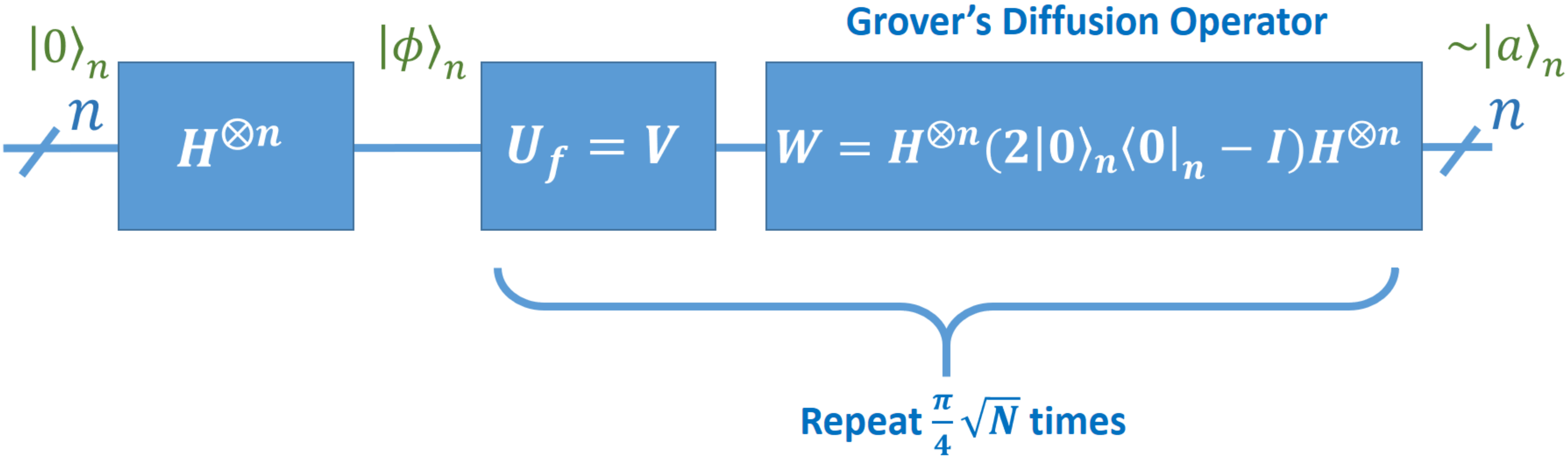
Introduction to Quantum Information and Quantum Machine Learning

Project - class 4

Dr Gustaw Szawioła, docent PUT
D. Sc. Eng. Przemysław Głowacki

1. Implementation of Grover’s Algorithm

The general form of Grover's algorithm



Importing the necessary libraries – Qiskit and Python

- # Import the necessary libraries
- `import math`
- `import numpy as np`
- `from math import sqrt`
- `from numpy import pi`
- `from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile`
- `from qiskit.quantum_info import Statevector, Operator`
- `from qiskit.circuit.library.standard_gates import XGate`
- `from qiskit.visualization import plot_histogram, plot_distribution`
- `from qiskit_aer import Aer`
- `from time import process_time`

Determining QPU, i.e. backend

```
# qiskit 2.2.1
backend = Aer.get_backend('unitary_simulator')
```

Phase oracle matrix construction

- The form of the function for which we want to construct a phase oracle, in other words: the form of the function we want to build (embed) into the unitary matrix

- $f(x) = \begin{cases} 1, & x = a \rightarrow |a\rangle \\ 0, & x \neq a \rightarrow |a_{\perp}\rangle \end{cases}$

- The general form of the phase oracle of Grover's algorithm

- $\hat{U}_f|x\rangle = \hat{U}_f|q_n \cdots q_1 q_0\rangle =$

$= (-1)^{f(x)}|q_n \cdots q_1 q_0\rangle$

$= \begin{cases} -|x\rangle = -|q_n \cdots q_1 q_0\rangle, & x = a \rightarrow |a\rangle \\ |x\rangle = |q_n \cdots q_1 q_0\rangle, & x \neq a \rightarrow |a_{\perp}\rangle \end{cases}$

Operation of the phase oracle operator for n=3

- Tensor space basis
- $|0\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle = |000\rangle$
- $|1\rangle = |0\rangle \otimes |0\rangle \otimes |1\rangle = |001\rangle$
- $|2\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle = |010\rangle$
- $|3\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle = |011\rangle$
- $|4\rangle = |1\rangle \otimes |0\rangle \otimes |0\rangle = |100\rangle$
- $|5\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle = |101\rangle$
- $|6\rangle = |1\rangle \otimes |1\rangle \otimes |0\rangle = |110\rangle$
- $|7\rangle = |1\rangle \otimes |1\rangle \otimes |1\rangle = |111\rangle$

- An example of the operation of the $\hat{U}_f$ oracle operator for $|a\rangle = |2\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle = |010\rangle$
- $\hat{U}_f\,|0\rangle = \hat{U}_f|000\rangle = |000\rangle$
- $\hat{U}_f\,|1\rangle = \hat{U}_f|001\rangle = |001\rangle$
- $\hat{U}_f\,|2\rangle = \hat{U}_f|010\rangle = -|010\rangle$
- $\hat{U}_f\,|3\rangle = \hat{U}_f|011\rangle = |011\rangle$
- $\hat{U}_f\,|4\rangle = \hat{U}_f|100\rangle = |100\rangle$
- $\hat{U}_f\,|5\rangle = \hat{U}_f|101\rangle = |101\rangle$
- $\hat{U}_f\,|6\rangle = \hat{U}_f|110\rangle = |110\rangle$
- $\hat{U}_f\,|7\rangle = \hat{U}_f|111\rangle = |111\rangle$

