Quantum computing Homework- 2 & 3

Adrian Kowalski

November 2025

Homework 2 & 3

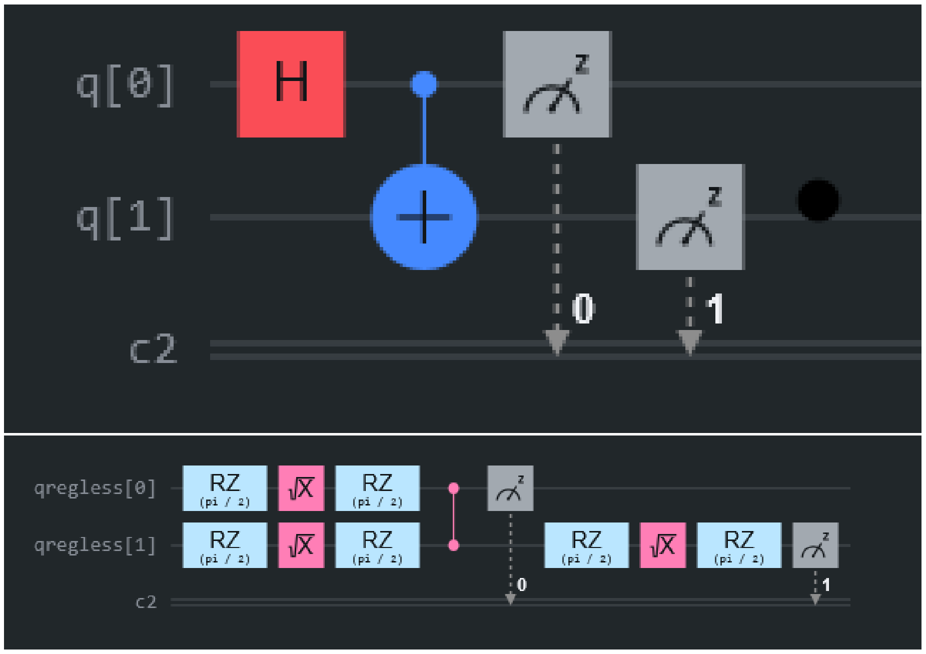
Reference book:

Chapter 3 and 4 of
Roberto Loredo. Learn Quantum Computing with Python and IBM Quantum Experience A hands-on introduction to quantum computing and writing your own quantum programs with Python.

Contents

1 Qiskit codes and results (Homework 2):

1.1 Composinig the circuit



1.1.1 QASM 2.0 Code:

OPEN QASM 2.0

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];
h q[0];
cx q[0], q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

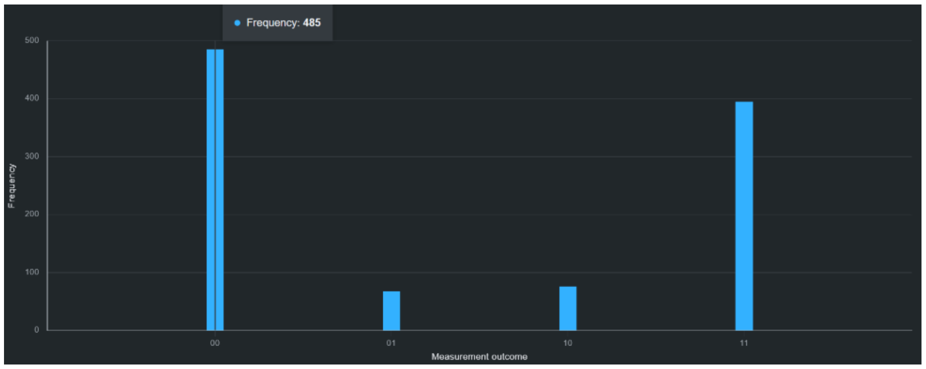
1.1.2 Qiskit Code:

Qiskit

```
from qiskit import QuantumRegister,
ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
```

