Introdução à Programação de Computadores Quânticos

Profs. Renato Portugal e Franklin Marquezino

38° JAI/CSBC – 2019

Estrutura do curso

- **PARTE DA MANHÃ**
  - Qubit, portas lógicas e circuitos quânticos
  - Porta lógicas quânticas de 1 qubit
  - *Composer* da IBM
  - Portas lógicas quânticas de 2 qubits
  - Portas lógicas quânticas de 3 ou mais qubits
  - Paralelismo quântico
  - Modelo padrão da computação quântica
  - Algoritmo de Grover no *composer*

- PARTE DA TARDE
  - Programando os computadores quânticos da IBM
  - Qasm
  - Qiskit
  - Escrevendo um programa quântico básico
  - Executando o programa quântico
  - Implementação do algoritmo de Grover

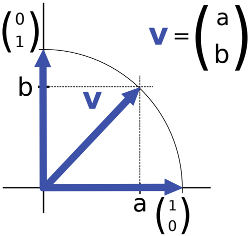
Introdução

Para aprender computação quântica (CQ):

- Boa notícia: não precisa fazer curso de mecânica quântica
- Má notícia: precisa fazer curso de álgebra linear
- Mecânica quântica para CQ = 4 regras de um jogo (por exemplo xadrez)

Breve revisão de álgebra linear

Espaço vetorial de 2 dimensões:



$$\mathbf{v} = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Norma de **v**:

$$\|\mathbf{v}\| = \sqrt{a^2 + b^2}$$

Qubit

Bit quântico (qubit)

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

Qubit genérico

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$$

com o vínculo

$$|a|^2 + |b|^2 = 1$$

Medição de um qubit

O estado (valor) do qubit antes da medida é

$$|\psi\rangle = a|0\rangle + b|1\rangle$$

com o vínculo

$$|a|^2 + |b|^2 = 1.$$

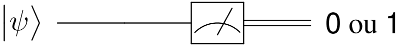
O resultado da medida é um bit clássico:

Bit 0 com probabilidade $|a|^2$

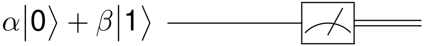
Bit 1 com probabilidade $|b|^2$

Circuito lógico

Representação na forma de circuito lógico:



Representação mais detalhada quando sabemos o estado do qubit:



$$\begin{cases} 0, & \text{com probabilidade } |\alpha|^2 \\ 1, & \text{com probabilidade } |\beta|^2 \end{cases}$$

Saída como histograma

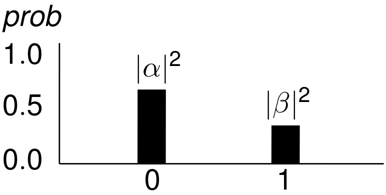


Figura: Histograma da distribuição de probabilidades da saída quando o estado do qubit $|\psi\rangle$ antes da medida é $\alpha|0\rangle + \beta|1\rangle$

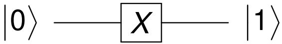
Exemplo de uma porta lógica

Uma porta lógica de 1 qubit é uma matriz 2×2 cuja vetor na saída tem a mesma norma do vetor de entrada.

Exemplo 1: porta X (NOT quântico)

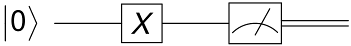
$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Representação em um circuito lógico:



Note que $|1\rangle = X|0\rangle$

Circuito completo (com medição)



$$\begin{cases} 0, \text{com prob } 0 \\ 1, \text{com prob } 1 \end{cases}$$

Exemplo de uma porta lógica

Exemplo 2: porta Hadamard H

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

Representação em um circuito lógico:

$$|0\rangle \;-\!\!-\!\!-\boxed{H}-\!\!-\!\!-\; |+\rangle$$

Note que $|+\rangle = H|0\rangle$ onde

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

Circuito completo (com medição)

$$|0\rangle \;-\!\!-\!\!-\boxed{H}-\!\!-\!\!-\boxed{\text{medição}}=\!=\!= \begin{cases} 0, \text{com prob } 0.5 \\ 1, \text{com prob } 0.5 \end{cases}$$

Exemplo prático

Usar o computador quântico da IBM

`https://quantum-computing.ibm.com/`

Estado de 2 qubits

Espaço vetorial de 4 dimensões

Base é formada por:

$$|0\rangle = |00\rangle = |0\rangle|0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad |1\rangle = |01\rangle = |0\rangle|1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

