Quantum Search Algorithm

Chirayu Agrawal (202003024) Jugal Patidar (202003032)

May 17, 2023

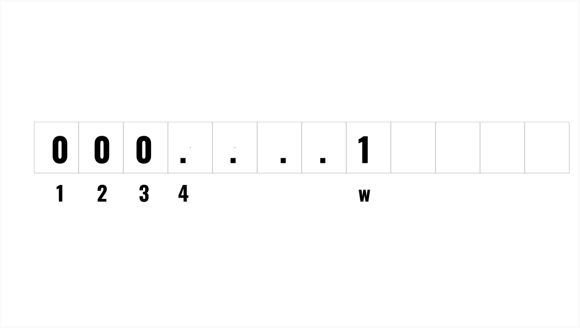
Dhirubhai Ambani Institute of Information and Communication Technology

Introduction

Grover's algorithm, also known as the quantum search algorithm, refers to a quantum algorithm for unstructured search that finds with high probability the unique input to a black box function that produces a particular output value, using just O($\sqrt{N}$) evaluations of the function, where N is the size of the function's domain.The analogous problem in classical computation cannot be solved in fewer than O(N) evaluations

Grover's Algorithm

suppose we are given an unsorted database, and our task is to find the position of a unique special element.



like in this figure, we are asked to give position of element "1" where other elements are 0.

We will use the Grover's algorithm to check triangle in a graph.

Classical algorithm

In classical algorithm we can provide the solution in average of N/2 iterations.
we can traverse the array compare each element and then give position.
The non-required elements provide no knowledge of our element.
In worst case scenario we have to traverse whole array, that is N elements and hence this method has time complexity of O(N). If the data is sorted, we can do binary search and get solution in O(logN) times but that is not the case.

Grover's Algorithm

Grover's algorithm is fast quantum mechanical algorithm for fast search.

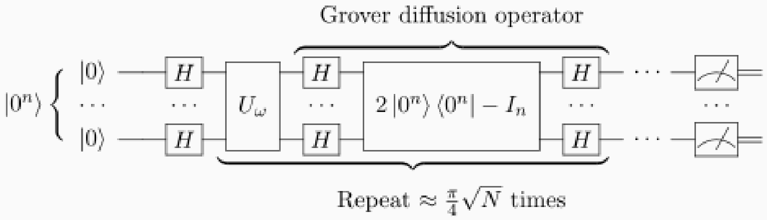
It takes help of qubit superposition and phase interference.

It finds the solution in O($\sqrt{N}$) time complexity that is if we have 100 elements, it can provide solution in max 10 runs.

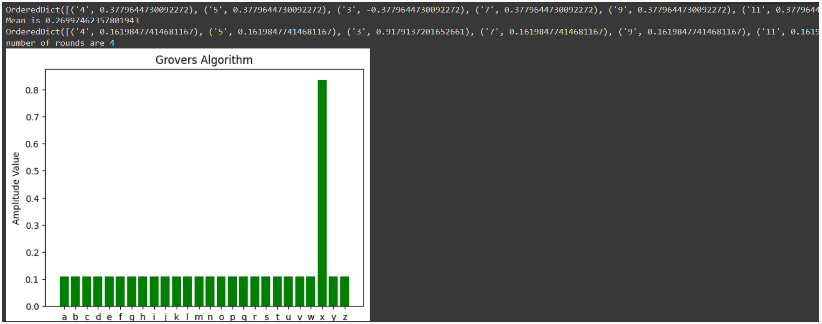
The algorithm

- First, we will start with balanced superposition and assign the value of -1 to chosen ket.
  we will call the matrix that implements the phase flip.
- Next we will invert all probability amplitudes around mean. Through this all the kets become 0 and our chosen ket goes to 1.
  we will call the matrix that flips around mean.
- Our goal is to find the minimum difference between the actual ground state energy and the energy we get by adjusting the parameters, as we want the most optimized solution.This is a minimization problem.

Circuit


Grover diffusion operator
$|0^n\rangle$
$|0\rangle$
H
U_ω
$2|0^n\rangle\langle 0^n| - I_n$
Repeat $\approx \frac{\pi}{4}\sqrt{N}$ times

Code output



```
OrderedDict([('4', 0.3779644730092272), ('5', 0.3779644730092272), ('3', -0.3779644730092272), ('7', 0.3779644730092272), ('9', 0.3779644730092272), ('11', 0.3779644
Mean is 0.26997462357801943
OrderedDict([('4', 0.16198477414681167), ('5', 0.16198477414681167), ('3', 0.9179137201652661), ('7', 0.16198477414681167), ('9', 0.16198477414681167), ('11', 0.1619
number of rounds are 4
```

Figure 1: In the output, we observe first amplitude of all elements are reduced while our element is negative shifted and then one more shift is done along mean and we get our output

Application

We will use the knowledge of grover's algorithm and will find whether a graph contains a triangle.

