Estudo de algoritmos quânticos e suas implementações

Giancarlo Ponte Gamberi, Calebe de Paula Bianchini

Universidade Presbiteriana Mackenzie – Faculdade de Computação e Informática
São Paulo, SP – Brazil

giangamberi@hotmail.com.br, calebe.bianchini@mackenzie.br

Abstract. *In this work, many quantum algorithms will be analyzed, among them, some will be selected, they will then receive an in-depth analysis of how the problem works and how it can be adapted to its process in a quantum computer, and a quantum solution using python`s qiskit library will be elaborated.*

Resumo*. Neste trabalho serão analisados diversos algoritmos quânticos, entre eles, alguns serão selecionados, estes então receberão uma análise aprofundada de como funciona o problema e de como ele pode ser adaptado para seu processo em um computador quântico, e caso seja possível, uma proposição de uma solução quântica para o algoritmo será proposta.*

1. Introdução

Atualmente a computação quântica é tema central de pesquisa em ciência básica e tecnologia de fronteira. Nas duas últimas décadas, o interesse pela área vinha da motivação de que os algoritmos quânticos fornecem ganhos computacionais consideráveis, em relação aos análogos clássicos. Como resultado, nos últimos anos experimentamos avanços tecnológicos que nos permitem começar a contornar alguns problemas práticos para a fabricação escalável dos computadores quânticos. [BRYLINSKI; CHEN, 2002].

Com a aproximação da supremacia quântica* torna-se cada vez mais interessante e necessário o desenvolvimento de algoritmos, sejam estes exclusivos a computação quântica, ou sejam problemas conhecidos e com solução clássica, para serem executados utilizando o novo paradigma apresentado pelos novos processadores.

Tendo esta necessidade em mente, esta pesquisa consistiu na pesquisa de algoritmos quânticos, entendendo-os a fundo, empregando um deles e por fim tentando propor um novo algoritmo com base no implementado. O algoritmo selecionado durante a pesquisa é o algoritmo de busca de Grover, e dois algoritmos quânticos diferentes foram propostos para tentar solucionar o problema de busca.

2. Computação quântica

Antes de iniciarmos de fato nosso estudo, é necessário entendermos o estado da arte e alguns conceitos empregados pelo novo paradigma e que foram utilizados durante o trabalho, estes conceitos, por serem relacionados a uma tecnologia nova, ainda são pouco disseminados, e, portanto, devem ser conceituados antes de iniciarmos o estudo.

A começar com os *qubits*, ou bit quânticos como podem ser referenciados, são o equivalente aos *bits* dos computadores clássicos em computadores quânticos, a grande diferença deles para sua versão clássica, é que estes somente assumem duas posições clássicas, logicamente 0 e 1, um *bit* necessariamente está em uma dessas posições, os

*Conforme definida por John Preskill (2012) em “Quantum computing and the entanglement frontier”

qubits em contrapartida podem assumir uma superposição, podendo operar em qualquer estado dentro desta superposição. A representação logica desta superposição é feita através de uma esfera de Bloch.

Um *qubit*, no entanto, diferente dos *bits* que sempre sabemos seus estados, é necessário ser medido para termos sua posição, e nesta medição o estado onde o *qubit* estava é perdido, e o resultado não nos retorna à posição deste *qubit*, e sim a probabilidade de ele estar no estado 0 ou 1, portanto apesar da maior capacidade computacional que um *qubit* nos proporciona, temos de ter cuidado com a saída dos dados, já que a leitura do processamento desfaz o processamento anterior.

Conforme mencionado, a esfera é a representação lógica da superposição de um *qubit* (Figura 1), esta esfera possui 3 eixos: x, y e z. Porém o eixo que merece de fato atenção é o eixo z, pois nas extremidades deste que se encontram os estados 0 e 1, onde é calculada a probabilidade durante a medição, note que com isso, apesar da liberdade de trabalharmos em qualquer quadrante dos eixos x e y, no final é a posição em z que vale para a medição.

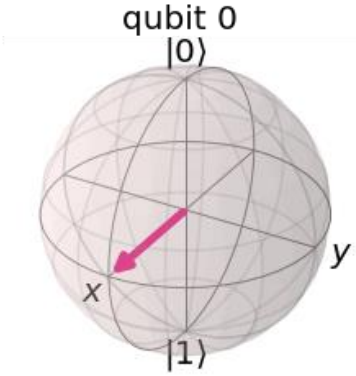

Figura 1. Esfera de Bloch gerada por uma função do qiskit, a seta avermelhada apontando para X representa o atual estado do *qubit*, note que este está em uma superposição uniforme, pois em relação ao eixo z (que não é explicitamente representado, sendo o eixo com os extremos sendo representados por |0⟩ e |1⟩) ele está exatamente no centro.

Outro conceito que temos que ter em mente quando falamos em computação quântica, é o de oráculos, este não é um conceito novo e nem exclusivo da computação quântica, porém é um conceito frequentemente empregado quando falamos de algoritmos quânticos, e, portanto, necessitamos ter um vislumbre para entender melhor o tema.

Um oraculo se trata de uma "caixa preta" dentro de um algoritmo, a ideia é que durante o processo, nossos dados passem por esta "caixa" e saiam dela, sem termos total ideia doque foi feito nela, se os dados foram processados ou não, e qual o tipo de processo que foi realizado. Na prática, e aqui consideramos como foi comumente utilizado nos algoritmos pesquisados, e como podemos pensar para facilitar o entendimento, se trata de um algoritmo desconhecido, onde por vezes é mencionado sua funcionalidade. Um exemplo fictício, mas baseado nos que serão vistos, seria a menção de um oraculo que ordena um vetor dentro de nosso algoritmo, nós não precisamos saber necessariamente o algoritmo utilizado para ordenação desse vetor, o que nos interessa é o fato de haver a necessidade dentro de nosso algoritmo, um outro que ordene nosso vetor.

Por último, vamos falar da *Qiskit*, como é chamada a biblioteca para python e desenvolvida pela IBM para simulação de algoritmos e circuitos quânticos, será a biblioteca que usaremos para implementação dos algoritmos, usando a biblioteca podemos criar circuitos quânticos e executa-los, seja via simulação (a biblioteca oferece alguns tipos de simulação diferentes, neste caso usaremos o mais utilizado e recomendado pela própria documentação da biblioteca, o simulador *aer*), ou via um processador quântico em nuvem da IBM (o processo de execução deste é demorado, por conta de utilizarmos uma fila publica de utilização, portanto majoritariamente utilizaremos a simulação).

2.1. Estado da Arte

Explicados alguns conceitos, é importante entendermos o atual estado da arte onde os algoritmos quânticos se encontram, por isso abordaremos e exploraremos diversos algoritmos quânticos e seu atual estado de pesquisa e desenvolvimento.

O local de exploração destes algoritmos é o Quantum Algorithm Zoo, um repositório gerido por Stephen Jordan da área de Microsoft Quantum e que o atualiza com frequência o repositório, os algoritmos são classificados em 4 tipos, sendo estes: “algébricos e número teóricos”, “oraculares”, “aproximação e simulação”, e por fim “otimização, numéricos, e aprendizado de máquina” (classificações traduzidas direto do repositório).

