```python
# implementation of the DJ algorithm on the simulator and the Melbourne backend
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit.tools.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# if you don't have an account you must create one, get an API token, and enable
# or save the account.
# once the account is saved you just need to use load below.
# see Qiskit terra for instructions.

#IBMQ.save_account('my API token')

IBMQ.load_account()

# currently existing backends
#backend = IBMQ.get_backend('ibmq_16_melbourne')
#backend = IBMQ.get_backend('ibmqx4')
#backend = IBMQ.get_backend('ibmqx2')
#backend = IBMQ.get_backend('ibmq_gasm_simulator')

q = QuantumRegister(14)
c = ClassicalRegister(14)
DJ = QuantumCircuit(q, c)

# quantum circuit using qubits 2,3,4,11
# on melbourne for these qubits the coupling map is CX2-3, CX4-3, CX11-3
# we take a balanced function f(x2,x4,x11)= x2 + x4 + x11
# this is a linear function f = <a, x> and the algo can also be viewed as the Be
# rnstein-Vazirani algo to find the vector a

# preparation
DJ.x(q[3])
DJ.barrier(q)
DJ.h(q[2])
DJ.h(q[3])
DJ.h(q[4])
DJ.h(q[11])
DJ.barrier(q)

# function f (oracle)
DJ.cx(q[2], q[3])
DJ.cx(q[4], q[3])
DJ.cx(q[11], q[3])
DJ.barrier(q)

# fourier analysis
DJ.h(q[2])
DJ.h(q[4])
DJ.h(q[11])

# measurement of bits 2,4
DJ.barrier(q)
DJ.measure(q[2], c[2])
DJ.measure(q[4], c[4])
DJ.measure(q[11], c[11])
```

Out[3]: <qiskit.circuit.measure.Measure at 0x9e2704a8>
In [4]: #drawing the circuit for verification

DJ.draw()
```
q2_0: |0>  
q2_1: |0>  
q2_2: |0>  
q2_3: |0>  
q2_4: |0>  
q2_5: |0>  
q2_6: |0>  
q2_7: |0>  
q2_8: |0>  
q2_9: |0>  
q2_10: |0>  
q2_11: |0>  
q2_12: |0>  
q2_13: |0>  
c2_0: 0  
c2_1: 0  
c2_2: 0  
c2_3: 0  
c2_4: 0  
c2_5: 0  
c2_6: 0  
c2_7: 0  
c2_8: 0  
c2_9: 0  
c2_10: 0  
c2_11: 0  
c2_12: 0  
c2_13: 0
```
In [6]: #simulation - we find the vector $a_2=1$, $a_4=1$, $a_{11}=1$ and other components 0

    backend = Aer.get_backend('qasm_simulator')
    job_sim = execute(DJ, backend)
    sim_result = job_sim.result()

    print(sim_result.get_counts(DJ))
    plot_histogram(sim_result.get_counts(DJ))

    {'00100000010100': 1024}

Out[6]:

In [7]: #experiment
    #you can also check for availability and current parameters of the backend before calling it.
    #see instructions in Quiskit terra.

    backend = IBMQ.get_backend('ibmq_16_melbourne')  # circuit above respects constraints of melbourne device.
    shots = 1024  # Number of shots to run the program (experiment); maximum is 8192 shots.
    max_credits = 3  # Maximum number of credits to spend on executions.

    job_exp = execute(DJ, backend=backend, shots=shots, max_credits=max_credits)
    job_monitor(job_exp)
In [8]: #results of experiment
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(DJ)

print(result_exp.get_counts(DJ))
plot_histogram([counts_exp])

Out[8]:

{"00100000000000": 96, "00000000000100": 87, "00100000000100": 159, "00000000
10100": 86, "00000000010000": 125, "00100000100000": 240, "00100000101000": 12
0, "00000000000000": 111}
#second implementation of the DJ algorithm on the simulator and the Melbourne backend.

```python
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit import IBMQ
from qiskit.tools.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# if you dont have an account you must create one, get an API token, and enable or save the account.
# once the account is saved you just need to use load below.
# see Qiskit terra for instructions.

#IBMQ.save_account('my API token')

IBMQ.load_accounts()

# currently existing backends
# backend = IBMQ.get_backend('ibmq_16_melbourne')
# backend = IBMQ.get_backend('ibmqx4')
# backend = IBMQ.get_backend('ibmqx2')
# backend = IBMQ.get_backend('ibmq_gasm_simulator')

q = QuantumRegister(14)
c = ClassicalRegister(14)
DJ = QuantumCircuit(q, c)

# quantum circuit using qubits 6, 7, 8, 9
# on melbourne for these qubits the coupling map is CX6-8, CX9-8, CX7-8
# we take a balanced function f(x2, x4, x11) = x6 + x7 + x9
# again this is in fact a Bernstein-Vazirani version and the algo even returns 'a' in the linear function f=<a, x>.

# preparation
DJ.x(q[8])
DJ.barrier(q)
DJ.h(q[6])
DJ.h(q[7])
DJ.h(q[8])
DJ.h(q[9])
DJ.barrier(q)

# function f (oracle)
DJ.cx(q[6], q[8])
DJ.cx(q[7], q[8])
DJ.cx(q[9], q[8])
DJ.barrier(q)

# fourier analysis
DJ.h(q[6])
DJ.h(q[7])
DJ.h(q[9])

# measurement of bits 2, 4
DJ.barrier(q)
DJ.measure(q[6], c[6])
DJ.measure(q[7], c[7])
DJ.measure(q[9], c[9])
```
Out[9]: <qiskit.circuit.measure.Measure at 0xa1e226c50>
In [10]: # drawing the circuit for verification

DJ.draw()
In [11]:
#simulation finds vector a
backend = Aer.get_backend('qasm_simulator')
job_sim = execute(DJ, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(DJ))
plot_histogram(sim_result.get_counts(DJ))
{'00001011000000': 1024}

Out[11]:

In [12]:
#experiment
#you can also check for availability and current parameters of the backend before calling it.
#see instructions in Qiskit terra.
backend = IBMQ.get_backend('ibmq_16_melbourne')  # circuit above respects constraints of melbourne device.
shots = 1024  # Number of shots to run the program (experiment); maximum is 8192 shots.
max_credits = 3  # Maximum number of credits to spend on executions.

job_exp = execute(DJ, backend=backend, shots=shots, max_credits=max_credits)
job_monitor(job_exp)
In [13]:
#results of experiment

result_exp = job_exp.result()
counts_exp = result_exp.get_counts(DJ)

print(result_exp.get_counts(DJ))
plot_histogram([counts_exp])

{'00001010000000': 184, '00001000000000': 54, '00000010000000': 61, '00000000000000': 52, '0000000011000000': 116, '0000000010000000': 280, '00000000000000': 67}