Guilherme Furtado Lopes de Macedo

Conceitos Básicos de Computação Quântica Aplicados no Qiskit.

Campo Grande

2024

Guilherme Furtado Lopes de Macedo

Conceitos Básicos de Computação Quântica Aplicados no Qiskit.

Monografia apresentada ao Instituto de Física da Universidade Federal de Mato Grosso do Sul, sendo parte dos requisitos para a obtenção do título de bacharel em Física.

Universidade Federal de Mato Grosso do Sul - UFMS

Instituto de Física - INFI

Orientador: Dr. João Vítor Batista Ferreira

Campo Grande

2024

Agradecimentos

A minha família, especialmente ao meu avô, Alberto Furtado (*in memoriam*) e avó Maria de Lourdes.

Aos professores do Instituto de Física, principalmente ao Professor João Vítor que me guiou e orientou.

Aos meus amigos, Matheus Auler, Dorivaldo Neto, Wesley Erick, Dario Amorim, Gustavo Marques e Julia Pires. Durante diversos momentos de desafio me emprestaram sua força, tempo, paciência e amizade.

A minha mãe Thays Barrios que durante todo o período da graduação me ajudou e confortou.

A meu pai Fernando Lopes pelos conselhos e ajuda durante todos os anos de Faculdade.

A meu irmão Gustavo Furtado pelos abraços e pela motivação.

Resumo

A mecânica quântica surgiu para solucionar as inconsistências que a teoria clássica era insuficiente para explicar, culminando em uma nova abordagem da realidade na Física. Com esta mudança diversas portas se abriram e muitas novas áreas se desenvolveram, uma delas é a computação quântica. A computação quântica nasce em meados de 1980 e se desenvolve com os avanços da Mecânica Quântica. Conceitos como Bits quânticos, ou qubits, portas lógicas quânticas, circuitos e algoritmos quânticos foram fundamentais para seu desenvolvimento e estudo. Em 2017, a IBM (International Business Machines Corporation) lançou um software de desenvolvimento aberto em linguagem Python, o Qiskit. Este rapidamente se tornou uma ferramenta útil para estudar computação quântica. Sendo assim, os conceitos básicos de computação quântica podem ser usados no Qiskit e estudados por ele. Nesta Revisão bibliográfica será feito um estudo dos conceitos básicos da computação quântica e algumas aplicações usando a ferramenta Qiskit.

Palavras-chave: Computação Quântica, Mecânica Quântica, Qiskit, Bits Quânticos, Portas Lógicas Quânticas, Circuitos Quânticos, Algoritmos Quânticos, Teleporte Quântico, Algoritmo de Deutsch, Algoritmo de Deutsch-Josza.

Abstract

Quantum mechanics emerged to resolve the inconsistencies that classical theory was unable to explain, culminating in a new approach to reality in physics. With this change, several doors opened and many new areas developed, one of which is quantum computing. Quantum computing was born in the mid-1980s and developed with advances in quantum mechanics. Concepts such as quantum bits, quantum logic gates, quantum circuits and algorithms were fundamental to its development and study. In 2017, IBM (International Business Machines Corporation) launched open development software in the Python language, Qiskit. This quickly became a useful tool for studying quantum computing. As such, the basic concepts of quantum computing can be used in Qiskit and studied using it. This literature review will study the basic concepts of quantum computing and some applications using the Qiskit tool.

Keywords:Quantum Computing, Quantum Mechanics, Qiskit, Quantum Bits, Quantum Logic Gates, Quantum Circuits, Quantum Algorithms, Quantum Teleportation, Deutsch Algorithm, Deutsch-Josza Algorithm.

Sumário

1 Introdução

A mecânica quântica surge no início do século XX como consequência da falta de respostas para problemas que a teoria clássica pecava em explicar, sendo o primeiro deles a radiação de corpo negro. Max Planck, em 1900, buscando solucionar a radiação de corpo negro, propõe a ideia da quantização da energia, com isso, a teoria quântica deu seus primeiros passos [1]. Futuramente em 1905, Albert Einstein questionou a teoria clássica da luz, por meio de uma ideia similar a de Planck, propondo que a luz poderia ser descrita como pequenos pacotes quantizados. Posteriormente, estes seriam chamados de fótons [2]. Em 1913, Niels Bohr traz uma nova compreensão sobre a estrutura atômica, descrevendo o elétron se movendo por órbitas circulares ao redor do núcleo do átomo, em que as órbitas possuiriam energias quantizadas [3]. O início da teoria quântica parecia promissor e inexplorado, gerando uma agitação no mundo da Física, porém, esse era apenas o início.

A consolidação da mecânica quântica aconteceria na década de 1920, tendo como grandes responsáveis Werner Heisenberg e Erwin Schrödinger. O ano de 1925 foi marcado por Heisenberg que tentou sanar os questionamentos que a teoria de Bohr deixou [4]. No mesmo ano, Max Born e Pascual Jordan introduziram a notação matricial na teoria quântica, a partir do trabalho de Heisenberg [5]. Em 1926, Erwin Schrödinger apresentou uma equação capaz de descrever a evolução de sistemas quânticos no tempo, por meio de uma função introduzida por ele chamada de função de onda [6]. Após anos, a teoria quântica evoluiu suficientemente para originar diversas áreas. Uma delas é a computação quântica, que teria seu início na década de 1980.

Em 1982, a possibilidade de aliar a computação com a mecânica quântica é apresentada por Richard Feynman, discutindo que uma descrição coesa de sistemas quânticos poderia ser um desafio, devido à natureza da teoria quântica. Ele sugere que serão necessárias novas abordagens e introduz pela primeira vez a ideia de um computador quântico [7]. Futuramente, em 1985, David Deutsch propôs o conceito de um computador quântico universal e apresentou o primeiro algoritmo quântico, mostrando a vantagem de se utilizar da computação quântica [8].

Em 1984, Charles Bennett e Gilles Brassard desenvolveram o protocolo BB84, estabelecendo as bases para a criptografia quântica e mostrando como o emaranhamento quântico poderia ser utilizado para criar comunicações seguras [9]. Em 1994, Peter Shor desenvolveu um algoritmo capaz de fatorar números grandes de modo eficiente, ameaçando a segurança da criptografia baseada em fatoração[10]. Posteriormente, Em 1996, Lov Grover apresentou um algoritmo de busca quântica que poderia realizar buscas em bases

de dados de forma efetiva, superando algoritmos clássicos para alguns casos[11]. Portanto, a mecânica quântica possibilitou, através de sua mudança no mundo da Física, uma nova abordagem para a computação que cresceu imensamente nos últimos anos. Na próxima seção, estudaremos os conceitos básicos da computação quântica.

2 Revisão bibliográfica

2.1 Bits quânticos

Na computação clássica o processamento/armazenagem de informação é feita utilizando um sistema físico que tem dois estados bem definidos e que podem ser bem determinados experimentalmente: *desligado* ou *ligado*, *tensão elétrica baixa* ou *tensão elétrica alta*, etc. Essa quantidade mínima de informação do sistema físico é denominada de bit e, por simplicidade, é codificado em dois valores numéricos: 0 ou 1.

Na computação quântica é utilizado um sistema físico que pode não ter um estado bem definido enquanto não se realiza sua medida. Além disso, permite-se que ocorram fenômenos como *superposição* de estados diferentes e *emaranhamento* de sistemas iguais ou diferentes: a mecânica quântica adiciona as características probabilísticas de estados que se sobrepõem ou se excluem. Por isso a informação sobre o sistema físico é representada por bits quânticos, comumente chamados de qubits. E ao invés de apenas dois valores numéricos é possível, a princípio, manipular uma quantidade infinita de valores antes de se realizar a medida e obter os valores clássicos 0 ou 1. O formalismo de Dirac e a álgebra linear se mostram eficientes como linguagem matemática da computação quântica: a descrição de estados quânticos de um sistema físico e de grandezas físicas é feita por matrizes e operações matriciais.

Seja um sistema físico com dois qubits bem definidos normalizados e ortogonais entre si. Eles formam uma base para qualquer outro estado quântico do mesmo espaço de Hilbert e são representadas pelos kets $|0\rangle$ e $|1\rangle$:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{1}$$

Um estado que seja combinação linear de $|0\rangle$ e $|1\rangle$ é representado por:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{2}$$

sendo α e β números complexos, de modo que $|\alpha|^2 + |\beta|^2 = 1$. Logo $|\psi\rangle$ pode representar infinitos estados. Alguns destes recebem denominações específicas, como os estados $|+\rangle$ e $|-\rangle$:

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \text{ e } |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle. \tag{3}$$

Segundo a mecânica quântica, para determinar o valor de um observável de um sistema em determinado estado é necessário medi-lo. Isso colapsa o estado original em um

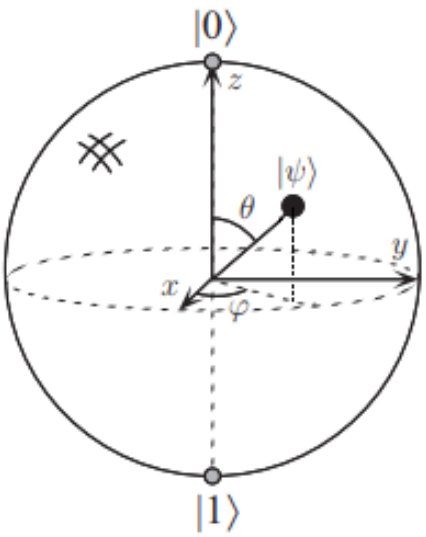
estado clássico, fazendo-o assumir um dos valores 0 ou 1 e com probabilidades determinadas pelos coeficientes. No caso $|+\rangle$ podemos obter resultado 0 com probabilidade $|\alpha|^2 = 1/2$ ou 1 com probabilidade $|\beta|^2 = 1/2$, onde $|\alpha|^2+|\beta|^2=1$ mostrando que o estado do qubit está normalizado. De maneira semelhante, se medirmos $|-\rangle$ teremos 50% de chance de obter 0 ou 1.

Tomando a Equação 2 é possível rescrevê-la:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle. \tag{4}$$

Os números θ e ϕ determinam um ponto na Esfera de Bloch. A Esfera de Bloch (Figura 1) oferece uma visualização do estado de um único qubit, sendo extremamente útil para o entendimento de alguns processos na computação quântica.

Figura 1 – Representação de qubit na Esfera de Bloch.



Fonte:Adaptado de [12], Capítulo 1 e página 15.