1.2 Results:

1.2.1 Histogram:



1.2.2 Measurments:

Qiskit

```
counts = results.get_counts(circuit)
print(counts)
```

{'00': 482, '01': 68, '10': 76, '11': 395}

2 Questions (Homework 2):

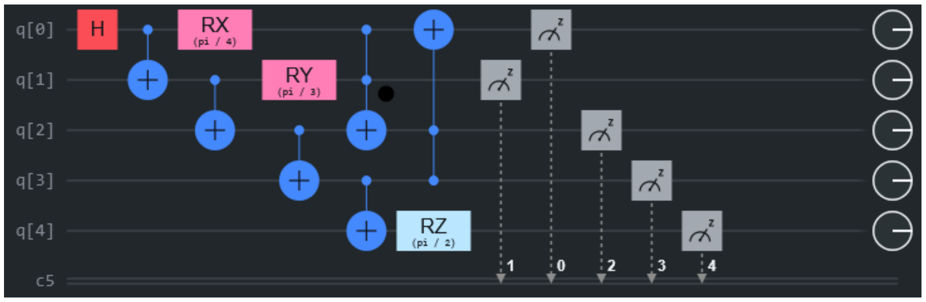
2.1 Quantum Lab notebooks are built upon which application editor?

IBM Quantum Lab notebooks are built upon the Jupyter Notebook environment. The features this provides are creating files, running kernels, running triggering cells. It also visualizes results by providing user with different type of graphs etc. including Matplotlib, Latex and so on.

2.2 How would you create a 5-qubit circuit, as we did in Chapter 2, Circuit Composer - Creating a Quantum Circuit?

The Circuit is made out of:

- one Hadamard gate (H)
- four CNOT gates (entangle all 5 qubits)
- three rotations each in diffrent axis (RX, RY, RZ)
- two toffoli gates (CCNOT Gates)
- and measurments at the end



2.3 To run the experiment on another real device, which quantum computer would you select if your quantum circuit has more than 5 qubits?

To run that experiment on a real device we would need it to have 5 qubits or more so for examples devices like:

1. ibmq_jakarta (7 qubits)
2. ibm_nairobi (7 qubits)
3. ibm_oslo (7 qubits)
4. imbq_guadalupe (16 qubits)

could do it. Today there are devices qith many more qubits but for such a small circuit we don't need anything bigger.

2.4 When you run on a real device, can you explain why you get extra values when compared to running on a simulator?

When running on a simulator, we assume perfect gates and no noise, so we get pure results free of any errors.
However, when we run on a real quantum device, such as NISQ devices, we observe extra or unexpected errors that do not align with the simulation. These differences are caused due to inherent imperfections in hardware, including gate errors, decoherence, and readout errors, which can introduce bit flips and phase errors.

