
QUANTUM ORACLES - COMO TRANSFORMAR PROBLEMAS CLÁSSICOS EM QUÂNTICOS

✉ Alexandre Silva

Ciências da Computação

UNIVEM - Centro Universitário Eurípides de Marília

ABSTRACT

A partir do uso de quantum Oracles e outros fatores quânticos, como a superposição, foram feitos 5 *mini-projetos*. O objetivo desses projetos foi tentar responder se é possível transformar certos problemas em quânticos e se realmente tal transformação vale a pena. Após os testes foi possível ver que, há casos em que a versão quântica apresenta um aproveitamento igual ou um pouco superior em alguns casos, contudo ainda é necessário o uso de computadores clássicos para conseguir melhores resultados.

1 Introdução

Hoje, não é difícil ver alguém falando sobre computação quântica e como essas máquinas vão mudar o nosso futuro. Contudo, muitas dessas frases acabam se levando por extrapolações e/ou usos indevidos de ficção. Neste artigo, mostrarei que nem tudo é possível ser feito com um computador quântico atual, assim como existem pequenas áreas que se beneficiam ao máximo dessa nova tecnologia.

Para esse feito, serão mostrados 5 *mini-projetos* usando o qiskit, *framework open source* da IBM para computação quântica, e resultados obtidos após executar os algoritmos quânticos e seus relativos em computação clássica.

Tais mini-projetos foram os seguintes: Explorador de Arquivos 3.1, conversão de milhas para quilômetros 3.2, Torres de Hanoi 3.3, Buckshot Roulette 3.4 e QRAM 3.5. Todas as implementações podem ser encontradas nesse repositório do GitHub. Para a criação desses algoritmos, foram usados os Quantum Oracles explorando alguns efeitos quânticos para cada caso específico.

2 Oracles

Partindo da ideia das *Oracle Turing Machines* [1][7][8][17], os Oracles são modelos matemáticos ideais, usados para abstrair certas partes de um algoritmo principal, em formato de caixa preta, facilitando a análise do algoritmo, assim como sua descrição matemática. Tais máquinas podem ser vistas também como uma função, recebendo entradas x e retornando $f(x)$ em tempo $O(1)$. Em computação clássica, esse modelo não possui implementação real, sendo usado apenas descrições formais para problemas de decisão.

Contudo, em computação quântica, é possível implementar esses componentes e tomar proveito de sua estrutura e efeitos quânticos para conseguir um *Speed-up* em relação aos algoritmos clássicos, como mostrado pelo algoritmo de Deutsch–Jozsa [11]. Além disso, os Oracles possuem um papel importante ao demonstrar a complexidade de um circuito, alguns dos meios utilizados são: profundidade (*depth*), calculando o maior caminho que uma informação percorre no circuito, ou ainda a quantidade de gates aplicados. No entanto, essas maneiras acabam se prejudicando ao *transpile* o circuito para uma outra máquina, variando então a complexidade de acordo com a topologia e com os gates fisicamente implementados. Para solucionar isso, outra maneira de calcular é inserir partes do circuito em um Oracle, e descrever sua complexidade a partir da quantidade de vezes que ele chamado, também conhecido como *query complexity* [2] [17].

2.1 Tipos de Oracles

A partir da definição dos Quantum Oracles, podemos classificá-los em relação a suas estruturas e maneiras de computar os dados.

2.1.1 Phase Oracle

O Phase Oracle, é o formato mais conhecido e usado em circuitos quânticos. Algoritmos como os de Deutsch–Jozsa, Grover, Simon e Bernstein–Vazirani, tomam proveito desse artifício para se sobressair em relação às soluções clássicas.

2.1.1.1 Funcionamento Padrão

Seu funcionamento, se baseia em atribuir uma fase global ao circuito, tomando proveito de fatores como *Phase Kickback* (fase passa do target do CNOT e é aplicado no qubit de controle), para conseguir modificar valores em superposição.

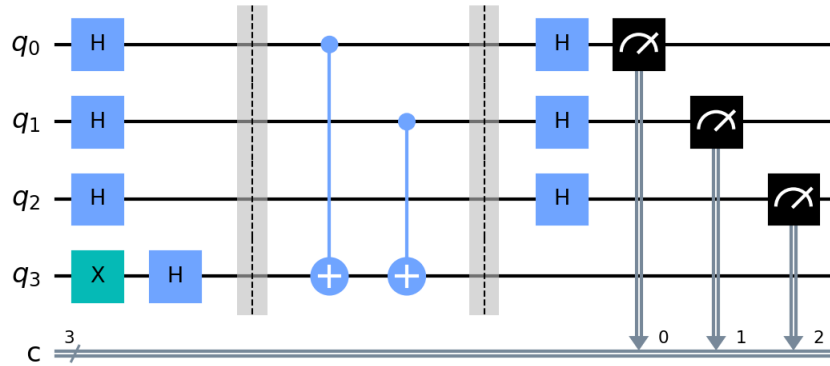


Figura 1: Exemplo - Phase Oracle

Na Imagem 1, foi introduzida uma fase π no qubit auxiliar (q_3) através do estado $|-\rangle$. Essa fase será responsável por modificar os valores na matriz unitária do circuito. Nessa configuração, os CNOTs agem de uma forma um tanto diferente do convencional, aqui, ao invés de apenas inverter o valor do qubit no target quando o qubit no control for 1, devido a fase, ele também agirá como um gate Z no control. Sendo assim, ao aplicar $CNOT |-\rangle |+\rangle$ (qubit menos significativo à direita), o estado se torna $\frac{1}{\sqrt{2}}(|0\rangle |-\rangle - |1\rangle |-\rangle)$, e ao remover a superposição com o H , a saída se torna: $\frac{1}{\sqrt{2}}(|+\rangle |1\rangle - |-\rangle |1\rangle)$. Dessa forma, o qubit que antes estava como controle do gate, sofre a ação do *Phase Kickback*, e seu estado padrão $|0\rangle$ é modificado pela fase e se torna $|1\rangle$. A partir disso, é possível encodar um certo valor binário dentro do Oracle e utilizá-lo para cálculos.

2.1.1.2 Versão Minimal Oracle

Além disso, esse não é o único formato possível de Phase Oracle. Por apenas aplicar uma fase em certas bit-strings, o qubit auxiliar pode ser removido e a fase pode ser adicionada através de gates Z controlados (ou outro gate capaz de aplicar uma fase π para certa bit-string), mas ainda assim mantendo a natureza unitária, podendo ser visto também como um Minimal Oracle.



Figura 2: Exemplo Phase Oracle como um Minimal Oracle

No exemplo da imagem 2, foi adicionado um gate MCP com a fase global π e dois gates X para inverter quais qubits queremos que tenham o valor 0, codificando assim o valor 011_2 ou 3_{10} .

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

Figura 3: Matriz unitária do Phase Oracle da imagem 2

É possível verificar então que ao criar esse circuito, é mantida a matriz identidade mas com a fase (-1) no valor 1 na coluna relativa à 011_2 .

2.2 Boolean Oracle

O Boolean Oracle, por sua vez, apresenta um funcionamento semelhante ao do Phase Oracle. Contudo, dessa vez não é provida uma fase. Dessa forma, o Oracle age como uma função Booleana convencional, mapeando as entradas para valores de saída.