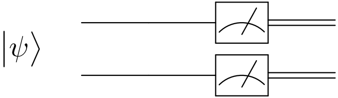
$$|2\rangle = |10\rangle = |1\rangle|0\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad |3\rangle = |11\rangle = |1\rangle|1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Estado de 2 qubits

Se

$$|\psi\rangle = c_0|00\rangle + c_1|01\rangle + c_2|10\rangle + c_3|11\rangle$$

então



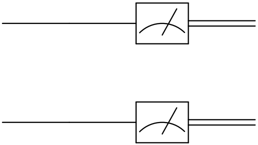
$$\begin{cases} 00, \text{ com probabilidade } |c_0|^2 \\ 01, \text{ com probabilidade } |c_1|^2 \\ 10, \text{ com probabilidade } |c_2|^2 \\ 11, \text{ com probabilidade } |c_3|^2 \end{cases}$$

O histograma tem 4 barras.

Estado de 2 qubits – Caso particular

Se qubit 1: $|\psi\rangle_1 = a|0\rangle + b|1\rangle$ e qubit 2: $|\psi\rangle_2 = c|0\rangle + d|1\rangle$

então

$|\psi\rangle_1$  $\begin{cases} 0, \text{com prob } |a|^2 \\ 1, \text{com prob } |b|^2 \end{cases}$

$|\psi\rangle_2$ $\begin{cases} 0, \text{com prob } |c|^2 \\ 1, \text{com prob } |d|^2 \end{cases}$

Estado de 2 qubits – Caso particular

Produto de Kronecker de vetores da base

$$|1\rangle \otimes |0\rangle = |10\rangle = |2\rangle$$

qubit 1: $|\psi\rangle_1 = a\,|0\rangle + b\,|1\rangle$ qubit 2: $|\psi\rangle_2 = c\,|0\rangle + d\,|1\rangle$

qubit 1 + qubit 2:

$$\begin{aligned} |\psi\rangle_1 \otimes |\psi\rangle_2 &= (a\,|0\rangle + b\,|1\rangle)(c\,|0\rangle + d\,|1\rangle) \\ &= a\,c\;|0\rangle + a\,d\;|1\rangle + b\,c\;|2\rangle + b\,d\;|3\rangle \end{aligned}$$

Produto de Kronecker

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a\,c \\ a\,d \\ b\,c \\ b\,d \end{bmatrix}$$

Compare com

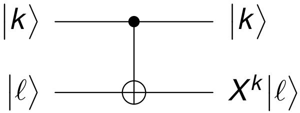
$$|\psi\rangle_1 \otimes |\psi\rangle_2 \quad = \quad a\,c\,|0\rangle + a\,d\,|1\rangle + b\,c\,|2\rangle + b\,d\,|3\rangle$$

2 qubits – Porta CNOT

Porta NOT controlada (CNOT or $C(X)$) é definida como

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

e representada por

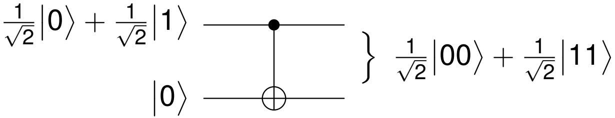


$\bullet$ = qubit de controle, $\oplus$ = qubit alvo, onde

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

Execício: Ache a matriz que representa CNOT.

Exemplo de CNOT



- O resultado é probabilístico:

  $|00\rangle$ com probabilidade $\frac{1}{2}$
  $|11\rangle$ com probabilidade $\frac{1}{2}$

Exemplo prático

Usar o computador quântico da IBM

`https://quantum-computing.ibm.com/`

Generalização para n qubits

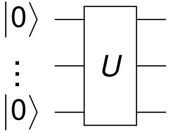
Base computacional:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots \quad |N-1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$N = 2^n$, n é o número de qubits.