3 Executing the quantum teleportation in qiskit circuit (Homework 3):

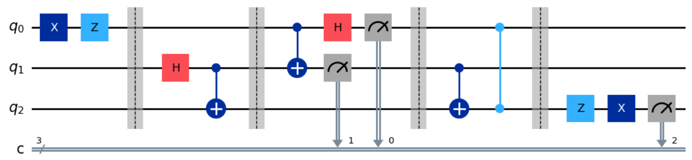
3.1 Composinig the circuit

3.1.1 Circuit in Qiskit code:

Qiskit

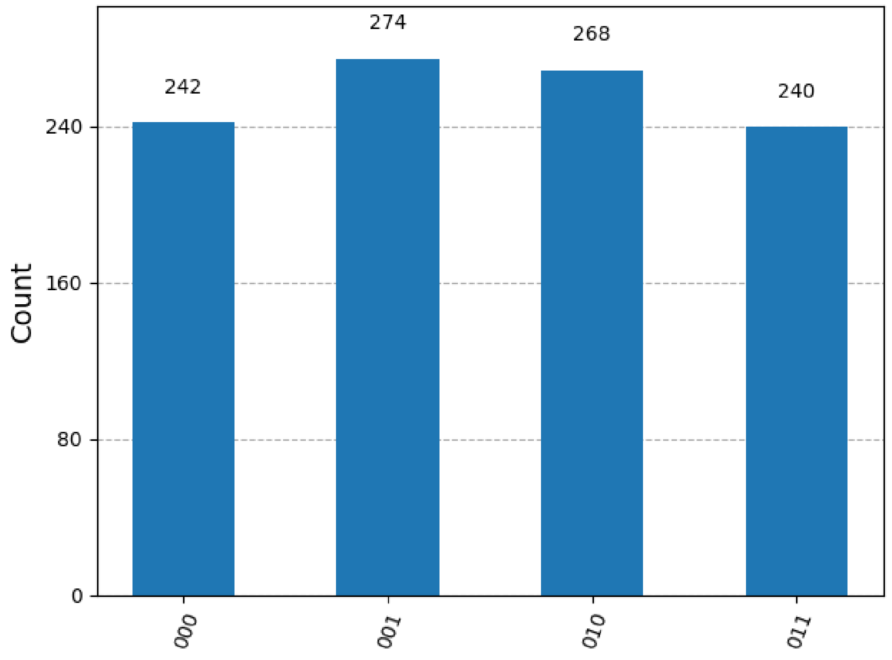
```
from qiskit import QuantumCircuit,
QuantumRegister, ClassicalRegister
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

q = QuantumRegister(3, "q")
c = ClassicalRegister(3, "c")
qc = QuantumCircuit(q, c)
qc.x(0)
qc.z(0)
qc.barrier()
qc.h(1)
qc.cx(1, 2)
qc.barrier()
qc.cx(0, 1)
qc.h(0)
qc.measure(0, 0)
qc.measure(1, 1)
qc.barrier()
qc.cx(1, 2)
qc.cz(0, 2)
qc.barrier()
qc.z(2)
qc.x(2)
qc.measure(2, 2)

qc.draw(output='mpl')
sim = AerSimulator()
job = sim.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()
print("Counts:", counts)
fig = plot_histogram(counts)
plt.show()
```

3.1.2 Circuit in gate form:



3.2 Results of the simulation:

3.2.1 Histogram:



3.2.2 Measurements:

Qiskit

```
counts = result.get_counts()
print("Counts:", counts)
```

Counts: {'010': 256, '000': 238, '011': 275, '001': 255}

4 Questions (Homework 3):

4.1 How would you create a circuit that entangles two qubits where each qubit is different (that is, 01, 10)?

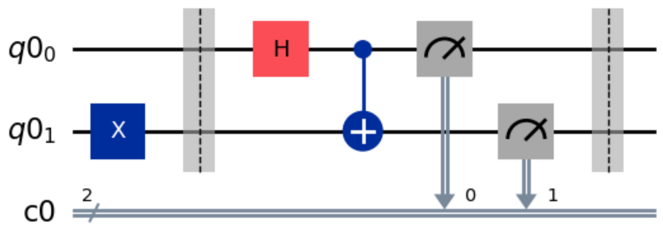
To create such a circuit i would use a X (NOT) gate on qbit[1], so from satrting state $|00\rangle$ we have $|01\rangle$ position. Then i would apply H gate to the first qbit[0] to put into superposition and then entangle them using CNOT gate where the qbitp[0] is the controling one. Lastly we measure the outcome.

