양자 컴퓨터 기초

유튜브: https://www.youtube.com/watch?v=rMrV1Bp_8P0

양자컴퓨터란?

양자컴퓨터와 고전컴퓨터

양자 게이트

실습

양자 컴퓨터란?

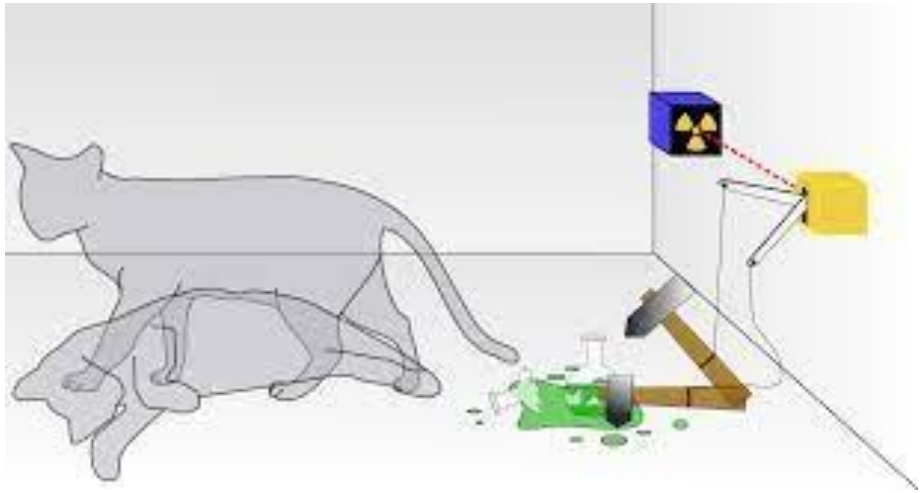
- **양자 컴퓨터**
  - 양자의 중첩과 얽힘 현상을 활용한 컴퓨터
  - 측정 전까지는 0과 1 동시에 공존

❖ 슈뢰딩거의 고양이

1. 고양이를 외부와 차단된 상자에 넣는다.
2. 라듐 핵이 붕괴하면 가이거계수기가 탐지한다.
3. 망치가 유리병을 내려쳐 깨게 돼 청산가리가 유출된다. 청산가리를 마신 고양이는 죽게 된다.

✓ 라듐이 붕괴할 확률은 1시간 뒤 50퍼센트 1시간 뒤 고양이는 죽었을까 살았을까?

-> 관측 전까지 고양이는 살아있는 상태와 죽어있는 상태 공존

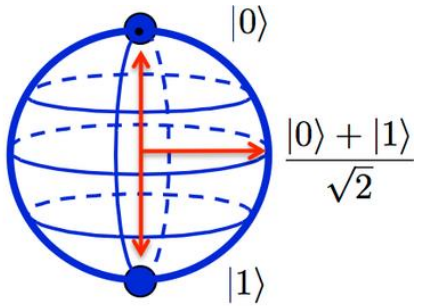


- 연산 속도가 굉장히 빠름

  ex) 최악의 경우 N번 탐색해야 하는 문제도 $\sqrt{N}$번 이면 탐색 가능

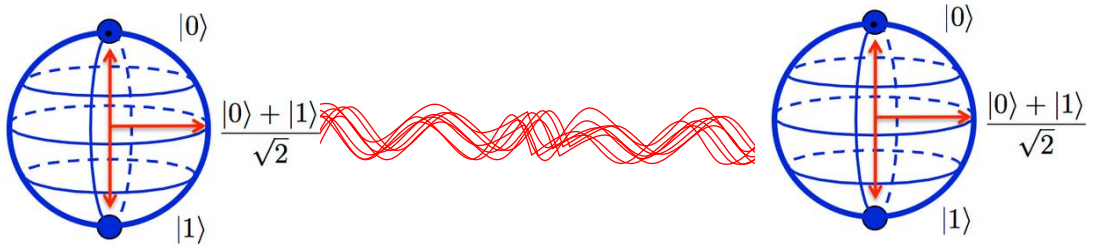
양자 컴퓨터란?

