UNIVERSIDADE FEDERAL DO PARANÁ

Curso de Introdução à Computação Quântica

Leonardo Wittica Crozetta

Curitiba

2025

1 Conceitos Matemáticos Preliminares

O primeiro passo que deve-se tomar ao iniciar os estudos em computação quântica é desenvolver uma boa compreensão matemática de alguns tópicos que nos serão úteis ao decorrer do curso. Nessa primeira aula, serão abordados os seguintes tópicos:

- Introdução aos Números Complexos;
- Números Complexos no Plano Numérico;
- Introdução às Matrizes;
- Multiplicação de Matrizes;
- Matrizes Unitárias e Hermitianas;
- Autovetores e Autovalores;
- Notação de Dirac;
- Produto Tensorial;

1.1 Introdução aos Números Complexos

Inicialmente, vamos entender o que são números imaginários. Sabe-se que a solução para a seguinte equação é

$$\begin{aligned} x^2 &= 4, \\ x &= \sqrt{4}, \\ x &= \pm 2. \end{aligned} \tag{1.1}$$

Porém, quando, tenta-se resolver

$$x^2 = -4, \tag{1.2}$$

ocorre um problema. Como um número elevado ao quadrado pode resultar em um número negativo? No conjunto dos reais, isso realmente não acontece. Porém, a equação (1.2) pode ser resolvida usando números imaginários. Para isso, introduz-se a seguinte raiz quadrada $\sqrt{-1} = i$, onde i é chamado de *unidade imaginária*. Dessa forma, resolvendo a equação (1.2), obtém-se

$$\begin{aligned} x^2 &= -4, \\ x &= \sqrt{-4}, \\ x &= \sqrt{4} \cdot \sqrt{-1}, \\ x &= \sqrt{4} \cdot i, \\ x &= \pm 2i. \end{aligned} \tag{1.3}$$

Os números que apresentam a unidade imaginária i são chamados de números imaginários. Com esses conceitos claros, pode-se definir o que é um número complexo, um número que possui uma parte real e uma parte imaginária. Por exemplo,

$$z = a + b \cdot i. \tag{1.4}$$

No exemplo (1.4), "a" representa a parte real e "b" a parte imaginária. Como z possui uma parte imaginária, ele é considerado um número complexo.
Na sequência, apresenta-se algumas operações matemáticas básicas envolvendo números complexos.

1.1.1 Soma e Subtração de Números Complexos

Para realizar a soma ou subtração de números complexos, é necessário realizar operações matemáticas das partes reais e imaginárias separadamente. Por exemplo, considere os seguintes números complexos

$$z = 4 + 9i, \tag{1.5}$$
$$g = 3 + 4i. \tag{1.6}$$

A soma de z com g é feita da seguinte forma

$$\begin{aligned} g + z &= (4 + 9i) + (3 + 4i), \\ &= (4 + 3) + (9 + 4)i, \\ &= 7 + 13.i. \end{aligned} \tag{1.7}$$

1.1.2 Multiplicação de Números Complexos

Para realizar a multiplicação entre dois números complexos, deve-se fazer a distributiva de cada termo

$$\begin{aligned} gz &= (4 + 9i)(3 + 4i), \\ &= 4 \cdot 3 + 4 \cdot 4i + 9i \cdot 3 + 9i \cdot 4i, \\ &= 12 + 16i + 27i - 36, \\ &= -24 + 43i. \end{aligned}$$

Lembrando que $i \cdot i = -1$.

1.1.3 Conjugado de um Número Complexo

O conjugado de um número complexo é o próprio número com o sinal da parte imaginária invertido. Para o caso de $\overline{z}$, tem-se

$$\overline{z} = 4 - 9i.$$

Pode-se representar o complexo conjugado com uma barra sobre o número. Além disso, também é possível utilizar asterisco $(*)$ para representar o complexo conjulgado

1.1.4 Outras Formas de Representar um Número Complexo

Foi observado que é possível escrever um número complexo na forma geral

$$z = a + bi.$$

Essa é a forma algébrica. Porém, existem outras formas de representar um número complexo, como as que mostraremos a seguir.

Na forma polar, um número complexo é representado pelo seu módulo r e seu argumento θ (o ângulo que z faz com o eixo real):

$$\begin{aligned} r &= \sqrt{a^2 + b^2}, \\ z &= r(\cos(\theta) + i\sin(\theta)). \end{aligned}$$

Outra forma de representar um número complexo é utilizando a forma exponencial. Ela é baseada na fórmula de Euler

$$e^{i\varphi} = \cos(\varphi) + i\sin(\varphi).$$

Portanto, podemos escrever

$$z = re^{i\theta}. \tag{1.8}$$

1.1.5 Números Complexos no Plano Numérico

Pode-se representar números complexos como vetores no plano. A grande diferença com o conjunto dos reais é que o eixo das abscissas estará relacionado com a parte real do número complexo e o eixo das ordenadas estará relacionado com a parte imaginária. Pode-se ultilizar a linguagem de programação Python para representar números complexos graficamente. Após instalar o Python em sua máquina basta seguir o código abaixo para representar um número complexo graficamente.

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

# Definindo o numero complexo
z = 3 + 4j

# Obter as partes real e imaginaria
a = z.real
b = z.imag

# Calcular o modulo e o angulo theta
r = np.abs(z)
theta = np.angle(z)  # em radianos

# Criar a figura e o eixo
plt.figure(figsize=(6, 6))
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)

# Plotar o vetor correspondente ao numero complexo
plt.quiver(0, 0, a, b, angles='xy', scale_units='xy', scale=1, color='black')

# Adicionar o ponto final do vetor e as coordenadas
plt.plot(a, b, 'ko')  # ponto vermelho no final do vetor
plt.text(a, b, f'  ({a}, {b}i)', fontsize=12, color='black')

# Adicionar a indica  o do angulo theta mais proximo da base do vetor
angle_annotation = np.linspace(0, theta, 100)
radius = 1.5  # raio menor para a base do vetor
plt.plot(radius * np.cos(angle_annotation), radius * np.sin(angle_annotation), 'b--',
    label=r'$\theta$')

# Adicionar um texto para theta na base do vetor
plt.text(radius * np.cos(theta/2) + 0.2, radius * np.sin(theta/2) + 0.2, r'$\theta$',
    fontsize=16, color='blue')