4.1.1 Circuit in Qiskit code:

Qiskit

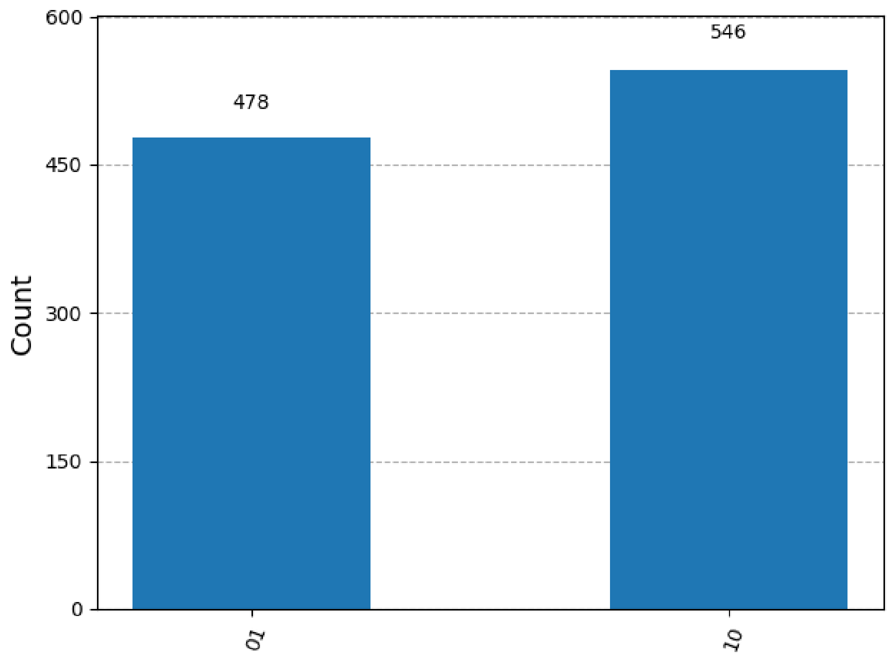
```
import numpy as np
from qiskit import QuantumCircuit,
QuantumRegister, ClassicalRegister
from qiskit.visualization import plot_histogram
from qiskit_aer import AerSimulator
import matplotlib.pyplot as plt

q = QuantumRegister(2)
c = ClassicalRegister(2)
qc = QuantumCircuit(q, c)
qc.x(1)
qc.barrier()
qc.h(0)
qc.cx(0, 1)
qc.measure(q,c)
qc.barrier()
qc.draw(output='mpl')

sim = AerSimulator()
result = sim.run(qc, shots=1024).result()
counts = result.get_counts()
print("Counts:", counts)
fig = plot_histogram(counts)
plt.show()
```

4.1.2 Circuit in gate form:



4.1.3 Results of the simulation:

Histogram:



Measurements

Counts: '01': 478, '10': 546

4.2 Which simulator is used to display the Bloch sphere?

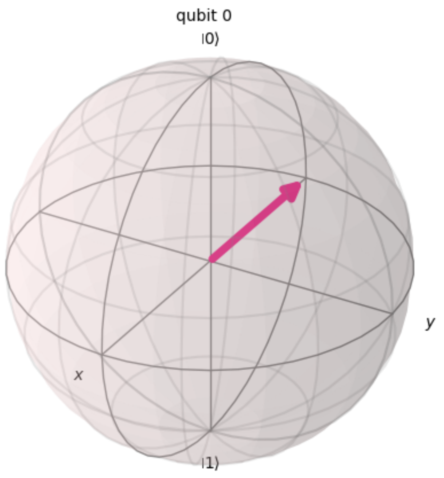
To display the Bloch sphere in qiskit, we would use a Statevector simulator.

4.2.1 Exemplary code in Qiskit:

Qiskit

```
from qiskit import QuantumCircuit
from qiskit.visualization import plot_bloch_multivector
from qiskit.quantum_info import Statevector
import matplotlib.pyplot as plt

qc = QuantumCircuit(1)
qc.x(0)
qc.h(0)

state = Statevector.from_instruction(qc)
plot_bloch_multivector(state)

plt.show()
```

4.2.2 Bloch sphere:



4.3 Execute the superposition experiment with the shots=1 parameter, then shots=1000, and then shots=8000. What is the difference?