- **양자 중첩**
  - 측정 되기 전까지 가능한 모든 상태가 확률적으로 중첩 되어있음.
  - 측정 시에 하나의 상태로 결정



Qubit

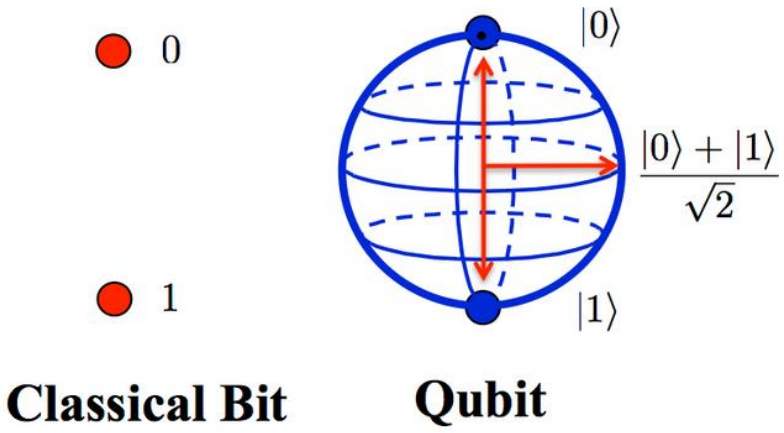
- **양자 얽힘**
  - 과거 상호작용했던 입자들을 멀리 떨어진 상태에도 연결된 관계 유지
  - 하나의 양자 상태를 결정하면 다른 하나의 양자 상태 또한 동시에 결정
  - 거리에 무관하게 발생



Qubit Qubit

양자 컴퓨터와 고전 컴퓨터

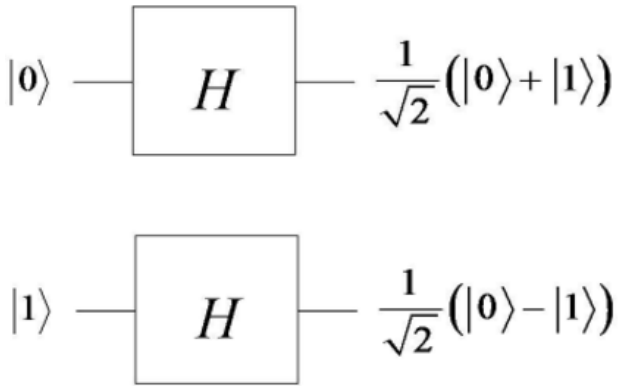
- **고전 컴퓨터와 비교**



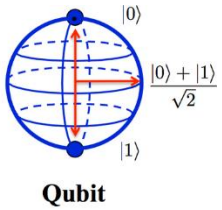
	고전 컴퓨터	양자 컴퓨터
연산 개념	0 1 정보를 0이나 1로 표현 0 1 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1개씩 순차적으로 계산 많은 시간 소요	0 1 0과 1을 중첩 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0 0 합쳐서 한번에 계산 순식간에 계산
상태	0 또는 1	0과 1 공존, 0과 1 사이의 확률 상태로 존재
단위	Bit	Qubit(quantum+ bit)

양자 게이트

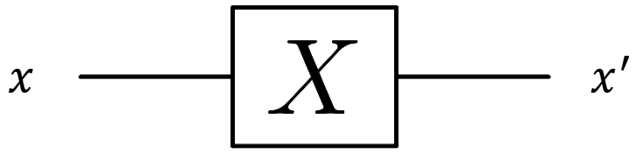
- H gate(Hadamard gate)



qubit을 중첩 상태로 만든다.



