Francisco Rouxinol
Instituto de Física Gleb Watghin
Rua Sergio Buarque de Holanda, 777
13083-859, Campinas, SP
e-mail: rouxinol@unicamp.br
tel: +55(19)3521-5462

Projeto Principal: “Algoritmo quântico de busca de Grover e de fatoração de Shor: implementação da simulação em computadores clássicos e quânticos”
Grande Área de conhecimento: Ciências Exatas e da Terra
Área de Conhecimento: Física
Subárea de Conhecimento: Física da Matéria Condensada
Aluno(a): Daniel Benvenutti
Orientador: Francisco Rouxinol
Instituição: Instituto de Física Gleb Wataghin, UNICAMP
Palavras chaves: Eletrodinâmica Quântica, Computação Quântica e Informação
Referente: Primeiro Relatório

Resumo

Resumo do projeto original: Nós apresentamos um projeto de pesquisa focado no desenvolvimento pelo aluno de um conjunto de ferramentas para simular algoritmos quânticos em computadores clássicos e quânticos. Serão tratados e discutidos os conceitos básicos de computação quântica e mecânica quântica, como também a implementação de portas quânticas e o algoritmo de busca de Grover e de fatoração de Shor. Com a implementação deste projeto é esperado que importantes conceitos de quântica, como medida, funções de onda de muitos corpos, e momento angular, sejam aprofundados como também o estudo de importantes tópicos avançados em física e engenharia.

1 Introdução

O *objetivo do projeto* era levar o estudante a pensar profundamente em como um computador baseado em lógica quântica funciona. Utilizando um conjunto de ferramentes computacionais o estudante deveria desenvolver em um computador clássico um conjunto de portas lógicas quânticas e simular diversos algorítimos quânticos (i.e. Grover[1], Shor[2], etc) e testar em computadores quânticos disponíveis *online* estes sistemas. *Em longo prazo*, este projeto está preparando o aluno para entender de forma mais clara importantes tópicos modernos de computação quântica, física, e programação em sistemas quânticos - preparando o caminho para o desenvolvimento de projetos mais avançados neste fascinante campo de pesquisa.

Um importante aspecto do projeto foi o desenvolvimento dos vetores de estado e portas lógicas utilizando as bibliotecas Scipy e Sympy existentes na linguagem Python [3], como na experiência de aplicar e desenvolver conceitos de física quântica numa das suas aplicações práticas, *a computação quântica*.

Durante os primeiros 3 bimestres, trabalhamos na preparação das diversas portas quânticas necessárias para execução dos algorítimos estudados neste projeto. Para escrever os programas utilizamos ambiente de desenvolvimento interativo Jupyter [4] com a linguagem de programação Python [3]. Foram desenvolvidos os algorítimos para a *inicialização do sistema*, a *operação de portas quânticas*, e a *medição dos estado*.

Nos 3 bimestres finais, trabalhamos na implementação da execução dos algorítimos e na aprendizagem da utilização dos computadores quânticos da IBM [5]. Terminamos o trabalho comparando os resultados da simulação dos algorítimos quânticos rodados em computadores clássico com os resultados das simulações dos algorítimos em computadores quânticos da IBM.

2 Atividades Principais

De forma resumida, durante o projeto,trabalhamos nas quatro etapas do cronograma. Inicialmente desenvolvemos os registradores de N-qubits. Cada registrador guardava a informação do estado composto por 3 sistemas de dois níveis (qubits), descrito por um único estado quântico conjunto $|\psi\rangle$. Utilizando estes vetores, preparamos algorítimos para preparar o sistema de interesse no estado de desejado.

Na segunda etapa, desenvolvemos funções que simulam uma porta quântica (análogo a uma porta lógica), necessárias para operar os qubits. Utilizando os elementos dos vetores de estado e a portas lógicas, a probabilidade de medir um determinado valor pode ser determinado, simulando-se uma medida no sistema quântico.

Com estas funções e algorítimos programados, foram preparados os algorítimos de Groover de busca e Shor de fatoração de números inteiros. Ambos funcionaram como esperado. O de Groover amplificando a probabilidade do estado buscado até ele se destacar entre os outros. E o de Shor obtendo como saída o período de uma função do algoritmo que é o passo do algoritmo para fatorar o número realizado no computador quântico.

Na última implementamos os algorítimos Groover de busca e Shor de fatoração de números inteiros no computador quântico da IBM e comparamos com os resultados obtidos em nosso programa.

Nas próximas seções descrevemos em mais detalhes as etapas desenvolvidas neste período. Também fornecemos o código utilizando nesta etapa na plataforma GitHub no endereço: `https://github.com/Danielgb23/ic_comp_quantica`

2.1 Registrador Quântico

A primeira parte do projeto e construir o simulador. Para simular o estado de N qubits precisamos de um vetor de 2^N elementos. Por exemplo, um vetor de 3 qubits com 8 elementos:

$$\psi = [0, 0, 0, 0, 0, 0, 0, 0]$$

Cada estado nesse vetor representa uma combinação das medidas possíveis dos qubits. O primeiro estado é $|000\rangle$ o segundo é $|001\rangle$, o terceiro é $|010\rangle$ e assim por diante até o oitavo, $|111\rangle$.

Em seguida precisamos poder medir esse estado quântico. Uma medida é representada por por uma escolha aleatória do computador levando como pesos probabilísticos os elementos do vetor de estado com maior módulo complexo ao quadrado.

$$[0, 1, 0, 0, 0, 0, 0, 0]$$

Por exemplo, o segundo elemento do vetor acima tem 100% de chance de ser medida.

$$[0.5, 0.5, 0.5, 0.5, 0, 0, 0, 0]$$

Já neste caso, temos $|0.5|^2 = 1/4$ de chance de ser medido no primeiro, segundo, terceiro ou quarto estados.

2.2 Portas quânticas

2.2.1 Porta de Hadamard

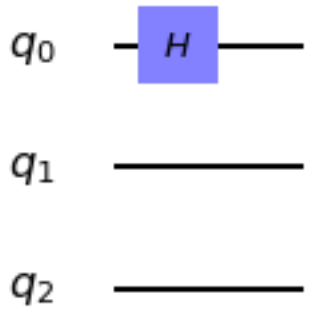

Figura 1: Porta de Hadamard

Uma das portas quânticas mais importantes é a *Porta de Hadamard*, H. Ela é extremamente interessante, pois coloca um qubit em uma superposição de dois estados. Por exemplo, o código:

```
qubit=[1,0] # estado inicial do qubit
qubit=hadamard(qubit, 1, 1) # hadamard no primeiro qubit
print(qubit)
```

retorna

$$[0.707106781186547, 0.707106781186547] = [1/\sqrt{2}, 1/\sqrt{2}] = |\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

2.2.2 Porta de Fase

Outra importante porta é a *Porta de Fase*, S. Ela muda a fase complexa do elemento do vetor que descreve o qubit.

$$S = \begin{vmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{vmatrix} \tag{1}$$

Então se usarmos uma dessas portas:

```
qubit=[0,1] # estado inicial do qubit
qubit=muda_fase(qubit, 1, math.pi/2, 1)
print(qubit)
medir_n(qubit, 100, 1)
```

obtemos:

$$|\psi\rangle = [0,\ 0 + 1.0i]$$

Que tem um número imaginário puro no segundo elemento do vetor, devido a mudança de fase de $\pi/2$.

2.2.3 Porta CNOT

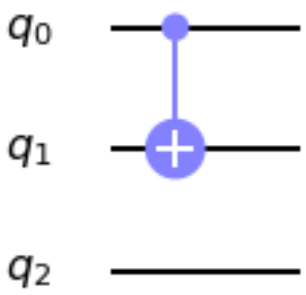

Figura 2: Porta CNOT

Portões CNOT são portas quânticas NOT (ou NÃO) controladas e funcionam da seguinte maneira: Há um qubit controlador e um qubit que sofre a operação NOT. Se e somente se o qubit controlador é $|1\rangle$ que o outro qubit é invertido de $|1\rangle$ para $|0\rangle$ ou de $|0\rangle$ para $|1\rangle$. Se o controlador for $|0\rangle$ o controlado mantém o seu estado. O qubit controlador não é alterado. Com essa porta podemos fazer o estado de emaranhamento quântico:

```
Psi=[0]*8 # estado inicial

seta_base(Psi,0)
#hadamard no segundo qubit para coloca-lo numa superposição
Psi=hadamard(Psi, 2, 3)

#CNOT para emaranhar os qubits
Psi=cnot3(Psi, 3)

print(Psi)
medir_n(Psi, 1000, 3)
```

$$|\psi\rangle = [0.71, 0, 0, 0.71, 0, 0, 0, 0]$$

Obteremos uma superposição entre os dois estados $|000\rangle$ e $|011\rangle$. Logo, há uma correlação na medida do segundo e terceiro qubit. Se um é medido zero (1) o outro também é, se um é medido 1 o outro também será.