Pode-se demonstrar que além da base computacional $|0\rangle$ e $|1\rangle$ também é possível escrever os estados de um qubit em outras bases, tal qual, a base $|+\rangle$ e $|-\rangle$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\frac{|+\rangle + |-\rangle}{\sqrt{2}} + \beta\frac{|+\rangle - |-\rangle}{\sqrt{2}} = \frac{\alpha+\beta}{\sqrt{2}}|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}|-\rangle. \tag{5}$$

Após medição nesta base obteríamos o estado $|+\rangle$ com probabilidade $\frac{|\alpha+\beta|^2}{2}$ e o estado $|-\rangle$ com probabilidade $\frac{|\alpha-\beta|^2}{2}$. Sendo assim, dado qualquer nova base $|a\rangle$ e $|b\rangle$ é possível reescrever estados definidos em uma base anterior.

Quando um sistema tem dois qubits, uma nova função de onda pode ser escrita:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{6}$$

Estes novos estados podem ser representados por matrizes coluna também:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{7}$$

Uma consequência desse tipo de estado é a possibilidade da medição de apenas um qubit. Por exemplo, ao medir apenas o primeiro qubit e o resultado for 0, o estado quântico resultante é uma parte do estado original renormalizado por um fator $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$. Isto é, a medição do primeiro qubit em 0 tem probabilidade $|\alpha_{00}|^2 + |\alpha_{01}|^2$ e o estado pós-medição é

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}. \tag{8}$$

2.1.1 Estados de Bell

Na década de 1960, John Bell, mostrou a possibilidade de existirem estados com dois qubits mais correlacionados entre si que qualquer equivalente clássico: eles não podem ser descritos como simples produtos de estados de um qubit [13]. Estes estados são conhecidos como estados de Bell (ou estados EPR) e estão mostrados na Tabela 1. Estes quatro estados de Bell são bastante importantes na computação quântica, sendo aplicados em processos como Teleporte Quântico, etc. Uma propriedade dos estados de Bell é que a medição do primeiro qubit já determina o valor do segundo qubit. Por exemplo, seja o estado de Bell β_{00}. Após a medição do primeiro qubit obtemos dois possíveis resultados: 0 com probabilidade 1/2, ocasionando no estado pós-medição $|\varphi'\rangle = |00\rangle$ e 1 com probabilidade 1/2, ocasionando no estado pós-medição $|\varphi'\rangle = |11\rangle$, por consequência, ao medir o segundo qubit ele sempre vai obter a mesma medida do primeiro.

Tabela 1 – Estados de Bell.

$(\|00\rangle + \|11\rangle)/\sqrt{2} \equiv \|\beta_{00}\rangle$
$(\|01\rangle + \|10\rangle)/\sqrt{2} \equiv \|\beta_{01}\rangle$
$(\|00\rangle - \|11\rangle)/\sqrt{2} \equiv \|\beta_{10}\rangle$
$(\|01\rangle - \|10\rangle)/\sqrt{2} \equiv \|\beta_{11}\rangle$

Fonte: [14].

É importante salientar que é impossível reescrever um estado de Bell em estados separados de seus qubits, isto é, $|\beta_{00}\rangle \neq (c_1 |0\rangle + c_2 |1\rangle) \otimes (c_3 |0\rangle + c_4 |1\rangle)$ sendo c_j um número complexo.

2.2 Porta lógicas quânticas

2.2.1 Porta lógicas de apenas um qubit

Portas lógicas em circuitos clássicos efetuam operações binárias. As portas lógicas quânticas tem as mesmas funções, contudo a maneira com que funcionam e são descritas é diferente. Digamos que tenhamos que levar um qubit no estado $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ para o estado $|\psi'\rangle = \alpha|1\rangle + \beta|0\rangle$, ou seja, queremos inverter o 0 e o 1. Semelhante a uma porta NOT clássica, a porta lógica quântica age linearmente no estado ψ. Na computação quântica, portas lógicas são similares a operadores unitários [1]. Portas lógicas efetuadas em apenas um qubit podem ser descritas como uma matriz 2 x 2. Para exemplificar melhor, apresentamos a porta lógica X (matriz de Pauli σ_x):

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{9}$$

Se efetuarmos a operação em $|\psi'\rangle$ temos:

$$X|\psi\rangle = |\psi'\rangle \rightarrow \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}. \tag{10}$$

Sendo assim, a porta X equivale à porta lógica NOT. Isto posto, podemos concluir que $X^\dagger X = I$:

$$X^\dagger X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \equiv I. \tag{11}$$

Outras portas lógicas são Y e Z (matrizes Pauli σ_y e σ_z), representadas por:

$$Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \text{ e } Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{12}$$

Além disso, temos também a porta lógica Hadamard, H, de suma importância para o entendimento de diversos tópicos importantes na computação quântica:

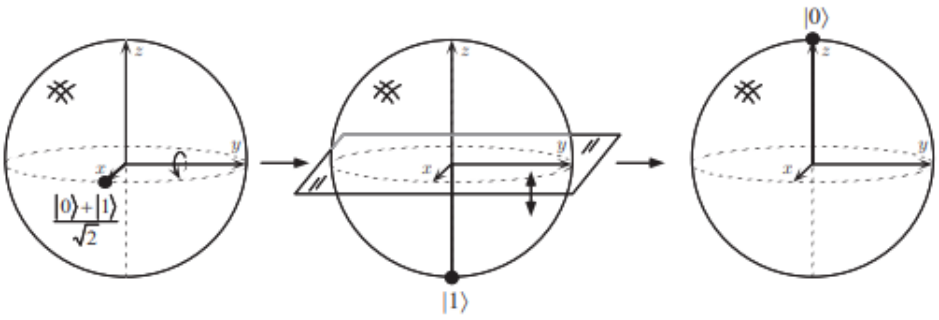
$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{13}$$

Na Figura 2 temos uma representação de uma porta lógica H atuando em $|\psi\rangle = |+\rangle$ através da Esfera de Bloch resultando em $|0\rangle$. O contrário também é válido, se a porta H está atuando em $|0\rangle$ resultará em $|+\rangle$.

$$H\,|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle\,. \tag{14}$$

[1] Se U é matriz unitária, então $U^\dagger U = UU^\dagger = 1$

Figura 2 – Porta lógica H atuando em $|\psi\rangle$ na representação da esfera de Bloch, sendo $H|+\rangle = |0\rangle$.



Fonte: [12] capitulo 1 e página 19.

Observamos que a operação Hadamard é simplesmente uma rotação na esfera em torno do eixo $\hat{y}$ por 90°, seguida por uma rotação em torno do eixo $\hat{x}$ por 180°.

Existe uma infinidade de portas lógicas possíveis. Outra forma de criá-las é combinando: a própria porta Hadamard é criada utilizando portas X e Z:

$$H = \frac{1}{\sqrt{2}}\left(\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}. \tag{15}$$

2.2.2 Porta lógica de múltiplos qubits

Um estado com dois qubits tem os estados possíveis $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$, ou combinação deles. Isto é interessante porque permite portas lógicas em que um dos qubits pode controlar a ação da porta no segundo qubit. Por exemplo a porta "CNOT" (porta NOT controlada) é representada pela matriz:

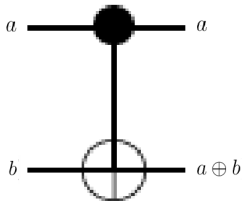
$$U_{CN} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}. \tag{16}$$

Aplicando U_{CN} em um estado de dois qubits, temos o seguinte:

$$|00\rangle \to |00\rangle; |01\rangle \to |01\rangle; |10\rangle \to |11\rangle; |11\rangle \to |10\rangle. \tag{17}$$

Quando o primeiro qubit é 0 ele mantém o segundo qubit, quando o primeiro qubit é 1 ele inverte o segundo qubit. A representação gráfica da porta CNOT está presente na Figura 3.

Figura 3 – Porta lógica CNOT.

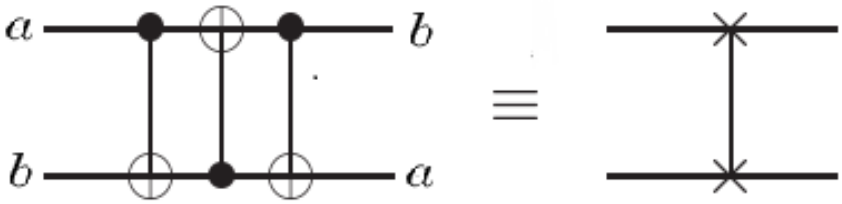


Fonte: Adaptado de [12], Capítulo 1 e página 24.

2.3 Circuitos quânticos

Circuitos em computadores clássicos consistem em fios e portas lógicas. Os fios carregam os bits de informação e as portas lógicas efetuam operações binárias neles. Circuitos quânticos são representados por fios e portas lógicas. Os fios são os qubits e devem ser lidos da esquerda para a direita. As portas são operações realizadas nos qubits, como a porta X. Este fio não é necessariamente físico e sim uma representação de um evento, seja a passagem de tempo, um fóton se movendo no espaço, etc. Na Figura 4 temos um exemplo de um circuito quântico básico:

Figura 4 – Circuito que alterna os estados de dois qubits.



Fonte: [12], Capítulo 1 e página 23.

Este circuito alterna os estados de dois qubits. Seja $|a, b\rangle$ o estado inicial, onde a é o primeiro qubit e b o segundo qubit. Na Figura 4, teremos três portas CNOT, sendo a segunda invertida, o qubit de controle neste caso é o segundo qubit. O estado inicial $|a, b\rangle$ após passar pela primeira porta CNOT muda: o primeiro qubit se manteve e o segundo inverte dependendo do valor do primeiro, efetuando a soma modular no segundo qubit obtemos o estado $|a, a \oplus b\rangle$.

$$|a, b\rangle \longrightarrow |a, a \oplus b\rangle. \tag{18}$$

Na segunda porta CNOT, o segundo bit, invertido ou não, é usado como controle de inversão do primeiro bit [2]

$$|a, a \oplus b\rangle \longrightarrow |a \oplus (a \oplus b), a \oplus b\rangle = |b, a \oplus b\rangle. \tag{19}$$

[2] Na soma base 2 temos $a \oplus a = b \oplus b = 0$, $0 + a = a$, $0 + b = b$

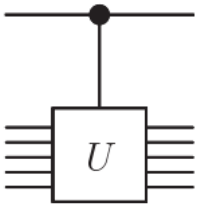
Na última porta CNOT, o primeiro qubit é usado novamente como controle do segundo e obtém-se a inversão desejada

$$|b, a \oplus b\rangle \rightarrow |b, (a \oplus b) \oplus b\rangle = |b, a\rangle. \tag{20}$$

De maneira similar, o primeiro qubit se mantém e soma modular é efetuada no segundo, por fim, invertendo o estado do qubit.

Generalizando: para qualquer matriz unitária U que atua em n qubits pode ser construída outra porta lógica controlada: U_c.

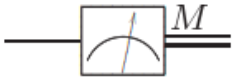
Figura 5 – Porta lógica controlled-U.



Fonte:[12], Capítulo 1 e página 24.

Na Figura 6, está exposto o símbolo que representa a medição de um qubit em um circuito. O resultado da medida é sempre um valor clássico: 0 ou 1.

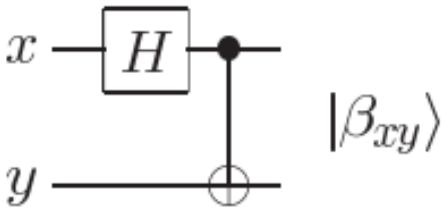
Figura 6 – Símbolo de medição em um circuito quântico.



Fonte:Adaptado de [12], Capítulo 1 e página 24.

Outro circuito extremamente útil e altamente utilizado - presente no circuito do teleporte quântico - consiste em uma porta lógica Hadamard seguida por uma porta lógica CNOT (Figura 7).

Figura 7 – Circuito para a criação de estados de Bell.



Fonte: Adaptado de [12], Capítulo 1 e página 26.

Este circuito cria uma sobreposição de estados através da aplicação da porta Hadamard, em seguida, gera emaranhamento a partir da porta CNOT. Dessa forma, os

estados de Bell são criados a depender do valor de x e y, de modo que, se as entradas são $x = 0$ e $y = 1$ obtemos o estado de Bell $|\beta_{01}\rangle$. Na Tabela 1, temos a tabela verdade do circuito de criação de estados de Bell.

Na próxima seção, mostramos como a combinação de qubits e portas lógicas permitem a construção de algoritmos quânticos.

2.4 Teleporte Quântico e algoritmos quânticos.

Nesta parte, iremos utilizar todos os conceitos abordados anteriormente para explicar o Teleporte quântico e os algoritmos de Deutsch's e Deutsch–Jozsa.

2.4.1 Teleporte quântico.

Alice e Bob são dois pesquisadores de fenômenos quânticos. Em um certo momento eles geraram um estado de Bell compartilhado, cada um portando um qubit. Depois de um certo intervalo de tempo, Alice quer "teleportar" um terceiro estado quântico para Bob, que o desconhece completamente. Este novo estado é denominado de $|\psi\rangle$. Alice cria um estado de Bell entre $|\psi\rangle$ e o seu qubit do estado de Bell compartilhado com Bob. Então ela mede os dois qubits em sua posse, obtendo um dos possíveis resultados: 00, 01, 10 e 11. Alice envia o resultado desta medição para Bob, usando meios clássicos. Ele, baseado nesta informação, efetua operações em seu qubit do estado de Bell compartilhado com ela, e reconstrói o estado quântico $|\psi\rangle$.

O circuito quântico que realiza esta tarefa tem diversas etapas. A primeira corresponde à criação, por Alice e Bob, de um dos quatro estados de Bell listados na Tabela 1. Cada um ficou de posse de um dos dois qubits. O qubit da Alice está emaranhado com o qubit do Bob, logo qualquer mudança em um interfere no outro. Como exemplo, vamos considerar que Alice e Bob criaram o estado de Bell $|\beta_{00}\rangle$ e denominamos o qubit da Alice de $Bell_0$ e o de Bob de $Bell_1$. Os algarismos 0 e 1 nestas denominações NÃO devem ser confundidos com os algarismos 0 e 1 nos kets. Para facilitar a compreensão, usaremos as cores vermelha e azul para os qubits pertencentes a Alice e Bob, respectivamente:

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{21}$$

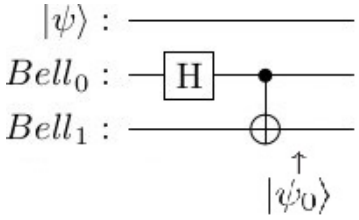
Na segunda etapa Alice manipula o estado quântico a ser teleportado $|\psi\rangle$. Como ele está em posse da Alice, vamos escrevê-lo como $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. Agora Alice tem posse de dois qubits e Bob tem um qubit. Estes três qubits constituem o estado quântico de entrada do circuito, que denominamos de $|\psi_0\rangle$:

$$|\psi_0\rangle = |\psi\rangle\,|\beta_{00}\rangle = \Big(\alpha\,|0\rangle + \beta\,|1\rangle\Big)\left(\frac{|00\rangle + |11\rangle}{\sqrt{2}}\right). \tag{22}$$

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}\left[\alpha\,|0\rangle\left(|00\rangle + |11\rangle\right) + \beta\,|1\rangle\left(|00\rangle + |11\rangle\right)\right]. \tag{23}$$

A Figura 8 é o circuito até o momento.

Figura 8 – Circuito do teleporte quântico evoluindo do estado inicial até $|\psi_0\rangle$.



Fonte: Autor

Alice faz um emaranhamento entre $|\psi\rangle$ e $Bell_0$. Primeiro ela usa o qubit $|\psi\rangle$ como controle de uma porta CNOT atuando em $Bell_0$. O resultado é o estado $|\psi_1\rangle$.

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}\left[\alpha|0\rangle\left(|00\rangle + |11\rangle\right) + \beta|1\rangle\left(|10\rangle + |01\rangle\right)\right]. \tag{24}$$

Depois ela aplica a porta Hadamard no qubit $|\psi\rangle$, resultando no estado $|\psi_2\rangle$.

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}\left[\alpha\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(|00\rangle + |11\rangle\right) + \beta\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\left(|10\rangle + |01\rangle\right)\right]. \tag{25}$$

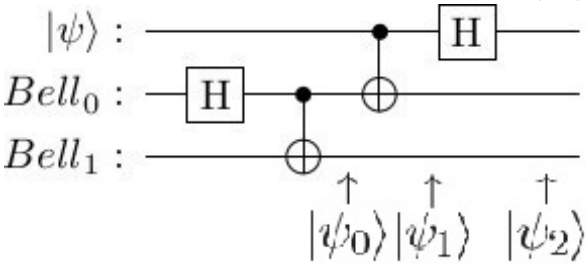
O termo $\alpha(|0\rangle+|1\rangle)(|00\rangle+|11\rangle)$ pode ser reescrito como $(\alpha\,|000\rangle+\alpha\,|011\rangle+\alpha\,|100\rangle+\alpha\,|111\rangle)$, e depois como $(|00\rangle\,\alpha\,|0\rangle + |01\rangle\,\alpha\,|1\rangle + |10\rangle\,\alpha\,|0\rangle + |11\rangle\,\alpha\,|1\rangle)$. Procedemos com os outros termos da mesma forma e reorganizamos.

$$|\psi_2\rangle = \frac{1}{2}\Big[|00\rangle(\alpha|0\rangle+\beta|1\rangle)+|01\rangle(\alpha|1\rangle+\beta|0\rangle)+|10\rangle(\alpha|0\rangle-\beta|1\rangle)+|11\rangle(\alpha|1\rangle-\beta|0\rangle)\Big]. \tag{26}$$

No estado $|\psi_2\rangle$ reorganizado, os qubits da Alice (vermelho) foram agrupados e o qubit do Bob (azul) tem os mesmos componentes α e β de $|\psi\rangle$. Este resultado é devido ao emaranhamento quântico.

A figura 9 é o circuito até o momento.

Figura 9 – Circuito do teleporte quântico evoluindo de $|\psi_0\rangle$ para $|\psi_1\rangle$ e $|\psi_2\rangle$.



Fonte: Autor

A terceira e última etapa é Alice medir os seus qubits (vermelho). O possível resultado é 00 ou 01 ou 10 ou 11.

Alice realiza a medida. O qubit $Bell_1$ do Bob (azul) irá colapsar em um dos quatro estados possíveis, indicado por $|\psi_3\rangle$. Se a medição de Alice for 00, então

$$00 \longmapsto |\psi_3\rangle \equiv [\alpha|0\rangle + \beta|1\rangle]. \tag{27}$$

Se a medição de Alice for 01, então

$$01 \longmapsto |\psi_3\rangle \equiv [\alpha|1\rangle + \beta|0\rangle]. \tag{28}$$

Se a medição de Alice for 10, então

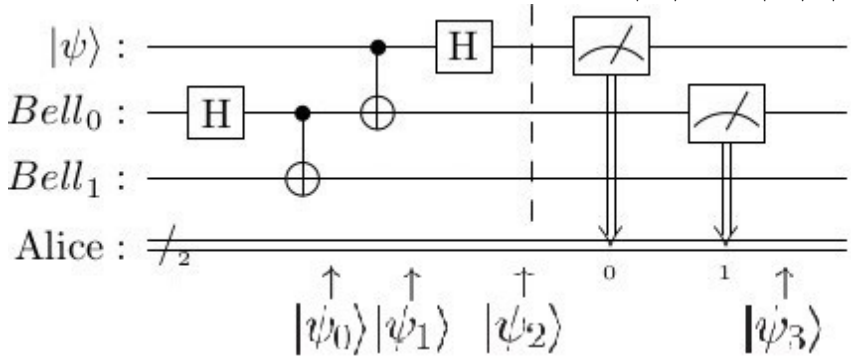
$$10 \longmapsto |\psi_3\rangle \equiv [\alpha|0\rangle - \beta|1\rangle]. \tag{29}$$

Se a medição de Alice for 11, então

$$11 \longmapsto |\psi_3\rangle \equiv [\alpha|1\rangle - \beta|0\rangle]. \tag{30}$$

A Figura 10 é o circuito até o momento:

Figura 10 – Circuito do teleporte quântico evoluindo de $|\psi_0\rangle$ para $|\psi_1\rangle$, $|\psi_2\rangle$ e $|\psi_3\rangle$.



Fonte: Autor

Alice entra em contato com Bob classicamente (internet, carta, etc) e informa o resultado obtido. Com a informação da Alice, Bob efetuará as operações necessárias no seu qubit (azul) para reproduzir o estado $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$.

No caso de Bob receber 00, $|\psi_3\rangle = \alpha|0\rangle + \beta|1\rangle$ é igual ao estado inicial de $|\psi\rangle$. Bob sabe que não é necessário realizar nenhuma operação $|\psi_4\rangle = |\psi\rangle$.

