QuIIN
Quantum Industrial Innovation

Minicurso - Introdução à Computação Quântica

Henrique Sobrinho Ghizoni

henrique.ghizoni@fbter.org.br

Yan Chagas

yan.chagas@fieb.org.br

SENAI CIMATEC

EMBRAPII
Empresa Brasileira de Pesquisa e Inovação Industrial

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Qiskit

- Termo geral que se refere a uma coleção de software para executar programas em computadores quânticos, desenvolvido pela IBM.

- Quanto ao SDK Qiskit é um SDK de código aberto para trabalhar com computadores quânticos no nível de circuitos quânticos estendidos (estáticos, dinâmicos e programados), operadores e primitivos.

`qiskit.circuit` - Para inicializar e manipular registradores, circuitos, instruções, portas, parâmetros e objetos de fluxo de controle.

`qiskit.circuit.library`- Uma vasta gama de circuitos, instruções e portas - blocos de construção essenciais para computações quânticas baseadas em circuitos.

- Para mais informações: https://docs.quantum.ibm.com/guides

Qiskit

```
# Instalando o Qiskit

!pip install qiskit
```

```
# Para uma melhor visualização dos circuitos
!pip install pylatexenc
```

Qiskit

Para criar um circuito é necessário importar algumas bibliotecas:

```
from qiskit import QuantumCircuit
from qiskit.providers.basic_provider import BasicProvider
from qiskit.visualization import plot_histogram
from qiskit.visualization import circuit_drawer
import pylatexenc
```

Qiskit – Estrutura básica para o hands on

```
#criacao do circuito quantico com 1 qubit e 1 registrador clássico
qc = QuantumCircuit( 'quantidade de qubits', 'quantidade de registradores clássicos')

qc.'porta que deseja aplicar'('qubit onde a porta será aplicada')

qc.measure_all() #Mede o estado do qubit e armazena o resultado no registrador clássico.

qc.measure('qubit','registrador clássico') # É possível realizar a medição dessa forma também
```

Qiskit – Estrutura básica para o hands on

```
#utilizando o QASM Simulator
backend = BasicProvider().get_backend("basic_simulator") # Define um simulador quântico (basic_simulator do
# Caso deseje uma quantidade específica de execuções:
result = backend.run(qc, shots=2048).result()

result = backend.run(qc).result() # Executa o circuito e obtém o resultado da simulação
counts = result.get_counts() # O resultado é armazenado na variável counts, que contém a contagem dos estad

print(counts)
qc.draw() # Desenha o circuito quântico de forma visual.
```

Hands On: Implementando Portas Lógicas Quânticas

1) Implemente a porta X em um qubit inicialmente no estado $|0\rangle$ e meça o resultado.

2) Agora, implemente a porta X novamente, mas desta vez atuando em um qubit previamente preparado no estado $|1\rangle$. Meça o resultado.

3) Aplique a porta H em um qubit e realize a medição.

4) Repita os experimentos anteriores realizando apenas uma medição (shot = 1). O que você observa?

5) Agora, repita os experimentos utilizando 100 medições (shots = 100). Qual a sua conclusão sobre os resultados obtidos?

6) Represente graficamente os resultados das atividades 1, 2 e 3 na esfera de Bloch utilizando a ferramenta online: https://bloch.kherb.io/.

Hands On – Implementando Circuitos de 2 qubits

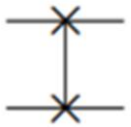
7. Crie um circuito quântico e implemente a porta CNOT.

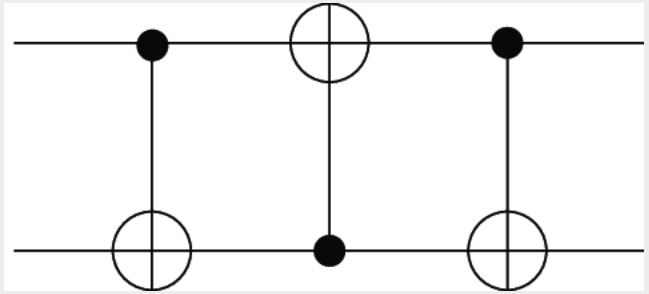
8. Crie um circuito quântico com a porta CNOT implementada, porém ela deve ativar quando o qubit de controle estiver no estado $|0\rangle$

[Discussão] Como representamos a CNOT na esfera de Bloch?

