QHack

Quantum Coding Challenges

🚀 **CHALLENGE COMPLETED** [View successful submissions]

Jump to code — Collapse text

The Itch to Switch

500 points

Backstory

Zenda and Reece have been busy photocopying qubits and making their old communication protocols coherent. Zenda asks Trine what this has to do with timbits. Trine replies: "*Timbits? I forgot all about them. I suppose I wanted to show you there is more in heaven and earth than qubits and entangled pairs!*" Reece objects: "*But why did you get us to do all those protocols with photocopiers?*" Trine looks confused for a moment, then a smile spreads over her face. "*That's right! We can use them to implement a SWAP gate using two CNOTs as opposed to the usual three. Let's do that as a warm-up for timbits!*"

Exchanging qubits

Did you know that there is no way for us to clone a quantum state? The no-cloning theorem states that there is no gate U such that

$$U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$$

for all states $|\psi\rangle$. However, if we only work with basis states $|j\rangle$, there exist operations such that

$$|j\rangle|0\rangle \mapsto |j\rangle|j\rangle.$$

Zenda and Reece are each in possession of one basis state, which we denote $|j\rangle_{Z_0}$ and $|k\rangle_{R_0}$ respectively. Trine tells them to send each other their basis state to each other without losing their own. *"If basis states can be cloned, then surely we can do this"*, claims Zenda confidently. *"Just give us two qubits in the $|0\rangle$ state to each of us and we're good to go."*

Trine thinks about this... *"It's too easy if I allow you to do whatever you want"*—she concludes. *"Let's make it more fun. I'll give you each one qubit from a Bell state*

$$|\Phi\rangle_{Z_1R_1} = \frac{1}{\sqrt{2}}(|0\rangle_{Z_1}|0\rangle_{R_1} + |1\rangle_{Z_1}|1\rangle_{R_1}).$$

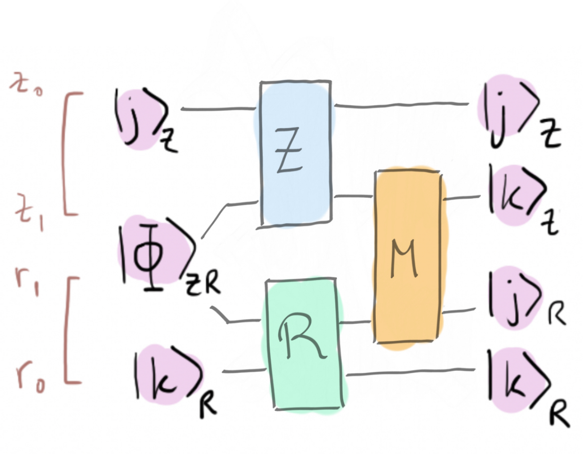
"Then you'll have to send your qubit to each other by acting only on the qubits in your possession."

Zenda and Reece try and try, but it seems like a futile task. *"We need more resources*—mumbles Reece. *"Mmm... disappointing"* says Trine. *"Then, I'll allow you to use a magic gate between your initially entangled qubits, but figure it out fast!"*

In this challenge, you will help Zenda and Reece figure out a quantum circuit that performs the operation

$$|j\rangle_{Z_0}|\Phi\rangle_{Z_1R_1}|k\rangle_{R_0} \mapsto |j\rangle_{Z_0}|k\rangle_{Z_1}|j\rangle_{R_1}|k\rangle_{R_0}$$

with the constraints imposed by Trine. This means that the circuit must be of the form shown in the image below.



In the above, Z is the operator Zenda applies on her qubits, R is the operator Reece applies on his qubits, and M is the magic operator provided by Trine. This operation is one of the building blocks you need to master to build a SWAP gate with only two CNOTs, without counting the distributed CNOTs (contained in the magic gate). See the optional reading below for more about this!

▶ **Laws of Infodynamics Part V: From three to two CNOTs**

▶ **Epilogue**

Challenge code

In the code below, you are given a number of functions:

- `zenda_operator`: Quantum function corresponding to the operator to be applied by Zenda on her qubits. **You must complete this function.**
- `reece_operator`: Quantum function corresponding to the operator to be applied by Reece on his qubits. **You must complete this function.**
- `magic_operator`: The magic operator provided by Trine to be applied on the initially entangled qubits Z_1 and R_1. **You must complete this function.**

Inputs and outputs

There are no inputs nor outputs for this challenge. You answer will be judged based on the fact that your circuit produces the correct final state for any combination of basis states $|j\rangle_{Z_0}$ and $|k\rangle_{R_0}$. This will be verified in the `check` function.

Code

Help

```python
import json
import pennylane as qml
import pennylane.numpy as np

def zenda_operator():
    """
    Quantum function corresponding to the operator to be applied
    Zenda in her qubits.This function does not return anything,
    you must simply write the necessary gates.
    """


    # Put your code here #


def reece_operator():
    """
    Quantum function corresponding to the operator to be applied
    Reece in his qubits.This function does not return anything,
    you must simply write the necessary gates.
    """


    # Put your code here #


def magic_operator():
    """
    Quantum function corresponding to the operator to be applied
    and "r1" qubits. This function does not return anything, you
    simply write the necessary gates.

    """


    # Put your code here #

```

```python
def bell_generator():
    """
    Quantum function preparing bell state shared by Reece and Ze
    """

    qml.Hadamard(wires=["z1"])
    qml.CNOT(wires=["z1", "r1"])


dev = qml.device("default.qubit", wires=["z0", "z1", "r1", "r0"]

@qml.qnode(dev)
def circuit(j, k):
    bell_generator()

    # j encoding and Zenda operation
    qml.BasisEmbedding([j], wires="z0")
    zenda_operator()

    # k encoding and Reece operation
    qml.BasisEmbedding([k], wires="r0")
    reece_operator()

    magic_operator()

    return qml.probs(wires=dev.wires)

```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:
    return None


def check(solution_output: str, expected_output: str) -> None:

    try:
        dev1 = qml.device("default.qubit", wires = ["z0", "z1"])
        @qml.qnode(dev1)
        def circuit1():
            zenda_operator()
            return qml.probs(dev1.wires)
        circuit1()
    except:
        assert False, "zenda_operator can only act on z0 and z1

    try:
        dev2 = qml.device("default.qubit", wires = ["r0", "r1"])
        @qml.qnode(dev2)
        def circuit2():
            reece_operator()
            return qml.probs(dev2.wires)
        circuit2()
    except:
        assert False, "reece_operator can only act on r0 and r1
    try:
        dev3 = qml.device("default.qubit", wires = ["z1", "r1"])
        @qml.qnode(dev3)
        def circuit3():
            magic_operator()
            return qml.probs(dev3.wires)
        circuit3()
    except:
        assert False, "magic_operator can only act on r1 and z1


    for j in range(2):
        for k in range(2):
            assert np.isclose(circuit(j, k)[10 * j + 5 * k], 1),

```

```python
test_cases = [['No input', 'No output']]
```

```python
for i, (input_, expected_output) in enumerate(test_cases):
    print(f"Running test case {i} with input '{input_}'...")

    try:
        output = run(input_)

    except Exception as exc:
        print(f"Runtime Error. {exc}")

    else:
        if message := check(output, expected_output):
            print(f"Wrong Answer. Have: '{output}'. Want: '{expe

        else:
            print("Correct!")
```

Copy all

Submit

Open Notebook

Reset