Começando pelos algoritmos algébricos e número teóricos, escolhemos já de início o de fatoração, cujo problema tradicional que conhecemos de fatoração de números inteiros possui uma solução para ser executada em modelos quânticos que foi proposta por Peter Shor desde 1994, e apesar de ainda estarem sendo estudadas variantes mais otimizadas que executem em tempo ainda melhor, esta solução apresenta crucial importância no atual cenário, pois é com este algoritmo de fatoração que podemos, em tempo hábil, e utilizando um computador quântico, quebrar a criptografia RSA, que por conta de sua importância, nos aprofundaremos no entendimento deste algoritmo durante a terceira parte deste trabalho.

O próximo algoritmo da nossa lista é cálculo do logaritmo que, conforme mostrado por Shor, pode ser calculado em tempo polinomial utilizando curvas elípticas, quebrando portando criptografias baseadas em curvas elípticas, não existe nenhum algoritmo clássico que chegue perto do tempo polinomial do algoritmo quântico, e por conta dessa diferença, nós estudaremos mais a fundo este algoritmo também.

Outro problema tradicional que também possui sua versão quântica, porém diferente dos problemas até então apresentados, o problema da soma dos subconjuntos, o tempo de resolução quântico não se mostra tão diferente em relação ao clássico mais rápido, se comparado com outros algoritmos quânticos e suas versões clássicas.

O primeiro de nossos algoritmos oraculares e selecionado para implementarmos. Este problema consiste em enviar um conjunto N de inputs permitidos a um oraculo, para qual um deles será aceito, retornando 1, enquanto os outros retornariam 0, o objetivo é encontrar justamente o aceito, a grosso modo podemos comparar este problema a um algoritmo de forca bruta, mas não nos limitando a este conceito, o algoritmo quântico de Lov Grover conseguiu reduzir significativamente o número de queries necessárias para se executar tal problema, e como a generalização daquele, que ficou conhecida como estimação de amplitude, se tornou um conceito importante para algoritmos quânticos, e

conforme mencionado na introdução deste trabalho, este foi escolhido para implementarmos a solução de Grover e propormos alternativas para suas resolução.

Em sequência temos o algoritmo de interpolação polinomial, neste algoritmo, o problema constitui, não em calcular o polinomial, este fica a cargo do oraculo, mas sim em separar as queries para serem enviadas ao oraculo, sendo que o algoritmo com polinomiais simples opera quase com metade das queries de um clássico.

Para problemas combinatórios otimizados, muitas vezes compensa muito mais utilizar resultados aproximados com margem de erro pequenas do que procurar a solução ótima em si, e com isso abre espaço para uma versão quântica do problema de otimização aproximada para gerar aproximações mais rapidamente e com margem de erro ainda menor, e apesar de um algoritmo clássico mais eficiente ter sido descoberto por consequência da versão quântica, ainda é um problema que está sendo pesquisado e desenvolvido para a área quântica.

O cálculo de funções Zeta em computadores clássicos, leva um tempo exponencial, enquanto em computadores quânticos o tempo é polinomial. Foi notado também pelo autor do repositório que devido a conexão entre os zeros das funções Zeta de Riemann e os autovalores de certos operadores quânticos, computadores quânticos podem ser capazes de aproximar eficientemente o número de soluções para equações em campos finitos.

Seguindo para os algoritmos de aprendizagem de máquina, onde o próprio autor do repositório evita mencionar somente um por conta dos diferentes tipos de problemas e algoritmos de aprendizado de máquina, neste caso muitos dos algoritmos quânticos inicialmente desenvolvidos, foram convertidos de volta para versões clássicas por conta destes conseguirem serem resolvidos em tempo polinomial, porem alternativas e técnicas ainda estão sendo desenvolvidos para tentar melhorar a performance quântica, sendo atualmente um dos focos de estudo da computação quântica, porém não nos aprofundaremos muito nestes algoritmos neste trabalho.

O algoritmo de programação dinâmica quântico é apresentado no repositório com um problema chamado path-in-the-hypercube (caminho no hipercubo), e muitos dos algoritmos clássicos que se utilizam de programação dinâmica podem ser modelados como instancias daquele, o autor menciona também que com a utilização da busca de Grover em conjunto com técnicas da programação dinâmica, o problema pode ser resolvido significativamente mais rápido em relação a sua versão clássica.

3. Estudo dos algoritmos

Enquanto analisávamos o atual estado da arte dos algoritmos, notamos três algoritmos que merecem destaque especial, os dois primeiros: fatoração e log discreto, onde em ambos se destaca Shor, por ter apresentado soluções precoces para estes, e que desafiam alguns dos mais importantes algoritmos de criptografia atuais como a RSA e criptografias que utilizam curvas elípticas. O terceiro algoritmo, o de busca, onde neste se destaca Grover, ganha foco pela utilidade deste, você pode ter notado que alguns outros algoritmos mencionados utilizam o algoritmo de Grover em sua resolução.

É sabido desde antes de Euclides que todo inteiro n é unicamente decomposto em um produto de primos. Matemáticos tem estado interessado na questão de como fatorar um número nesse produto de primos por tanto tempo quanto. Foi apenas na década de

1970, no entanto, que pesquisadores aplicaram os paradigmas da teoria da ciência da computação a teoria dos números e observaram os tempos de execução assintóticos da fatoração de algoritmos [Shor 1996, traduzido].

Peter W. Shor propôs um algoritmo quântico para solução do problema da fatoração ainda em 1996, e apesar de já existirem versões melhores atualmente vamos focar na solução proposta por Shor por ser referência quando se trata de fatoração em um computador quântico.

O algoritmo proposto por Shor consistem em, dado um número par n, e escolhido um número aleatório x (mod n), devemos calcular a ordem r de x tal que $x^r \equiv 1 \pmod{n}$, e então calculamos o grande divisor comum entre $x^{\frac{r}{2}} - 1, n$ utilizando o algoritmo euclidiano, o número encontrado então é um fator de n, repete-se então o processo até que n tenha sido inteiramente fatorado, e antes de continuarmos para a parte onde como isto seria implementado em um computador quântico, devemos notar alguns pontos de tal algoritmo: Um número aleatório x (mod n) (ou seja, um número entre 0 e n) é escolhido pois, estatisticamente é mais vantajoso do que se cada número do grupo mod n fosse tratado individualmente e sequencialmente, em seguida, calculamos $x^r \equiv 1 \pmod{n}$ pois quando aplicado ao máximo divisor comum valida que o número obtido é um fator de n. As provas matemáticas podem ser encontradas no artigo de Shor [6].

Antes de revisarmos como o algoritmo pode ser adaptado para um computador quântico, primeiro temos que entender a transformada de Fourier quântica, que nada mais é que a transformada rápida de Fourier adaptada para a operação quântica, que consiste no uso de duas portas logicas quânticas e na execução na ordem correta de tais portas, estas são: $R_j = \begin{vmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{vmatrix}$ e $S_{j,k} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta_{k-j}} \end{vmatrix}$ onde $\theta_{k-j} = \pi/2^{k-j}$.