PORTA SWAP

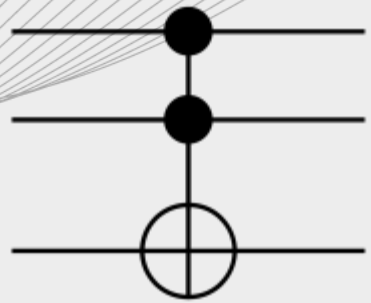
- Troca os estados de dois qubits,

Símbolo	Matriz	Comportamento
	$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\text{SWAP}\,\|00\rangle = \|00\rangle$ $\text{SWAP}\,\|01\rangle = \|10\rangle$ $\text{SWAP}\,\|10\rangle = \|01\rangle$ $\text{SWAP}\,\|11\rangle = \|11\rangle$ $\text{SWAP}\,\|ab\rangle = \|ba\rangle$ $a,b=0,1$

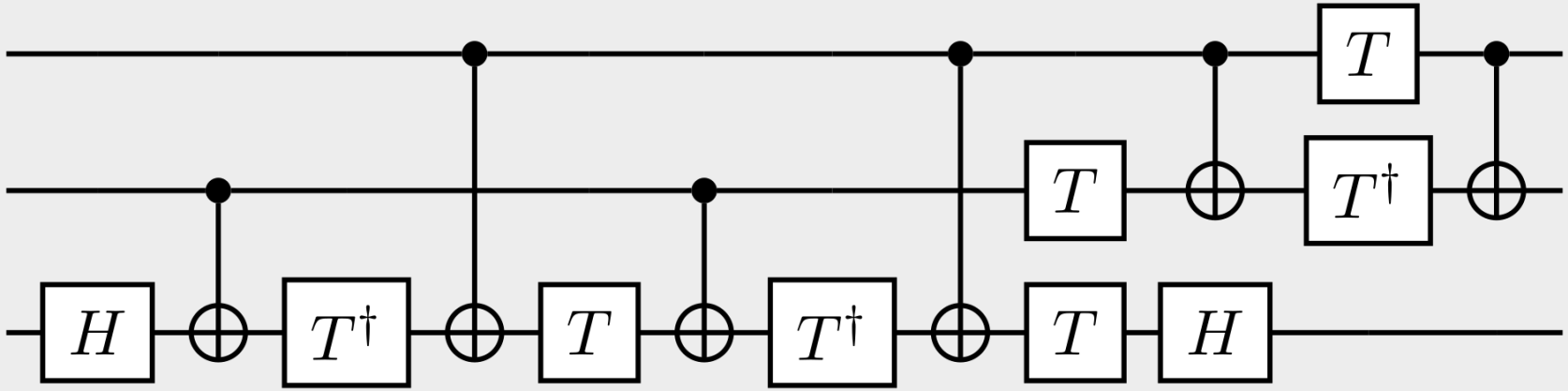


PORTA TOFFOLI

- Também conhecida como CCNOT, ativará a porta X no terceiro qubit mediante as condições dos dois qubits de controle serem atendidas.

