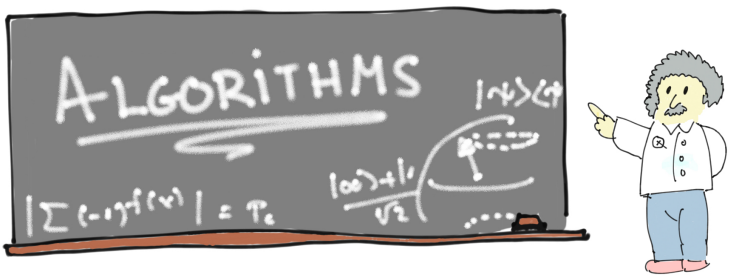

Algorithms: Quantum Counting [400 points]

Version: 1

Algorithms

The **Algorithms** category challenges will serve to give you a sense of commonly-used routines that are helpful when using real quantum computers. The quantum algorithms that you will see in this category are procedures that mathematically demonstrate some advantage over traditional/classical solutions such as the Deutsch-Jozsa algorithm, applications of the Quantum Fourier Transform (QFT), Grover's algorithm, and Quantum Phase Estimation (QPE).

Although these algorithms have very specific applications, they are commonly related to real-world problems. One of the clearest examples of this is the well-known Shor factorization algorithm, which revealed that finding prime factors of a number is equivalent to finding the period of certain functions, which can be achieved through QPE. It is for this reason that it is so important to know these basic ideas of quantum computation. Let's get to it! And good luck!

Problem statement [400 points]

The quantum counting algorithm is a variation of Grover's search. In this case, instead of having a function

$$f : \{0, \ldots N-1\} \to \{0, 1\}$$

that takes a value of 1 only for one element in the domain, it may take the value 1 for arbitrarily many elements. Our objective is to find how many values x there are such that $f(x) = 1$. This can be approximately done via the quantum counting algorithm, which can be implemented by following the steps enumerated below.

1. Build the Grover operator $G = DU$ for *four* qubits, where U is the oracle:

$$U|x\rangle = (-1)^{f(x)}\,|x\rangle$$

and D is the diffusion operator. Write a function `oracle_matrix` that takes the list of elements x of $\{0, \dots N-1\}$ such that $f(x) = 1$ and returns the oracle U. Use it to build the Grover operator G for 4 qubits. The diffusion matrix for 4 qubits is already implemented for you in the `diffusion_matrix()` function.

2. Feed the Grover operator G into the Quantum Phase Estimation algorithm. Use four target and four estimation qubits. Write a circuit that prepares the target qubits in the $|+\rangle$ state and then implements the QPE algorithm with G as the unitary. The output should be the probabilities of measuring each computational basis state.

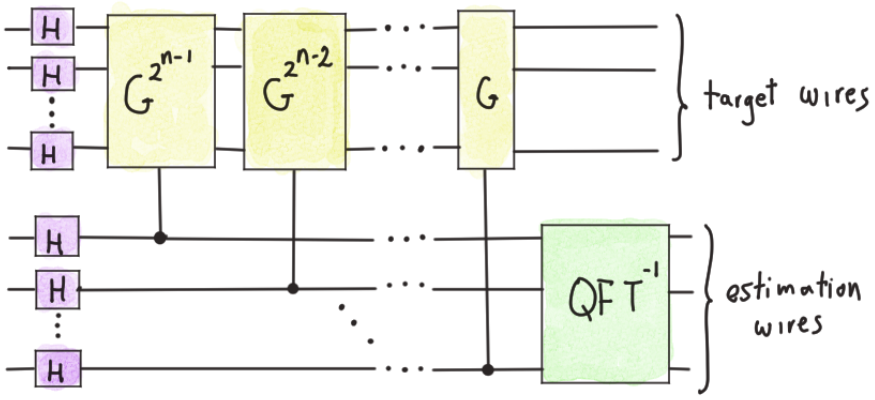

Figure 1: The circuit you must construct

3. Out of the measured probabilities on the estimation wires, select the wire that has the maximum probability and map it to its decimal representation. For example, if the state is $|0101\rangle$, we read 0101 as a binary number and turn it into a decimal, which is 5. Define $\theta =$ decimal value $\times \frac{\pi}{8}$. Build a function that returns the (approximate) number of solutions

$$M = 16 \sin^2\left(\frac{\theta}{2}\right).$$

The provided template `quantum_counting_template.py` contains a few functions that you need to complete:

- `oracle_matrix`: returns the matrix representation of the oracle U.
- `circuit`: this is where you must construct the circuit in Figure 1. It returns `qml.probs(estimation_wires)`.
- `number_of_solutions`: this is where you must calculate the approximate number of solutions given by M.

- `relative_error`: the number of elements x such that $f(x) = 1$, as calculated by the quantum counting algorithm, is an estimate of the actual answer. Thus, you are asked to calculate the relative percentage error

$$E = \frac{\text{quantum counting estimation} - \text{true number of elements}}{\text{true number of elements}} \times 100\%.$$

  While this error is large in some cases, it can be reduced by increasing the number of estimation wires.

Input

- `list(int)`: A list of elements x such that $f(x) = 1$.

Output

- `float`: The percentage error in the quantum counting estimation of the number of elements x such that $f(x) = 1$.

Acceptance Criteria

In order for your submission to be judged as "correct":

- The outputs generated by your solution when run with a given `.in` file must match those in the corresponding `.ans` file.
- Your solution must take no longer than the `30s` specified below to produce its outputs.

You can test your solution by passing the `#.in` input data to your program as stdin and comparing the output to the corresponding `#.ans` file:

```
python3 {name_of_file}.py < 1.in
```

WARNING: Don't modify the code outside of the `# QHACK #` markers in the template file, as this code is needed to test your solution. Do not add any print statements to your solution, as this will cause your submission to fail.

Specs
Time limit: **30 s**

Version History

Version 1: Initial document.