Date : 9/9/25

```
TASK 8: Grover’s algorithm for a 3-qubits database
```

Aim: To implement Grover’s quantum search algorithm for a 3-qubit search space (8 items) using Cirq and demonstrate that the marked item (target state) can be found with high probability after the optimal number of iterations.

Algorithm - Grover’s algorithm for a 3-qubits database

1. Initialize 3 qubits to |0⟩.
2. Create uniform superposition by Hadamard gates H⊗3.
3. Repeat k = 2 times: Apply oracle marking the target bit string with phase flip on the marked state. Apply diffusion operator (inversion about the mean).
4. Measure the qubits to obtain results peaked at the target state with high probability.

```
!pip install cirq


import cirq
import numpy as np
import matplotlib.pyplot as plt


def grover_3_qubit(target_binary):
    qubits = [cirq.GridQubit(0, i) for i in range(3)]
    circuit = cirq.Circuit()

    # Initialize superposition
    circuit.append(cirq.H.on_each(*qubits))

    # Number of Grover iterations
    N = 2 ** 3
    iterations = int(np.floor(np.pi / 4 * np.sqrt(N)))

    for _ in range(iterations):
        # Oracle
        apply_oracle(circuit, qubits, target_binary)

        # Diffusion
        apply_diffusion(circuit, qubits)

    # Measurement (only once at the end)
    circuit.append(cirq.measure(*qubits, key='result'))

    return circuit, qubits
```

```python
def apply_oracle(circuit, qubits, target_binary):
    # Apply X gates where target bit is 0
    for i, bit in enumerate(target_binary):
        if bit == '0':
            circuit.append(cirq.X(qubits[i]))

    # Multi-controlled Z via H + CCX
    circuit.append(cirq.H(qubits[-1]))
    circuit.append(cirq.CCX(qubits[0], qubits[1], qubits[2]))
    circuit.append(cirq.H(qubits[-1]))

    # Undo X gates
    for i, bit in enumerate(target_binary):
        if bit == '0':
            circuit.append(cirq.X(qubits[i]))


def apply_diffusion(circuit, qubits):
    circuit.append(cirq.H.on_each(*qubits))
    circuit.append(cirq.X.on_each(*qubits))

    circuit.append(cirq.H(qubits[-1]))
    circuit.append(cirq.CCX(qubits[0], qubits[1], qubits[2]))
    circuit.append(cirq.H(qubits[-1]))

    circuit.append(cirq.X.on_each(*qubits))
    circuit.append(cirq.H.on_each(*qubits))


def analyze_results(counts, target):
    total = sum(counts.values())
    success = counts.get(int(target, 2), 0)
    success_rate = success / total * 100

    print(f"Measurement results for target |{target}>:")
    for state in range(8):
        bitstr = format(state, '03b')
        count = counts.get(state, 0)
        pct = count / total * 100
        marker = "<-- Target" if bitstr == target else ""
        print(f"State |{bitstr}>: {count} times ({pct:.2f}%) {marker}")

    print(f"\nSuccess rate: {success_rate:.2f}% (optimal ~94% after 2 iterations)")

    states = [format(i, '03b') for i in range(8)]
    values = [counts.get(i, 0) for i in range(8)]
    colors = ['red' if s == target else 'blue' for s in states]

    plt.bar(states, values, color=colors)
    plt.title(f"Grover's Algorithm Results (Target: |{target}>)")
    plt.xlabel("States")
    plt.ylabel("Counts")
    plt.show()
```

```
if __name__ == "__main__":
    target = "101"
    circuit, qubits = grover_3_qubit(target)

    print("Circuit diagram:")
    print(circuit)

    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=1000)

    counts = result.histogram(key='result')
    analyze_results(counts, target)
```

```
Collecting cirq
  Downloading cirq-1.6.1-py3-none-any.whl.metadata (16 kB)
Collecting cirq-aqt==1.6.1 (from cirq)
  Downloading cirq_aqt-1.6.1-py3-none-any.whl.metadata (4.7 kB)
Collecting cirq-core==1.6.1 (from cirq)
  Downloading cirq_core-1.6.1-py3-none-any.whl.metadata (4.8 kB)
Collecting cirq-google==1.6.1 (from cirq)
  Downloading cirq_google-1.6.1-py3-none-any.whl.metadata (4.9 kB)
Collecting cirq-ionq==1.6.1 (from cirq)
  Downloading cirq_ionq-1.6.1-py3-none-any.whl.metadata (4.7 kB)
Collecting cirq-pasqal==1.6.1 (from cirq)
  Downloading cirq_pasqal-1.6.1-py3-none-any.whl.metadata (4.7 kB)
Collecting cirq-web==1.6.1 (from cirq)
  Downloading cirq_web-1.6.1-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: requests~=2.32 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: attrs>=21.3.0 in /usr/local/lib/python3.12/dist-packages (f
Collecting duet>=0.2.8 (from cirq-core==1.6.1->cirq)
  Downloading duet-0.2.9-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: matplotlib~=3.8 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: networkx~=3.4 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: numpy>=1.26 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: pandas~=2.1 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: sortedcontainers~=2.0 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: scipy~=1.12 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: sympy in /usr/local/lib/python3.12/dist-packages (from circ
Requirement already satisfied: typing_extensions>=4.2 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: tqdm>=4.12 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: google-api-core~=2.22 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: proto-plus~=1.25 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: protobuf<6.0dev,>=5.26.1 in /usr/local/lib/python3.12/dist-
Collecting typedunits (from cirq-google==1.6.1->cirq)
  Downloading typedunits-0.0.1-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.many
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/p
```