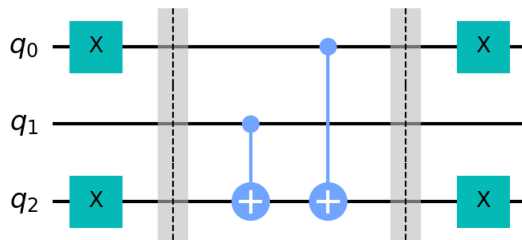


Figura 4: Exemplo de Oracle Booleano

O Oracle implementado na figura 4, pode ser reutilizado para o algoritmo de Deutsch-Jozsa, basta apenas introduzir uma fase, e o Boolean Oracle se comportará como um Phase Oracle.

2.3 Minimal oracle

Como já citado anteriormente, o minimal oracle possui uma função que em sua essência é unitária, não requerendo qubits adicionais $|x\rangle \rightarrow |f(x)\rangle$.

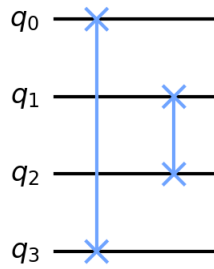


Figura 5: Exemplo de minimal oracle

Lembrando que este pode também adicionar fases ao circuito.

2.4 Simon's Oracle

O Oracle de Simon, é uma instância do Boolean Oracle. Neste visamos encontrar os períodos da função implementada, ou seja, dado x e $f(x) = y$ existe ao menos uma função em que $f(y) = x$?

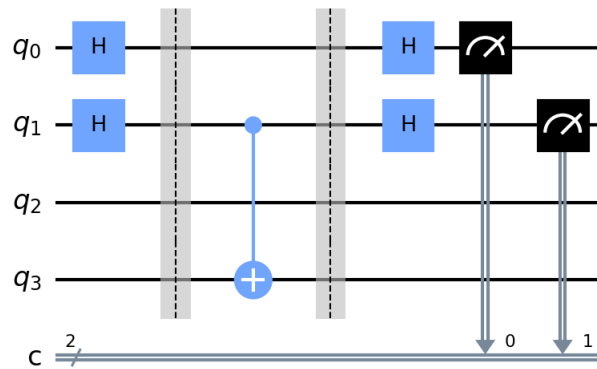


Figura 6: Exemplo do algoritmo de Simon

Nesse algoritmo, configuramos uma chave s dentro do oracle, e ao executar o algoritmo temos os possíveis períodos da função, sendo necessário rotinas de pós processamento para identificar o valor correto.

2.5 QFT(Quantum Fourier Transformation) Oracle

Por fim, o Oracle QFT aplica a versão quântica da transformada de Fourier, projetando os valores de entrada na base X (também conhecido como base de Fourier).

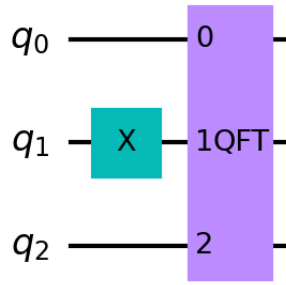


Figura 7: Exemplo do algoritmo de QFT

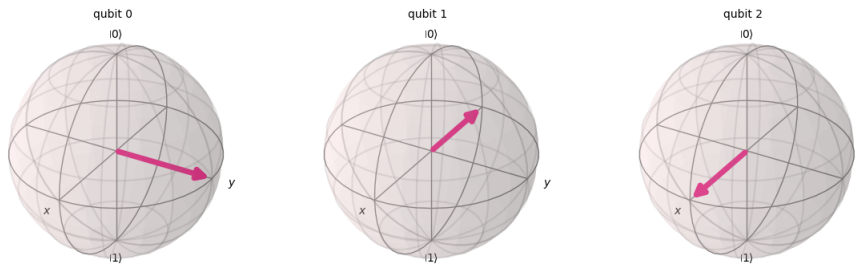


Figura 8: Valores mapeados na base de Fourier

3 Desenvolvimento

3.1 File Explorer

Imagine um computador quântico com um sistema operacional quântico (semelhante aos computadores convencionais, mas dessa vez seguindo as leis da mecânica quântica). Pensando nas partes desse sistema operacional, como seria possível pegar arquivos da memória usando a computação quântica?

3.1.1 Algoritmos usados

3.1.2 Grover

Um dos algoritmos mais comuns para a área é o algoritmo de Grover. Esse algoritmo realiza buscas em "bancos de dados" (bit strings) desorganizados em tempo $O(\sqrt{2^n})$ onde n é o número de qubits usados. Nele, usamos um circuito do qual amplifica-se a probabilidade de encontrar os valores marcados no Oracle na saída. Tal algoritmo segue o seguinte padrão:

1. Configuramos todas as possíveis bit strings, ou seja aplicamos uma superposição uniforme $H^{\otimes n}$
2. aplicamos o phase oracle U_f do qual implementa uma função que marca os valores que desejamos encontrar
3. aplicamos o operador de Grover Diffuser, $\mathbb{I} - 2|s\rangle\langle s|$, sendo s o estado com o valor que queremos

Convertendo para um circuito temos algo semelhante a:

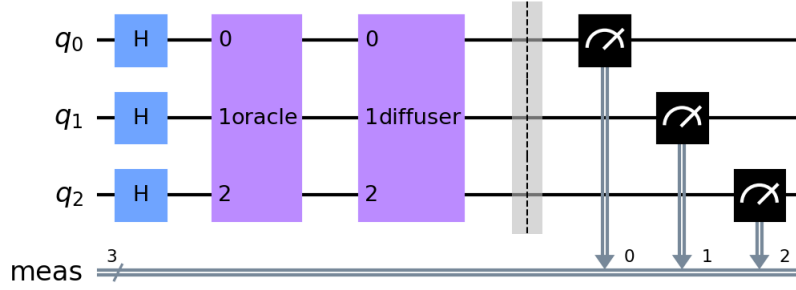


Figura 9: Exemplo algoritmo de Grover

Para 1 ou 2 valores, podemos usar a configuração acima. Mas maiores quantidade de valores, precisamos adicionar o conjunto Oracle + Diffuser k vezes, sendo $k \approx \frac{\pi}{4\sqrt{\frac{a}{2^N}}} - \frac{1}{2}$, sendo a o numero de valores marcados.

3.1.3 Diferença de conjuntos

Utilizando os phase oracles, podemos criar dois Oracles distintos com ranges de valores diferentes e sobrepor seus valores, realizando a operação de diferença de conjuntos.

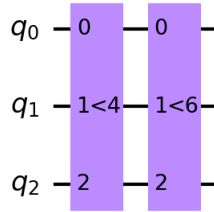


Figura 10: Exemplo de diferença de conjuntos

Nesse exemplo foi encodado no primeiro oracle o set $\{000, 001, 010, 0110\}$ e no segundo $\{000, 001, 010, 011, 100, 101\}$.

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
0	0	0	0	0	0	0	1

Figura 11: Resultado da diferença de conjuntos

Como pode ser visto, apenas os valores $\{100, 101\}$ permaneceram com a fase, representando então a sobreposição delas.

3.1.4 Solução para o problema

Para a solução do problema, podemos criar em uma hash function $C(v)$ da qual recebe o path de um arquivo e retorna uma bit string respectiva. Com essa função em mãos, podemos utilizar o conjunto dos valores retornados e encoda-los em um phase oracle, agindo como uma especie de Look Up Table para os arquivos existentes na máquina.

No entanto, para ter sucesso na pesquisa, é necessário utilizar um segundo Oracle encondando $S = P - s$, sendo s o conjunto de arquivos do qual estamos procurando e S os arquivos restantes, aproveitando-se então da diferença de conjuntos para encontrar então apenas os valores desejados para a pesquisa.

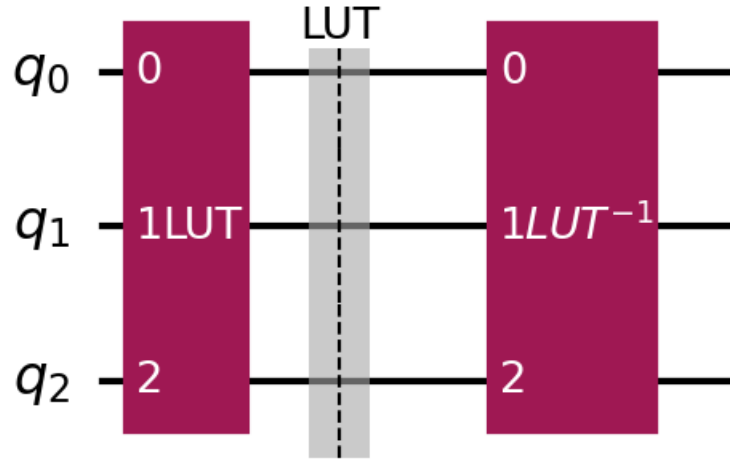


Figura 12: Diferença de conjuntos com as Look Up Tables

Sendo assim, o primeiro oracle age como o HD da marquina, marcando todos os arquivos existentes, e o segundo age como o mediador da pesquisa.

Com isso, adicionamos a Look Up Table final ao algoritmo de Grover.

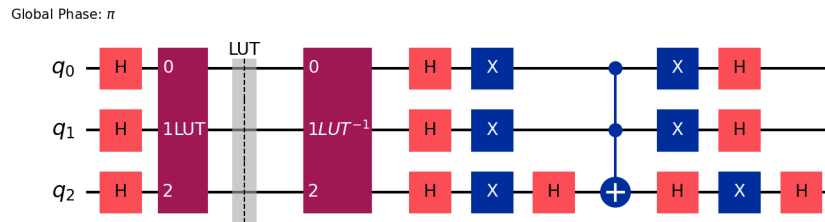


Figura 13: File explorer circuito

3.1.5 Resultados

Para um caso hipotetico de sistema completamente quântico, certamente esse é um das maneiras de encontrar arquivos em meio a todos os outros.

Contudo, como o objetivo dessa pesquisa sugere, para utilizar esse modelo em um sistema clássico tomando proveito da computação quântica, não se mostra como a melhor opção.

Para sistemas convencionais, dos quais utilizam métodos baseados em árvores, não é possível tirar qualquer proveito aqui, sendo O polinomial contra O quadrático.

Mesmo perante os testes clássicos feitos durante o desenvolvimento, esse protocolo só se mantém útil quando a pesquisa era feita de forma linear.

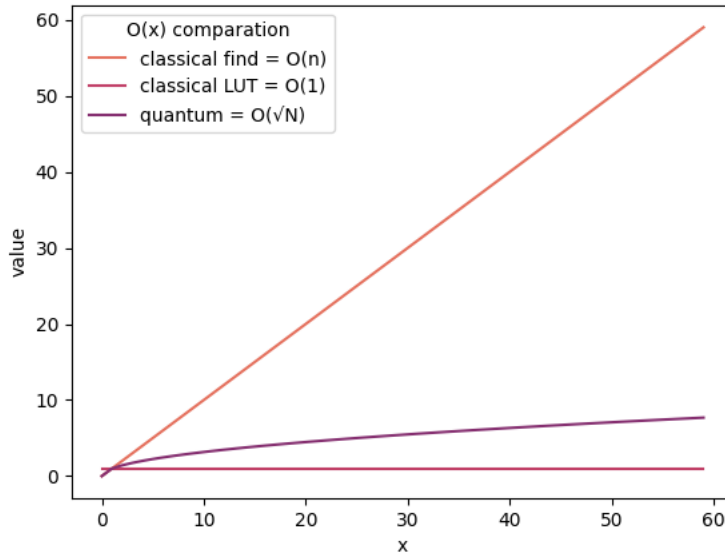


Figura 14: Comparação algoritmos usados na pesquisa

Sendo assim, o algoritmo de Grover deve ser pensando para casos do qual a versão clássica possui complexidade $\geq O(n)$.

3.2 Milhas para quilômetros

O segundo problema testado foi a conversão de milhas para Km. Essa idéia se deu a após a descoberta de um algoritmo capaz de calcular fibonacci com circuitos quânticos.

3.2.1 algoritmo de Fibonacci

A versão quântica usada para calcular fibonacci apresentada em [15], demonstra que utilizando um circuito que coloca em superposição todas as bit strings com n qubits que não possuem valores uns consecutivos, é possível encontrar o valor n de fibonacci.

Sua implementação se dá da seguinte maneira:

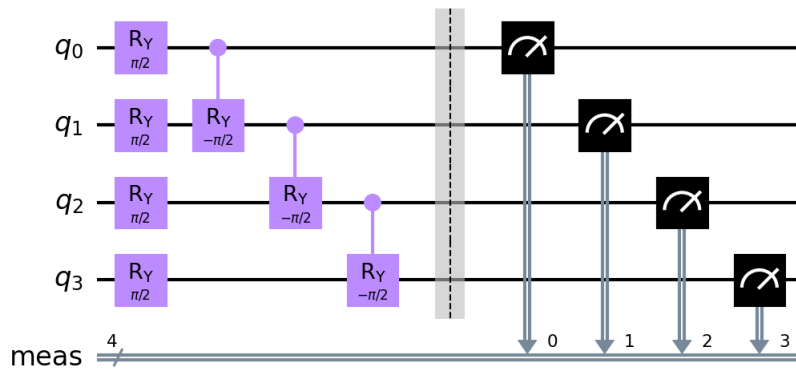


Figura 15: Exemplo Algoritmo de Fibonacci

Tendo como resultado:

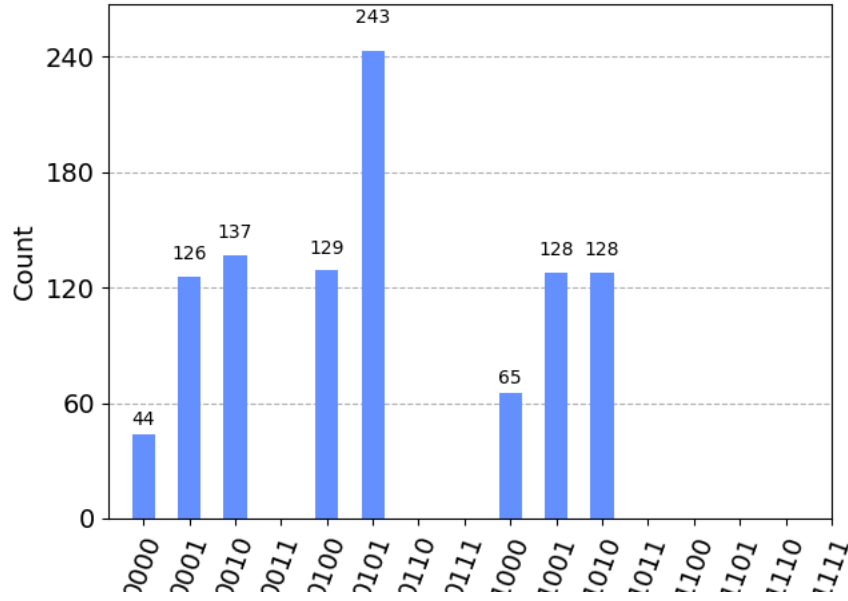


Figura 16: Resultado exemplo Fibonacci - F(4)

Essa implementação inicia a cadeia partir do valor 2, sendo assim: 2, 3, 5, 8, 13, 21, Repare que na figura 16, a quantidade de número de valores encontrados (nesse caso 8) é o mesmo do que o 4º valor da sequência. Ou seja, para esse circuito, o valor de qubits n representa o valor n de fibonacci que queremos encontrar. Sendo assim, podemos pensar em um oracle implementado $F(n)$ do qual a entrada n é igual a $|0\rangle^{\otimes n}$.

