CSBB 311 : QUANTUM COMPUTING

LAB ASSIGNMENT 8 : Quantum Walk Search Algorithm

Submitted By:

Name: KARTIK MITTAL

Roll No: 221210056

Branch: CSE

Semester: 5th Sem

Group: 2

Submitted To: Dr. VS Pandey

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

2024

Theory -

1. Introduction to Quantum Walk Search Algorithm

- The Quantum Walk Search Algorithm is a quantum computing paradigm inspired by the classical random walk. It enhances search efficiency in structured and unstructured datasets by leveraging quantum superposition and interference.

2. Types of Quantum Walks

- **Discrete-Time Quantum Walks**:These involve stepwise evolution determined by a unitary operator. They are particularly useful for algorithms on graphs and lattice structures.
- **Continuous-Time Quantum Walks**:These are governed by the time evolution of a quantum system under a Hamiltonian. They are better suited for certain combinatorial problems and searches.

3. Workflow of Quantum Error Correction

- **Initialization**: The algorithm initializes the quantum system in a superposition of all possible states, representing all potential solutions to the search problem.
- **Quantum Walk Evolution**: A quantum walk operator is applied iteratively, ensuring that the probability amplitude evolves across the search space while preserving quantum coherence.
- **Marking the Solution**: A phase oracle is used to mark the correct solution by modifying its amplitude, enabling its identification.
- **Amplitude Amplification**: Quantum interference is employed to amplify the probability of finding the correct solution while suppressing others.

4. Importance of Quantum Error Correction

- They outperform classical algorithms in specific scenarios, such as searching through unstructured data or solving graph traversal problems.
- By integrating with other quantum techniques, such as Grover's search or quantum annealing, quantum walks can address complex computational challenges, enhancing the practicality of quantum computing in domains like cryptography, optimization, and machine learning.