2.3 Algoritmo de Groover

O algoritmo de Groover é um algoritmo de busca. Dado um vetor de possíveis respostas, procuramos neste vetor, um valor específico, que indica qual é a resposta correta. Com uma algoritmo normal de varredura em um computador clássico no pior caso temos que verificar todos os n elementos. Já no algoritmo de Groover precisamos de apenas $\sqrt{n}$ buscas.

Apesar disso a nossa lista precisa ser armazenada num oráculo quântico que é um operador que será utilizado nos nossos qubits. O que é o mais difícil.

O algoritmo de Groover funciona usando uma técnica chamada amplificação de amplitude. A fase do elemento do vetor que é a resposta procurado é invertida, e em seguida, com o operador de difusão de Groover é amplificada. Isso é

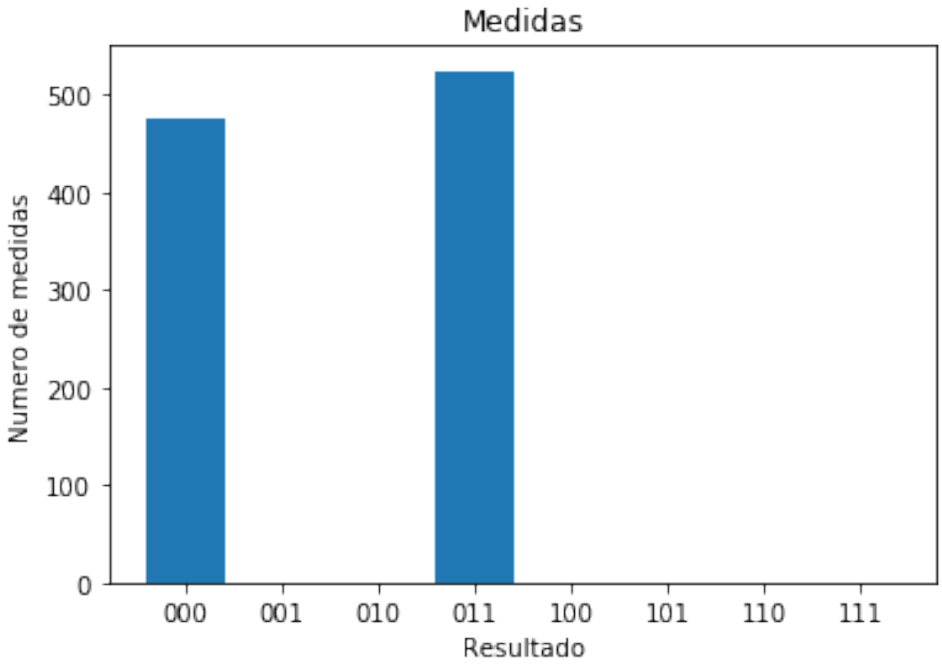

Figura 3: Histograma com valores obtidos para 1000 medidas de uma Porta CNOT

repetido um número ótimo de vezes, o que deixa a resposta correta com uma maior probabilidade de ser medida.

Aqui temos uma execução do código:

```
Psi=[1,0,0,0,0,0,0,0]

count=1
while count <= 3:    #passa o loop pelo vetor psi
    Psi=hadamard(Psi, count, 3) #hadamard em todos os qubits
    count = count + 1

resposta=6

#blocos do operador de difusao de groover
Psi=bloco_groover(Psi, 3, resposta)
Psi=bloco_groover(Psi, 3, resposta)

print(Psi)
medir_n(Psi, 1000, 3)
```

O vetor resposta encontrado é:

$$|\psi\rangle = [-0.08, -0.08, -0.08, -0.08, -0.08, -0.08, 0.97, -0.08]$$

Onde a amplitude do 7 elemento '110' (6 já que se começa do 0) é 0.97 enquanto a dos outros é de apenas -0.08. É importante lembrar que a probabilidade medida durante o experimento será ψ^2

Em seguida avaliamos o desempenho da simulação do algoritmo de Groover no meu computador. Abaixo os resultados:

Depois o algoritmo de Groover foi executado utilizando-se matrizes esparsas. Que são muito úteis nessas simulações, já que as matrizes dos operadores têm muitos zeros.

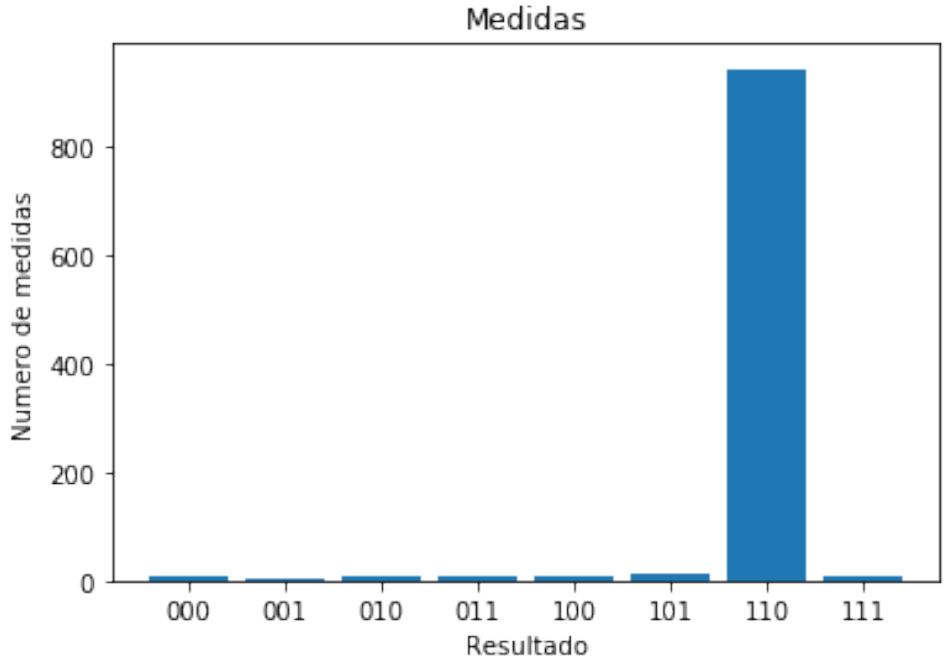

Figura 4: Histograma com valores obtidos para 1000 medidas para o algorítimo de Groover

Qubits	tempo
1	1.20s
2	1.15s
3	1.36s
4	3.51s
5	29.33s
6	353.69s

Tabela 1: Tempos de simulação para diferentes números de qubits

```
qubits=input()
resposta=1

Psi=[0]*2**qubits
Psi[0]=1

count=1
while count <= qubits:    #passa o loop pelo vetor psi
        Psi=sp_hadamard(Psi, count, qubits) #hadamard em todos os qubits
        count = count + 1

#blocos do operador de difusao de groover
count=1
   #passa o loop pelo vetor psi
while count <= round(math.pi/4*math.sqrt(2**qubits)):
        Psi=sp_bloco_groover(Psi, qubits, resposta)
        count = count + 1
print(Psi)
print("done")
```

Qubits	tempo
1	0.86s
2	0.92s
3	0.91s
4	0.97s
5	1.04s
6	1.16s
7	1.42s
8	2.16s
9	5.06s
10	19.40s
11	105.38s
12	626.17s
13	353.69s

Tabela 2: Tempos de simulação para diferentes números de qubits utilizando matrizes esparsas

Nota-se a grande melhoria no desempenho e como é possível utilizar um número muito maior de qubits com o mesmo computador.

2.4 Algoritmo de Shor

O algoritmo de Shor é um algoritmo de fatoração de números inteiros. Um dos seus passos é acelerado se executado em um computador quântico. Abaixo os passos do algoritmo:

Dado um número C que se deseja encontrar os fatores:

1. Cheque se C é impar e não é potência de algum inteiro pequeno. Se for um desses encontramos um fator de C e terminamos.
2. Pegue qualquer inteiro no intervalo $1 < a < C$.
3. Encontre o $mdc(a, C)$ (maior divisor comum). Se o mdc for maior que 1 por sorte, encotramos ja um fator de C e terminamos.
4. Encontre o menor inteiro p tal que $a^p \equiv 1 \, mod \, C$. A expressão $p \equiv q \, mod \, C$ quer dizer a congruência modular. Ou seja $p - q$ é um inteiro multiplo de C. Isso pode ser escrito também como $p \, mod \, C = q \, mod \, C$.
5. Se p é impar ou se p é par e $a^{p/2} \equiv -1 \, mod \, C$ volte para 2 e escolha um novo a.
6. Os números $P_{\pm} = mdc(a^{p/2} \pm 1, C)$ são fatores não triviais de C.