# Ajustar limites do grafico
plt.xlim(-10, 10)
plt.ylim(-10, 10)

# Adicionar rotulos
plt.xlabel('Parte Real')
plt.ylabel('Parte Imagin ria')

# Grid para melhor visualizacaoo
plt.grid(True)

# Salvar a figura como imagem
plt.savefig('numero_complexo.png')
plt.show()
plt.close()
```

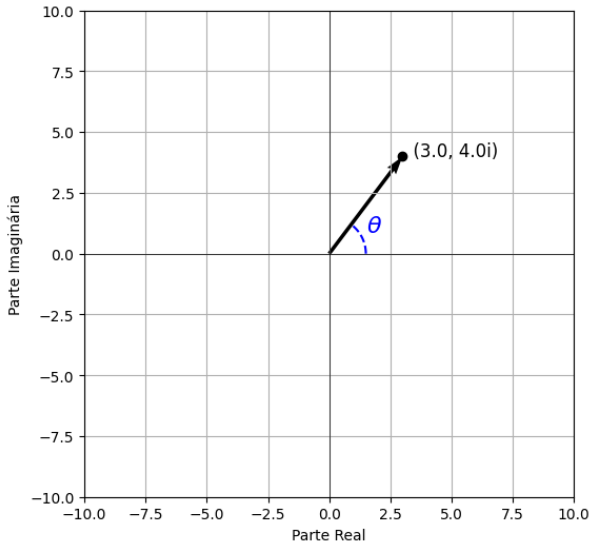


Figure 1.1: Representação de um número complexo graficamente.

Ao rodar esse código, obtemos a imagem mostrada na figura 1.1, representando o número complexo $z = (3 + 4i)$.

1.2 Introdução a Matrizes

Matrizes são arranjos retangulares de números, símbolos ou expressões organizados em linhas e colunas. Elas são fundamentais na álgebra linear e serão fundamentais para o nosso objetivo.
Uma matriz A de dimensão $m \times n$ é representada por

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}.$$

Onde:

- m é o número de linhas
- n é o número de colunas
- a_{ij} representa o elemento na linha i e coluna j

1.2.1 Propriedades de Matrizes

A primeira propriedade que vamos abordar é a soma e subtração de matrizes. Suponha duas matrizes A e B. Por exemplo, para realizar a operação de soma, basta somar os elementos das matrizes que possuem o mesmo i e j:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

$$A + B = \begin{pmatrix} (a_{11} + b_{11}) & (a_{12} + b_{12}) \\ (a_{21} + b_{21}) & (a_{22} + b_{22}) \end{pmatrix}.$$

1.2.2 Multiplicação de Matrizes por um Escalar

Supponha k é um número real e que a matriz A é dada por

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}.$$

Portanto, Ak será efetuada

$$Ak = \begin{pmatrix} ka_{11} & ka_{12} & \dots & ka_{1n} \\ ka_{21} & ka_{22} & \dots & ka_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ka_{m1} & ka_{m2} & \dots & ka_{mn} \end{pmatrix}.$$

1.2.3 Multiplicação de Matrizes

Para que a multiplicação de matrizes seja possível, é preciso que o número de colunas da primeira matriz seja igual ao número de linhas da segunda matriz. Os elementos da nova matriz serão então dados por

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}. \tag{1.9}$$

Isso significa que, para calcular cada elemento c_{ij} da matriz resultante, deve-se multiplicar cada elemento da i-ésima linha de A pelo elemento correspondente da j-ésima coluna de B, e então somar os resultados dos prodtuos. Por exemplo, considere as matrizes:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

$$A \cdot B = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

$$A \cdot B = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}.$$

1.2.4 Vetores Coluna

Denomina-se vetor coluna, a matriz com a seguinte característica

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}. \tag{1.10}$$

1.2.5 Vetores Linha

Denomina-se vetor de linha, a matriz com a seguinte estrutura

$$A = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix}. \tag{1.11}$$

1.2.6 Matriz identidade

Chama-se de Matriz Identidade (I) matrizes que em sua diagonal principal, possuem apenas o número 1 e nos outros termos 0. Por exemplo, a matriz identidade 2x2 é dada por

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1.12}$$

A matriz identidade é o elemento neutro multiplicativo da algebra linear

$$A \cdot I = A. \tag{1.13}$$

1.2.7 Determinante

O determinante de uma matriz traz diversas informações sobre ela. O determinante será fundamental quando for preciso encontrar os autovalores de uma matriz. Para isso, iremos mostrar como achar o determinante de matrizes 3×3, para matrizes de ordens superiores é mais proveitoso a utilização da biblioteca NumPy do Python devido a sua complexidade.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

$$\det(A) = (a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}) - (a_{13}a_{22}a_{31} + a_{11}a_{23}a_{32} + a_{12}a_{21}a_{33}).$$

.

1.2.8 Matriz Inversa

As matrizes inversas são denotadas pelo expoente -1. Dada uma matriz quadrada A, sua inversa A^{-1} é definida pela equação:

$$A \cdot A^{-1} = I, \tag{1.14}$$

onde I é a matriz identidade da mesma ordem que A. Essa equação indica que, ao multiplicarmos uma matriz por sua inversa, obtemos a matriz identidade.

Contudo, nem toda matriz possui inversa. Uma matriz só é inversível se for quadrada ,ou seja, tiver o mesmo número de linhas e colunas , e não for singular, o que significa que seu determinante deve ser diferente de zero

$$\det(A) \neq 0$$

Se o determinante for nulo, a matriz é chamada de singular e não admite inversa. Quando a matriz A é inversível, sua inversa pode ser calculada pela seguinte fórmula

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A).$$

Nesta expressão, $\det(A)$ representa o determinante da matriz A, e $\text{adj}(A)$ é a matriz adjunta de A.
A matriz adjunta é definida como a transposta da matriz dos cofatores de A. Para construí-la:

1. Calcula-se o cofator de cada elemento a_{ij} da matriz. O cofator é obtido eliminando-se a linha i e a coluna j e calculando o determinante da submatriz restante, multiplicado por $(-1)^{i+j}$.
2. Organiza-se esses cofatores em uma nova matriz chamada matriz de cofatores.
3. A adjunta é, então, a transposta dessa matriz de cofatores.

Por exemplo

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

$$\det(A) = a_{11}a_{22} - a_{12}a_{21},$$

Para uma matriz 2×2, os cofatores são

$$C_{11} = +a_{22}, \quad C_{12} = -a_{21}, \quad C_{21} = -a_{12}, \quad C_{22} = +a_{11}.$$

Portanto, a matriz dos cofatores é

$$\text{Cof}(A) = \begin{pmatrix} a_{22} & -a_{21} \\ -a_{12} & a_{11} \end{pmatrix},$$

A matriz adjunta é a transposta da matriz dos cofatores

$$\text{adj}(A) = \text{Cof}(A)^T = \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix},$$

Se $\det(A) \neq 0$, a matriz inversa é dada por

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A) = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \cdot \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}.$$

1.2.9 Matrizes Unitárias

Matrizes unitárias são aquelas matrizes que possuem a seguinte propriedade

$$U \cdot U^\dagger = I. \tag{1.16}$$

O simbolo † significa o conjugado transposto de uma matriz. Primeiramente, sabe-se que o conjugado de um número significa inverter o sinal da parte imaginária. Analogamente, com matrizes o conjugado de uma matriz é obtido invertendo o sinal da parte complexa de todos os seus elementos. Por exemplo, para

$$A = \begin{pmatrix} 1+2i & 3-i \\ 2+3i & -i \end{pmatrix},$$

$$A^* = \begin{pmatrix} 1-2i & 3+i \\ 2-3i & i \end{pmatrix}.$$

Uma matriz transposta é obtida ao trocar as linhas pelas colunas da matriz original. Em termos simples, as linhas da matriz se tornam colunas, e as colunas se tornam linhas. Por exemplo

a transposta de A, denotada por A^T, será:

$$A^T = \begin{pmatrix} 1+2i & 2+3i \\ 3-i & -i \end{pmatrix}.$$

Portanto, o transposto conjugado de uma matriz (†) é a junção das duas operações anteriores. Ou seja

$$A^\dagger = \begin{pmatrix} 1-2i & 2-3i \\ 3+i & +i \end{pmatrix}.$$

Assim, matrizes unitárias são matrizes que sempre satisfazem (1.16) Por exemplo:

$$U = \frac{1}{\sqrt{13}} \begin{pmatrix} 3+2i & 0 \\ 0 & 2-3i \end{pmatrix}$$

$$U^\dagger = \frac{1}{\sqrt{13}} \begin{pmatrix} 3-2i & 0 \\ 0 & 2+3i \end{pmatrix}$$

$$UU^\dagger = \left(\frac{1}{\sqrt{13}}\right)^2 \begin{pmatrix} (3+2i)(3-2i) & 0 \\ 0 & (2-3i)(2+3i) \end{pmatrix} = \frac{1}{13} \begin{pmatrix} 9+4 & 0 \\ 0 & 4+9 \end{pmatrix} = \frac{1}{13} \begin{pmatrix} 13 & 0 \\ 0 & 13 \end{pmatrix} = I.$$

Logo, U é uma matriz unitária, pois satisfaz $UU^\dagger = I$.

1.2.10 Matrizes Hermitianas

As matrizes chamadas Hermitianas que são iguais à suas transpostas conjulgadas ou seja

$$H = H^{\dagger}. \tag{1.17}$$

1.2.11 Autovetores e Autovalores

Quando aplicado uma transformação em um vetor, e o mesmo mantém sua direção, independentemente do valor de seu módulo, isso mostra que é possível evidenciar um escalar no resultado da nossa transformação, assim obtendo o vetor inicial multiplicado por este escalar. Por exemplo

$$\begin{pmatrix} 2 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} = 4 \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}.$$

Dessa forma, é denominado o a esse vetor o termo de **autovetor**.O escalar evidenciado é denominado **autovalor**,

$$A\vec{v} = \lambda\vec{v}. \tag{1.18}$$

Portanto, $\vec{v}$ é um autovetor e λ é um autovalor. Porém, nem sempre é imediato encontrar uma expressão do tipo (1.18). Para achar os autovalores e autovetores sem precisar especular valores, devemos realizar os seguintes passos:

1. Encontrar os autovalores
$$\det(A - \lambda I) = 0. \tag{1.19}$$
Onde A é a matriz com que estamos lidando, λ são os autovalores e I é a matriz identidade. Este determinante nos resultará em um polinômio, e as raízes deste polinômio serão os nossos autovalores.

2. Encontrar os autovetores

   Para cada solução do nosso polinômio λ, deve-se resolver a equação abaixo para encontrar os autovetores correspondentes,

$$(A - \lambda I)\vec{v} = 0 \tag{1.20}$$

Exemplo: Dada a matriz

$$A = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix}.$$

Calculamos

$$A - \lambda I = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1-\lambda & 1 \\ 4 & 1-\lambda \end{pmatrix}.$$

O determinante é

$$\det(A - \lambda I) = (1-\lambda)(1-\lambda) - (4)(1) = 0.$$

Resolvendo

$$\lambda^2 - 2\lambda - 3 = 0.$$

$$\lambda_1 = 3, \quad \lambda_2 = -1.$$

Portanto, os autovalores são 3 e -1. Para $\lambda = 3$

$$A - 3I = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix} - \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ 4 & -2 \end{pmatrix}.$$

Resolvendo

$$\begin{pmatrix} -2 & 1 \\ 4 & -2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

obtemos um sistema a ser resolvido

$$-2v_1 + v_2 = 0,$$
$$4_v 1 - 2v_2 = 0.$$

Da equação $-2v_1 + v_2 = 0$, temos

$$v_2 = 2v_1,$$

portanto, o autovetor associado a $\lambda = 3$ é

$$\vec{v} = \begin{pmatrix} c \\ 2c \end{pmatrix}, \quad c \in \mathbb{R}.$$

Vamos determinar o valor de c utilizando normalização, ou seja, exigindo que $\|\vec{v}\| = 1$. Calculamos a norma de $\vec{v}$

$$\|\vec{v}\| = \sqrt{c^2 + (2c)^2} = \sqrt{c^2 + 4c^2} = \sqrt{5c^2} = |c|\sqrt{5}.$$

Impondo a normalização

$$|c|\sqrt{5} = 1 \quad \Rightarrow \quad |c| = \frac{1}{\sqrt{5}} \quad \Rightarrow \quad c = \pm\frac{1}{\sqrt{5}}.$$

Portanto, os autovetores normalizados associados a $\lambda = 3$ são

$$\vec{v} = \pm\frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Para $\lambda = -1$

$$A - (-1)I = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix} - (-1) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 4 & 2 \end{pmatrix}.$$

Resolvendo

$$\begin{pmatrix} 2 & 1 \\ 4 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

obtemos o sistema

$$2v_1 + v_2 = 0,$$
$$4v_1 + 2v_2 = 0.$$

Da equação $2v_1 + v_2 = 0$, temos

$$v_2 = -2v_1,$$

portanto, o autovetor associado a $\lambda = -1$ é

$$\vec{v} = \begin{pmatrix} c \\ -2c \end{pmatrix}, \quad c \in \mathbb{R}.$$

Vamos determinar o valor de c utilizando normalização, ou seja, exigindo que $\|\vec{v}\| = 1$.

Calculamos a norma de $\vec{v}$

$$\|\vec{v}\| = \sqrt{c^2 + (-2c)^2} = \sqrt{c^2 + 4c^2} = \sqrt{5c^2} = |c|\sqrt{5}.$$

Impondo a normalização

$$|c|\sqrt{5} = 1 \quad \Rightarrow \quad |c| = \frac{1}{\sqrt{5}} \quad \Rightarrow \quad c = \pm\frac{1}{\sqrt{5}}.$$

Portanto, os autovetores normalizados associados a $\lambda = -1$ são

$$\vec{v} = \pm\frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

1.2.12 Notação de Dirac

A notação de Dirac é algo fundamental no estudo da computação quântica, ela é uma maneira de representar vetores coluna e vetores linha que foram vistos anteriormente. É uma notação compacta e poderosa usada principalmente em mecânica quântica para descrever estados quânticos e operações sobre esses estados. Ela foi introduzida por Paul Dirac e é amplamente utilizada devido à sua simplicidade e eficiência na representação de vetores e operadores em espaços de Hilbert. A notação de Dirac é separada em duas partes: o **ket** e o **bra**.

O **ket** é a notação que representa os vetores coluna e pode ser escrito da seguinte forma

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{1.21}$$

O **bra** é a notação que representa os vetores linha e pode ser escrito da seguinte forma

$$\langle\phi| = \begin{pmatrix} \alpha & \beta \end{pmatrix}. \tag{1.22}$$

Iremos explorar mais essa notação mais tarde no texto.

1.2.13 Produto Tensorial

O produto tensorial é uma operação fundamental em álgebra linear e teoria dos tensores, sendo uma generalização da multiplicação vetorial. Ele permite combinar dois ou mais espaços vetoriais para formar um novo espaço vetorial que codifica informações sobre as estruturas dos espaços originais. Ou seja, por definição temos que dados dois espaços vetoriais V e W, sobre um mesmo corpo K (como os números reais ou complexos), o produto tensorial $V \otimes W$ é um novo espaço vetorial que contém combinações lineares de pares ordenados de vetores $v \in V$ e $w \in W$. Assim, o produto tensorial é feito da seguinte forma

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \otimes \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} a_0 \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \\ a_1 \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{pmatrix}.$$

2 Mecânica Quântica

Nesta sessão iremos fazer um breve resumo sobre o que é a mecânica quântica e algumas motivações do porquê é interessante o estudo sobre computação quântica. A partir do século XIX, diversos fenômenos desafiavam a física atual, hoje em dia chamada de física clássica. Dentre esses fenômenos, podemos citar a radiação do corpo negro, o efeito fotoelétrico, que concedeu a Albert Einstein o seu prêmio Nobel, o comportamento ondulatório do elétron, entre outros. Através da mecânica quântica, tentamos explicar esses fenômenos de forma compatível com o que se observa experimentalmente.

O que é a computação quântica

Computação quântica é um campo multidisciplinar que compreende aspectos da ciência da computação, da física e da matemática, utilizando a mecânica quântica para resolver problemas complexos mais rapidamente do que em computadores tradicionais.

O que são computadores quânticos

Os computadores quânticos são capazes de resolver certos tipos de problemas mais rapidamente do que os computadores tradicionais, aproveitando os efeitos da mecânica quântica, como superposição e emaranhamento quântico. Algumas aplicações que os computadores quânticos podem fornecer aumento de velocidade incluem aprendizado de máquina, otimização e simulação de sistemas físicos. Além disso, podem ser usados para a otimização de portfólios em finanças ou a simulação de sistemas químicos, resolvendo problemas que atualmente são impossíveis até mesmo para os supercomputadores mais poderosos do mercado. Então, fica evidente que ao estudar computação quântica, podemos trabalhar com um leque de possibilidades, desde a criação de hardwares que possibilitem a construção de computadores quânticos cada vez mais eficientes até o desenvolvimento de algoritmos para resolver problemas físicos que ainda não são possíveis de resolver ou lidar com os computadores ditos clássicos. Atualmente, diversos sistemas físicos têm sido utilizados como base para a implementação de qubits e a construção de computadores quânticos. Entre os principais sistemas usados em computação quântica, destacam-se os íons aprisionados, que utilizam átomos carregados presos por campos eletromagnéticos, sendo altamente precisos e apresentando baixos níveis de erro; os supercondutores, que consistem em circuitos operando a temperaturas extremamente baixas, adotados por empresas como IBM, Google e Rigetti; os pontos quânticos (quantum dots), que são nanocristais semicondutores capazes de representar qubits; os fótons, utilizados na computação quântica óptica, que empregam partículas de luz para transportar e manipular

informação quântica, sendo promissores para comunicações seguras e redes quânticas; os defeitos em diamantes, especialmente os centros de vacância de nitrogênio, que exploram imperfeições na estrutura do diamante para armazenar e manipular informações quânticas, com potencial em sensores quânticos; e os átomos neutros, que são resfriados e manipulados com lasers, permitindo o controle preciso de grandes conjuntos de qubits.

2.1 Introdução ao Qubit e Superposição

Os computadores clássicos, os mesmos usam bits, $0's$ e $1's$ para armazenar e processar toda a sua informação. De maneira análoga, os computadores quânticos também irão usar uma espécie de bit, chamado *qubit* que é o que permite ao computador quântico realizar tarefas mais rápido. Assim como em computadores clássicos, ainda vamos usar $0's$ e $1's$, mas agora vamos representá-los da seguinte forma:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \tag{2.1}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.2}$$

Chamamos essa base de base computacional, e é geralmente empregada em protocolos de computação quântica.

Superposição

Por definição, a superposição é quando uma partícula pode ser descrita como a combinação linear de vários estados possíveis. Então, um *qubit* está em superposição quando seu estado é descrito por uma combinação linear dos estados base $|0\rangle$ e $|1\rangle$, ou seja,

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

onde α e β são números complexos que satisfazem a condição de normalização $|\alpha|^2 + |\beta|^2 = 1$. Essa superposição significa que o qubit não está apenas em um estado definido, mas em uma combinação dos dois, e o resultado de uma medição colapsará o estado para $|0\rangle$ com probabilidade $|\alpha|^2$ ou para $|1\rangle$ com probabilidade $|\beta|^2$. De forma geral, podemos representar um *qubit* como um vetor coluna (1.21).

Medição Quântica

De acordo com a mecânica quântica, quando é feito a medição de um sistema, o mesmo colapsa para o estado medido, isto é, a superposição do sistema é destruída. Na computação quântica não é diferente, quando medimos um *qubit*, ele colapsa para o estado medido, seja $|0\rangle$ ou $|1\rangle$, e a superposição é destruída. Portanto, quando temos um *qubit* do seguinte formato

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{2.3}$$

Os termos α e β estão associados à a probabilidade desse *qubit* se encontrar no estado $|0\rangle$ e $|1\rangle$, respectivamente. Dessa forma, a probabilidade de encontrar o nosso *qubit* no estado $|0\rangle$ e $|1\rangle$ é dada por

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha|0\rangle + \beta|1\rangle.$$

$$\text{Probabilidade de } |0\rangle : |\alpha|^2, \tag{2.4}$$

$$\text{Probabilidade de } |1\rangle : |\beta|^2. \tag{2.5}$$

Por exemplo

$$|\psi_1\rangle = \begin{pmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{pmatrix}.$$

$$|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle,$$

onde $\alpha = \frac{\sqrt{3}}{2}$ e $\beta = \frac{1}{2}$.

$$|\alpha|^2 = \left(\frac{\sqrt{3}}{2}\right)^2 = \frac{3}{4}, \quad |\beta|^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4}.$$

Portanto:

- A probabilidade de medir o estado $|0\rangle$ é $|\alpha|^2 = \frac{3}{4}$.
- A probabilidade de medir o estado $|1\rangle$ é $|\beta|^2 = \frac{1}{4}$.

Como esperado, a soma das probabilidades é igual a 1, o que é uma condição necessária para qualquer estado quântico válido:

$$|\alpha|^2 + |\beta|^2 = \frac{3}{4} + \frac{1}{4} = 1.$$

2.2 Representando um qubit na Esfera de Bloch

A Esfera de Bloch é uma representação gráfica usada na mecânica quântica para descrever o estado de um qubit. Esta fornece uma maneira intuitiva de visualizar os estados quânticos de um qubit e é fundamental para entender os conceitos de superposição e rotação de estados quânticos. Pode-se representar um *qubit* $|\psi\rangle$ como um ponto na superfíce da esfera Bloch. Para isso podemos usar algumas bibliotecas do python, como, *numpy*, *qutip* e *matplotlib*.