Modelo padrão da computação quântica

- Circuito genérico



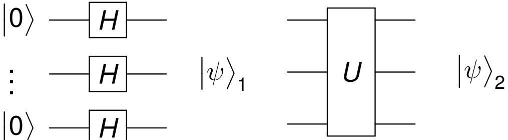
$$\Big\} \; c_0|0\rangle + \cdots + c_{2^n-1}|2^n-1\rangle$$

- A porta U deve satisfazer $UU^\dagger = I$
- Faça uma medição na base computacional
- O resultado é probabilístico:

$$\begin{array}{cc} 0\cdots 0 & \text{com probabilidade} \;\; |c_0|^2 \\ \vdots & \\ 1\cdots 1 & \text{com probabilidade} \;\; |c_{2^n-1}|^2 \end{array}$$

Paralelismo Quântico

Entrada: $|0\rangle \otimes \cdots \otimes |0\rangle$ com n termos



Saída

int.: $\quad |\psi_1\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes n} = \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle + |2\rangle + \cdots + |2^n - 1\rangle)$

Saída final: $\quad |\psi_2\rangle = \frac{1}{\sqrt{2^n}}\left(U \cdot |0\rangle + U \cdot |1\rangle + \cdots + U \cdot |2^n - 1\rangle\right)$

Amplificação de Amplitude

Queremos achar $|x\rangle$.

A saída é: $|\psi_2\rangle = \frac{1}{\sqrt{2^n}}\left(U \cdot |0\rangle + U \cdot |1\rangle + \cdots + U \cdot |2^n - 1\rangle\right)$

ou: $|\psi_2\rangle = c_0|0\rangle + \cdots + c_x|x\rangle + \cdots + c_{2^n-1}|2^n - 1\rangle$

Escolhemos U tal que $|c_x| \approx 1$

Consequentemente $|c_0| \approx |c_1| \approx \cdots \approx |c_{2^n-1}| \approx 0$

O resultado é $|x\rangle$ com probabilidade ≈ 1

Conjunto de portas lógicas quânticas universais

$$\{\mathrm{CNOT}, H, T\}$$

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

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

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\frac{\mathrm{i}\pi}{4}} \end{bmatrix}$$

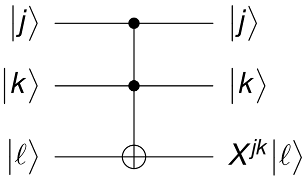
Dado erro $\epsilon > 0$, qualquer matriz unitária U pode ser escrita como produto matricial e tensorial de matrizes CNOT, H e T dentro do erro ϵ.

3 qubits – Porta Toffoli

A porta Toffoli $C^2(X)$ é definida como

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

representada por



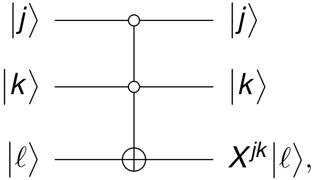
A porta Toffoli tem 2 controles e um alvo.

Porta Toffoli controlada por zeros

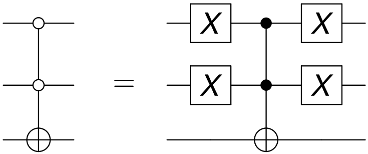
Definida como

$$|j_1\rangle|j_2\rangle|j_3\rangle \longmapsto |j_1\rangle|j_2\rangle X^{(1-j_1)(1-j_2)}|j_3\rangle.$$

Representada por



Equivalência com a porta Toffoli



Algoritmo de Grover

O Algoritmo de Grover

- Banco de dados com N elementos não ordendos
- Queremos saber se um elemento x_0 pertence ou não
- Complexidade do algoritmo clássico: $O(N)$
- Complexidade do algoritmo quântico: $O(\sqrt{N})$ (Grover 96)
- Algoritmo se passa em um sub-espaço real do espaço de Hilbert

O Algoritmo de Grover - Passo 1

- Vamos supor que $N = 2^n$ e os elementos são números menores que N, podendo ter repetições.
- O número procurado x_0 pode pertencer ou não.
- Passo 1: Colocar o computador quântico no estado

$$|\mathrm{d}\rangle = H^{\otimes n}|0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$$

Exemplo: $x_0 = 110 = 6$. $N = 8$.