E a ordem correta de execução, é a aplicação de R_j em ordem reversa de R_{l-1} a R_0, e entre cada iteração entre R_{j+1} e R_j, aplicamos todas as portas $S_{j,k}$ onde $k > j$, ou seja: $R_{l-1}\, S_{l-2,l-1}\, R_{l-2}\, S_{l-3,l-1}\, S_{l-3,l-2}\, R_{l-3} \ldots R_1\, S_{0,l-1}\, S_{0,l-2} \ldots S_{0,2}\, S_{0,1}\, R_0$.

Portanto, quando for mencionada a aplicação da QFT (Quantum Fourier Transform, a transformada de Fourier quântica recém introduzida), estamos aplicando esta sequência de operações.

Seguindo então para o algoritmo de fatoração para um computador, dados x e n, calculamos q, a potência de 2 para $n^2 \leq q < 2n^2$, tais variáveis então servirão de base para o sistema quântico. Iniciamos com dois registradores quânticos, no primeiro, precisamos aplicar uma superposição uniforme de estados que representam os números a$(mod\ q)$, enquanto no segundo registrador nós calculamos $x^a(mod\ n)$ reversivamente a partir do primeiro registrador, e em sequência, aplicamos a QFT no primeiro registrador medindo então o primeiro registrador para obtenção do resultado, encerrando a parte quântica, mas ainda necessitando de uma tratativa ainda em um computador clássico, os resultados devem ser convertidos para decimais, a fração formada por um dos decimais c dividido por q, se reduzida para um denominador menor que n, nos retorna a fração d/r, com o valor de r podemos então calcular o grande divisor comum entre $x^{\frac{r}{2}} - 1, n$, que nos retorna um fator de n.

Uma observação que necessita ser apontada, existe a possibilidade que, dos diferentes valores de r retornados, alguns podem não ser ideais, porém as probabilidades demonstradas por Shor não serão abordadas neste trabalho.

Shor demonstrou no mesmo artigo como seria o log discreto em um computador quântico utilizando 3 registradores. Seguindo o algoritmo proposto por Shor, o log discreto de x que respeite um número primo p e g um gerador tal que $1, g, g^2, \dots, g^{p-2}$ que comprime todos os resíduos não zero (mod p) é dado por r tal que $g^r \equiv x \pmod{p}$. Inicialmente encontramos q, uma potência de 2 tal que q é próximo de p com $p \leq q < 2p$ e em seguida colocamos os dois primeiros registradores em superposição uniforme, e no terceiro computamos $g^a x^{-b} \pmod{p}$, para em seguida aplicar a transformada de Fourier Quântica, conforme vimos anteriormente, e por fim observamos o circuito.

Dos estados observados, nós então os classificamos como "bons" ou "maus", para classificarmos, existe uma condição que precisa ser cumprida, para identificarmos se um estado é "bom", em caso positivo, nós podemos recuperar um candidato r através de uma fórmula utilizando os estados medidos e a quantidade de estados na condição "bom", e caso este candidato r falhe na verificação, basta repetirmos o circuito em uma quantidade polinomial de vezes até que encontremos o candidato r correto. A fórmula de obtenção de r, a condição para definição de um estado "bom" e a prova da polinomialidade de repetições são demonstradas por Shor, mas não as abordaremos neste trabalho.

Imagine uma lista Telefônica contendo N nomes, arranjados em ordem aleatória. Para se encontrar o telefone de alguém com probabilidade de ½, qualquer algoritmo clássico (seja ele determinístico ou probabilístico) vai necessitar observar no mínimo N/2 nomes. [Grover 1996, traduzido]

Grover em 1996 propôs um algoritmo de busca para ser aplicado em sistemas quânticos para o clássico problema de busca em um vetor não ordenado, o vetor neste caso, serão os estados dos *qubit*.

Conforme vimos em seções anteriores, este algoritmo é classificado como oracular, ou seja, haverá uso de um oraculo no meio do algoritmo, este oraculo, por definição, se trata de uma "caixa preta", ou seja, algo pelo qual nosso sistema passara, que pode ou não executar algum processamento ou alteração em nosso sistema, na pratica, podemos definir como algum algoritmo (seja ele conhecido ou não, ou que tenha alguma finalidade clara, ou puramente para teste) que será executado anteriormente ao algoritmo de busca de Grover, cujo algum resultado especifico deverá ser encontrado.

O algoritmo inicia-se com a aplicação do oraculo em nosso sistema uniformemente superposto, a próxima etapa necessita ser executado $O(\sqrt{N})$, onde N é a quantidade de *qubits*:

Os estados dos quais queremos buscar, nos rotacionamos por π radianos, enquanto os outros deixamos inalterados, em sequência aplicamos a matriz de difusão, definida por Grover como $D = WRW$, onde W seria a porta Hadamard, usada mais frequentemente para colocar o sistema em estado superposto, e R é definido como uma matriz diagonal, onde a diagonal pode ser definida como um vetor onde o primeiro elemento é 1, e o restante -1:

$R_{i,j} = 0 \text{ se } i \neq j$

$R_{i,i} = 1 \text{ se } i = 0$

$R_{i,i} = \ -1 \text{ se i} \neq 0$

Terminado o algoritmo, medimos o sistema, e podemos observar que o estado que foi rotacionado logo no início, possui uma probabilidade de pelo menos ½.

Este algoritmo de Grover se tornou amplamente utilizado em conjunto com outros algoritmos por incrivelmente facilitar encontrar um estado específico que esteja rotacionado π radianos, e por conta de sua ampla utilização, este será o algoritmo implementado na seção III deste trabalho.

4. Elaboração de uma Solução

Conforme mencionado, o algoritmo elaborado foi o algoritmo de Busca, a começar pela implementação do algoritmo proposto por Grover, esta aplicação utilizou-se de um notebook python ipynb, e para a parte quântica a biblioteca *qiskit*.

Inicialmente executamos o algoritmo com um oraculo que simplesmente vai marcar (aqui, marcar se refere a rotação de π radianos que é aplicada ao estado que se deseja encontrar) algum estado em superposição de escolha aleatória, e aplicamos a matriz de difusão, onde em seguida observamos os resultados, após tais etapas, um oraculo mais elaborado, com um exemplo mais prático foi elaborado, para termos uma melhor noção da capacidade e usabilidade do algoritmo.

Uma vez feita a importação das bibliotecas *qiskit*, numpy (para auxiliar na manipulação dos dados) e pyplot, do matplotlib (para visualização dos resultados) (Código 1), e como mencionado anteriormente foi selecionado um estado qualquer, criamos através da biblioteca um circuito quântico, neste exemplo, usamos um circuito de 3 *qubits* (Código 1).

```
1.      # numero de qubits
2.      n=3
3.      # criacao do circuito
4.      algoritmoGrover = QuantumCircuit(n)
5.
```

Código 1. Trecho necessário para declaração do nosso circuito, este chamado de algoritmoGrover, com n = 3 *qubits*

Aqui, chegamos a algumas maneiras de realizar a rotação necessária, o método ideal de se executar, seria usando uma porta Z, que realiza justamente a operação de que necessitamos, porém para o nosso caso de 3 *qubits*, rotacionar o qubit gera uma situação indesejada com mais de um estado marcado, por conta disto preferimos aplicar uma matriz diagonal especificando somente um estado.