Implementation of a quantum gate representing the phase oracle operator

Matrix-based operator construction for phase oracle

```
# Construction of the Uf matrix
nn=3
oracle=np.identity(2**nn)
oracle[2,2]=-1
print(oracle)
Uf=Operator(oracle)
Operator.is_unitary(Uf)
```

- The form of the "oracle" variable that is the basis for the construction of the U_f oracle operator

```
[[ 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.]]
```

- Result of checking the unitarity of the constructed operator

```
True
```

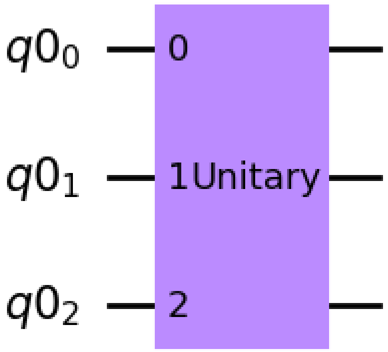
Implementation of a quantum gate representing the phase oracle operator

U_f as matrix gate

- # Creating quantum registers,
- # classical registers and a quantum circuit
- # representing the created Uf operator
- # Number of qubits and bits
- n0=nn
- # Quantum Register
- q0 = QuantumRegister(n0)
- # "Empty" quantum circuit
- # for a gate called 'Uf’
- CircuitUf = QuantumCircuit(q0,name='Uf')

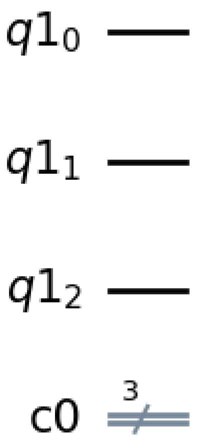
- # Attaching the Uf operator to the circuit
- # representing the Uf gate
  CircuitUf.append(Uf,[q0[0],q0[1],q0[2]])
- # Sketch of a quantum circuit
  CircuitUf.draw(output='mpl’)

- # Transforming the Uf operator
- # to the „uf” quantum gate
- # marked as Uf
- uf=CircuitUf.to_gate()



Creating an "empty" matrix of a quantum circuit of Grover's algorithm

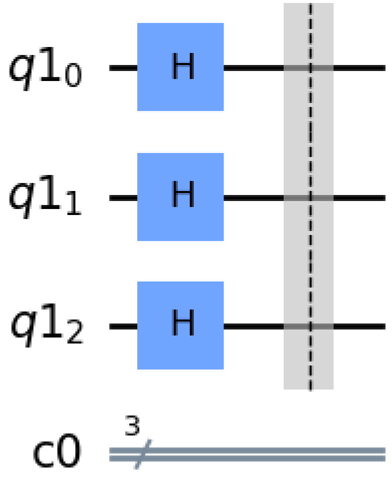
```
• # Number of qubits and bits
• n=nn
• #  Quantum Register
• q = QuantumRegister(n)
• # Classical Register
• c = ClassicalRegister(n)
• # "Empty" quantum circuit
• # - the core of Grover's algorithm
• Circuit = QuantumCircuit(q,c)
• # Sketch of a quantum circuit
  Circuit.draw(output='mpl')
```



Quantum circuit initiating the $|\phi\rangle$ state

```
• # State |fi> initiation
• Circuit.h(q[0])
• Circuit.h(q[1])
• Circuit.h(q[2])
• Circuit.barrier()
• # Sketch of a quantum circuit
  Circuit.draw(output='mpl')
```

Suggestion: do it in a loop for n qubits



Determining the optimal number r of iterations of the Grover operator $\hat{G} = \hat{V}\hat{W}$

- Example
- $n = 3$
- $N = 2^n = 8$
- $r = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor \sim \mathcal{O}(\sqrt{N})$,

  where $\lfloor x \rfloor = \max(m \in \mathbb{Z}: m \leq x)$
- $r = \left\lfloor \frac{\pi}{4}\sqrt{8} \right\rfloor = \lfloor 2{,}22144 \rfloor = 2$

Qiskit code

```
repeat=math.floor((pi/4)*sqrt(2**n))
print(repeat)
[out] 2
```

Implementation of Grover's diffusion operator $\widehat{W} = 2\,|\phi\rangle\langle\phi| - \hat{\mathbb{I}}$

Calculation of matrix elements of the diffusion operator

$$\widehat{W}_{x,x'} = \langle x|\widehat{W}|x'\rangle =$$

$$= \langle q_n \cdots q_1 q_0|\widehat{W}|q'_n \cdots q'_1\, q'_0\rangle$$

in basis $\{|x\rangle\} = \{|q_n \cdots q_1 q_0\rangle\}$

- Operator $\widehat{W}$ matrix form
- $$\widehat{W} = \frac{2}{2^n}\begin{pmatrix} 1-\frac{2^n}{2} & 1 & 1 & 1 & 1 \\ 1 & 1-\frac{2^n}{2} & 1 & 1 & 1 \\ 1 & 1 & \ddots & 1 & 1 \\ 1 & 1 & 1 & 1-\frac{2^n}{2} & 1 \\ 1 & 1 & 1 & 1 & 1-\frac{2^n}{2} \end{pmatrix}$$

Equivalent form of the diffusion operator

Use of the Hadamard gates

$$\widehat{W} = 2\,|\phi\rangle\langle\phi| - \hat{\mathbb{I}} = \widehat{H}^{\otimes n}\left(2|0\rangle_{n\,n}\langle 0| - \hat{\mathbb{I}}\right)\widehat{H}^{\otimes n}$$

Attention! Mathematical proof at the lecture.