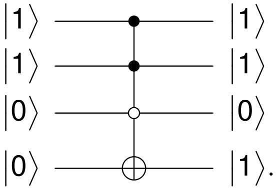
$$|\mathrm{d}\rangle = \frac{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |\,\mathbf{110}\rangle + |111\rangle}{\sqrt{8}}$$

Passo 2

Passo 2: Definir

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

Exemplo: circuito de F_{x_0} no caso $N = 8$ e $x_0 = 6 = 110$



Algoritmo de Grover

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

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

Passo 1: Prepare o estado inicial $|\mathrm{d}\rangle| - \rangle$

Passo 2: Aplique $(G\, F_{x_0})^t$, onde $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$

Passo 3: Faça uma medição do primeiro registrador

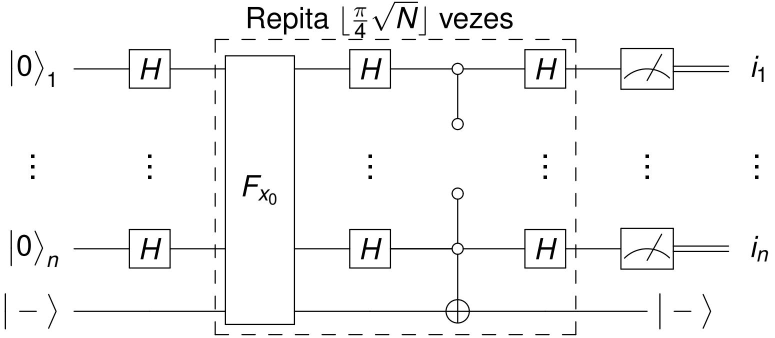
Onde

$$G = \left(2\,|\mathrm{d}\rangle\langle \mathrm{d}| - I_N\right) \otimes I_2,$$

e

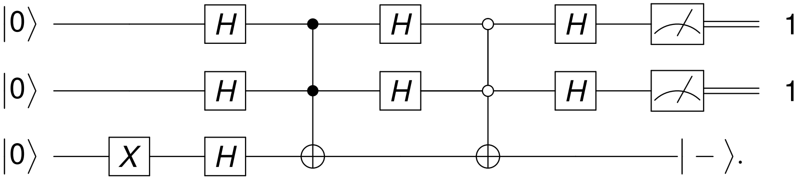
$$| - \rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Circuito do Algoritmo de Grover



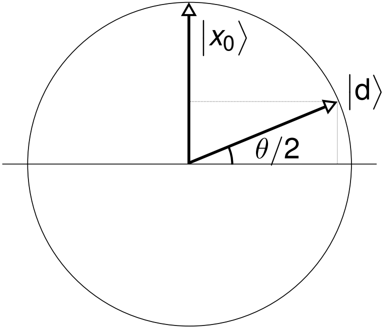
Circuito para $N = 4$

$N = 4$ e $x_0 = 11$



Análise do algoritmo

Início:



Temos que

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

Portanto,

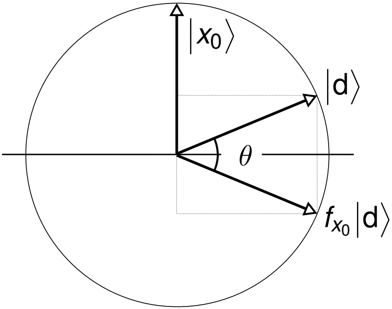
$$\theta \approx \frac{2}{\sqrt{N}}.$$

Análise do algoritmo

Note que (f_{x_0} versão reduzida de F_{x_0})

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

Portanto

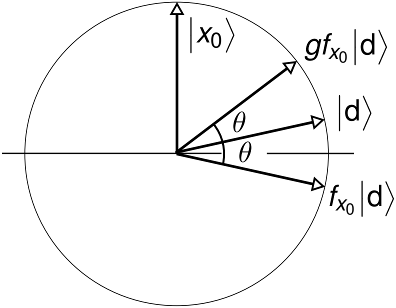


Análise do algoritmo

O próximo passo é aplicar

$$g = 2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N$$

cujo efeito é



pois

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

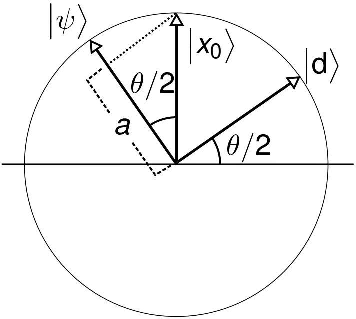
$$g|\mathrm{d}^\perp\rangle = (2\,|\mathrm{d}\rangle\langle\mathrm{d}| - I_N)\,|\mathrm{d}^\perp\rangle = 2\,|\mathrm{d}\rangle\langle\mathrm{d}|\mathrm{d}^\perp\rangle - |\mathrm{d}^\perp\rangle = -|\mathrm{d}^\perp\rangle$$

Análise do algoritmo

Queremos saber quantas iterações r tal que $r\theta = \pi/2$:

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

O vetor $|\psi\rangle$ é o estado final.



Cálculo da probabilidade de sucesso

O estado do computador quântico é

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

A probabilidade de sucesso é maior ou igual do que o módulo ao quadrado da amplitude de $\left|x_0\right\rangle$ na decomposição de $\left|\psi\right\rangle$ na base computacional (veja a na fig. anterior). A projeção de $\left|x_0\right\rangle$ no estado final é no máximo $\cos(\theta/2)$. Assim

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

Referência

Curso *programaquantica* no GitHub:

`https://github.com/programaquantica/`

Estrutura do curso

- PARTE DA MANHÃ
  - Qubit, portas lógicas e circuitos quânticos
  - Porta lógicas quânticas de 1 qubit
  - *Composer* da IBM
  - Portas lógicas quânticas de 2 qubits
  - Portas lógicas quânticas de 3 ou mais qubits
  - Paralelismo quântico
  - Modelo padrão da computação quântica
  - Algoritmo de Grover no *composer*

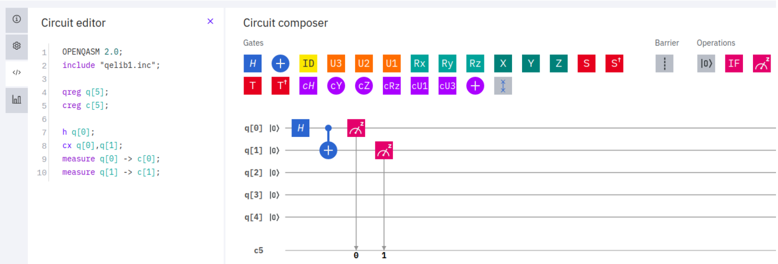
- **PARTE DA TARDE**
  - Programando os computadores quânticos da IBM
  - Qasm
  - Qiskit
  - Escrevendo um programa quântico básico
  - Executando o programa quântico
  - Implementação do algoritmo de Grover

Programando os computadores quânticos da IBM

- Já demonstramos uso dos computadores de 5 qubits usando *composer*
- Muito simples, pois podemos pegar portas lógicas e arrastar para circuito
- Porém não útil para programas grandes; neste caso, melhor usar **Qasm** (*Quantum Assembly Language*)

Quantum assembly

- Qasm: linguagem de baixo nível para circuitos do IBM Q Experience
- Podem ser escritos de diversas formas: sempre que circuito é criado no *composer*, código Qasm é gerado
- Código gerado pode ser visualizado e editado no browser



(Reprint Courtesy of IBM Corporation ©)

Comandos básicos

- Comentários: iniciar linha com `//`
- Primeira linha (exceto comentário) deve ser comando `OPENQASM`, seguido da versão do Qasm
- Comandos devem encerrar com ponto-e-vírgula
- Comando `include`: permite incluir código de arquivo externo
- Normalmente incluímos pelos menos `qelib1.inc`
- Praticamente obrigatórios: comandos `qreg` `creg`
  Exemplo:
  - `qreg q[5]`: registrador “q”, com 5 qubits
  - `creg c[5]`: registrador “c”, com 5 bits

Portas lógicas

- Para incluir porta lógica com Qasm: nome da porta seguido de qubits sobre os quais deve atuar
- Se `qelib1.inc` foi incluído, todas portas que vimos no *composer* estão disponíveis
- Exemplo (Hadamard): `h q[0]`
- Exemplo (CNOT): `cx q[0], q[1]`
- Qubits são numerados a partir de zero

Medição

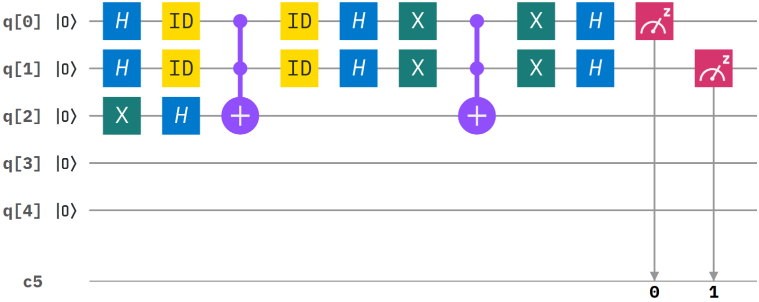
- Comando `measure` seguido de
  - registrador quântico para medir
  - registrador clássico para resultado
- Por exemplo: `measure q[0] -> c[0]`

