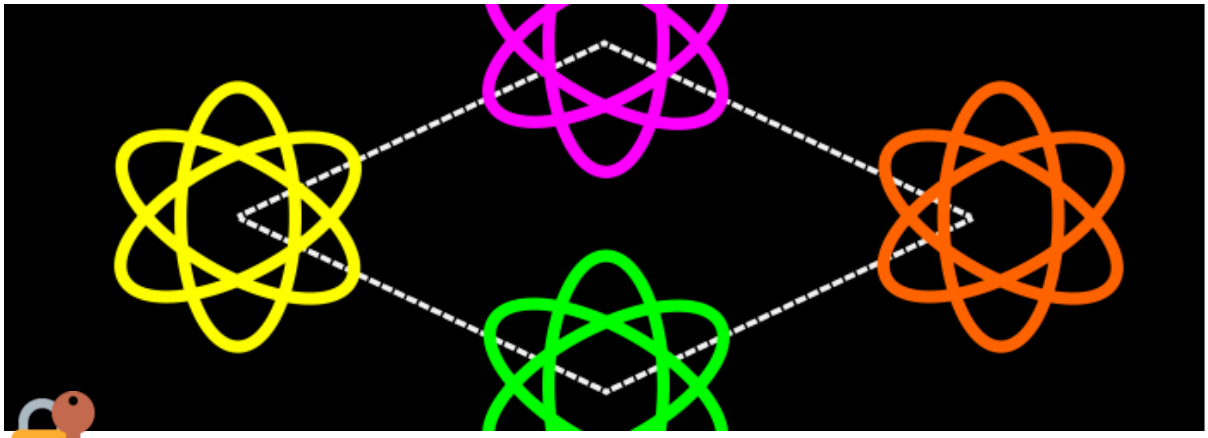


Kuantum Programlama

Ahmet Kasım Erbay —16.07.2021

Kuantum Bilgi Teknolojilerinin konuşulmaya başlandığı günümüz dünyasında bilginin güvenliği üzerine yeniden düşünüyor ve değişen bakış açılarıyla sistemlerimizi bilginin korunması için yeniden yapılandırılmayı konuşuyoruz. Klasik fizik ve matematik üzerine inşa edilmiş günümüz bilgisayarları artık yerlerini kuantum dünyasının, parçacık altı fiziği ve karmaşık matematiksel yapıların birleşimiyle inşa edilmiş kuantum bilgisayarlara bırakmaktadır. Parçacık altı fiziği dendiğinde Newton fiziğinin kurallarının geçerli olmadığı farklı bir dünyaya giriş yapmış oluyoruz. Bu dünyayla deterministik olaylardan ziyade olasılıksal durumların da hesaba katıldığı 1 ve 0 dan farklı bir evrene giriş yapmaktayız. Matematiğin reel sayılar uzayından çıkıp kompleks uzay içerisinde top koşturduğu bir zamanda diğer bütün matematik kuramlarımız ve sistemlerimiz gibi şifreleme, bilgi güvenliği ve haberleşmelerimiz de bu değişime ayak uydurmaktadır. Reel dünyanın klasikliğinden çıkıp modern çağların kuantum düzenine giriş yapıyoruz. Bu çalışmada kuantum şifreleme hesaplamalarını mümkün kılan matematik, fizik ve algoritma alt yapılarından bahsedilecektir. Qiskit kütüphanesi kullanarak kuantum devrelerinin ve kapılarının simülasyonlarını gerçekleştirelim. Başlamadan önce Anaconda ortamının kurulması gerekmektedir.

İndirmeler İçin Kaynaklar

İlk olarak aşağıdaki kod parçacığını Jupyter Notebook üzerinden çalıştırarak Qiskit'in yüklü olup olmadığını kontrol edebilirsiniz.

```
import qiskit

versyonlar = qiskit.__qiskit_version__
print("Qiskit versyonu: ",versyonlar['qiskit'])
print()
print("Qiskit Modüllerinin Versyonları:")
for i in versyonlar:
    print(i,"->",versyonlar[i])
```

Çıktı olarak aşağıdaki gibi bir sonuç almak programlarımızı çalıştırmak için yeterli olacaktır.

```
In [1]: import qiskit

        versyonlar = qiskit.__qiskit_version__
        print("Qiskit versyonu: ",versyonlar['qiskit'])
        print()
        print("Qiskit Modüllerinin Versyonları:")
        for i in versyonlar:
            print(i,"->",versyonlar[i])

        Qiskit versyonu:  0.27.0

        Qiskit Modüllerinin Versyonları:
        qiskit-terra -> 0.17.4
        qiskit-aer -> 0.8.2
        qiskit-ignis -> 0.6.0
        qiskit-ibmq-provider -> 0.14.0
        qiskit-aqua -> 0.9.2
        qiskit -> 0.27.0
        qiskit-nature -> None
        qiskit-finance -> None
        qiskit-optimization -> None
        qiskit-machine-learning -> None
```

Yukarıdaki sonuç alınamaması halinde buradaki kaynaklar takip edilerek indirlebilir;

- GitHub Qiskit İndirme Sayfası

Alternatif olarak;

1. İlk olarak Anaconda oratamının kurulması,
2. Başlat menüsünden "**Anaconda Prompt**" yazarak `pip install qiskit` komutu çalıştırılabilir.
3. Görselleştirmeler açısından "**Anaconda Prompt**" → `jupyter notebook` komutunu çalıştırılıp Jupyter Notebookta çalışmak daha verimli olacaktır.
4. İndirmeler tamamlandıktan sonra yukarıdaki kod parçacığı tekrardan denenebilir.

Klasik Sistemde Bilgisayarlar

Bit Operatörleri

Bildiğimiz üzere günümüz bilgisayarları saymada kullandığımız onluk taban yerine ikilik taban üzerine kurulmuştur. Bu sisteme göre bilgisayarlarda sadece iki rakam bulunmaktadır. 1 ve 0... Bu şekilde tasarlanmış sistemlere **bit sistemleri** denmektedir. Bir ve sıfır bu sistemin birimlerini ifade eder.

Matematik bize bu birimler üzerinde işlem yapmamızı sağlayan operatörler sunmaktadır. Bunlar;

- Birim Operatörü: $I(0) = 0\,, I(1) = 1$
- Değil Operatörü: $NOT(0) = 1\,, NOT(1) = 0$
- ZERO(Sıfır): $ZERO(0) = 0\,, ZERO(1) = 0$
- ONE(1): $ONE(0) = 1\,, ONE(1) = 1$

Aşağıda bu operatörlerin mantık tabloları da gösterilmiştir;

$$NOT = \begin{array}{c|cc} & \mathbf{0} & \mathbf{1} \\ \hline \mathbf{0} & 0 & 1 \\ \mathbf{1} & 1 & 0 \end{array} \qquad ZERO = \begin{array}{c|cc} & \mathbf{0} & \mathbf{1} \\ \hline \mathbf{0} & 1 & 1 \\ \mathbf{1} & 0 & 0 \end{array} \qquad ONE = \begin{array}{c|cc} & \mathbf{0} & \mathbf{1} \\ \hline \mathbf{0} & 0 & 0 \\ \mathbf{1} & 1 & 1 \end{array}.$$

NOT ve $Identity(I)$ operatörlerinin tersinir; diğer operatörlerin ise tersinir olmadığı yani tersinin alınamadığına dikkat etmek gereklidir. Mesela NOT operatörünün çıktısı 1 ise işleme giren bitin 0 olduğunu rahatlıkla söyleyebiliriz.

Bu sınıflandırma önemlidir çünkü kuantum operatörleri tersinirdir. Klasik sistemde kullandığımız tersinir operatörler ile kuantum sistemlerde de karşılaşıyoruz.

Şimdi klasik bilgisayarlardaki bit sistemini hilesiz bir paranın ön ve arka yüzü olarak düşünüp simülasyonlar gerçekleştirelim.

Aşağıdaki örnekte para 100, 1000 ve 10000 kere atılarak sonuçlar gözlenmiştir.

```
from random import randrange

sum0 = sum1 = 0

N = [100,1000,10000] # Paranın kaç kere atıldığı

for j in N:
    for i in range(j):

        coin_flap = randrange(2) # Paranın atıldığı
 kısım
        if(coin_flap == 0):
            sum0 +=1
        elif(coin_flap == 1):
            sum1 +=1

    print(f"\n{j} Deneme") # Kaç deneme olduğu
    print(f"{sum1} tane Y \n{sum0} tane T")
    print(sum1/sum0, end ="\n") # Yazı ve Turanın or
anı
```

```
                    # ÇIKTI BÖLÜMÜ
100 Deneme:

55 tane Y
45 tane T
1.2222222222222223
----------------------
1000 Deneme:

548 tane Y
552 tane T
0.9927536231884058
----------------------
10000 Deneme:
5579 tane Y
5521 tane T
1.0105053432349211
----------------------
```

Örneklerden de anlaşıldığı gibi bir ve sıfır dengeli olarak dağılmıştır. Yani yazı ve tura gelme olasılıkları deneme sayısı arttıkça 0.5' e yaklaşmaktadır. Bunun yanında elimizde sadece 0 ve 1 ihtimalleri bulunmaktadır. bunlar dışında herhangi bir durumla karşılaşmıyoruz.

Şimdi de bir hileli zar deneyi gerçekleştirip sonuçları inceleyelim. Bu zarda hile oranı $\%60$ tır.

```
import random

sum0 = sum1 = 0
N = [100,1000,10000,100000]

for j in N:
    for i in range(j):
        coin_flap = random.randrange(100)
        if coin_flap >= 60: # 60 sayısı paranın hile
oranını gösterir.
            sum0 +=1
        else:
            sum1 +=1

    print(f"\n{j} Deneme")
    print(f"{sum1} tane Y \n{sum0} tane T")
    print(sum1/sum0, end ="\n")
```

```
100 Deneme:

57 tane Y
43 tane T
1.3255813953488371
------------------
1000 Deneme:

682 tane Y
418 tane T
1.631578947368421
------------------
10000 Deneme:
6608 tane Y
4492 tane T
1.471059661620659
------------------
```

Buradaki gibi bir hileli para 0 ve 1 birimlerinin olasılıksal olarak birbirine eşit olmayan bit değerlerini ifade eder. Yani bilgisayarların okuyabildiği bitleri rastegele ürettiğimiz bu durumda bir değer diğerinden daha fazla sıklıkta görülecektir.

Olasılıksal Bitler

Şimdi biraz matematik katarak hileli bir paranın olasılıksal durumlarını vektörler ile gösterelim. İleride bu vektörleri **olasılıksal bit**ler olarak tanımlayacağız.

$$\begin{pmatrix} Pr(0) \\ Pr(1) \end{pmatrix}$$

Burada $Pr(0),$ paranın 0 (yazı) gelme olasılığını ifade eder. Buna ek olarak $Pr(0) + Pr(1) = 1$ eşitliği sağlanmaktadır. Yukarıdaki gibi bir gösterim olasılıksal bitlerin vektörel gösterimi olarak adlndırılmaktadır. Bu gösterimde 0 ve 1 bitleri sırasıyla $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ ve $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ olasılıksal bitlere denk gelmektedir.

Bu gösterimi pekiştirmek için bir de hileli bir zar atalım. Zarın yüzlerindeki hile aşağıda sıralandığı gibi olsun. Elde edeceğimiz vektör zar için olasılık vektörü olacaktır.

$$Pr(1) : Pr(2) : Pr(3) : Pr(4) : Pr(5) : Pr(6) = 1 : 4 : 7 : 2 : 8 : 3.$$

Hileli bir zardaki yüzlerin olasılıksal dağılımları

```
zar = [1,4,7,2,8,3]
toplam = 0

for i in zar:
    toplam+=i

for i in range(len(zar)):
    zar[i] = zar[i]/toplam

print(zar)
```

```
# Çıktı Bölümü
[0.04, 0.16, 0.28, 0.08, 0.32, 0.12]
```

Bu örneklerden de yola çıkarak vektörümüz $[x_1, x_2, ...x_n]$ olmak üzere, $\sum_{0<i<n} x_i = 1$ eşitliğini sağlayan vektörlere **olasılık vektörleri** denmektedir.

2 Paranın Atıldığı Bir Oyun

Dyelim ki elimizde 2 tane para olsun. Birinci para 0.6 olasılıkla yazı, 0.4 olasılıkla tura gelmektedir. İkinci para ise 0.3 olasılıkla yazı, 0.7 olasılıkla tura gelmektedir. Oyunumuzu ise şu kurala göre oynayalım;

- Başlangıçta iki para da yazı üstte olacak şekilde yerleştirilsin.
- Eğer ilk para yazı gelirse birinci para tekrar atılıyor.
- Eğer ilk para tura gelirse ikinci para atıyor.

Bu oyundaki tüm durumları matrix kullanarak göstermemiz mümkündür.

$$Paralar = \begin{array}{c|cc} \text{Son} \backslash \text{İlk} & \textbf{Yazı} & \textbf{Tura} \\ \hline \textbf{Yazı} & 0.6 & 0.3 \\ \textbf{Tura} & 0.4 & 0.7 \end{array} = \begin{array}{c|cc} \text{Son} \backslash \text{İlk} & \textbf{0} & \textbf{1} \\ \hline \textbf{0} & 0.6 & 0.3 \\ \textbf{1} & 0.4 & 0.7 \end{array}$$

İki kez para attıksan sonra yazı ve tura gelme olasılıklarını python kullanrak şöyle gösterebiliriz;

```
# Başlangıç Durumu
# İlk durumda paralar yazı tarafı üstte kalacak şekilde durmakta.
# Bu sebepten yazı gelme olasılığı 1, tura gelme olasılığı 0.
prob_yazi = 1
prob_tura = 0
#
# Birinci parayı atıyoruz
#
# yazı gelme olasılığı matrixin ilk satırı ile hesaplanıyor
yeni_prob_yazi = prob_yazi * 0.6 + prob_tura * 0.3
# tura gelme olasılığı matrixin ikinci satırı ile hesaplanıyor
yeni_prob_tura = prob_yazi * 0.4 + prob_tura * 0.7
# olasılıkların güncellenmesi
prob_yazi = yeni_prob_yazi #0.6
prob_tura = yeni_prob_tura #0.4
#
# İkinci parayı atıyoruz
#
# Aynı hesaplama tekrar yapılıyor
yeni_prob_yazi = prob_yazi * 0.6 + prob_tura * 0.3
yeni_prob_tura = prob_yazi * 0.4 + prob_tura * 0.7

prob_yazi = yeni_prob_yazi
prob_tura = yeni_prob_tura

# Olasılık durumları
print("Yazı gelme olasılığı: ",round(prob_yazi,6))
print("Tura gelme olasılığı: ",round(prob_tura,6))
----------------------------------------------------------------
# ÇIKTI

Yazı gelme olasılığı:  0.48
Tura gelme olasılığı:  0.52
```

Aynı oyunu aynı protokoller ile matematiksel olarak da gösterebiliriz;

- Bizim olasılıksal matriximiz: $\begin{pmatrix} 0.6 & 0.3 \\ 0.4 & 0.7 \end{pmatrix}$
- Başlangıç koşulumuz: $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Yani yazı ile başladık.

Bu şekilde ilk para atımından sonra $\begin{pmatrix} 0.6 & 0.3 \\ 0.4 & 0.7 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix}$ tür.

İkinci adımda ise $\begin{pmatrix} 0.6 & 0.3 \\ 0.4 & 0.7 \end{pmatrix} \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} = \begin{pmatrix} 0.36 + 0.12 \\ 0.24 + 0.28 \end{pmatrix} = \begin{pmatrix} 0.48 \\ 0.52 \end{pmatrix}$ sonucunu elde ediyoruz.

Bu bölümde olasılıksal bitler ile nasıl çalışıldığını öğrenmiş olduk. Matrix gösterimi sayesinde operatörlerin kullanımını kolaylaştırmış olduk. Peki iki bitlik işlemlerde nasıl gösterimler uygulamalıyız?

İki Bitlik Olasılıksal Sistemler

Klasik bilgisayarlarda iki bit durumu 0 ve 1 olarak gösteriliyordu. Olasılıksal olarak ise bitleri $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ ve $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ olarak gösteriyoruz. İki bitlik sistemlere geçtiğimiz zaman $00, 01, 10, 11$ kompozit durumları ortaya çıkmaktadır. Olasılıksal vektörlerine ulaşmak için ise tensor çarpım[1] kullanırız.

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \otimes \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} \\ b_1 \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_1 \cdot a_2 \\ a_1 \cdot b_2 \\ b_1 \cdot a_2 \\ b_1 \cdot b_2 \end{pmatrix}.$$

Bu çarpıma göre 01 bitinin denk geldiği vektör $\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}$ vektörüne denk gelmektedir. 2 bitli sistemlerde diğer vektörler numpy kullanılarak da gösterebiliriz.

```
import numpy as np

sıfır = np.array([1.0,0.0])
bir = np.array([0.0,1.0])

print(np.tensordot(sıfır,sıfır,axes=0),end="\n\n")
print(np.tensordot(sıfır,bir,axes=0),end="\n\n")
print(np.tensordot(bir,sıfır,axes=0),end="\n\n")
print(np.tensordot(bir,bir,axes=0),end="\n\n")
```

Tensor çarpım kullanarak Qbitlerin nasıl oluşturulduğunu aşağıdan inceleyebiliriz.