No caso de Bob receber 01, $|\psi_3\rangle = \alpha|1\rangle + \beta|0\rangle$ é diferente do estado inicial $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Nesse caso Bob sabe que deve aplicar uma porta X no seu qubit. Esta porta tem como característica inverter o estado:

$$|\psi_4\rangle = X|\psi_3\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \tag{31}$$

Assim Bob recupera o estado $|\psi\rangle$.

No caso de Bob receber 10, $|\psi_3\rangle = \alpha|0\rangle - \beta|1\rangle$ é diferente do estado inicial $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Nesse caso, Bob deve aplicar uma porta Z no seu qubit. Esta tem como característica inverter o sinal do segundo termo do estado:

$$|\psi_4\rangle = Z|\psi_3\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ -\beta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \tag{32}$$

Assim Bob recupera o estado $|\psi\rangle$.

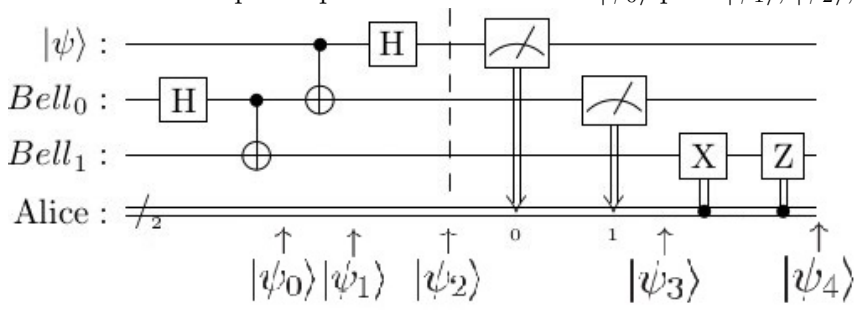
No caso de Bob receber 11, $|\psi_3\rangle = \alpha|1\rangle - \beta|0\rangle$ é diferente do estado inicial $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Nesse caso Bob deve aplicar uma porta X e uma porta Z no seu qubit[3]:

$$|\psi_4\rangle = ZX|\psi_3\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -\beta \\ \alpha \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \tag{33}$$

Assim Bob recupera o estado $|\psi\rangle$.

As operações que Bob deve realizar são condicionais. Se a informação clássica enviada por Alice tiver algarismo 1 no segundo bit (01 ou 11), Bob deve aplicar a porta X. Se a informação enviada por Alice tiver algarismo 1 no primeiro bit (10 ou 11), Bob deve aplicar uma porta Z. O circuito total do teleporte quântico é:

Figura 11 – Circuito do teleporte quântico evoluindo de $|\psi_0\rangle$ para $|\psi_1\rangle$, $|\psi_2\rangle$, $|\psi_3\rangle$ e $|\psi_4\rangle$.



Fonte: Autor

$|\psi_4\rangle$ é o estado teleportado, igual ao estado original ψ. Em nenhum momento Alice transmitiu por meios clássicos o estado original.

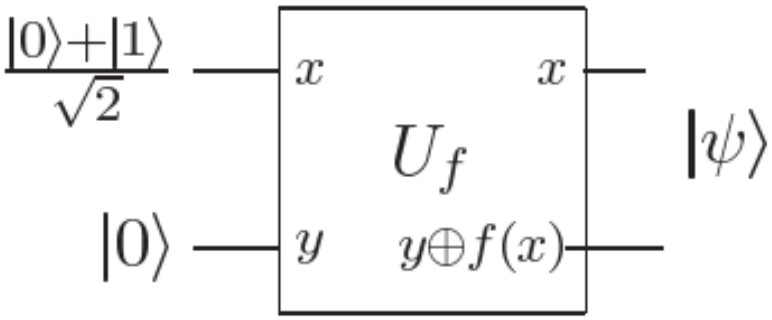
2.4.2 Paralelismo quântico.

Para podermos discutir o algoritmo de Deutsch's, primeiro é necessário comentar o paralelismo quântico. Esta é uma ferramenta fundamental em vários algoritmos: utilizando o paralelismo, um computador quântico pode processar uma função binária $f(x)$ para múltiplos valores de x ao mesmo tempo. Seja $f(x)$ uma função que tem como entrada e saída números binários: $\{0,1\} \to \{0,1\}$. Qualquer que seja o procedimento indicado

[3] a ordem das portas pode ser invertida

por $f(x)$, esta operação pode ser considerada como um sistema de dois qubits de entrada no estado $|x, y\rangle$, aplicadas em uma sequência correta de portas lógicas que transforma o estado $|x, y\rangle$ em $|x, y \oplus f(x)\rangle$. Denominaremos este processo $|x, y\rangle \to |x, y \oplus f(x)\rangle$ como U_f.

Figura 12 – Circuito quântico aplicando o paralelismo quântico.



Fonte: [12], Capítulo 1 e página 31.

A Figura 12 mostra o estado $|+\rangle$ no qubit de entrada x. O estado de saída $|\psi\rangle$ ficará em superposição e poderá ser descrito como:

$$|\psi\rangle = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}. \tag{34}$$

Este processo é notável, pois consegue obter informações de $f(0)$ e $f(1)$ ao mesmo tempo e é conhecido como *paralelismo quântico*. A superposição de estados físicos é uma característica exclusiva da Mecânica Quântica é já foi mencionada durante a descrição da porta Hadamard. Se temos mais qubits de entrada basta usar mais portas. No caso em que $n = 2$ temos que o estado de entrada x seria:

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}. \tag{35}$$

Para n qubits de entrada é preciso n portas Hadamard. Escrevemos $H\otimes^n$ para demonstrar a atuação conjunta de todas as portas Hadamard simultaneamente. Ao efetuar a transformação em qubits no estado $|0\rangle$ o resultado é dado por:

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle. \tag{36}$$

A entrada x agora é uma superposição de 2^n estados. Durante o procedimento U_f, as interferências dos estados entre si avaliam todos os qubits de entrada de uma vez só. Portanto, se em $n+1$ qubits no estado $|0\rangle^{\otimes n} |0\rangle$ for aplicada a transformação de Hadamard nos n primeiros qubits e após isso for aplicado U_f o estado de saída será:

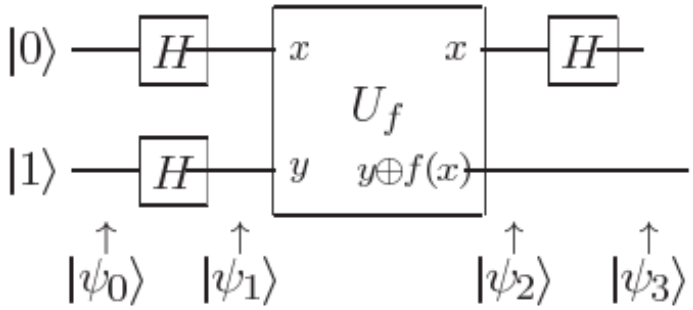
$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle|f(x)\rangle. \tag{37}$$

ou seja, o paralelismo quântico permite que todos os possíveis valores de $f(x)$ sejam avaliados simultaneamente em apenas uma iteração. Para um computador clássico efetuar esta mesma tarefa para um número pequeno de bits é simples. No entanto, quantos mais bits forem adicionados, mais difícil será realizar essa tarefa classicamente.

2.4.3 Algoritmo de Deutsch.

O algoritmo de Deutsch se utiliza do paralelismo quântico e de uma propriedade conhecida como interferência. Na Figura 13, está exposto o circuito utilizado no algoritmo de Deutsch. Analisando a figura vemos que, temos um sistema de dois qubits e em ambos estão atuando uma porta Hadamard e depois o já determinado U_f e por fim uma última porta Hadamard no primeiro qubit.

Figura 13 – Circuito quântico aplicando o algoritmo de Deutsch.



Fonte: [12], Capítulo 1 e página 33.

Sendo assim, $|\psi_0\rangle$ é o estado inicial, $|\psi_1\rangle$ é estado após a atuação das portas Hadamard, $|\psi_2\rangle$ é o estado após a atuação de U_f e $|\psi_3\rangle$ é o estado após a atuação da última porta Hadamard. Estes estados são descritos na forma de:

$$|\psi_0\rangle = |01\rangle \tag{38}$$

$$|\psi_1\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{39}$$

Logo, aplicando U_f no estado $|x\rangle\,(|0\rangle - |1\rangle)/\sqrt{2}$ obtemos um estado $(-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)/\sqrt{2}$ esboçado na Equação 40:

$$|\psi_2\rangle = \begin{cases} \pm\left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{se } f(0) = f(1) \\ \pm\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] & \text{se } f(0) \neq f(1). \end{cases} \tag{40}$$

Algo importante de ser mencionado é que, após a aplicação de U_f o estado do qubit superior mudou enquanto e estado do qubit inferior se manteve o mesmo, esse fenômeno é conhecido como retorno de fase. Após a atuação da última porta Hadamard obtemos ψ_3:

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{se } f(0) = f(1) \\ \pm|1\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{se } f(0) \neq f(1). \end{cases} \tag{41}$$

Como $f(0) \oplus f(1)$ é 0 se $f(0) = f(1)$ e 1 caso contrário, temos:

$$|\psi_3\rangle = \pm|f(0) \oplus f(1)\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{42}$$

Isso mostra a capacidade de um circuito quântico, com apenas uma interação, conseguir extrair efetivamente informação de $f(0) \oplus f(1)$, a nossa função $f(x)$. Classicamente, quando lidamos com este tipo de caso não poderíamos interagir com ambos estados 0 e 1 ao mesmo tempo, devido o caráter excludente dos bits. A natureza da mecânica quântica nos permitiu obter informação com menos interações e mais eficiência.

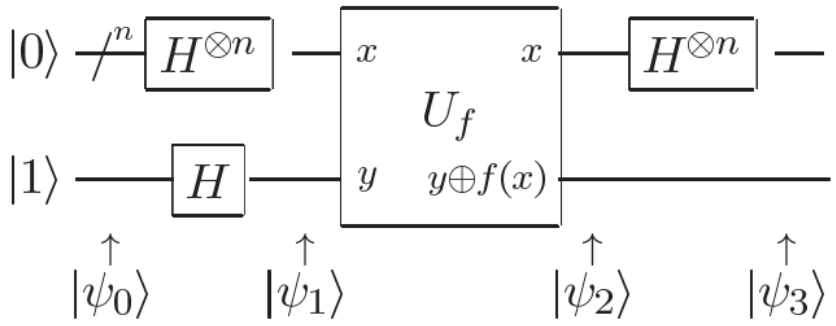
2.4.4 Algoritmo de Deutsch-Josza.