Como editar

- Pode-se editar programa Qasm em qualquer editor de textos e depois colar código-fonte no IBM Q Experience
- Circuito correspondente é mostrado, pode-se passar a editá-lo do modo visual
- Bastante conveniente ao escrever circuitos grandes.
- Código Qasm pode ser gerado implicitamente usando linguagens de alto nível
- A seguir, veremos o Qiskit para escrever programas quânticos usando Python.

Exercício

Escreva o código Qasm para o circuito abaixo:



(Reprint Courtesy of IBM Corporation ©)

Exercício

Desenhe o circuito para o código abaixo:

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];
creg c[5];

id q[0];
id q[1];
id q[2];
h q[2];
cx q[1],q[2];
tdg q[2];
cx q[0],q[2];
t q[2];
cx q[1],q[2];
t q[1];
tdg q[2];
cx q[0],q[2];
cx q[0],q[1];
t q[2];
t q[0];
tdg q[1];
h q[2];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

Requisitos

- **Qiskit:** kit de desenvolvimento de software da IBM para computação quântica
- Código aberto e gratuito
- Disponível em `https://qiskit.org`
- Compatível com Linux, MacOS, Windows
- Precisa de Python 3.5 ou mais recente

O novo site do IBM Q já tem um ambiente todo configurado, mas vamos aprender a instalar localmente

Instalação

- Instalar Python, caso não tenha
  - `www.python.org`, ou
  - `www.anaconda.com/distribution` (preferível)

Instalação

- Instalar Python, caso não tenha
  - `www.python.org`, ou
  - `www.anaconda.com/distribution` (preferível)
- Recomendável instalar pip, caso não tenha
  - `https://pip.pypa.io`

Instalação

- Instalar Python, caso não tenha
  - `www.python.org`, ou
  - `www.anaconda.com/distribution` (preferível)
- Recomendável instalar pip, caso não tenha
  - `https://pip.pypa.io`
- Instalar qiskit
  - Pode baixar de `https://qiskit.org`, mas com pip é mais fácil:
    `pip install qiskit`
    ou, se tiver planos de continuar além desse curso:
    `pip install qiskit[visualization] qiskit-aqua`
  - Atenção: quando há duas versões do Python instaladas, pode ser necessário usar pip3 em vez de pip

Instalação

- Instalar Python, caso não tenha
  - `www.python.org`, ou
  - `www.anaconda.com/distribution` (preferível)
- Recomendável instalar pip, caso não tenha
  - `https://pip.pypa.io`
- Instalar qiskit
  - Pode baixar de `https://qiskit.org`, mas com pip é mais fácil:
    `pip install qiskit`
    ou, se tiver planos de continuar além desse curso:
    `pip install qiskit[visualization] qiskit-aqua`
  - Atenção: quando há duas versões do Python instaladas, pode ser necessário usar pip3 em vez de pip
- Recomendável instalar Jupyter
  - Pode baixar de `https://jupyter.org`, mas como pip é mais fácil:
    `pip install jupyter`

Apresentando o Jupyter

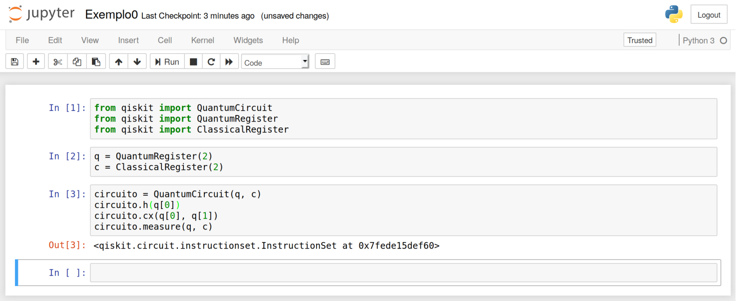
- Recomendamos usar Jupyter: vá ao terminal de comando e digite `jupyter notebook`
- Jupyter é aberto no browser padrão
- Navegue pelas pastas e abra arquivos `ipynb` (*notebooks*)
- Crie novos notebooks clicando em *New*, escolha Python 3

