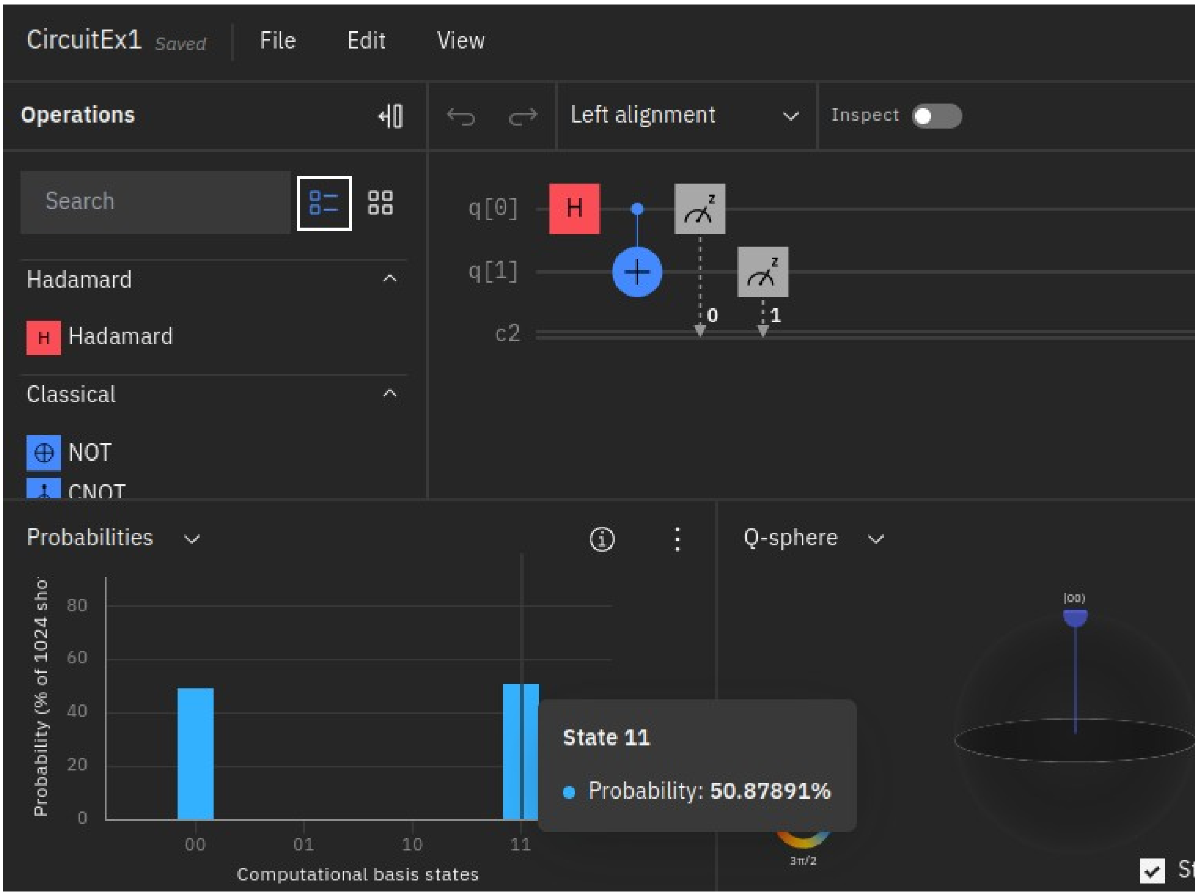
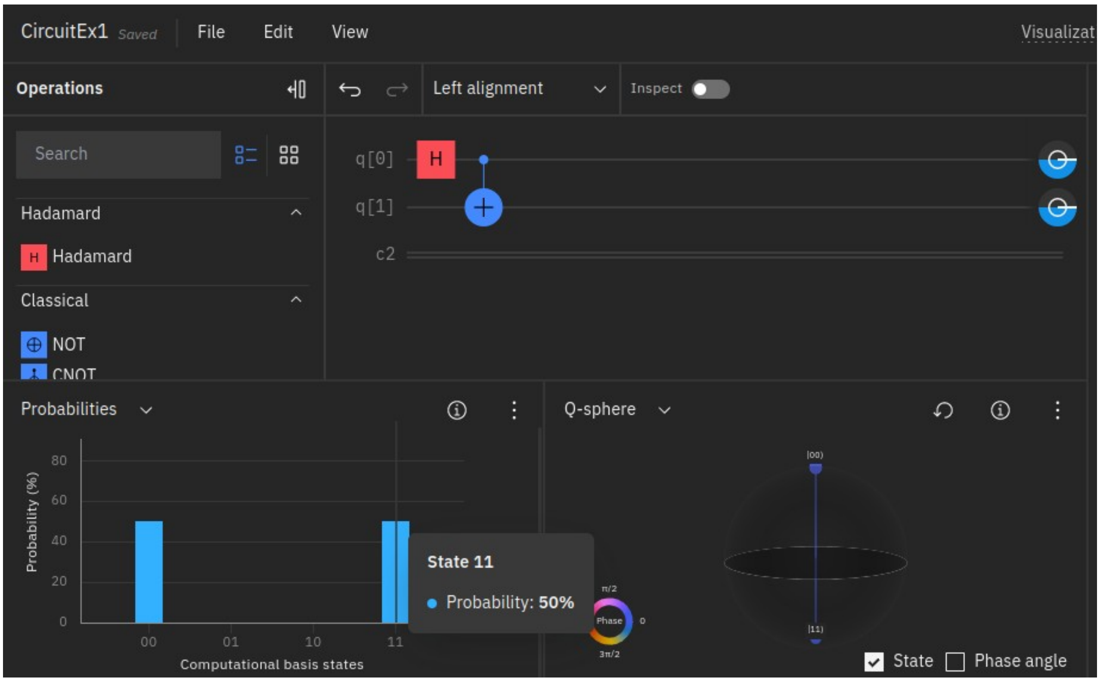
EXERECICI 4