Semelhante ao teleporte quântico para o entendimento do algoritmo de Deutsch-Josza vamos propor um cenário conhecido como Deutsch's Problem: Alice, na Europa, escolhe um número x entre 0 e $2^n - 1$ e envia para Bob, no Brasil. Bob utiliza uma função $f(x)$, cujos resultados possíveis são 0 e 1. Bob disse a Alice que usaria uma função constante para todos os valores de x, ou usaria uma função balanceada, isto é, uma função que seja 0 para metade dos valores de x e 1 para a outra metade. O objetivo de Alice é determinar com exatidão que tipo de função Bob utilizou o mais rápido possível.

Se este problema fosse ser abordado classicamente Alice enviaria apenas um valor de x por vez para Bob, necessitando de uma grande quantidade de envios para finalmente entender o comportamento da função utilizada por Bob. Contanto, se Alice e Bob são capazes de trocar qubits e Bob utilizar U_f, Alice pode conseguir determinar a natureza da função em apenas um envio.

Em paralelo com o algoritmo de Deutsch, Alice tem n qubits para armazenar os possíveis valores de x e um qubit compartilhado com Bob, o qual, armazenará a resposta. Sendo assim, Alice primeiro coloca todos seus qubits em superposição, Bob se utilizando do paralelismo quântico determina $f(x)$ e deixa a resposta armazenada em um qubit. Seguindo o processo, Alice interfere nos estados dos qubits utilizando uma transformação de Hadamard em todos os qubits, exceto naquele que a resposta está armazenada. Desta forma, Alice mede os qubits e determina se a função $f(x)$ é constante ou balanceada. Este processo pode ser visto na Figura 14:

Figura 14 – Circuito quântico aplicando o algoritmo de Deutsch-Josza.



Fonte: [12], Capítulo 1 e página 35.

Onde $|\psi_0\rangle$ é o estado inicial, $|\psi_1\rangle$ o estado após passar pelas portas Hadamard, $|\psi_2\rangle$ o estado após a aplicação de U_f e $|\psi_3\rangle$ o estado após passar pelas últimas portas Hadamard. Estes estados são descritos como:

$$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle. \tag{43}$$

$$|\psi_1\rangle = \sum_{x\in\{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{44}$$

$$|\psi_2\rangle = \sum_{x} \frac{(-1)^{f(x)}|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{45}$$

Após $|\psi_2\rangle$ a transformação de Hadamard ja foi efetuada, mas para entendê-la olhemos como ela afeta um estado $|x\rangle$, ou seja, se checarmos pelos casos em que $x = 0$ e $x = 1$ vemos que para um único qubit temos:

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \tag{46}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle. \tag{47}$$

Combinando a Equação 46 e 47, obtemos:

$$H|x\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}(-1)^x|1\rangle. \tag{48}$$

$$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{z\in\{0,1\}} (-1)^{xz}|z\rangle. \tag{49}$$

Logo:

$$|\psi_3\rangle = \sum_z \sum_x \frac{(-1)^{x\cdot z + f(x)}|z\rangle}{2^n} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{50}$$

Após realizar estes processos Alice olha para os registros de todos os qubits, notando que, a amplitude de $|0\rangle^{\otimes n}$ é 1 ou -1 dependendo de $f(x)$, sendo assim, ao olharmos $|\psi_3\rangle$ é unitário, então, a amplitude de todos os qubits devem ser 0. Portanto, se $f(x)$ é balanceada, logo as amplitudes positivas e negativas se cancelam e a medição deve ser 0 em pelo menos um qubit, portanto, se Alice medir apenas zeros a função é constante, se ela medir 1 em pelo menos um qubit a função é balanceada.

3 Implementação no Qiskit

A densidade de informação retida na computação quântica é de fato impressionante, e pensar em como efetivar cada porta lógica, circuito e algoritmo é bem desafiador. O Qiskit é uma ferramenta útil para que a teoria da computação quântica seja colocada em prática. O Qiskit é uma biblioteca de desenvolvimento computacional fornecida pela IBM (International Business Machines Corporation) que permite um ambiente para efetuar a programação de computadores quânticos e será usada amplamente neste capítulo. Primeiramente, alguns conceitos básicos apresentados anteriormente serão vistos novamente, porém agora usando o Qiskit.

O Qiskit pode ser utilizado em qualquer ambiente que suporte a linguagem de programação Python, o processo para a instalação do Qiskit pode ser visto em [15]. Os códigos utilizados podem ser vistos de forma integral no Anexo 1.

3.1 Portas lógicas e circuitos quânticos implementados no Qiskit.

Para realizar qualquer processo no Qiskit primeiro é necessário usar a biblioteca. Com isso, seremos capazes de criar nossos próprios circuitos. Desta forma, olhemos para o circuito para criação de estados de Bell apresentado na Figura 7. Primeiro, importaremos um comando que nos permitirá criar um circuito quântico:

```
1 from qiskit import QuantumCircuit
```

Após isso, criaremos nosso circuito com dois qubits, utilizando o comando "QuantumCircuit(2)" onde 2 representa o número de qubits. Adicionaremos uma porta Hadamard e uma porta CNOT, Para isso, utilizamos "qc" justamente para utilizar o circuito criado.

Além disso, o termo por completo "qc.h(0)" e "qc.cnot(0,1)" tem mesma estrutura, contudo CNOT é uma porta controlada. Em "qc.h(0)" o "h" tem a função de determinar a porta lógica a ser usada e entre os parenteses o qubit em que atuará, em "qc.cnot(0,1)" o "cnot" determina a porta lógica e entre seus parenteses o primeiro número representa o qubit de controle e o segundo o qubit controlado. Por fim, uma última observação, Nota-se que os qubits que "qc.h(0)" e "qc.cnot(0,1)" estão atuando tem suas posições determinadas por 0 e 1, isto porque no código a contagem de qubits começa pela posição 0. Isto dito, o código é:

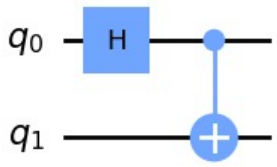
```
1 qc= QuantumCircuit(2)
2 qc.h(0)
3 qc.cnot(0,1)
```

Por fim, usaremos o comando "qc.draw()" para obtermos o desenho do circuito programado.

```
qc.draw()
```

Ao executar esse código receberemos o seguinte resultado:

Figura 15 – Circuito para Criação de Estados de Bell implementado no Qiskit.



Fonte: Autor

3.2 Teleporte quântico implementado no Qiskit

Para conseguirmos implementar o teleporte quântico no Qiskit primeiro é necessário importar toda a biblioteca que vai ser utilizada:

```
from qiskit.circuit import QuantumCircuit
#comando para a cria  o de um circuito quantico
from qiskit.primitives import Sampler
# comando para efetuar as diversas simulacoes
from qiskit.visualization import plot_histogram
# comando para vizualiazacao de um histograma
from qiskit.circuit import ClassicalRegister, QuantumRegister
#comandos para criar registros classicos e quanticos
from qiskit_aer.primitives import Sampler
# comando para efetuar simulacao
from qiskit.circuit import Parameter
# comando para criacao de um parametro
from qiskit.result import marginal_counts
#comando para marginalizar dados
import numpy as np
# comandos para funcoes matematicas
import matplotlib.pyplot as plt
# comando para criar graficos
```

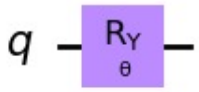
Sendo assim, conforme a teoria apresentada na seção 2.4.1 vamos criar a primeira parte do teleporte quântico, um estado ligado entre Alice e Bob, contudo, para futuramente conseguirmos distinguir o estado original e o estado teleportado criemos o parâmetro θ e também o estado inicial.

Adicionaremos agora o comando “QuantumRegister(1, ’q’)” para criar um nome para nosso qubit do estado inicial e indicar o número de qubits. Em seguida criamos o circuito quântico e adicionamos “qc.ry(theta, 0)” para no futuro nos basearmos nos resultados desse parâmetro. Por fim, executamos o comando “qc.measure_all()” para medir e armazenar os resultados do circuito “qc” e o comando “qc.draw(’mpl’)” para obtermos o circuito feito no estilo determinado “mpl”.

```
theta = Parameter('$\theta$')

qr = QuantumRegister(1, 'q')
qc = QuantumCircuit(qr)
qc.ry(theta, 0)
qc.measure_all()
qc.draw('mpl')
```

O resultado obtido é visto na Figura 16

Figura 16 – Porta lógica parâmetro no circuito quântico para a simulação do Teleporte Quântico.



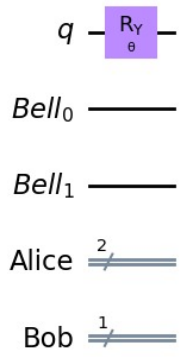
Fonte: Autor

Após isso faremos o circuito para efetuar o teleporte quântico, criamos primeiramente os qubits de Alice e Bob no Estado de Bell e os registros clássicos para armazenar as medidas. As estruturas desses comando seguem a mesma lógica do “QuantumRegister(1, ’q’)”:

```
qr = QuantumRegister(1, 'q')
qct = QuantumCircuit(qr)
qct.ry(theta, 0)
bell = QuantumRegister(2, 'Bell')
alice = ClassicalRegister(2, 'Alice')
bob = ClassicalRegister(1, 'Bob')
qct.add_register(bell, alice, bob)
qct.draw('mpl')
```

O resultado obtido é visto na Figura 17:

Figura 17 – Circuito do Teleporte quântico com os qubits q, $Bell_0$, $Bell_1$ e o parâmetro R_Y implementado no qiskit.



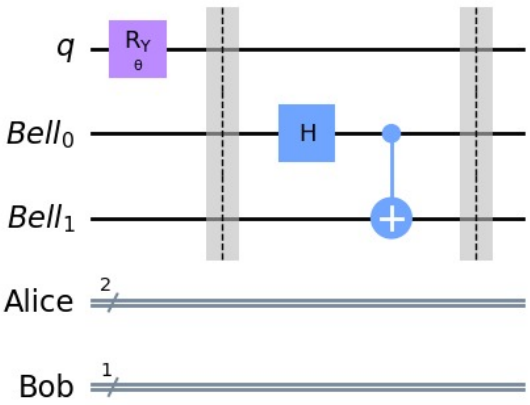
Fonte: Autor

Após isso, criamos os estados ligados entre os qubits de Alice e Bob. o comando "qct.barrier()" é um recurso para segmentar o circuito, o deixando mais organizado, sendo mais fácil de entender o papel de cada parte:

```
qct.barrier()
qct.h(1)
qct.cx(1, 2)
qct.barrier()
qct.draw('mpl')
```

O resultado obtido é visto na Figura 18:

Figura 18 – Circuito do Teleporte quântico com os qubits q, $Bell_0$, $Bell_1$, o parâmetro R_Y, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits $Bell_0$ e $Bell_1$ separados por barreiras implementado no qiskit.