O computador quântico é responsável pelo 4º passo desse algoritmo. O resto dos passos como encontrar o mínimo divisor comum (mdc) são feitos rapidamente em um computador clássico.

Esse passo se chama encontrar o período pois $f(x) = a^x \, mod \, C$ é uma função periódica com período p. Dividimos o registrador quântico em duas partes: o registrador x com L qubits iniciado em $|0 \ldots 0\rangle$ e o f com M qubits iniciado em $|0 \ldots 01\rangle$:

O quarto passo:

1. Aplique portas de Hadamard em todos os L qubits do registrador x, representado por $H^{\otimes L}$. Isso coloca o registrador x numa superposição de todos os 2^L estados possíveis

2. Multiplique o registrador f por $a^x \, mod \, C$ fazendo com que fique com o valor de $f(x)$.
3. Meça o registrador f (esse passo é opcional)
4. Faça uma transformada de Fourier quântica inversa (IQFT em inglês) no registrador x. A IQFT permite encontrar o período da função.
5. Meça a saída $\bar{x}$ de do IQFT (lembrando que temos que ler a saída do IQFT na direção contrária). Shor provou que $\bar{x}/2^L$ é igual a aproximadamente s/p onde s é um inteiro qualquer. Usamos isso para encontrar p. Por exemplo $\bar{x}/2^L = 0.32 \cong 1/3 = 2/6 = 3/9$ então p é $3, 6, 9, \cdots$ Checamos em seguida esses valores para ver se $a^p \equiv 1 \, mod \, C$.

```
# Início do programa
a=2
#psi comeca em |0000001>
qubits=7
psi=[0]*(2**qubits)
seta_base(psi,1)

#coloca em superposicao os bits de L
psi=sp_hadamard(psi, 3, qubits) #hadamard em l0
psi=sp_hadamard(psi, 2, qubits) #hadamard em l1
psi=sp_hadamard(psi, 1, qubits) #hadamard em l2
print "qubits L em superposição"
print psi

#parte de f de x
#multiplica por a se l0=1
psi=shor_fx(psi, 1, a, 15, 3, 4)
 #multiplica por a^2 se l1=1
psi=shor_fx(psi, 2, a**2, 15, 3, 4)
#multiplica por a^3  se l2=1
psi=shor_fx(psi, 3, a**4, 15, 3, 4)
# terminado e multiplicando por a^(l2l1l0)

print "-------------------------"
print "matrizes de permutação aplicadas para econtrar f(x)"
print psi

#IQFT transformada de fourier quantica inversa
psi=sp_hadamard(psi, 1, qubits) #hadamard em l2

#mudanca de fase de pi/2 controlada por l2 em l1
psi=sp_Cfase(psi, 2, 1, qubits, math.pi/2)

#mudanca de fase de pi/4 controlada por l2 em l0
psi=sp_Cfase(psi, 3, 1, qubits, math.pi/4)

psi=sp_hadamard(psi, 2, qubits) #hadamard em l1

#mudanca de fase de pi/2 controlada por l2 em l1
psi=sp_Cfase(psi, 3, 2, qubits, math.pi/2)

psi=sp_hadamard(psi, 3, qubits) #hadamard em l0
```

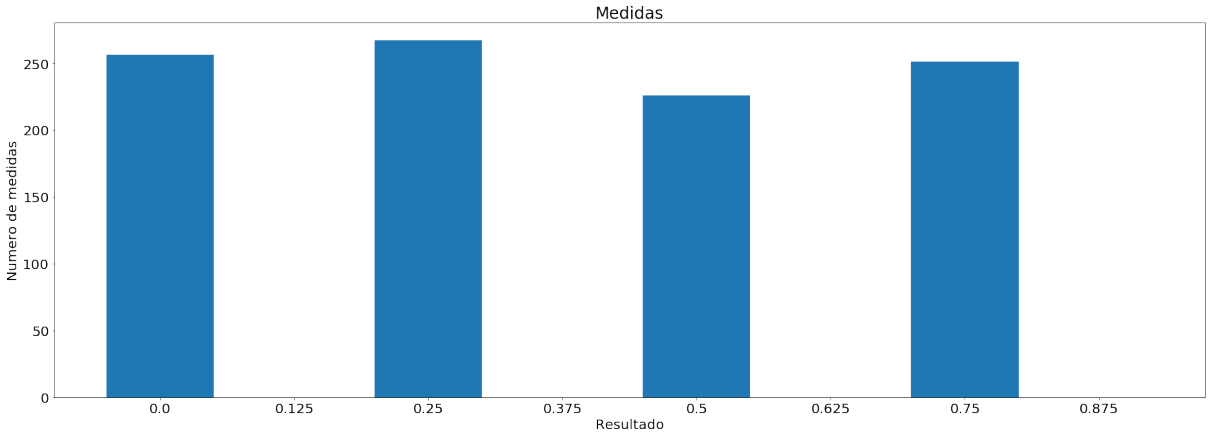

Figura 5: Histograma com valores obtidos para a aplicação do algoritmo de Shor com 7 qubits fatorando 15 com a=2

```
43
44 print "-------------------------"
45 print "transformada de fourier quântica inversa aplicada"
46 print "para encontrar o período da função"
47 print psi
48
49 medir_xbarra_shor(Psi, 1000, 3, 4)
```

Agora medimos o resultado no registrador obtendo vários valores para $\bar{x}/2^L$ onde $\bar{x}$ é o valor do registrador x após a aplicação do IQFT com os bits invertidos. Esses valores são a frequência da função e seus harmônicos $s\omega$ onde s é um inteiro. Ou seja obtemos como medida um valor s/p para a medida de $\bar{x}/2^L$ onde s é um inteiro qualquer e p é o valor procurado do período para o algoritmo de Shor. Como obtemos quatro valores principais podemos concluir que p=4. Na prática, podemos contar o número de picos principais na medidas de saída. 4 picos equivale a um período de 4 por exemplo. Também podemos usar frações parciais para aproximar o valor desejado a partir das resultados da saída do computador quântico.

Também fizemos simulações do algorítimo de Shor para um número N arbitrário de qubits:

```
1  # Inicio do programa
2  a=10
3  L=6
4  M=6
5  C=21
6
7  #psi comeca em |00 ... 01>
8  qubits=L+M
9  psi=[0]*(2**qubits)
10 seta_base(psi,1)
11
12
13 #coloca em superposicao os bits de L
14 j=1
15 while j<=L :
16     psi=sp_hadamard(psi, j, qubits) #hadamard
17     print "had", j
18     j = j + 1
```

```
19
20
21  #parte de f de x
22  j=1
23  while j<=L :
24  #multiplica por a^3  se l2=1 terminado e multiplicando por a^(l2l1l0)
25      psi=shor_fx(psi, j, a**(2**(j-1)), C, L, M)
26      print "fx", j
27      j = j + 1
28
29
30  #IQFT transformada de fourier quantica inversa
31  i=1
32
33  while i <= L:
34
35      psi=sp_hadamard(psi, i, qubits) #hadamard em l2
36      #print "hadamard", i
37      j=1
38      while j <= L-i:
39       #mudanca de fase de pi/2 controlada por l2 em l1
40          psi=sp_Cfase(psi, i+j, i, qubits, math.pi/(2**j))
41          #print i+j, "controls pi/",2**j, "on", i
42          j = j + 1
43      print "IQFT", i
44      i = i + 1
45
46
47  print 'terminado'
48
49  #plota
50  Psi=psi.toarray()[0]
51  medir_xbarra_shor(Psi, 1000, L, M)
```

