양자계산과 보안 기말프로젝트

인공지능사이버보안학과
2021271314
김건희

Index

01 Shor’s Order Finding Algorithm

- N = 15, a = 4
- $U|x\rangle = |4x \ (mod \ 15)\rangle$, where $x \in \{0, 1, 2, 3, \cdots, 14\}$
- $x = 4 \because 4 \cdot 4 \ mod \ 15 = 1$
- $i.e.$ 양자 회로 알고리즘의 결과로 2가 나옴.

02 Shor's Order Finding Algorithm using IBM composer

```
pip install qiskit
```

```
pip install qiskit-aer
```

```python
# pylint: disable=invalid-name
import matplotlib.pyplot as plt
import numpy as np
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.visualization import plot_histogram
```

```python
# Specify variables
N_COUNT = 8  # number of counting qubits
a = 4
N = 15
```

- qiskit과 qiskit-aer 모듈을 설치한다.
- 코드를 실행하기 위한 필요한 패키지들을 임포트한다.
- N_COUNT는 카운트에 사용할 큐비트의 개수이다. 해당 값을 8로 지정한다.
- N = 15, a = 4

02 Shor's Order Finding Algorithm using IBM composer

```python
def c_xmod15(power):
    """Controlled multiplication by x mod 15"""
    if power != 4:
        raise ValueError("'power' must be 4")
    U = QuantumCircuit(4)
    for _iteration in range(power):
            U.swap(1, 3)
            U.swap(0, 2)
    U = U.to_gate()
    U.name = f"x^{power} mod 15"
    c_U = U.control()
    return c_U
```

→ c_amod15라는 함수는 주어진 a와 power에 따라 15로 나눈 나머지를 구하는 연산을 수행한다.

- QuantumCircuit 객체 U를 생성한다.
- 이 회로는 4개의 양자 비트를 사용한다.
- 4개의 큐비트로 이루어진 양자 회로 U를 생성한다.
- 회로는 power 횟수만큼 반복되는 swap 연산을 수행한다.
- U.swap(1,3)은 1번 큐비트와 3번 큐비트의 위치를 바꿔주는 것, U.swap(0,2)는 0번 큐비트와 2번 큐비트의 위치를 바꿔주는 것을 의미한다.
- U 회로를 게이트로 변환하고, a^power mod 15라는 이름을 지정한다.
- 이 게이트를 제어 게이트로 만들어 c_U로 반환

02 Shor's Order Finding Algorithm using IBM composer

```python
def qft_dagger(n):
    """n-qubit QFTdagger the first n qubits in circ"""
    qc = QuantumCircuit(n)
    # Don't forget the Swaps!
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT†"
    return qc
```

→ qtf_dagger 함수는 n개의 큐피트에 대해 QTF†을 수행하는 양자 회로를 생성한다.

- n개의 큐비트를 가진 양자 회로 qc를 생성 후, qc.swap(qubit, n-qubit-1)을 사용하여 큐비트의 위치를 바꿔준다.
- 중첩된 for 루프를 사용하여 QFT†을 구성한다.
- 바깥쪽 루프는 j를 0부터 n-1까지 반복하며, 안쪽 루프는 m을 0부터 j-1까지 반복한다.(이때, qc.cp(-np.pi/float(2**(j-m)), m, j)는 제어된 phase shift 게이트를 적용하는 것을 의미한다.)
- qc.h(j)는 H 게이트(약한 측정 게이트)를 적용하는 것을 의미한다.
- 마지막으로, 양자 회로의 이름을 "QFT†"로 지정하고, qc를 반환한다.

02 Shor's Order Finding Algorithm using IBM composer

```python
# Create QuantumCircuit with N_COUNT counting qubits
# plus 4 qubits for U to act on
qc = QuantumCircuit(N_COUNT + 4, N_COUNT)

# Initialize counting qubits
# in state |+>
for q in range(N_COUNT):
    qc.h(q)

# And auxiliary register in state |1>
qc.x(N_COUNT)

# Do controlled-U operations
for q in range(N_COUNT):
    qc.append(c_amod15(a, 2**q),
             [q] + [i+N_COUNT for i in range(4)])

# Do inverse-QFT
qc.append(qft_dagger(N_COUNT), range(N_COUNT))

# Measure circuit
qc.measure(range(N_COUNT), range(N_COUNT))
qc.draw(fold=-1)  # -1 means 'do not fold'
```