```
# Importando as bibliotecas necessarias
import numpy as np
from qutip import Bloch, basis
import matplotlib.pyplot as plt

# Parametros do qubit (coeficientes alfa e beta)
alpha = 1/np.sqrt(3)
beta = (np.sqrt(2)/np.sqrt(3)) * np.exp(1j * np.pi / 4)

# Definindo o estado quantico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os rotulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y

b.add_states(psi)
b.show()
```

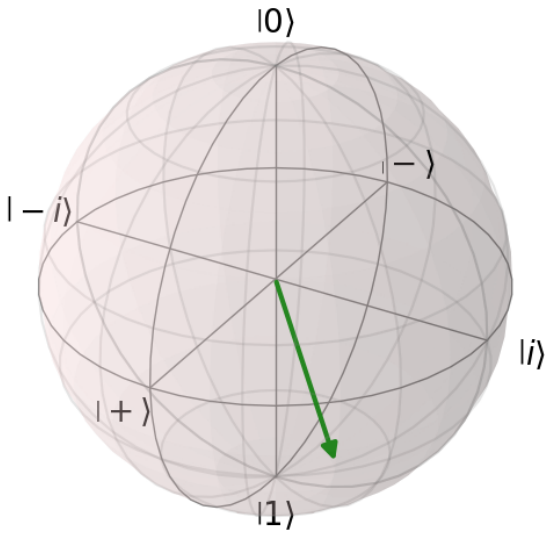


Figure 2.2: Representação do estado $|\psi\rangle$ na esfera de Bloch

Portanto, quanto mais perto do estado $|1\rangle$, maior a probabilidade desse *qubit* ser encontrado no estado $|1\rangle$, e o mesmo vale para os demais estados. Essa relação pode ser visualizada na Figura2.2, onde o vetor de estado aponta para uma região próxima ao polo correspondente ao estado $|1\rangle$.

2.3 Manipulando um Qubit com Portas de Um Único Qubit

Sabe-se que computadores clássicos usam portas lógicas para manipular e processar informações. Os computadores quânticos também usam portas lógicas para manipular as informações, mas essas portas têm diferenças quando comparadas com as portas clássicas, e são essas portas que serão abordadas nesta seção. Existem três portas lógicas fundamentais no estudo da computação quântica: as portas $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$. Pode-se representar essas portas de forma matricial como

$$\mathbb{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{2.6}$$

$$\mathbb{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}. \tag{2.7}$$

$$\mathbb{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.8}$$

Para entender melhor o comportamento dessas portas com os *qubits*, iremos aplicar as três portas a um *qubit* qualquer e representá-lo na esfera de Bloch, para que fique evidente o tipo de transformação que cada porta causa.

2.3.1 Porta $\mathbb{X}$

O *qubit* $|\psi\rangle$ na esfera de Bloch, dado por

$$|\psi\rangle = 1|0\rangle + 0|1\rangle. \tag{2.9}$$

Ou seja, esse *qubit* possui probabilidade maxíma de ser encontrado no estado $|0\rangle$. A partir do seguinte código pode-se vizualizar o estado na esfera de Bloch

```
# Importando as bibliotecas necessarias
!pip install qutip
import numpy as np
from qutip import Bloch, basis
import matplotlib.pyplot as plt
```

```
# Parametros do qubit (coeficientes alfa e beta)
alpha = 1
beta = 0

# Definindo o estado quantico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os rotulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y

b.add_states(psi)
b.show()
```

Ao aplicar a porta $\mathbb{X}$, que é equivalente a uma operação de NOT, o *qubit* $|\psi\rangle$ será transformado de $|0\rangle$ para $|1\rangle$. Na esfera de Bloch, a operação $\mathbb{X}$ corresponde a uma rotação de 180° ao longo do eixo x. Ultilizando o python pode-se representar está rotação da seguinte forma

```
# Importando as bibliotecas necessarias
!pip install qutip
import numpy as np
from qutip import Bloch, basis,sigmax
import matplotlib.pyplot as plt

# Parametros do qubit (coeficientes alfa e beta)
alpha = 1
beta = 0

# Definindo o estado quantico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

#Aplicando a porta X
X_gate = sigmax()  # Matriz da porta X

psi_after = X_gate*psi  # Aplicacao da porta X no estado

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os rotulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y

b.add_states(psi_after)
b.show()
```

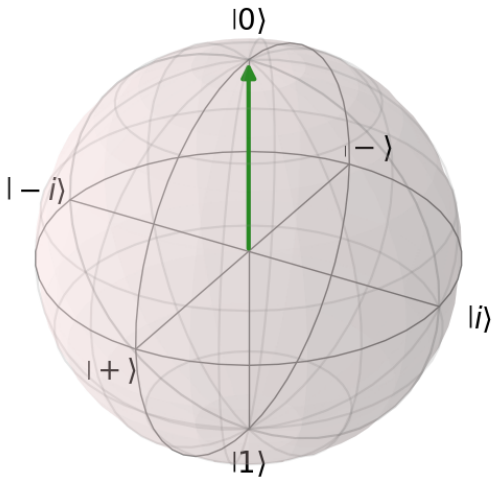


Figure 2.3: Representação de um Qubit com estado $|0\rangle$ na esfera de Bloch

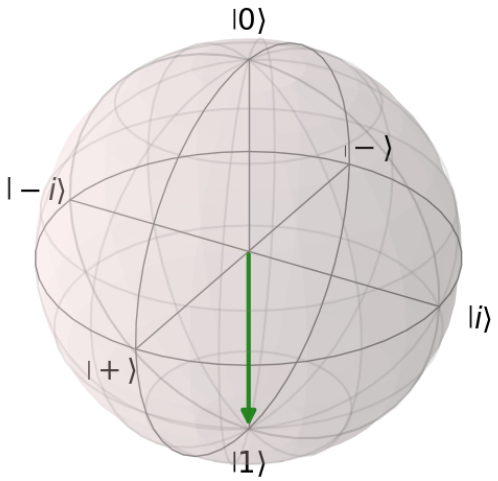


Figure 2.4: Representação de um Qubit após aplicado a porta $\mathbb{X}$ na esfera de Bloch

É evidente que, ao aplicar a porta lógica $\mathbb{X}$, o qubit é submetido a uma rotação de π radianos em torno do eixo x. Por esse motivo, essa porta é frequentemente denominada *bit-flip*, uma vez que realiza uma inversão do estado do qubit ao longo do eixo x, mostrado na Figura2.4.

2.3.2 Porta $\mathbb{Y}$

Para entender o comportamento da porta $\mathbb{Y}$ sobre um qubit, vamos definir os estados dois , $|-i\rangle$ e $|i\rangle$ como

$$|-i\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - i|1\rangle\right),$$

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

Partindo da representação do estado $|-i\rangle$ na esfera de Bloch, figura2.7

```
!pip install qutip


import numpy as np
from qutip import Bloch, basis
import matplotlib.pyplot as plt

# Coeficientes para o estado |i>
alpha = 1 / np.sqrt(2)
beta = -1j / np.sqrt(2)

# Definindo o estado quantico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os rotulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y
b.zlabel = [r'$|0\rangle$', r'$|1\rangle$']  # Opcional: Eixo Z

# Adicionando o estado a esfera
b.add_states(psi)

# Mostrando a esfera de Bloch
b.show()
```

Ao aplicar a porta $\mathbb{Y}$ tem-se que o estado resultante é $|i\rangle$, portanto esta porta opera uma rotação de π entorno do eixo y da esfera de Bloch. Pode-se representar esse estado na esfera de Bloch com o código abaixo.

```
!pip install qutip


import numpy as np
from qutip import Bloch, basis
import matplotlib.pyplot as plt

# Coeficientes para o estado |i>
alpha = 1 / np.sqrt(2)
beta = 1j / np.sqrt(2)

# Definindo o estado quantico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os rotulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y
b.zlabel = [r'$|0\rangle$', r'$|1\rangle$']  # Opcional: Eixo Z

# Adicionando o estado a esfera
b.add_states(psi)

# Mostrando a esfera de Bloch
b.show()
```

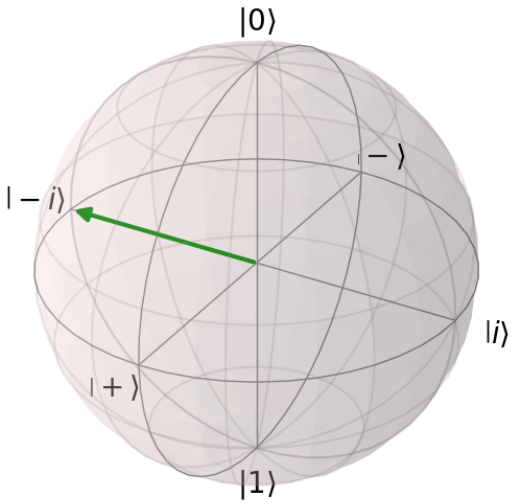


Figure 2.5: Representação do estado $|-i\rangle$ na esfera de bloch

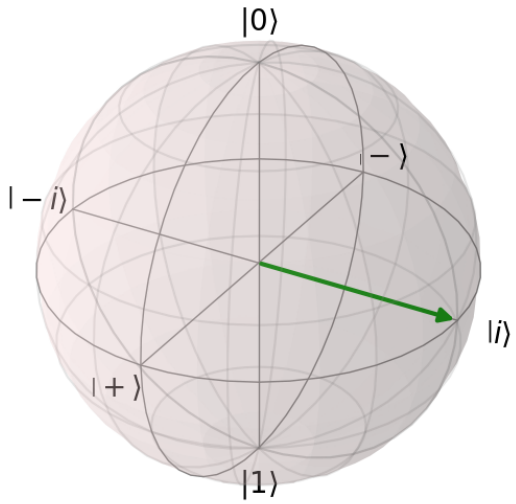


Figure 2.6: Representação do estado $|i\rangle$ na esfera de bloch

2.3.3 Porta $\mathbb{Z}$

Assim como a anterior a porta $\mathbb{Z}$ realiza uma rotação entorno do eixo z da esfera. Para esclarecer o comportamento da porta $\mathbb{Z}$ sobre um qubit, vamos introduzir superficalmente dois estados muito importantes na computação quântica, os estados $|-\rangle$ e $|+\rangle$, mais a diante, iremos nos aprofundar mais em ambos, no momento apenas vamos defini-los como

$$|-\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right),$$

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

Partindo da representação do estado $|-\rangle$ na esfera de Bloch, figura2.7

```
# Importando as bibliotecas necess rias
!pip install qutip
import numpy as np
from qutip import Bloch, basis
import matplotlib.pyplot as plt

# Par metros do qubit (coeficientes alfa e beta)
alpha = 1 / np.sqrt(2)
beta = -1 / np.sqrt(2)

# Definindo o estado qu ntico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os r tulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y

b.add_states(psi)
b.show()
```

Ao aplicar a porta $\mathbb{Z}$ tem-se que o estado resultante é $|+\rangle$, portanto a esta porta opera uma rotação de π entorno do eixo y da esfera de Bloch. Pode-se representar esse estado na esfera de Bloch com o código abaixo.

```
# Importando as bibliotecas necessarias
!pip install qutip
import numpy as np
from qutip import Bloch, basis, sigmay
import matplotlib.pyplot as plt

# Parametros do qubit (coeficientes alfa e beta)
alpha = 1 / np.sqrt(2)
beta = -1 / np.sqrt(2)

# Definindo o estado quantico |psi> = alpha|0> + beta|1>
psi = alpha * basis(2, 0) + beta * basis(2, 1)

#Aplicando a porta Y
Y_gate = sigmay()  # Matriz da porta Y

psi_after = Y_gate*psi  # Aplicacao da porta Y no estado

# Criando a esfera de Bloch
b = Bloch()

# Personalizando os rotulos dos eixos
b.xlabel = [r'$\left| + \right\rangle$', r'$\left| - \right\rangle$']  # Eixo X
b.ylabel = [r'$\left| i \right\rangle$', r'$\left| -i \right\rangle$']  # Eixo Y