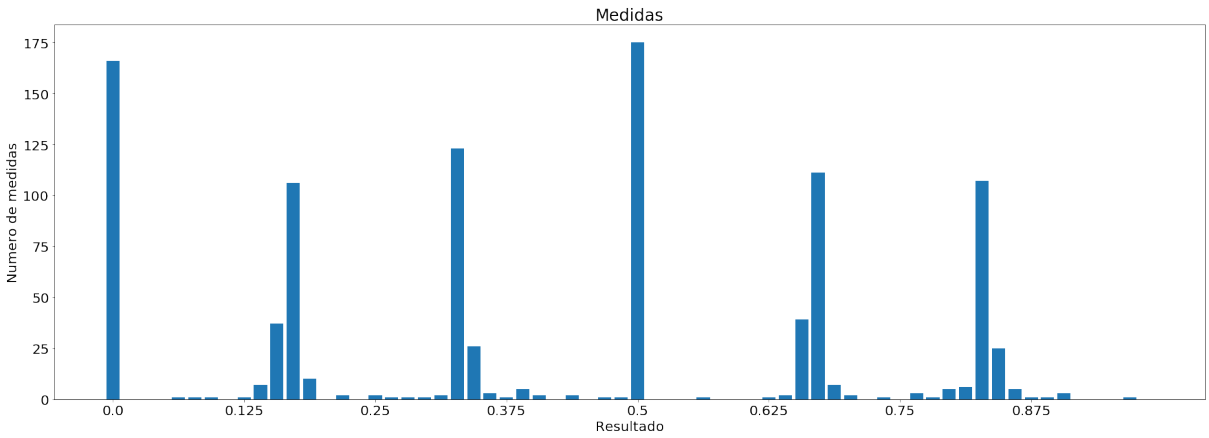

Figura 6: Histograma com valores obtidos para a aplicação do algoritmo de Shor com 12 qubits fatorando 21 com a=10

2.5 Implementação no computador Quântico

Após a simulação do qubits foram utilizados os computadores quânticos disponibilizados pela IBM na IBM Quantum experience [5]. Primeiro serão introduzidas novas portas quânticas que foram utilizadas e depois serão demonstrados os Algoritmos de Groover e Shor executados nessas máquinas. A linguagem utilizada para programar essas máquinas foi o Qiskit que é baseada em python. E que apresenta ferramentas para o desenvolvimento de circuitos para computadores quânticos e gera o QASM que é o Assembly para os computadores quânticos da IBM.

2.5.1 Porta NOT

A porta NOT é simples. Ela inverte as amplitudes dos valores $|0\rangle$ e $|1\rangle$ de um qubit. $X|\Psi\rangle = X(a|0\rangle + b|1\rangle) = b|0\rangle + a|1\rangle$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2}$$

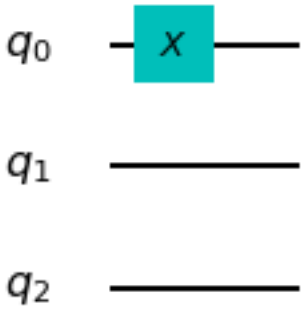

Figura 7: Porta NOT

Para usar uma porta no Qiskit. Declaramos um circuito e inserimos uma porta como a de NOT. A figura da porta é gerada pelo caderno Qiskit de Jupyter.

```
qc= QuantumCircuit(3)
qc.(0)
qc.draw()
```

2.5.2 Porta de Toffoli

A porta de Toffoli é uma porta CNOT controlada por dois qubits.

2.5.3 Porta SWAP e porta de Fredkin

A porta SWAP troca dois qubits de lugar a de Fredkin é uma porta de SWAP controlada(CSWAP).

A Porta de SWAP e construída usando três portas CNOT alternadas.

A porta de Fredkin usa uma porta de Toffoli no lugar da CNOT do meio. Note que duas portas CNOT consecutivas revertem uma a outra e que uma porta de Toffoli com a sua entrada que não será trocada em 0 é o mesmo que se não houvesse nenhuma porta.

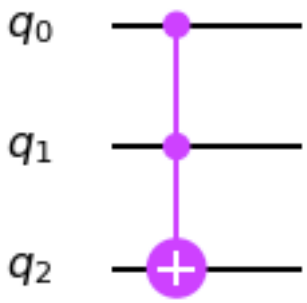

Figura 8: Porta de Toffoli

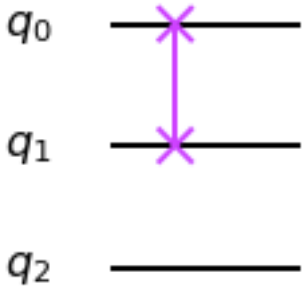

Figura 9: Porta SWAP

2.5.4 Portas de mudança de fase e porta Y

Embora no computador da IBM haja como usar uma porta de mudança de fase com valor de fase arbitrário. Usaremos as portas já configuradas com ângulos específicos.

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{3}$$

$S = T^2$, $\theta = \pi/2$
$Z = T^4$, $\theta = \pi$
$S^\dagger = T^6$, $\theta = -\pi/2$
$T^\dagger = T^7$, $\theta = -\pi/4$
Assim como a porta Y que é uma combinação da porta X com a Z:

$$Y = XZ = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{4}$$

2.5.5 Algoritmo de Groover

Para executar o algoritmo de Groover na máquina da IBM temos duas limitações principais. A primeira é que o algoritmo fica longo a medida que aumentamos o número de qubits. Isso porque precisamos repetir o oráculo e bloco de difusão de Groover que são pedaços do circuito $\pi/4\sqrt{2^N}$ vezes arredondado onde N é o número de qubits. O que já deixa o circuito longo para a máquina da IBM se usarmos 3 qubits. E dependendo da máquina que utilizarmos como a IBM Q 14 Melbourne que tem mais qubits e portanto é mais suscetível a ruídos . Já será mais difícil distinguir a solução.

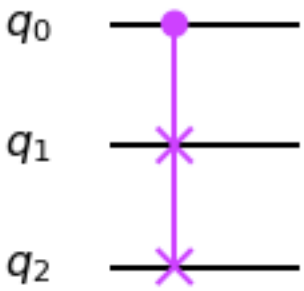

Figura 10: Porta de Fredkin

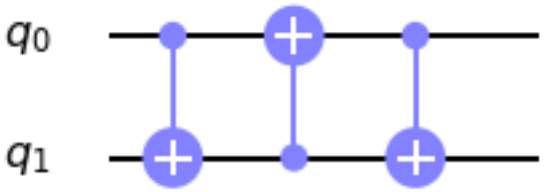

Figura 11: Porta SWAP a partir de portas CNOT

A segunda limitação são as portas que podemos utilizar. Numa simulação com matrizes fica simples fazer uma porta que inverte a fase de apenas um elemento da base mas agora temos que implementar isso com outras portas quânticas fundamentais. Felizmente para três qubits podemos usar uma porta Z controlada por dois qubits que pode ser feita com uma porta de Toffoli e duas de Hardamard nas entradas do qubit que será negado. Essa porta inverte a fase do componente $|111\rangle$ apenas. Para usar em outros elementos podemos colocar portas NOT na entrada e saída dessa porta para que mude a fase de outra combinação de qubits.

Código do algoritmo de Groover em qiskit:

```
#Hadamards em todos os qubits
hadamards=QuantumCircuit(3)
#Inicializa todos os qubits em superposicao
hadamards.h(0)
hadamards.h(1)
hadamards.h(2)
hadamards.draw()

#---------------------------------

resposta=5

#oraculo quantico ##########################################
oracle=QuantumCircuit(3)
#coloca portas X antes dos bits 0 da resposta
if(resposta % 2 == 0):
    oracle.x(0)
if((resposta % 4) //2 == 0):
```