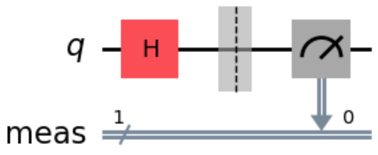
The only difference is that with bigger number of shots we get closer to the 50% split between our two states of qbit in measurements.

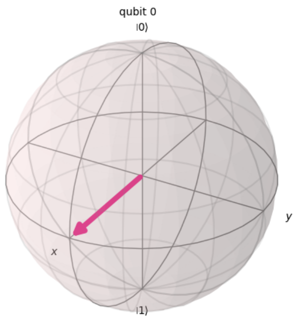
4.3.1 Circuit in Qiskit code:

Qiskit

```
from qiskit.visualization import plot_bloch_multivector,
plot_histogram
from qiskit.quantum_info import Statevector
from qiskit_aer import AerSimulator
import matplotlib.pyplot as plt

shots_list = [1, 1000, 8000]

qc = QuantumCircuit(1)
qc.h(0)
qc_state = qc.copy()
qc.measure_all()
qc.draw(output='mpl')

state = Statevector.from_instruction(qc_state)
plot_bloch_multivector(state)
plt.show()

all_states = ['0','1']

for shots in shots_list:
    sim = AerSimulator()
    result = sim.run(qc, shots=shots).result()
    counts = result.get_counts()

    full_counts = {state: counts.get(state, 0)
        for state in all_states}
    plot_histogram(full_counts)
    plt.title(f"Histogram for {shots} shots")
    plt.show()
```

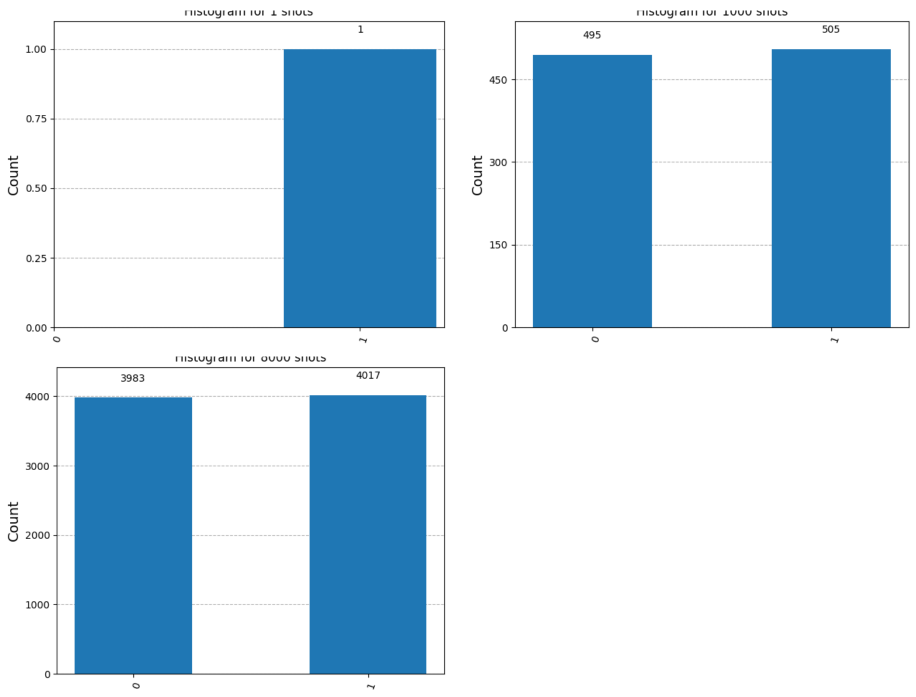
4.3.2 Circuit in gate form and bloch sphere:





4.3.3 Results of the simulation:

Histograms:



Measurements

Shots = 1 Counts: '0': 0, '1': 1
Shots = 1000 Counts: '0': 495, '1': 505
Shots = 8000 Counts: '0': 3983, '1': 4017

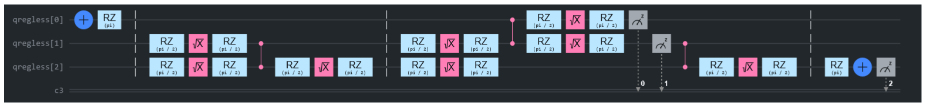
4.4 Run the quantum teleportation experiment on a real quantum device and describe the results compared to the simulator's results. What's different, if anything, and why?