b.add_states(psi_after)
b.show()
```

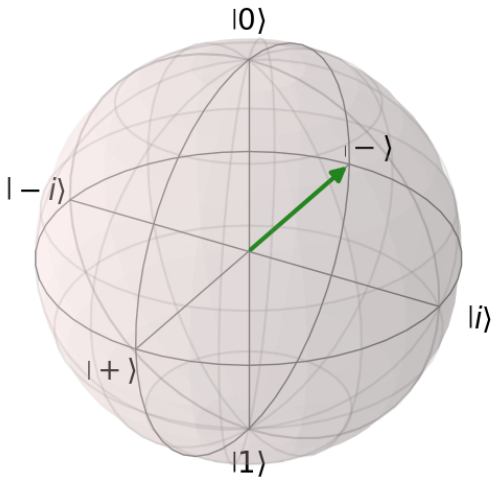


Figure 2.7: Representação do estado $|-\rangle$ na esfera de bloch

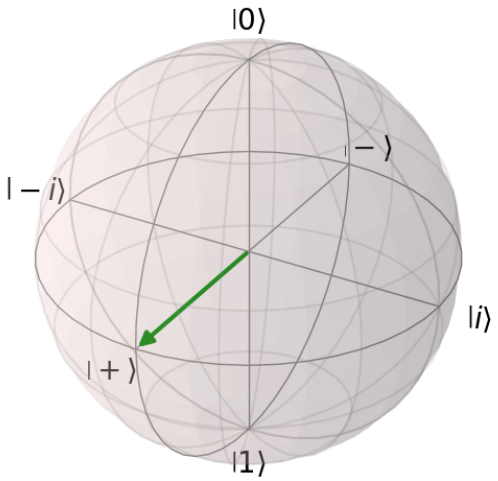


Figure 2.8: Representação do estado $|+\rangle$ na esfera de bloch

Portanto, é evidente que ao aplicar alguma das portas obtemos uma rotação do *qubit* em π radianos ao redor de algum dos eixos. Dessa forma, caso a mesma porta seja aplicada duas vezes, é obtido que o *qubit* retornará ao seu estado inicial. Por consequência, temos que essas **portas são suas próprias inversas**, podemos provar isso da seguinte forma

$$\begin{aligned} |\psi\rangle =& \alpha|0\rangle + \beta|1\rangle, \\ \mathbb{X}|\psi\rangle =& \alpha|1\rangle + \beta|0\rangle = |\zeta\rangle, \\ \mathbb{X}|\zeta\rangle =& \alpha|0\rangle + \beta|1\rangle = |\psi\rangle. \end{aligned}$$

2.3.4 Como aplicar uma porta lógica

Para aplicar qualquer uma das portas lógicas em um *qubit*, basta realizar o produto matricial entre porta e o *qubit*, por exemplo

$$\mathbb{X}|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}.$$

Com esse exemplo, pode-se observar que a primeira coluna de uma porta lógica representa o valor que o estado $|0\rangle$ retornará após a aplicação da porta, e a segunda coluna representa o estado que $|1\rangle$ irá retornar após a aplicação da porta. Ou seja, se tomado a primeira coluna da porta $\mathbb{X}$, tem-se $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, que é o estado $|1\rangle$. Portanto, quando a porta $\mathbb{X}$ é aplicada, o estado $|0\rangle$ ira se tornar o estado $|1\rangle$, e a mesma lógica se aplica para a segunda coluna. Outra propriedade muito importante das portas lógicas é que são sempre lineares e reversíveis. Portanto, pode-se distribuir uma porta lógica para cada estado do *qubit* individualmente

$$\mathbb{X}|\psi\rangle = \mathbb{X}(\alpha|0\rangle + \beta|1\rangle) = \alpha\mathbb{X}|0\rangle + \beta\mathbb{X}|1\rangle$$

Assim, é útil ultilizar estas propriedades para calcular o resultado das aplicações de portas lógicas sobre um *qubit*, por exemplo

$$\mathbb{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},$$

$$\mathbb{Y}|\psi\rangle = \alpha\mathbb{Y}|0\rangle + \beta\mathbb{Y}|1\rangle,$$

$$\mathbb{Y}|\psi\rangle = \alpha \begin{pmatrix} 0 \\ i \end{pmatrix} + \beta \begin{pmatrix} -i \\ 0 \end{pmatrix},$$

Fatorando o i, e -i, obtemos

$$\mathbb{Y}|\psi\rangle = \alpha i \begin{pmatrix} 0 \\ 1 \end{pmatrix} - \beta i \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

$$\mathbb{Y}|\psi\rangle = \alpha i|0\rangle - \beta i|1\rangle.$$

2.4 Introdução à Fase

Foi apresentado anteriormente que, na esfera de Bloch, pode-se rotacionar um estado aplicando portas lógicas no mesmo. A porta lógica $\mathbb{Z}$, faz uma rotação de π ao redor do eixo z, o que é definido como fase. Porém, quando essa porta é aplicada, a probabilidade do *qubit* estar em alguns estados possíveis não é alterado. Apesar de não parecer útil, a fase é um dos principais motivos de os computadores quânticos serem tão poderosos. Iremos discutir mais tarde o uso das fases. Nesse momento, vamos definir como escrevê-las matematicamente, mas antes vamos apresentar com mais detalhes os quatro principais estados em que a fase está presente

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \tag{2.10}$$

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle, \tag{2.11}$$

$$|i\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{i}{\sqrt{2}}|1\rangle, \tag{2.12}$$

$$|-i\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle. \tag{2.13}$$

Os estados $|+\rangle$, $|-\rangle$, $|i\rangle$ e $|-i\rangle$ representam superposições coerentes do qubit entre os estados base $|0\rangle$ e $|1\rangle$, diferenciando-se pelas fases relativas entre esses componentes. Esses estados são fundamentais na computação quântica porque permitem manipular a informação quântica em diferentes direções da esfera de Bloch, não se restringindo aos eixos clássicos $|0\rangle$ e $|1\rangle$. Para representar uma fase, vamos pensar no estado $|+\rangle$ e aplicá-lo à (2.10)

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \tag{2.14}$$

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

Ou seja, quando é aplicado uma rotação de π ao redor do eixo z, o segundo termo do *qubit* foi multiplicado pelo fator -1. É possível reescrever esse fator -1 na forma $e^{i\varphi}$, onde φ nesse caso é π radianos. Assim, o estado $|+\rangle$ pode ser descrito da da seguinte forma

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + e^{i\varphi}\frac{1}{\sqrt{2}}|1\rangle.$$

A vantagem de evindenciar o termo polar apenas no segundo item da expressão deve-se ao fato de que, ao aplicar uma fase nos dois termos, chamada de fase global, não temos nenhuma mudança física no sistema. Portanto, ela é irrelevante. Assim, vamos tratar apenas das fases relativas.

2.5 A Porta Hadamard

Foi mencionado anteriormente que na região equatorial da esfera de Bloch, temos 4 estados: $|+\rangle$, $|-\rangle$, $|i\rangle$, $|-i\rangle$. Apesar de estes terem a mesma probabilidade de um *qubit* estar no estado $|0\rangle$ ou $|1\rangle$, eles possuem diferentes fases. Com isso, podemos definir a porta Hadamard como

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.15}$$

Ao aplicar essa porta lógica ao estado $|0\rangle$, ela irá retornar o estado $|+\rangle$, e caso aplicada no estado $|1\rangle$, retornará $|-\rangle$. E o inverso é válido também.

$$|H\rangle|1\rangle = |+\rangle \tag{2.16}$$
$$|H\rangle|0\rangle = |-\rangle \tag{2.17}$$
$$|H\rangle|+\rangle = |1\rangle \tag{2.18}$$
$$|H\rangle|-\rangle = |0\rangle \tag{2.19}$$

A porta Hadamard é fundamental na computação quântica, pois ela é capaz de gerar o estado de superposição . Além do mais a porta lógica mostra a importância das fases relativas nos *qubits*, pois apesar de $|+\rangle$ e $|-\rangle$ possuírem a mesma probabilidade de serem colapsdos para os estados $|1\rangle$ e $|0\rangle$, elas se diferenciarem apenas pela fase relativa. Assim ao aplicado a porta Hadamard, cada estado retorna resultados completamente distintos.

3 Múltiplos Qubits

3.1 Representando Múltiplos Qubits Matematicamente

Agora que foi definido o que são *qubits*, como podemos representá-los na esfera de Bloch, e como podemos operar portas lógicas neles, vamos adentrar no estudo de múltiplos *qubits*. Até então, foi tratado sistemas de apenas um *qubit*, mas as situações que iremos nos deparar geralmente requerem o tratamento de mais de um *qubit*. O primeiro passo é entender como representar múltiplos *qubits*. Para isso, é necessário o uso do produto tensorial abordado na seção 1.2.13. Por exemplo, para representar 2 *qubits* no estado $|0\rangle$ aplicamos

$$|\psi\rangle = |0\rangle \otimes |0\rangle. \tag{3.1}$$

Pode-se simplificar essa notação para simplesmente

$$|\psi\rangle = |00\rangle. \tag{3.2}$$

Com isso, é útil definir um estado de dois *qubits* genéricos apartir do produto tensorial

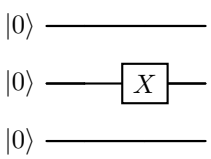
$$(\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle. \tag{3.3}$$

É recomendado que o leitor faça o produto tensorial conforme explicado na seção 1.2.13. Lembre-se de realizar a distributiva para cada elemento ao calcular o produto tensorial.

3.2 Circuitos Quânticos

Os circuitos quânticos são fundamentais para manipulação de *qubits* específicos de estados de múltiplos *qubits*. Por exemplo, dado o estado de três *qubits* qual é o circuito que aplica a porta $\mathbb{X}$ no segundo qubit? A resposta para essa pergunta é através do uso do seguinte circuito quântico.

$$|000\rangle \tag{3.4}$$



Assim cada linha desse circuito representa cada 1 dos *qubits* do estado. Portanto apenas no *qubit* do meio é aplicado a porta $\mathbb{X}$ e por final é realizado a medição em todos os *qubits*. Após a medição temos

$$|010\rangle. \tag{3.5}$$

Podemos mostrar isso matematicamente

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

a porta X é

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

queremos aplicar X apenas no segundo qubit, ou seja

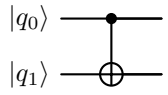
$$U = I \otimes X \otimes I,$$

aplicando a operação ao estado

$$(I \otimes X \otimes I)\,|000\rangle = |010\rangle\,.$$

3.3 Portas Multi-Qubit

Quando ultizado varios *qubits* se faz necessário portas que atuam em mais de um *qubit* possibilitando a criação lógicas mais complexas para o circuito. A primeira porta que será tratada, é a $CNOT$ ou porta NOT controlada.

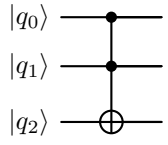


No diagrama têm-se duas notações $\otimes$ e $\bullet$. O $\bullet$ é defindo como *qubit* de controle e $\otimes$ o *qubit* alvo. Em geral o *qubit* de controle ele impõe uma condição, e se ela for satisfeita a porta lógica é aplicada ao *qubit* alvo. Na porta CNOT, quando o *qubit* de controle for $|1\rangle$ a porta lógica $\mathbb{X}$ é aplicada ao *qubit* alvo. Por exemplo, defindo o segundo *qubit* como *qubit* de controle e o terceiro sendo *target qubit* temos

$$CNOT(|010\rangle) = |011\rangle$$

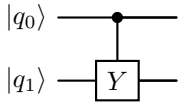
3.3.1 Porta Toffoli

Analogamente ao CNOT a porta de Toffoli funciona da mesma maneira, a diferença é que a porta de Toffoli possui dois *qubits* de controle, isso é, o *qubit* alvo terá em si aplicado a porta $\mathbb{X}$ se tanto o primeiro e o segundo *qubit* de controle forem $|1\rangle$, dessa maneira a Porta de Toffoli é comumente chamda de $CCNOT$.



3.3.2 Outras portas de controle

Pode-se extender estes mesmos conceitos para qualquer porta lógica. É evidente que para as outras portas é ultilizado uma notação diferente para o *qubit* alvo, onde agora este é representado por uma caixa com o inicial da porta a ser aplicado ao *target qubit*. Neste caso é aplicado a porta Y.



3.4 Medindo Sistemas de um Qubit

Como medir a probabilidade de um *qubit* em um estado com dois ou mais *qubits*? Por exemplo, vamos supor que queremos medir a probabilidade do segundo *qubit* no seguinte estado ser $|1\rangle$

$$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{4}|01\rangle + \frac{\sqrt{3}}{4}|11\rangle. \tag{3.6}$$

Para isso, deve-se analisar todas as superposições onde o segundo *qubit* é $|1\rangle$ e somar as suas probabilidades. Portanto, a probabilidade de encontrar o estado $|1\rangle$ no estado $|\psi\rangle$ é

$$|\psi\rangle = \frac{1}{2}\,|0\rangle\,|0\rangle + \frac{1}{4}\,|0\rangle\,|1\rangle + \frac{\sqrt{3}}{4}\,|1\rangle\,|1\rangle\,, \tag{3.7}$$

os termos com o segundo qubit igual a $|1\rangle$ são

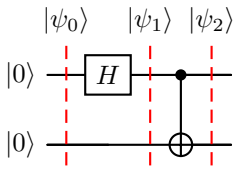
$$\frac{1}{4}\,|0\rangle\,|1\rangle\,, \quad \frac{\sqrt{3}}{4}\,|1\rangle\,|1\rangle\,,$$

$$P(\text{segundo qubit} = |1\rangle) = \left|\frac{1}{4}\right|^2 + \left|\frac{\sqrt{3}}{4}\right|^2 = \frac{1}{4}. \tag{3.8}$$

$$\tag{3.9}$$

3.4.1 Emanharamento Quântico e Estados de Bell

Supondo o seguinte circuito



Inicialmente no estado $|\psi_0\rangle$ temos

$$|\psi_0\rangle = |00\rangle \tag{3.10}$$

Então, é aplicado a porta Hadamard no primeiro *qubit*. Após a aplicação, em $|\psi_1\rangle$, é obtido o estado

$$|\psi_1\rangle = H \otimes |\psi_0\rangle = H \otimes \mathbb{I}|00\rangle, \tag{3.11}$$

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle, \tag{3.12}$$

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle). \tag{3.13}$$

Assim, é aplicado a porta $CNOT$, onde o primeiro *qubit* é o controle e o segundo é o alvo

$$|\psi_2\rangle = CNOT|\psi_1\rangle, \tag{3.14}$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{3.15}$$