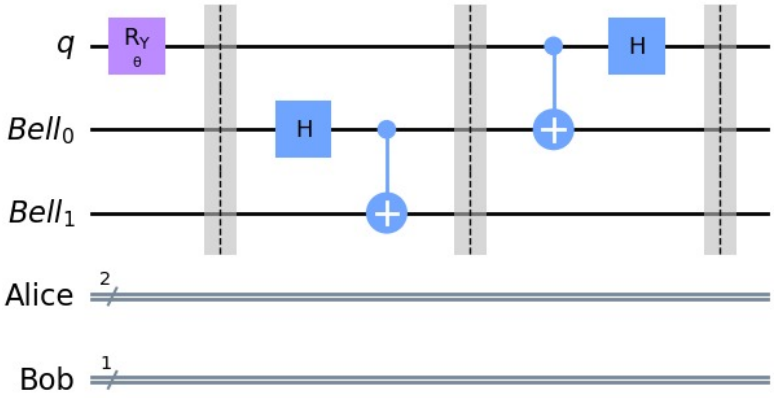
Fonte: Autor

Adiante, criamos uma porta CNOT com controle em q, para mapear suas informações, então, aplicamos uma Porta Hadamard em q:

```
qct.cx(0, 1)
qct.h(0)
qct.barrier()
qct.draw('mpl')
```

O resultado obtido pode ser visto na Figura 19:

Figura 19 – Circuito do Teleporte quântico com os qubits q, $Bell_0$, $Bell_1$, o parâmetro R_Y, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits $Bell_0$ e $Bell_1$, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits q e $Bell_0$ separados por barreiras implementado no qiskit.



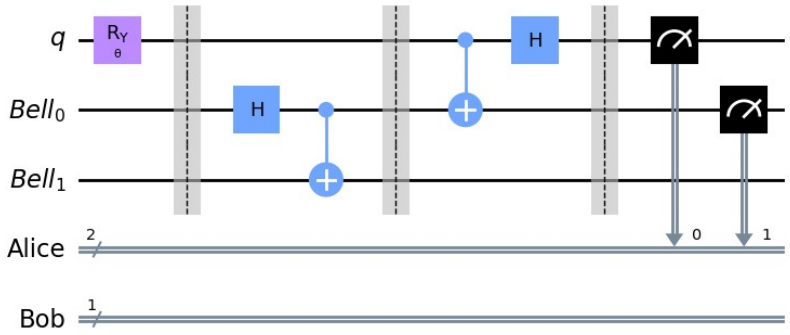
Fonte: Autor

O processo efetuado até aqui é o mesmo feito nas equações 22, 23, 24, 25 e 26. Por fim, Alice mede seus qubits e os registra. o comando "qct.measure([qr[0], bell[0]], alice)" mede o "QuantumRegister" do circuito e afere o estado colapsado do qubit, as primeiras duas posições determinam a posição dos qubits a serem medidos e a terceira, em que registro clássico deve ser armazenado. Nesse caso o de Alice para ela poder transmitir as informações das medidas para Bob na próxima etapa:

```
qct.measure([qr[0], bell[0]], alice)
qct.draw('mpl')
```

O resultado pode ser visto na Figura 20:

Figura 20 – Circuito do Teleporte quântico com os qubits q, $Bell_0$, $Bell_1$, o parâmetro R_Y, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits $Bell_0$ e $Bell_1$, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits q e $Bell_0$, duas medições dos qubits q e $Bell_0$ separados por barreiras, implementado no qiskit.



Fonte: Autor

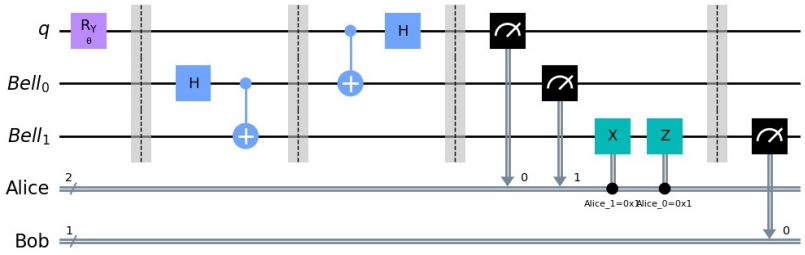
Neste momento, Alice enviaria classicamente as informações para Bob, que dependendo dos resultados de Alice, performará as operações necessárias para obter o qubit original- ou estado inicial- e por fim medi-las. Para esta parte a programação se torna de suma importância, pois podemos usar portas condicionadas. Ou seja, as medições feitas influenciam em quais portas são postas, estas representando as modificações que Bob deve utilizar para receber o estado teleportado igual ao estado inicial.

Para isso usamos "qct.x(bell[1]).c_if(alice[1],1)" e "qct.z(bell[1]).c_if(alice[0],1)" que representam em sua estrutura a condicionalidade. O termo "qct.x(bell[1])..." indica a aplicação de uma porta lógica X no qubit nomeado "bell[1]" e o termo "...c_if(alice[1],1)" indica que se o registro clássico "Alice[1]" medir 1 a porta logica X deve ser aplicada, do contrário esse comando é ignorado, ou seja, o caso em que "Alice[1]" medir 0. A mesma lógica se aplica a "qct.z(bell[1]).c_if(alice[0],1)" onde a porta Z será a aplicada dependendo do valor obtido na medição:

```
1  qct.x(bell[1]).c_if(alice[1],1)
2  qct.z(bell[1]).c_if(alice[0],1)
3  qct.barrier()
4  qct.measure(bell[1], bob)
5  qct.draw('mpl')
```

O resultado obtido pode ser visto na Figura 21:

Figura 21 – Circuito do Teleporte quântico com os qubits q, $Bell_0$, $Bell_1$, o parâmetro R_Y, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits $Bell_0$ e $Bell_1$, uma porta Hadamard e uma porta CNOT gerando emaranhamento entre os qubits q e $Bell_0$, duas medições dos qubits q e $Bell_0$, as portas condicionais X, Z e a medição do qubit $Bell_1$ separados por barreiras, implementado no qiskit.



Fonte: Autor

Neste ponto, o processo realizado é o mesmo das Equações 27, 28, 29 e 30 e as Equações 31, 32 e 33, demonstram como cada medição influencia nas portas lógicas X e Z condicionalmente aplicadas.

Com o circuito montado podemos efetuar sua simulação. Para isso, usaremos o nosso parâmetro θ como $5\pi/7$ e usamos os comandos "Sampler()" e "sampler . run (qc . assign_parameters (theta : angle))" para executar o circuito inicial de Alice e o teleportado para Bob.

Por fim, o comando "sampler . run (qc . assign_parameters (theta : angle))" tem em sua função relacionar o valor do parâmetro "theta" ao ângulo de fase do qubit o que impacta diretamente em sua medida. Consequentemente, o teleporte quântico é checado por este parâmetro, garantindo que estado inicial e teleportado eram iguais:

```
angle = 5*np.pi/7

sampler = Sampler()
job_static = sampler.run(qc.assign_parameters({theta: angle}))
job_dynamic = sampler.run(qct.assign_parameters({theta: angle}))
```

Com isso, temos os resultados de todos os qubits que foram medidos e como só estamos interessados nas medições do qubit de Alice, usaremos o comando "marginal_counts (job_dynamic . result () . quasi_dists [0]. binary_probabilities () , indices =[2] , format_marginal = False , marginalize_ memory = False)" para conseguirmos extrair apenas os dados necessários.

```
tele_counts = marginal_counts(job_dynamic.result().quasi_dists[0].
 binary_probabilities(), indices=[2], format_marginal = False ,
 marginalize_memory = False )
```

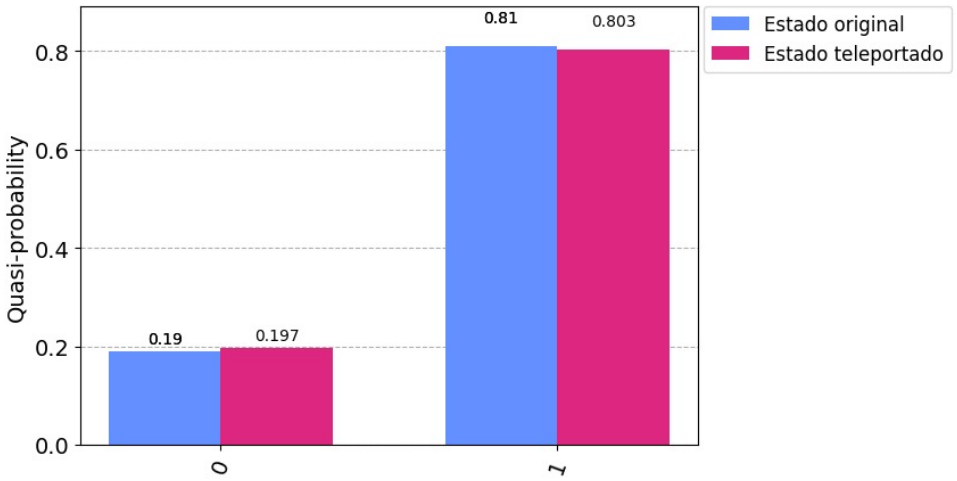
Por fim, usaremos o comando "plot_histogram([job_static.result().quasi_dists[0].binary_probabilities(),tele_counts], legend=legend)" para ver a recorrência das medidas de

Alice e Bob, se tivermos marginalizados os dados corretamente veremos que a distribuição de probabilidade do estado original e teleportado são praticamente iguais:

```
legend = ['Estado original', 'Estado teleportado']
plot_histogram([job_static.result().quasi_dists[0].binary_probabilities
    (), tele_counts], legend=legend)
```

O resultado obtido pode ser visto na Figura 22:

Figura 22 – Distribuição de quasi-probabilidade do estado original e teleportado.



Fonte: Autor

Em suma, a implementação do teleporte quântico no Qiskit demonstra como os estados de Bell e a natureza dos qubits fornecem uma poderosa ferramenta para a realização de uma tarefa que classicamente seria inviável.