Bir Boyutlu QBitler

- $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$
- $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$

İki Boyutlu Qbitler

- $|00\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$
- $|01\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}$
- $|10\rangle = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}$
- $|11\rangle = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$

Üç Boyutlu Qbitler

- $|000\rangle = |00\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}$
- $|100\rangle = |10\rangle \otimes |0\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{pmatrix}$

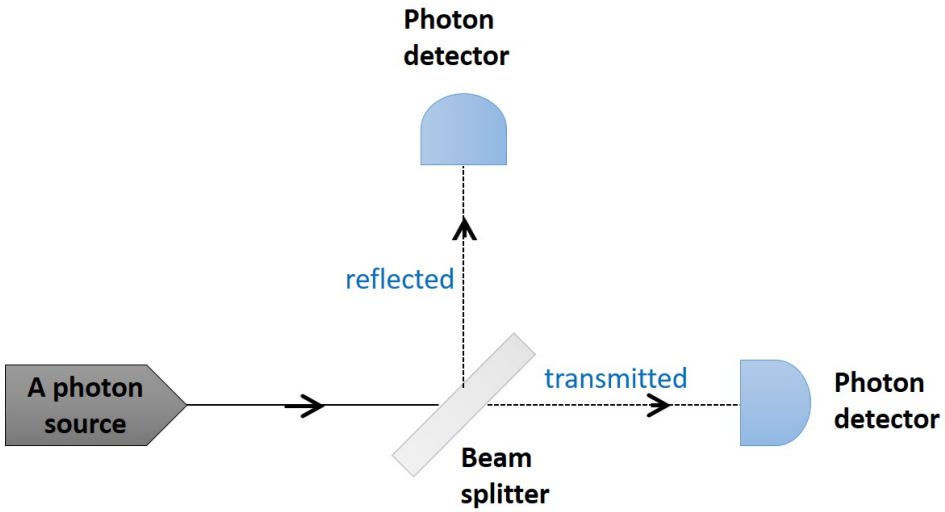
- $|010\rangle = |01\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
- $|110\rangle = |11\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
- $|001\rangle = |00\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
- $|101\rangle = |10\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$
- $|011\rangle = |01\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$
- $|111\rangle = |11\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$

Kuantum Sistemlere Giriş

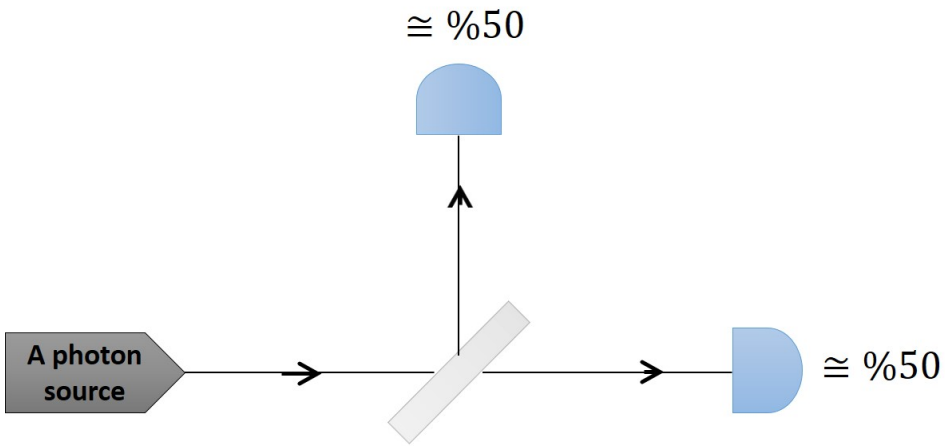
Bu bölüme kadar yaptığımız işlemler ve kullandığımız matematik aslında Newton ve sonrası klasik matematiği içermektedir. 0 ve 1 ler dünyasından biraz daha olasılıkların olduğu veya eşit olasılıkta olmadıkları durumları incelemeye çalıştık. Kuantum sistemlerde bilgiler, klasik sistemlerde olduğu gibi elektrik gerilimi ile aktarılan sinyaller değil foton denilen en küçük enerji paketçikleri ile taşınır. Dolayısıyla fotonların tabi olduğu fizik kurallarına uymak zorunda kalıyoruz. Maddelerin kütleleriyle çalışıldığı fizikten dalga fiziğine ve etkilerine maruz kalıyoruz. Bu gariplikleri bir dizi deney ile açıklamaya çalışalım.

Deneyimizde,

- Fotonlar (photon), yukarıdaki deneylerdeki havaya attığımız para,
- Yansıtıcı zar (beam splitter), paranın iki yüzünü temsil ediyor. Bu zar yarı yarıya geçirgen bir zardır.
- Foton yakalayıcılar (photon detectors) ise gözlem yapan kişileri temsil ediyor.

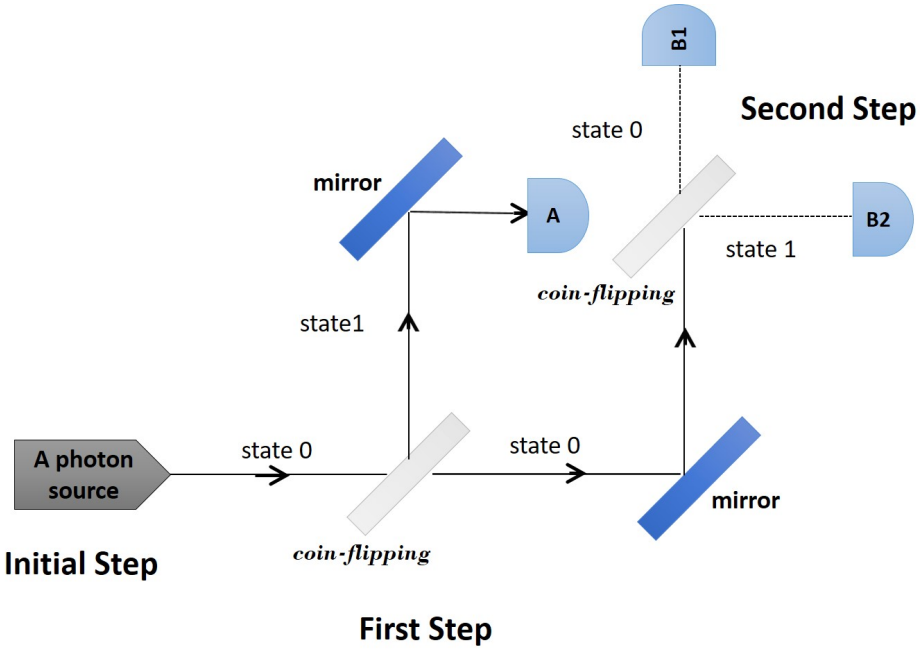


İlk olarak yukarıda gösterildiği gibi bir fotonu bir zardan geçirelim. Normal olarak foton, $\%50$ ihtimal ile her iki alıcıya ulaşmasını beklemekteyiz.



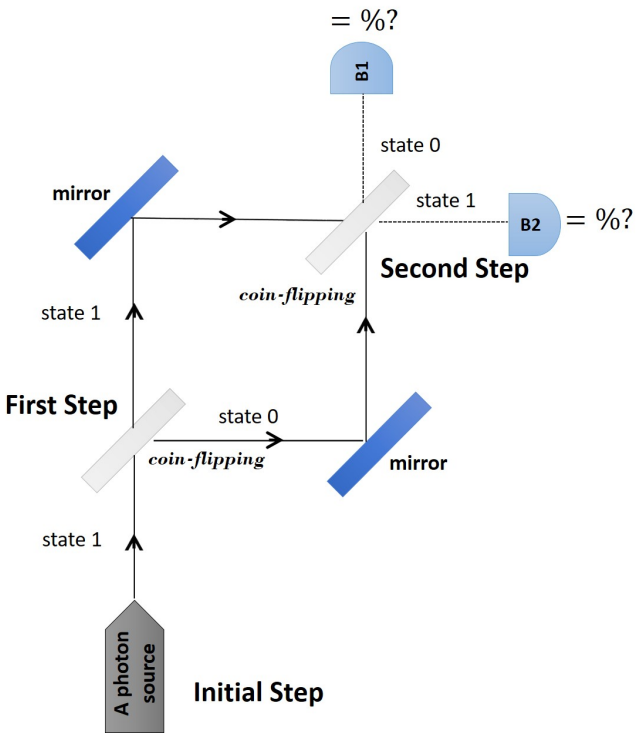
1. DENEY

Deneysel sonuçlar beklentiler ile aynı olarak gerçekleşiyor. Yani yapılan tekrarlı deneylerin yarısında bir kapıda diğer yarısında diğer kapıda foton gözlemleniyor. Şimdi deneyimize bazı eklentiler yaparak genişletelim. Aşağıdaki düzenekte $\%50$ ihtimalle A kapısına, $\%25$'er ihtimalle $B1$ - $B2$ kapılarına çıkmayı beklemekteyiz. Ve deneylerde de gerçekten bu sonuca ulaşmaktayız.



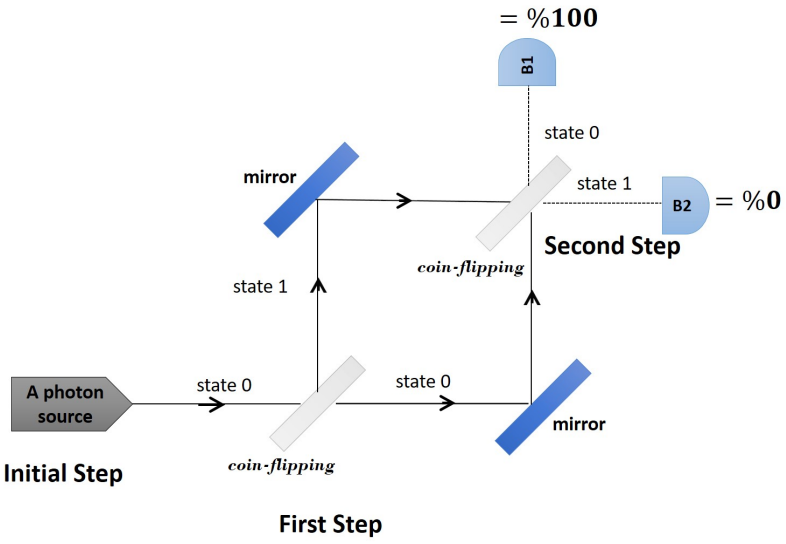
2.DENEY

3. deneyimizde klasik matematiğimizin burada çalışmadığını gözleyeceğiz. Bu deneyde beklentimiz $\%50$ ihtimaller ile iki kapıda da fotonu görmek yönünde.



3.DENEY

Fakat yapılan gerçek deneylerde şaşırtıcı sonuçlar gözleniyor. Ve **tüm** tekrarlı deneylerde fotonun $B1$ kapısına çarptığı gözleniyor.



Bu deney klasik fiziğin parçacığın hareketlerini açıklamakta yetersiz kaldığını göstermektedir. Bu noktada yeni bir matematiksel model geliştirmek gerekmektedir.

QBitler

Qbitler, kuantum bilgisayarlarda bilgi taşınan birimleri ifade eder[2]. Klasik bilgisayarlardaki gibi 0 ve 1 lere denk olarak $|0\rangle$ ve $|1\rangle$ Qbitleri kullanılır. Dikkat edilmesi gereken nokta klasik bilgisayarlarda bir bit 0 veya 1 olabilirken kuantum düzeninde $|0\rangle$ ve $|1\rangle$ veya bunların lineer kombinasyonlarını alabilir. Qbitlerin genel denklemi; $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle; \alpha, \beta \in \mathbb{C}$ ve $|\alpha|^2 + |\beta|^2 = 1$ şeklinde verilir. Bu kurallar dikkate alınarak bir Qbitin içinde bulunabileceği olasılıksal durumları kodlayabiliriz.

```
from random import randrange

# Verilen bir vektörün olasılıksal olarak mümkünlüğünü test eden fonksyon
def is_quantum_state(quantum_state):
    l = quantum_state[0]**2+quantum_state[1]**2
    if((l-1)**2 < 0.000001):return 1
    else: return 0

# Random olarak Kuantum halde bulunan Qbitler oluşturan fonksyon
def random_quantum_state():
    quantum_state=[0,0]

    for i in range(2):
        a = randrange(101)
        quantum_state[i] = a
    length = 0

    for i in range(len(quantum_state)):
        length += quantum_state[i]**2
    length = length**0.5

    for i in range(len(quantum_state)):
        quantum_state[i] = quantum_state[i]/length

    b = randrange(2)
    if(b<1):
        quantum_state[b] = quantum_state[b]*(-1)
    return quantum_state

print(random_quantum_state())
```

Not: $|.\rangle$ notasyonu ket notasyonu olarak adlandırılır. Bu, kuantum mekaniğinde vektörler için kullanılır. $\langle .|$ ise ket vektörünün konjuge transposu olarak kullanılır.

Elektronların dönüş yönlerine göre yukarı ve aşağı şeklinde sınıflandırıldıklarını biliyoruz. Fotonlarında aynı şekilde polarize oluşlarına göre sınıflandırabiliyoruz. Bu da fotonlar üzerinde bilgiyi taşımak için kullanmamıza olanak sağlıyor. Fakat fotonların hangi açıda durduklarını gözlem yapana kadar tam olarak kestiremiyoruz.

Süperpozisyon

Klasik fizikte süperpozisyon olayının bir örneği olmadığından yeni bir konsepttir. Fotonların gözlem yapana kadar farklı pozisyonlarda durmaları ve olasılıksal olarak farklı değerler almalarından ötürü klasik bitlerden farklı bilgileri taşırken kullanılır. Süperpozisyonun bu belirsizliği aslında anahtar dağıtımında kullanılan BB84 algoritmasının temelini oluşturmaktadır.

Örnek olarak; eğer sistem $-\frac{\sqrt{2}}{\sqrt{3}}|0\rangle + \frac{1}{\sqrt{3}}|1\rangle = \begin{pmatrix} -\frac{\sqrt{2}}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}$ halinde ise bunun anlamı gözlem yaparsak $\frac{2}{3}$ olasılıkla $|0\rangle$, $\frac{1}{3}$ olasılıkla $|1\rangle$ Qbitleriyle karşılaşacağız demektir.

Gözlem yaptıktan sonra ise Qbitler $|0\rangle$ veya $|1\rangle$ olarak indirgeniyor ve süperpozisyondan çıkmış oluyoruz.

QBitlerin Görselleştirilmesi [3]

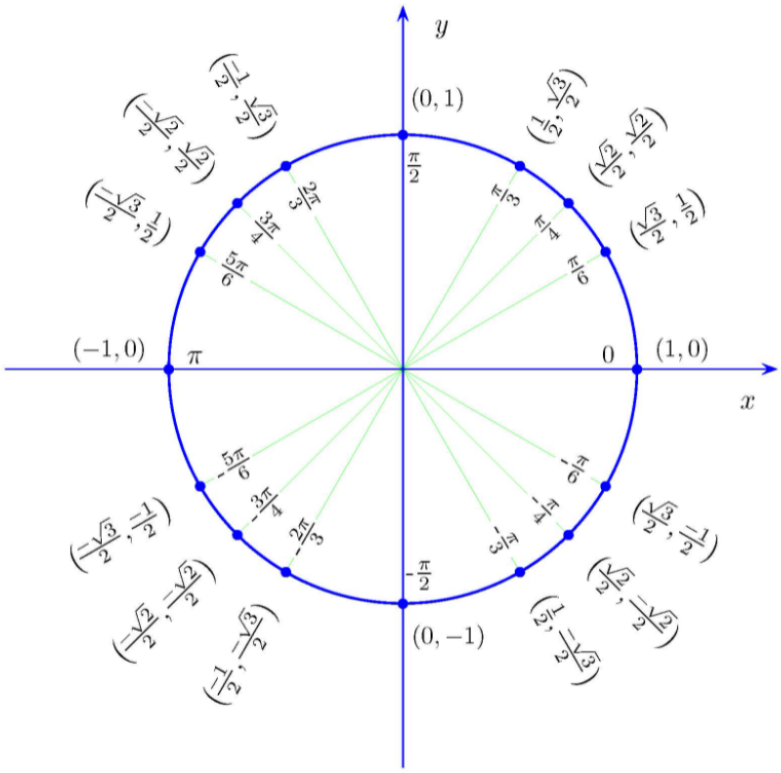
Bu bölümde Qbitlerin kompleks sayılarla bağlantılarını kullanarak görselleştirilmeleri üzerine çalışacağız. Bloch küresi kullanarak bu görselleştirmeyi yapmak kolaylaşıyor.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle\,;\ \alpha,\beta \in \mathbb{C}; |\alpha|^2 + |\beta|^2 = 1$$

Euler eşitliğini kullanarak yukarıdaki denklemi yeniden düzenleyebiliriz.

$\alpha = r_0.e^{i.\kappa_0}$ ve $\beta = r_1.e^{i.\kappa_1}$; r_i genlikleri, κ_i açıları ifade ediyor.

Böylelikle her bir c_i komplex uzayda birim çember üzerinde noktalara denk geliyor.



Bir Qbit gösteriminde iki tane c_i olduğundan 4 bilinmeyen ile uğraşmak zorunda kalıyoruz.

$$|\psi\rangle = r_0.e^{i.\kappa_0}|0\rangle + r_1.e^{i.\kappa_1}|1\rangle$$

Fakat QBit ler ile uğraştığımızdan birim uzunlukta bir norm ile çarpmak kuantum durumunu değiştirmediğinden aşağıdaki matematiksel işlemler ile Bloch küresi üzerinde Qbitleri canlandırabiliyoruz. Çünkü 4 değişkenli bir denklemden 3 değişkene inmiş oluyoruz. Bunun için;

$|\psi\rangle = e^{i.\gamma}|\psi\rangle, \gamma = -\kappa_0$ değeri ile çalışırsak;

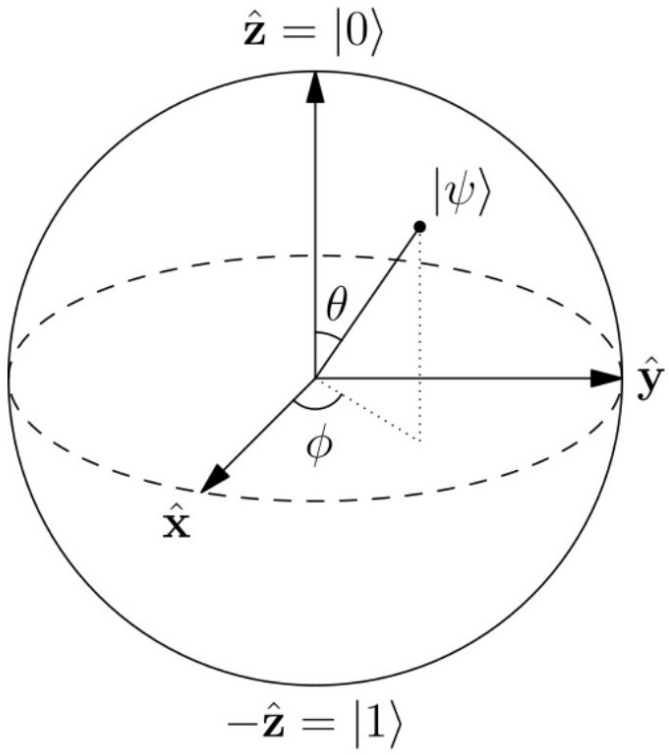
$$e^{i.-\kappa_0}|\psi\rangle = e^{i.-\kappa_0}(r_0.e^{i.\kappa_0}|0\rangle + r_1.e^{i.\kappa_1}|1\rangle) = r_0|0\rangle + r_1.e^{i.(\kappa_1-\kappa_0)}|1\rangle$$

Sadece r_0, r_1 ve $\phi = \kappa_1 - \kappa_0$ değişkenleri kaldı.