Este simples circuito evidência algo muito interessante, caso seja feito a medida do primeiro *qubit* com ele no estado $|0\rangle$, automaticamente sabemos que o segundo *qubit* será $|0\rangle$, sem precisar medi-lo. De forma análoga, se o primeiro *qubit* for $|1\rangle$, o segundo também será $|1\rangle$. Essa propriedade de saber o valor do segundo *qubit* sem precisar medi-lo é chamada de *emaranhamento quântico*, um tipo de correlação não local. Matematicamente podemos dizer que um *qubit* está emaranhado caso não seja possível fatorá-lo no produto tensorial de vários *qubits* únicos. Por exemplo

$$|\psi\rangle = \frac{\sqrt{3}}{2\sqrt{5}}|00\rangle + \frac{1}{2\sqrt{5}}|01\rangle + \frac{\sqrt{3}}{\sqrt{5}}|10\rangle + \frac{1}{\sqrt{5}}|11\rangle = \left(\frac{1}{\sqrt{5}}|0\rangle + \frac{2}{\sqrt{5}}|1\rangle\right) \otimes \left(\frac{\sqrt{3}}{2}|0\rangle + \frac{1}{2}|1\rangle\right). \tag{3.16}$$

Neste caso, o estado $|\psi\rangle$ não está emaranhado, pois foi possível fatorá-lo em um produto tensorial. Já no caso

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{3.17}$$

Temos um emaranhamento, pois não é possível separá-lo em produtos tensoriais de estados individuais.

3.4.2 Emaranhamentos

O **emaranhamento quântico** é uma propriedade fundamental dos sistemas quânticos composta por dois ou mais qubits. Dizemos que qubits estão emaranhados quando o estado do sistema como um todo não pode ser descrito apenas pelos estados individuais de cada qubit. Intuitivamente, isso significa que a informação do sistema está compartilhada entre os qubits de uma forma que não é possível separá-la: ao medir um qubit, o estado do outro qubit é instantaneamente afetado, mesmo que eles estejam distantes.

Definição: Um estado de dois qubits $|\psi\rangle_{AB}$ é **emaranhado** se ele não puder ser escrito como o produto de dois estados individuais:

$$|\psi\rangle_{AB} \neq |\phi\rangle_A \otimes |\chi\rangle_B .$$

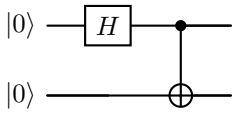
Fiscamente pode-se pensar em dois qubits que foram preparados juntos e depois separados. Se eles estiverem em um estado emaranhado, a medição de um deles revelará imediatamente o estado do outro. Mas o mais curioso é que, antes da medição, nenhum dos qubits possui um estado bem definido por si só, apenas o sistema total tem uma descrição completa. Isso mostra que o emaranhamento não é apenas uma correlação estatística como na física clássica, mas uma conexão profunda entre os sistemas.

Estado maximamente emaranhado: Um estado **maximamente emaranhado** é aquele em que os qubits estão completamente correlacionados, e nenhuma informação pode ser obtida observando um deles isoladamente. Toda a informação está no sistema como um todo. Por exemplos os estados de Bell

$$\begin{aligned}
\left|\Phi^+\right\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \\
\left|\Phi^-\right\rangle &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle), \\
\left|\Psi^+\right\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), \\
\left|\Psi^-\right\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle).
\end{aligned}$$

são exemplos de estados **maximamente emaranhado**. Se medirmos o primeiro qubit e obtivermos $|0\rangle$, o segundo qubit colapsa para $|0\rangle$; se obtivermos $|1\rangle$, o segundo qubit colapsa para $|1\rangle$. Isso acontece mesmo sem nenhuma comunicação entre os qubits. Existem outros estados maximamente emaranhados como o estado GHZ que apresenta máximo emaranhamento com três ou mais *qubits*

Estados maximamente emaranhados podem ser gerados com circuitos simples compostos por portas Hadamard e CNOT. Por exemplo para criar o estado $|\Phi^+\rangle$ começamos com dois qubits no estado $|00\rangle$



Resultado

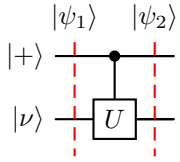
$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \left|\Phi^+\right\rangle.$$

Estado parcialmente emaranhado: Um estado é chamado de **parcialmente emaranhado** quando os qubits estão correlacionados de forma quântica, mas o grau de emaranhamento não é máximo. Esses estados ainda não podem ser descritos como um produto de estados individuais, mas as correlações entre os qubits não atingem a intensidade ou a simetria de estados maximamente emaranhados, como os estados de Bell, por exemplo.

$$|\psi\rangle = \frac{3}{5}|00\rangle + \frac{4}{5}|11\rangle$$

Esse é um exemplo de **estado parcialmente emaranhado**. Ele não pode ser fatorado como um produto de dois qubits individuais (logo, é emaranhado), mas a distribuição de probabilidades não é equilibrada: a probabilidade de o sistema estar no estado $|11\rangle$ é maior que no estado $|00\rangle$. Diferente do estado de Bell $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, que tem simetria perfeita e emaranhamento máximo, aqui a assimetria nos coeficientes indica que o grau de emaranhamento é menor.

3.5 Fase de *kickback*



Dado o circuito no primeiro *qubit*, temos o estado $|+\rangle$, e no segundo *qubit*, o estado $|\nu\rangle$. Definimos que $|\nu\rangle$ é um autovetor de U. Dessa forma, se aplicado a porta U em $|\nu\rangle$, obtemos

$$U|\nu\rangle = e^{i\theta}|\nu\rangle. \tag{3.18}$$

Logo em seguida no estado $|\psi_1\rangle$, temos

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\nu\rangle + |1\rangle|\nu\rangle). \tag{3.19}$$

Aplicando a porta U Controlada, temos que

- No primeiro estado de superposição, nada acontece, pois o *qubit* de controle é $|0\rangle$.

- No segundo estado, aplicamos a porta U em $|\nu\rangle$, que é o alvo.

Assim

$$|\psi_2\rangle = U|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\nu\rangle + |1\rangle e^{i\theta}|\nu\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle e^{i\theta})|v\rangle. \tag{3.20}$$

Com isso, é evidente que o estado $|v\rangle$ permaneceu inalterado, enquanto uma **fase relativa** foi adicionada ao *qubit* alvo. Isso demonstra que, quando o estado é um autoestado da porta lógica que é aplicada, ela "chuta" a fase relativa para dentro do *qubit* alvo. Por isso, chamamos esse fenômeno de **fase kickback**.

4 Algoritmos Quânticos

4.1 Teleporte Quântico

O **teleporte quântico** é um protocolo fundamental da informação quântica que permite transferir o estado de um qubit de um lugar (Alice) para outro (Bob), **sem mover fisicamente o qubit**. Esse protocolo depende do entrelaçamento quântico e da comunicação clássica para funcionar.

4.1.1 Teoria e Conceito

O objetivo do protocolo é transmitir um estado quântico arbitrário, como

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

de Alice para Bob, sem enviar fisicamente o qubit. Para isso, Alice e Bob compartilham um par de qubits entrelaçados no estado de Bell:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_{A2}|0\rangle_B + |1\rangle_{A2}|1\rangle_B),$$

onde:

- $A1$: qubit com Alice contendo o estado $|\psi\rangle$.
- $A2$: qubit de Alice emaranhado com Bob.
- B: qubit de Bob que receberá o estado.

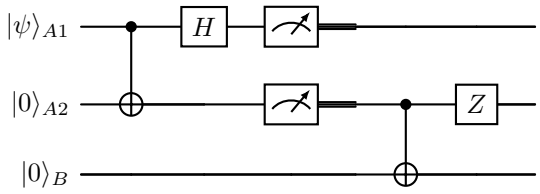
O protocolo usa dois recursos:

- Um par de qubits maximamente emaranhados entre Alice e Bob (A2 e B).
- Comunicação clássica de dois bits após uma medição por parte de Alice.

4.1.2 Etapas do Protocolo

1. Alice possui dois qubits: $A1$ com o estado $|\psi\rangle$ e $A2$, entrelaçado com o qubit de Bob B.
2. Alice aplica a porta CNOT com controle em $A1$ e alvo em $A2$.
3. Em seguida, ela aplica uma porta Hadamard em $A1$.
4. Alice mede os qubits $A1$ e $A2$, e envia os dois bits clássicos a Bob.
5. Bob aplica correções (X, Z) em seu qubit B dependendo dos resultados da medição para reconstruir $|\psi\rangle$.

4.2 Circuito do Protocolo com Nomes de Qubits



Legenda:

- $A1$: qubit com o estado quântico que será teleportado.
- $A2$: qubit de Alice emaranhado com Bob.
- B: qubit de Bob, que receberá o estado final.
- Após as medições de $A1$ e $A2$, Bob aplica as correções em B conforme os bits recebidos, para obter o estado teleportado

Bits de Alice	Correção de Bob em B
00	I (nenhuma operação)
01	X
10	Z
11	$Z \cdot X$

Exemplo prático de teleporte quântico

Suponha que Alice deseje teleportar para Bob o estado quântico

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \text{com } |\alpha|^2 + |\beta|^2 = 1.$$

Eles compartilham um par emaranhado no estado de Bell

$$\left|\phi^+\right\rangle_{AB} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Alice tem dois qubits: o qubit que ela quer enviar ($|\psi\rangle$) e o primeiro qubit do par de Bell. Bob tem o segundo qubit do par.

O estado combinado dos três qubits (ordem: $|\psi\rangle \otimes |\phi^+\rangle_{AB}$) é

$$\begin{aligned}
|\Psi\rangle_{123} &= |\psi\rangle_1 \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)_{23}, \\
&= \frac{1}{\sqrt{2}}\left[\alpha\,|0\rangle_1 (|00\rangle_{23} + |11\rangle_{23}) + \beta\,|1\rangle_1 (|00\rangle_{23} + |11\rangle_{23})\right], \\
&= \frac{1}{\sqrt{2}}\left[\alpha\,|000\rangle + \alpha\,|011\rangle + \beta\,|100\rangle + \beta\,|111\rangle\right].
\end{aligned}$$

Agora, Alice aplica a porta CNOT no qubit 1 (controle) e 2 (alvo), e depois a porta Hadamard no qubit 1

Aplicando CNOT (1 → 2):

O estado $|011\rangle$ não é alterado, pois o qubit de controle é $|0\rangle$. O estado $|100\rangle$ se torna $|110\rangle$ e $|111\rangle$ se torna $|101\rangle$.

O estado após a CNOT é

$$\frac{1}{\sqrt{2}}\left[\alpha\,|000\rangle + \alpha\,|011\rangle + \beta\,|110\rangle + \beta\,|101\rangle\right].$$

Aplicando Hadamard no qubit 1

Lembrando que

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

Aplicando o Hadamard em cada termo do estado:

$$\begin{aligned}
\alpha\,|000\rangle &\to \alpha\frac{1}{\sqrt{2}}(|000\rangle + |100\rangle), \\
\alpha\,|011\rangle &\to \alpha\frac{1}{\sqrt{2}}(|011\rangle + |111\rangle), \\
\beta\,|110\rangle &\to \beta\frac{1}{\sqrt{2}}(|010\rangle - |110\rangle), \\
\beta\,|101\rangle &\to \beta\frac{1}{\sqrt{2}}(|001\rangle - |101\rangle).
\end{aligned}$$

Substituindo no estado final e agrupando os termos:

$$|\Psi'\rangle = \frac{1}{2}\big[\alpha(|000\rangle + |100\rangle + |011\rangle + |111\rangle) + \quad \beta(|010\rangle - |110\rangle + |001\rangle - |101\rangle)\big].$$

Agrupando os estados pela medição de Alice (qubits 1 e 2)

$$|\Psi'\rangle = \frac{1}{2}\left[|00\rangle_{12}(\alpha|0\rangle_3 + \beta|1\rangle_3) + |01\rangle_{12}(\alpha|1\rangle_3 + \beta|0\rangle_3) + |10\rangle_{12}(\alpha|0\rangle_3 - \beta|1\rangle_3) + |11\rangle_{12}(\alpha|1\rangle_3 - \beta|0\rangle_3)\right].$$

Assim, após medir os qubits 1 e 2, o qubit 3 (de Bob) colapsa em um dos seguintes estados:

Medição de Alice	**Estado de Bob**	**Correção**
$\|00\rangle$	$\alpha\|0\rangle + \beta\|1\rangle$	I (identidade)
$\|01\rangle$	$\alpha\|1\rangle + \beta\|0\rangle$	Porta X
$\|10\rangle$	$\alpha\|0\rangle - \beta\|1\rangle$	Porta Z
$\|11\rangle$	$\alpha\|1\rangle - \beta\|0\rangle$	Porta X seguida de Z

- **Caso 00**: Bob já possui o estado correto $(\alpha|0\rangle + \beta|1\rangle) = |\psi\rangle$.
- **Caso 01**: A porta X inverte $|0\rangle \to |1\rangle$:
  $X(\alpha|1\rangle + \beta|0\rangle) = \alpha|0\rangle + \beta|1\rangle = |\psi\rangle$.
- **Caso 10**: A porta Z aplica uma fase negativa ao estado $|1\rangle$:
  $Z(\alpha|0\rangle - \beta|1\rangle) = \alpha|0\rangle + \beta|1\rangle = |\psi\rangle$.
- **Caso 11**: Primeiro aplica-se a porta X e depois a porta Z:
  $X(\alpha|1\rangle - \beta|0\rangle) = \alpha|0\rangle - \beta|1\rangle$, e depois $Z(\alpha|0\rangle - \beta|1\rangle) = \alpha|0\rangle + \beta|1\rangle = |\psi\rangle$.

Portanto, ao comunicar os dois bits clássicos da sua medição para Bob, Alice permite que ele recupere exatamente o estado $|\psi\rangle$ original com as operações corretas.

4.2.1 Implementação em Python (Qiskit)