Para a aplicação de matrizes diagonais ao nosso circuito a biblioteca *qiskit* possui uma função simplificada para tal, necessitando somente dos *qubits* onde será executada, e um vetor que representa a diagonal da matriz.

Para especificar o estado, desenvolvemos a função *converteNumeroDiagonalOraculo* (Código 2) que recebe um número inteiro que represente um estado, e o número de *qubits* do sistema, e retorna um vetor pronto para ser executado em conjunto com a função *diagonal* da biblioteca, a aplicação podemos

observar no Código 4, o circuito gerado do oraculo na Figura 2 e a leitura dos estados nas figuras 2 e 3.

```
1.  def converteNumeroDiagonalOraculo(number,  qubits):
2.      if (2**qubits) < (number-1):
3.          return -1
4.      aux = np.ones(2**qubits,dtype=int)
5.      aux[number] = -1
6.      return aux
```

Código 2. Função elaborada para o trabalho conforme mencionado acima

```
1.  #aplica hadamard em um circuito para a lista de qbits especificada
2.  def inicializaSobreposicao (qc, qubits):
3.      for q in qubits:
4.          qc.h(q)
5.      return qc
```

Código 3. Função auxiliar para aplicar a porta Hadamard nos *qubits* especificados, quando falamos de superposição uniforme, falamos da aplicação destas portas

```
1.  #Variaveis para Oraculo Teste
2.  numeroProcurado = 4
3.  groverDiagonal = converteNumeroDiagonalOraculo(numeroProcurado,n)
4.
5.  #inicializando/oraculo teste
6.  algoritmoGrover = inicializaSobreposicao(algoritmoGrover,allQbits)
7.  algoritmoGrover.diagonal(list(groverDiagonal),allQbits)
8.  algoritmoGrover.draw()
```

Código 4. A preparação do nosso circuito para procurar o estado 100

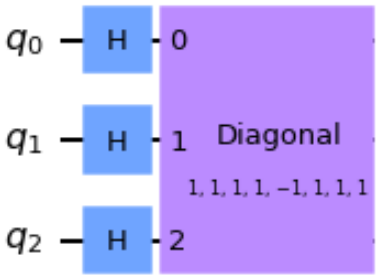

Figura 2. Representação em forma de diagrama utilizada pelo próprio *qiskit* de nosso oraculo

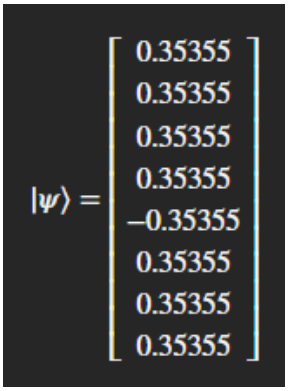

Figura 3. Vetor de estados que foi obtido através do simulador *aer*, do *qiskit*, note que todos os estados estão idênticos, com exceção do estado 100, que está negativo, e é aquele que queremos buscar.

Chegando então no núcleo do algoritmo de Grover, precisamos aplicar a matriz de difusão, que conforme mencionada é definida como D = WRW, sendo W a aplicação da matriz que deixa os estados em superposição uniforme, que se trata da aplicação da porta Hadamard nos *qubits* necessários, se notarem bem, a nossa função auxiliar *inicializaSobreposicao* (Código 3) executa exatamente esta operação, portanto somente nos resta a matriz R, que precisamos elaborar.

Como a matriz R se trata de uma matriz diagonal, aplicamos a função *diagonal* já embarcada na biblioteca, ou seja, elaboramos o vetor que represente a diagonal, para isso, criamos usando o numpy um vetor de tamanho 2^n preenchido com números 1, onde n é quantidade de *qubits* (no caso deste exemplo: 3) e invertemos o sinal de todos, com exceção do primeiro (Código 5).

Agora que temos W e R, podemos aplicar sem problemas a matriz D, e medir os resultados da execução, conforme o código 5 abaixo:

```
1.    def matrizDifusao(qc, qubits):
2.        # D = WRW
3.        # Matriz de difusao = Hadamard -> diagonal[1,-1,-1,-1] -> Hadamard
4.        R = np.ones(2**len(qubits),dtype=int)
5.        for i in range(1,2**n):
6.            R[i] = -1
7.        inicializaSobreposicao(qc,qubits)
8.        qc.diagonal(list(R), qubits)
9.        inicializaSobreposicao(qc,qubits)
10.       return qc
```

Código 5. Função que aplica a matriz de difusão ao circuito

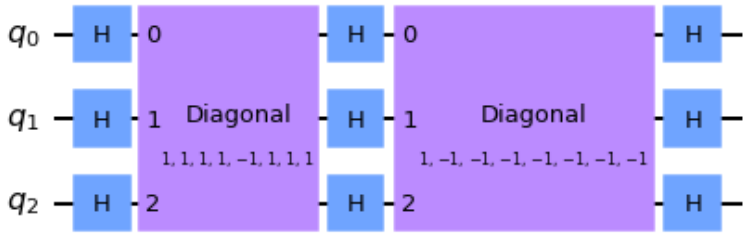

Figura 4. Circuito depois de aplicado o oraculo e a matriz de difusão

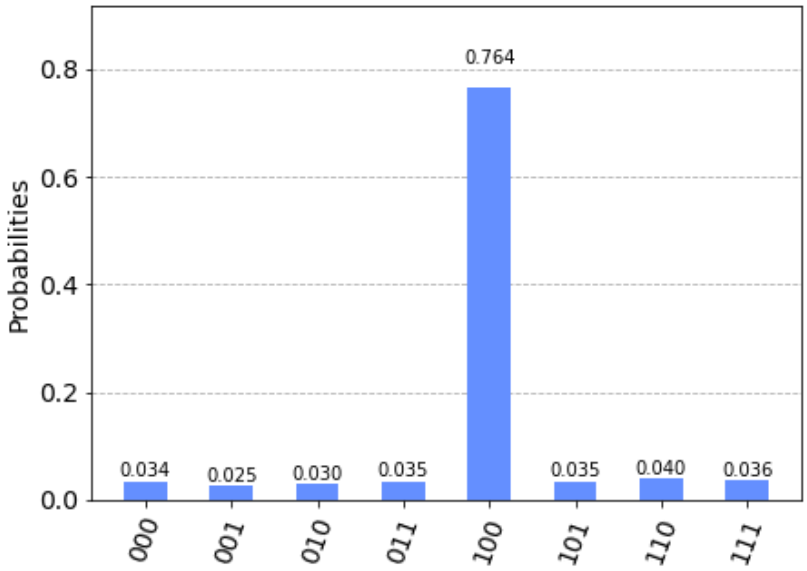

Figura 5. Plot da simulação da observação dos resultados

Como podemos observar na figura 5, o algoritmo funcionou como o esperado, o estado desejado foi rotacionado e encontrado, tornando muito simples encontrarmos qualquer estado desejado, podendo nos dar liberdade para trabalharmos com os estados livremente, bastando ao final rotacionar os estados para a posição certa e aplicarmos a matriz de difusão, que quando medirmos, podemos claramente encontrar qual o resultado correto.