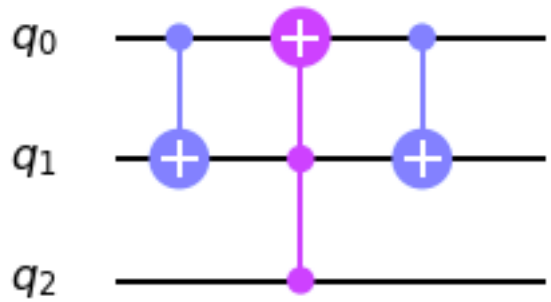

Figura 12: Porta de Fredkin a partir de portas CNOT e de Toffoli

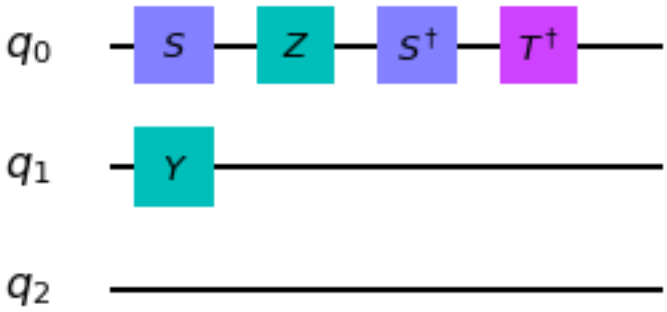

Figura 13: Portas de mudança de fase e porta Y

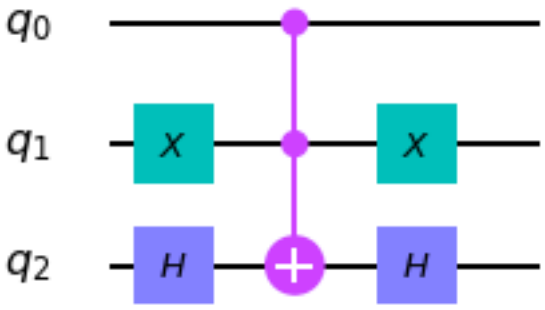

Figura 14: Oráculo quântico para o valor 5 = |101⟩.*NotenasportasXnoqubitque*0

```
19      oracle.x(1)
20  if( (resposta % 8) //4 == 0):
21      oracle.x(2)
22
23  #porta Z em 2 controlada por 0 e 1
24  oracle.h(2)
25  oracle.ccx(0,1,2)
26  oracle.h(2)
27
28  #coloca portas X depois dos bits 0 da resposta
```

```
if(resposta % 2 == 0):
    oracle.x(0)
if((resposta % 4) //2 == 0):
    oracle.x(1)
if( (resposta % 8) //4 == 0):
    oracle.x(2)
oracle.draw()
#-------------------------------------
#Operador de difusão de groover
difusao=QuantumCircuit(3)
#hadamards
difusao=difusao+hadamards

#portas x para o estado |000>
difusao.x(0)
difusao.x(1)
difusao.x(2)

#porta Z controlada
difusao.h(2)
difusao.ccx(0,1,2)
difusao.h(2)

difusao.x(0)
difusao.x(1)
difusao.x(2)

#hadamards
difusao=difusao+hadamards

difusao.draw()

#---------------------------------
#parte de medida do circuito
mdir=QuantumCircuit(3,3)
grvc=QuantumCircuit(3)
mdir.measure(range(3),range(3)) #circuito para a medicao dos qubits

#oraculo e o operador de difusao sao repetidos 2 vezes
grvc=hadamards+oracle+difusao+oracle+difusao+mdir

grvc.draw()
```

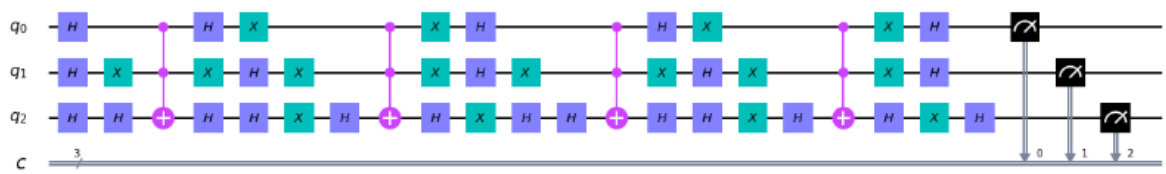

Figura 15: Circuito de Groover para três qubits

O programa foi executado duas vezes. A primeira foi uma simulação. Logo

apresenta o resultado sem ruídos. Código para simular o circuito:

```
#simulacao de um computador quantico
backend_sim = Aer.get_backend('qasm_simulator')
job_sim = execute(grvc, backend_sim, shots=1024)
result_sim = job_sim.result()
plot_histogram(result_sim.get_counts(grvc))
```

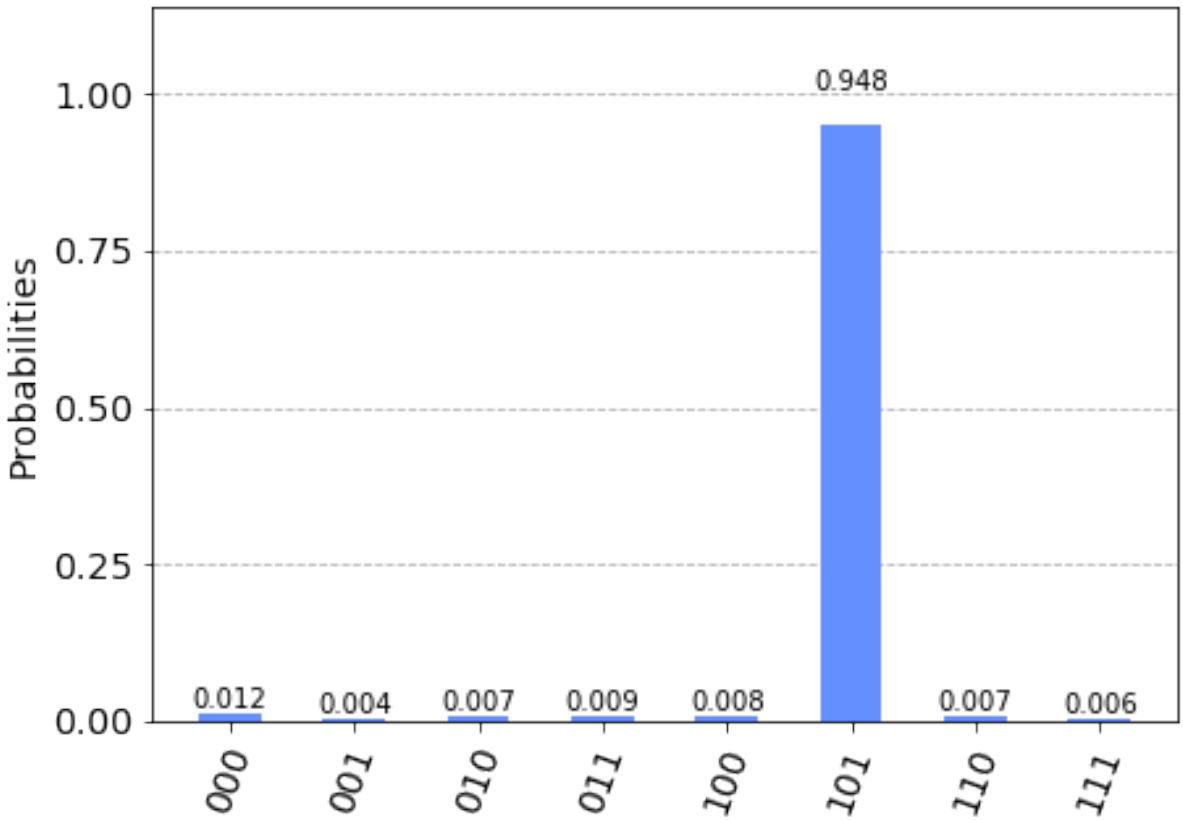

Figura 16: Resultado da simulação de QASM num computador clássico da IBM

A segunda é a execução no computador quântico. Podemos notar como aparecem mais das outras medidas que não são o resultado:

Código para executar o circuito na máquina quântica da IBM:

```
#executa o circuito em um computador quantico real
from qiskit.tools.monitor import job_monitor
backend = provider.get_backend('ibmqx2')

job = execute(grvc, backend=backend, shots=4096)
job_monitor(job)

##################################
resultado_comp = job.result()
plot_histogram(resultado_comp.get_counts(grvc))
```

2.5.6 Algoritmo de Shor

No algoritmo de Shor a maior limitação para a implementação em um computador quântico real é a porta quântica que faz a multiplicação por $a^x \, mod \, C$. Já