```
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.visualization import plot_histogram
from math import sqrt
import matplotlib.pyplot as plt

def teleportar_estado(alpha, beta):
    qc = QuantumCircuit(3, 2)

    # Qubit 0: A1 - estado a ser teleportado
    # Qubit 1: A2 - qubit de Alice, entrelacado com Bob
    # Qubit 2: B  - qubit de Bob

    # 1. Inicializa o estado a ser teleportado em A1
    qc.initialize([alpha, beta], 0)

    # 2. Criar par emaranhado entre A2 e B
    qc.h(1)
    qc.cx(1, 2)

    # 3. Operacoes de Alice (A1 e A2)
    qc.cx(0, 1)
    qc.h(0)

    # 4. Medicoes de Alice
    qc.measure(0, 0)
    qc.measure(1, 1)

    # 5. Simulacao
    sim = Aer.get_backend('aer_simulator')
    qc = transpile(qc, sim)
    result = sim.run(qc, shots=1024).result()
    counts = result.get_counts()

    print("Resultado das medicoes de Alice:", counts)
    plot_histogram(counts)
    plt.show()

# Exemplo: teleportar (1/ 2 )|0> + (1/ 2 )|1>
teleportar_estado(1/sqrt(2), 1/sqrt(2))
```

4.2.2 Possíveis Erros e Desafios

1. Desalinhamento de qubits

Se os qubits não forem devidamente emaranhados, o teleporte falhará.

2. Ruído

O canal quântico pode introduzir erros no emaranhamento ou na medição.

3. Correções não aplicadas

Bob precisa aplicar a operação correta baseada nos bits clássicos. Sem essa etapa, o estado $|\psi\rangle$ não será restaurado.

4.2.3 Conclusão

O teleporte quântico permite transmitir um estado quântico sem transportá-lo fisicamente, utilizando emaranhamento e comunicação clássica. Ele é uma aplicação chave da informação quântica e uma base para futuras tecnologias de comunicação quântica.

4.3 Codificação superdensa

A **codificação superdensa** é um protocolo quântico que permite enviar **dois bits clássicos** usando apenas **um único qubit**, aproveitando o fenômeno do emaranhamento quântico. Esse protocolo faz uso da propriedade de que um estado emaranhado pode carregar mais informação do que um estado clássico tradicional.

4.3.1 Teoria e Conceito

O objetivo do protocolo é enviar dois bits clássicos com um *qubit*, assim podemos definir um bit clássico como a unidade básica de informação na computação.

Imagine que Alice e Bob compartilham um par de *qubits* **maximamente emaranhados**, que estão em um estado de Bell. Onde o primeiro qubit pertence a Alice e o segundo a Bob. O estado compartilhado pode ser representado por

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

Agora, Alice tem como objetivo enviar dois bits clássicos para Bob. Para isso, ela pode manipular o *qubit* que lhe pertence, realizando operações quânticas sobre ele, e então enviá-lo para Bob. Com essas operações, ela codifica a informação que deseja transmitir.

4.3.2 Como Alice Codifica os Bits Clássicos

Alice pode enviar os seguintes estados, aplicando portas quânticas ao seu *qubit*:

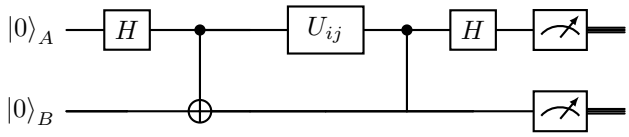
- **Para enviar** $|00\rangle$: Alice não realiza nenhuma operação no seu *qubit*, mantendo-o no estado inicial $|\Psi\rangle$.
- **Para enviar** $|01\rangle$: Alice aplica a porta de Pauli X no seu *qubit*.
- **Para enviar** $|10\rangle$: Alice aplica a porta de Pauli Z no seu *qubit*, que adiciona uma fase ao estado $|1\rangle$, transformando $|1\rangle$ em $-|1\rangle$, mas deixando $|0\rangle$ inalterado.
- **Para enviar** $|11\rangle$: Alice aplica as portas X e Z no seu *qubit*, o que gera uma combinação das operações anteriores.

4.3.3 Como Bob Decodifica os Bits Clássicos

Após Alice realizar a operação desejada e enviar o *qubit* para Bob, Bob aplica uma série de operações para recuperar os bits clássicos. As etapas que Bob segue são:

1. **Aplicação da Porta** $CNOT$: Bob aplica a porta $CNOT$, onde o *qubit* de Alice (controle) é usado para controlar a operação sobre o *qubit* de Bob (alvo).
2. **Aplicação da Porta Hadamard** H: Em seguida, Bob aplica a porta Hadamard H no *qubit* de Alice (controle). Isso coloca o *qubit* de Alice em uma superposição.
3. **Medição do *Qubit* de Alice**: Finalmente, Bob mede o *qubit* de Alice. O resultado dessa medição é a combinação dos bits clássicos enviados por Alice.

4.4 Circuito do Protocolo



Legenda:

- H: Porta de Hadamard.
- $CNOT$: Controle no qubit de Alice, alvo no de Bob.
- U_{ij}: Operações aplicadas por Alice para codificar os dois bits $ij \in \{00, 01, 10, 11\}$:
  - $00 \rightarrow I$
  - $01 \rightarrow X$
  - $10 \rightarrow Z$
  - $11 \rightarrow XZ$
- As últimas $CNOT$ e H são usadas por Bob para decodificar os bits.
- As medidas finais retornam os bits i e j.

4.4.1 Exemplo Prático

Caso 1: Alice deseja enviar 00

- Alice não aplica nenhuma operação.
- Estado permanece:

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{4.1}$$

- Bob aplica uma porta CNOT (controle: qubit de Alice)

$$\text{CNOT}(|00\rangle) = |00\rangle\,, \quad \text{CNOT}(|11\rangle) = |10\rangle\,,$$

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle). \tag{4.2}$$

- Bob aplica Hadamard no primeiro qubit

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

$$|\Psi'\rangle = H \otimes I\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)\right) = \frac{1}{2}[(|0\rangle + |1\rangle)\,|0\rangle + (|0\rangle - |1\rangle)\,|0\rangle]. = |00\rangle \tag{4.3}$$

- **Bob mede:** $|00\rangle$

Caso 2: Alice deseja enviar $|01\rangle$

- Alice aplica a porta X (NOT) no seu qubit

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

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle). \tag{4.4}$$

- Bob aplica CNOT

$$\text{CNOT}(|10\rangle) = |11\rangle\,, \quad \text{CNOT}(|01\rangle) = |01\rangle\,,$$

$$|\Psi'\rangle \frac{1}{\sqrt{2}}(|11\rangle + |01\rangle). \tag{4.5}$$

- Bob aplica Hadamard no primeiro qubit

$$ket\Psi' = H \otimes I\left(\frac{1}{\sqrt{2}}(|11\rangle + |01\rangle)\right) = \frac{1}{2}[(|0\rangle - |1\rangle)|1\rangle + (|0\rangle + |1\rangle)|1\rangle] = |01\rangle\,. \tag{4.6}$$

- **Bob mede:** $|01\rangle$

Caso 3: Alice deseja enviar $|10\rangle$

- Alice aplica a porta Z

$$Z|0\rangle = |0\rangle\,, \quad Z|1\rangle = -|1\rangle\,,$$

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle). \tag{4.7}$$

- Bob aplica CNOT

$$\text{CNOT}(|00\rangle) = |00\rangle\,, \quad \text{CNOT}(|11\rangle) = |10\rangle\,,$$

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |10\rangle). \tag{4.8}$$

- Bob aplica Hadamard

$$|\Psi'\rangle = \frac{1}{2}[(|0\rangle + |1\rangle)|0\rangle - (|0\rangle - |1\rangle)|0\rangle] = |10\rangle\,. \tag{4.9}$$

- **Bob mede:** $|10\rangle$

Caso 4: Alice deseja enviar $|11\rangle$

- Alice aplica X seguido de Z

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|10\rangle - |01\rangle). \tag{4.10}$$

- Bob aplica CNOT

$$\text{CNOT}(|10\rangle) = |11\rangle\,, \quad \text{CNOT}(|01\rangle) = |01\rangle\,,$$

$$|\Psi'\rangle = \frac{1}{\sqrt{2}}(|11\rangle - |01\rangle). \tag{4.11}$$

- Bob aplica Hadamard

$$|\Psi'\rangle = \frac{1}{2}[(|0\rangle - |1\rangle)|1\rangle - (|0\rangle + |1\rangle)|1\rangle] = |11\rangle\,. \tag{4.12}$$

- **Bob mede:** $|11\rangle$.

4.4.2 Implementação do Protocolo em Python

```
# Importando as bibliotecas necessarias
from qiskit import QuantumCircuit # Removed execute from import
from qiskit_aer import AerSimulator # Corrected import for Aer
# from qiskit.assembler import assemble # Added import for assemble
from qiskit.visualization import plot_histogram
import numpy as np
from IPython.display import display
import matplotlib.pyplot as plt # Added import for matplotlib.pyplot

# Funcao para Superdense Coding
def superdense_coding(bits: str):
    if bits not in ["00", "01", "10", "11"]:
        raise ValueError("Bits must be one of: '00', '01', '10', '11'")

    qc = QuantumCircuit(2, 2)

    # Criar estado emaranhado
    qc.h(0)
    qc.cx(0, 1)

    # Codificacao dos bits por Alice
    if bits == "01":
        qc.x(0)
    elif bits == "10":
        qc.z(0)
    elif bits == "11":
        qc.z(0)
        qc.x(0)

    # Decodificacao de Bob
    qc.cx(0, 1)
    qc.h(0)

    # Medidas - Adjusted to match output order
    qc.measure(0, 1) # Measure qubit 0 into classical bit 1
    qc.measure(1, 0) # Measure qubit 1 into classical bit 0

    # Simulador Aer
    simulator = AerSimulator()
    # qobj = assemble(qc, shots=1024)
    result = simulator.run(qc, shots=1024).result()
    counts = result.get_counts()

    print(f"Bits enviados por Alice: {bits}")
    print(f"Resultado da medicao de Bob: {counts}")


# Exemplo: enviar os bits "10"
superdense_coding("10")
```

4.4.3 Possíveis Erros e Desafios

1. Interrupções ou ruídos no canal de comunicação

Se o *qubit* enviado por Alice sofre interferência ou decaimento devido a ruídos no canal, a codificação superdensa pode falhar, e Bob pode não recuperar os bits corretamente.

2. Medições incorretas

Se Bob não aplicar as operações de maneira correta (por exemplo, errar na ordem de aplicação das portas $CNOT$ ou Hadamard), ele pode obter o valor errado dos bits.

3. Emaranhamento insuficiente

Se Alice e Bob não emaranharem os *qubits* de maneira correta no início (ou se houver perda de emaranhamento), o protocolo falhará, e a informação não será transmitida corretamente.

4.4.4 Conclusão

A codificação superdensa é uma poderosa aplicação do emaranhamento quântico para transmitir mais informação do que seria possível em um sistema clássico. Com apenas um *qubit*, é possível transmitir dois **bits clássicos**, desde que Alice e Bob sigam corretamente os passos descritos. A segurança e a eficiência desse protocolo dependem da integridade do emaranhamento e da precisão das operações quânticas.

4.5 Algoritmo de Deutsch

O **Algoritmo de Deutsch** constitui um dos exemplos mais elementares e didáticos da superioridade da computação quântica frente à clássica em determinadas tarefas. Seu objetivo é determinar, com apenas uma avaliação de uma função booleana, se esta é constante ou balanceada, utilizando os princípios da superposição e da interferência quântica.

4.5.1 Importância do Paralelismo Quântico

O **paralelismo quântico** é uma propriedade essencial que decorre da capacidade dos sistemas quânticos de se encontrarem em superposições lineares de estados. Considerando um sistema quântico preparado em uma superposição de todos os possíveis estados de entrada, a aplicação de uma operação unitária que implementa uma função f avalia simultaneamente $f(x)$ para todos os x na superposição. Matematicamente, para um registrador quântico com n qubits, o estado inicial pode ser escrito como

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle ,$$

e a aplicação do operador unitário U_f transforma esse estado em

$$U_f |\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |f(x)\rangle .$$

Dessa forma, todas as avaliações da função f sobre as entradas x são processadas em paralelo, em um único passo computacional. Essa propriedade é fundamental para o ganho de eficiência nos algoritmos quânticos, como o algoritmo de Deutsch, que explora o paralelismo para determinar a natureza da função f com apenas uma única chamada ao operador U_f, enquanto que um computador clássico exigiria avaliar f para múltiplos valores de x.

4.5.2 Teoria

Seja $f : \{0,1\} \to \{0,1\}$ uma função booleana. Há duas possibilidades:

- f é **constante** se $f(0) = f(1) \in \{0,1\}$,
- f é **balanceada** se $f(0) \neq f(1)$.

A computação clássica requer duas consultas para determinar a natureza de f. O algoritmo de Deutsch resolve esse problema com uma única chamada à função, modelada como uma operação unitária U_f.

4.5.3 Aritmética Modular

A aritmética modular é fundamental no algoritmo de Shor. Dados dois números inteiros a e b, a operação $a \mod b$ calcula o resto da divisão de a por b. Por exemplo

$$10 \mod 3 = 1.$$

Pode-se dizer que a é congruente a b módulo n se $a \equiv b \mod n$, ou seja, se n divide $a - b$.

4.5.4 Operador U_f

O operador quântico associado à função f é definido da seguinte forma:

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

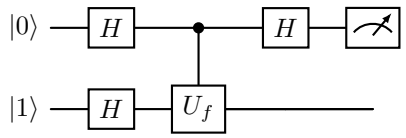
onde $\oplus$ denota a soma módulo 2 (XOR).

4.5.5 Descrição do Circuito Quântico

O circuito é composto por dois qubits:

- O primeiro qubit serve como entrada da função f,
- O segundo qubit atua como auxiliar, permitindo a implementação de U_f.

O circuito realiza as seguintes operações:



4.5.6 Evolução do Estado Quântico

1. Estado Inicial O sistema é preparado no estado

$$|\psi_0\rangle = |0\rangle \otimes |1\rangle = |01\rangle .$$

2. Aplicação da Porta de Hadamard em Ambos os Qubits Aplica-se a porta de Hadamard H a cada qubit

$$\begin{aligned} |\psi_1\rangle &= (H \otimes H)\,|01\rangle = H\,|0\rangle \otimes H\,|1\rangle\,, \\ &= \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right), \\ &= \frac{1}{2}\left(|00\rangle - |01\rangle + |10\rangle - |11\rangle\right). \end{aligned}$$

3. Aplicação do Operador U_f Aplicamos U_f ao estado $|\psi_1\rangle$. Para cada termo $|x\rangle\,|y\rangle$, temos

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

A operação resulta em

$$|\psi_2\rangle = \frac{1}{2}\left(|0\rangle\,|0 \oplus f(0)\rangle - |0\rangle\,|1 \oplus f(0)\rangle + |1\rangle\,|0 \oplus f(1)\rangle - |1\rangle\,|1 \oplus f(1)\rangle\right).$$

Vamos analisar os dois casos possíveis para f.

4.5.7 Caso 1: $f(0) = f(1) = 0$ (Função Constante)

Neste caso, as somas $y \oplus f(x) = y$, e o operador U_f não altera o segundo qubit. Assim:

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

4. Aplicação de Hadamard no Primeiro Qubit Agora aplicamos $H \otimes \mathbb{I}$ sobre $|\psi_2\rangle$. Devemos aplicar H ao primeiro qubit de cada termo

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{2}\left(H\,|0\rangle\,|0\rangle - H\,|0\rangle\,|1\rangle + H\,|1\rangle\,|0\rangle - H\,|1\rangle\,|1\rangle\right), \\ &= \frac{1}{2\sqrt{2}}\left[(|0\rangle + |1\rangle)\,|0\rangle - (|0\rangle + |1\rangle)\,|1\rangle + (|0\rangle - |1\rangle)\,|0\rangle - (|0\rangle - |1\rangle)\,|1\rangle\right]. \end{aligned}$$

Expandindo os produtos:

$$
\begin{aligned}
|\psi_3\rangle &= \frac{1}{2\sqrt{2}} \left[|00\rangle + |10\rangle - |01\rangle - |11\rangle + |00\rangle - |10\rangle - |01\rangle + |11\rangle \right], \\
&= \frac{1}{2\sqrt{2}} \left[2\,|00\rangle - 2\,|01\rangle \right], \\
&= \frac{1}{\sqrt{2}} \left(|00\rangle - |01\rangle \right).
\end{aligned}
$$

5. Medição A medição do primeiro qubit no estado $|\psi_3\rangle$ resultará com probabilidade 1 no valor $|0\rangle$. Concluímos, portanto, que a função f é constante.

4.5.8 Caso 2: $f(0) = 0, f(1) = 1$ (Função Balanceada)

Neste cenário, aplicando U_f ao estado $|\psi_1\rangle$, temos:

$$
|\psi_2\rangle = \frac{1}{2} \left(|0\rangle\,|0\rangle - |0\rangle\,|1\rangle + |1\rangle\,|1\rangle - |1\rangle\,|0\rangle \right).
$$

Reorganizando os termos:

$$
|\psi_2\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle).
$$

4. Aplicação da Porta Hadamard no Primeiro Qubit Aplicando $H \otimes \mathbb{I}$, temos:

$$
|\psi_3\rangle = \frac{1}{2} \left(H\,|0\rangle\,|0\rangle - H\,|0\rangle\,|1\rangle - H\,|1\rangle\,|0\rangle + H\,|1\rangle\,|1\rangle \right)
$$

Usando as transformações já vistas, obtemos:

$$
\begin{aligned}
|\psi_3\rangle &= \frac{1}{2\sqrt{2}} \left[(|0\rangle + |1\rangle)\,|0\rangle - (|0\rangle + |1\rangle)\,|1\rangle - (|0\rangle - |1\rangle)\,|0\rangle + (|0\rangle - |1\rangle)\,|1\rangle \right] \\
&= \frac{1}{2\sqrt{2}} \left[|00\rangle + |10\rangle - |01\rangle - |11\rangle - |00\rangle + |10\rangle + |01\rangle - |11\rangle \right] \\
&= \frac{1}{2\sqrt{2}} \left[2\,|10\rangle - 2\,|11\rangle \right] \\
&= \frac{1}{\sqrt{2}} (|10\rangle - |11\rangle)
\end{aligned}
$$

5. Medição Neste estado final, a medição do primeiro qubit resulta com probabilidade 1 no estado $|1\rangle$. Assim, determinamos que a função f é balanceada.

4.5.9 Implementação do Protocolo em Python