Apartat 2



Com es pot veure en la imatge, la probabilitat de |0>|0> es quasi del 50 % i la de |1>|1> també, si no poso mesures, ho fa amb el 50% de probabilitat exacte per a cada estat, tal i com es pot veure en la següent imatge:



In [14]:
```
from ibm_quantum_widgets import CircuitComposer
from qiskit import QuantumCircuit, transpile, Aer, IBMQ, execute
from numpy import pi
from qiskit.tools.visualization import *

# Build
#------

# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)

# Add a H gate on qubit 0
circuit.h(0)

# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)

# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])

# END
```

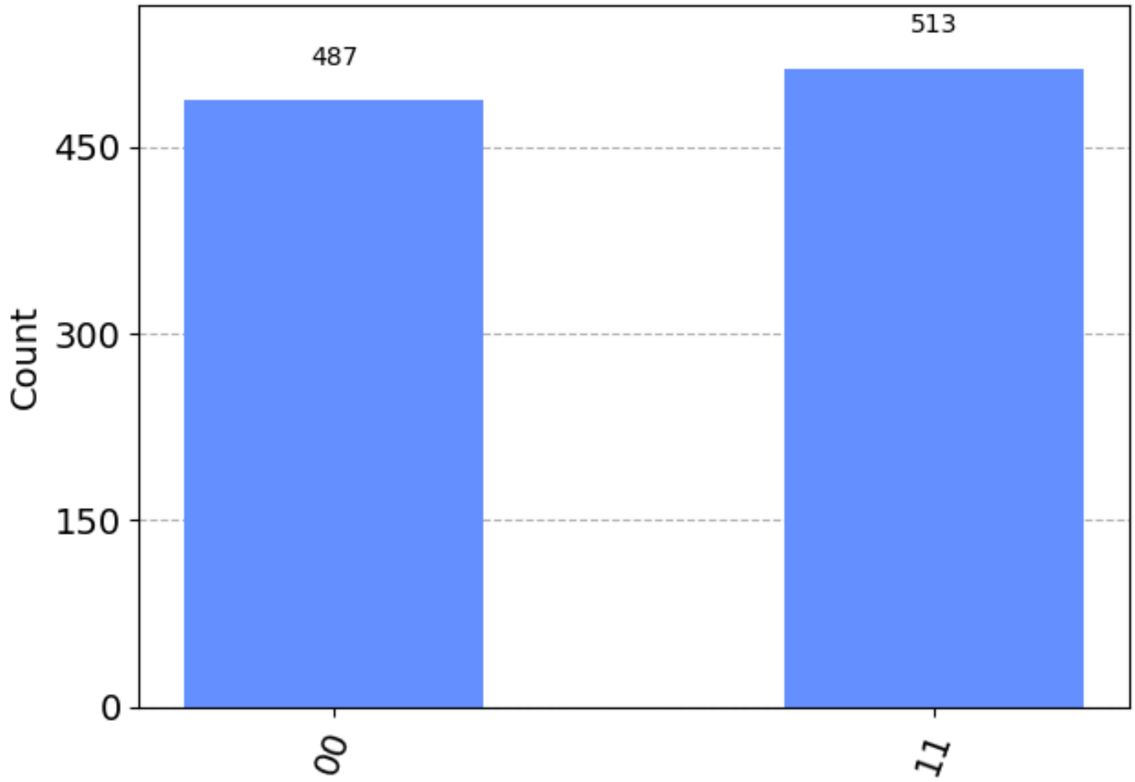
Out[14]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f0dd69d73d0>
```

In [12]:
```
# Execute
#--------

# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')

# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1000)

# Grab results from the job
result = job.result()

# Return counts
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)

# END
```

```
Total count for 00 and 11 are: {'00': 487, '11': 513}
```

In [16]:
```
# Visualize
#----------

# Import draw_circuit, then use it to draw the circuit
from ibm_quantum_widgets import draw_circuit
draw_circuit(circuit)

# Analyze
#--------

# Plot a histogram
plot_histogram(counts)

# END
```

Out[16]:



Exercici 4

Apartat 3 (explicació)

Com es pot veure en el jupyter notebook, obtenim 487 cops |00> i 513 cops |11>, de les 1000 execucions totals que s'han fet de l'algorisme.En percentatges seria 48,7% de les execucións obtenim |00> i 51,3% de les execucions, obtenm|11>. Més o menys es el mateix que a l'apartat anterior que sortia un 50 % de cada. Per tant si que surt pràcticament el mateix que en l'apartat anterior.

In [37]:
```
from ibm_quantum_widgets import CircuitComposer
from qiskit import QuantumCircuit, transpile, Aer, IBMQ, execute
from numpy import pi
from qiskit.tools.visualization import *

# Build
#------

# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)

# Add a H gate on qubit 0
circuit.h(0)

# Add a NOT gate on qubit 1
circuit.x(1)
# Add a H gate on qubit 1
circuit.h(1)

# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)

# Add a H gate on qubit 1
circuit.h(1)

# Add a H gate on qubit 0
circuit.h(0)

# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])

# END
```

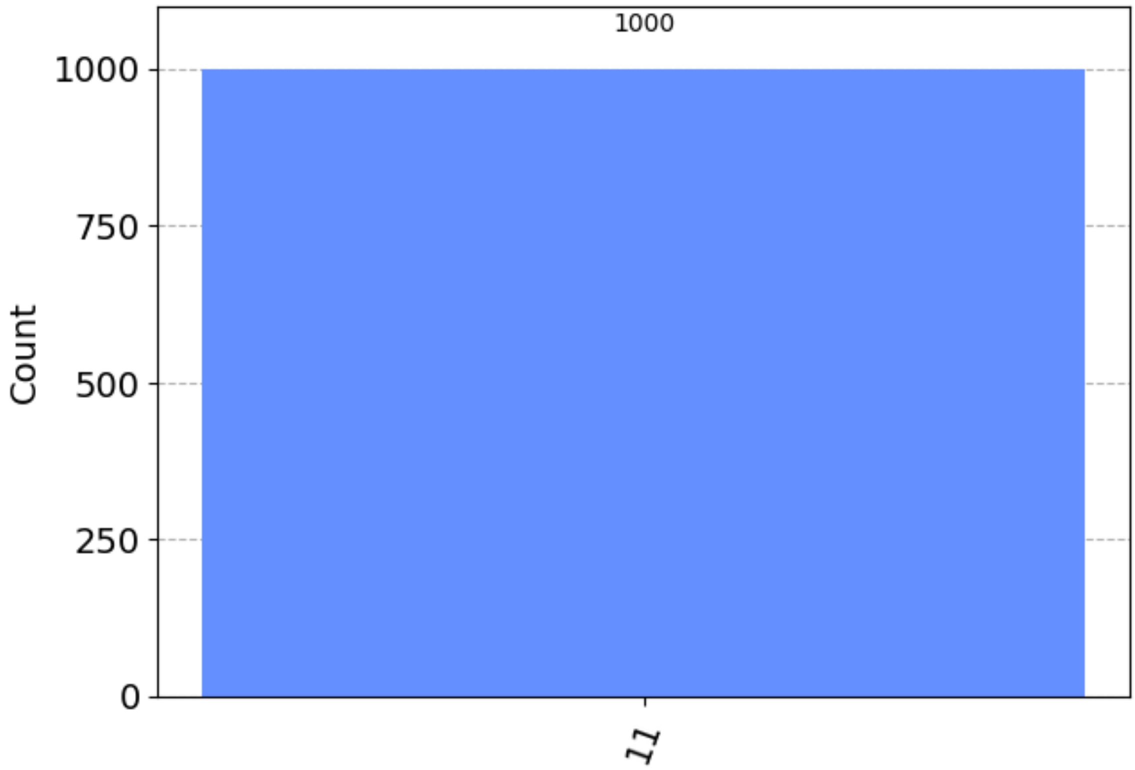
Out[37]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f0dd642b1f0>
```

In [38]:
```
# Execute
#--------

# Use Aer's qasm_simulator
simulator = Aer.get_backend('qasm_simulator')

# Execute the circuit on the qasm simulator
job = execute(circuit, simulator, shots=1000)

# Grab results from the job
result = job.result()

# Return counts
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)

# END
```

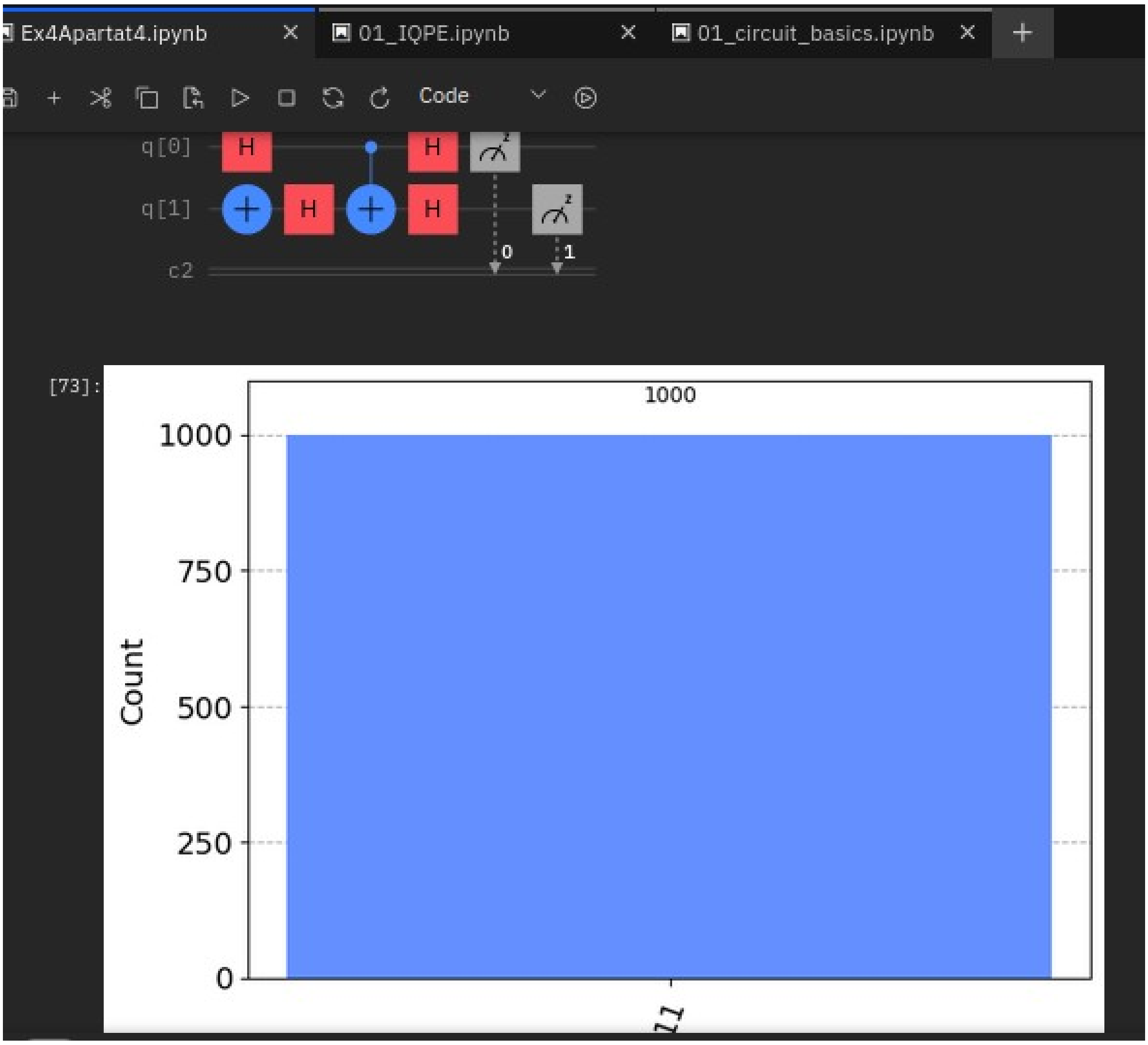
```
Total count for 00 and 11 are: {'11': 1000}
```

In [39]:
```
# Visualize
#----------

# Import draw_circuit, then use it to draw the circuit
from ibm_quantum_widgets import draw_circuit
draw_circuit(circuit)

# Analyze
#--------

# Plot a histogram
plot_histogram(counts)

# END
```

Out[39]:



Apartat 4 Ex 4



Com es pot observar en la imatge, a la part de sobre, es pot veure el dibuix de l'algoritme implementat, el qual correspón a l'lalgorisme de Deutch. Inicialment, els qubit 0 esta a 0 i el qubit 1 amb la porta not estará a 1.
La part de sota es pot veure el resultat de les 1000 execucions del circuit. Com es pot veure, totes les execucions han donat el mateix (1) i per tant, es una funció constant. Això es normal, ja que en el qubit 0 li passem un 0, per tant f(0) = 0 i en el qubit 1 li passem un 1 al fer la porta not al principi, i per tant estem indicant que f(1) = 1.

In [1]:
```
from ibm_quantum_widgets import CircuitComposer
from qiskit import QuantumCircuit, transpile, Aer, IBMQ, execute, Quantum
from numpy import pi
from qiskit.tools.visualization import *
import numpy as np

# importing Qiskit
from qiskit.providers.ibmq import least_busy

# Build
#------

# Create a Quantum Circuit acting on the q register
circuit = QuantumCircuit(2, 2)

# Add a H gate on qubit 0
circuit.h(0)

# Add a NOT gate on qubit 1
circuit.x(1)
# Add a H gate on qubit 1
circuit.h(1)

# Add a CX (CNOT) gate on control qubit 0 and target qubit 1
circuit.cx(0, 1)

# Add a H gate on qubit 1
circuit.h(1)

# Add a H gate on qubit 0
circuit.h(0)

# Map the quantum measurement to the classical bits
circuit.measure([0,1], [0,1])

# END
```