3.3 Algoritmo de Deutsch implementado no Qiskit.

O código estudado foi retirado de [16]. Para implementarmos o algoritmo de Deutsch no Qiskit devemos primeiro importar os comandos necessários da biblioteca:

```
from qiskit import QuantumCircuit
# Comando para a criacao de um circuito quantico
from qiskit_aer import AerSimulator
#comando para efetuar diversas simulacoes
```

Seguindo a teoria apresentada na seção 2.4.3 conseguimos implementar o algoritmo de Deutsch. Diferente do teleporte quântico, neste algoritmo utilizamos uma porta U_f (Figura 12). A porta lógica U_f não tem formato estabelecido. Essa nomenclatura é utilizada para representar uma porta aleatória e pode ter algumas portas lógicas simples em sua composição ou ser longa com muitas portas e qubits.

Para implementarmos a porta U_f devemos primeiro criá-la usando o comando “def funcao_deutsch(case: int)”, este define um nome para a função que programaremos e

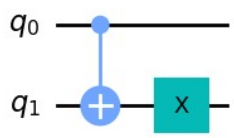
dentro de seus parenteses a variável que será utilizada-“case” e sua categoria “int”. Em seguida definimos através do comando condicional “if case not in [1, 2, 3, 4]” , checada essa condição será efetuado pelo código o próximo comando “raise ValueError("‘case‘ must be 1, 2, 3, or 4.")” definindo que, se o caso escolhido não for 1, 2, 3 ou 4 o valor será determinado novamente para um desses 4 casos.

Criamos o circuito quântico utilizando o comando “f = QuantumCircuit(2)”, o comando condicional “if case in [2, 3]” (Linha 5), checada a condição se o caso for 2 ou 3 uma porta CNOT será adicionada, o comando condicional “if case in [3, 4]” (Linha 7), checada a condição se o caso for 3 ou 4 uma porta X será adicionada. Por fim, o comando “return f” retorna a “função” resultante e o comando “funcao_deutsch(3).draw()” nos retorna a porta U_f.

```
1  def funcao_deutsch(case: int):
2    if case not in [1, 2, 3, 4]:
3      raise ValueError("‘case‘ must be 1, 2, 3, or 4.")
4    f = QuantumCircuit(2)
5      if case in [2, 3]:
6      f.cx(0, 1)
7    if case in [3, 4]:
8      f.x(1)
9    return f
10
11 funcao_deutsch(3).draw()
```

O resultado obtido é visto na Figura 23:

Figura 23 – Circuito utilizado como porta U_f para o algoritmo de Deutsch implementado no Qiskit.



Fonte: Autor

Este pequeno circuito apesar de simples nos retorna uma porta U_f para podermos implementar o algoritmo de Deutsch. Para criarmos o algoritmo usamos o comando “def compilar_ circuito(funcao: QuantumCircuit):”. definimos “n = funcao.num_qubits - 1” que representa o número de qubits, “qc = QuantumCircuit(n + 1, n)” criando nosso circuito quântico com “n+1” qubits e . Usando diversos comandos adicionamos uma sequência de portas lógicas, já vistas antes. Os comandos principais e que ainda não foram utilizados são “qc.h(range(n + 1))”, “qc.compose(funcao, inplace=True)” e “qc.measure(range(n), range(n))”.

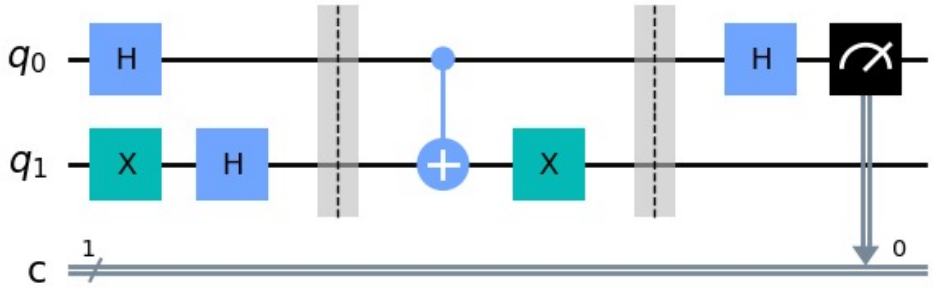
O comando “qc.h(range(n + 1))” tem um funcionamento simples, determinando e “...(range(n + 1))” determina a posição em que a porta será aplicada. O comando “range(n + 1)” determina que a porta foi aplicada em todas as posições dentro do alcance do qubits. Ou seja, cada qubit com exceção do primeiro terá uma porta Hadamard aplicada. Esta é a Transformação de Hadamard onde são aplicados “n” portas Hadamard.

O comando “qc.compose(funcao, inplace=True)” tem um funcionamento simples, utilizar da função de Deutsch criada anteriormente e adicionar no algoritmo. “qc.measure(range(n), range(n))” mede os qubits em todas as “n” posições e armazena os resultados nos registros clássicos.

```
def compilar_circuito(funcao: QuantumCircuit):

  n = funcao.num_qubits - 1
  qc = QuantumCircuit(n + 1, n)

  qc.x(n)
  qc.h(range(n + 1))

  qc.barrier()
  qc.compose(funcao, inplace=True)
  qc.barrier()

  qc.h(range(n))
  qc.measure(range(n), range(n))

  return qc

compilar_circuito(funcao_deutsch(3)).draw()
```

O resultado pode se visto na Figura 24:

Figura 24 – Circuito do Algoritmo de Deutsch implementado no Qiskit.



Fonte: Autor

Por fim, podemos então determinar se a função é constante ou balanceada. O comando “result = AerSimulator().run(qc, shots=1, memory=True).result()” simula nosso cir-

cuito criado “qc” uma vez e armazena os resultados, “measurements = result.get_memory()” extrai os resultados em formato binário. O comando “if measurements[0] == "0"” checa se os resultados medidos foram 0, seguida essa condição ele retornará se a função é constante, caso contrário ela é balanceada. O comando “f = funcao_deutsch(3)” gera nossa porta U_f e “algoritmo_deutsch(f)” executa nosso código:

```
def algoritmo_deutsch(funcao: QuantumCircuit):

  qc = compilar_circuito(funcao)

  result = AerSimulator().run(qc, shots=1, memory=True).result()
  measurements = result.get_memory()
  if measurements[0] == "0":
    return "constante"
  return "balanceada"

f = funcao_deutsch(3)
algoritmo_deutsch(f)
```

O resultado final é:

```
  'balanceada'
```

O algoritmo de Deutsch usa do paralelismo quântico e com uma iteração consegue determinar se a função é constante ou balanceada retirando informação de um único qubit. Ou seja, ao implementarmos o algoritmo de Deutsch no Qiskit, mostramos que, o paralelismo quântico nos permite lidar e processar de maneira mais eficiente que qualquer computador clássico. O algoritmo de Deutsch, apesar de mostrar a capacidade dos computadores quânticos, tem a sua função $f(x)$ muito simples, contudo, o algoritmo de Deutsch-josza invés de interagir com 2 qubits tem um número maior de qubits interagindo.

3.4 Algoritmo de Deutsch-Josza implementado no Qiskit.

O código estudado foi retirado de [16]. A implementação do algoritmo de Deutsch-Josza se torna um pouco mais complicada devido a porta U_f que para este caso é mais complicada de ser programada, Para isso, importaremos as bibliotecas necessárias:

```
from qiskit import QuantumCircuit
# Comando para a criacao de um circuito quantico
from qiskit_aer import AerSimulator
#comando para efetuar diversas simulacoes
import numpy as np
# comandos para funcoes matematicas
```

Para definirmos a porta U_f precisamos conseguir criar uma função que tenha 50% de chance de ser balanceada e 50% de chance de ser constante. Para isso, usaremos o Qiskit para criar uma função condizente e aleatória.

Usando o comando "def funcao_DJ(num_qubits)" criamos uma função que será responsável por fornecer uma porta U_f completamente aleatória. O comando da linha 2 "qc = QuantumCircuit(num_qubits + 1)" cria nosso circuito quântico, nas linhas 3 e 5 usamos o comando "if np.random.randint(0, 2)" para gerar uma chance de 50% de aplicar uma porta lógica X nos qubits e 50% de os manterem iguais. O comando "return qc" retorna nossa "função".

```
1 def funcao_DJ(num_qubits):
2   qc = QuantumCircuit(num_qubits + 1)
3   if np.random.randint(0, 2):
4     qc.x(num_qubits)
5   if np.random.randint(0, 2):
6     return qc
```

Na linha 3 do código usando a biblioteca "numpy" aleatoriamente invertemos o estado de 50% dos qubits aplicando a porta X e na linha 5 50% dos qubits se mantém iguais. Agora, escolheremos metade das possíveis entradas do circuito sem que elas se repitam:

```
1 on_states = np.random.choice(range(2**num_qubits),  2**num_qubits // 2,
      replace=False)
```

Nota-se que, "range(2**num_qubits)" gera um intervalo de 0 a $2^n - 1$ que representa todos os possíveis estados, "2**num_qubits // 2" determina quantos estados serão selecionados aleatoriamente e "replace=False" garante a não repetição de nenhum estado já selecionado. Agora, aplicaremos portas X para inverter os estados dos qubits após a primeira parte, apesar de repetitivo, quanto mais os estados dos qubits forem invertidos mais a porta U_f será complicada:

```
1 def add_cx(qc, bit_string):
2   for qubit, bit in enumerate(reversed(bit_string)):
3   if bit == "1":
4     qc.x(qubit)
5   return qc
```

O comando "for qubit, bit in enumerate(reversed(bit_string))" vai criar um par (qubit, valor_do_bit) que servira para o comando "if bit == "1"" consiga determinar em que qubits serão adicionados as portas X, ou seja, quando o valor do bit for 1 uma porta X sera adicionado no seu respectivo qubit, a qual, é determinado por (qubit, valor_do_bit). Por fim, serão adicionadas portas CNOT e barreiras no circuito:

```
1 for state in on_states:
2   qc.barrier()
```

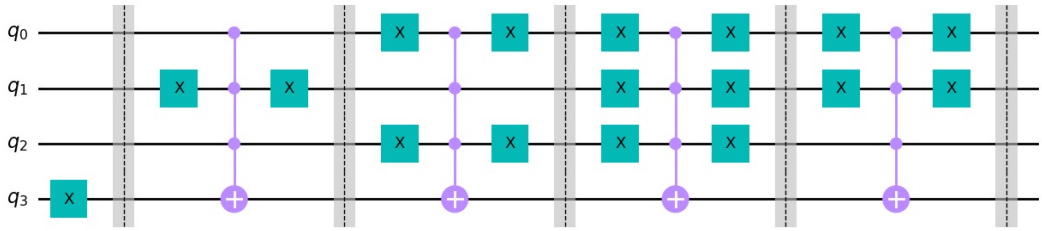
```
    qc = add_cx(qc, f"{state:0b}")
    qc.mcx(list(range(num_qubits)), num_qubits)
    qc = add_cx(qc, f"{state:0b}")

qc.barrier()

return qc
```

O comando “for state in on_states” faz com que uma interação com os estados selecionados pelo “on_states” seja efetuada adicionando barreiras, portas X, portas CNOT com diversos controles e pedimos que a função seja retornada. Usando o comando “draw”:

```
    funcao_DJ(3).draw()
```

O resultado pode ser visto na Figura 25:

Figura 25 – Circuito utilizado como porta U_f para o algoritmo de Deutsch-Josza implementado no Qiskit.