```
    from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

def deutsch_algorithm(U_f):
    # Inicializa um circuito com 2 qubits e 1 bit cl ssico para leitura
    qc = QuantumCircuit(2, 1)

    # Prepara o estado inicial |0>|1>
    qc.x(1)  # Aplica X no segundo qubit para preparar |1>

    # Aplica Hadamard em ambos os qubits
    qc.h(0)
    qc.h(1)

    # Aplica o operador Uf
    qc.append(U_f, [0,1])

    # Aplica Hadamard no primeiro qubit
    qc.h(0)
```

```
    # Mede o primeiro qubit
    qc.measure(0, 0)

    # Executa o circuito no simulador
    backend = Aer.get_backend('qasm_simulator')
    job = execute(qc, backend, shots=1024)
    result = job.result()
    counts = result.get_counts()

    # Mostra o circuito e o resultado
    print(qc.draw())
    print("Resultados da medicao:", counts)
    plot_histogram(counts)
    plt.show()

# Define Uf para funcao constante f(x)=0
def constant_Uf():
    qc = QuantumCircuit(2)
    # Para f(x)=0, Uf    a identidade, nao altera os qubits
    return qc.to_gate(label='Uf_const')

# Define Uf para funcao balanceada f(0)=0, f(1)=1
def balanced_Uf():
    qc = QuantumCircuit(2)
    # Uf implementa CNOT: |x>|y> -> |x>|y XOR x>
    qc.cx(0,1)
    return qc.to_gate(label='Uf_bal')

# Executar para funcao constante
print("Executando para funcao constante f(x)=0")
deutsch_algorithm(constant_Uf())

# Executar para funcao balanceada
print("Executando para funcao balanceada f(0)=0, f(1)=1")
deutsch_algorithm(balanced_Uf())
```

4.5.10 Possíveis Erros e Desafios

Embora o algoritmo de Deutsch seja conceitualmente simples, a implementação prática pode envolver alguns desafios. A seguir estão listados alguns dos erros e dificuldades mais comuns ao tentar implementar esse algoritmo, seja em simulações quânticas ou em um computador quântico real:

1. Erro na implementação da operação U_f

A operação U_f é a parte central do algoritmo e depende da definição da função f. Um erro comum é a implementação incorreta dessa operação, resultando em um estado final incorreto. Como U_f depende diretamente da função f, erros podem surgir se o código que implementa f não for preciso. Isso pode ocorrer devido a um mapeamento incorreto dos valores de entrada ou problemas com a maneira como o $f(x)$ é aplicado aos qubits.

2. Difusão de fases

Quando se trabalha com estados quânticos superpostos, é essencial que as fases dos diferentes termos no estado quântico sejam mantidas corretamente. Um erro comum é a perda de fase durante a execução de operações quânticas, o que pode ocorrer ao não aplicar corretamente as portas de fase, como a porta Hadamard ou a operação U_f. Isso pode resultar em uma interferência errada durante a medição, levando a resultados incorretos.

3. Falta de superposição inicial adequada

O algoritmo de Deutsch depende da criação de uma superposição inicial do qubit de entrada. Um erro comum é não aplicar corretamente a porta Hadamard no início, o que pode impedir que a superposição seja criada corretamente. Sem a superposição, o algoritmo não será capaz de explorar a paralelização quântica, e a vantagem quântica será perdida.

4. Gerenciamento de fases globais

Embora as fases globais (como -1) não afetem o resultado final da computação, em algumas implementações de algoritmos quânticos, incluindo o de Deutsch, erros no gerenciamento de fases globais podem levar a confusão. Um erro na manipulação de fases globais pode causar um comportamento inesperado nos estados finais, especialmente quando múltiplas portas Hadamard são aplicadas. A fase global não altera o resultado final da medição, mas pode dificultar o diagnóstico de problemas se não for bem compreendida.

5. Ruído e decoerência em computadores quânticos reais

Em simuladores quânticos, o algoritmo de Deutsch pode ser executado sem problemas. No entanto, ao implementar o algoritmo em um computador quântico real, problemas como ruído quântico e decoerência podem afetar os resultados. O ruído quântico pode resultar em erros na execução de portas quânticas, enquanto a decoerência pode fazer com que a superposição quântica se desfaça antes da medição. Isso pode levar a uma medição incorreta, o que torna essencial o uso de técnicas de correção de erros quânticos.

6. Interferência e conflitos de interpretação de resultados

Como a medição no computador quântico depende de interferência entre os estados, pode haver casos em que a interferência construtiva ou destrutiva não ocorre como esperado. Isso pode ser um desafio para entender a razão pela qual o algoritmo não funciona corretamente em alguns casos. Em implementações mais complexas, como quando há múltiplos qubits, o controle de interferência torna-se mais difícil e os resultados podem ser misturados devido a interações imprevistas entre os qubits.

7. Entendimento e implementação de funções balanceadas

Para o sucesso do algoritmo de Deutsch, é fundamental entender a estrutura da função f. O desafio de distinguir entre uma função balanceada e uma constante se dá justamente porque essas funções podem ter comportamentos complexos. Se o comportamento da função não for corretamente mapeado e implementado no circuito quântico, o algoritmo pode falhar ao identificar corretamente o tipo de função.

8. Implementação de funções não triviais

A função f no algoritmo de Deutsch pode ser simples ou mais complexa dependendo do problema em questão. Em alguns casos, é necessário implementar funções não triviais que envolvem mais de um qubit ou interações mais complicadas. Nesses casos, é importante verificar se todas as operações quânticas, como U_f, estão sendo corretamente realizadas.

4.5.11 Conclusão

O algoritmo de Deutsch é um dos primeiros algoritmos que ilustram o poder da computação quântica. Ele é capaz de determinar se uma função binária é constante ou balanceada com apenas uma execução, algo que seria impossível com um algoritmo clássico sem realizar duas consultas à função. A utilização de superposição e emaranhamento quântico são as chaves para esse ganho de eficiência.

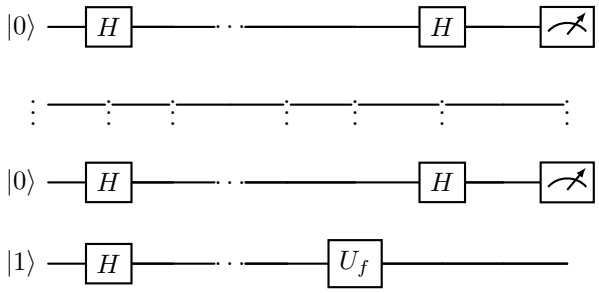
4.6 Algoritmo de Deutsch-Jozsa

O algoritmo de Deutsch-Jozsa é uma generalização do algoritmo de Deutsch, que busca determinar se uma função $f : \{0,1\}^n \to \{0,1\}$ é **constante** ou **balanceada**. Uma função é considerada:

- **Constante**: quando $f(x)$ retorna o mesmo valor para todas as entradas x;
- **Balanceada**: quando $f(x)$ retorna 0 para metade das entradas e 1 para a outra metade.

O algoritmo quântico resolve esse problema em apenas **uma única avaliação** da função $f(x)$ com o auxílio de um oráculo. Em contrapartida, o algoritmo clássico requer, no pior caso, $2^{n-1} + 1$ avaliações para garantir a mesma resposta.

4.6.1 Circuito Quântico



4.6.2 Etapas do Algoritmo

O algoritmo de Deutsch-Jozsa segue os seguintes passos:

1. Inicialização

Começamos com um registrador de n-qubits no estado $|0\rangle^{\otimes n}$ e um qubit auxiliar no estado $|1\rangle$

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

Aqui, $|0\rangle^{\otimes n}$ representa n qubits no estado $|0\rangle$, ou seja,

$$|0\rangle^{\otimes n} = |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle \quad (n \text{ vezes}).$$

Por exemplo, para $n = 2$

$$|0\rangle^{\otimes 2} = |0\rangle \otimes |0\rangle.$$

2. Superposição Inicial

Aplicamos a porta de Hadamard (H) em todos os n-qubits do registrador principal e no qubit auxiliar. Após a operação, o estado é transformado em

$$|\Psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Aqui, o estado $|x\rangle$ representa a superposição de todos os 2^n estados possíveis dos n-qubits.

3. Implementação do Oráculo

O oráculo U_f é uma operação unitária que codifica a função $f(x)$ no estado quântico. Ele é definido como

$$U_f : |x\rangle \otimes |y\rangle \to |x\rangle \otimes |y \oplus f(x)\rangle.$$

Após aplicar o oráculo, o estado se torna

$$|\Psi_2\rangle = U_f|\Psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes \frac{1}{\sqrt{2}} \left(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle\right).$$

Isso pode ser reescrito como

$$|\Psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle \otimes \frac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle\right).$$

A informação sobre $f(x)$ agora está codificada na fase $(-1)^{f(x)}$ associada a cada estado $|x\rangle$.

4. Transformada de Hadamard Final

Aplicamos novamente a porta de Hadamard em todos os n-qubits do registrador principal. O estado resultante é

$$|\Psi_3\rangle = H^{\otimes n} \otimes \mathbb{I}|\Psi_2\rangle = H^{\otimes n} \left(\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle \right).$$

Este passo amplifica ou cancela as amplitudes dos estados $|x\rangle$, dependendo da natureza da função $f(x)$.

5. Medição

Por fim, medimos o registrador principal. O resultado será

$$\text{Resultado} = \begin{cases} |0\rangle^{\otimes n}, & \text{se } f(x) \text{ é constante,} \\ \text{Outro estado,} & \text{se } f(x) \text{ é balanceada.} \end{cases}$$

Exemplo

Seja uma função $f : \{0,1\}^2 \to \{0,1\}$ definida como:

$$f(00) = 0, \quad f(01) = 1, \quad f(10) = 1, \quad f(11) = 0,$$

ou seja, uma função **balanceada**. Vamos analisar o funcionamento do algoritmo passo a passo.

1. Estado Inicial Inicializamos três qubits

$$|\psi_0\rangle = |00\rangle \otimes |1\rangle.$$

2. Aplicação das portas de Hadamard Aplicamos Hadamard aos três qubits

$$|\psi_1\rangle = (H^{\otimes 2} \otimes H)|\psi_0\rangle = \left(\frac{1}{2} \sum_{x=0}^{3} |x\rangle \right) \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right).$$

3. Aplicação do Oráculo U_f O Oracle atua da seguinte forma

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle.$$

Quando aplicado ao estado $|\psi_1\rangle$, temos

$$|\psi_2\rangle = \frac{1}{2\sqrt{2}} \sum_{x=0}^{3} (-1)^{f(x)}|x\rangle \left(|0\rangle - |1\rangle\right),$$

ou seja, o segundo registrador é fatorável e podemos omiti-lo nas análises posteriores

$$|\psi_2\rangle = \frac{1}{2} \sum_{x=0}^{3} (-1)^{f(x)}|x\rangle.$$

Substituindo os valores de $f(x)$

$$|\psi_2\rangle = \frac{1}{2} \left[(-1)^0|00\rangle + (-1)^1|01\rangle + (-1)^1|10\rangle + (-1)^0|11\rangle \right],$$

$$|\psi_2\rangle = \frac{1}{2} \left[|00\rangle - |01\rangle - |10\rangle + |11\rangle \right].$$

4. Aplicação de Hadamard novamente Aplicamos Hadamard novamente aos dois primeiros qubits

$$|\psi_3\rangle = H^{\otimes 2}|\psi_2\rangle.$$

Utilizamos a identidade

$$H^{\otimes 2}|x\rangle = \frac{1}{2}\sum_{z=0}^{3}(-1)^{x\cdot z}|z\rangle,$$

onde $x \cdot z$ é o produto interno (bitwise XOR seguido de soma módulo 2).
Então

$$|\psi_3\rangle = \frac{1}{2}\sum_{x}(-1)^{f(x)}H^{\otimes 2}|x\rangle = \frac{1}{4}\sum_{x,z}(-1)^{f(x)}(-1)^{x\cdot z}|z\rangle.$$

Agrupando os coeficientes de $|z\rangle$, temos

$$|\psi_3\rangle = \sum_{z}\left(\frac{1}{4}\sum_{x}(-1)^{f(x)+x\cdot z}\right)|z\rangle.$$

Vamos calcular os coeficientes explicitamente
- Para $z = 00$

$$\sum_{x}(-1)^{f(x)}(-1)^{x\cdot 00} = (-1)^0 + (-1)^1 + (-1)^1 + (-1)^0 = 1 - 1 - 1 + 1 = 0.$$

- Para $z = 01$

$$x \cdot z = x_1x_2 \cdot 01 = x_2, \text{ então } (-1)^{f(x)+x_2}.$$
$$\Rightarrow (-1)^{0+0} + (-1)^{1+1} + (-1)^{1+0} + (-1)^{0+1} = 1 + 1 - 1 - 1 = 0.$$

Idem para $z = 10$ e $z = 11$: todos os coeficientes são zero.
Logo:

$$|\psi_3\rangle = 0.$$

Na prática, devido ao cancelamento total de $|00\rangle$, temos que **a medição não retornará $|00\rangle$**, evidenciando que $f(x)$ não é constante.

5. Medição A medição do primeiro registrador nunca resultará em $|00\rangle$, o que nos leva a concluir que $f(x)$ é **balanceada**, como esperado.

4.6.3 Implemetação do Protocólo em Python