3.2.2 Implementação

Para aproximar o valor de milhas para quilômetros, podemos utilizar a relação de fibonacci da seguinte forma:

$F_{km} = F_{milhas}(n + 1)$, sendo assim, para conseguir o valor relativo em quilômetros, basta pegar o próximo valor na sequência.

milhas	km
1	2
2	3
3	5
5	8

Tabela 1: valores aproximados de milhas para Km

No entanto, dessa forma, temos um problema, a quantidade necessária de qubits. Para entradas n tendendo ao infinito, precisaremos de infinitos qubits. Para resolver isso, foi criado um algoritmo clássico simples para quebrar numero desejado em partes menores que podem ser calculadas em um circuito com menos qubits, $f : (n) \rightarrow ((n_1, i), (n_2, i), \dots)$, mapeando então a entrada n para uma lista de tuplas contendo o número base e a quantidade de aplicações necessária i .

Com isso, o circuito final é apresentado da seguinte forma:

Algorithm 1 Algoritmo quântico para a conversão

```
for cada número base  $n$  do  
  Aplique o oracle  $F(n)$   
  Faça as medições nos qubits  
  reset os qubits usados  
end for  
verifique o resultado de cada bit string  
multiplique cada resultado com o valor  $i$  correspondente
```

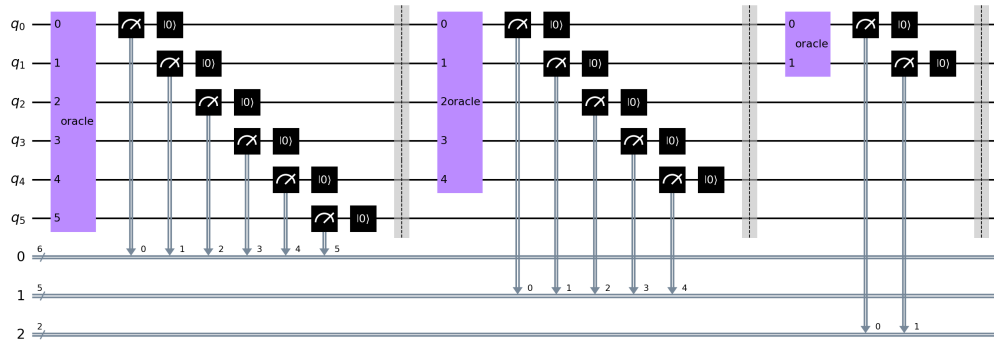


Figura 17: Circuito de conversão implementado

3.2.3 Resultados

Usando esse método é possível alcançar os valores esperados. Contudo existem alguns pontos que torna esse método pior do que a versão clássica:

1. Quantidade necessária de medições e tempo de execução
Para cada vez que medirmos o circuito, precisamos de uma quantidade alta de shots para alcançar um resultado melhor, aumentando também o tempo necessário para executar desse versão.

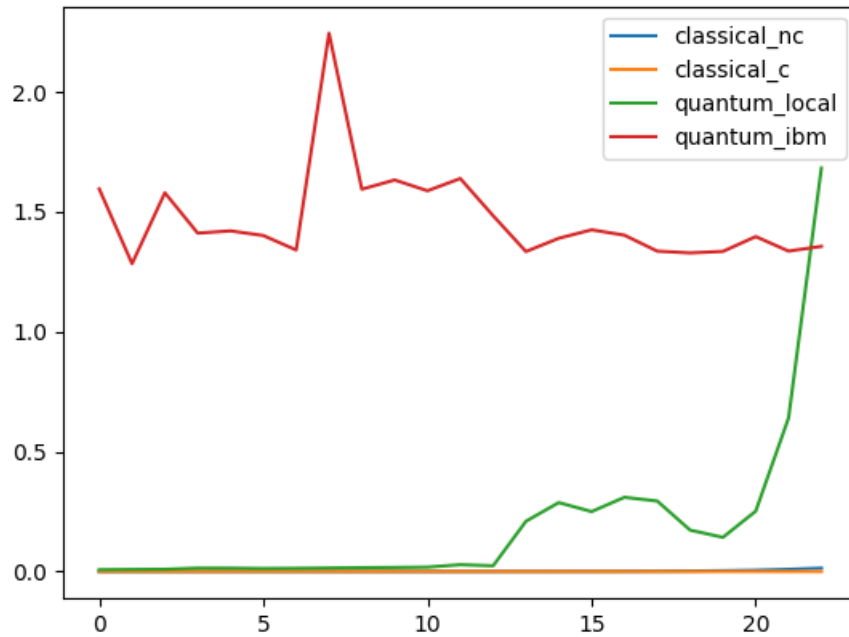


Figura 18: Comparação tempos de execução

Como é possível ver em 18, o tempo das versões clássicas com e sem memoization possuem tempo praticamente constante de execução em comparação com as versões quânticas em simuladores e hardwares reais.

2. Erros

Como a maioria dos algoritmos quânticos da era NISQ(noisy intermediate-scale quantum), os erros também estão presentes, e por utilizarmos muitos gates de dois qubits, esses erros podem se intensificar dependendo do hardware usado.

3. Imprecisão conforme a entrada cresce ao infinito

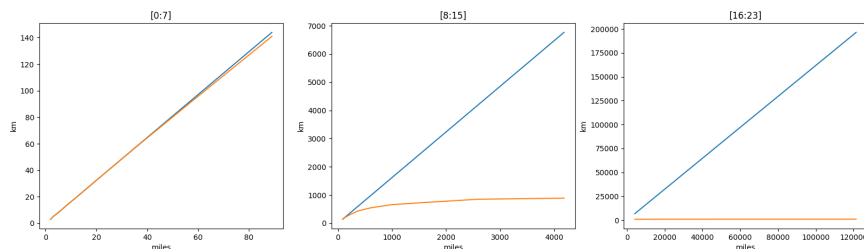


Figura 19: Comparação resultados versão clássica e quântica

Como é possível ver em 19, valores pequenos possuem uma boa precisão com os números esperados(em azul), mas a partir de certo ponto eles começam a se distanciar.

4. necessidade de intervenção clássica

Por fim, esse algoritmo requer que seja primeiro verificado quais são os valores de fibonacci necessários para cada parte, além de ser necessário pós processamento após as medições.