- N_COUNT 수의 카운팅 큐비트를 초기화하고, 각 큐비트에 Hadamard 게이트 qc.h(q)를 적용하여 상태를 $|+\rangle$로 설정한다.
- 보조 레지스터를 $|1\rangle$ 상태로 초기화합니다. qc.x(N_COUNT)를 사용하여 보조 레지스터의 첫 번째 큐트를 반전시킨다.
- N_COUNT 수의 카운팅 큐비트를 대상으로 제어된 U 연산을 수행합니다. 이를 위해 for 루프를 사용하여 각 카운팅 큐비트에 대해 c_amod15(a, 2**q) 게이트를 적용한다.

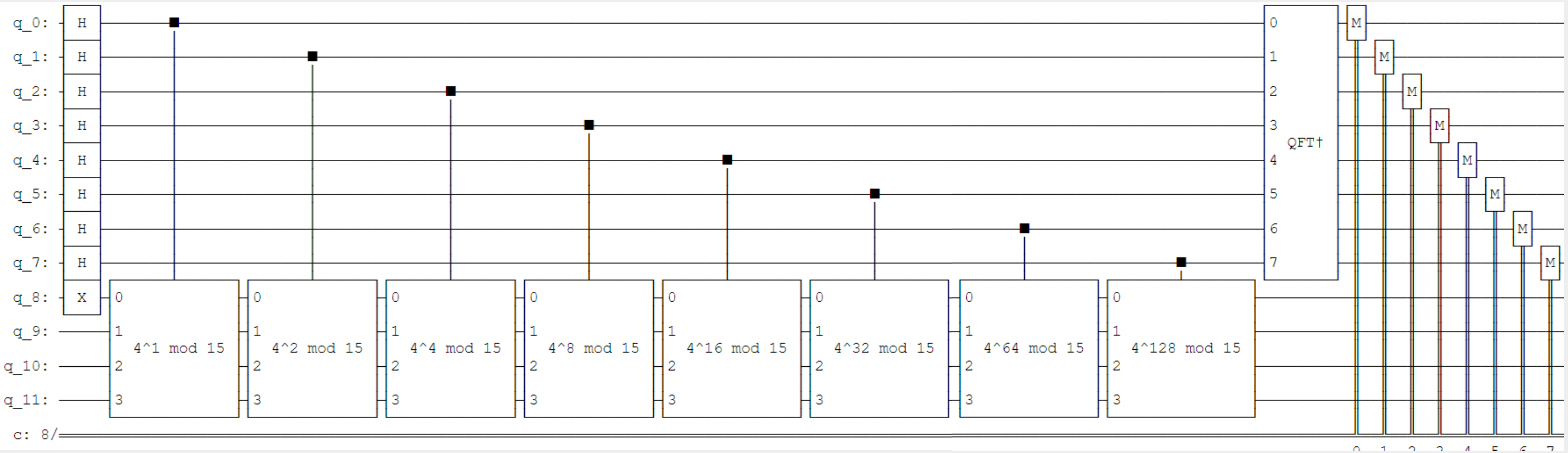
→ N_COUNT 수의 카운팅 큐비트와 4개의 큐비트를 가지는 양자 회로를 생성한다.

02 Shor's Order Finding Algorithm using IBM composer

```python
# Create QuantumCircuit with N_COUNT counting qubits
# plus 4 qubits for U to act on
qc = QuantumCircuit(N_COUNT + 4, N_COUNT)

# Initialize counting qubits
# in state |+>
for q in range(N_COUNT):
    qc.h(q)

# And auxiliary register in state |1>
qc.x(N_COUNT)

# Do controlled-U operations
for q in range(N_COUNT):
    qc.append(c_amod15(a, 2**q),
             [q] + [i+N_COUNT for i in range(4)])

# Do inverse-QFT
qc.append(qft_dagger(N_COUNT), range(N_COUNT))

# Measure circuit
qc.measure(range(N_COUNT), range(N_COUNT))
qc.draw(fold=-1)  # -1 means 'do not fold'
```

- 이때, qc.append(c_amod15(a, 2**q), [q] + [i+N_COUNT for i in range(4)])를 사용하여 제어된 U 연산을 구현한다.
- 그다음, 역-QFT(QFT†)를 수행한다.
- qc.append(qft_dagger(N_COUNT), range(N_COUNT))를 사용하여 역-QFT 게이트를 적용한다.
- Qc.measure(range(N_COUNT), range(N_COUNT)) 를 사용하여 카운팅 큐비트를 측정한다.
- Qc.draw(fold=-1)를 사용하여 양자 회로를 그립니다.