Uma propriedade interessante do algoritmo de Grover, é a capacidade de podermos marcar mais de um estado, por isso, reexecutamos o teste, porém desta vez com dois estados rotacionados.

Para rotacionar os dois estados, ao invés de aplicarmos a diagonal como descrita anteriormente, podemos usar a porta CZ (*controled Z*, ou Z controlada), que se trata da porta de rotação Z, mas controlada por um outro *qubit*, para nos limitarmos a somente os estados desejados, que nesse caso serão os estados 011 e 111 (Código 6).

```
1.    algoritmoGrover = inicializaSobreposicao(algoritmoGrover,allQbits)
2.    algoritmoGrover.cz(0,1)
```

Código 6. Versão do algoritmo para o novo teste proposto

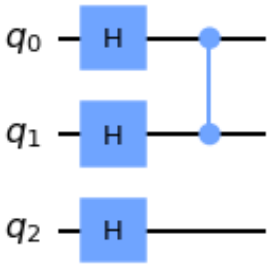

Figura 6. Novo Diagrama de nosso oraculo

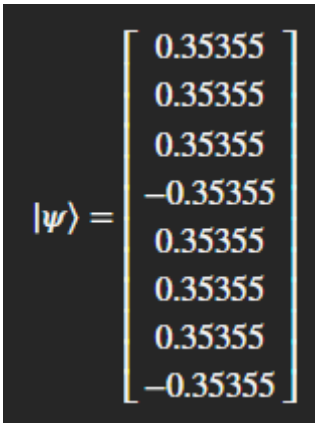

Figura 7. Vetor de estados gerado pela simulação, podemos observar os dois estados rotacionados, são eles que desejamos buscar

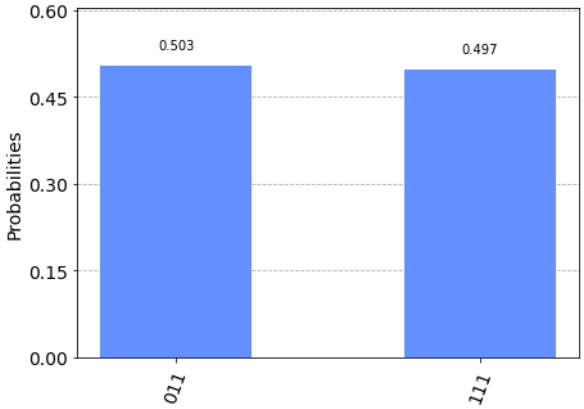

Figura 8. Plot do resultado da simulação observada, após aplicarmos a matriz de difusão

Pelos resultados deste segundo teste, observamos a possibilidade do algoritmo de Grover não se limitar a um estado de busca, mas múltiplas buscas, contanto que estes estejam devidamente rotacionados.

A seguir, veremos duas alternativas que foram elaboradas, a primeira, que se utiliza das probabilidades medidas nos *qubits* (os estados) em conjunto com portas quânticas, e a segunda, que se utiliza das rotações nos *qubits*, ambas tendo como entrada um vetor desordenado contendo números de 0 a 15, e um número, também de 0 a 15, que seria o número a ser buscado em tais vetores, o *range* 0 a 15 foi escolhido para simular a base hexadecimal, mantendo um *range* próximo ao utilizado classicamente, de forma que

facilitasse o entendimento do ocorrido no circuito, evitando perda de precisão que *ranges* maiores causariam.

A primeira proposta, que chamamos de algoritmo de busca quântico por probabilidade, ou ABQP para facilitar, e o segundo, chamado de algoritmo de busca quântico por rotações, ou ABQR, por trabalhar com rotações nos *qubits*.

4.1 Algoritmo de busca quântico por probabilidade

A ideia desenvolvida neste algoritmo é de codificar os números do vetor de entrada no vetor de estados, o vetor de entrada precisa ter $2^n - 1$ elementos, para que na última posição do vetor de estados seja posicionado o número a ser buscado, a porta proposta então subtrairia cada estado pelo estado buscado, resultando então em um vetor de estados onde o número buscado teria medição zero.

Para aplicar tal algoritmo iniciamos montando um vetor 2^n, onde a última posição contém nosso número buscado, enquanto o restante das posições possui números de um vetor, com este vetor montado, podemos iniciar a tratativa para fazê-lo representar amplitudes de um circuito, e então, utilizando a função *StatePreparation* do *qiskit*, termos um circuito que tenha este vetor como seu vetor de estados.

A normalização para deixar o vetor na sua forma de estados pode ser realizada dividindo cada elemento pela raiz da soma dos elementos quadráticos, este vetor normalizado é traduzido para o circuito utilizando a função *StatePreparation* do *qiskit*.

A porta logica quântica elaborada que cumpriria com a subtração das probabilidades, seria uma matriz n por n onde a diagonal teria valor 1, os valores da última linha teriam valor -1, e o restante das posições seria composto por 0, a representação desta matriz se dá por:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix}$$

Porém, tal porta precisa também ser transformada, para ficar no formato de uma matriz unitária para que possamos utilizá-la no nosso circuito, para tal conversão, utilizamos o procedimento de Gram-Schmidt, e aplicamos ao circuito que por sua vez fica pronto para ser executado.

Alguns testes foram realizados com este algoritmo proposto, no caso o número de *qubits* utilizado foi 3, o vetor a ser buscado foi aleatoriamente estruturado como a seguir: $[7,12,4,15,3,13,11]$ e o número de busca foi sendo alterado para se analisar os resultados nas simulações. O portão logico e sua matriz unitária, respectivamente para este caso de teste:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0.7071 & -0.4082 & -0.2887 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.8165 & -0.2887 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.8660 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.8944 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9129 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9258 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9354 & 0.3536 \\ -0.7071 & -0.4082 & -0.2887 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \end{pmatrix}$$

Nos primeiros testes executados, utilizando a mesma configuração de simulação utilizada no algoritmo de Grover (isto é, utilizando o simulador *aer* do *qiskit*), o algoritmo infelizmente não demonstrou uma boa precisão após passar pelo portão, gerando muita flutuação e resultados insatisfatórios (Figura 9).