Para amenizar tal problema, é necessário criar outros algoritmos capazes de encodar o valor requerido diretamente no circuito sem maiores intervenções clássicas, necessitando apenas de pós processamento (em certos casos).

3.3 Torres de Hanoi Hanoi

Para o terceiro teste, foi implementado uma versão das torres de Hanoi usando os Oracles como meio de encoding.

3.3.1 Implementação

Para implementar as torres de hanoi em um oracle, são necessários $(\lfloor \log_2 x \rfloor + 1) * 3$ qubits, sendo x o número de discos com a seguinte ordem $|t_{n-1}t_{n-2}\dots t_0\rangle |a_{n-1}a_{n-2}\dots a_0\rangle |s_{n-1}s_{n-2}\dots s_0\rangle$, sendo t a ultima torre, a a torre do meio e s a primeira torre, e $n = \frac{nqubits}{3}$.

Com essa configuração, os numeros de 1 à x são codificados em um phase oracle, e em seguida são realizadas operações de *swap* para modificar a posição dos valores binários e mover os valores do n qubits menos significativos, para os n qubits mais significativos.

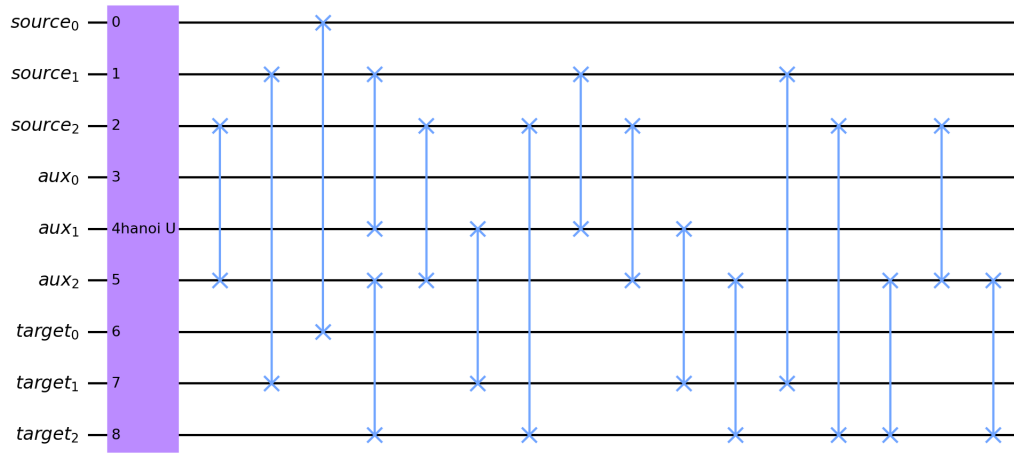


Figura 20: Torre de hanoi com 4 discos

Nesse formato, é possível utilizar o unitário gerado pelo oracle, assim como o algoritmo de Grover para verificar o resultado.

3.3.2 Resultados

Nessa implementação, o algoritmo segue o mesmo padrão da versão clássica, sendo o melhor ou pior desempenho dependendo estritamente do hardware usado.

3.4 Buckshot Roulette

Buckshot Roulette é um jogo feito pelo desenvolvedor Mike Klubnika para computador que toma como base a premissa de reinventar a infame roleta russa. No jogo, você é desafiado por um demônio (dealer), e caso você ganhe você ganha uma recompensa, mas caso contrário o jogo reinicia.

Para esse projeto, tomamos como base a primeira rodada do jogo buscando encontrar a melhor estratégia para maximizar os ganhos.

Nessa primeira rodada temos 3 vidas, 2 balas falsas e 1 bala verdadeira.

A dinamica funciona da seguinte forma: você começa jogando,tendo duas possíveis escolhas, atirar em você mesmo ou no dealer. Caso você atire em você mesmo e a bala for real, você perde uma vida, mas você joga novamente na próxima rodada, caso a bala seja falsa você joga novamente. Agora, caso a escolha seja atirar no dealer, se a bala for real você ganha, se não o dealer joga.

3.4.1 Versão clássica

Antes de crair o circuito, foi feita a modelagem do jogo usando a estrutura em árvore.

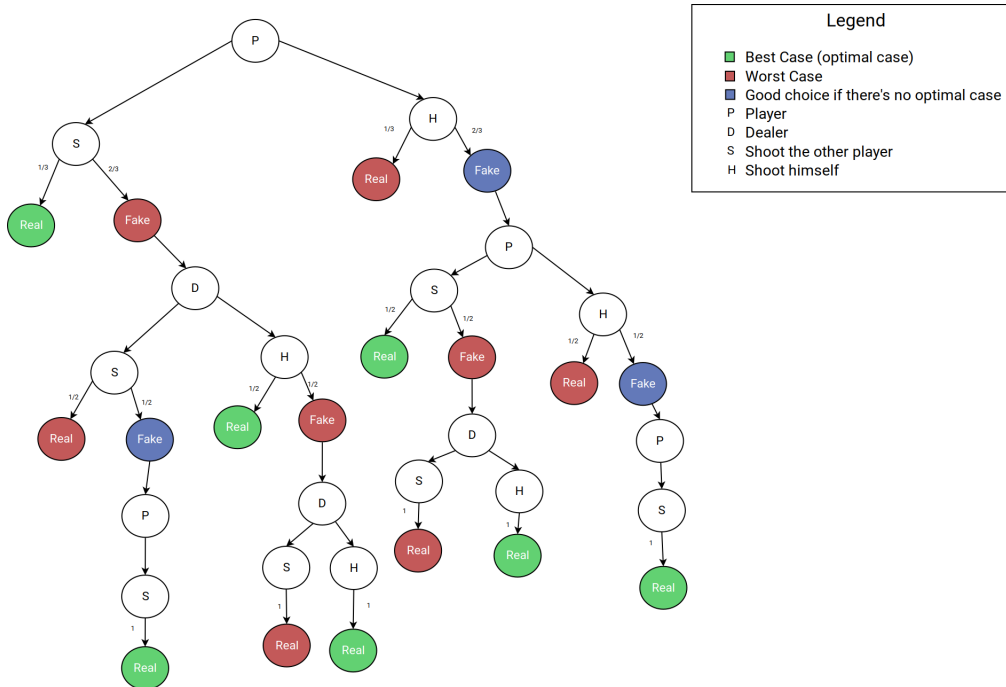


Figura 21: Buckshot Roulette diagrama

Note que nessa estrutura, o dealer na ultima bala, ainda pode escolher atirar ou não nele mesmo, isso acontece pois o jogo implementa ações randomicas para o dealer.

Seguindo essa estrutura, podemos tentar encontrar o melhor caminho dentre a árvore e simular os resultados da partida.

