Deutsch Algorithm

Deutsch's Algorithm was the first algorithm to prove that quantum computers can perform some tasks faster than classical ones, although the speedup in this case is a trivial one.

The problem looks somewhat like this :-

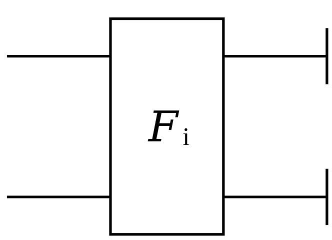
```
Consider 4 functions f₀, f₁, f₂ and f₃. Each of them can take 0 or 1 as input.

f₀(0) = 0, f₀(1) = 0 -> Constant function
f₁(0) = 0, f₁(1) = 1 -> Balanced function
f₂(0) = 1, f₂(1) = 0 -> Balanced function
f₃(0) = 1, f₃(1) = 1 -> Constant function

Given one of these four functions at random, how many evaluations must we make to co
nfirm whether the given function is balanced or constant.
```

Classically, it takes at least two evaluations to confirm whether the given function is balanced or constant.

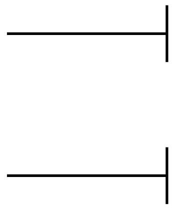
Using quantum computers, we can confirm whether the given function is balanced or not using one evaluation.

First we construct gates that correspond to the 4 functions. Here, let's consider a circuit, which takes the qubits $|x\rangle$ and $|y\rangle$, and returns the qubits $|x\rangle$ and $|y \oplus f_i(x)\rangle$ respectively, where i can be any random number between 0 to 3. Let's call this circuit F_i .

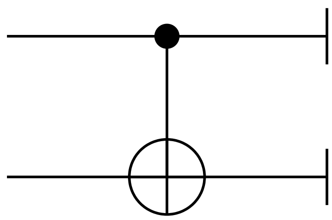
```
using Yao, YaoPlots
```



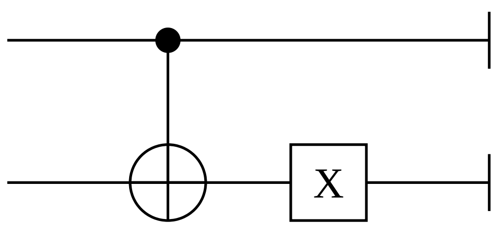
Below is my implementation of F_i



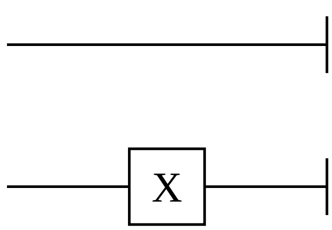
```
begin
    F₀ = chain(2)
    plot(F₀)
end
```



```
begin
    F₁ = chain(2, control(1, 2=>X))
    plot(F₁)
end
```




```
begin
    F₂ = chain(2, control(1, 2=>X), put(2=>X))
    plot(F₂)
end
```




```
begin
    F₃ = chain(2, put(2=>X))
    plot(F₃)
end
```

```
circuit = 1×4 Array{ChainBlock{2},2}:
           nqubits: 2
          chain
            …  nqubits: 2
          chain
          └─ put on (2)
             └─ X
```

```
circuit = [F₀ F₁ F₂ F₃]
```

```
r = 2
```

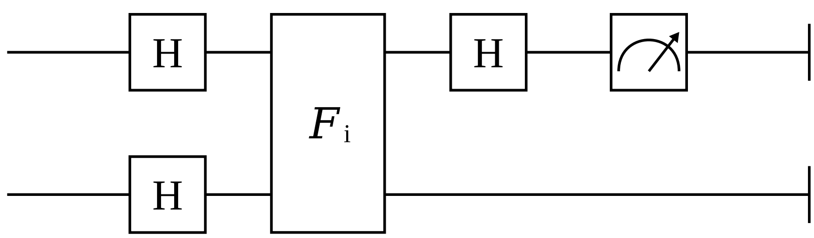
```
r = rand(0:3)
```

The new question now is :-

> Given one of the four circuits at random, how many evaluations would it take to find whether the underlying function is balanced or constant?

If, in the above circuits we pass 0 or 1, it'll be the same as classical computers - it'll take 2 evaluations. To achieve this, we'll pass a superposition of 0 and 1, to the F_i circuit, and then we'll pass an H gate to the first qubit and measure it.

The final circuit for Deutsch Algorithm where the input is $|01\rangle$ looks somewhat like this



Below is the final implementation, using the F_i we constructed before.

```
BitBasis.BitStr{1,Int64}[1 ₍₂₎]
```

```
begin
    input = ArrayReg(bit"10") #Remember, the circuit takes the qubits in reverse order
    result = input |> chain(2, repeat(H,1:2), put(1:2=>circuit[r+1]), put(1=>H)) |> r->measure(r,1)
end
```

```
"Balanced"
```

```
result[1] == bit"1" ? "Balanced" : "Constant"
```

As you can see from the value of r, the result is correct.