Apresentando o Qiskit

- Qiskit é composto por Terra, Aer, Ignis e Aqua; vamos usar os dois primeiros
- Qualquer um com conhecimentos básicos de computação quântica e Python possa programar os computadores quânticos da IBM
- Para executar programas diretamente nos computadores da IBM precisa da *API Token*, obtida no site do IBM Q Experience
- Qiskit facilita também a execução de experimentos em simuladores clássicos de alto desempenho

Jupyter Notebook

- Abra a interface do Jupyter Notebook



Módulos básicos

- Inclua as seguintes linhas no início do código:

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

Definindo o circuito

- Digamos que nosso circuito tem 2 qubits, e o resultado ocupa 2 bits. Nesse caso:

```
q = QuantumRegister(2)
c = ClassicalRegister(2)
```

- Os nomes `q` e `c` são apenas variáveis
- Opcionalmente, é possível definir apelido para os registradores, para as visualizações:

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

- Já podemos definir nosso circuito quântico:

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

Incluindo as portas

- Circuito ainda está vazio, precisamos incluir portas, medições etc.
- Para incluir porta de Hadamard no primeiro qubit, por exemplo:

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

Outras portas de 1 qubit

- Outras portas de 1 qubit também estão disponíveis por meio de comandos semelhantes
- Todas as portas elementares do *composer* estão também no Qiskit:

```
circuito.id(q[0])
circuito.x(q[0])
circuito.y(q[0])
circuito.z(q[0])
circuito.s(q[0])
circuito.sdg(q[0])
circuito.t(q[0])
circuito.tdg(q[0])
```

Portas de rotação

- Portas de rotação $R_X(\theta)$, $R_Y(\theta)$ e $R_Z(\theta)$ podem ser incluídas:

```
theta = 1.5
circuito.rx(theta, q[0])
circuito.ry(theta, q[0])
circuito.rz(theta, q[0])
```

Portas com mais qubits

- Não dá para escrever algoritmos muito interessantes só com portas de 1 qubit
- Precisamos pelo menos de uma porta CNOT:

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

Medições

- Precisamos ainda efetuar uma medição
- Por isso que ao definirmos o circuito criamos um registrador clássico

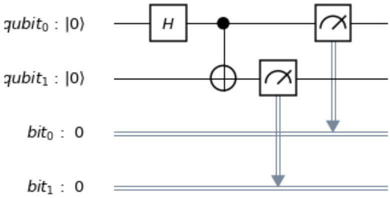
```
circuito.measure(q, c)   # mede tudo
circuito.measure(q[0], c[0]) # só o primeiro
```

Visualização

- Para termos certeza de que fizemos tudo corretamente, é interessante visualizar o circuito
- Podemos facilmente obter essa visualização no Qiskit:

```
%matplotlib inline  # recomendável no Jupyter
circuito.draw(output='mpl')
```

- Também poderia ser `output='latex'`



Geração de código Qasm

- Qiskit permite obter o código-fonte Qasm para os circuitos gerados:

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

- No exemplo anterior, teríamos:

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

Executando circuito

- Agora já podemos executr o circuito em um simulador ou em um computador quântico real
- Como? Forma mais imediata seria gerar o código Qasm no Qiskit e em seguida copiá-lo no *composer*
- No entanto, podemos fazer direto pelo Qiskit, sem o *composer*
- Para simular, precisamos do seguinte:

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

Backends

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

Qasm simulator

- Para executar no `qasm_simulator` circuito precisa ter pelo menos uma medição
- Temos que dizer quantas repetições vamos simular:

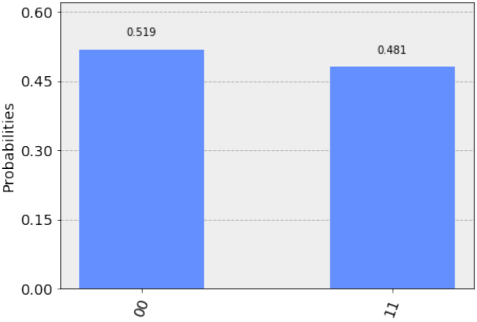
```
backend = BasicAer.get_backend('qasm_simulator')
job = execute(circuito, backend, shots=1024)
```