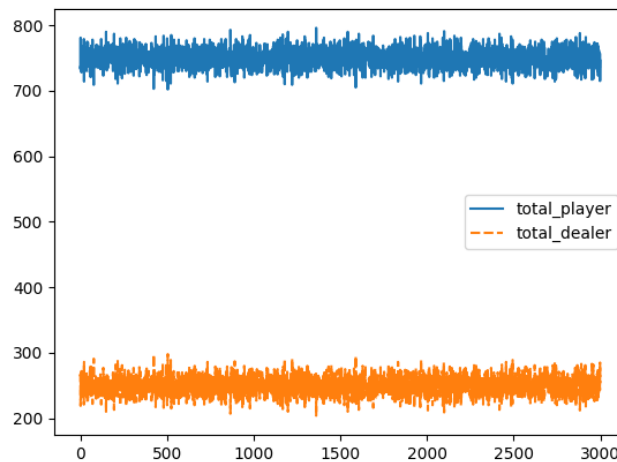


Figura 22: Buckshot Roulette clássico - melhor estratégia

Após testar os possíveis caminhos, o melhor resultado obtido foi esse apresentado acima em 22. Com um pouco de investigação, foi possível entender que a melhor estratégia encontrada foi o player começar atirando no dealer. Isso pois ao seguir tal caminho, ele tem uma chance a menos de perder a rodada, já que a chance de perder logo no começo é eliminada.

rodada	ação	resultado da ação	resultado da partida
1	player atira no dealer	real	player ganha
1	player atira no dealer	fake	-
2	dealer atira no player	real	dealer ganha
2	dealer atira no player	fake	-
2	dealer atira nele mesmo	real	player ganha
2	dealer atira nele mesmo	fake	-
3	player atira no dealer	real	player ganha
3	dealer atira no player	real	dealer ganha
3	dealer atira nele mesmo	real	player ganha

Tabela 2: melhor estratégia - possíveis resultados

3.4.2 Versão quântica

A partir dessa ideia, um circuito quântico foi modelado imitando o primeiro round, e um oracle foi usado para cada jogador implementando sua estratégia em seu interior.

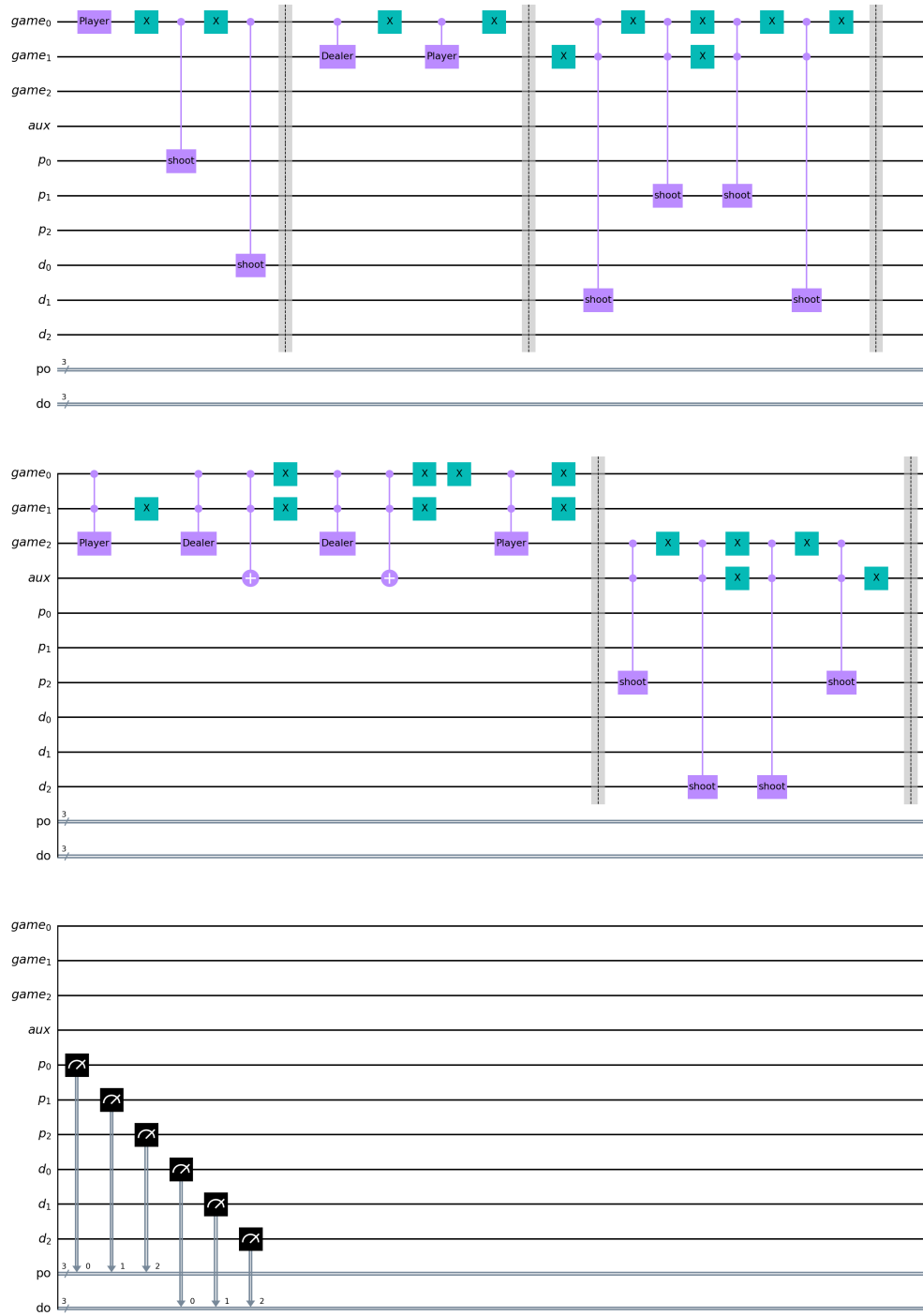


Figura 23: Circuito para o buckshot roulette

Além disso, para encontrarmos a melhor estratégia para o player, foi inserido dois parâmetros dentro do oracle, sendo possível inserir qualquer valor θ e ϕ para encontrar a melhor rotação na bloch sphere.

Após verificar uma grande variedade de valores possíveis, a rotação que entregou o melhor resultado foi $\theta = 3.0853981633974477$, $\phi = 5.685398163397447$. Usando essa estratégia, os resultados foram semelhantes a versão clássica usando o simulador Aer:

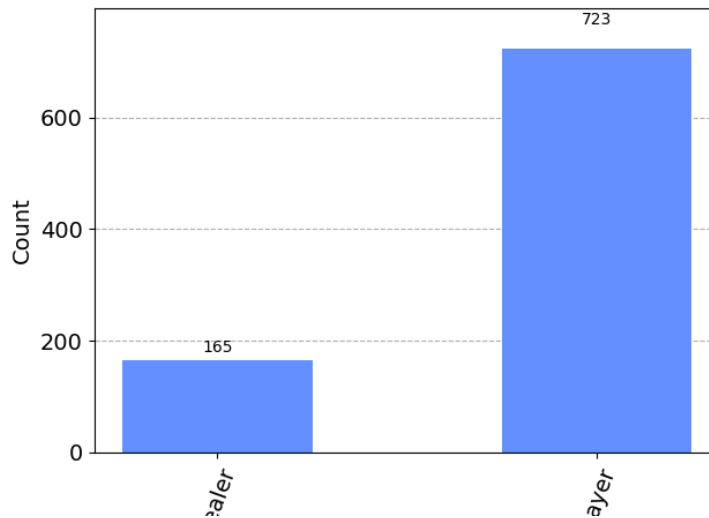


Figura 24: Resultado Buckshot Roulette quântico

Observando a bloch sphere do estado gerado por essa rotação, é possível ver também que a estratégia de fato se assemelha a versão classica, com o player preferindo atirar no dealer a maior parte do tempo (o valor 1 aqui representa atirar no outro jogador e 0 atirar em si mesmo).