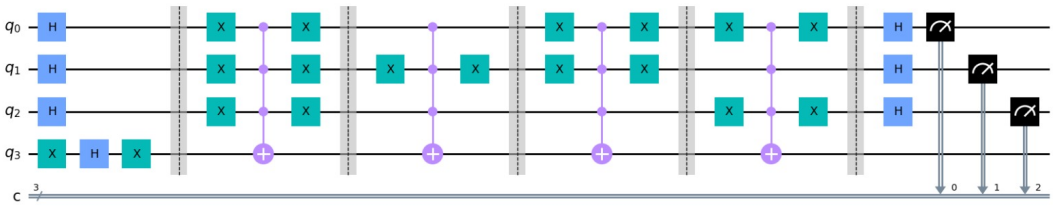
Fonte: Autor

Este resultado é um dos diversos que pode ser obtido, cada vez que esse código for executado uma porta U_f diferente será gerada. Com a nossa porta U_f determinada, podemos escrever o algoritmo Deutsch-Josza:

```
def compile_circuit(function: QuantumCircuit):
    n = function.num_qubits - 1
    qc = QuantumCircuit(n + 1, n)
    qc.x(n)
    qc.h(range(n + 1))
    qc.compose(function, inplace=True)
    qc.h(range(n))
    qc.measure(range(n), range(n))
    return qc
```

Usando o comando “draw” podemos apresentar o circuito do algoritmo de Deutsch-Josza na Figura 26:

Figura 26 – Circuito do Algoritmo de Deutsch-Josza implementado no Qiskit.



Fonte: Autor

Por fim, implementando o algoritmo de Deutsch-Josza:

```
def dj_algoritmo(function: QuantumCircuit):
  qc = compile_circuit(function)

  result = AerSimulator().run(qc, shots=1, memory=True).result()
  measurements = result.get_memory()
  if "1" in measurements[0]:
    return "balanceada"
  return "constante"
```

O algoritmo de Deutsch-Josza é extremamente eficiente: um computador clássico teria que realizar iterações na ordem de $2^n - 1/2$ para conseguir determinar se uma função é balanceada ou constante. Esse ganho de eficiência só ocorre devido ao paralelismo quântico que, mais uma vez, se mostra uma ferramenta extremamente útil para diversos algoritmos quânticos.

4 Considerações Finais.

Em suma, vimos que a mecânica quântica forneceu as ferramentas necessárias para o estudo dos conceitos básicos da computação quântica, evoluindo de portas lógicas simples para algoritmos capazes de realizar operações lógicas extremamente eficientes. Por diversas vezes algoritmos quânticos simples, ultrapassaram a eficiência de algoritmos clássicos, mostrando a versatilidade e a utilidade da computação quântica.

Os conceitos básicos mostram mais uma vez que a computação quântica é uma área de pesquisa em evolução com imenso potencial teórico e prático. O Qiskit se mostrou uma ferramenta extremamente útil que capacitou aplicar os conceitos estudados de modo intuitivo e eficaz.

Anexo 1 - Códigos completos

Neste anexo estão contidos os códigos completos feitos e utilizados.

Código 1: Circuito para criação dos estados de Bell.

```
from qiskit import QuantumCircuit

qc= QuantumCircuit(2)

qc.h(0)
qc.cnot(0,1)

qc.draw()

```

Código 2: Teleporte quântico.

```
from qiskit.circuit import QuantumCircuit
from qiskit.primitives import Estimator, Sampler
from qiskit.visualization import plot_histogram
from qiskit.circuit import ClassicalRegister, QuantumRegister
from qiskit_aer.primitives import Sampler
from qiskit.circuit import Parameter
from qiskit.result import marginal_counts

import numpy as np
import matplotlib.pyplot as plt

theta = Parameter('')

qr = QuantumRegister(1, 'q')
qc = QuantumCircuit(qr)
qc.ry(theta, 0)
qc.measure_all()

qc.draw('mpl')

qr = QuantumRegister(1, 'q')
qct = QuantumCircuit(qr)
qct.ry(theta, 0)
bell = QuantumRegister(2, 'Bell')
alice = ClassicalRegister(2, 'Alice')
bob = ClassicalRegister(1, 'Bob')
qct.add_register(bell, alice, bob)
qct.barrier()
qct.h(1)
```

```
qct.cx(1, 2)
qct.barrier()
qct.cx(0, 1)
qct.h(0)
qct.barrier()
qct.measure([qr[0], bell[0]], alice)
qct.x(bell[1]).c_if(alice[1],1)
qct.z(bell[1]).c_if(alice[0],1)
qct.barrier()
qct.measure(bell[1], bob)

qct.draw('mpl')

angle = 5*np.pi/7

sampler = Sampler()
job_static = sampler.run(qc.assign_parameters({theta: angle}))
job_dynamic = sampler.run(qct.assign_parameters({theta: angle}))

tele_counts = marginal_counts(job_dynamic.result().quasi_dists[0].
    binary_probabilities(), indices=[2], format_marginal = False ,
    marginalize_memory = False )

legend = ['Estado original', 'Estado teleportado']
plot_histogram([job_static.result().quasi_dists[0].binary_probabilities
    (), tele_counts], legend=legend)
```

Código 3: Algoritmo de Deutsch

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
def funcao_deutsch(case: int):
  if case not in [1, 2, 3, 4]:
    raise ValueError("`case` must be 1, 2, 3, or 4.")
    f = QuantumCircuit(2)
  if case in [2, 3]:
    f.cx(0, 1)
  if case in [3, 4]:
    f.x(1)
  return f

funcao_deutsch(3).draw()

def compilar_circuito(funcao: QuantumCircuit):

  n = funcao.num_qubits - 1
  qc = QuantumCircuit(n + 1, n)

```

```
  qc.x(n)
  qc.h(range(n + 1))

  qc.barrier()
  qc.compose(funcao, inplace=True)
  qc.barrier()

  qc.h(range(n))
  qc.measure(range(n), range(n))

  return qc

compilar_circuito(funcao_deutsch(3)).draw()

def algoritmo_deutsch(funcao: QuantumCircuit):

  qc = compile_circuit(funcao)

  result = AerSimulator().run(qc, shots=1, memory=True).result()
  measurements = result.get_memory()
  if measurements[0] == "0":
    return "constante"
  return "balanceada"

f = funcao_deutsch(3)
algoritmo_deutsch(f)
```

Código 4: Algoritmo de Deutsch-Josza

```
def add_cx(qc, bit_string):
  for qubit, bit in enumerate(reversed(bit_string)):
            if bit == "1":
        qc.x(qubit)
    return qc

  for state in on_states:
            qc.barrier()
            qc = add_cx(qc, f"{state:0b}")
            qc.mcx(list(range(num_qubits)), num_qubits)
            qc = add_cx(qc, f"{state:0b}")

            qc.barrier()

    return qc

  funcao_DJ(3).draw()

  def compile_circuit(function: QuantumCircuit):
```

```
    n = function.num_qubits - 1
    qc = QuantumCircuit(n + 1, n)
    qc.x(n)
    qc.h(range(n + 1))
    qc.compose(function, inplace=True)
    qc.h(range(n))
    qc.measure(range(n), range(n))
    return qc

    def dj_algoritmo(function: QuantumCircuit):
    qc = compile_circuit(function)

    result = AerSimulator().run(qc, shots=1, memory=True).result()
    measurements = result.get_memory()
    if "1" in measurements[0]:
    return "balanceada"
    return "constante"
```

Referências

1 PLANCK, M. On the law of distribution of energy in the normal spectrum. *Annalen der physik*, v. 4, n. 553, p. 1, 1901. 7

2 EINSTEIN, A. On a heuristic point of view concerning the production and transformation of light. *Annalen der Physik*, Princeton University Press, v. 17, n. 132, p. 1–16, 1905. 7

3 BOHR, N. I. on the constitution of atoms and molecules. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 26, n. 151, p. 1–25, 1913. 7

4 HEISENBERG, W. Quantum-theoretical re-interpretation of kinematic and mechanical relations. *Z. Phys*, v. 33, p. 879–893, 1925. 7

5 BORN, M.; JORDAN, P. Zur quantenmechanik. *Zeitschrift für Physik*, Springer, v. 34, n. 1, p. 858–888, 1925. 7

6 SCHRÖDINGER, E. Quantisierung als eigenwertproblem. *Annalen der physik*, WILEY-VCH Verlag Leipzig, v. 385, n. 13, p. 437–490, 1926. 7

7 FEYNMAN, R. P. Simulating physics with computers. In: *Feynman and computation*. [S.l.]: cRc Press, 2018. p. 133–153. 7

8 DEUTSCH, D. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, The Royal Society London, v. 400, n. 1818, p. 97–117, 1985. 7

9 BENNETT, C. H.; BRASSARD, G. Quantum cryptography: Public key distribution and coin tossing. *Theoretical computer science*, Elsevier, v. 560, p. 7–11, 2014. 7

10 SHOR, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In: IEEE. *Proceedings 35th annual symposium on foundations of computer science*. [S.l.], 1994. p. 124–134. 7

11 GROVER, L. K. A fast quantum mechanical algorithm for database search. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. [S.l.: s.n.], 1996. p. 212–219. 8

12 NIELSEN, M. A.; CHUANG, I. L. *Quantum computation and quantum information*. Cambridge: Cambridge university press, 2010. 10, 13, 14, 15, 20, 21, 23

13 BELL, J. S. On the einstein podolsky rosen paradox. *Physics Physique Fizika*, APS, v. 1, n. 3, p. 195, 1964. 11

14 WONG, T. G. *Introduction to classical and quantum computing*. Nebraska: Rooted Grove Omaha, 2022. 11

15 QISKIT. *How to Install Qiskit | Coding with Qiskit 1.x | Programming on Quantum Computers*. 2024. <https://youtu.be/dZWz4Gs_BuI?si=OQgswzP14kg5la57>, Acessado em 24/08/2024. 25

16 QISKIT. *Quantum query algorithms*. 2024. <https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/quantum-query-algorithms#deutschs-algorithm>, Acessado em 24/08/2024. 32, 35