Equivalent form of the operator

$$2\,|0\rangle_{n\,n}\langle 0| - \hat{\mathbb{I}}$$

Use of the Hadamard gates

$$2\,|0\rangle\langle 0| - \hat{\mathbb{I}} = \hat{X}^{\otimes n}\left(2|1\rangle_{n\,n}\langle 1| - \hat{\mathbb{I}}\right)\hat{X}^{\otimes n}$$

where $|1\rangle_n = |11\cdots 1\rangle$,

$${}_n\langle 1| = \langle 11\cdots 1|$$

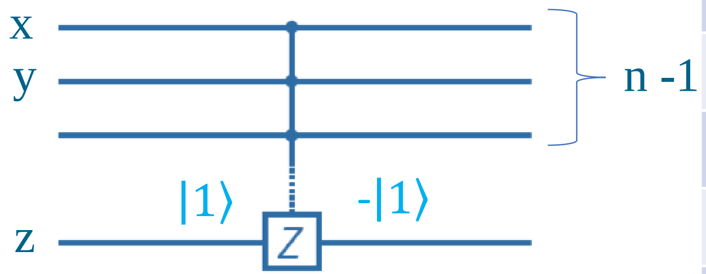
$$2\,|1\rangle_{n\,n}\langle 1| - \hat{\mathbb{I}} = -\left(\hat{\mathbb{I}} - 2|1\rangle_{n\,n}\langle 1|\right)$$

Attention! Mathematical proof at the lecture.

Multi-qubit controlled operator gate

$$\widehat{C_{n-1}Z} = \hat{\mathbb{I}} - 2|1\rangle_{n\,n}\langle 1|$$

Symbol of multi-qubit controlled gate $\widehat{C_{n-1}Z}$



$$\widehat{C_{n-1}Z} = \hat{\mathbb{I}}_n - 2|1\rangle_{n\,n}\langle 1|$$

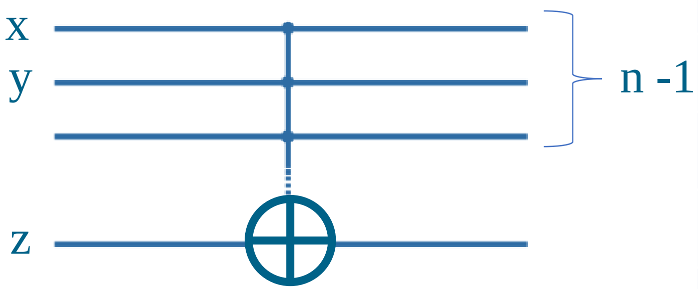
"Truth" table for $\widehat{C_{n-1}Z}$ (n=3)

z	y	x	z'	y'	x'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	−1	1	1

Multi-qubit controlled operator gate

$$\widehat{C_{n-1}X}$$

Symbol of multi-qubit controlled gate $\widehat{C_{n-1}X}$



$$\widehat{C_{n-1}X} = \hat{\mathbb{I}}_2 \otimes \hat{\mathbb{I}}_{n-1} + \hat{X} \otimes |1\rangle_{n-1\,n-1}\langle 1|$$

Gate Implementation $\widehat{C_{n-1}X}$

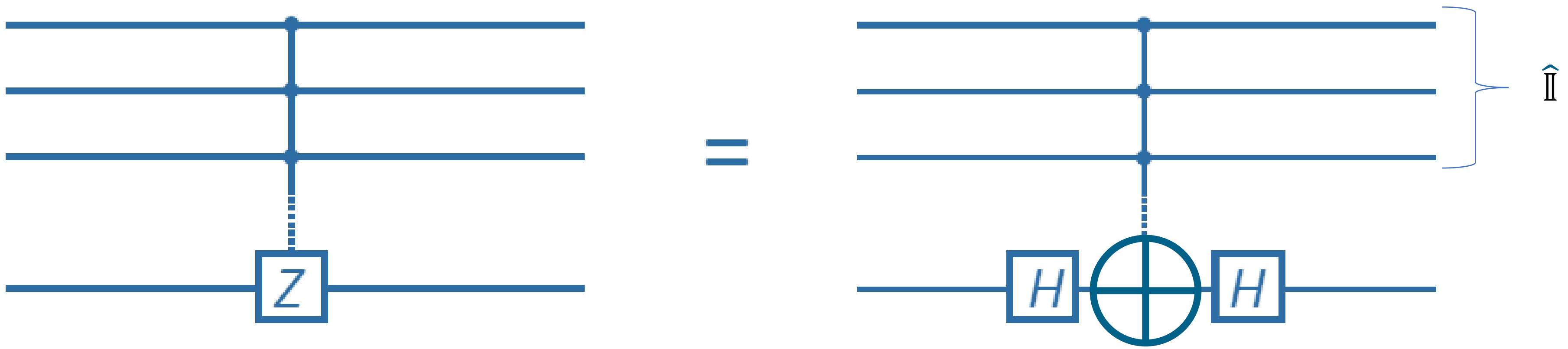
```
mccx=XGate().control(n0-1)
```

"Truth" table for $\widehat{C_{n-1}X}$ (n=3)

z	y	x	z'	y'	x'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	0	1	1

Gate Implementation $\widehat{C_{n-1}Z} = \hat{\mathbb{I}} - 2|1\rangle_{n\,n}\langle 1|$

