PENNY GAMES

Games: Switches [500 points]

Version: 1

Games

Quantum computers help us solve many problems, but we can also use them to explore some of the most mysterious aspects of quantum mechanics. We learn so much better when we have fun, so why not use some crazy games and experiments to explore the boundaries of quantum theory? These fun coding challenges, ranging from entangling full-blown animals to using quantum circuits to cheat our way into victory, teach us a lot about why quantum computing is so powerful.

In the **Games** category, we will be exploring some of the weirdest quantum experiments proposed in the literature. These will enlighten us about how quantum mechanics is different from classical physics, but will also give rise to deeper philosophical questions. Let's have some fun!

Problem statement [500 points]

You are in a room with three switches that turn on a light bulb located in a different room. Your goal is to identify which switches are working (there may be more than one). Both the switches and light bulbs are represented by a 0 or a 1 when they are turned off or on, respectively.

To determine which switches work, we might naively turn on any number of switches and then go to the other room to check if the light is on or not. However, these switches have been set up in an adversarial way. If two *working* switches are turned on, one of the switches turns the light on, but the other one will turn the light off. If we turn on all three switches at once and see the light on, it can be due to two different situations: either one switch works or all three work. It can't be the case that only two work, as they would have cancelled each other's effect, and we would see the light off.

Translating this idea to the quantum world, let's imagine that we are given an oracle to which we can ask questions. We can model each switch as a qubit, where the cumulative three-switch state is given by $|s_0, s_1, s_2\rangle$. For example, if

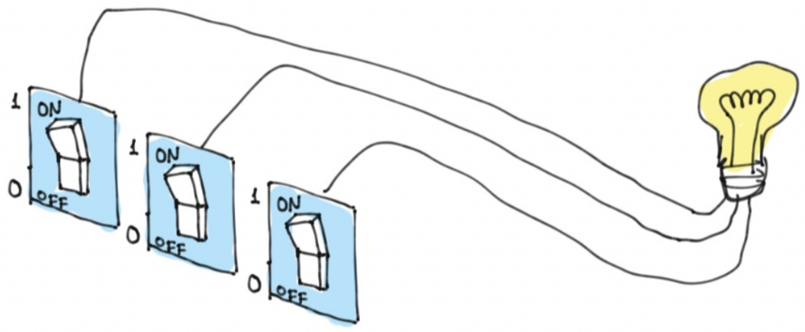

Figure 1: Switches

we send the state $|s_0, s_1, s_2\rangle = |100\rangle$ through the oracle and the light bulb turns on, it means that the first switch is indeed working. We will also include the light bulb in our quantum circuit as an "ancillary" qubit as follows.

Let's take a closer look at how this oracle works. Suppose, for example, that only the first and third switches are operational:

- If we enter the state $|100\rangle|0\rangle$ (where the fourth qubit by default will start at 0) the output will be $|100\rangle|1\rangle$, since the first switch is working and will activate the light.
- If we enter the state $|110\rangle|0\rangle$ the output will be $|110\rangle|1\rangle$ because, even if we have turned on the second switch, it does not work and will not produce any change.
- If we enter the state $|111\rangle|0\rangle$ the output will be $|111\rangle|0\rangle$ since the first switch works (turns the light on), but the third switch works as well (and we will go from on to off).

Given an arbitrary oracle, you are asked to determine which switches work by querying the oracle *once* (i.e., you may only run your circuit once). Classically, such a query is equivalent to visiting the light bulb room only once. Specifically, your code will:

- create a circuit that contains a given light bulb oracle and perform some pre- and post-processing around the oracle
- output something sensible that will help you interpret which switches are working

In the provided template called `game_switches_template.py`, there is a function called `circuit` that you need to complete. The `oracle()` function is already called within `circuit`, but you must add some gates around it. The finalized `circuit` function must return a list of switches that work.

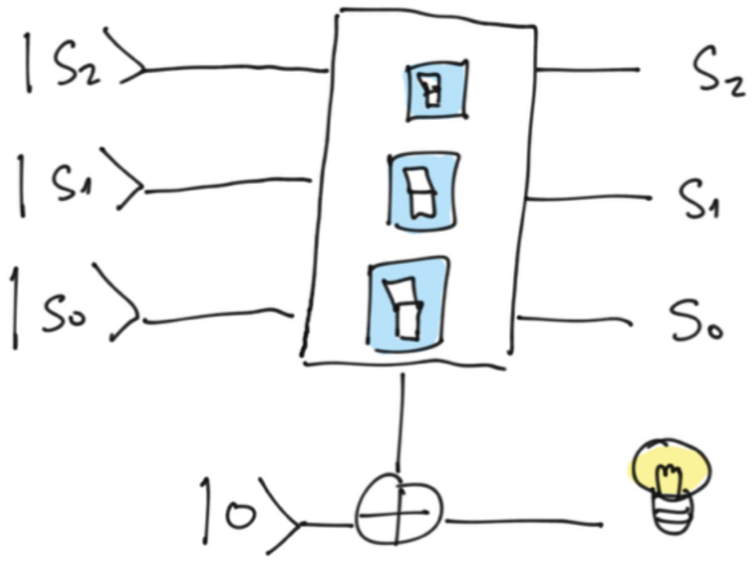

Figure 2: Oracle

Input

- `list(int)`: A list of integers that define how the oracle is created.

Output

- `list(int)`: A list of integers indicating which switches work. E.g., `[0,2]` means that the first and third switches are working.

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 `60s` 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: **60 s**

Version History

Version 1: Initial document.