Algorithm 1 Deutsch-Josza

Require: n length of binary string, $f(a)$ is blackbox that contains a constant or balanced boolean function

Ensure: oracle determines $f(a)$ type, returns true if constant false if balanced.

1: $|x\rangle^{\otimes n} |y\rangle \rightarrow |0\rangle^{\otimes n} |1\rangle$

2: $|x\rangle^{\otimes n} |y\rangle \rightarrow \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} |i\rangle (|0\rangle - |1\rangle) \rightarrow |+\rangle^{\otimes n} |-\rangle$

3: $|x\rangle^{\otimes n} |y\rangle \rightarrow |x\rangle^{\otimes n} |y \oplus f(x)\rangle \rightarrow \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} (-1)^{f(i)} |i\rangle (|0\rangle - |1\rangle)$

4: $|x\rangle^{\otimes n} \rightarrow \frac{1}{2^n} \sum_{j=0}^{2^n-1} [\sum_{i=0}^{2^n-1} (-1)^{i \cdot j + f(i)}] |j\rangle \rightarrow |\psi\rangle$

5: $result = |\langle \psi | 0 \rangle^{\otimes n}|^2$

6: **return** $true?(result == 1) else false$

EXPLANATION

1: We consider an n-qubit register x and a 1-qubit register y, denoted as: $|x\rangle^{\otimes n} |y\rangle$ where $|x\rangle^{\otimes n}$ is the collective expression for the tensor product of n qubits. Our compuation basis is the Z-basis which consists of eigen vectors $|0\rangle$ and $|1\rangle$. We initialize the two registers: first one all qubits are in the $|0\rangle$ state; the second one, it is in the $|1\rangle$ state.

2: Apply the Hadamard gate to all n + 1 qubits. Recall single qubit application: $H|0\rangle \rightarrow |+\rangle$ and $H|1\rangle \rightarrow |-\rangle$. H-gate allows us thus to put a qubit in supersposition. Now, the formulation to apply H-gate to n qubits expressed collectively is:

$$H(x, n) \rightarrow \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} (-1)^{x \cdot i} |i\rangle$$

where $|i\rangle$ is n-qubits and $i \cdot j$ is the bitwise inner product with i fixed and j varies over all combinations in the computational basis. For example, if we let n=2 and apply H to our first register $|00\rangle$:

$$H(|00\rangle, 2) \rightarrow \frac{1}{\sqrt{2^2}} \sum_{i=0}^{2^2-1} (-1)^{00 \cdot i} |i\rangle =$$

$$\frac{1}{2}([(-1)^{00 \cdot 00}] |00\rangle + [(-1)^{00 \cdot 01}] |01\rangle + [(-1)^{00 \cdot 10}] |10\rangle + [(-1)^{00 \cdot 11}] |11\rangle =$$

$$\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) =$$

$$\frac{1}{2}(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}) =$$

$$|++\rangle$$

This is because in the bitwise inner product, we interpret the values as bits, so for example $00 \cdot 00 \rightarrow 0\&0 \oplus 0\&0 = 0$, we AND the corresponding bits and XOR the pairs since

GF(2,+) is XOR. Hence all our coefficients go to +1. For our purposes, then we can get rid of the $(-1)^{i \cdot j}$ part and hence our formulation in line 2 for the first register is: $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$. For the above: i runs from 0 to 3, so 0,1,2,3 which map to 00,01,10,11 in our computational basis. In the inner product, we interpret as bits and in the ket we interpret as the tensor product. In the summations, we have the linear sum of the four 2-qubit tensor products. However, if we were to change our test and work with a different input, we may leave it depending on how our coefficients turn out and simplifications we make. If we consider say: $H(|11\rangle, 2)$ then, we must use the general formulation and we compute:

$H(x, n) = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} (-1)^{x \cdot i} |i\rangle =$

$H(|11\rangle, 2) = \frac{1}{2}([(-1)^{11 \cdot 00}] |00\rangle + [(-1)^{11 \cdot 01}] |01\rangle + [(-1)^{11 \cdot 10}] |10\rangle + [(-1)^{11 \cdot 10} |11\rangle =$

$\frac{1}{2}(|00\rangle + (-1) |01\rangle + (-1) |10\rangle + (-1) |11\rangle) \rightarrow |--\rangle$

In the algorithm, we have the $(|0\rangle - |1\rangle)$ to account for the one qubit $|y\rangle$, so our formulation is as: $\frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} |i\rangle (|0\rangle - |1\rangle)$. Which maps our registers from $|0\rangle^{\otimes n} |1\rangle$ to $|+\rangle^{\otimes n} |-\rangle$.