Since $\hat{Z} = \widehat{H}\,\widehat{X}\,\widehat{H}$ we also have, $\widehat{H}^2 = \hat{\mathbb{I}}^2$



$$\widehat{C_{n-1}Z} = \left(\widehat{H}\otimes\widehat{\mathbb{I}_{n-1}}\right)\widehat{C_{n-1}X}\left(\widehat{H}\otimes\widehat{\mathbb{I}_{n-1}}\right)$$

Implementation of $\widehat{W}$ operator

- $\widehat{W} = 2\,|\phi\rangle\langle\phi| - \hat{\mathbb{I}} = \hat{H}^{\otimes n}\left(2\,|0\rangle\langle 0| - \hat{\mathbb{I}}\right)\hat{H}^{\otimes n} =$

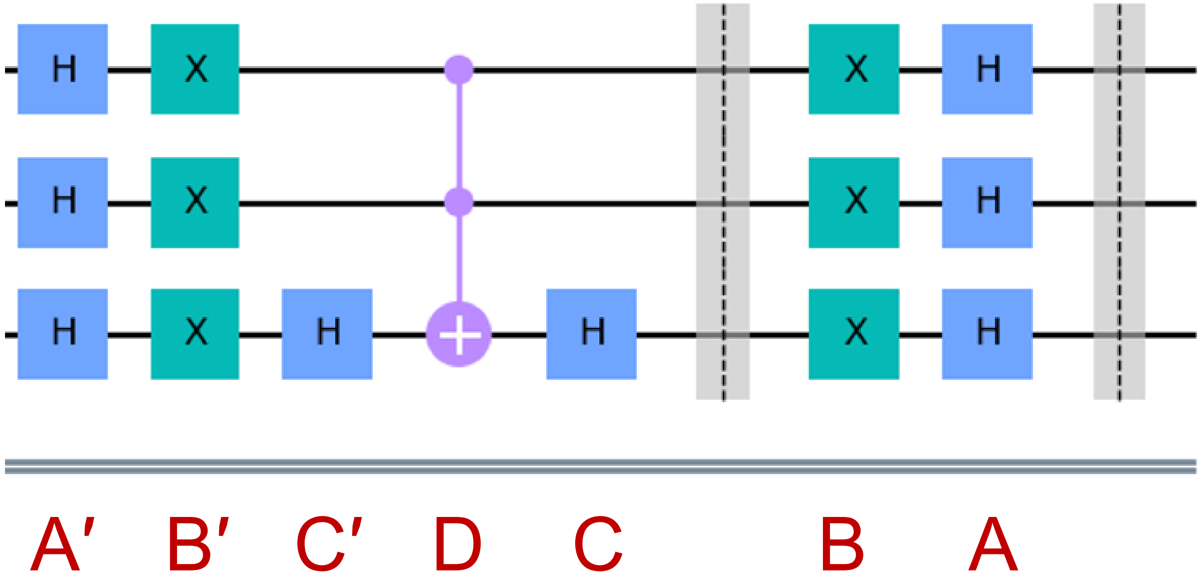
$$= \hat{H}^{\otimes n}\underbrace{\left(\hat{X}^{\otimes n}\left(2|1\rangle_{n\,n}\langle 1| - \hat{\mathbb{I}}\right)\hat{X}^{\otimes n}\right)}_{=2\,|0\rangle\langle 0| - \hat{\mathbb{I}}}\hat{H}^{\otimes n} =$$

$$= \hat{H}^{\otimes n}\hat{X}^{\otimes n}\underbrace{\left(-\widehat{C_{n-1}Z}\right)}_{=-(\hat{\mathbb{I}}-2|1\rangle_{n\,n}\langle 1|)}\hat{X}^{\otimes n}\hat{H}^{\otimes n} =$$

$$= -\hat{H}^{\otimes n}\hat{X}^{\otimes n}\underbrace{\left(\hat{H}\otimes\widehat{\mathbb{I}_{n-1}}\right)\widehat{C_{n-1}X}\left(\hat{H}\otimes\widehat{\mathbb{I}_{n-1}}\right)}_{=\widehat{C_{n-1}Z}}\hat{X}^{\otimes n}\hat{H}^{\otimes n}$$

A′ B′ C′ D C B A

Quantum circuit of the operator $\widehat{W}$



A fragment of the code implementing the gate

```
# A fragment of the code implementing the operator W
```

- `Circuit.barrier()`
- `Circuit.h(q[0])`
- `Circuit.h(q[1])`
- `Circuit.h(q[2])`
- `Circuit.x(q[0])`
- `Circuit.x(q[1])`
- `Circuit.x(q[2])`
- `Circuit.h(q[2])`
- `Circuit.append(mccx, [q[0],q[1],q[2]])`
- `Circuit.h(q[2])`
- `Circuit.x(q[0])`
- `Circuit.x(q[1])`
- `Circuit.x(q[2])`
- `Circuit.h(q[0])`
- `Circuit.h(q[1])`
- `Circuit.h(q[2])`
- `Circuit.barrier()`