02 Shor's Order Finding Algorithm using IBM composer



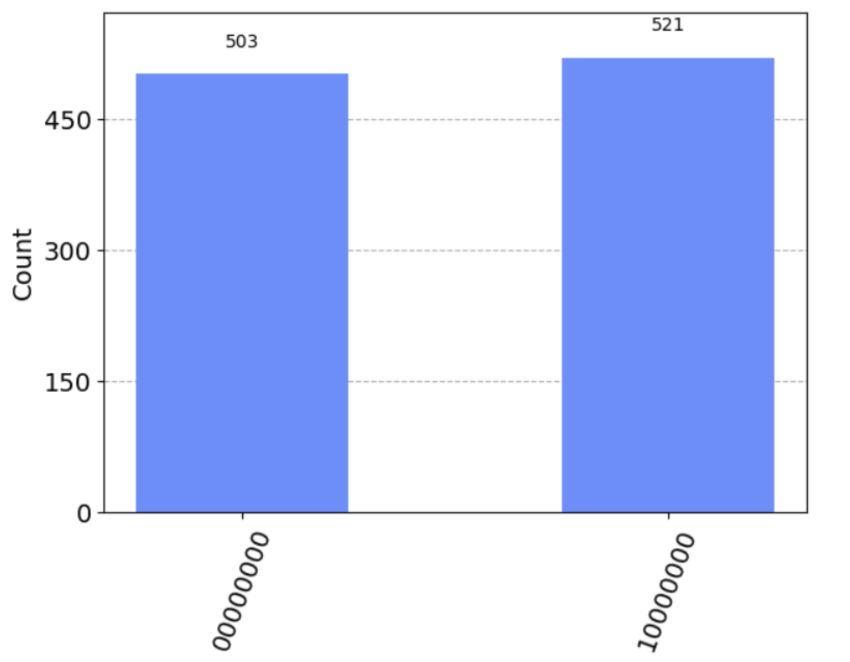
→ 전체 회로 결과

03 Result analysis

In [16]:

```
aer_sim = Aer.get_backend('aer_simulator')
t_qc = transpile(qc, aer_sim)
counts = aer_sim.run(t_qc).result().get_counts()
plot_histogram(counts)
```

Out[16]:



→ 측정 결과의 확률 분포를 히스토그램으로 시각화

- Aer 모듈에서 'aer_simulator' 백엔드를 가져와 aer simulator 변수에 할당한다.
- transpile 함수를 사용하여 양자 회로 qc를 시뮬레이션 백엔드 aer_sim에 맞게 변환한다. (이 단계는 회로를 백엔드에서 실행 가능한 형태로 변환한다.)
- aer_sim 백엔드를 사용하여 변환된 회로 t_qc를 실행하고, 실행 결과인 Result 객체에서 get_counts() 메서드를 호출하여 측정 결과의 확률 분포를 얻는다.
- plot_histogram 함수를 사용하여 측정 결과의 확률 분포를 히스토그램으로 시각화한다.

03 Result analysis

위수 a x n을 했을 때 모듈러에 대해 1이 남는게 위수. 2 x 8 = 16 모듈러 15 는 1

a0 = 0, a1 = 2

a^r ≡ 1 (mod N)을 만족하는 r 값은 r = 2이다. 4^2 (mod 15) = 1을 만족하기 때문이다.

gcd(N, a^(r/2) + 1) = p, gcd(N, a^(r/2) - 1) = q

먼저 a^(r/2) 값을 계산해보자.

a^(r/2) = 4^(2/2) = 4^1 = 4 이다.

gcd(15, 4 + 1) = gcd(15, 5) = 5 → p = 5

gcd(15, 4 - 1) = gcd(15, 3) = 3 → q = 3

따라서 p = 5, q = 3 값을 구할 수 있다.

THANK YOU