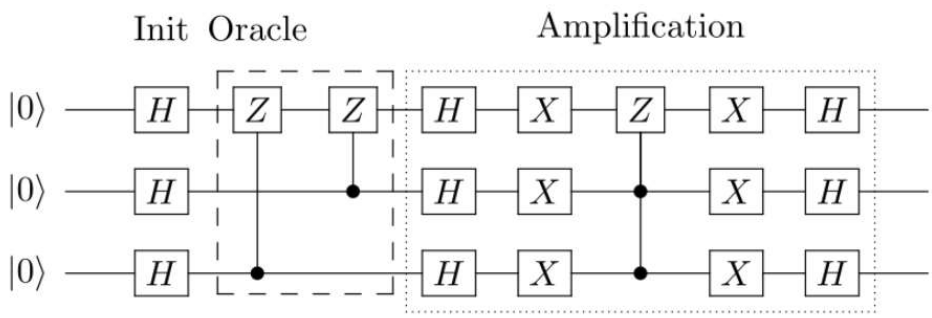


1. Apply Hadamard gates to 3 qubits initialized to $|000\rangle$ to create a uniform superposition:

$$|\psi_1\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

2. Mark states $|101\rangle$ and $|110\rangle$ using a phase oracle:

$$|\psi_2\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

3. Perform the reflection around the average amplitude:
   1. Apply Hadamard gates to the qubits

$$|\psi_{3a}\rangle = \frac{1}{2}(|000\rangle + |011\rangle + |100\rangle - |111\rangle)$$

   2. Apply X gates to the qubits

$$|\psi_{3b}\rangle = \frac{1}{2}(-|000\rangle + |011\rangle + |100\rangle + |111\rangle)$$

   3. Apply a doubly controlled Z gate between the 1, 2 (controls) and 3 (target) qubits

$$|\psi_{3c}\rangle = \frac{1}{2}(-|000\rangle + |011\rangle + |100\rangle - |111\rangle)$$

   4. Apply X gates to the qubits

$$|\psi_{3d}\rangle = \frac{1}{2}(-|000\rangle + |011\rangle + |100\rangle - |111\rangle)$$

   5. Apply Hadamard gates to the qubits

$$|\psi_{3e}\rangle = \frac{1}{\sqrt{2}}(-|101\rangle - |110\rangle)$$

4. Measure the 3 qubits to retrieve states $|101\rangle$ and $|110\rangle$

In [12]:

```
#initialization
import matplotlib.pyplot as plt
import numpy as np

# importing Qiskit
from qiskit import IBMQ, Aer, assemble, transpile
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit.providers.ibmq import least_busy

# import basic plot tools
from qiskit.visualization import plot_histogram
```

We create a phase oracle that will mark states |101> and |110> as the results (step 1)

In [13]:

```
qc = QuantumCircuit(3)
qc.cz(0, 2)
```

```python
qc.cz(1, 2)
oracle_ex3 = qc.to_gate()
oracle_ex3.name = "U$_\omega$"
```

General Diffuser

In [14]:

```python
def diffuser(nqubits):
    qc = QuantumCircuit(nqubits)
    # Apply transformation |s> -> |00..0> (H-gates)
    for qubit in range(nqubits):
        qc.h(qubit)
    # Apply transformation |00..0> -> |11..1> (X-gates)
    for qubit in range(nqubits):
        qc.x(qubit)
    # Do multi-controlled-Z gate
    qc.h(nqubits-1)
    qc.mct(list(range(nqubits-1)), nqubits-1)  # multi-controlled-toffoli
    qc.h(nqubits-1)
    # Apply transformation |11..1> -> |00..0>
    for qubit in range(nqubits):
        qc.x(qubit)
    # Apply transformation |00..0> -> |s>
    for qubit in range(nqubits):
        qc.h(qubit)
    # We will return the diffuser as a gate
    U_s = qc.to_gate()
    U_s.name = "U$_s$"
    return U_s
```

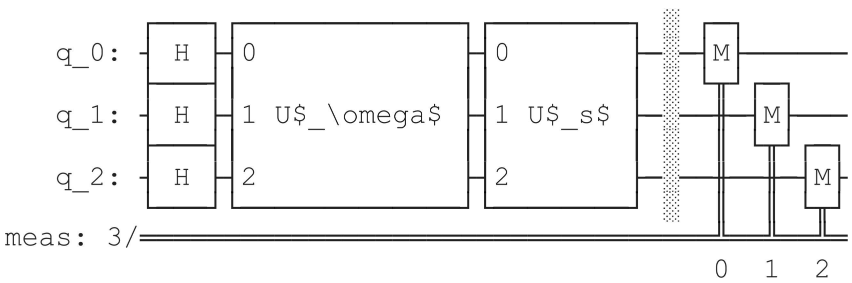
In [15]:

```python
def initialize_s(qc, qubits):
    """Apply a H-gate to 'qubits' in qc"""
    for q in qubits:
        qc.h(q)
    return qc
```

In [16]:

```python
n = 3
grover_circuit = QuantumCircuit(n)
grover_circuit = initialize_s(grover_circuit, [0,1,2])
grover_circuit.append(oracle_ex3, [0,1,2])
grover_circuit.append(diffuser(n), [0,1,2])
grover_circuit.measure_all()
grover_circuit.draw()
```