Visualizando resultado

- Resultado armazenadas no objeto `job`, difícil de ler
- Podemos extrair o resultado e realizar uma contagem:

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

- Objeto `contagem` é um dicionário do Python
- Para visualizar como histograma, comando `plot_histogram(contagem)`



Statevector simulator

- O backend `statevector_simulator` é utilizado de forma semelhante:

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

- Agora queremos um vetor de estado, não em uma contagem de resultados de medições! O comando é o seguinte:

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

- Retorna um *array* de números complexos
- Visualizações úteis, como por exemplo o comando `plot_state_city(estado)`

Unitary simulator

- Para simular no `unitary_simulator`, não pode ter medições! (Afinal, medição não é operação unitária)
- Os comandos são os seguintes:

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

- Resultado é matriz unitária, pode ser extraída com:

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

Outros backends

- Os backends que vimos até agora são executados localmente
- Qiskit também tem backends que direcionam jobs para IBM:
  - para computadores quânticos reais
  - para simulação HPC
- Tem que acessar conta no IBM Q Experience e copiar API Token:

```
from qiskit import IBMQ
IBMQ.save_account('Colar_Token_Aqui')
```

Usando a conta

- Para acessar conta IBM Q a partir do Qiskit, usar comando `IBMQ.load_accounts()`
- Para obter lista completa de backends, usae comando `IBMQ.backends()`
- Atualmente há quatro:
  - `ibmqx4`, para executar no IBM Q 5 Tenerife (5 qubits)
  - `ibmqx2`, para executar no IBM Q 5 Yorktown (5 qubits)
  - `ibmq_16_melbourne`, para executar no IBM Q 14 Melbourne (14 qubits)
  - `ibmq_qasm_simulator`, para simular remotamente em HPC (até 32 qubits)

Exemplo de execução

- Para executar no IBM Q 5 Yorktown com 1024 repetições:

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

- Job entra na fila para ser executado no computador quântico
- Espera pode demorar
- Para saber posição do job na fila em tempo real:

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

Dica

Antes mesmo de executar, você pode verificar o status do computador

Exemplo:

```
ibmqx4 = IBMQ.get_backend('ibmqx4')
ibmqx4.status()
```

Visualizar resultado

- Quando job termina, podemos extrair e visualizar o resultado:

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

Implementando Grover

- Já estudamos algoritmo de Grover na parte da manhã
- Já pronto no Qiskit Aqua, mas aqui vamos fazer implementação mais econômica
- Vamos implementar versão melhorada do algoritmo de Grover para lista com $N = 4$ elementos

Definindo os registradores

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

Preparando a superposição

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

- Está funcionando? Basta simular até esse ponto:

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

Construindo oráculo

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

- Vejam texto do tutorial para detalhes sobre oráculo
- Podem simular com statevector sempre que quiserem tirar dúvidas
- Podem visualizar circuito (nesse caso, comando `circuito.barrier()` pode ser útil)
- Para executar no IBM Q 5 Tenerife, é recomendável inverter posição dos qubits

Dica

Para verificar o mapa de conectividades dos qubits:

Exemplo:

```
ibmqx4 = IBMQ.get_backend('ibmqx4')
plot_gate_map(ibmqx4)
```

Construindo operador de Grover

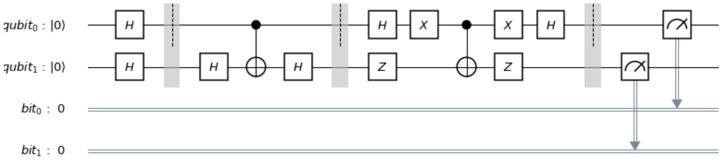
```
circuito.h(q[0])
circuito.z(q[1])

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

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

Finalizando

- Agora só falta medir!

  `circuito.measure(q, c)`

- Para visualizar circuito, usamos os comandos que já aprendemos:



Referência

Curso *programaquantica* no GitHub:

`https://github.com/programaquantica/`

Para aprender mais

Aos alunos de graduação motivados a fazer pesquisa, fica o convite:

- LNCC (Petrópolis/RJ): Mestrado e Doutorado em Modelagem Computacional
- UFRJ (Rio de Janeiro/RJ): Mestrado e Doutorado em Engenharia de Sistemas e Computação