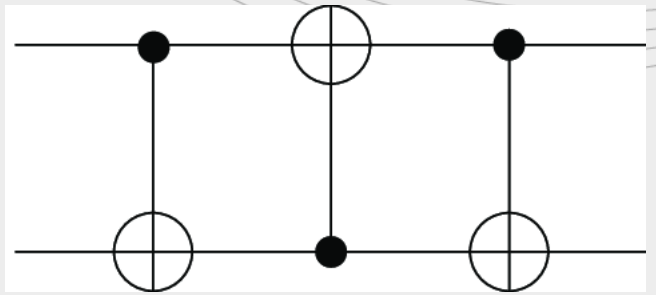


$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

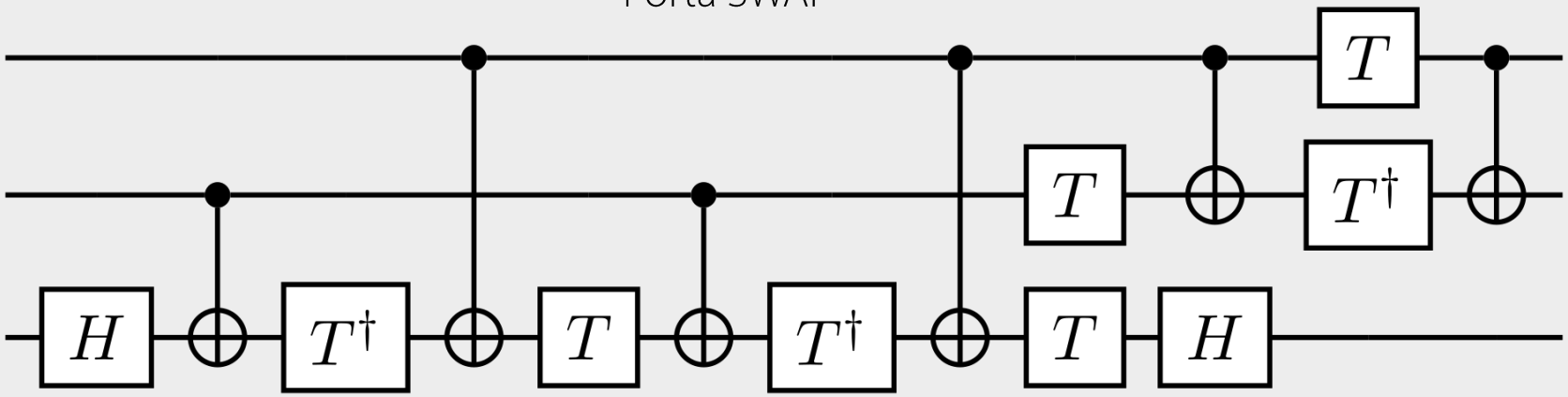


Hands On – Implementando Circuitos de 2 qubits

9. Crie um Circuito e Implemente a porta SWAP
10. Crie um Circuito e Implemente a porta Toffoli
11. [DESAFIO] Implemente as portas SWAP e Toffoli, através dos circuitos alternativos.



Porta SWAP



Porta Toffoli

Algoritmo de Deutsch

- Desenvolvido por David Deustch em 1985

- Primeiro algoritmo quântico a mostrar ganho computacional em relação a contrapartida clássica.

Problema de Deutsch: é um desafio da computação quântica onde temos um oráculo (uma "caixa preta") que implementa uma função $f: \{0,1\} \rightarrow \{0,1\}$ que recebe um número binário de n bits e retorna 0 ou 1.
A função pode ser ou **balanceada.**

O objetivo é descobrir, com o menor número possível de chamadas ao oráculo, se a função é constante ou balanceada.

Solução Clássica

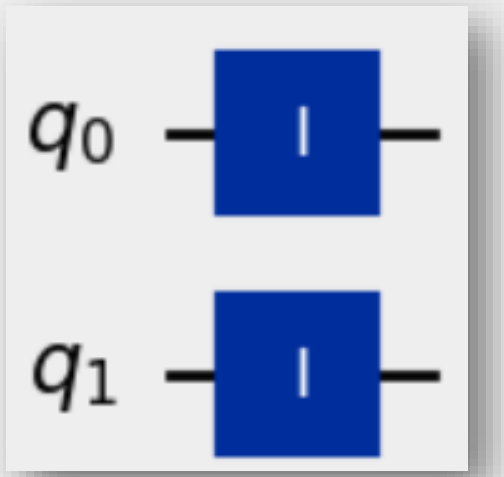
- Realizar metade das medições mais uma, ou seja
- Medir $2^{n-1} + 1$
- Quanto maior o v i se tornando o problema

Possibilidades

f_0	f_1	f_2	f_3
$0 \to 0$	$0 \to 0$	$0 \to 1$	$0 \to 1$
$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 1$

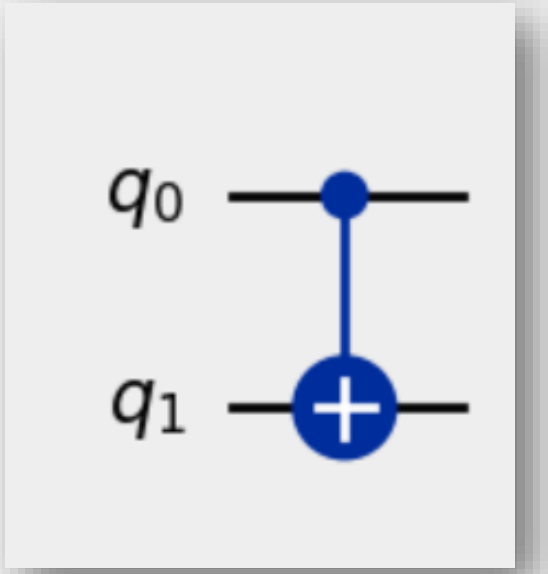
Possibilidades

f_0	f_1	f_2	f_3
$0 \to 0$	$0 \to 0$	$0 \to 1$	$0 \to 1$
$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 1$



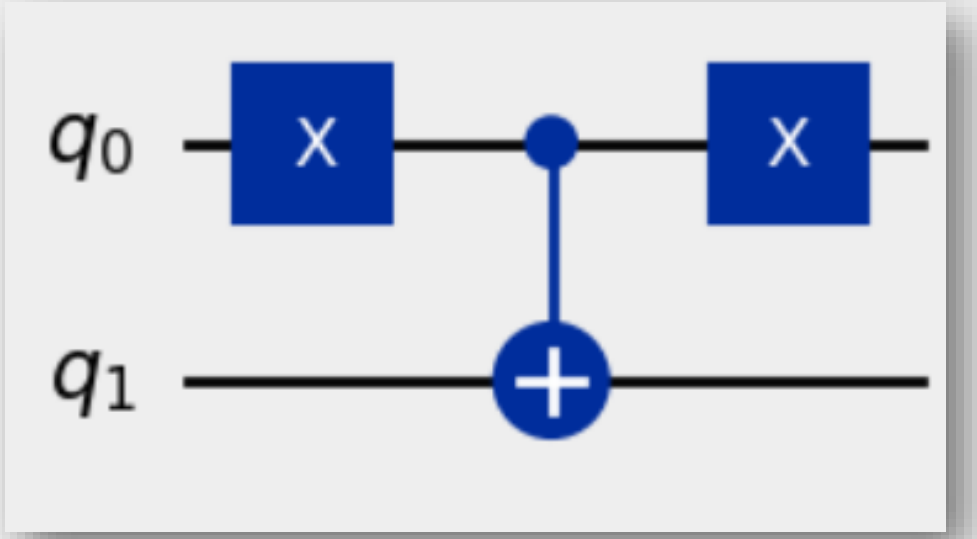
Possibilidades

f_0	f_1	f_2	f_3
$0 \to 0$	$0 \to 0$	$0 \to 1$	$0 \to 1$
$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 1$



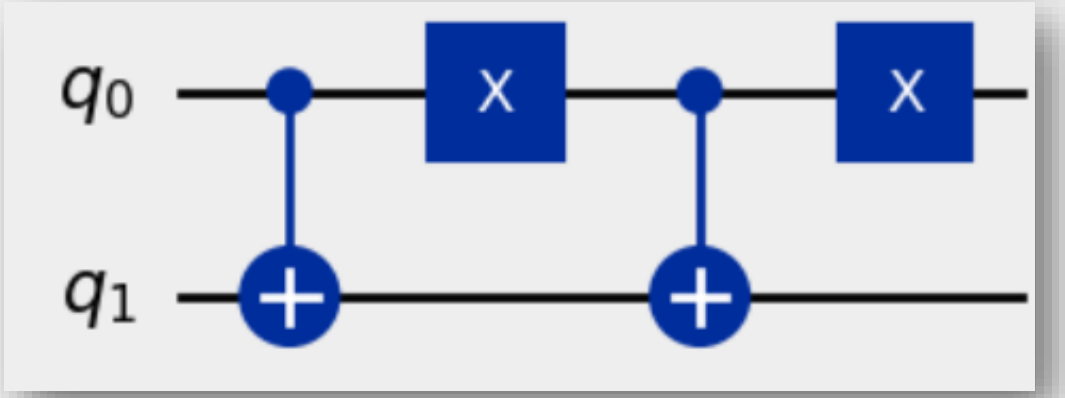
Possibilidades

f_0	f_1	f_2	f_3
$0 \to 0$	$0 \to 0$	$0 \to 1$	$0 \to 1$
$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 1$