Out[16]:



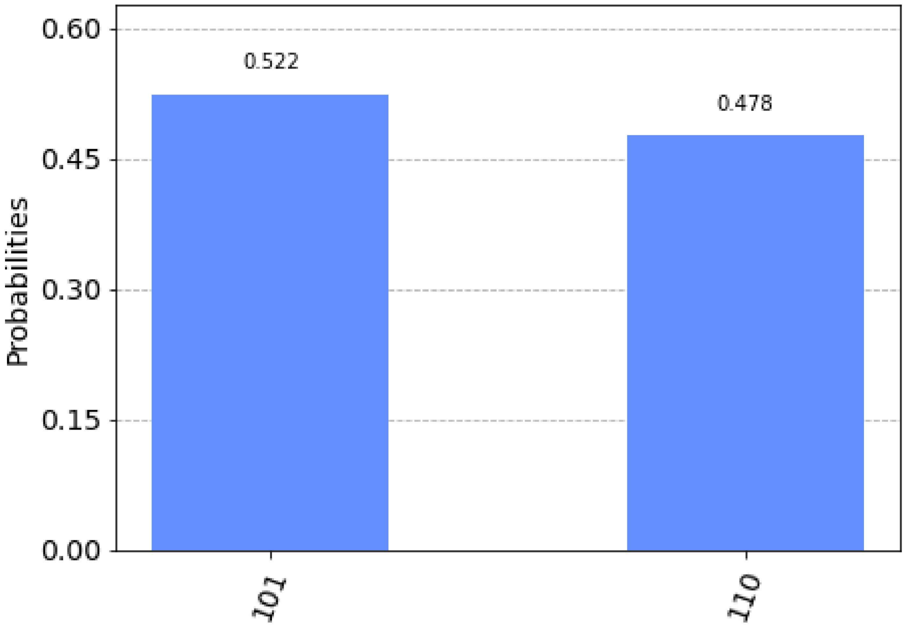
Result(Sumilator)

In [17]:

```python
aer_sim = Aer.get_backend('aer_simulator')
transpiled_grover_circuit = transpile(grover_circuit, aer_sim)
qobj = assemble(transpiled_grover_circuit)
```

```
results = aer_sim.run(qobj).result()
counts = results.get_counts()
plot_histogram(counts)
```

Out[17]:



Result(Real Device)

In [19]:

```
# Load IBM Q account and get the least busy backend device
provider = IBMQ.load_account()
provider = IBMQ.get_provider("ibm-q")
device = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >
                                   not x.configuration().simulator and x.status().op
print("Running on current least busy device: ", device)
```

```
Running on current least busy device:  ibmq_quito
```

In [20]:

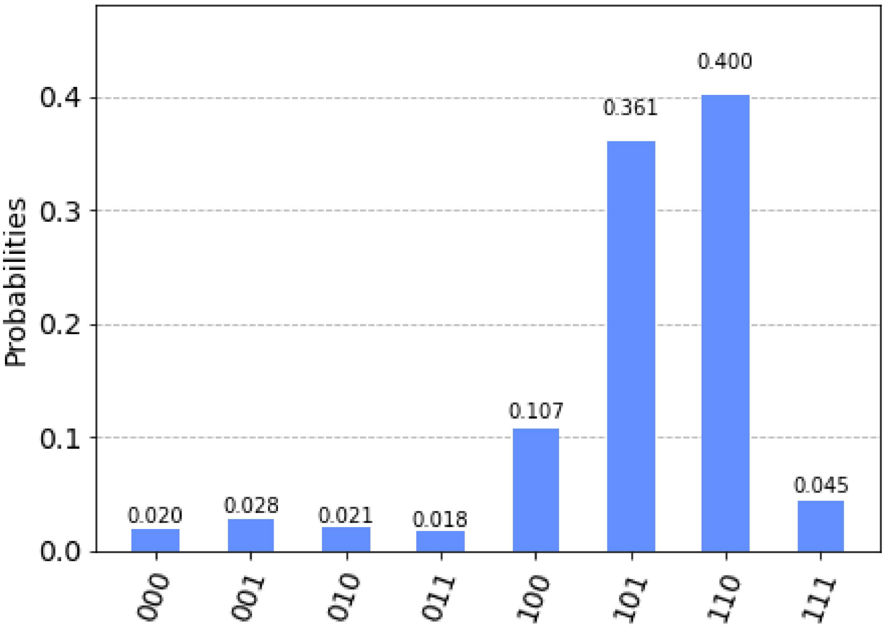
```
# Run our circuit on the least busy backend. Monitor the execution of the job in the
from qiskit.tools.monitor import job_monitor
transpiled_grover_circuit = transpile(grover_circuit, device, optimization_level=3)
job = device.run(transpiled_grover_circuit)
job_monitor(job, interval=2)
```

```
Job Status: job has successfully run
```

In [21]:

```
# Get the results from the computation
results = job.result()
answer = results.get_counts(grover_circuit)
plot_histogram(answer)
```

Out[21]:



Probabilities
0.4
0.3
0.2
0.1
0.0
0.020
0.028
0.021
0.018
0.107
0.361
0.400
0.045
000
001
010
011
100
101
110
111

In [ ]: