Microsoft Q# Coding Contest - Summer 2020 - Warmup

A1. Distinguish I from X

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an operation that implements a single-qubit unitary transformation: either the identity (I gate) or the X gate. The operation will have Adjoint and Controlled variants defined.

Your task is to perform necessary operations and measurements to figure out which unitary it was and to return 0 if it was the I gate or 1 if it was the X gate.

You are allowed to apply the given operation and its adjoint/controlled variants exactly once.

You have to implement an operation which takes a single-qubit operation as an input and returns an integer. Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Intrinsic;

    operation Solve (unitary : (Qubit => Unit is Adj+Ctl)) : Int {
        // your code here
    }
}
```

A2. Distinguish I from Z

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an operation that implements a single-qubit unitary transformation: either the identity (I gate) or the Z gate. The operation will have Adjoint and Controlled variants defined.

Your task is to perform necessary operations and measurements to figure out which unitary it was and to return 0 if it was the I gate or 1 if it was the Z gate.

You are allowed to apply the given operation and its adjoint/controlled variants exactly once.

You have to implement an operation which takes a single-qubit operation as an input and returns an integer. Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Intrinsic;

    operation Solve (unitary : (Qubit => Unit is Adj+Ctl)) : Int {
        // your code here
    }
}
```

A3. Distinguish Z from S

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an operation that implements a single-qubit unitary transformation: either the Z gate or the S gate. The operation will have Adjoint and Controlled variants defined.

Your task is to perform necessary operations and measurements to figure out which unitary it was and to return 0 if it was the Z gate or 1 if it was the S gate.

You are allowed to apply the given operation and its adjoint/controlled variants exactly **twice**.

You have to implement an operation which takes a single-qubit operation as an input and returns an integer. Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Intrinsic;

    operation Solve (unitary : (Qubit => Unit is Adj+Ctl)) : Int {
        // your code here
    }
}
```

A4. Distinguish I ⊗ X from CNOT

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an operation that implements a two-qubit unitary transformation: either $I \otimes X$ (the X gate applied to the second qubit and no effect on the first qubit) or the CNOT gate with the first qubit as control and the second qubit as target. The operation will have Adjoint and Controlled variants defined.

Your task is to perform necessary operations and measurements to figure out which unitary it was and to return 0 if it was $I \otimes X$ or 1 if it was the CNOT gate.

You are allowed to apply the given operation and its adjoint/controlled variants exactly once.

You have to implement an operation which takes a two-qubit operation `unitary` as an input and returns an integer. The operation `unitary` will accept an array of qubits as input, but it will fail if the array is empty or has one or more than two qubits. Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Intrinsic;

    operation Solve (unitary : (Qubit[] => Unit is Adj+Ctl)) : Int {
        // your code here
    }
}
```

A5. Distinguish Z from -Z

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an operation that implements a single-qubit unitary transformation: either the Z gate or the -Z gate (i.e., the $-|0\rangle\langle 0| + |1\rangle\langle 1|$ gate: $(-Z)(\alpha|0\rangle + \beta|1\rangle) = -\alpha|0\rangle + \beta|1\rangle$). The operation will have Adjoint and Controlled variants defined.

Your task is to perform necessary operations and measurements to figure out which unitary it was and to return 0 if it was the Z gate or 1 if it was the -Z gate.

You are allowed to apply the given operation and its adjoint/controlled variants exactly **once**.

You have to implement an operation which takes a single-qubit operation as an input and returns an integer. Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Intrinsic;

    operation Solve (unitary : (Qubit => Unit is Adj+Ctl)) : Int {
        // your code here
    }
}
```

B1. Increment

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a unitary operation on a register of N qubits that increments the number written in the register modulo 2^N.

Your operation should take a register of type `LittleEndian` - an array of qubits that encodes an unsigned integer in little-endian format, with the least significant bit written first (corresponding to the array element with index 0). The "output" of your solution is the state in which it left the input qubits.

For example, if the qubits passed to your operation are in the state $\frac{1}{\sqrt{2}}(|11\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|3\rangle + |1\rangle)$, they should be transformed to the state $\frac{1}{\sqrt{2}}(|(3+1) \mod 2^2\rangle + |(1+1) \mod 2^2\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |2\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$.

Your code should have the following signature (note that your operation should have Adjoint and Controlled variants defined for it; `is Adj+Ctl` in the operation signature will generate them automatically based on your code):

```
namespace Solution {
    open Microsoft.Quantum.Arithmetic;
    open Microsoft.Quantum.Intrinsic;

    operation Solve (register : LittleEndian) : Unit is Adj+Ctl {
        // your code here
    }
}
```

Your code is not allowed to use measurements or arbitrary rotation gates (so, for example, using the library operation IncrementByInteger will cause runtime error). This operation can be implemented using just the X gate and its controlled variants.

B2. Decrement

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement a unitary operation on a register of N qubits that decrements the number written in the register modulo 2^N.

Your operation should take a register of type `LittleEndian` - an array of qubits that encodes an unsigned integer in little-endian format, with the least significant bit written first (corresponding to the array element with index 0). The "output" of your solution is the state in which it left the input qubits.

For example, if the qubits passed to your operation are in the state $\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |2\rangle)$, they should be transformed to the state $\frac{1}{\sqrt{2}}(|(0-1) \mod 2^2\rangle + |(2-1) \mod 2^2\rangle) = \frac{1}{\sqrt{2}}(|3\rangle + |1\rangle) = \frac{1}{\sqrt{2}}(|11\rangle + |10\rangle)$.

Your code should have the following signature (note that your operation should have Adjoint and Controlled variants defined for it; `is Adj+Ctl` in the operation signature will generate them automatically based on your code):

```
namespace Solution {
    open Microsoft.Quantum.Arithmetic;
    open Microsoft.Quantum.Intrinsic;

    operation Solve (register : LittleEndian) : Unit is Adj+Ctl {
        // your code here
    }
}
```

Your code is not allowed to use measurements or arbitrary rotation gates (so, for example, using the library operation IncrementByInteger will cause runtime error). This operation can be implemented using just the X gate and its controlled variants.

C. Prepare state |01⟩ + |10⟩ + |11⟩

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given two qubits in state $|00\rangle$. Your task is to prepare the following state on them:

$$\frac{1}{\sqrt{3}}(|01\rangle + |10\rangle + |11\rangle)$$

This task is very similar to problem A1 of the Winter 2019 contest, but this time you are not allowed to use any gates except the Pauli gates (X, Y and Z), the Hadamard gate and the controlled versions of those. However, you are allowed to use measurements.

You have to implement an operation which takes an array of 2 qubits as an input and has no output. The "output" of your solution is the state in which it left the input qubits.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.Intrinsic;

    operation Solve (qs : Qubit[]) : Unit {
        // your code here
    }
}
```

D1. Quantum Classification - 1

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a training dataset, in which each entry is a features vector (an array of 2 real numbers) and a label 0 or 1 indicating the class to which this vector belongs.

Your goal is to use this dataset to train a quantum classification model that will accurately classify a validation dataset - a different dataset generated using the same data distribution as the training one. The error rate of classifying the validation dataset using your model (the percentage of incorrectly classified samples) should be less than 5

- The quantum classification library that will use your model to classify the data is documented here.
- This tutorial has an end-to-end example of training a model using this library as a Python notebook.
- You can find examples of training a model and using it for classification here.

Input

Your code will not be given any inputs. Instead, you should use the provided dataset file to train your model.

The training dataset is represented as a JSON file and consists of two arrays, "Features" and "Labels". Each array has exactly 200 elements. Each element of the "Features" array is an array with 2 elements, each of them a floating-point number between -1 and 1. Each element of the "Labels" array is the label of the class to which the corresponding element of the "Features" array belongs, 0 or 1.

Output

Your code should return the description of the model you'd like to use in the following format:

- The model is described using a tuple `(ControlledRotation[], (Double[], Double))`.
- The first element of the tuple describes circuit geometry of the model as an array of controlled rotation gates.
- The second element of the tuple describes numeric parameters of the model and is a tuple of an array of rotation angles used by the gates and the bias used to decide the class of the model.

Your code should have the following signature:

```
namespace Solution {
    open Microsoft.Quantum.MachineLearning;

    operation Solve () : (ControlledRotation[], (Double[], Double)) {
        // your code here
    }
}
```

Please refer to the documentation and examples for details on each parameter.

Note

Note that majority of the data analysis is going to happen "offline" before you submit the solution. The solution has to contain only the description of the trained model, not the training code itself - if you attempt to train the model "online" in your submitted code during the evaluation process, it will very likely time out.

Training your model offline is likely to involve:

- Defining the circuit structure that your model will use.
- Generating several parameter seed vectors - the values from which training the model will start.
- Selecting appropriate hyperparameters of the training process (learning rate, batch size, tolerance, maximal number of iterations etc.)
- Training a number of classification models (one per each seed vector and hyperparameter combination)
- Selecting the best trained model and submitting it.

D2. Quantum Classification - 2

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This problem statement is exactly the same as in the problem "Quantum Classification - 1"; the only difference is the training dataset used (and the model you need to train on it).

You are given a training dataset, in which each entry is a features vector (an array of 2 real numbers) and a label 0 or 1 indicating the class to which this vector belongs.

Your goal is to use this dataset to train a quantum classification model that will accurately classify a validation dataset - a different dataset generated using the same data distribution as the training one. The error rate of classifying the validation dataset using your model (the percentage of incorrectly classified samples) should be less than 5

- The quantum classification library that will use your model to classify the data is documented here.
- This tutorial has an end-to-end example of training a model using this library as a Python notebook.
- You can find examples of training a model and using it for classification here.

Input

Your code will not be given any inputs. Instead, you should use the provided dataset file to train your model.

The training dataset is represented as a JSON file and consists of two arrays, "Features" and "Labels". Each array has exactly 200 elements. Each element of the "Features" array is an array with 2 elements, each of them a floating-point number between -1 and 1. Each element of the "Labels" array is the label of the class to which the corresponding element of the "Features" array belongs, 0 or 1.

Output

Your code should return the description of the model you'd like to use in the following format:

- The model is described using a tuple `(ControlledRotation[], (Double[], Double))`.
- The first element of the tuple describes circuit geometry of the model as an array of controlled rotation gates.
- The second element of the tuple describes numeric parameters of the model and is a tuple of an array of rotation angles used by the gates and the bias used to decide the class of the model.

Your code should have the following signature:

```
namespace Solution {
   open Microsoft.Quantum.MachineLearning;

   operation Solve () : (ControlledRotation[], (Double[], Double)) {
      // your code here
   }
}
```

Please refer to the documentation and examples for details on each parameter.

Note

Note that majority of the data analysis is going to happen "offline" before you submit the solution. The solution has to contain only the description of the trained model, not the training code itself - if you attempt to train the model "online" in your submitted code during the evaluation process, it will very likely time out.

Training your model offline is likely to involve:

- Defining the circuit structure that your model will use.
- Generating several parameter seed vectors - the values from which training the model will start.
- Selecting appropriate hyperparameters of the training process (learning rate, batch size, tolerance, maximal number of iterations etc.)
- Training a number of classification models (one per each seed vector and hyperparameter combination)
- Selecting the best trained model and submitting it.

Codeforces (c) Copyright 2010-2022 Mike Mirzayanov
The only programming contests Web 2.0 platform