When running on a simulator, we assume perfect gates and no noise, so we get pure results free of any errors.
However, when we run on a real quantum device, such as NISQ devices, we observe extra or unexpected errors that do not align with the simulation. These differences are caused due to inherent imperfections in hardware, including gate errors, decoherence, and readout errors, which can introduce bit flips and phase errors. And because of that in measurements, we get results that might be interpreted as someone tampering with our teleportation.

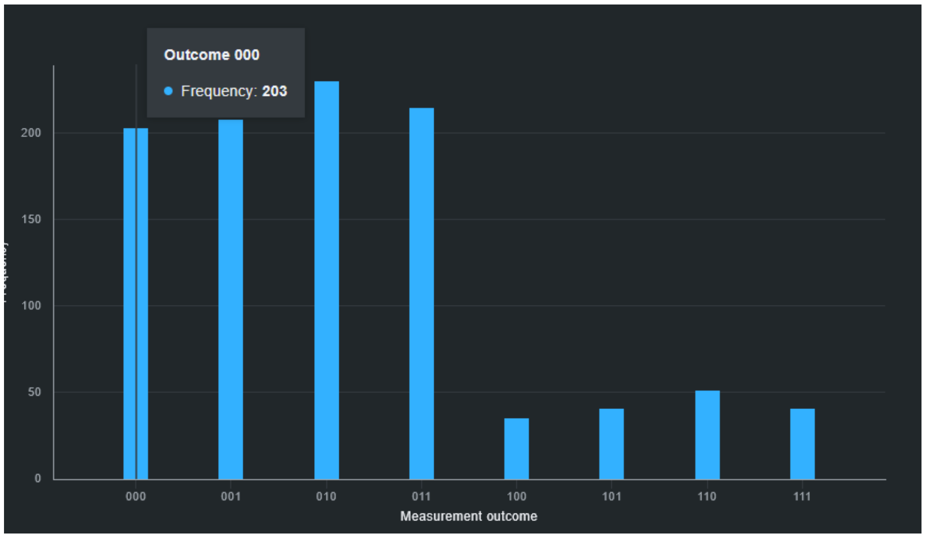
Quantum computer used:

ibm_torino - 133 qbits

4.4.1 Circuit in gate form:



4.4.2 Histogram:



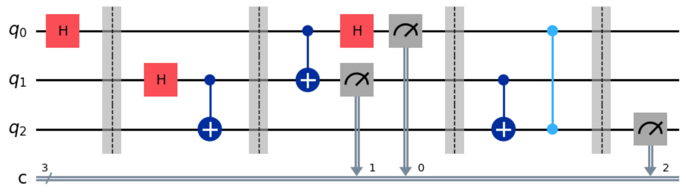
4.5 Use your program and try any other quantum state. Try to teleporting the state zero plus one over the square root of two. Explain with this quantem state what are the outcomes.