Possibilidades

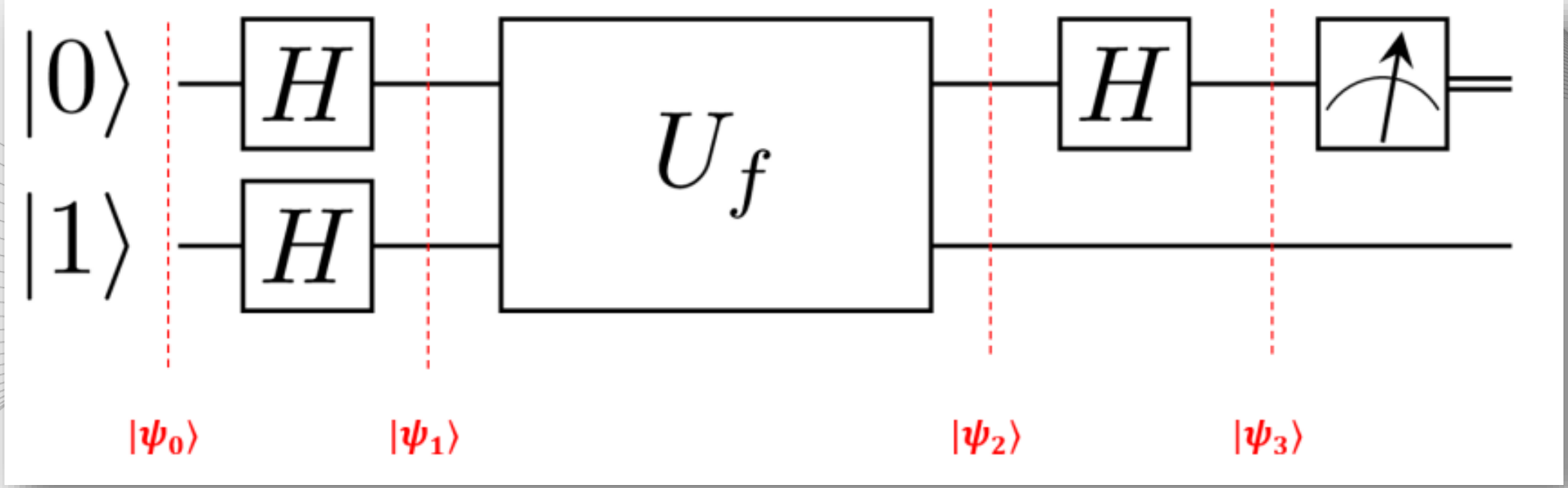
f_0	f_1	f_2	f_3
$0 \rightarrow 0$	$0 \rightarrow 0$	$0 \rightarrow 1$	$0 \rightarrow 1$
$1 \rightarrow 0$	$1 \rightarrow 0$	$1 \rightarrow 0$	$1 \rightarrow 1$



Solução Quântica

Algorimo de Deustch	
Entrada	$\|0\rangle \otimes \|1\rangle$
Passo 1	Preparar os estados $\|0\rangle$ e $\|1\rangle$
Passo2	Gerar superposição aplicando H nos qubits
Passo 3	Aplicar U_f
Passo 4	Aplicar H em todos qubits
Saída	$\begin{cases} 0 \ se \ f \ \text{é constante} \\ 1 \ se \ f \ \text{é balanceada} \end{cases}$

Solução Quântica



Solução Quântica

$f(0) \oplus f(1)$	**Saída**	**Tipo da função**
$0 \oplus 0$	0	Constante
$0 \oplus 1$	1	Balanceada
$1 \oplus 0$	1	Balanceada
$1 \oplus 1$	0	Constante

Implementando o Circuito de Deustch

```
from qiskit import QuantumCircuit
from qiskit.providers.basic_provider import BasicProvider
from qiskit.visualization import plot_histogram
from qiskit.visualization import circuit_drawer
import pylatexenc
```

Implementando o Circuito de Deustch

```python
def deutsch_function(case: int):
    # Essa função gera um circuito quântico para um dos quatro casos possíveis.
    if case not in [1, 2, 3, 4]:
        raise ValueError("`case` must be 1, 2, 3, or 4.")

    f = QuantumCircuit(2)
    if case in [2, 3]:
        f.cx(0, 1)
    if case in [3, 4]:
        f.x(1)
    return f
```

Implementando o Circuito de Deustch

```
for i in range(1, 5):
    qc = deutsch_function(i)
    print(f"Circuito para o caso {i}:")
    display(qc.draw(output="mpl", style={"backgroundcolor": "#EEEEEE"}))
```

Implementando o Circuito de Deustch

```python
def compile_circuit(function: QuantumCircuit):
    # Compilando o circuito do algoritmo de Deustch.

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

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

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

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

    return qc
```

Implementando o Circuito de Deustch

```
display(compile_circuit(deutsch_function(3)).draw(output="mpl", style={"backgroundcolor": "#EEEEEE"}))
```

Implementando o Circuito de Deustch

```python
def deutsch_algorithm(function: QuantumCircuit):
    # Determina se a função é constante ou balanceada.

    qc = compile_circuit(function)

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

Implementando o Circuito de Deustch

```
for i in range(1,5):
    print(f"Caso {i}:")
    f = deutsch_function(i)
    display(deutsch_algorithm(f))
```

QuIIN
Quantum Industrial Innovation

Obrigado !

Colocar um QR code aqui com as oportunidades do Quiin

SENAI CIMATEC

EMBRAPII
Empresa Brasileira de Pesquisa e Inovação Industrial

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO