The task of this homework assignment was to program the Deutsch algorithm and Simon algorithm in our quantum simulator, run them for $n = 1$ and show the evolution of the states in some fashion. I will describe my implementation and results in the following sections, technical details are described more in the README file of the project or at https://github.com/radajakub/qsim.

1 Deutsch-Josza algorithm

First, I implemented the Deutsch-Josza algorithm for arbitrary number of qubits n. As oracle, I used the standard circuit implementation for both balanced and constant oracles, the type for a given run can be selected from commandline argument.

The constant oracle is implemented either as $U_f = I^{\otimes n+1}$ for output 0 or $U_f = I^{\otimes n} \otimes X$ for output 1. The balanced oracle is implemented as a chain of CNOT gates with controls on qubits $1, ..., n$ and target on the ancilla qubit $n + 1$.

Measurement is then done on all except the ancilla qubit and results are reported as an ASCII bar chart. The decision whether the oracle is balanced or constant is made by counting the results and selecting the majority.

1.1 Example

Here, I will demonstrate the functionality and results of my implementation for $n = 1$. I will show all possible oracles to confirm that they work correctly. The description will be done on the actual text output of the program so it can serve as a manual to the reader as well.

Constant For constant function we expect to measure 0 on the non-ancilla qubits. Thus for $n = 1$, we want to measure 0 for both outputs of the constant function.

To run a constant oracle with output 1 use this command `./deutsch -type constant -output 1 -n 1 -verbose 1`. Don't forget to run 'make deutsch' beforehand.

The output then looks similarly to this:

```
Steps of the circuit [G is applied gate, Q is state vector in a step]:
$ (0) Q: [1, 0, 0, 0]^T
$ (1) G: I * X
$ (2) Q: [0, 1, 0, 0]^T
$ (3) G: H * H
$ (4) Q: [0.5, -0.5, 0.5, -0.5]^T
$ (5) G: (I * X)(I * I)
$ (6) Q: [-0.5, 0.5, -0.5, 0.5]^T
$ (7) G: H * I
$ (8) Q: [-0.707107, 0.707107, 0, 0]^T

Measurements:
0 |##################################################| 100% (1024/1024)

Function is CONSTANT
```

First block of the output shows the evolution of the state vector. The steps denoted with `G` show symbolic representation of the applied gate, if defined. Just a side note to gates, sometimes there is a matrix multiplication, in such case the gates are exectued from right to left. The steps denoted with `Q` show the state vector at the given step, which is given by a position of a barrier in the code.

The second block then shows the results of the measurements with percentage of total number of shots. We can see that the function is indeed constant because we measured 0 with probability 1.

Similary for the output 0:

```
Steps of the circuit [G is applied gate, Q is state vector in a step]:
$ (0) Q: [1, 0, 0, 0]^T
$ (1) G: I * X
$ (2) Q: [0, 1, 0, 0]^T
$ (3) G: H * H
$ (4) Q: [0.5, -0.5, 0.5, -0.5]^T
$ (5) G: I * I
$ (6) Q: [0.5, -0.5, 0.5, -0.5]^T
$ (7) G: H * I
$ (8) Q: [0.707107, -0.707107, 0, 0]^T

Measurements:
0 |##################################################| 100% (1024/1024)

Function is CONSTANT
```

The results are exactly the same but we can notice that the oracle gate at step (5) is just an identity, compared to the previous case.

Balanced For balanced function we expect to measure 1 on the non-ancilla qubits. Thus for $n = 1$, we want to measure 1. In this case, there is no `-output` parameter.

To run a balanced oracle use this command `./deutsch -type balanced -n 1 -verbose 1`. Don't forget to run 'make deutsch' beforehand.

The output then looks similarly to this:

```
Steps of the circuit [G is applied gate, Q is state vector in a step]:
$ (0) Q: [1, 0, 0, 0]^T
$ (1) G: I * X
$ (2) Q: [0, 1, 0, 0]^T
$ (3) G: H * H
$ (4) Q: [0.5, -0.5, 0.5, -0.5]^T
$ (5) G: CX[0,1]
$ (6) Q: [0.5, -0.5, -0.5, 0.5]^T
$ (7) G: H * I
$ (8) Q: [0, 0, 0.707107, -0.707107]^T

Measurements:
1 |##################################################| 100% (1024/1024)

Function is BALANCED
```

Again, the result is almost identical to the constant case, the only difference in the circuit history is the CNOT gate between qubits 0 and 1. And we correctly measure 1 with probability 1.

2 Simon's algorithm

The more interesting algorithm is Simon's algorithm. Here, we want to distinguis between a one-to-one function and a two-to-one function. The function is given by a *secret* bit-string. If the bit-string is all 0, the function is one-to-one, otherwise it is two-to-one. I again give the option to specify the secret key as an argument. The number of bits in this bit-string determines the number of qubits used. Note that for larger n it takes a lot of memory and time because the matrices are implemented as dense and the dimension grows exponentially with n.

The oracle in this case is implemented as first copying the input qubits to the ancilla qubits ($|x\rangle|0\rangle \to |x\rangle|x\rangle$) by CNOT gates between corresponding pairs of qubits and second applying CNOT with a control qubit on first non-zero position in the secret bit-string on all the ancilla qubits.

We measure the non-ancilla qubits and record all possible results. Then, from these measurements we can try to guess the secret bit-string by solving a system of linear equations. I implemented it as a Gaussian-Jordan elimination on a augmented matrix. If the only solution of the system is the trivial solution of all 0s, the function is one-to-one and the secret bit-string is all 0s. Otherwise, we find the used secret bit-string and report it.

2.1 Example

First, we run the algorithm for the secret bit-string 0. Thus, $n = 1$ and the function is one-to-one. The command to run it in this setting is `./simon -secret 0 -verbose 1`.

```
Steps of the circuit [G is applied gate, Q is state vector in a step]:
$ (0) Q: [1, 0, 0, 0]^T
$ (1) G: H * I
$ (2) Q: [0.707107, 0, 0.707107, 0]^T
$ (3) G: Uf
$ (4) Q: [0.707107, 0, 0, 0.707107]^T
$ (5) G: H * I
$ (6) Q: [0.5, 0.5, 0.5, -0.5]^T
Measurements:
0 |########################.........................| 46.7773% (479/1024)
1 |#########################.........................| 53.2227% (545/1024)

secret: 0 (correct) THE FUNCTION IS ONE TO ONE
```

We can see that we measured both outcomes 0 and 1, which means that the solution of the system of linear equations is the trivial case. That means that the function is one-to-one.

For the secret bit-string 1, the command is `./simon -secret 1 -verbose 1`.

```
Steps of the circuit [G is applied gate, Q is state vector in a step]:
$ (0) Q: [1, 0, 0, 0]^T
$ (1) G: H * I
$ (2) Q: [0.707107, 0, 0.707107, 0]^T
$ (3) G: Uf
$ (4) Q: [0.707107, 0, 0.707107, 0]^T
$ (5) G: H * I
$ (6) Q: [1, 0, 0, 0]^T
Measurements:
0 |##################################################| 100% (1024/1024)
```

```
secret: 1 (correct) THE FUNCTION IS TWO TO ONE
```

In this case, we measured only 0 which means that the secret key can be also someting else than the trivial solution, more specifically 1. Thus, the function is two-to-one and the secret key is 1.

3 qubit case I think that the $n = 1$ case is not very interesting so I want to demonstrate it for bigger number of qubits because I spent quite some time on it to get it right.

The command `./simon -secret 101` gives the following output (I am omitting verbose because the states are too big):

```
Measurements:
000 |############.....................................| 24.8047% (254/1024)
101 |##############.................................| 28.0273% (287/1024)
010 |###########.....................................| 23.5352% (241/1024)
111 |###########.....................................| 23.6328% (242/1024)

secret: 101 (correct) THE FUNCTION IS TWO TO ONE
```

We see that we measured 4 out of 8 possible outcomes, which is consistent with the fact that the function is two-to-one. The secret key found by Gaussian elimination is 101.

On the other hand for a one-to-one function, the command `./simon -secret 000` gives the following output:

```
Measurements:
000 |######.........................................| 13.6719% (140/1024)
111 |######.........................................| 12.3047% (126/1024)
010 |#####..........................................| 11.5234% (118/1024)
001 |#####..........................................| 11.4258% (117/1024)
110 |######.........................................| 13.1836% (135/1024)
100 |######.........................................| 12.9883% (133/1024)
011 |#####..........................................| 11.5234% (118/1024)
101 |######.........................................| 13.3789% (137/1024)

secret: 000 (correct) THE FUNCTION IS ONE TO ONE
```

Here we measured all 8 possible outcomes, which means that the function is one-to-one with secret bit-string 000 because Gaussian elimination could find only the trivial solution.