4.5.1 Cirquit in Qiskit code:

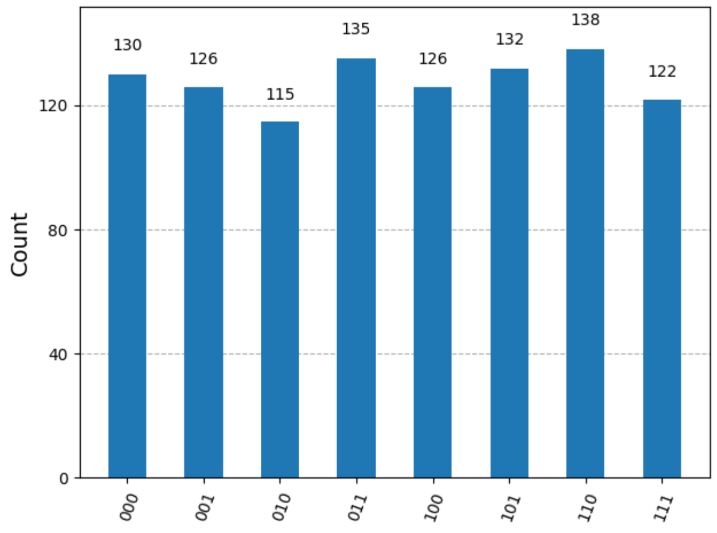
Qiskit

```
import numpy as np
from qiskit import QuantumCircuit,
QuantumRegister, ClassicalRegister
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

q = QuantumRegister(3, "q")
c = ClassicalRegister(3, "c")
qc = QuantumCircuit(q, c)
qc.h(0)
qc.barrier()
qc.h(1)
qc.cx(1, 2)
qc.barrier()
qc.cx(0, 1)
qc.h(0)
qc.measure(0, 0)
qc.measure(1, 1)
qc.barrier()
qc.cx(1, 2)
qc.cz(0, 2)
qc.barrier()
qc.measure(2, 2)

qc.draw(output='mpl')
sim = AerSimulator()
job = sim.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()
print("Counts:", counts)
fig = plot_histogram(counts)

plt.show()
```

4.5.2 Circuit in gate form:



4.5.3 Histogram:



4.5.4 Measurements:

Counts: {'000': 130, '001': 126, '010': 115, '011': 135, '100': 126, '101': 132, '110': 138, '111': 122}

4.5.5 Explanation:

We start with q[0] in state = $|0\rangle$, then we apply Hadamard gate to it:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$q[0] = |+\rangle$$

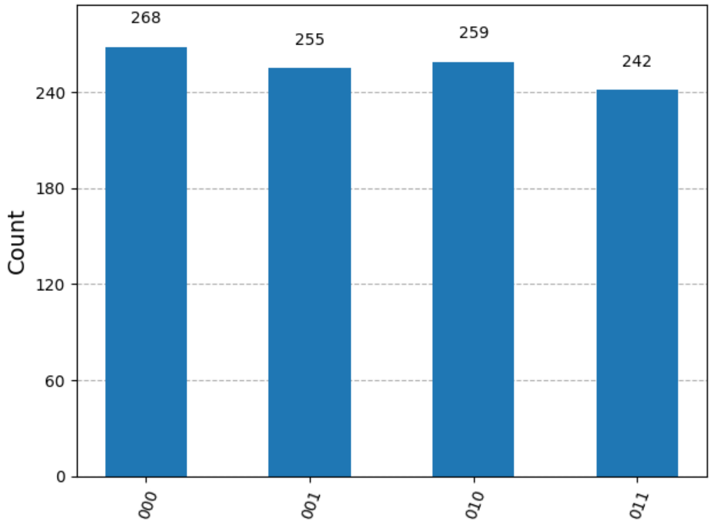
We proceed with quantum teleportation algorithm, and at the end we should see that superposition on qbit q[2]. Which would yield results with split close to 50% on the first qbit.
That means we created pairs:

1. '000': 130 , '100': 126;
2. '001': 126, '101': 132;
3. '010': 115, '110': 138;
4. '011':135, '111':122;

Pairs are roughly split in the middle, which support our theory.

4.5.6 Histogram:

To verify we can add another H gate at the end to q[2] that would collapse superposition back to 0.



As we can see the results are back to normal.
We successfully teleported the superposition from :

$$q[0] = |+\rangle$$

to:

$$q[2] = |+\rangle$$