incluirmos, por exemplo, uma porta Toffoli, ou uma porta CNOT que não respeite o mapa de conectividade do computador quântico, o Qiskit consegue “compilar” o circuito de modo que ele seja executável no hardware real. Em princípio, isso poderia simplificar bastante nosso trabalho de escrever programas em Qiskit, porém geraria circuitos muito maiores depois de compilados. Como os computadores quânticos atuais ainda possuem baixo tempo de coerência, um circuito maior representa também uma taxa de erros muito maior.

Apêndice – Portas do *Composer* da IBM

Portas de 1 qubit

Porta Hadamard [$H^2 = \text{ID}$]

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

Matrizes de Pauli [$X^2 = Y^2 = Z^2 = \text{ID}$]

$$\text{ID} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Porta fase e porta fase conjugada [$S^2 = (S^\dagger)^2 = Z$]

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$$

Porta $\pi/8$ ou porta T e sua conjugada [$T^2 = S$ e $(T^\dagger)^2 = S^\dagger$]

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, \quad T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}$$

Porta U_1 [$U_1(\lambda) = U_3(0, 0, \lambda)$]

$$U_1(\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$$

Porta U_2 [$U_2(\phi, \lambda) = U_3(\pi/2, \phi, \lambda)$]

$$U_2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{bmatrix}$$

Porta U_3

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos \frac{\theta}{2} & -e^{i\lambda} \sin \frac{\theta}{2} \\ e^{i\phi} \sin \frac{\theta}{2} & e^{i(\lambda+\phi)} \cos \frac{\theta}{2} \end{bmatrix}$$

Porta R_x

$$R_x(\theta) = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

Porta R_y

$$R_y(\theta) = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

Porta R_z

$$R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$$

Portas de 2 qubits

Porta CNOT ou cX com alvo em q[0] [\oplus de cor azul e inversão]

$$\text{CNOT} = \begin{bmatrix} \text{ID} & 0 \\ 0 & X \end{bmatrix}$$

Porta CNOT ou cX com alvo em q[1] [\oplus de cor azul]

$$\text{CNOT} = \begin{bmatrix} X & 0 \\ 0 & \text{ID} \end{bmatrix}$$

Porta cY com alvo em q[0]

$$cY = \begin{bmatrix} \text{ID} & 0 \\ 0 & Y \end{bmatrix}$$

Porta cZ com alvo em q[0]

$$cZ = \begin{bmatrix} \text{ID} & 0 \\ 0 & Z \end{bmatrix}$$

Porta cH com alvo em q[0]

$$cH = \begin{bmatrix} \text{ID} & 0 \\ 0 & H \end{bmatrix}$$

Porta $swap$ [$swap(|\psi_1\rangle|\psi_2\rangle) = |\psi_2\rangle|\psi_1\rangle$]

$$\begin{array}{c} \text{---} \times \text{---} \\ | \\ \text{---} \times \text{---} \end{array} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Portas de 3 qubits

Porta Toffoli ou ccX com alvo em q[0] [\oplus de cor lilás e inversão]

$$ccX = \begin{bmatrix} \text{ID} & & & \\ & \text{ID} & & \\ & & \text{ID} & \\ & & & X \end{bmatrix}$$

Agradecimentos

Os autores agradecem Cauê Teixeira por correções importantes.