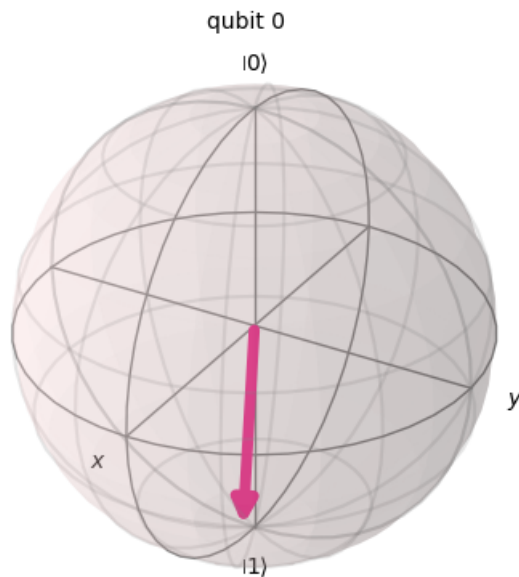


Figura 25: Melhor estratégia Buckshot Roulette quântico - Bloch Sphere

3.4.3 Conclusões

Para esse problema, não há uma competição certa entre as duas versões, uma vez que uma é diretamente inspirada na outra.

Além disso, a versão quântica possui ainda a possibilidade de explorar mais valores do que a versão clássica, deixando o player mais aberto a escolha de estratégias.

Em relação a erros providos pelo hardware não afeta diretamente os resultados, uma vez que mesmo com os erros a proporção se mantém.

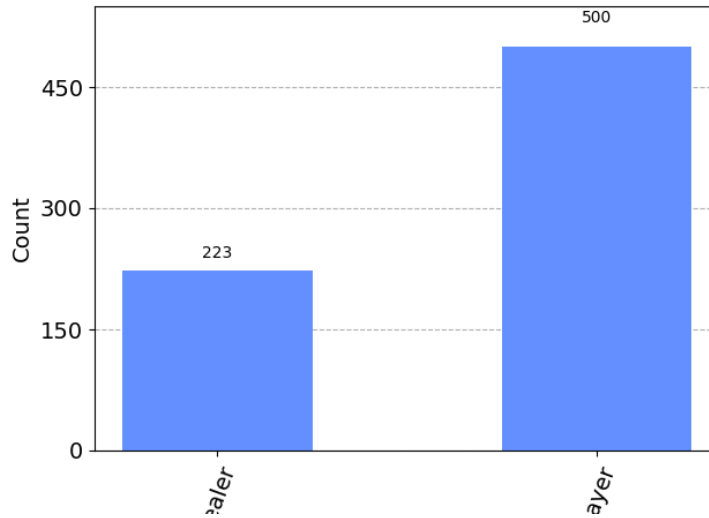


Figura 26: Resultados Buckshot Roulette usando o fake backend Melbourne da IBM

Note também que o total de partidas ganhas não chega ao total jogado, 1000 partidas no total, isso pois, pelo design do circuito, não é possível verificar a jogada do player anterior, sendo possível continuar jogando mesmo que um dos players já tenham perdido, o que foi necessário correções usando pós processamento após a simulação.

Em suma, ambas as simulações atingiram o mesmo resultado e foi demonstrado que é possível usar aqui o quantum oracle como uma representação de um player dentro do circuito.

3.5 QRAM

Por fim, o último projeto realizado foi o de uma QRAM utilizando os oracles para encodar os valores desejados. Nessa versão, foi testado maneiras de criar QROMs (com dados estaticos dentro), e uma possível maneira de utilizar uma QRAM habil para escrita.

Neste, foi levado em consideração o armazenamento de estados quânticos, e não de bitstrings clássicas.

3.5.1 QROM implementação

Primeiro, foi feita uma versão de QROM, do qual utiliza n qubits para endereços e m qubits para a criação dos estados. Não necessariamente os valores precisam estar correlacionados, podemos ter $n = 3, m = 10$. Isso pois, nesse formato, podemos mapear diversas superposições diferentes e aplicálas quando certo endereço for chamado. Sendo assim, o algoritmo aqui mostrado armazena os valores a partir da configuração de gates controlados interiores ao oracle, criando uma superposição apenas quando certo valor de entrada é inserido.

A entrada do circuito segue o seguinte formato: $|0\rangle^{\otimes m} |a_{n-1}a_{n-2}\dots a_0\rangle$

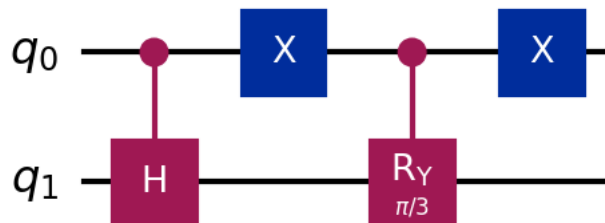


Figura 27: Exemplo circuito - QROM

A partir desse circuito, podemos abstrair para um oracle e utilizar em um circuito maior, chamando-o novamente sempre que for necessário um certo estado ou ainda colocar os endereços em superposição e ter uma mistura de superposições na saída.

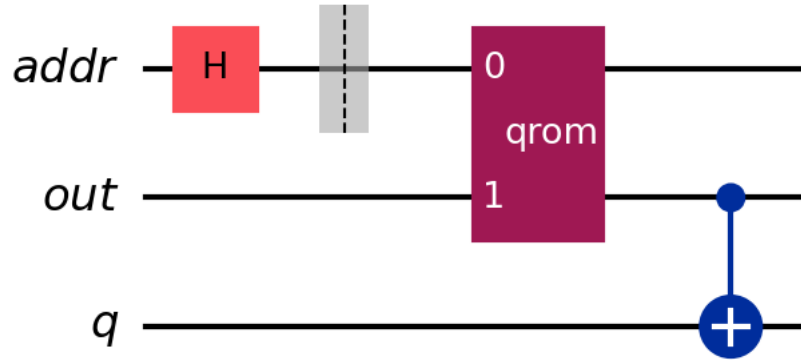


Figura 28: Exemplo circuito usando a QROM

Nessa configuração, os estados em superposição serão colocados no qubit denominado *out*, sendo possível se aproveitar dela em outros qubits, como nesse caso o qubit *q*.

Contudo, devido ao no-cloning theorem, não é possível copiar esse estado para outro qubit, então só é possível aqui se aproveitar dos qubits de saída, ou utilizada do teleporte de estados para destruir o estado interno do oracle e mover para outros qubits desejados.

3.5.2 QRAM read and write - implementação

Para criar uma QRAM com a possibilidade de escrever também, podemos explorar o teleporte quântico já citado antes. Com isso, podemos ter n qubits, sendo cada qubit um endereço único, e utilizar do teleporte para mover um estado que estava no circuito, para o domínio da QRAM.