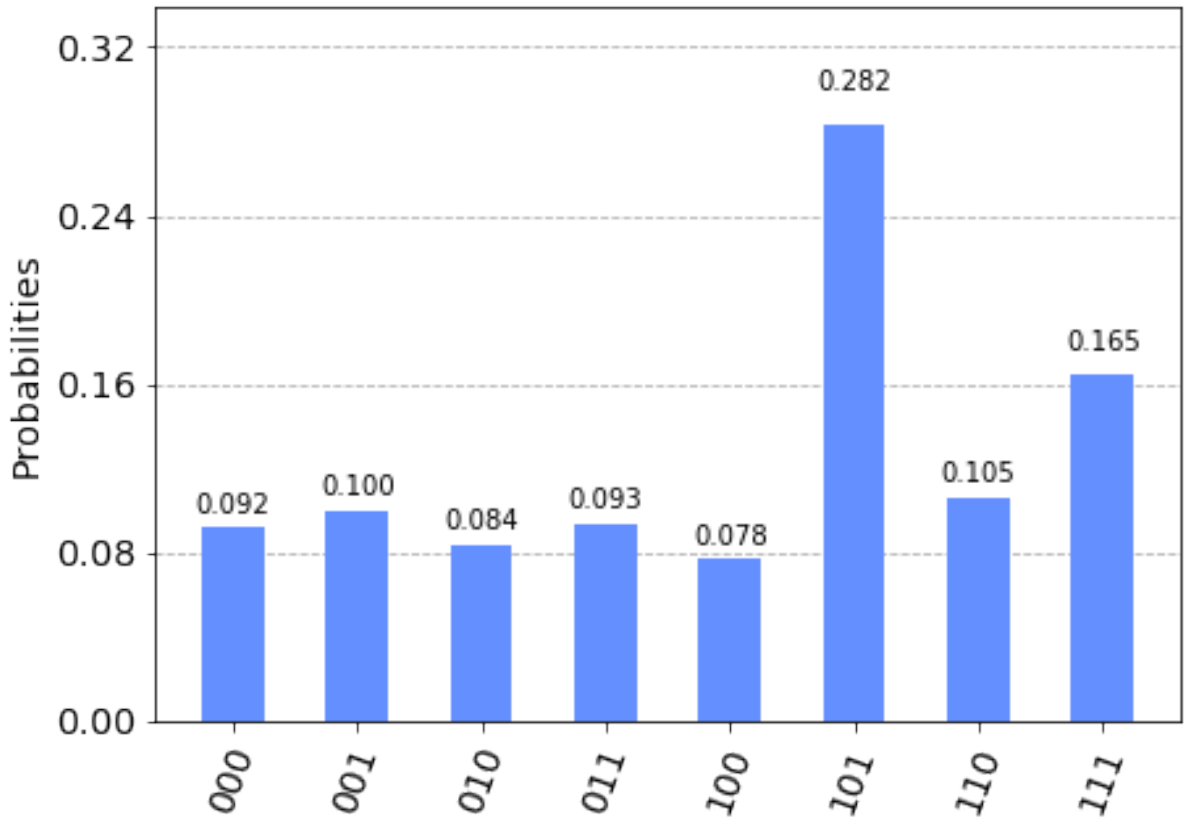

Figura 17: Resultado do programa com algoritmo de Grover rodado no computador quântico ibmqx2

que temos que implementá-la através de outras portas mais fundamentais e não diretamente através da matriz como na simulação. A transformada de Fourier Quântica é simples de implementar assim como a superposição inicial.

Primeiro implementei um circuito mais simples com 5 qubits usando $C = 15$ e $a = 4$ que têm um período de apenas 2 o que permite encolher bastante o circuito.

Para fazer o multiplicador por $a^x \, mod \, 15 = 4^x \, mod \, 15$. Dividimos ele em duas partes: $4^1 \, mod \, 15$ controlado pelo qubit de x menos significativo e $4^2 \, mod \, 15$ controlado pelo mais significativo. Assim se o qubit de x menos significativo for 1 multiplicamos f por $4 \, mod \, 15$ e o mesmo para o mais. Obtendo assim $4^{x_0+x_1} \, mod \, 15$. Como $4^2 \, mod \, 15 = 16 \, mod \, 15 = 1$ precisamos fazer apenas o do bit menos significativo. Para multiplicar o número 1 por 4 sempre que o qubit menos significativo de x for 1 basta usar uma porta CNOT que zera a porta com 1 do digito menos significativo com peso 1 e uma que seta o qubit com peso 4.

```
hadamards=QuantumCircuit(5)
#Inicializa os qubits do registradores de L em superposicao
hadamards.h(3)
hadamards.h(4)
hadamards.draw()
#######################################
#multiplicador
mult=QuantumCircuit(5)
mult.x(0)#inicializa o registrador f em 1

mult.cx(3,2)#seta q2(f2) se q3(x0) for um
mult.cx(3,0)#zera q0(f0) se q3(x0) for um
```

```
mult.draw()
################################
#transformada de Fourier quantica
from math import pi
iqft=QuantumCircuit(5)
iqft.h(4)
iqft.crz(pi/2, 4, 3)
iqft.h(3)
iqft.draw()

###################################
#parte de medida

mdir=QuantumCircuit(5,2)
shor=QuantumCircuit(5)
mdir.measure([3, 4],[1,0]) #circuito para a medicao dos qubits

#oraculo e o operador de difusoa sao repetidos 2 vezes
shor=hadamards+mult+iqft+mdir

shor.draw()
```

Para rodar o circuito usamos o mesmo código do algoritmo de Groover só que agora no computador quântico **ibmq 16 melbourne**.

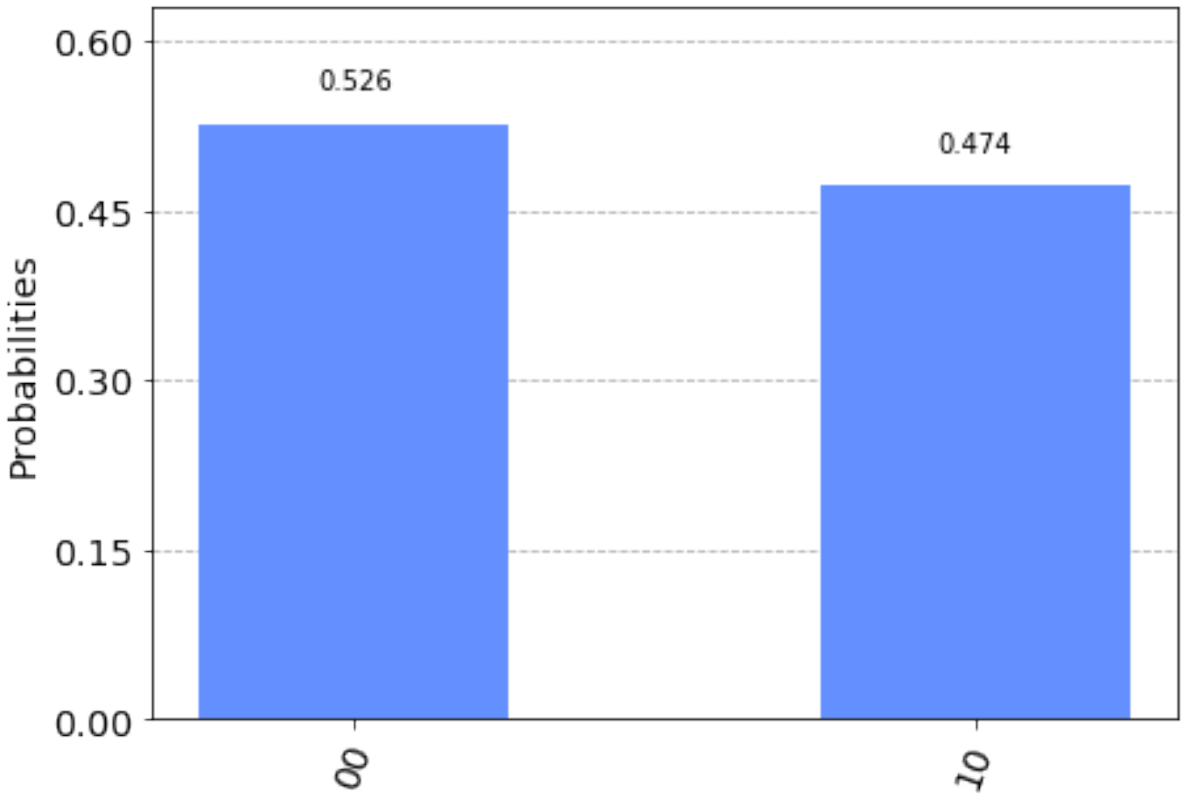

Figura 18: Resultado da simulação do Algoritmo de Shor com C=15 a=4

Agora executamos o algoritmo com a=2 o que vai requerer 6 qubits. O novo multiplicador é assim: $a^x \, mod \, 15 = 2^x \, mod \, 15$. Dividimos ele em 3 partes: $2^1 \, mod \, 15$ controlado pelo qubit de x menos significativo, $2^2 \, mod \, 15$ controlado pelo qubit do