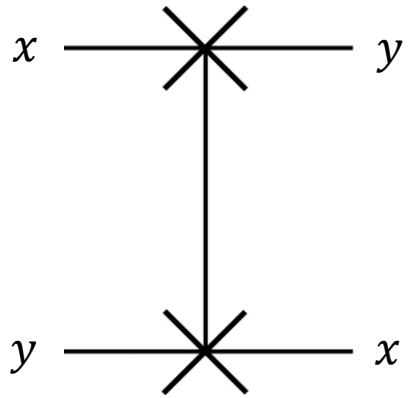
- X gate (NOT gate)



대상 qubit 값 반전

Ex) 0 -> 1
1 -> 0

- Swap gate

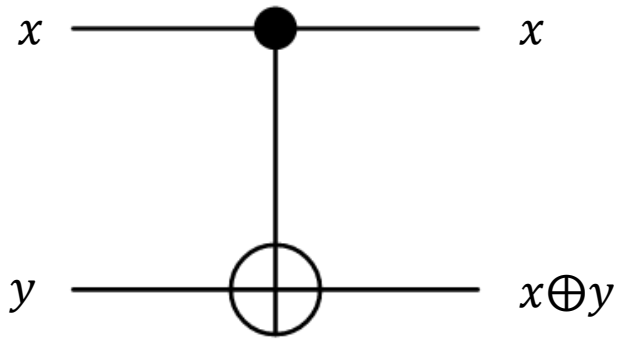


두 개의 qubit 값 교환

ex) temp = x
x= y
y= temp

양자 게이트

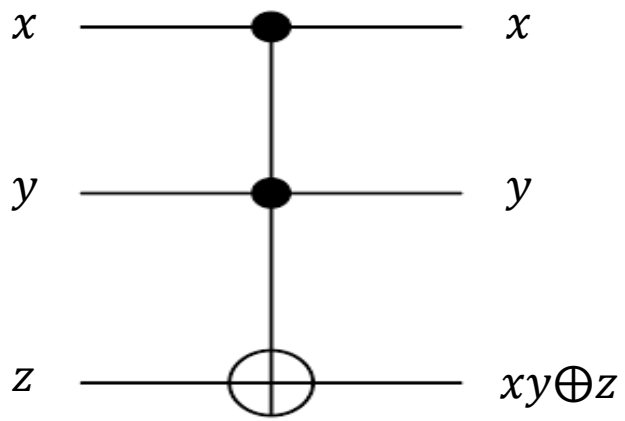
- CNOT gate



1 개의 qubit -> control qubit
Control qubit가 1일 때 대상 qubit 값 반전

ex) x – control qubit, y – 대상 qubit
x==1 -> y 값 반전

- Toffoli gate(CCNOT gate)



두 개의 qubit -> control qubit
Control qubit 두 개 모두 1일때만 대상 qubit 값 반전

ex) x, y – control qubit, z- 대상 qubit
x==1 AND y==1 -> z 값 반전

실습

```python
from projectq import MainEngine
from projectq.backends import ClassicalSimulator, ResourceCounter
from projectq.ops import H, CNOT, Measure, Toffoli, X, All, Swap


def main(eng):
    a = eng.allocate_qubit()        ← Qubit 선언
    b = eng.allocate_qureg(5)       ← 5개의 큐비트 배열
    c = eng.allocate_qubit()

    #X gate
    X | a

    #CNOT gate
    CNOT | (a,b[1])          ← 앞의 qubit이(a) control qubit

    #Toffoli gate
    Toffoli | (a,b[1],c)     ← 앞의 두 qubit이(a,b[1]) control qubit

    #Swap gate
    Swap | (a,c)
```

```python
Resource = ClassicalSimulator()
eng = MainEngine(Resource)
main(eng)
print(Resource)
```

```python
Resource = ResourceCounter()
eng = MainEngine(Resource)
main(eng)
print(Resource)
```

사용한 양자 자원 확인

```
Gate class counts:
    AllocateQubitGate : 2
    DeallocateQubitGate : 2
    MeasureGate : 4
    SwapGate : 2
    XGate : 1

Gate counts:
    Allocate : 2
    Deallocate : 2
    Measure : 4
    Swap : 2
    X : 1
```