Out[1]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f1ffa6dec50>
```

In [2]:
```
n = 2
```

In [3]:
```
# Load our saved IBMQ accounts and get the least busy backend device with
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration(
                                   not x.configuration().simulator and x.
print("least busy backend: ", backend)
```

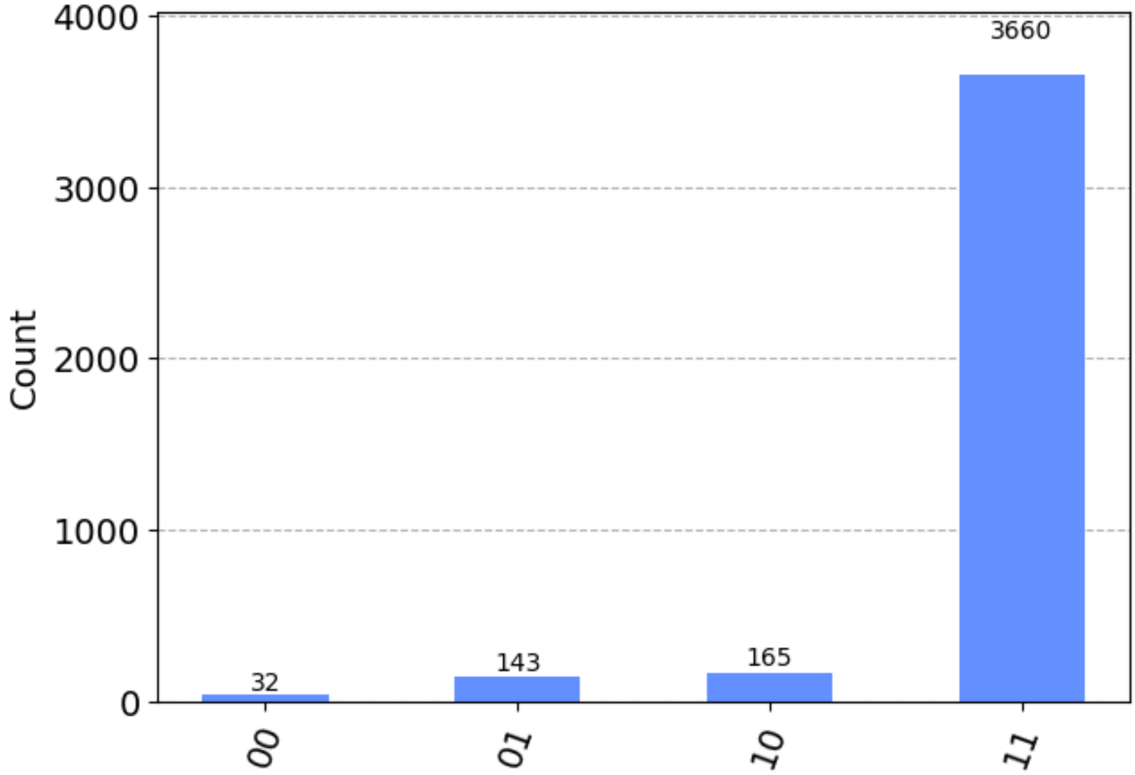
```
/tmp/ipykernel_192/2347876859.py:2: DeprecationWarning: The qiskit.IBMQ e
ntrypoint and the qiskit-ibmq-provider package (accessible from 'qiskit.p
roviders.ibmq`) are deprecated and will be removed in a future release. I
nstead you should use the qiskit-ibm-provider package which is accessible
from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_i
bm_provider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMPro
vider'
  IBMQ.load_account()
least busy backend:  ibmq_lima
```

In [4]:
```
# Run our circuit on the least busy backend. Monitor the execution of the
from qiskit.tools.monitor import job_monitor

transpiled_dj_circuit = transpile(circuit, backend, optimization_level=3)
job = backend.run(transpiled_dj_circuit)
job_monitor(job, interval=2)
```

```
Job Status: job has successfully run
```

In [5]:
```
# Get the results of the computation
results = job.result()
answer = results.get_counts()

plot_histogram(answer)
```

Out[5]:



Exercici 4

Apartat 5 (explicació)

Com es pot veure en l’execució del jupyter-notebook, s’ha obtingut la majoria de cops 1 (de les 4000 execucions, 3660 cops, el resultat és 1). 1 és el resultat esperat. Com que se suposa que Deutch-Jozsa és un algorisme determinista en el qual encerta el 100 % no té gaire sentit que no sigui cert segons els resultats, ja que surt pocs cops 0. Però mirant a la mateixa pàgina web, posa que els altres valors són deguts a errors de l’ordinador al fer els càlculs quàntics.