Hints for Challenge 4c

QFT Adder

The quantum addition with the Quantum Fourier Transform was first proposed by Draper. (see [Addition on a quantum computer (http://arxiv.org/abs/quant-ph/0008033)](http://arxiv.org/abs/quant-ph/0008033))

As also shown in the paper: [Quantum arithmetic with the Quantum Fourier Transform (https://arxiv.org/abs/1411.5949)](https://arxiv.org/abs/1411.5949) by Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin, QFT adder uses QFT to compute the addition of two integers in the "phase" basis. This consists of the following three steps.

1. Apply QFT on the output register
2. Apply phase rotation on the output register according to the integer to be added
3. Apply inverse QFT on the output register

Let's look at the details of each step. Assuming two inputs $|x\rangle = |x_0 \dots x_{n-1}\rangle$ and $|y\rangle = |y_0 \dots y_{n-1}\rangle$ with n qubits, we want to compute the following addition.

$$|x\rangle|y\rangle \rightarrow |x\rangle|x + y\ (mod\ 2^n)\rangle$$

The key operator is the "controlled" phase operation

$$CP|x\rangle|y\rangle = e^{i\frac{2\pi xy}{2^n}}|x\rangle|y\rangle.$$

(Note that when $n = 1$ and rotation angle is π, CP corresponds to the two-qubit gate, CZ.)

In addition, the QFT and IQFT operation can be respectively represented as

$$|a\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{i\frac{2\pi ak}{2^n}}|k\rangle \xrightarrow{\text{IQFT}} |a\rangle.$$

In step 1, the register $|y\rangle$ is converted into the phase basis encoding.

$$|x\rangle|y\rangle \xrightarrow{\text{QFT}_y} \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{i\frac{2\pi yk}{2^n}}|x\rangle|k\rangle$$

Next, in step 2, we want to change the coefficient of the phase basis $|k\rangle$ from $e^{i\frac{2\pi yk}{2^n}}$ to $e^{i\frac{2\pi(x+y)k}{2^n}}$.

This operation can be realized with CP as mentioned above.

$$\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{i\frac{2\pi yk}{2^n}}|x\rangle|k\rangle \xrightarrow{\text{CP}} \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{i\frac{2\pi yk}{2^n}}e^{i\frac{2\pi xk}{2^n}}|x\rangle|k\rangle = \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{i\frac{2\pi(x+y)k}{2^n}}|x\rangle|k\rangle$$

Finally, in step 3, the IQFT takes the result back into the computational basis:

$$\frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{i\frac{2\pi(x+y)k}{2^n}}|x\rangle|k\rangle \xrightarrow{\text{IQFT}_y} |x\rangle|x + y\ (mod\ 2^n)\rangle.$$

Since the (I)QFT operations are well prepared in Qiskit library, the only thing we have to design is the CP operation.

To realize this operation, we can sequentially apply controlled phase rotation gates according to the input integer x.

Besides, since we are adding the constant values to $|y\rangle$ in the problem 4c, we can hold the constant value as a classical data, which would significantly save the number of qubits that is required for $|x\rangle$.

Constraint Testing

For the constraint testing part, we are to realize the following state.

$$\left(\sum_{x\in\{0,1\}^t} |x\rangle_{index} |\cos t(x)\rangle_{data}\right)|0\rangle_{flag}$$
$$\xrightarrow{\text{constraint\_testing}} \sum_{x\in\{0,1\}^t} |x\rangle_{index} |\operatorname{cost}(x)\rangle_{data} |\operatorname{cost}(x) \geq C_{\max}\rangle_{flag}$$

To selectively set the flag register for those indices with costs larger than C_{max}, we can use the overflow of the most significant digit. Let c be the number of digits for the largest $\operatorname{cost}(x)$ in the data qubits. If we want to check whether $\operatorname{cost}(x)$ exceeds the value of C_{max}, we can add a appropriate constant value to the data register and see whether the carry occurs on the most significant digit. This step requires an additional qubit to store the carry. That's why the default number of qubits for data register is set to

```
data_qubits = math.ceil(math.log(max_c, 2)) + 1 if not max_c & (max_c - 1) == 0 else math.
ceil(math.log(max_c, 2)) + 2
```

Penalty Dephasing

In this section, we expect participants to implement `penalty_dephasing` function without explicitly using C_{max}. In the original paper, they set $2^c = C_{max} + w$ in constraint testing process, and here in penalty dephasing part, they are still using the same settings. (That is, using w to lift the C_{max} up to 2^c.) Since we have already computed the value of c in the previous process which can be included in the qubit size of data register, we do not have to compute c here again. Of course, C_{max} is available in the `penalty_dephasing` function since C_{max} is defined globally in the `solver_function` .