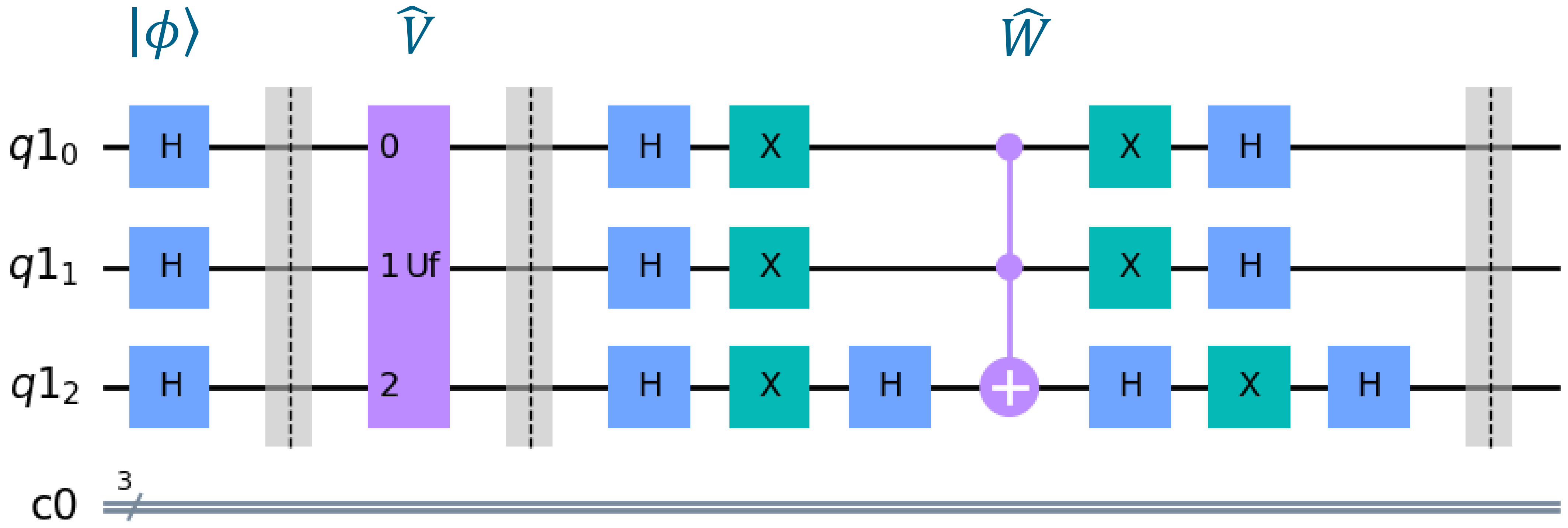
2. Final implementation of Grover's algorithm for n=3

A complete for loop (for n=3) calling $r = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor = 2$ times the operator $\widehat{V}\widehat{W}$

```
for ii in range(repeat):
    Circuit.append(uf,[0,1,2])
    Circuit.barrier()
    # Początek implemenatcji
    # operatora dyfuzji W
    Circuit.h(q[0])
    Circuit.h(q[1])
    Circuit.h(q[2])
    Circuit.x(q[0])
    Circuit.x(q[1])
    Circuit.x(q[2])
    Circuit.h(q[2])
Circuit.append(mccx, [q[0],q[1],q[2]])
    Circuit.h(q[2])
    Circuit.x(q[0])
    Circuit.x(q[1])
    Circuit.x(q[2])
    Circuit.h(q[0])
    Circuit.h(q[1])
    Circuit.h(q[2])
# The end of the implementation of diffusion operator W
    Circuit.barrier()
    print("N=",ii+1)
    # print(Circuit)
    display(Circuit.draw(output='text'))
```

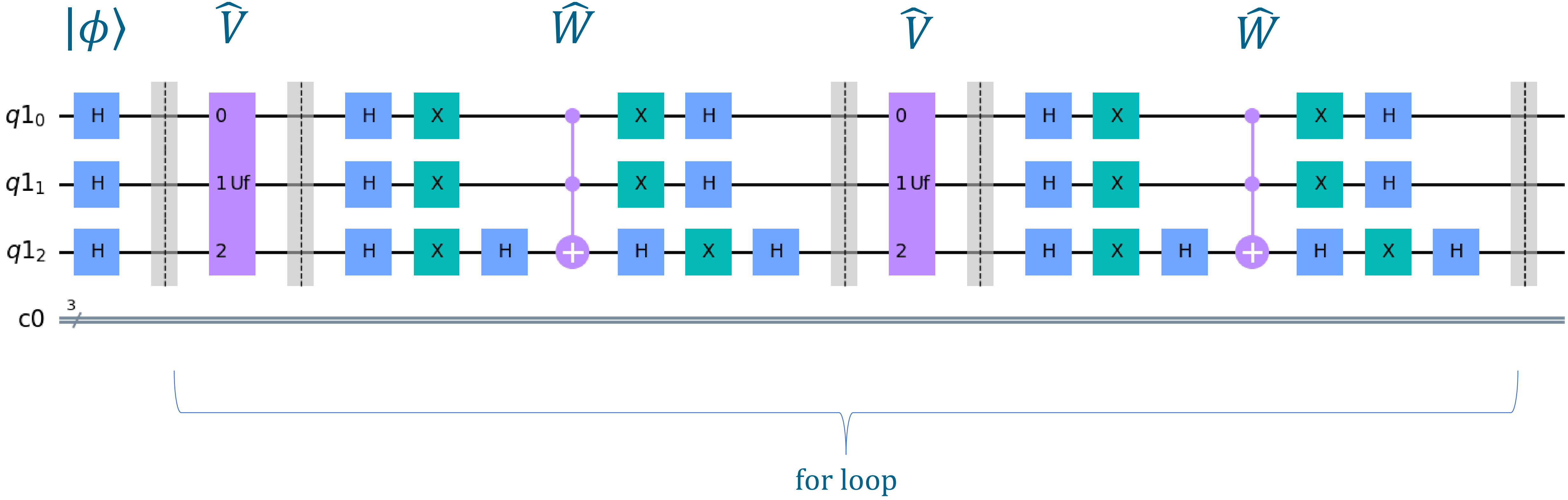
Single call of the operator $\hat{W}\,\hat{V}$

$\hat{W}\,\hat{V}$



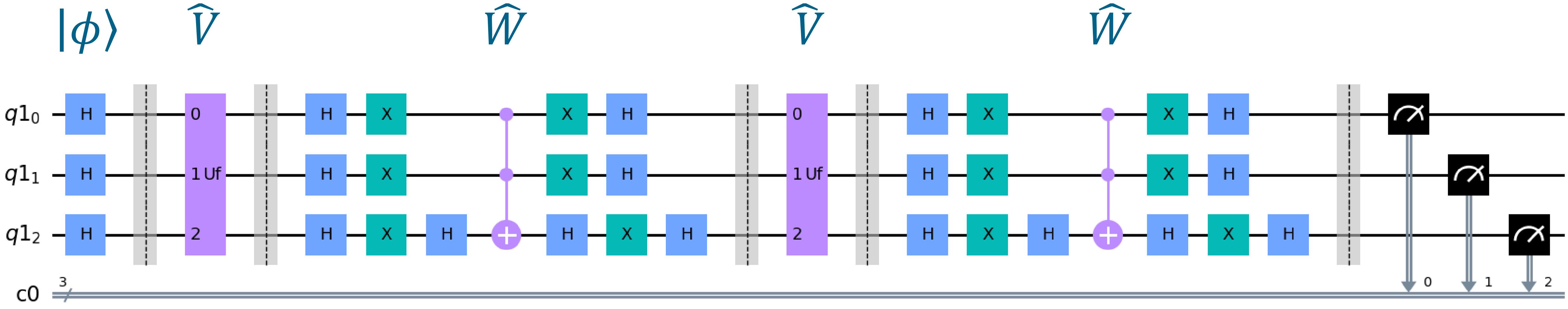
Double call of the operator $\widehat{W}\,\widehat{V}$:

$\widehat{W}\,\widehat{V}\,\widehat{W}\,\widehat{V}$



Adding final measurement operations to the quantum circuit

```
Circuit.measure(q[0],c[0])
Circuit.measure(q[1],c[1])
Circuit.measure(q[2],c[2])
Circuit.draw(output='mpl') # draw mpl
```



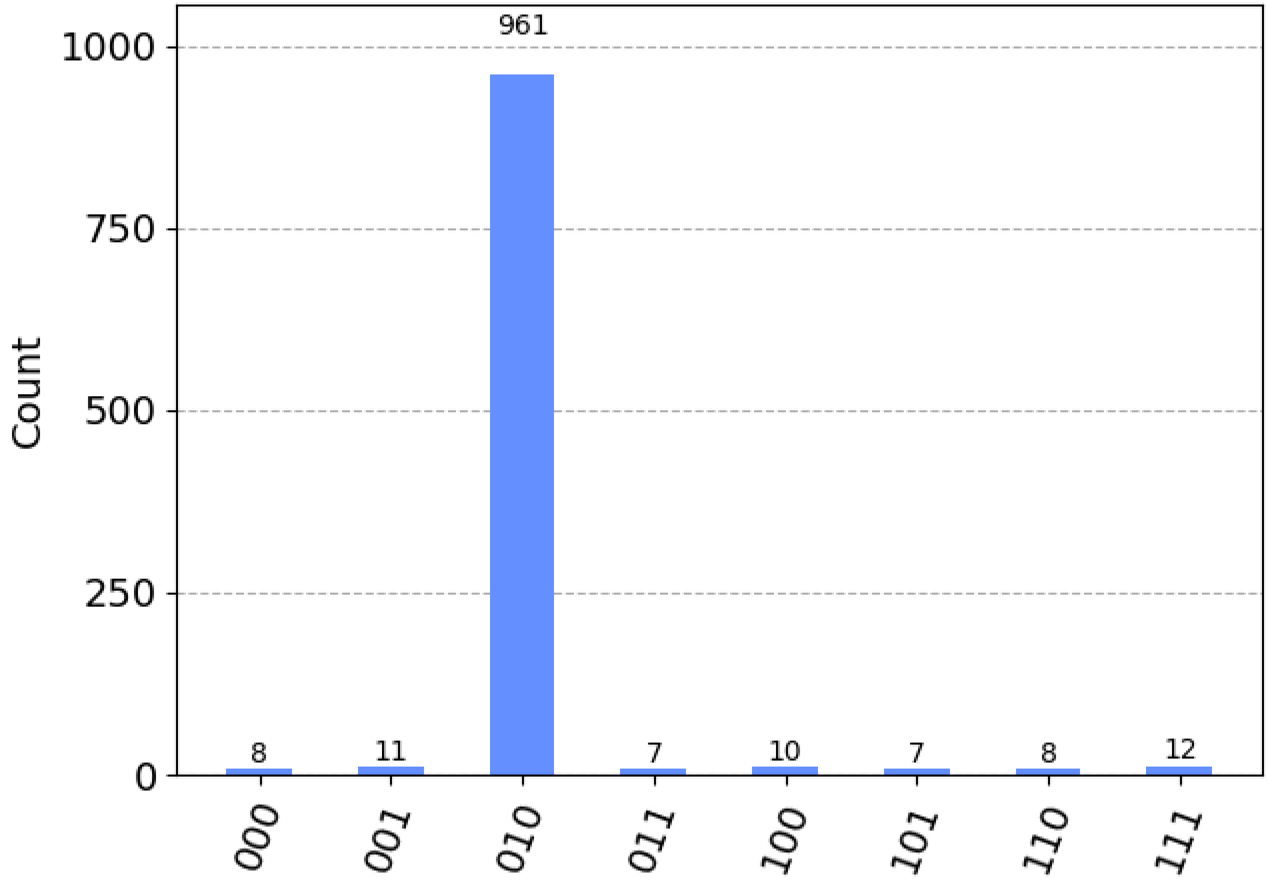
Selecting the 'qasm_simulator' computational backend and performing the calculations