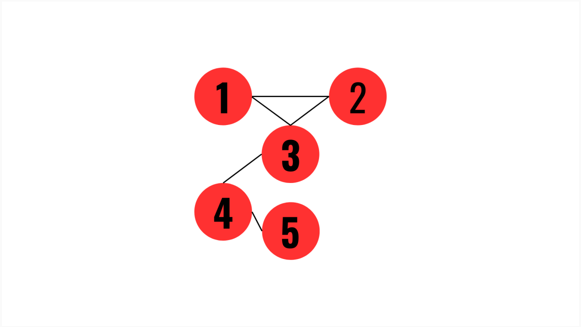


Figure 2: like in above figure, we can see triangle made by nodes 1,2 and 3.

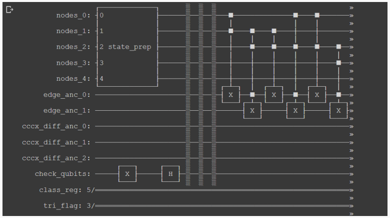
solution

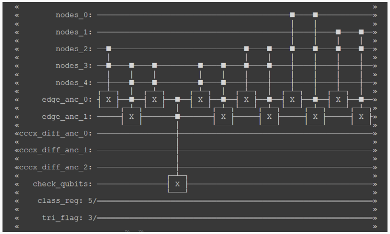
Let's start by focusing on the given example, which involves finding a triangle in a graph consisting of five nodes. In order to accomplish this, we will examine all subgraphs within the main graph and determine if any of them form a triangle. To facilitate this process, we will require five qubits, with each qubit representing a node in the graph. The state of each qubit will indicate whether the corresponding node is present in any subgraph or not. For instance, considering the aforementioned graph, the triangle is formed by nodes 1, 2, and 3, which can be represented by the state 11100. Nodes with a state of 1 are part of the subgraph (triangle), while nodes with a state of 0 are not involved.

Solution

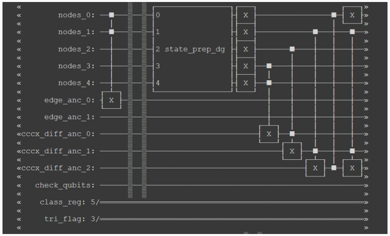
To prepare the state, we can achieve it by applying Hadamard gates to all five qubits, resulting in a superposition of all possible states ranging from 00000 to 11111. Once the state is prepared, we will require three iterations of rotation over the oracle and diffusion steps. However, considering that each 1 in the state represents an active node, we can optimize our search. Instead of examining the entire Hilbert space, we only need to focus on the subgraphs consisting of three nodes. This allows us to streamline our search process and increase efficiency.

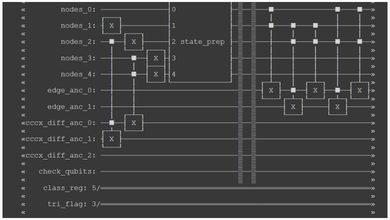
Output


nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
0
1
2 state_prep
3
4
X
H
X
X
X
X
X
X

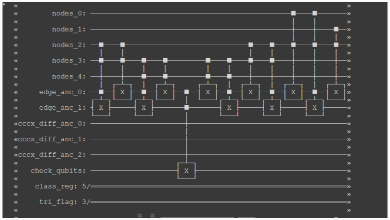

nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
X
X
X
X
X
X
X
X
X
X
X
X
X

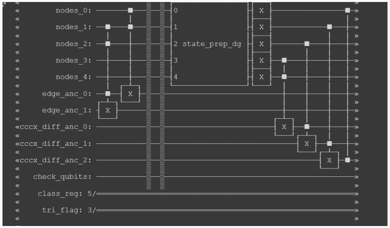
Output


nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
state_prep_dg
X

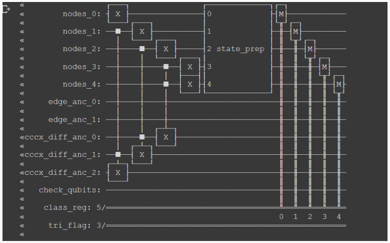

nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
state_prep
X

Output

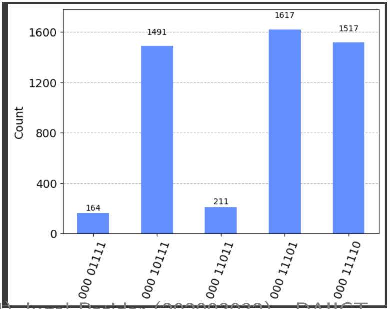

nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
X


nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
state_prep_dg
0
1
2
3
4
X

Output



nodes_0:
nodes_1:
nodes_2:
nodes_3:
nodes_4:
edge_anc_0:
edge_anc_1:
cccx_diff_anc_0:
cccx_diff_anc_1:
cccx_diff_anc_2:
check_qubits:
class_reg: 5/
tri_flag: 3/
X
state_prep
M
0 1 2 3 4



Count
0
400
800
1200
1600
164
1491
211
1617
1517
000 01111
000 10111
000 11011
000 11101
000 11110

Through the output of our code we can see in steps how we got our triangle from the graph.

code link : `https://amazon-braket-g6.notebook.us-east-1.sagemaker.aws/lab/tree/Braket%20examples/getting_started/0_Getting_started/groveralgotofindtri.ipynb`

Account Id - 272143330867
IAM User name - 003
Password - Daiict@123