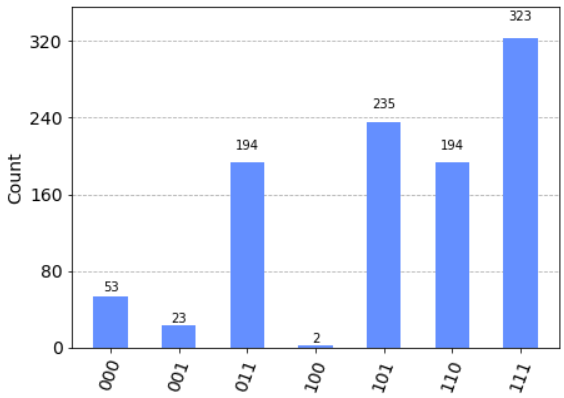

Figura 9. Resultados da simulação buscando o número 7 (Primeira posição do vetor)

Na Figura 9 acima vemos o resultado da medição do circuito final, conforme descrito o número buscado deveria ter a probabilidade mais baixa, nesse caso, 000, porém observamos claramente que não foi o caso, já que 001, 010 e 100 tiveram probabilidades mais baixas. Nos vetores de estado 1 e 2 abaixo, observamos o comportamento dos estados do nosso algoritmo durante a execução.

$$[0.25032, 0.42912, 0.14304, 0.5364, 0.10728, 0.46488, 0.39336, 0.25032]$$

Vetor de estados 1. Codificado nas probabilidades, note que o último elemento é igual ao primeiro, pois o valor 0.25 seria a representação do número 7.

$$[-0.2148, 0.13376, -0.05145, 0.42439, 0.06214, 0.46633, 0.45646, -0.56881]$$

Vetor de estados 2. Depois do circuito passar pelo portão proposto, o resultado esperado.

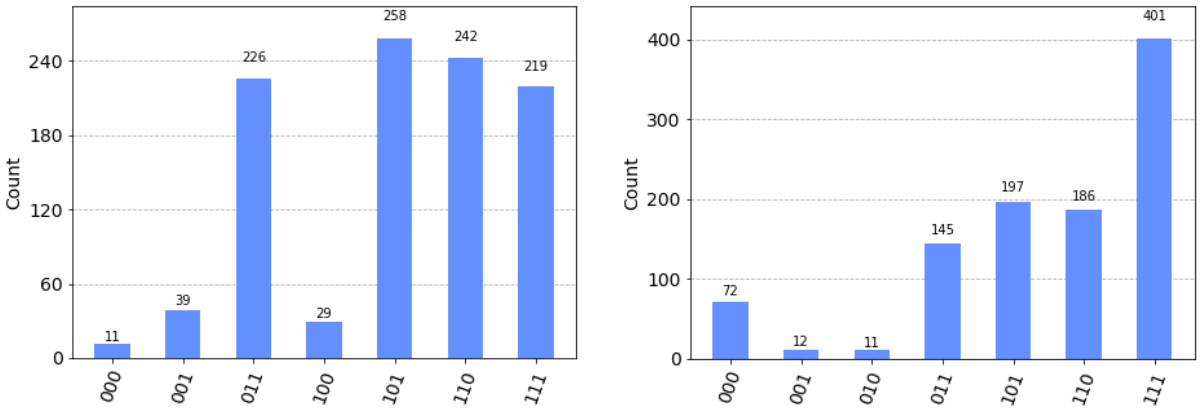

Figura 10. Resultados das buscas dos números 3 e 13 no mesmo vetor, como podemos ver, os resultados não são os esperados.

Observando os estados, concluímos que o problema está no portão proposto, que por ter sido convertido para uma matriz unitária perdeu a precisão que seria necessária para nos dar um resultado correto, porém durante a elaboração de possíveis correções para esta porta, foi pensado na ideia de utilizar-se das rotações dos *qubits* para guardar e manipular os dados, ao invés de usarmos os estados das medições, por conta de um melhor desempenho desta segunda alternativa, o foco maior será nele.

4.2 Algoritmo de busca quântico por rotação

A ideia deste algoritmo, é de codificar os números do vetor em cada *qubit* ao longo do perímetro do primeiro quadrante do eixo y do *qubit*, uma vez esses dados codificados, aplicamos uma segunda rotação igual para cada *qubit*, com ângulo definido pelo número buscado, de tal forma que o *qubit* que guarda o número correspondente ao que estamos buscando, fique em uma superposição uniforme, e por fim aplicamos portas H em cada um, o nosso *qubit* desejado então teria leitura em 100% de chance de estar em 0, enquanto outros *qubits* com valores diferentes dos buscados não trarão tamanha certeza na medição.

Visualmente, isto é, convertendo nossos *qubits* para a visualização de suas respectivas esferas de Bloch conseguimos entender melhor o algoritmo proposto, por isso foi elaborada a visualização etapa a etapa, utilizando a função do *qiskit* que gera uma visualização das esferas de Bloch do nosso circuito podemos acompanhar o resultado de cada rotação aplicada.

Ainda considerando a regra utilizada anteriormente, trabalhamos com números de 0 a 15, se normalizarmos nosso vetor para tal *range*, conforme executado anteriormente, e multiplicarmos por 90 graus radianos ($\frac{\pi}{2}$) obtemos os graus radianos que precisamos para codificar os dados nos *qubits*, tal rotação é facilmente alcançada utilizando o portão *ry* do *qiskit* (o resultado desta etapa em um circuito de teste podemos observar na figura 11, note como a seta representado a posição do *qubit* se desloca ao longo da circunferência do eixo *y*).

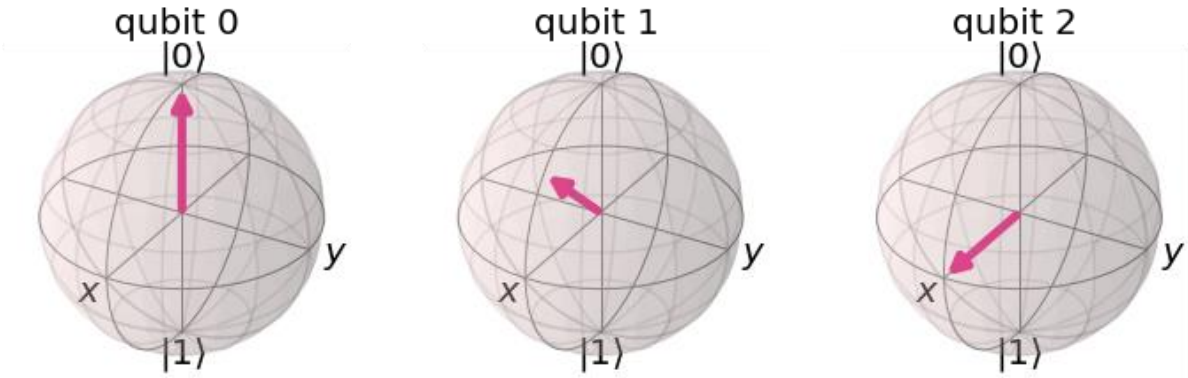

Figura 11. Acima três esferas de Bloch representando três *qubits* diferentes durante nossa etapa de codificação, contendo os valores 0, 8 e 15, respectivamente.