- # Choosing a quantum simulator (or processor).
- backend = BasicAer.get_backend('qasm_simulator')
- # Transpile the circuit for the specific backend (needed in qiskit 1.x)
- transpiled_circuit = transpile(Circuit, backend_sim)
- # Performing quantum calculations
- job_sim0 = backend_sim.run(transpiled_circuit, shots=2**10)   n+7
- sim_result0 = job_sim0.result()
- # Numerical presentation of measurement results
- print(sim_result0.get_counts(Circuit))

{'010': 961, '000': 8, '110': 8, '111': 12, '100': 10, '001': 11, '011': 7, '101': 7}

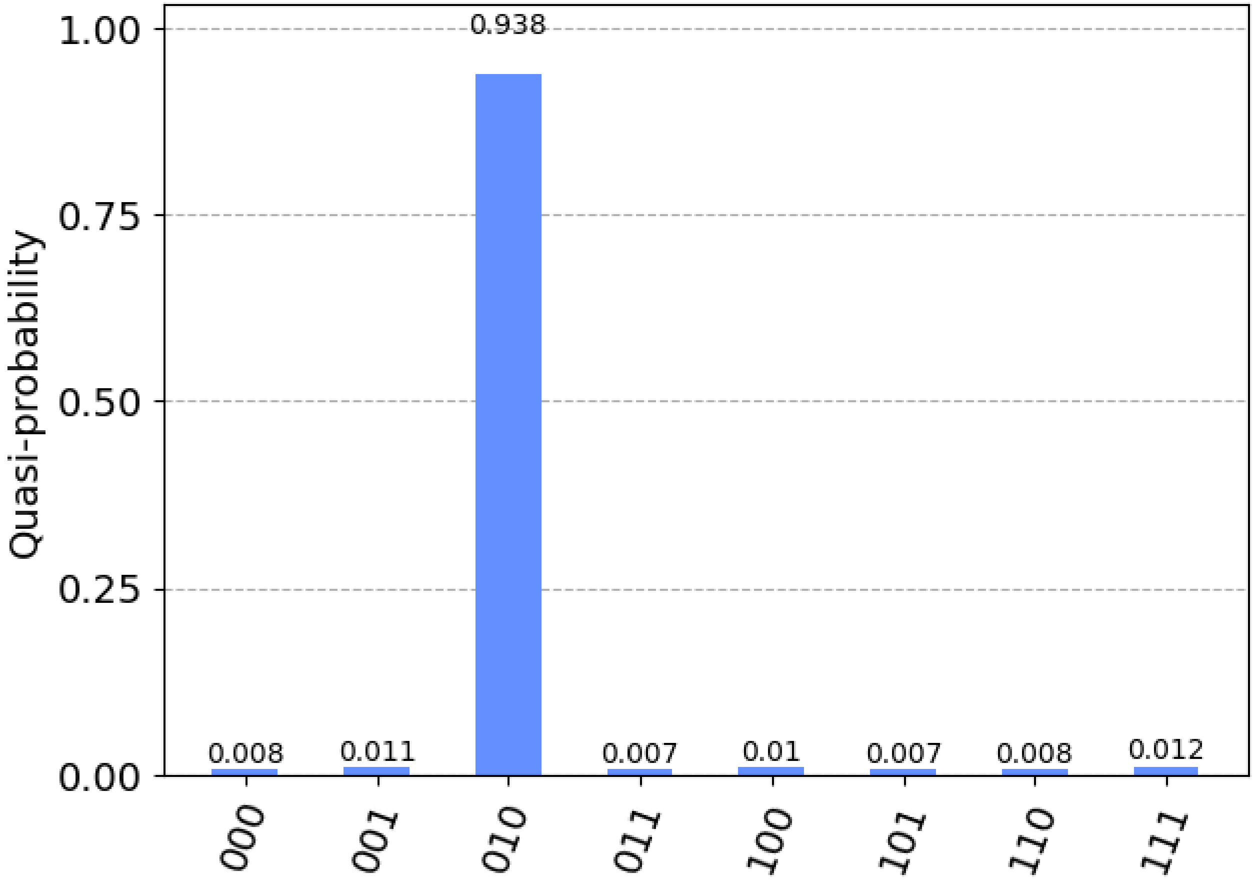
Histogram of counts

- `# Graphical presentation of X measurement results`
- `plot_histogram (sim_result0.get_counts(Circuit))`



Probability distribution

- `# Graphical presentation of X measurement results`
- `plot_distribution (sim_result0.get_counts(Circuit))`



3. Task for project

Tasks:

1. For a given n (n=2,…,6), determine the optimal number $r = \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ of necessary operator repetitions $\hat{G} = \widehat{W}\,\hat{V}$.
2. Modify the program to implement Grover's algorithm (file: ***IQIQML_Project_4_tutorial_Q221.ipynb***) adapting it to any number of qubits n (we will research the algorithm for $2 \leq n \leq 6$). The program modification should also include the option to change r to any natural number.
3. For $n = 6$ i $a = (Student\_ID\_number)_{mod\ 2^n}$ [for example 136225 %64 -> out: 33] simulate Grover's algorithm, determine the probability p_a of detecting state $|a\rangle$ (which is the solution to Grover's problem) in each subsequent iteration step s, with $s = 1,\ldots,r$, i.e. from a single step to the optimal number of iterations. Plot the graph of $p_a(s)$.
4. Repeat point 3. for s=*1*,…,$\left\lfloor \frac{\pi}{2}\sqrt{2^n} \right\rfloor$.
5. For the number of qubits $n = 2, \ldots, 6$ and $a = (Student\_ID\_number)_{mod\ 2^n}$ perform simulations of Grover's algorithm for the optimal number of iterations (for a given n) and based on this, determine the graph $p_a(n)$