Son durumda ise;

$$|\psi\rangle = cos(\theta).|0\rangle + sin(\theta).e^{i.\phi}|1\rangle$$

formülünü elde ederiz. Bunun görselleştirmesi ise;



Bu gösterim ile aslında kuantum bilgisayarlarının bit sistemlerinin klasiklerden neden daha üstün olduğu görülebiliyor. Şekilden de görüldüğü üzere $z = |0\rangle$ ve $z = |1\rangle, \theta = 0$ olduğu durumlar için geçerli oluyor. Qbit, küre üzerinde diğer tüm kuantum durumlarında bulunabiliyor.

Kuantum Operatörleri

Kuantum durumlarının olasılıksal durumlarını vektörler aracılığı ile göstermiştik. Bunların matrix gösterimleri üzerinden mantık kapıları aracılığı ile manipüle edilebileceğini de tekrardan hatırlatalım. Klasik bilgisayarlardaki tersinir operatörler, kuantum operatörleri için de bir seçenek oluşturuyor. (Not ve $Identity$ ...)

Yukarıda gösterdiğimiz Bloch küresine tekrar dönecek olursak; küre üzerinde rotasyon ve yansıtma işlemlerini yapabilecek her türlü matrix, bu sayede, kuantum operatörü olarak sınıflandırılır. **Daha açık bir ifadeyle Kuantum vektörünün boyutunu değiştirmeyen her $n \times n$ matrix kuantum operatörüdür.**

- $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
- $X(NOT) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
- $CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
- $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
- $H(Hadamard) = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$

Hadamard Kapısı

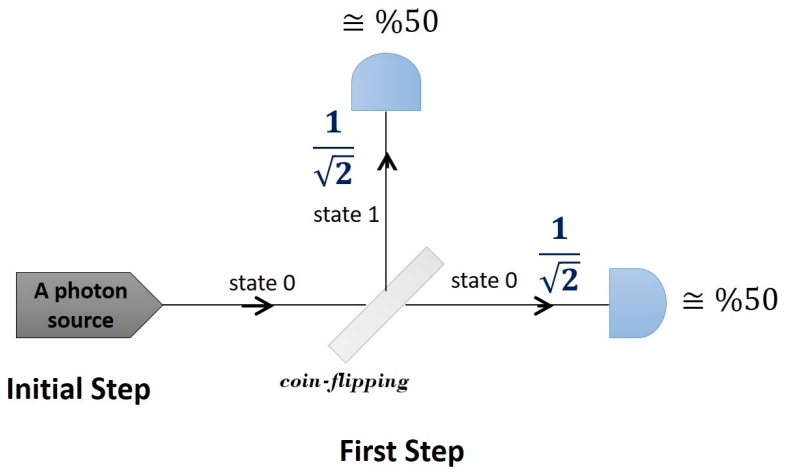
Hadamard kapısı Qubitler üzerinde para çevirme deneyi yapmamıza yarayan kapıdır ya da en azından böyle söyleyebiliriz. Şimdi birlikte QBitler üzerindeki etkisini inceleyelim.

$$H|0\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

Her iki Qbit üzerinde uygulandığında $\frac{1}{2}$ olasılıkla $|0\rangle$ ve $\frac{1}{2}$ olasılıkla $|1\rangle$ elde etmekteyiz. İşin ilginç tarafı tam olarak burada ortaya çıkıyor. Çünkü Hadamard kapısı QBitleri süperpozisyon durumuna sokuyor. Sadece $|0\rangle$ Qbiti Hadamard kapısından geçince gözlem yapılması halinde eşit olasılıklarla $|0\rangle$ ve $|1\rangle$ Qbitlerine indirgenebilir hale geliyor.

Eşit olasılıklardan bahsettiğimiz için de başta yaptığımız para atma deneylerine benzetebiliriz.



Yukarıda paylaştığımız `is_quantum_state(quantum_state)` ve `random_quantum_state()` fonksyonlarını kullanarak Qbitlere Hadamard kapısı uygulayalım.Aslında Hadamard kapısının Qubitler üzerinde geçerli bir operatör olduğunu görmüş olacağız.

```
hadamard = np.array([[1/sqrt(2),1/sqrt(2)],[1/sqrt
```

```
Qubit kontrolü: [ 0.91036648 -0.41380294]
```

```
(2),-1/sqrt(2)]])
qbits = []

for i in range(10):
    qbits.append(random_quantum_state())

for i in qbits:
    i = hadamard@i
    print(f"Qubit kontrolü: {i}")

for i in qbits:
    print(is_quantum_state(i))
```

```
Qubit kontrolü: [-0.09053575 -0.99589321]
Qubit kontrolü: [-0.62919823 -0.77724487]
Qubit kontrolü: [ 0.20521484 -0.97871695]
Qubit kontrolü: [ 0.18654062 -0.98244725]
Qubit kontrolü: [ 0.98320383 -0.18251089]
Qubit kontrolü: [ 0.05471245 -0.99850215]
Qubit kontrolü: [ 0.92996072 -0.36765889]
Qubit kontrolü: [0.98682767 0.16177503]
Qubit kontrolü: [-0.6974876  -0.71659685]
1
1...
```

Kuantum Dolaşıklık

2016 yılında Çin, Micius adlı uydu aracılığıyla yeryüzüne kurduğu iki istasyon arasında kuantum dolaşıklığı ve diğer sistemler kullanarak tamamen güvenli veri iletimi gerçekleştirdi[4]. Kuantum dolaşıklılığının matematiksel gösterimini Hadamard kapısını bir adım daha ilerleterek gözlemleyebiliriz. Yukarıda yaptığımız hesaplardan,

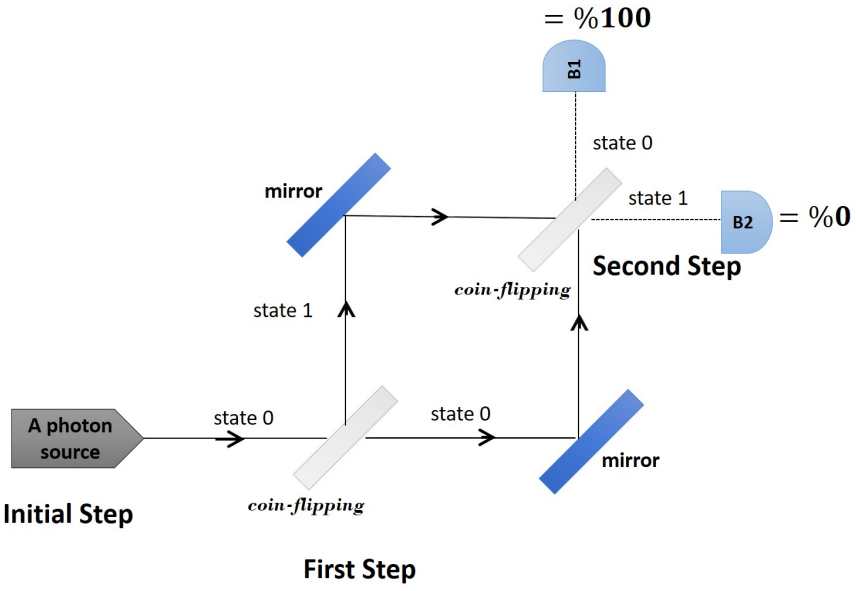
$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \qquad H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

çıktılarını elde etmiştik. İncelemek istediğimiz denklemler ise şu şekilde;

$$HH|0\rangle = H(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) = \frac{1}{\sqrt{2}}(H|0\rangle) + \frac{1}{\sqrt{2}}(H|1\rangle) = \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) + \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|0\rangle - \frac{1}{2}|1\rangle = |0\rangle$$

$$HH|1\rangle = H(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) = \frac{1}{\sqrt{2}}(H|0\rangle) - \frac{1}{\sqrt{2}}(H|1\rangle) = \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) - \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle - \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle = |1\rangle$$

Burada dönen hikaye şöyle ilerliyor; yukarıda Hadamard kapısının Qbitleri süperpozisyona soktuğunu gösterip geçerli bir kuantum kapısı olduğunu söyledik. Şimdi ise süperpozisyonda olan Qbitlere Hadamard uyguladığımızda nasıl indirgendiklerini görmüş olduk. Bu indirgeme durumu daha önceden sonucunu kavrayamadığımız deneylerden birisini matematiksel açıklaması olarak gösterilmektedir.



Deneyde birinci zardan geçtikten sonra fotonların her iki yönden gitme ihtimalleri bulunmaktaydı. "State-0,1" olarak gösterilen çizgiler işlmelerimizdeki $H\left|0\right\rangle$ ve $H\left|1\right\rangle$ bölümlerine denk gelmektedir. Süperpozisyon halindeki Qbitlerin $B1$ ve $B2$ foton algılayıcıları tarafından görülme imkanı bulunmaktadır. Fakat ikinci Hadamard işleminden sonra Qbitler indirgeniyor ve elimizde $\left|0\right\rangle$ veya $\left|1\right\rangle$ kalıyor.

Aşağıda bu deneylerin Qiskit kütüphanesi kullanılarak simülasyonları paylaşılmıştır. Şimdi yukarıda yaptığımız deneylerden 2 tanesini birlikte takrarlayalım. Qiskit kütüphanesi kullanılan bölümlerin çalıştırılması için bazı indirmeleri yapmamız gerekmektedir. Buradan o indirmelere ulaşabilirsiniz.

```
#1.DENEY
#Quantum devreler için metodları ve modülleri ekleyelim
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute, Aer

# 1 Qbit ile register oluşturlaım
qreg1 =  QuantumRegister(1)


# ölçümden sonra gözlem yapabilmek için bir de klasik register tanımlayalım
creg1 = ClassicalRegister(1)

# Kuantum devremizi oluşturalım
mycircuit1 = QuantumCircuit(qreg1,creg1)

# Hadamard kapısının uygulayalım
mycircuit1.h(qreg1[0])

# gözlem (ölçüm) yapıp soucu klasik bite kaydedelim
mycircuit1.measure(qreg1,creg1)

#devreyi çizdirelim
mycircuit1.draw(output='mpl')

# 10000 deneyden sonraki sonuçları yazdırıp inceleyelim
job = execute(mycircuit1,Aer.get_backend('qasm_simulator'),shots=10000)
counts1 = job.result().get_counts(mycircuit1)
print(counts1)
-------------------------------------------------------------------------
ÇIKTI: {'0': 4941, '1': 5059}
```

Gerçekten de yarı yarıya bir oranla foton ilk yarıktan geçiyor ve bu sonucu elde ediyoruz.

```
#3. DENEY
---------
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute, Aer

#Registerların tanımlanması, devrenin kurulması
qreg2 =  QuantumRegister(1)
creg2 = ClassicalRegister(1)
mycircuit2 = QuantumCircuit(qreg2,creg2)

# İki kere Hadamard kapısının uygulanması
mycircuit2.h(qreg2[0])
mycircuit2.h(qreg2[0])

#GÖzlem yapılması (foton algılayıcı)
mycircuit2.measure(qreg2,creg2)

# Deneyin 10000 kere tekrarlanması
job = execute(mycircuit2,Aer.get_backend('qasm_simulator'),shots=10000)
counts2 = job.result().get_counts(mycircuit2)
print(counts2)
-------------------------------------------------------------------------
#ÇIKTI:{'0': 10000}
```

Buradan da görüldüğü gibi tüm tekrarlardan sonra iki kere Hadamard kapısı uygulamak fotonların aynı algılayıcı üzerine düşmesine sebep oluyor.

2 Qbitlik Sistemler

Şimdiye kadar tek Qbitlik sistemlerde çalıştık. Bazı kapıların nasıl uygulandığını gösterdik ve ilk kuantum devremizi kurarak daha önceden ulaştığımız deney sonuçların kuantum registerlar üzerinde denedik. Bu bölümde iki Qbit üzerinde çalışacağız. Peki iki Qbit tek sistemde nasıl gösterilir?

Yukarıda tensor çarpımından bahsetmiştik. Tensor çarpım Hilbert uzaylarında tanımlı iki vektörü çarparak bir üst sistemi elde etmemizi sağlayan matematiksel operatördür. Bizim durumumuzda iki Qbitin tensör çarpımı iki Qbitlik sisteme ulaşmamızı sağlayacak olan işlemdir. Yani bundan sonra $|00\rangle$, $|01\rangle$, $|10\rangle$ ve $|11\rangle$ Qbitleri ile çalışabileceğiz. Vektörel olarak elde etmek için ise;

$\begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} = |10\rangle$ örneğini verebiliriz.

ControlNot Operatörü

KontrolNot operatörü iki bitlik sistemler üzerine uygulanabilen bir kapıdır. Eğer ilk bit 1 ise ikinci bite NOT operatörü uygular. Nasıl uygulandığı aşağıda gösterilmiştir.

- $CNOT\,|00\rangle = |00\rangle$
- $CNOT\,|01\rangle = |01\rangle$
- $CNOT\,|10\rangle = |11\rangle$
- $CNOT\,|11\rangle = |10\rangle$

Bu noktada kuantum dolaşıklığı tekrar kullanalım ve veri aktarımının nasıl gerçekleştiğini açıklamaya çalışalım. elimizdeki örnekte Ali ve Burak haberleşmeye çalışsın.

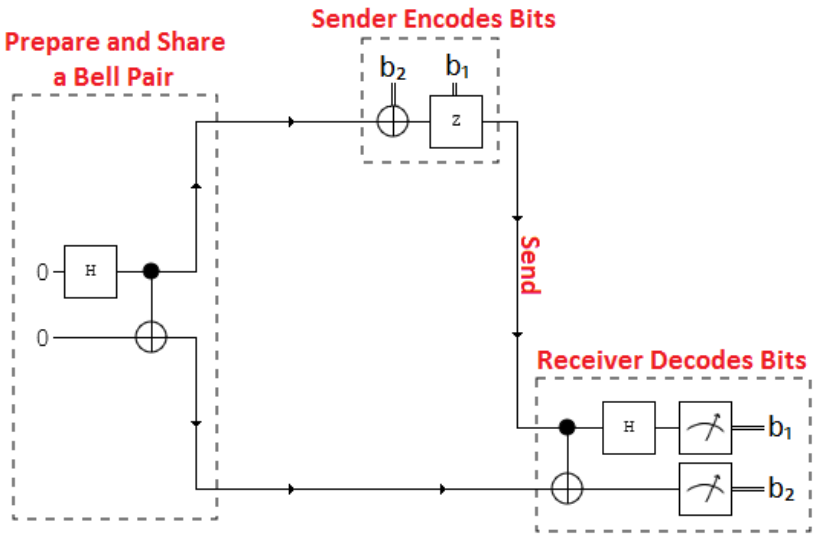
- Başlangıçta Ali ve Burak ellerinde $|0\rangle$ Qbitine sahip olsun.
- Ali kendi Qbitine Hadmard kapısı uygulasın bu durumda elinde $\begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix}$ halinde bir Qbit bulunacak.
- Sonra ikisi ellerindeki Qbitleri çarpım durumuna alsınlar. Yani tensor çarpımına alacaklar. Elde edeceğimiz kuantum durumu $\begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\0\\\frac{1}{\sqrt{2}}\\0\end{pmatrix}$ olacaktır.
- Bu halde bulunan Qbitlere $CNOT$ kapısı uygulayalım ve sonucu görelim: $\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}}\\0\\\frac{1}{\sqrt{2}}\\0\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{pmatrix} = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$

Tam olarak bu işlemler gerçekleştikten sonra Ali ve Burak'ın Qbitleri dolaşık hale geliyor. Yani iki Qbit birbirlerinden fiziksel olarak uzaklaşsalar bile ya ikisi de 0 ya da ikisi de 1 olarak gözlemleniyor.

İşte bu Qbitlerden birisi elinde başka bir Qbit bulunan birisine aktarıldığında veri iletimi gerçekleşmiş oluyor. Bu veri iletişiminin nasıl olduğuna geçebiliriz artık.

Superdense Kodlama

Süperdense kodlama, önceden dolaşık hale getirilmiş Qbitler kullanılarak veri iletiminin gerçekleştirilmesidir.



Protokol:

Elimizde yukarıda uygulandığı gibi dolaşık halde bulunan Qubitler olduğunu var sayalım. $|\phi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = \frac{1}{\sqrt{2}}\left(\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$

- Ali iki bitlik klasik (x, y) bilgisine sahip olsun: $(x, y) \in \{0, 1\}$. Yani 4 tane kombinasyon var: (x, y) : $(0, 0), (0, 1), (1, 0)$ veya $(1, 1)$.
- Eğer $x = 1$ ise, Ali elindeki Qbite Z kapısını uygular.
- Eğer $y = 1$ ise, Ali NOT kapısını uygular.

Not: Bu kurala göre eğer gönderdiği bilgi $(0, 0)$ ise I kapısı uygulanır.

- Sonunda da kendi Qbitini Buraka gönderir.

Elindeki Qbitlere, Burak sırasıyla $CNOT$ ve H kapılarını uygulayarak mesajı deşifre eder.

Mesela, Ali'nin göndermek istediği bilgi $(0, 1)$ olsun.

- O zaman elindeki dolaşık halde olan Qbitlere NOT kapısı uygular. Yani;

$$NOT = \begin{pmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}, |00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \implies NOT(\tfrac{1}{\sqrt{2}}|00\rangle + \tfrac{1}{\sqrt{2}}|11\rangle) \implies$$

$$\begin{pmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix} \cdot \tfrac{1}{\sqrt{2}}\left(\begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right) =$$

$$\tfrac{1}{\sqrt{2}}\begin{pmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix} \cdot \begin{pmatrix}1\\0\\0\\1\end{pmatrix} = \tfrac{1}{\sqrt{2}}\begin{pmatrix}0\\1\\1\\0\end{pmatrix} = \tfrac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$$

Bu noktada Ali, Qbiti Buraka iletir.