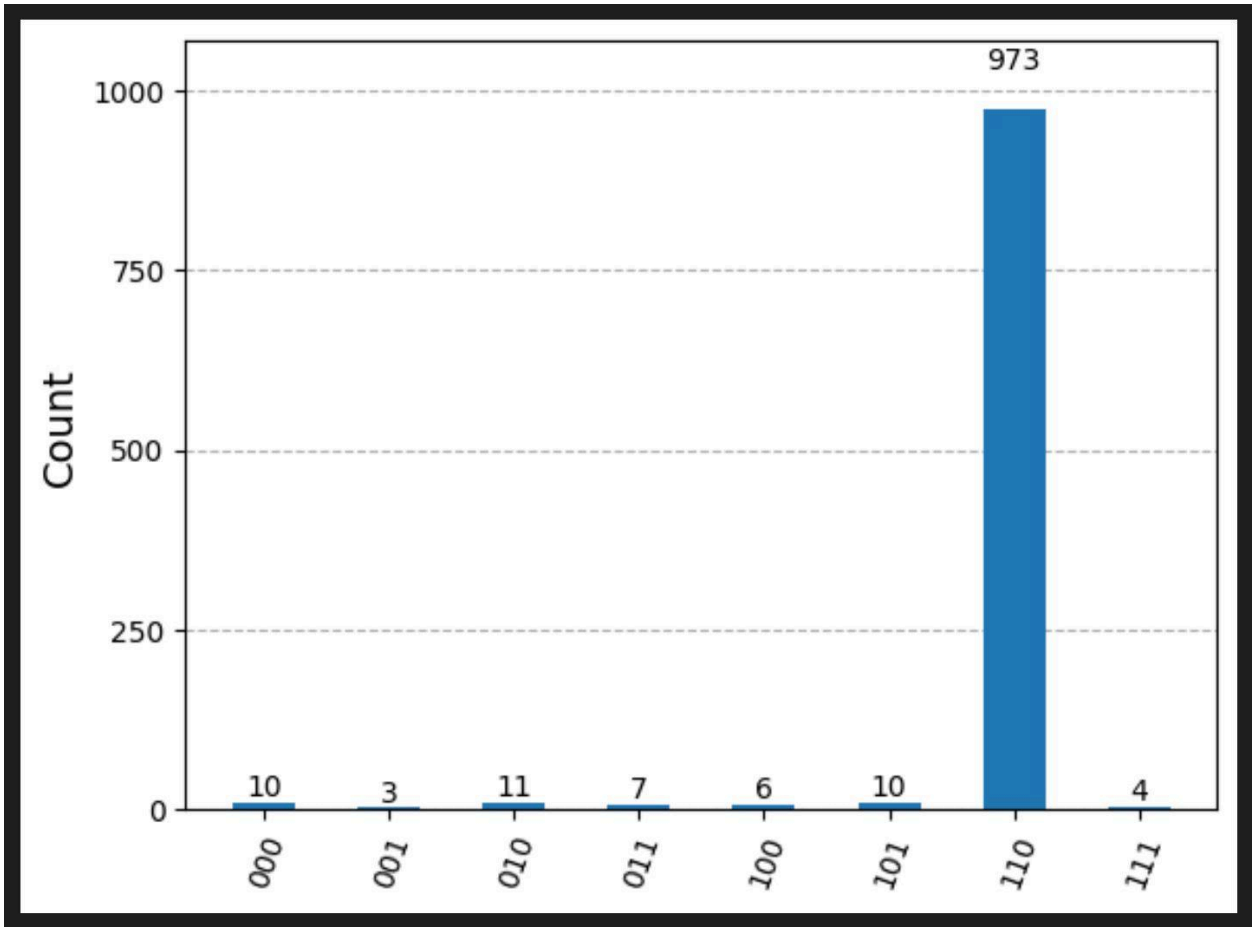
Code (Quantum Walk Search Algorithm) -

```python
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt
import numpy as np

# Function to create the Grover diffusion operator
def diffusion_operator(n_qubits):
    qc = QuantumCircuit(n_qubits)
    qc.h(range(n_qubits))
    qc.x(range(n_qubits))
    qc.h(n_qubits - 1)
    qc.mcx(list(range(n_qubits - 1)), n_qubits - 1)  # Multi-controlled Toffoli
    qc.h(n_qubits - 1)
    qc.x(range(n_qubits))
    qc.h(range(n_qubits))
    return qc.to_gate(label="Diffusion")

# Function to create the oracle (marks the solution state)
def oracle(n_qubits, marked_state):
    qc = QuantumCircuit(n_qubits)
    marked_state_bin = format(marked_state, f'0{n_qubits}b')
    for i, bit in enumerate(marked_state_bin):
        if bit == '0':
            qc.x(i)
    qc.h(n_qubits - 1)
    qc.mcx(list(range(n_qubits - 1)), n_qubits - 1)  # Multi-controlled Toffoli
    qc.h(n_qubits - 1)
    for i, bit in enumerate(marked_state_bin):
        if bit == '0':
```

```
            qc.x(i)
    return qc.to_gate(label="Oracle")

# Number of qubits and target state
n_qubits = 3  # Number of qubits
marked_state = 3  # Target state (e.g., |011⟩ -> decimal 3)

# Quantum Circuit for Quantum Walk Search
qc = QuantumCircuit(n_qubits)

# Initial superposition
qc.h(range(n_qubits))

# Visualize the quantum circuit after initial Hadamard gates
print("Quantum Circuit After Initial Superposition:")

# Render the quantum circuit in a matplotlib figure and display
fig = qc.draw(output='mpl')

# Display the circuit plot
plt.show()  # This will display the quantum circuit

# Number of Grover iterations
num_iterations = int(np.pi / 4 * np.sqrt(2 ** n_qubits))

for _ in range(num_iterations):
    # Apply oracle
    qc.append(oracle(n_qubits, marked_state), range(n_qubits))
    # Apply diffusion operator
    qc.append(diffusion_operator(n_qubits), range(n_qubits))

# Measurement
qc.measure_all()

# Simulate the circuit using AerSimulator
simulator = Aer.get_backend('aer_simulator')
transpiled_qc = transpile(qc, simulator)
result = simulator.run(transpiled_qc, shots=1024).result()
counts = result.get_counts()

# Plot the measurement results (final state probabilities)
print("Measurement Results:", counts)
plot_histogram(counts)
plt.show()
```

Output -

```
Quantum Circuit After Initial Superposition:
     ┌───┐
q_0: ┤ H ├
     ├───┤
q_1: ┤ H ├
     ├───┤
q_2: ┤ H ├
     └───┘
Measurement Results: {'001': 3, '110': 973, '100': 6, '010': 11, '000': 10, '101': 10, '111': 4, '011': 7}
```



Conclusion -

- **Efficiency-Scalability Trade-off**:Quantum walk search algorithms also exhibit an efficiency-scalability trade-off. While they provide significant speedups for specific search problems, their implementation demands complex quantum circuitry and precise control over qubits.
- **Algorithmic Significance**:Quantum walk search algorithms are pivotal for advancing the capabilities of quantum computing. They showcase the power of quantum mechanics in solving computational problems more efficiently than classical methods, especially in structured and combinatorial search spaces.