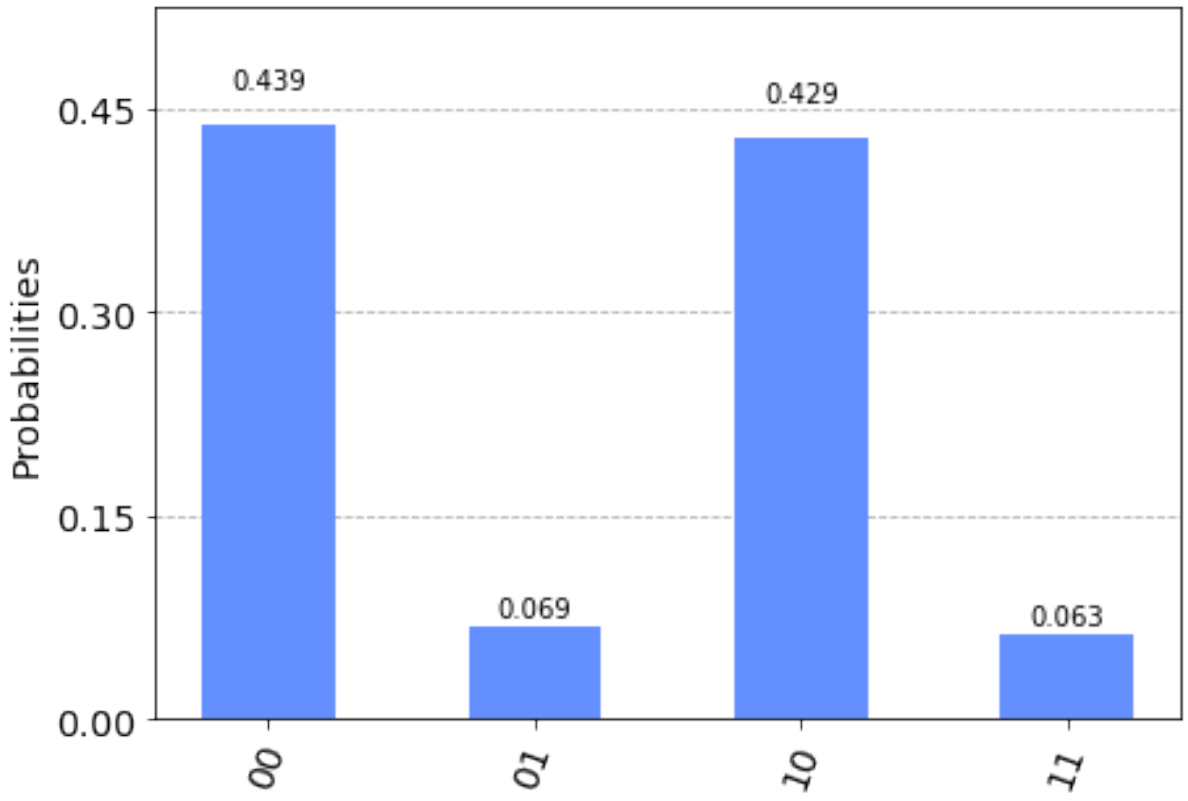

Figura 19: Resultado da execução do Algoritmo de Shor com C=15 a=4 no **ibmq 16 melbourne**

meio e $2^4 \, mod \, 15$ controlado pelo mais significativo. Assim se o qubit de x menos significativo for 1 multiplicamos f por $2 \, mod \, 15$ e o mesmo para os outros. Obtendo assim $4^{x_0+x_1+x_2} \, mod \, 15$. Como $2^4 \, mod \, 15 = 16 \, mod \, 15 = 1$ precisamos fazer apenas o do meio e o do qubit menos significativo. Para multiplicar um número por 2 módulo 15 vamos usar três portas de Fredkin. Para números binários, a multiplicação por dois é um deslocamento para a esquerda. Como fazemos o módulo 15 do valor também note que os números dão a volta pela esquerda de volta a direta, como uma rotação. Para fazer esse deslocamento usamos portas de Fredkin para fazer SWAPs controlados nos qubits de f. Para multiplicar por $2^2 = 4$ o circuito vai funcionar de maneira semelhante. Só que com um deslocamento de dois qubits.

```
hadamards=QuantumCircuit(7)
#Inicializa os qubits do registradores de L em superposicao
hadamards.h(4)
hadamards.h(5)
hadamards.h(6)
hadamards.x(0)#inicializa o registrador f em 1
hadamards.draw()
############################
#multiplica f por 2^1 mod 15
mult21=QuantumCircuit(7)

#fredkin controlado por q4(x0) em q2(f2) e q3(f3)
mult21.cx(2,3)
mult21.ccx(4,3,2)
mult21.cx(2,3)

```

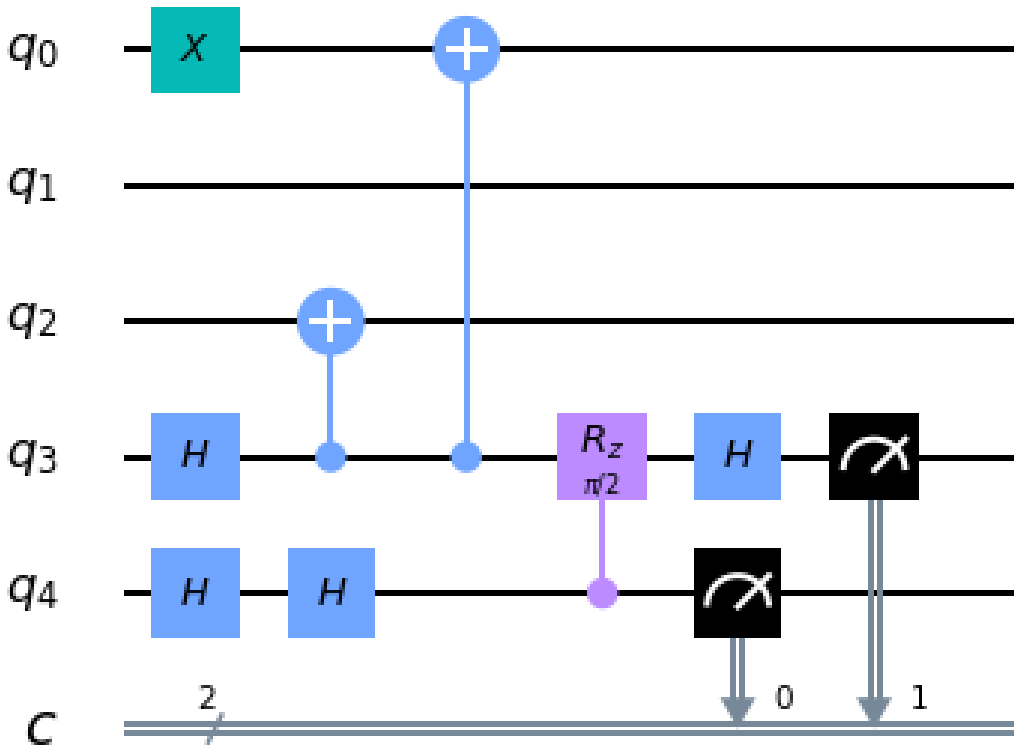

Figura 20: Circuito do Algoritmo de Shor com C=15 a=4

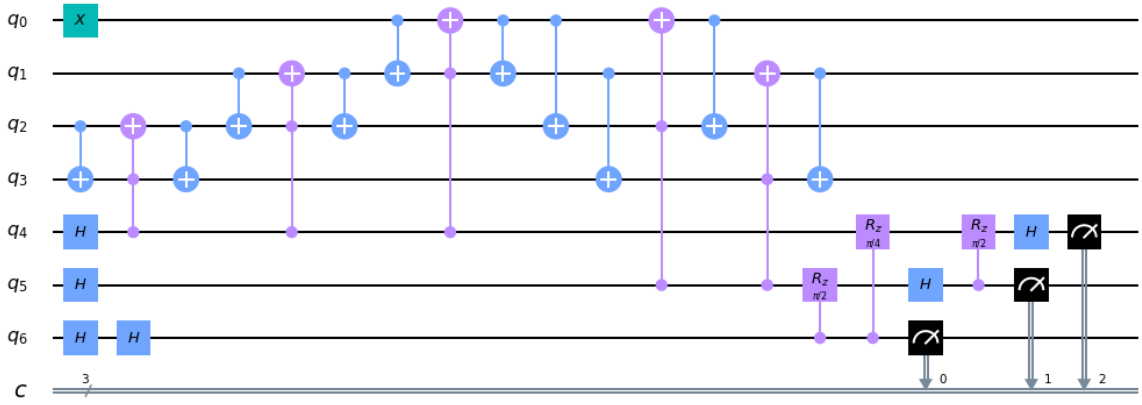

Figura 21: Circuito do Algoritmo de Shor com C=15 a=2

```
17 mult21.cx (1,2)
18 mult21.ccx (4,2,1)
19 mult21.cx (1,2)
20
21 mult21.cx (0,1)
22 mult21.ccx (4,1,0)
23 mult21.cx (0,1)
24
25 mult21.draw()
```