3: In this step we apply the hidden function or oracle, our unitary transformation that maps $|x\rangle^{\otimes n} |y\rangle \rightarrow |x\rangle^{\otimes n} |y \oplus f(x)\rangle$. The second register is required in order to compute with a reversible unitary. In the unitary, we generate the string at random.. for our purposes. All we care about is n and the unitary. There is not much to reformulate from line 2 to 3: $|x\rangle^{\otimes n} |y \oplus f(x)\rangle \rightarrow \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} (-1)^{f(i)} |i\rangle (|0\rangle - |1\rangle)$. Since $a \oplus 0 = a$ and $a \oplus 1 = \overline{a}$, we can express $|y \oplus f(x)\rangle$ as $(|0\rangle - |1\rangle) \oplus f(x)$ since we factor our $\frac{1}{\sqrt{2}}$. Then, $(|0\rangle - |1\rangle) \oplus f(x) \rightarrow (|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle \rightarrow (|f(x)\rangle - |1 \oplus f(x)\rangle$. Since $f(x)$ outputs 0 or 1, $(|f(0)\rangle - |1 \oplus f(0)\rangle \rightarrow (|0\rangle - |1\rangle)$ and $(|f(1)\rangle - |1 \oplus f(1)\rangle \rightarrow (|1\rangle - |0\rangle)$ or $-(|0\rangle - |1\rangle)$. We can account for $f(x)$ added phase by $(-1)^{f(x)}$. As you can see, we are able to evaluate the function in parallel. Recall that a qubit in supersposition can be in a continuum of different states and that a system of n qubits can "exist in any superposition of the 2^n basis states". However, for ours purposes, we only care about the different input states in the classical regime. The tensor product allows us to evaluate $f(i)$ a total of 2^n times in one run, which is contained in our resulting state of step 3 ("the state space of a composite quantum system is the tensor product of the state spaces of its subsystems").

4: Here we apply Hadamard gate to only the first register which allows us to measure in it later in the computational basis. For this, the second register is now irrelevant. Recall HH= I, H is its own inverse. If f constant, a second application of H should allows us to measure a probability that the final state is in $|+\rangle^{\otimes n} \rightarrow |0\rangle^{\otimes n}$ of 1. For n=2 we have:

$\mathrm{H}(|++\rangle, 2) = \sum_{j=0}^{2^n-1} \sum_{i=0}^{2^n-1} (-1)^{i \cdot j} (-1)^{f(i)} |j\rangle$

assuming $f(x)$ is constant and outputs 0 for all inputs, then our formulation simplifies to:

$\sum_{j=0}^{2^n-1} \sum_{i=0}^{2^n-1} (-1)^{i \cdot j} |j\rangle =$

$\frac{1}{4}([(-1)^{00 \cdot 00} + (-1)^{00 \cdot 01} + (-1)^{00 \cdot 10} + (-1)^{00 \cdot 11}] |00\rangle + [(-1)^{01 \cdot 00} + (-1)^{01 \cdot 01} + (-1)^{01 \cdot 10} + (-1)^{01 \cdot 11}] |01\rangle + [(-1)^{10 \cdot 00} + (-1)^{10 \cdot 01} + (-1)^{10 \cdot 10} + (-1)^{10 \cdot 11}] |10\rangle + [(-1)^{11 \cdot 00} + (-1)^{11 \cdot 01} + (-1)^{11 \cdot 10} + (-1)^{11 \cdot 11}] |11\rangle) =$

$$\frac{1}{4}(4 |00\rangle + 0 |01\rangle - 0 |10\rangle + 0 |11\rangle) = |00\rangle - |10\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle.$$