A seguir, aplicamos a mesma regra de normalização e conversão para radianos no nosso número a ser buscado, e extraímos seu complemento, para tal, podemos subtrair nosso número por 15 antes de normalizá-lo (o número que subtraímos se dá pelo *range* sendo trabalhado), ou subtraí-lo por 90 graus depois de convertido a radianos, independente a abordagem, precisamos obter o módulo deste. Com o número a ser buscado devidamente tratado, aplicamos uma rotação em y em cada *qubit* com tal valor (Na figura 12 observamos esta aplicação, note como o único *qubit* em superposição uniforme é o *qubit* 1).

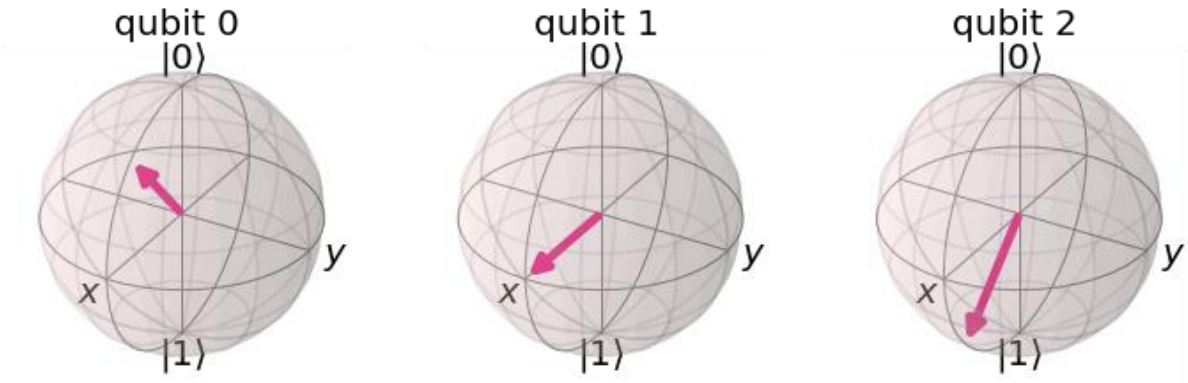

Figura 12. Etapa de "busca" do número 8 a ser encontrado.

Por fim aplicamos portas hadamard em cada *qubit*, isto fará com que o nosso *qubit* buscado desfaça a superposição, enquanto os outros ainda a manterão, assim quando medirmos nosso circuito, o *qubit* buscado será o único com certeza de estar em 0, portanto sendo facilmente identificável (Como demonstrado na figura 13 e na medição da figura 14).

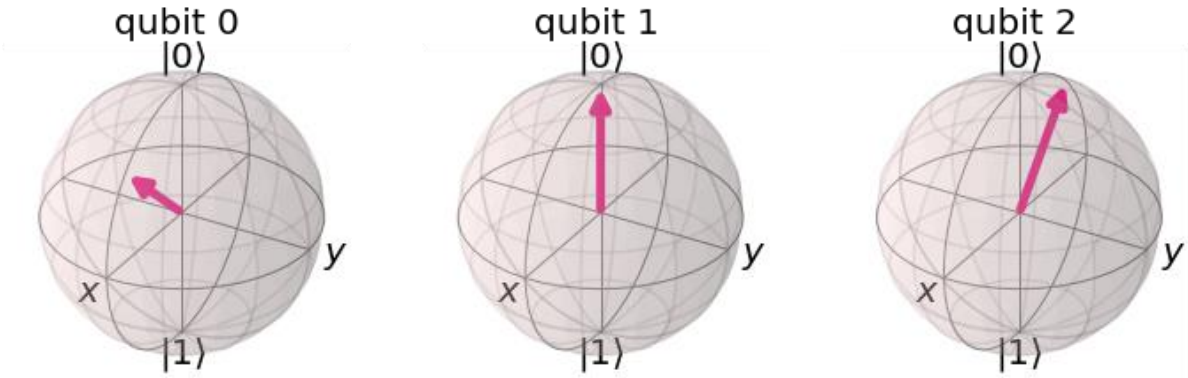

Figura 13. Circuito da figura 12 após passarmos cada *qubit* pela porta hadamard.

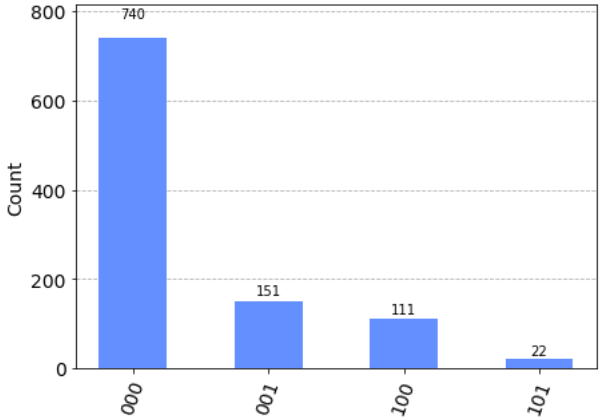

Figura 14. Resultado da medição do circuito exemplo utilizando o simulador *aer*.

Diversos testes foram elaborados utilizando este algoritmo, alterando para vários cenários tanto o número buscado quanto o vetor que seria buscado, apresentando resultados suficientemente satisfatórios, como observamos nas figuras 15 e 16.

Analisando o algoritmo e os resultados, notamos que números com valores próximos do que buscamos podem apresentar falsos positivos, enquanto números mais espaçados mantém uma acurácia muito melhor, isto poderia ser resolvido se reduzíssemos o *range* dos números utilizados, já que como vimos anteriormente, quando codificamos os números nos *qubits*, *ranges* menores aumentariam o ângulo de diferença entre os números, e, portanto, aumentado a precisão nestes casos em que temos números muito próximos.

Outro ponto que podemos identificar dos resultados, é a capacidade de um *qubit* de armazenar e processar informação em relação aos *bits* clássicos, nesse nosso algoritmo, cada *qubit* foi capaz de armazenar um valor hexadecimal, e processá-lo por si só, e apesar da diferença de um processador quântico não ser o ponto em discussão, vale ser observada essa capacidade superior.

Partindo para uma análise de complexidade, e pensando que para um vetor a ser buscado de tamanho *n* nós tenhamos igualmente *n qubits* para que nosso algoritmo não tenha que passar por adaptações (que no caso consistiriam em separar nosso vetor para reutilizar os *qubits*, mas que não é nosso intuito abordar nesse trabalho), notamos um grande custo na etapa de codificação dos dados, que por precisar tratar os dados, tem custo *n*, mas uma vez os dados codificados, a etapa de aplicar a rotação para encontrar o

nosso número e a aplicação da nossa porta hadamard, que podem ser integrados ao circuito previamente, portanto sendo executados em tempo constante.

