QHack

Quantum Coding Challenges

CHALLENGE COMPLETED [View successful submissions]

Jump to code — Collapse text

The Lazy Colleague

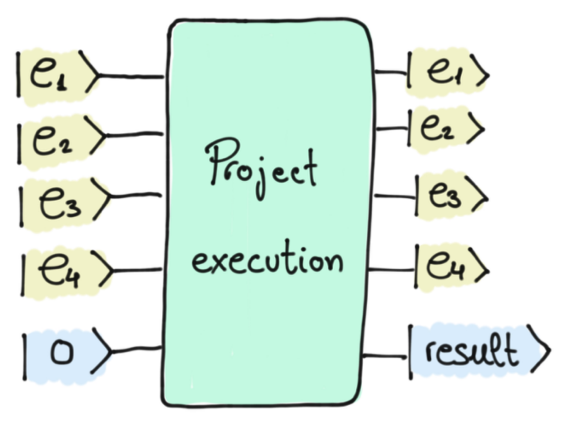
400 points

Backstory

It is very common to work in teams, but it is just as common to find a teammate who decides not to work. However, colleagues do not usually tell the boss, so this individual goes unnoticed. Zenda is supervising four employees, and it is known that one of them never works. But who is it?

Finding the lazy employee

The project Zenda's team is working on can only be completed if **at least three people** in the team are working. Let's model this situation in a circuit:



In this diagram, the qubit e_i refers to the i-th employee, which will take the value 1 if this employee is chosen to work on the project. The output state labelled *result* will take the value 0 if the project was not completed and 1 if it was. Let us imagine that employee e_1 is the one who does not work. Then, if we apply the operator to the state $|1\rangle|1\rangle|0\rangle|1\rangle|0\rangle$ (that is, we select e_1, e_2 and e_4 for the project), the output will be $|1\rangle|1\rangle|0\rangle|1\rangle|0\rangle$. As we can see, the last qubit is still $|0\rangle$, i.e. the project has not been carried out. This is because there are only two employees that actually work on the project, and a minimum of three is required.

Zenda wants to know who the lazy worker is, executing as few projects as possible. For this reason, they ask you to help her with your quantum skills. You are asked to discover who the lazy employee is, using a single shot and a single call to the "Project execution" operator.

Challenge code

On one hand, you are asked to complete `circuit` (you only need to apply gates). You can only call the `project_execution` operator once, which is already incorporated in the template. On the other hand, you must complete `process_output`, which will take the output of your circuit and will return who the lazy guy is.

The `project_execution` function will be generated when testing the solution; if you want to experiment with the function output in the notebook, you can temporarily replace `project_execution` with an operator of the form `qml.MultiControlledX(wires=['e1', 'e2', 'e4', 'result'])`. In this case, the absence of "e3" on the wires means that in this project, "e3" will be the lazy employee. Just remember to switch it back to project_execution before submitting, so that your function uses the correct project_execution during testing!

You may find it useful to do some tests in Quirk before you start coding.

Output

To judge this challenge, we will arbitrarily generate 5000 different projects (`project_execution`), which we will send one by one to the circuit to check that your prediction is correct ("e1", "e2", "e3" or "e4"). Therefore, in this case, there will be no public and private test cases. Good luck!

Code

Help

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

```python
dev = qml.device("default.qubit", wires=["e1", "e2", "e3", "e4", "result"], shots=1)
dev.operations.add("op")

wires = ["e1", "e2", "e3", "e4", "result"]
```

```python
@qml.qnode(dev)
def circuit(project_execution):
    """This is the circuit we will use to detect which is the lazy worker. Remember
    that we will only execute one shot.

    Args:
        project_execution (qml.ops):
            The gate in charge of marking in the last qubit if the project has been finished
            as indicated in the statement.

    Returns:
        (numpy.tensor): Measurement output in the 5 qubits after a shot.
    """
    # Put your code here #

    wire_labels = ["e1","e2","e3","e4"]
    weights=np.zeros(16)
    weights[7]=0.5
    weights[11]=0.5
    weights[13]=0.5
    weights[14]=0.5

    # initialize
    qml.QubitStateVector(weights,wires=["e1","e2","e3","e4"])
    qml.PauliX("result")
    qml.Hadamard("result")

    project_execution(wires=wires)

    qml.PauliX(wires="e4")
    qml.PauliX(wires="e3")
    qml.PauliX(wires="e2")
    qml.CNOT(["e1", "e2"])
    qml.CNOT(["e2", "e3"])
    qml.CNOT(["e3", "e4"])
    qml.ctrl(qml.Hadamard, control='e2')(wires='e3')
```

```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:
    samples = 5000

    solutions = []
    output = []

    for s in range(samples):
        lazy = np.random.randint(0, 4)
        no_lazy = list(range(4))
        no_lazy.pop(lazy)

        def project_execution(wires):
            class op(qml.operation.Operator):
                num_wires = 5

                def compute_decomposition(self, wires):
                    raise ValueError("You cant descompose this gate")

                def matrix(self):
                    m = np.zeros([32, 32])
                    for i in range(32):
                        b = [int(j) for j in bin(64 + i)[-5:]]
                        if sum(np.array(b)[no_lazy]) == 3:
                            if b[-1] == 0:
                                m[i, i + 1] = 1
                            else:
                                m[i, i - 1] = 1
                        else:
                            m[i, i] = 1
                    return m
```

```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: '{expected_output}'.")

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

Copy all Reset Open Notebook Submit