5: Here we would measure the probability that the first register is in state we initialized it to in step 1: $|00\rangle^{\otimes n}$ and determine if the hidden function is constant or balanced. Because for constant, the function may output all 0s or all 1s, the coefficient for the first TERM in the formulation of line 4 would be ± 1 and 0 for all the rest. In the computation above for n=2, we can see clearly that the term $|00\rangle$ ends up with a +1 coefficient since we assumed $f(i) = 0 \forall i$ and we can see the rest have a coefficient of 0. If we assumed a balanced function which always outputs a 1, then the coefficient for the first term would be -1 and 0 for all others. For a balanced function which outputs 0s for half the bits and 1s for the other half, the the coefficient for the first term would be 0... and hence orthogonal to the initial state $|00\rangle^{\otimes n}$, so the probability that that final state vector is in the initialized value would clearly be 0. This is the interference effect in which wrong results cancel quickly while correct results are reinforced.

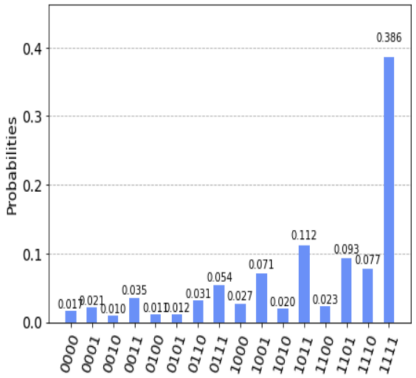
The way measurement is done... we compute the square of the absolute value of the inner product of the actual final state with the expected state. When we measure, we compute the probability over the possible classical values in the computational basis.

6: We return our determined corresponding to the type of the hidden function.

In the classical this regime this pruoblem has different approaches. We can solve with at least two queries for a balanced boolean function. However, for constant, it's another story. We can try inspecting bits at random and query $O(\frac{1}{\epsilon})$; if the result is the same after two queries though... we can truncate the algorithm early to determine the type with a % level of confidence. We also compute the probability that it is constant. However, to obtain 100% confident answer, we need $2^{n-1} + 1$ queries. So we can appreciate the power of computing computing in this beautiful algorithm.... with just one run. Below you can see the output probabilities for a small system using IBM jupyter notebook and python running on real quantum hardware:

```
# Experiment with Real Devices
# Load our saved IBMQ accounts and get the least busy backend device with greater than or equal to (n+1) qubits
API_TOKEN=[redacted]
#https://quantum-computing.ibm.com/lab/docs/iql/manage/account/ibmq
IBMQ.save_account(API_TOKEN,overwrite=True)
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= (n+1) and not x.configuration().simulator and x.status().operational==True))
print("least busy backend: ", backend)
shots = 1024
transpiled_dj_circuit = transpile(dj_circuit, backend, optimization_level=3)
qobj = assemble(transpiled_dj_circuit, backend)
job = backend.run(qobj)
job_monitor(job, interval=2)
# Get the results of the computation
results = job.result()
answer = results.get_counts()
plot_histogram(answer)
#the most likely result is 1111. The other results are due to errors in the quantum computation.
```

```
ibmqfactory.load_account:WARNING:2021-05-24 09:52:54,153: Credentials are already in use. The existing account in the session will be replaced.
least busy backend:  ibmq_manila
/Users/samiracarolinaolivamadrigal/Library/Python/3.7/lib/python/site-packages/ipykernel_launcher.py:13: DeprecationWarning: Passing a Qobj to Backend.run is deprecated and w
ill be removed in a future release. Please pass in circuits or pulse schedules instead.
  del sys.path[0]
Job Status: job has successfully run
```



REFERENCES

https://www.sciencedirect.com/topics/engineering/boolean-function
https://www.sciencedirect.com/topics/engineering/quantum-parallelism
https://www.sciencedirect.com/topics/engineering/entangled-state
https://www.sciencedirect.com/topics/physics-and-astronomy/quantum-entanglement
https://indico.cern.ch/event/970906/attachments/2151237/3627130/PIQC%20Lecture%204.pdf
https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html#4.1-Constant-Oracle-
https://qiskit.org/textbook/what-is-quantum.html#Explaining-the-Double-Quantum-Coin-Toss
https://www.cl.cam.ac.uk/teaching/0910/QuantComp/notes.pdf