- Burak bu Qbitlere önce $CNOT$ uygular: $\frac{1}{\sqrt{2}}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\cdot\begin{pmatrix}0\\1\\1\\0\end{pmatrix}=\frac{1}{\sqrt{2}}\begin{pmatrix}0\\1\\0\\1\end{pmatrix}=\frac{1}{\sqrt{2}}(|01\rangle+|11\rangle)$ elde eder.
- Şimdi ise 1. Qbite Hadamard kapısı uygularsa

$\frac{1}{\sqrt{2}}\Big((H|0\rangle\otimes|1\rangle)+(H|1\rangle\otimes|1\rangle)\Big)=\frac{1}{\sqrt{2}}\Big((\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\otimes|1\rangle)+(\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\otimes|1\rangle)\Big)=$
$\frac{1}{\sqrt{2}}\Big((\frac{1}{\sqrt{2}}(|01\rangle+|11\rangle))+(\frac{1}{\sqrt{2}}(|01\rangle-|11\rangle))\Big)=|01\rangle$ Qbitini elde edecektir. Bu da Ali'nin iletmek istediği mesajın klasik bit haline denk gelmektedir.

Superdense kodlama yüksek güvenlikli bir veri iletimi sağlamaktadır. Bunu şu şekilde anlamak mümkündür; farz edelim ki Ali ve Burak Qbitlerini dolaşık hale getirip veri iletimi yapmak istediler. Araya girip gönderdikleri Qbit üzerinde işlem yapamıyoruz. Çünkü elimizde aynı dolaşıklığa ait bir Qbit bulunmamaktadır. Eğer halihazırdaki dolaşık Qbitlere müdahele edersek iletilen Qbit dolaşık halden çıkacağı için Ali ve Burak mesaj kanallarına müdahale edildiğini anlayacaklardır.

Bu bölüm ile birlikte kuantum bilgisayarların haberleşmeleri için gereken Qbit işlemlerini ve matematiksel alt yapılarını görmüş olduk.

Kuantum Algoritmaları

Grover Algoritması[5]

Kuantum programlamada, arama algoritması olarak bilinen Grover algoritması, 1996 yılında Lov Grover tarafından geliştirilmiştir. Bu algoritma, sıralı olmayan bir veri dizinde klasik bilgisayarların $O(N)$ zamanda yaptığı arama işini $O(\sqrt{N})$ zamanda yaparak kuantum bilgisayarlarının hesaplama kabiliyetlerinin ne derece kuvvetli olduğunun bir ispatıdır aynı zamanda.

Günümüzde çoğu şirket veri tabanlarında tuttukları bilgileri ilişkisel olarak saklamadıklarından (yani düzensiz yapıda olduğundan) yapay zeka ve makine öğrenmesi sistemleri ile verilerini işlemekte ve buradan elle tutulur sonuçlar çıkarmaya çalışmaktadır. İnternette her gün artan veri boyutları göz önünde bulundurulduğunda, bu veri tabanlarında arama yapmanın ne kadar zamana mal olduğu da göz ardı edilmemelidir. Bu veriler şehirlerde yaşayan vatandaşların veri tabanları , bir üniversitenin öğrenci kayıtlarının tutulduğu ya da Google'ın müşterilerinin bilgilerini tuttuğu geniş çaplı veri tabanları olabilir.

Grover algoritmasının hızımızı ne kadar artırdığını gözlemlemek için şöyle bir örnek verelim. Mesela elimizde 1 bitlik aramayı 1 saniyede yapan bir klasik bilgisayar olduğunu farz edelim. 1.000.000 bitlik veride arama yapma süresi klasik bilgisayarlar için 1.000.000 saniye olacaktır. Bu da yaklaşık olarak 12 gün beklememiz gerektiği anlamına gelmektedir. Yukarıda da bahsedildiği gibi $O(\sqrt{N})$ sürede yapmak ise $\sqrt{1.000.000}=1000$ saniyede aynı işlemi çözebiliriz demektir. Yani 17 dakika... Şimdi birlikte Grover algoritmasının nasıl çalıştığını açıklamaya çalışalım.

Elimizde bir veri seti olsun: `my_list = [2,4,9,5,8,0,7,6]`. Rastgele bir eleman seçip arama yapmaya çalışalım, diyelim ki 7 sayısı. Klasik bir bilgisayarda bu sorunu çözmek için döngü yapıları kullanılır ve tek tek her elemana "Sen 7 misin?" sorusu sorulur. En kötü senaryoyu düşünürsek, yani 7 sayısının en son eleman olduğunu, bu sayıya ulaşmak listenin uzunluğu kadar işlem gerektirmektedir. İşte $O(N)$ ifadesinin açıklaması budur. N uzunluğa sahip bir listede arama yapmanın maliyeti.

`my_list` listesinde arama yaparak kaç adımda 7 sayısına ulaşacağımızı birlikte görelim;

```
my_list = [2,4,9,5,8,0,7,6]

# 7 sayısına eşit ise True döndüren bir foksyon.
def the_oracle(my_input):
    winner = 7
    if my_input is winner: response = True
    else: response = False
    return response

for index,deneme_sayisi in enumerate(my_list):
    if the_oracle(deneme_sayisi) is True:
        print("Kazanan sayının bulunduğu index %i"%index)
        print("%i kere liste içinde arama yapıldı."%(index+1))
        break
-----------------------------------------------
# ÇIKTI:
Kazanan sayının bulunduğu index 6
7 kere liste içinde arama yapıldı.
```

Yani 7 sayısına ulaşana kadar elemanları gezmiş olduk. 8 sayılık bir listede arama yapmak kolay tabi. Fakat listenin uzunluğunun devasa boyutlara ulaşması yapılan hesabın ne kadar yük oluşturduğunu bize göstermektedir. Peki kuantum evreninde bu algoritma nasıl çalışmaktadır? Bunu anlamak için bazı konseptlerden bahsedelim. Bu konseptleri açıklarken 3 Qbitlik bir sistem geliştirip kullanalım. 3 Qbitlik sistemlerin vektörler ile gösterimlerini buraya tıklayarak inceleyebilirsiniz. 3 Qbitlik çözümlere sahip D veri setinde arama yapalım yani 8 elemanlı $D = \{|000\rangle, |001\rangle, ..., |111\rangle\}$(durumlarımız $\{|0\rangle \rightarrow |7\rangle\}$ demektir).

— `the_oracle` — Fonksyonu

Yukarıda da örneği bulunan bu fonksyon, aradığımız elemanın işaretini değiştiren bir göreve sahiptir. 7 sayısı dışındaki tüm sayıların değerini `False` olarak ayarlamaktadır. Ya da kuantum örenğimize geçecek olursak D kümesindeki elemanlardan bizim aradığımız 3 Bitlik çözüm olan w ya eşit olsun. Matematiksel ifade ile;

$$U_w |x\rangle = \begin{cases} |x\rangle & \text{if } x \neq w \\ -|x\rangle & \text{if } x = w \end{cases}$$

Yani aradığımız eleman $w = |101\rangle$ ise U_w matrisi aşağıdaki gibi olacaktır. Tam olarak aradığımız elemanı diğerlerinden ayrıştıran bir foksyon.<

$$U_w = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Bu matris ifadesini manuel olarak biz kendi elimiz ile belirlemiş olduk. Başka bir ifade ile köşengende yer alan altıncı terimi -1 olarak kendi elimizle yazdık bunu daha fonksyonel bir ifade ile yazacak olursak işlerimiz daha kolay olacaktır. Bu durumda `the_oracle` ;$o(|x\rangle)$ fonksyonu tanımlayalım alacağı x değeri için eğer $x = w$ eşitliği varsa $o(|x\rangle) = 1$ ve $x \neq w$ için ise $o(|x\rangle) = 0$ değerlerini dönsün. `the_oracle` fonksyonumuz ve ona denk gelen matrisimiz şöyle güncellenecektir;

$$U_w |x\rangle = (-1)^{o(|x\rangle)} |x\rangle$$

$$U_w = \begin{pmatrix} (-1)^{o(|0\rangle)} & & & & & & 0 \\ 0 & (-1)^{o(|1\rangle)} & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot & (-1)^{o(|2^n-1\rangle)} \end{pmatrix}$$

Bu `the_oracle` fonksyonu aradığımız elemanı bulurken kullanılmak üzere burada bırakıp ikinci önemli başlığımız olan **genlik yükseltmesine** geçeceğiz.