Referências

- [1] C. G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, and K. Bertels. The engineering challenges in quantum computing. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pages 836–845, March 2017.
- [2] S. Axler. *Linear Algebra Done Right*. Springer, New York, 1997.
- [3] S. Barnett. *Quantum Information*. Oxford University Press, New York, 2009.
- [4] G. Benenti, G. Casati, and G. Strini. *Principles of Quantum Computation and Information: Basic Tools and Special Topics*. World Scientific Publishing, River Edge, 2007.
- [5] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, 1973.
- [6] J. A. Bergou and M. Hillery. *Introduction to the Theory of Quantum Information Processing*. Springer, 2013.
- [7] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.
- [8] R. B. Boppana and M. Sipser. Chapter 14 - The complexity of finite functions. In J. V. Leeuwen, editor, *Algorithms and Complexity*, Handbook of Theoretical Computer Science, pages 757 – 804. Elsevier, Amsterdam, 1990.
- [9] A. Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, pages 733–744, 1977.
- [10] C. S. Calude and E. Calude. The road to quantum computational supremacy. *ArXiv:1712.01356*, 2019.
- [11] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open quantum assembly language. *arXiv e-prints 1707.03429*, pages 1–24, Jul 2017.
- [12] M. Davis. *Computability & Unsolvability*. Dover Publications, 1982.
- [13] E. Desurvire. *Classical and Quantum Information Theory: An Introduction for the Telecom Scientist*. Cambridge University Press, 2009.
- [14] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Royal Society London Ser. A*, pages 96–117, 1985.
- [15] D. E. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.
- [16] D. Dieks. Communication by EPR devices. *Physics Letters A*, 92(6):271 – 272, 1982.
- [17] L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79(2):325–328, 1997.
- [18] M. Hayashi, S. Ishizaka, A. Kawachi, G. Kimura, and T. Ogawa. *Introduction to Quantum Information Science*. Springer, 2014.
- [19] M. Hirvensalo. *Quantum Computing*. Springer, 2010.
- [20] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, New York, 2007.

- [21] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, Boston, 2002.
- [22] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, July 1961.
- [23] S. Lang. *Introduction to Linear Algebra*. Undergraduate Texts in Mathematics. Springer, New York, 1997.
- [24] R. J. Lipton and K. W. Regan. *Quantum Algorithms via Linear Algebra: A Primer*. MIT Press, 2014.
- [25] A. Mandviwalla, K. Ohshiro, and B. Ji. Implementing Grover’s algorithm on the IBM quantum computers. In *2018 IEEE International Conference on Big Data*, pages 2531–2537, 2018.
- [26] D. C. Marinescu and G. M. Marinescu. *Approaching Quantum Computing*. Pearson/Prentice Hall, Michigan, 2005.
- [27] N. D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, New York, 2007.
- [28] M. A. Nielsen and I. L. Chuang. *Computação Quântica e Informação Quântica*. Editora Bookman, 2005.
- [29] J. L. Park. The concept of transition in quantum mechanics. *Foundations of Physics*, 1(1):23–33, Mar 1970.
- [30] R. Portugal. *Quantum Walks and Search Algorithms*. Springer, Cham, 2018.
- [31] R. Portugal, C. C. Lavor, L. M. Carvalho, and N. Maculan. *Uma Introdução à Computação Quântica*, volume 8 of *Notas em Matemática Aplicada*. SBMAC, São Carlos, 1st edition, 2004.
- [32] J. Preskill. Quantum computing in the NISQ era and beyond. *arXiv:1801.00862*, 2018.
- [33] E. Rieffel and W. Polak. *Quantum Computing, a Gentle Introduction*. MIT Press, Cambridge, 2011.
- [34] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, Dec 1938.
- [35] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.
- [36] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.
- [37] J. Stolze and D. Suter. *Quantum Computing, Revised and Enlarged: A Short Course from Theory to Experiment*. Wiley-VCH, 2008.
- [38] G. Strang. *Linear Algebra and Its Applications*. Brooks Cole, 1988.
- [39] T. Toffoli. Reversible computing. In J. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, pages 632–644. Springer, Berlin, 1980.
- [40] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.
- [41] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, Inc., New York, NY, USA, 1987.
- [42] C. P. Williams. *Explorations in Quantum Computing*. Springer, 2008.
- [43] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*,

299:802–803, 1982.

- [44] N. S. Yanofsky and M. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008.

Como citar este trabalho:

```
@INPROCEEDINGS{PM:2019,  
  AUTHOR= {Renato Portugal and Franklin Marquezino},  
  TITLE= {{Introdução à Programação de  
  Computadores Quânticos}},  
  BOOKTITLE= {CSBC 2019 -- 38° JAI},  
  ADDRESS= {Belém -- Pará},  
  DAYS= {14-18},  
  MONTH= {jul},  
  YEAR= {2019},  
  PAGES= {1--51},  
}
```

R. Portugal and F. Marquezino. Introdução à Programação de Computadores Quânticos. In CSBC 2019 – 38° JAI, pages 1–51, Belém – Pará, jul 2019.