실습

- X gate

```
def main(eng):
    a = eng.allocate_qubit()
    b = eng.allocate_qureg(5)
    c = eng.allocate_qubit()

    All(Measure) | a
    print(int(a))

    #X gate
    X | a

    All(Measure) | a
    print('a-> X gate: ', int(a))
```

```
/Users/yujin/PycharmProjects/Ascon/quatu
a-> X gate: 0
a-> X gate: 1
<projectq.backends._sim._classical_simul
```

NOT 연산과 동일

- CNOT gate

```
def main(eng):
    a = eng.allocate_qubit()
    b = eng.allocate_qureg(5)
    c = eng.allocate_qubit()

    CNOT | (a,b[1])

    All(Measure) | a
    All(Measure) | b

    print('if a ==0 ,CNOT(a,b[1]):',int(b[1]))

    #X gate
    X | a

    # a==1

    #CNOT gate
    CNOT | (a,b[1])

    All(Measure) | a
    All(Measure) | b
    print('if a ==1 , CNOT(a,b[1]):', int(b[1]))
```

```
/Users/yujin/PycharmProjects/Ascon/quatumTutor
if a ==0 ,CNOT(a,b[1]): 0
if a ==1 , CNOT(a,b[1]): 1
<projectq.backends._sim._classical_simulator.C
```

XOR 연산과 동일

실습

- Toffoli gate

```
def main(eng):
    a = eng.allocate_qubit()
    b = eng.allocate_qureg(5)
    c = eng.allocate_qubit()

    Toffoli | (a, b[1], c)
    All(Measure) | a
    All(Measure) | b
    All(Measure) | c

    print('if a==0 and b[1]==0, Toffoli(a,b[1],c) : ', int(c))

    #X gate
    X | a
    Toffoli | (a, b[1], c)
    All(Measure) | a
    All(Measure) | b
    All(Measure) | c
    print('if a==1 and b[1]==0, Toffoli(a,b[1],c) : ', int(c))

    X | b[1]

    Toffoli | (a, b[1], c)
    All(Measure) | a
    All(Measure) | b
    All(Measure) | c
    print('if a==1 and b[1]==1, Toffoli(a,b[1],c) : ', int(c))
```

```
/Users/yujin/PycharmProjects/Ascon/quatumTutorial/bin/
if a==0 and b[1]==0, Toffoli(a,b[1],c) :  0
if a==1 and b[1]==0, Toffoli(a,b[1],c) :  0
if a==1 and b[1]==1, Toffoli(a,b[1],c) :  1
<projectq.backends._sim._classical_simulator.Classical
```

AND 연산과 동일

- Swap gate

```
def main(eng):
    a = eng.allocate_qubit()
    b = eng.allocate_qureg(5)
    c = eng.allocate_qubit()

    X | a

    All(Measure) | a
    All(Measure) | c

    print('a : ', int(a))
    print('c : ', int(c))

    # Swap gate
    Swap | (a, c)

    All(Measure) | a
    All(Measure) | c

    print('Swap ->')
    print('a : ', int(a))
    print('c : ', int(c))
```

```
/Users/yujin/PycharmProjects/Ascon/quatumTutorial/
a :  1
c :  0
Swap ->
a :  0
c :  1
```

마무리

- 그 외 문법

```
with Compute(eng):
    CNOT(a, b[1])
    X | (c)
Uncompute(eng)      ← Compute 안의 내용을 Reverse(역연산)

with Control(eng, a):  ← 양자 얽힘 상태로 만듦
    CNOT(b[1], c)          (if 문 대신 사용)
```

Compute/Uncompute를 통해 역연산
→ 큐비트 수를 효율적으로 사용할 수 있음

Control를 통해 양자 얽힘 상태로 만듦
→ a가 1일 때만 Control 내의 코드 수행

큐비트는 중첩된 상태임으로 if문 사용불가(특정 상태 결정 지을 수 없음)
→ if문 대신 Control 사용

- 양자 최적화
  - qubit 수 줄임
  - Toffoli depth를 줄임
  - Full depth를 줄임

Qubit 수와 depth 는 trade-off 관계
-> depth와 qubit 모두 중요하므로 적절하게 구현 해야함

Q & A