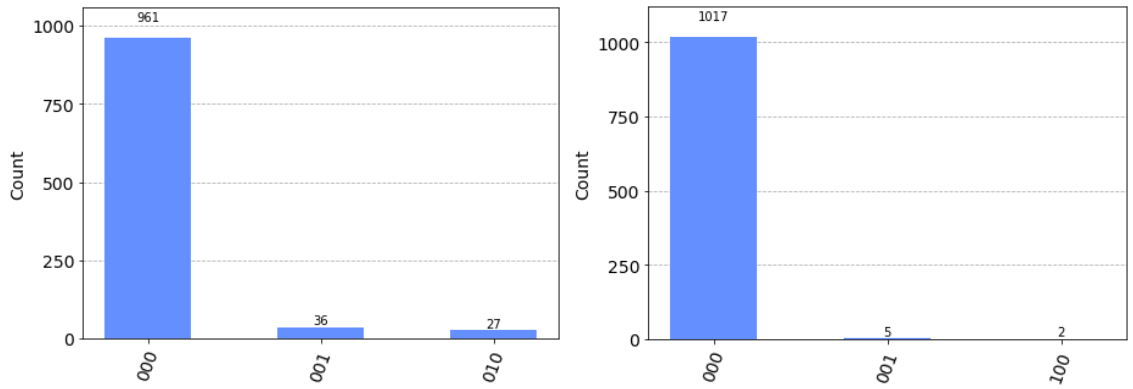

Figura 15. Resultados para testes com os respectivos vetores: [7,14,11] e [4, 5, 6] buscando os respectivos números: 11 e 5

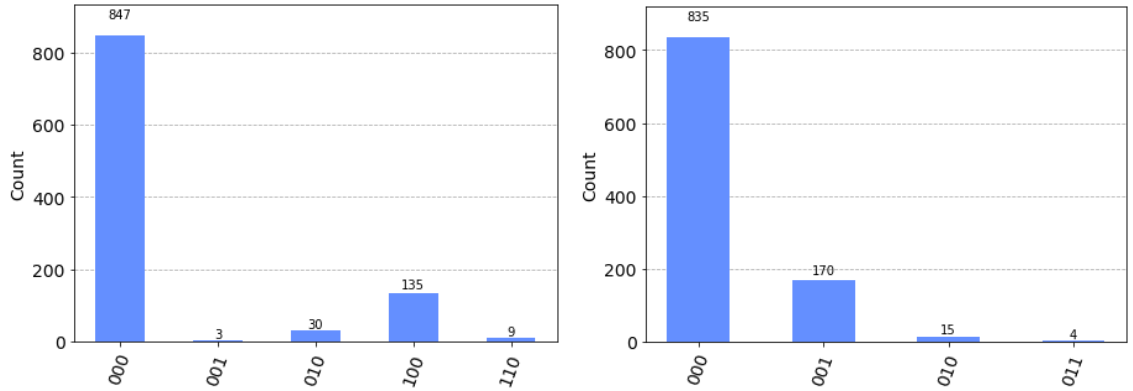

Figura 16. Resultados para testes com os respectivos vetores: [4, 8, 12] e [15, 10, 7] buscando os respectivos números: 5 e 7

5. Considerações finais

Analisamos e entendemos no início deste trabalho diversos algoritmos quânticos, alguns entre eles que possuem inclusive uma versão clássica, e conseguimos identificar a grande diferença dos paradigmas de ambas, onde apesar de realizarem as mesmas, ou muito parecidas, tarefas, seu funcionamento quase que por completo é alterado, não apenas por ter sido adaptado para sua versão quântica, mas pelo funcionamento dos *qubits* permitirem maneiras de processarmos os dados que computadores clássicos não conseguem de modo hábil.

O algoritmo escolhido para ser trabalhado e estudado foi o algoritmo de busca de Grover, tal como os algoritmos propostos, nota-se facilmente uma grande dificuldade em codificar os dados clássicos em *qubits*, onde dependendo da quantidade de dados clássicos trabalhados, estes serão responsáveis pela maior parte da complexidade do algoritmo.

Dentre os algoritmos propostos, foram abordadas duas estratégias quânticas, uma através das probabilidades, e uma através de rotações, a começar pelo das probabilidades, percebemos dificuldade em trabalhar com este tipo de abordagem, pelo fato de sistemas quânticos possuírem ruído por conta da forma como funcionam os sistemas quânticos e perderem precisão facilmente se não trabalhados com cuidado. Referente a abordagem através de rotações, conseguimos analisar e entender melhor o funcionamento dos *qubits* e a grande diferença que eles apresentam em relação a suas versões clássicas. O *notebook python* onde estes algoritmos foram executados e trabalhados está publicado no *GitHub**.

6. Agradecimentos

Os autores agradecem ao MackCloud (https://mackcloud.mackenzie.br), Centro Multidisciplinar de Computação Científica e Nuvem da Universidade Presbiteriana Mackenzie; e à FAPESP, Fundação de Amparo à Pesquisa do Estado de São Paulo, processo no. 2018/25225-9, pelo apoio na realização desta pesquisa.

7. Referências

Brylinski, Ranee K.; Chen, Goong (2002) "*Mathematics of Quantum Computation*", In: 1. ed. New York: Chapman and Hall/CRC. 448 p. ISBN 9780429122798.

Divicenzo, David P. (1998) "Quantum gates and circuits", In: Royal Society, [s. l.], ano 1998, v. 454, ed. 1969, 8 jan. 1998. DOI https://doi.org/10.1098/rspa.1998.0159. Disponível em: https://royalsocietypublishing.org/doi/10.1098/rspa.1998.0159.

Pasquale, R.; Bianchini, C. (2020) "*UM ESTUDO EXPLORATÓRIO DAS FALHAS DE SEGURANÇA DE ALGORITMOS DE CRIPTOGRAFIA NA ERA DA COMPUTAÇÃO QUÂNTICA*" In: Jornada de Iniciação Científica e Mostra de Iniciação Tecnológica - ISSN 2526-4699, Brasil, dez. 2020. Disponível em: http://eventoscopq.mackenzie.br/index.php/jornada/xvijornada/paper/view/2027/1377.

Rabelo, Wilson R.M. e Costa, Maria Lúcia M. (2018) "*Uma abordagem pedagógica no ensino da computação quântica com um processador quântico de 5-qbits*", In: Revista Brasileira de Ensino de Física [online]. 2018, v. 40, n. 4. Disponível em: https://doi.org/10.1590/1806-9126-RBEF-2018-0038.

Arute, Frank et al. (2019) "*Quantum supremacy using a programmable superconducting processor*", In: Nature, Nature, ano 2019, v. 574, p. 505-510, 23 out. 2019. DOI https://doi.org/10.1038/s41586-019-1666-5. Disponível em: https://rdcu.be/cBpMi.

Peter W. Shor. (1997) "*Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*", In: SIAM Journal on Computing, 26(5):1484-1509, 1997. DOI https://doi.org/10.48550/arXiv.quant-ph/9508027.

Lov K. Grover (1996) "*A fast quantum mechanical algorithm for database search*", In: https://arxiv.org/abs/quant-ph/9605043v3. DOI https://doi.org/10.48550/arXiv.quant-ph/9605043. Originally published in Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), May 1996, pages 212-219.

Josn Preskill (2012) "*Quantum computing and the entanglement frontier*", In: https://arxiv.org/abs/1203.5813v3. DOI https://doi.org/10.48550/arXiv.1203.5813.

*URL para o repositório do notebook com as implementações:
https://github.com/giangamberi/Estudo-de-algoritmos-quanticos-e-suas-implementacoes