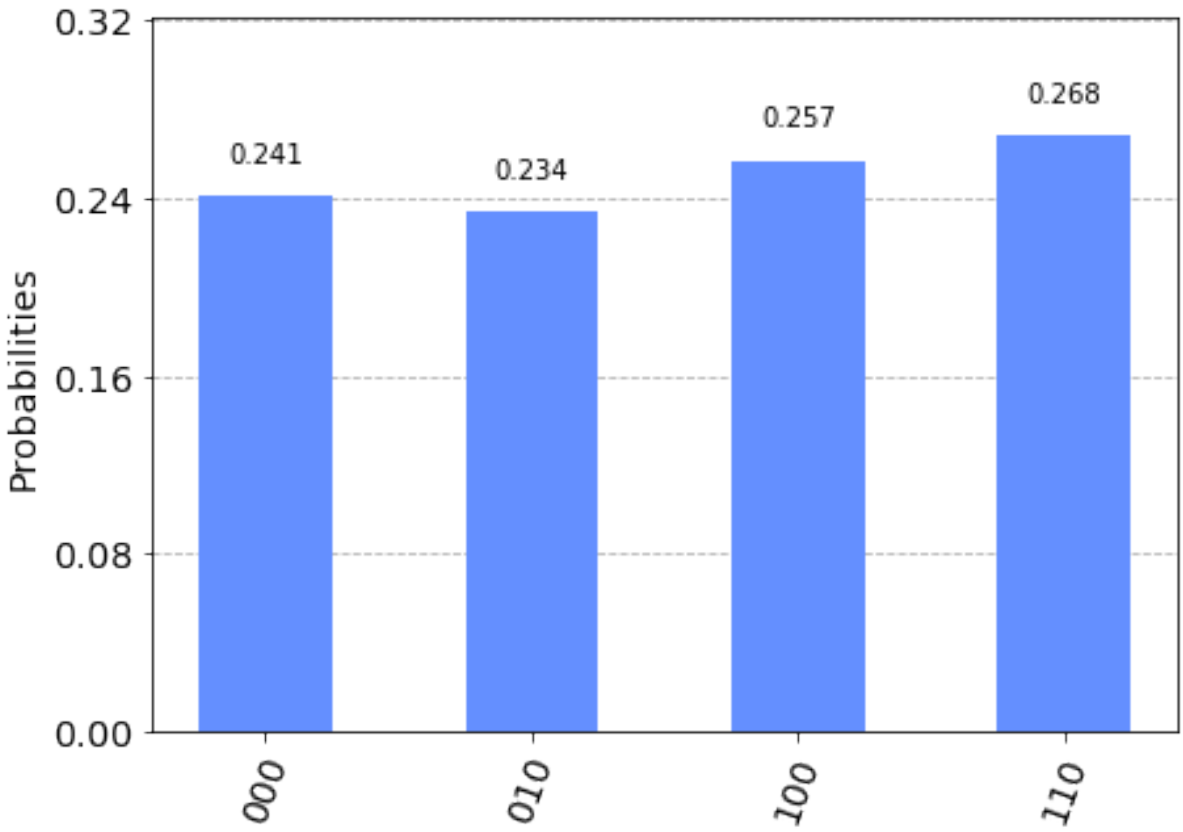

Figura 22: Resultado da simulação do Algoritmo de Shor com C=15 a=2

```
##################################
#multiplica f por 2^2 mod 15
mult22=QuantumCircuit(7)

#fredkin controlado por q5(x1) em q0(f0) e q2(f2)
mult22.cx (0,2)
mult22.ccx (5,2,0)
mult22.cx (0,2)

mult22.cx (1,3)
mult22.ccx (5,3,1)
mult22.cx (1,3)

mult22.draw()
###################################
#IQFT em x

from math import pi
iqft=QuantumCircuit(7)
iqft.h(6)
iqft.crz(pi/2, 6, 5)
iqft.crz(pi/4, 6, 4)

iqft.h(5)
iqft.crz(pi/2,5, 4)

```

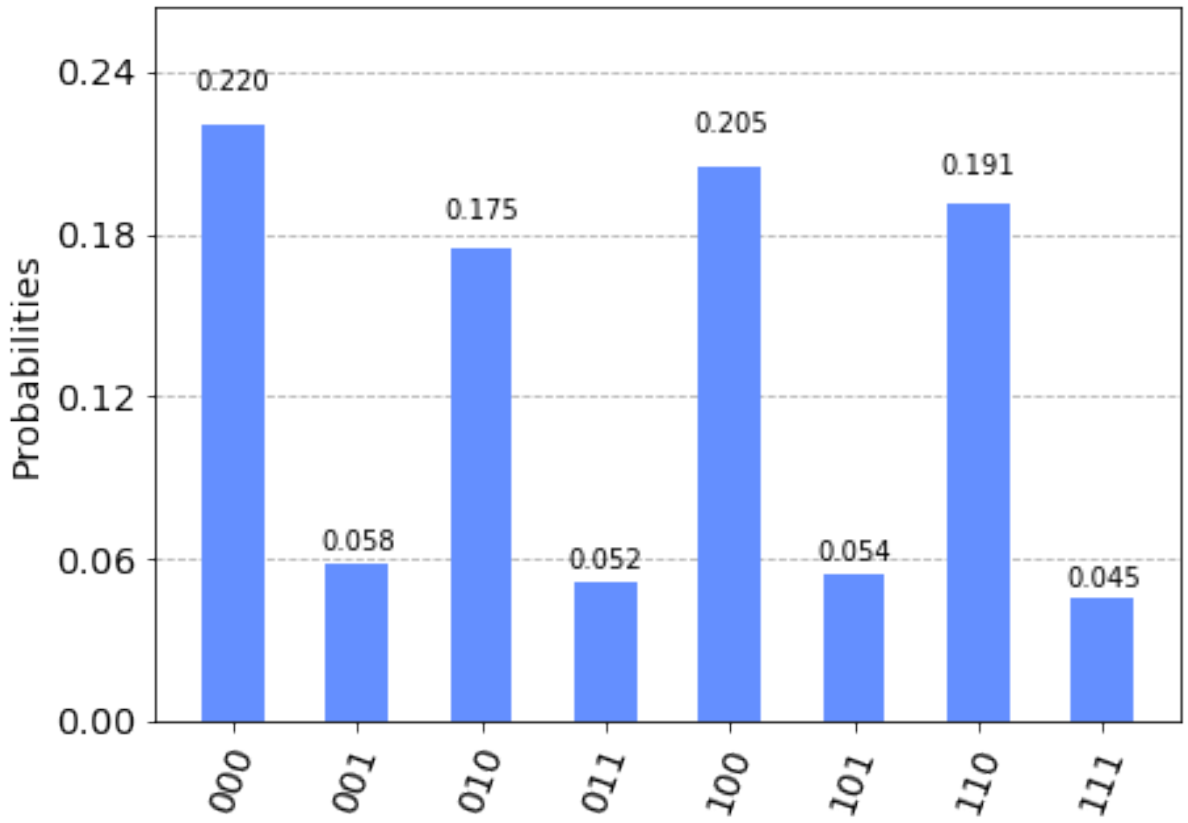

Figura 23: Resultado da execução do Algoritmo de Shor com C=15 a=2 no **ibmq 16 melbourne**

```
iqft.h(4)
iqft.draw()

 ###################################
 #medida

mdir=QuantumCircuit(7,3)
shor=QuantumCircuit(5)
mdir.measure([4, 5, 6],[2,1,0]) #circuito para a medicao dos qubits(bits de medida ja

#oraculo e o operador de difusoa sao repetidos 2 vezes
shor=hadamards+mult21+mult22+iqft+mdir

shor.draw()
```

3 Conclusões

Ao longo da realização do projeto, tivemos contato com diversas áreas do conhecimento, relacionado principalmente com a computação quântica. Dentro da própria física, conceitos como a superposição quântica e fases da função de onda foram revisitados para melhor compreensão, como também um estudo aprofundado dos algorítimos de Groover de busca e Shor de fatoração de números inteiros. Desenvolvemos as implementações destes algorítimos em Python e fizemos simulações para diversas situações. Utilizando os computadores quânticos da IBM implementamos os algoritmos de Shor e Groover e comparamos os resultados com

nossa simulação clássica, obtendo resultados similares.

Em conclusão, desenvolvemos com sucesso todo o processo de implementação, simulação e análise de algorítimos quânticos, e fizemos um estudo de tópicos avançados de mecânica quântica com enfase em computação quântica.

Referências

[1] Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the Annual ACM Symposium on Theory of Computing*, Part F1294:212–219, 5 1996. ISSN 07378017. doi: 10.1145/237814.237866. URL http://arxiv.org/abs/quant-ph/9605043.

[2] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 10 1997. ISSN 00975397. doi: 10.1137/S0097539795293172. URL https://epubs.siam.org/doi/10.1137/S0097539795293172.

[3] Guido Van Rossum and Fred L Drake. The Python Language Reference 2.6.2. page 109, 2011.

[4] Thomas Kluyver, Benjamin Ragan-kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. *Jupyter Notebooks—a publishing format for reproducible computational workflows*. 2016. ISBN 9781614996491. doi: 10.3233/978-1-61499-649-1-87. URL https://www.medra.org/servlet/aliasResolver?alias=iospressISBN&isbn=978-1-61499-648-4&spage=87&doi=10.3233/978-1-61499-649-1-87.

[5] IBM. IBM Q Experience, 2019. URL https://quantumexperience.ng.bluemix.net/qx/experience.