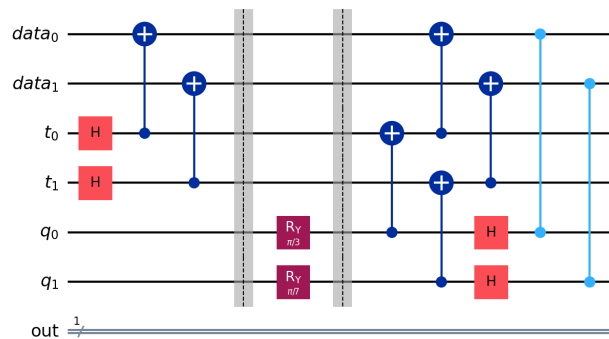


Figura 29: Exemplo circuito - QRAM

Aqui é necessário n qubits para n endereços (cada endereço aqui é um qubit de *data*) e n qubits para o teleporte (qubits nomeados como *t*). Mesmo crescendo linearmente, podemos aqui utilizar sobre-escrever valores nas posições desejadas, assim como interferir com outras superposições apenas teleportando novos valores para o qubit i .

3.5.3 Conclusões

Com esse projeto e com a literatura usada [24][23], é possível entender que versões quânticas de memória são difíceis de ser implementadas e ainda não foi possível tomar proveito do todo o seu potencial usando as superposições e estados de outras bases a não ser a base computacional $(0, 1)$.

Após as implementações, foi possível entender que, criar tais memórias pode parecer uma tarefa simples, mas devido a fatores como, complexidade de mapear dados, complexidade de utilizar a memória (já que é necessário reaplicá-la toda vez que for requisitado seu uso), no clonagem, etc. Foi possível entender que tal modelo não é ainda viável, sendo necessário mais pesquisas focadas no tema, assim como possíveis implementações em um Hardware futuro.

Em suma, utilizar tais circuitos agora não é das tarefas mais fáceis, assim como não necessariamente trará resultados no momento.

3.6 Conclusão

Com isso, foi mostrado que a computação quântica ainda tem muito potencial, contudo é possível ver que certos fatores prejudicam o seu uso no momento, assim como a falta de certos recursos.

No momento, é claro que computadores quânticos são máquinas extremamente úteis para diversos casos, e de fato existem áreas que tomaram extremo proveito com elas, como criptografia, machine learning, química, física, matemática, etc. Mostrando que a computação clássica não está condenada, mas sim uma nova era está surgindo com a junção da computação clássica e computação quântica.

Em resumo, é possível tirar proveito da computação quântica para problemas que conhecemos classicamente. No entanto, é necessário averiguar se há algum fator quântico que pode explorar o problema de uma forma que classicamente seria impossível ou não usuais, como no caso dessa pesquisa os Oracles.

Referências

- [1] Robert I. Soare. Turing oracle machines, online computing, and three displacements in computability theory. *Annals of Pure and Applied Logic*, 160(3):368–399, 2009. Computation and Logic in the Real World: CiE 2007.
- [2] Ryan O'Donnell. Lecture 5: Quantum query complexity, 09 2015.
- [3] Dave Bacon. Cse 599d -quantum computing simon's algorithm, 2006.
- [4] Robin Kothari. An optimal quantum algorithm for the oracle identification problem. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.
- [5] Ryan O'Donnell. Lecture 13: Lower bounds using the adversary method, 10 2015.
- [6] Laurel Brodkorb and Rachel Epstein. The entscheidungsproblem and alan turing, 12 2019.
- [7] Sadika Amreen and Reazul Hoque. Oracle turing machines.
- [8] Subrahmanyam Kalyanasundaram. mod04lec23 - oracle turing machines, 09 2021.
- [9] Martin Davis. Turing reducibility?, 11 2006.
- [10] Mahesh Viswanathan. Reductions 1.1 introduction reductions, 2013.
- [11] Yale Fan. A generalization of the deutsch-jozsa algorithm to multi-valued quantum logic. In *37th International Symposium on Multiple-Valued Logic (ISMVL'07)*. IEEE, May 2007.
- [12] Takashi Yamakawa and Mark Zhandry. Classical vs quantum random oracles. Cryptology ePrint Archive, Paper 2020/1270, 2020. <https://eprint.iacr.org/2020/1270>.
- [13] Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. classical communication and computation, 1998.
- [14] Javier Sanchez-Rivero, Daniel Talaván, Jose Garcia-Alonso, Antonio Ruiz-Cortés, and Juan Manuel Murillo. Some initial guidelines for building reusable quantum oracles, 2023.
- [15] Austin Gilliam, Marco Pistoia, and Constantin Gonciulea. Canonical construction of quantum oracles, 2020.
- [16] Elham Kashefi, Adrian Kent, Vlatko Vedral, and Konrad Banaszek. Comparison of quantum oracles. *Physical Review A*, 65(5), May 2002.
- [17] Niklas Johansson and Jan-Åke Larsson. Quantum simulation logic, oracles, and the quantum advantage. *Entropy*, 21(8), 2019.
- [18] William Zeng and Jamie Vicary. Abstract structure of unitary oracles for quantum algorithms. *Electronic Proceedings in Theoretical Computer Science*, 172:270–284, December 2014.
- [19] Alp Atici. Comparative computational strength of quantum oracles, 2004.
- [20] Kathiresan Sundarappan. How to build oracles for quantum algorithms, 04 2022.

-
- [21] Zhifei Dai, Robin Choudhury, Jinming Gao, Andrei Iagaru, Alexander V Kabanov, Twan Lammers, and Richard J. Price. View of the role of quantum algorithms in the solution of important problems.
 - [22] Don Ross. Game Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2024 edition, 2024.
 - [23] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16), April 2008.
 - [24] Samuel Jaques and Arthur G. Rattew. Qram: A survey and critique, 2023.
 - [25] Tomasz Zawadzki and Piotr Kotara. A python tool for symbolic analysis of quantum games in ewl protocol with ibm q integration. <https://github.com/tomekzaw/ewl>.
 - [26] Piotr Frackiewicz. Application of the ewl protocol to decision problems with imperfect recall, 2011.
 - [27] Jens Eisert, Martin Wilkens, and Maciej Lewenstein. Quantum games and quantum strategies. *Physical Review Letters*, 83(15):3077–3080, October 1999.
 - [28] Muhammad Usman. Kilometres to miles conversion — approximation of fibonacci series, 09 2019.
 - [29] Lidia André. Tower of hanoi – lidia andré, 03 2021.
 - [30] diptokarmakar47. How to solve the tower of hanoi problem - an illustrated algorithm guide, 01 2019.
 - [31] Towers of hanoi: A complete recursive visualization, 05 2020.
 - [32] GeeksforGeeks. Program for tower of hanoi, 05 2014.
 - [33] Faisal Shah Khan and Ning Bao. Quantum prisoner’s dilemma and high frequency trading on the quantum cloud. *Frontiers in Artificial Intelligence*, 4, 11 2021.
 - [34] Alexis R. Legón and Ernesto Medina. Dilemma breaking in quantum games by joint probabilities approach. *Scientific Reports*, 12, 08 2022.
 - [35] Brian Siegelwax. Quantum memory: Qram. what is it and why do we need it? making quantum algorithms thrive., 01 2022.
 - [36] Gabriel Landi. Density matrices and composite systems.
 - [37] V. Vijayakrishnan and S. Balakrishnan. Role of two-qubit entangling operators in the modified eisert–wilkins–lewenstein approach of quantization. *Quantum Information Processing*, 18, 03 2019.
 - [38] Real Python. Scientific python: Using scipy for optimization – real python.
 - [39] scipy optimize minimize scalar scipy v1.12.0 manual.
 - [40] Matt Davis. Optimization (scipy.optimize) — scipy v0.19.0 reference guide.
 - [41] scipy.optimize.minimize — scipy v1.6.0 reference guide.