Deutsch–Jozsa Algorithm

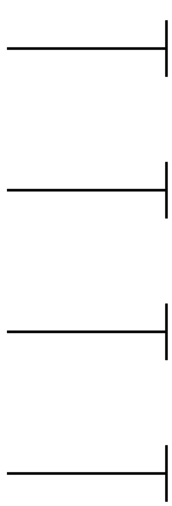
The Deutsch-Jozsa Algorithm is the *general* version of the Deutsch Algorithm. The problem is : –

```
The functions are now of n variables. The inputs to each of these n variables can ei
ther be 0 or 1. So can be the output. The function can either be constant, where all
the inputs get sent to 0 or all the inputs get sent to 1, or balanced, where half th
e inputs get sent to 0 and the rest get sent to 1.
To illustrate this, let's take an example of 4 qubits. Which means 2^4 possible input
s, as each input can either be 0 or 1.
Let's check each input -
(0,0,0,0)   (1,1,1,1)
(0,0,0,1)   (1,1,1,0)
(0,0,1,0)   (1,1,0,1)
(0,1,0,0)   (1,0,1,1)
(1,0,0,0)   (0,1,1,1)
(0,0,1,1)   (1,1,0,0)
(1,1,0,0)   (0,0,1,1)
(0,1,1,0)   (1,0,0,1)
So if f(0,0,0,0) = 0, f(all 2^4 combinations) = 0
or f(0,0,0,0) = 1, f(all 2^4 combinations) = 1
The function is constant. Else, it's balanced.
```

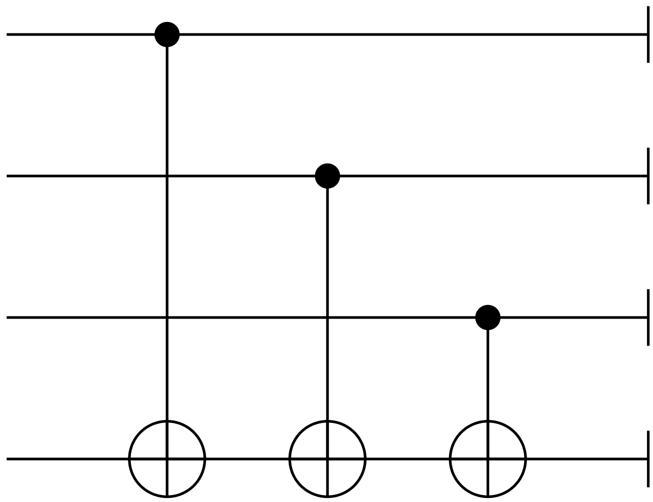
Deutsch Algorithm is a case of Deutsch-Jozsa Algorithm, where the input is 1 qubit. Drawing inspiration from our above circuit, let's make a circuit for n inputs.

n = 3

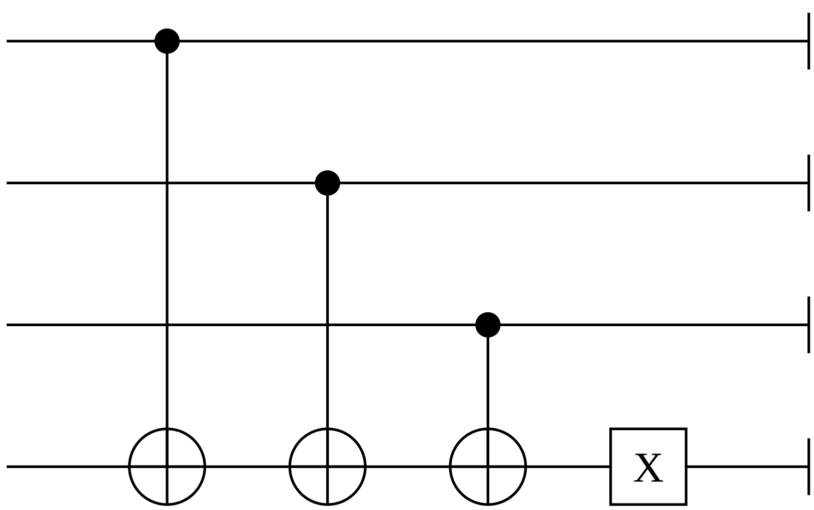
```
n = 3
```



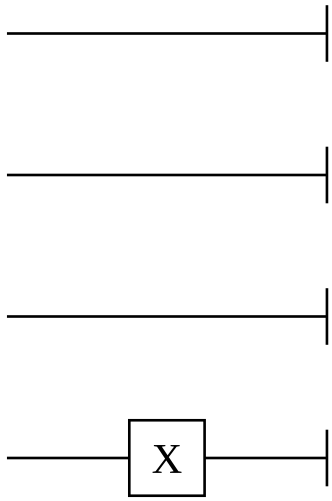
```julia
begin
    ℱ₀ = chain(n+1)
    plot(ℱ₀)
end
```



```julia
begin
    ℱ₁ = chain(n+1, [control(k, n+1=>X) for k in 1:n])
    plot(ℱ₁)
end
```




```julia
begin
    ℱ₂ = chain(n+1, chain(n+1, [control(k, n+1=>X) for k in 1:n]), put(n+1=>X))
    plot(ℱ₂)
end
```




```
begin
    ℱ₃ = chain(n+1, put(n+1=>X))
    plot(ℱ₃)
end
```

```
ℱ = 1×4 Array{ChainBlock{4},2}:
     nqubits: 4
    chain
      …  nqubits: 4
    chain
    └─ put on (4)
       └─ X
```

```
ℱ = [ℱ₀ ℱ₁ ℱ₂ ℱ₃]
```

```
newr = 0
```

```
newr = rand(0:3)
```

```
▸ BitBasis.BitStr{3,Int64}[000 ₍₂₎]
```

```
begin
    i = join(ArrayReg(bit"1"), zero_state(n))
    output = i |> chain(n+1, repeat(H, 1:n+1), put(1:n+1=>ℱ[newr+1]), repeat(H,1:n))
|> r->measure(r,1:n)
end
```

If output of the measured qubits is 000, then the function is constant, else if, the output is 111, the function is balanced

```
"Constant"
```

```
output == measure(zero_state(n)) ? "Constant" : "Balanced"
```

Please note, you can change the value of n above to get the Deutsch-Josza Algorithm for different inputs. $n = 1$ will give the Deutsch Algorithm.