The End

Zadania:

1. Dla danego n $(n = 2, \dots, 6)$ wyznacz optymalną liczbę $r = \left\lfloor \frac{\pi}{4}\sqrt{2^n} \right\rfloor$ niezbędnych powtórzeń operatora $\hat{G} = \hat{W}\,\hat{V}$.
2. Zmodyfikuj program do implementacji algorytm Grovera (plik ….) dostosowując go do dowolnej liczby kubitów n (będziemy prowadzić badania algorytmu dla $2 \leq n \leq 6$) . Modyfikacja programu powinna również uwzględniać opcję zmiany r na dowolną liczbę naturalną.
3. Dla $n = 6$ i $a = (numer\_indeksu)_{mod\ 2^n}$ przeprowadź symulację algorytmu Grovera wyznacz prawdopodobieństwo p_a wykrycia stanu $|a\rangle$ (stanowiącego rozwiązanie problemu Grovera) w każdym kolejnym kroku iteracyjnym s przy czym $s = 1, \dots, r$, tzn. od pojedynczego kroku do optymalnej liczby iteracji. Wykreśl wykres $p_a(s)$.
4. Powtórz punkt trzeci dla $s=1, \dots, \left\lfloor \frac{\pi}{2}\sqrt{2^n} \right\rfloor$.
5. Dla liczby kubitów $n = 2, \dots, 6$ i $a = (numer\_indeksu)_{mod\ 2^n}$ przeprowadź symulacje algorytmu Grovera dla optymalnej liczby iteracji (dla danego n) i na tej podstawie wyznacz wykres $p_a(n)$

- Kodowanie podstawowe $|x\rangle$ i celem jest znalezienie wektora $|a\rangle$

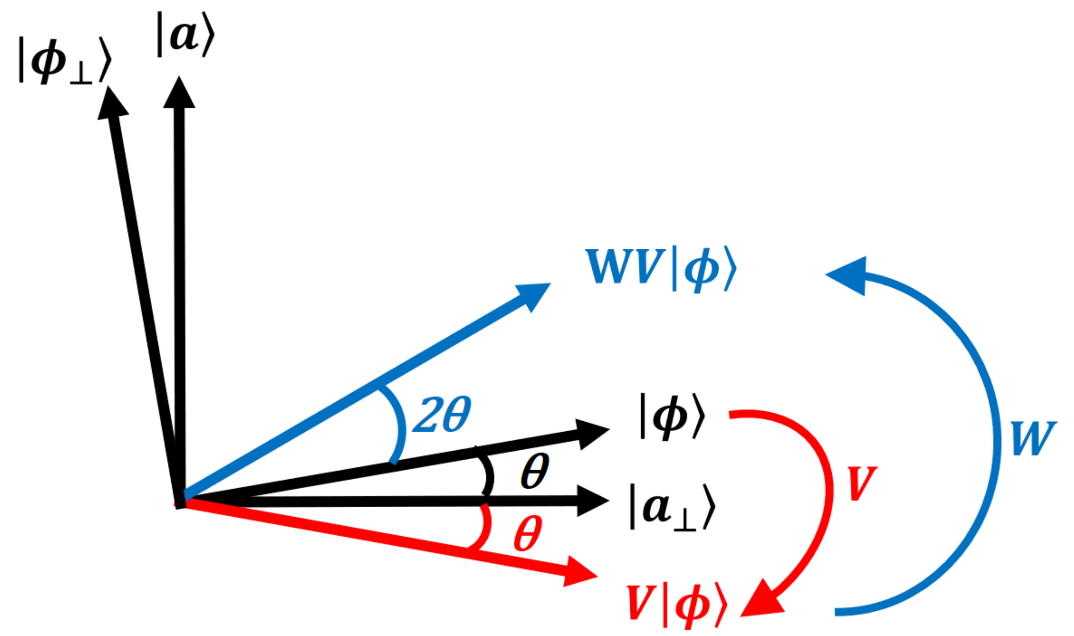
$$f(x) = \begin{cases} 1, & x = a \;\rightarrow |a\rangle \\ 0, & x \neq a \;\rightarrow |a_\perp\rangle \end{cases}$$

i. $|a\rangle$: wektor docelowy

ii. $|a_\perp\rangle$: wektor prostopadły

- $\langle a_\perp | a \rangle = 0$
- $|a_\perp\rangle = \frac{1}{\sqrt{2^n - 1}} \sum_{\substack{x=0 \\ x \neq a}}^{2^n-1} |x\rangle$
- $|\phi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$

Operator $\widehat{W}$ działając na nowy wektor $\hat{V}|\phi\rangle$ obraca ten wektor do wektora $\widehat{W}\hat{V}|\phi\rangle$, który jest o kąt 2θ bliżej wektora docelowego, szukanego wektora $|a\rangle$



Algorytm Grovera sprowadza się powtarzania procedury $\widehat{W}\hat{V}$, tzn. $\underbrace{\widehat{W}\hat{V} \cdots \widehat{W}\hat{V}}_{\sqrt{N}} |\phi\rangle$