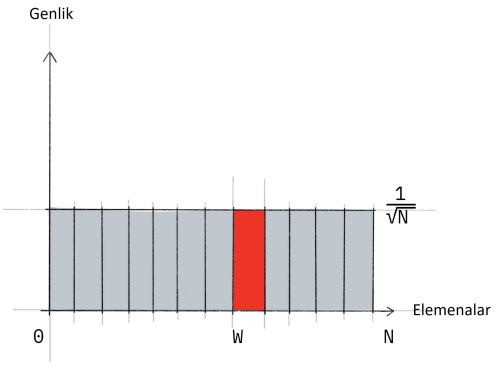
Genlik Yükseltmesi (Amplitude Amplification)

Başlamadan hatırlatalım; arama listeye bakmadan önce aranan elemanın nerede olduğunu bilmiyoruz. Bu sebeple rastgele yapacağımız tahmin ile listedeki herhangi bir elemana eşit olasılıklarla ulaşağız. Bu durum uniform süperpozisyon olarak adlandırılmaktadır. Hadamark kapısının Qbitleri eşit olasılık haline soktuğunu açıklamıştık. Genlik yükseltmesi yönteminin temel amacı bu eşit olasılık durumunu kendi lehimize çevirip aradığımız sonucun gözlem sonucunda gelme ihtimalini artırabilmektir. Eşit olasılıklı N boyutlu ihtimaller uzayı N boyutlu bir Hadamard kapısı anlamına gelmektedir. Qbitlerin gösterimlerini buradan inceleyebilirsiniz. Hadamard uygulanmış D kümemiz aşağıdaki gibi görünecektir;

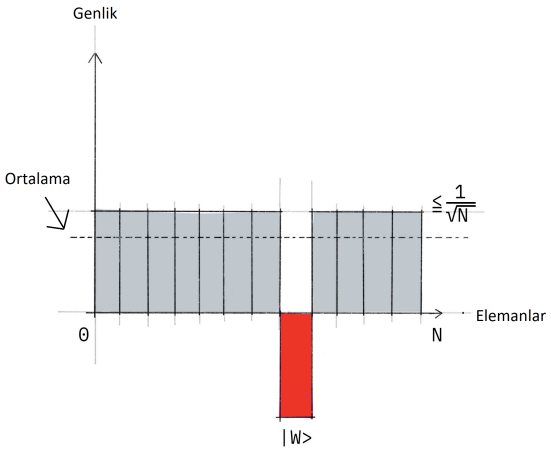
$$H^{\otimes k} |s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

Bu halde iken $D = \{|000\rangle, |001\rangle, ..., |111\rangle\}$ veri setimizde ölçüm yapacak olursak eşit olasılıkla süperpozisyonda bulunan elemanların hepsini $\frac{1}{2^3}$olasılıkla elde edebileceğimizi anlarız.Yani elimizde Hadamard uygulanmış halde $\{\frac{1}{2^3}|000\rangle + \frac{1}{2^3}|001\rangle + ... + \frac{1}{2^3}|111\rangle\}$ Qbitleri bulunmaktadır. Burada genlik yükseltmesi yaparak `the_oracle` fonksyounun işaretlediği elemanın olasılık genliğini artırırken diğer ihtimalleri azaltmayı amaçlamaktadır.

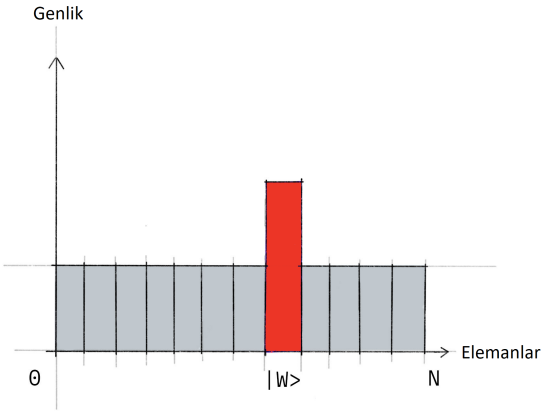
Bunu yaparken şöyle bir yol izlenmektedir:



1—Elimizdeki elemanların genlikleri ve işaretlenmiş eleman.



2— the_oracle() fonksyonunun işaretlerdiği eleman



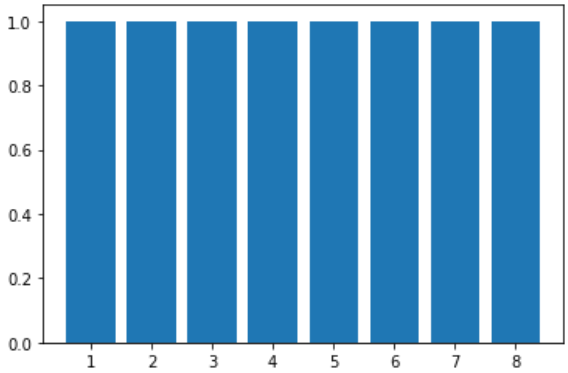
3— Ortalamalarına göre simetri alarak olasılık artırma

Bu durumu python kullanarak açıklamaya çalışalım. 8 elemanlı bir liste hazırlayalım. Bu listenin 4. elemanını arayalım.

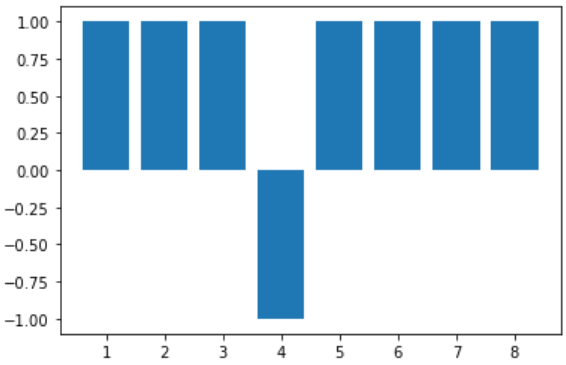
```
from matplotlib.pyplot import bar

labels = []
L = []
for i in range(8):
    labels = labels + [i+1]
    L  = L + [1]

#print(labels,L)
# listenin elemanlarını görselleştirelim.
bar(labels,L)
```



Şimdi 4. elemanı işaretleyelim. Yani işaretini ters çevirelim.



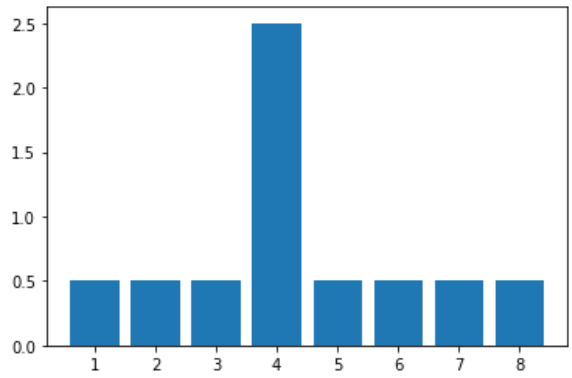

```
L[3] = -1 * L[3]

print(L)
bar(labels,L)
# ÇIKTI
[1, 1, 1, -1, 1, 1, 1, 1]
```

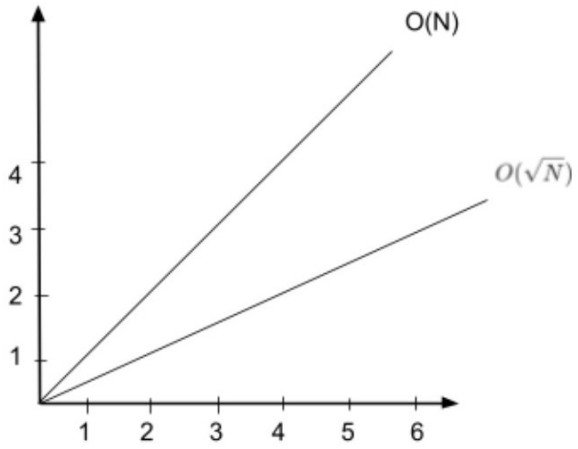
```
toplam = 0
```

```
for i in range(len(L)):
    toplam += L[i]
# tüm elemanların ortalamasını alma
ortalama = toplam / len(L)
# ortalamaya göre simetriğinin alınması
for i in range(len(L)):
    deger= L[i]
    yeni_deger= ortalama- (L[i]-ortalama)
    L[i] = yeni_deger

print(L)
# listenin elemanlarını görselleştirelim.
bar(labels,L)
-----------------------------------------------
#ÇIKTI
[0.5, 0.5, 0.5, 2.5, 0.5, 0.5, 0.5, 0.5]
```



Eğer ortalamaya göre simetri alma operatörü olarak U_s işlemini atarsak bu operatörü $U_s = \mu + (\mu - \alpha_{|x\rangle})$ olarak tanımlanır, μ ortalamayı, $\alpha_{|x\rangle}$ise $|x\rangle$ Qbitinin ortalamaya uzaklığını ifade etmektedir. Bu tanımlamalardan sonra Qbit üzerinde yaptığımız işlem tam olarak $|s_{t+1}\rangle = U_s U_w \ |s_t\rangle$ olarak yazılmaktadır, 1 adımdan sonra dolaşık halde bulunan Qbitlerin olasılık dağılımı. t adımdan sonra ise $|s_t\rangle = (U_s U_w)^t \ |s_0\rangle$ olarak yazılır. Peki ama bu işlemi kaç kere uygularsak aranan elemanın olasılığı 1'e yaklaşmış olur. Cevap yaklaşık $\sqrt{N}$ olarak karşımıza çıkmaktadır. Son olarak ne kadar hızlandığımızı gösteren bir grafiğe bakalım:



Ne kadar hızlandığımızın resmi

Bu bölümde Grover algoritmasının nasıl çalıştığını Hadamard kapısı ile olan ilişkisini, `the_oracle` fonksyonu ve genlik yükseltme metodunu açıklamış olduk.

Kaynakça

[1] https://en.wikipedia.org/wiki/Tensor_product#Tensor_product_of_linear_maps

[2]https://www.researchgate.net/publication/254040387_Quantum_computers_Registers_gates_and_algorithms

[3]http://akyrillidis.github.io/notes/quant_post_7

[4]https://www.scientificamerican.com/article/china-reaches-new-milestone-in-space-based-quantum-communications/

[5]https://quantum-computing.ibm.com/composer/docs/iqx/guide/grovers-algorithm