```
    from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram
import random

def deutsch_jozsa_oracle(n, tipo='balanceado'):
    """Cria um oraculo constante ou balanceado para n bits"""
    oracle = QuantumCircuit(n + 1)

    if tipo == 'constante':
        # f(x) = 0 ou f(x) = 1 para todo x
        if random.choice([True, False]):
            pass  # f(x) = 0 => nao altera nada
        else:
            oracle.x(n)  # f(x) = 1 => aplica X no qubit de saida
    elif tipo == 'balanceado':
        # f(x) e balanceada: metade dos x da 0, metade da 1
        for i in range(n):
            if random.choice([True, False]):
                oracle.cx(i, n)
    else:
        raise ValueError("Tipo deve ser 'constante' ou 'balanceado'")
```

```
    return oracle

def deutsch_jozsa_algorithm(n, tipo='balanceado'):
    """Executa o algoritmo Deutsch-Jozsa para n bits"""
    circuit = QuantumCircuit(n + 1, n)

    # Inicializa o ultimo qubit em |1>
    circuit.x(n)
    # Aplica Hadamard em todos os qubits
    circuit.h(range(n + 1))

    # Adiciona o oraculo
    oracle = deutsch_jozsa_oracle(n, tipo)
    circuit.append(oracle.to_gate(), range(n + 1))

    # Aplica Hadamard nos primeiros n qubits
    circuit.h(range(n))

    # Mede os primeiros n qubits
    circuit.measure(range(n), range(n))

    # Executa o circuito
    backend = Aer.get_backend('qasm_simulator')
    result = execute(circuit, backend, shots=1).result()
    counts = result.get_counts()

    return circuit, counts

# Exemplo de uso:
n = 4
tipo = 'balanceado'  # ou 'constante'
circuito, resultado = deutsch_jozsa_algorithm(n, tipo)

# Mostra o circuito e o resultado
print(circuito.draw())
print("Resultado da medicao:", resultado)
```

4.6.4 Conclusão

O algoritmo de Deutsch-Jozsa é um dos exemplos mais simples e elegantes do poder da computação quântica, demonstrando a vantagem exponencial que ela oferece em comparação com algoritmos clássicos. Ele fornece uma base para algoritmos quânticos mais avançados, como o de Grover e Shor.

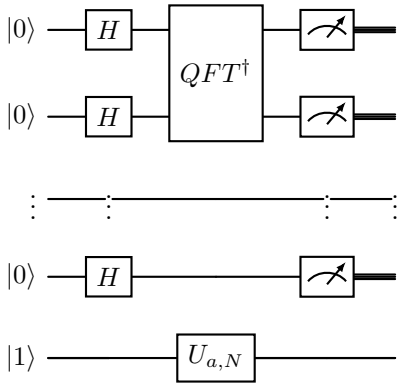
4.7 Algoritmo de Shor

O algoritmo de Shor é um algoritmo quântico para fatoração de números inteiros que oferece uma aceleração exponencial em relação aos melhores algoritmos clássicos conhecidos. Ele é particularmente importante por sua capacidade de quebrar sistemas criptográficos baseados em RSA.

4.7.1 Problema da Fatoração

Dado um número composto N (não primo), queremos encontrar seus fatores primos p e q tal que $N = p \times q$.

4.7.2 Circuito Quântico



4.7.3 Etapas do Algoritmo

1. Escolha do Número Aleatório

Escolhemos um inteiro a tal que $1 < a < N$ e $\text{mdc}(a, N) = 1$. Se $\text{mdc}(a, N) \neq 1$, já encontramos um fator.

2. Registro Quântico

Preparamos dois registros quânticos:

- Registro de controle com $t = 2\log_2 N$ qubits no estado $|0\rangle$
- Registro de trabalho com $\log_2 N$ qubits no estado $|1\rangle$

O estado inicial é:

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

3. Superposição

Aplicamos a porta Hadamard no registro de controle:

$$|\psi_1\rangle = \left(\frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle\right) \otimes |1\rangle.$$

4. Aplicação do Oráculo Modular

Aplicamos a operação $U_{a,N}|x\rangle|y\rangle = |x\rangle|y \cdot a^x \mod N\rangle$

$$|\psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle|a^x \mod N\rangle.$$

5. Transformada de Fourier Quântica

Aplicamos QFT no registro de controle

$$|\psi_3\rangle = \frac{1}{2^t} \sum_{x=0}^{2^t-1} \sum_{k=0}^{2^t-1} e^{2\pi i x k/2^t} |k\rangle|a^x \mod N\rangle.$$

6. Medição

Medimos o registro de controle, obtendo um valor k. Com alta probabilidade, $k/2^t$ será próximo de c/r onde r é o período que procuramos.

7. Frações Contínuas

Usamos o algoritmo de frações contínuas para encontrar r a partir da aproximação $k/2^t \approx c/r$.

8. Obtenção dos Fatores

Se r é par e $a^{r/2} \not\equiv -1 \mod N$, calculamos:

$$\text{mdc}(a^{r/2} \pm 1, N).$$

para obter os fatores de N.

Exemplo Numérico: Fatorando N = 15

Vamos fatorar $N = 15$ usando $a = 7$

1. $\text{mdc}(7, 15) = 1$ (sem fatores triviais)
2. Encontramos o período r tal que $7^r \equiv 1 \mod 15$

$$\begin{aligned} 7^1 &\equiv 7 \mod 15, \\ 7^2 &\equiv 4 \mod 15, \\ 7^3 &\equiv 13 \mod 15, \\ 7^4 &\equiv 1 \mod 15. \quad (\text{período } r = 4) \end{aligned}$$

3. Como $r = 4$ é par, continuamos
4. $7^{4/2} = 7^2 = 49 \equiv 4 \mod 15$.
5. $4 \not\equiv -1 \mod 15$, então calculamos

$$\begin{aligned} \text{mdc}(4 - 1, 15) &= \text{mdc}(3, 15) = 3, \\ \text{mdc}(4 + 1, 15) &= \text{mdc}(5, 15) = 5. \end{aligned}$$

6. Fatores encontrados: 3 e 5.

4.7.4 Implementação em Python

```
from qiskit import QuantumCircuit, Aer, execute
from qiskit.algorithms import Shor
from qiskit.utils import QuantumInstance
import math

def shor_algorithm(N, a=None):
    # Configura  o do backend qu ntico
    backend = Aer.get_backend('qasm_simulator')
    quantum_instance = QuantumInstance(backend, shots=1024)

    # Execu  o do algoritmo de Shor
    shor = Shor(quantum_instance=quantum_instance)
    result = shor.factor(N, a=a)

    return result

# Exemplo de uso
N = 15
result = shor_algorithm(N)
print(f"Fatores␣de␣{N}:␣{result.factors}")

# Vers o simplificada para entender a matem tica
def shor_classical(N):
    if N % 2 == 0:
        return 2
    if math.isqrt(N)**2 == N:
        return math.isqrt(N)
```

```
    while True:
        a = random.randint(2, N-1)
        k = math.gcd(a, N)
        if k != 1:
            return k

        # Encontrar o per odo r (parte qu ntica simulada classicamente)
        for r in range(1, N):
            if pow(a, r, N) == 1:
                break

        if r % 2 != 0:
            continue

        y = pow(a, r//2, N)
        if y == N-1:  # y      -1 mod N
            continue

        p = math.gcd(y - 1, N)
        q = math.gcd(y + 1, N)
        if p != 1 and q != 1:
            return (p, q)

print(f"Fatores␣(simula  o␣cl ssica):␣{shor_classical(15)}")
```

4.7.5 Dificuldades e Desafios do Algoritmo de Shor

Embora o algoritmo de Shor seja uma solução promissora para o problema da fatoração de números inteiros em computadores quânticos, sua implementação prática enfrenta diversos desafios. Abaixo, destacamos as principais dificuldades:

Requisitos de hardware quântico

O algoritmo de Shor exige um número significativo de qubits estáveis e de alta qualidade para realizar cálculos com precisão. Atualmente, os computadores quânticos disponíveis têm limitações em termos de número de qubits e fidelidade das operações, dificultando a execução de instâncias grandes do algoritmo.

Correção de erros quânticos

Os sistemas quânticos são extremamente suscetíveis a ruídos e erros de decoerência, o que torna indispensável o uso de técnicas avançadas de correção de erros. Isso aumenta a complexidade do hardware e do tempo necessário para executar o algoritmo.

Estimativa precisa do período

A eficiência do algoritmo depende da capacidade de estimar com precisão o período r da função modular exponencial. Flutuações ou imprecisões na estimativa do período podem levar a resultados incorretos, forçando a repetição de etapas do algoritmo.

Conversão para formato clássico

Os resultados produzidos pelo computador quântico precisam ser interpretados em um contexto clássico. Traduzir as informações quânticas para um formato utilizável de forma eficiente pode ser um desafio técnico.

Escalabilidade

Enquanto o algoritmo é eficiente em teoria, sua aplicação prática em números muito grandes (como os usados em criptografia RSA) requer computadores quânticos com recursos ainda inexistentes. Escalar o algoritmo para esses níveis permanece um desafio significativo.

Limitações teóricas

Apesar de o algoritmo de Shor ser eficiente para a fatoração de números inteiros, sua eficácia em outros contextos, como problemas criptográficos alternativos que não dependem da fatoração, é limitada. Isso exige o desenvolvimento de novos algoritmos quânticos para outros cenários de aplicação.

Desafios de implementação experimental

Projetar circuitos quânticos que implementem o algoritmo de Shor, especialmente a operação modular exponencial e a Transformada de Fourier Quântica, com eficiência e baixo consumo de recursos, é tecnicamente desafiador. Cada passo precisa ser cuidadosamente ajustado para evitar interferências. Essas dificuldades refletem o estado atual da tecnologia quântica e representam áreas de pesquisa ativa na computação quântica. Apesar dos desafios, os avanços contínuos em hardware e algoritmos prometem superar muitos desses obstáculos no futuro.

4.7.6 Possíveis erros e cuidados

A implementação do algoritmo de Shor envolve diversas etapas críticas que podem ser suscetíveis a erros. Nesta seção, destacamos os principais pontos de atenção para evitar falhas e garantir a precisão dos resultados.

Escolha do número a

A escolha de a no intervalo $1 < a < N$ é um passo fundamental. É necessário garantir que o máximo divisor comum (MDC) de a e N seja igual a 1. Caso contrário, o algoritmo falha ao determinar os fatores de N e deve ser reiniciado.

Estimativa do período (r)

A precisão na estimativa do período r da função modular exponencial é crucial. Um valor incorreto de r pode levar a fatores errados ou até mesmo a falha completa na obtenção dos fatores primos. Recomenda-se repetir o processo caso o valor estimado de r seja ímpar ou não satisfaça as condições do algoritmo.

Ruídos e erros quânticos

Os sistemas quânticos são extremamente suscetíveis a ruídos, o que pode corromper o estado quântico durante a execução do algoritmo. É fundamental garantir que o ambiente do computador quântico seja o mais estável possível e que sejam utilizadas técnicas de correção de erros quânticos.

Fatoração de diferença de quadrados

Ao fatorar $a^r - 1$ como $(a^{r/2} - 1)(a^{r/2} + 1)$, erros nos cálculos ou arredondamentos podem comprometer a obtenção dos fatores corretos. Verifique sempre os resultados usando o MDC para confirmar se os valores encontrados são divisores de N.

Dependência do número de qubits

O algoritmo exige um número elevado de qubits para representar estados quânticos de forma precisa. Uma quantidade insuficiente de qubits pode limitar o tamanho dos números N que podem ser fatorados e introduzir erros nos cálculos.

Armazenamento e processamento de resultados

Os resultados obtidos pelo algoritmo devem ser convertidos corretamente para o domínio clássico. Erros nesse processo podem levar a interpretações erradas dos valores estimados.

Limitações experimentais

Implementações experimentais do algoritmo podem enfrentar limitações práticas devido à fidelidade das portas quânticas, tempos de coerência insuficientes ou erros de inicialização. Cuidados devem ser tomados para mitigar essas limitações por meio de ajustes no hardware e nos algoritmos.

Repetição do algoritmo

Em casos onde os valores obtidos para r ou os fatores primos não são válidos, é necessário reiniciar o algoritmo. Este comportamento não é um erro, mas uma característica inerente ao algoritmo de Shor, especialmente em números compostos com propriedades atípicas. Esses cuidados são essenciais para maximizar a probabilidade de sucesso ao implementar o algoritmo de Shor, especialmente em sistemas quânticos atuais, que ainda estão em desenvolvimento.

4.7.7 Conclusão

O algoritmo de Shor demonstra o poder da computação quântica para resolver problemas práticos importantes.Embora nossa implementação em Python seja simplificada, ela captura a essência do método. Computadores quânticos reais implementam a parte de encontrar o período usando propriedades quânticas como superposição e interferência através da Transformada de Fourier Quântica. Até o momento, o maior número fatorado experimentalmente usando uma implementação do algoritmo de Shor em um computador quântico real é **21**, como demonstrado por:

- Monz et al. (2016) usando um sistema de íons aprisionados com 5 qubits (*Realization of a scalable Shor algorithm*)
- Vandersypen et al. (2001) usando NMR (ressonância magnética nuclear) para fatorar 15
- Martin-López et al. (2012) usando fotônica para fatorar 21

Referências

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 10th Anniversary Edition, 2010.

[2] P. W. Shor, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994.

[3] E. Martín-López et al., *Experimental realization of Shor's quantum factoring algorithm using qubit recycling*, Nature Photonics 6, 773-776 (2012).

[4] T. Monz et al., *Realization of a scalable Shor algorithm*, Science 351, 1068-1070 (2016).

[5] IBM Quantum Learning, *Shor's Algorithm Implementation*, https://learning.quantum-computing.ibm.com/.