Simulation of Shor's Algorithm Report

José Semedo
ist178294

Rodrigo Bernardo
ist178942

1 Introduction

In this project we studied and implemented a simulator of Shor's quantum algorithm for integer factorization. The problem is reduced to the problem of order finding. Although it is not known if order finding is hard in a classical setting, Shor [1] demonstrated that it is solvable in polynomial time if one has access to a quantum computer, thus showing integer factorization is solvable in polynoial time.

2 Language and Tools

Our simulation is implemented in Python 3, together with the numpy library.

3 Implementation Overview

3.1 Memory

The algorithm takes an odd integer N, such that it is not a prime nor a power of a prime, and an integer $x, 1 < x < N$, and tries to find the multiplicative order of x modulo N.

It starts by allocating $t + n$ qubits, with $n = \lceil \log_2 N \rceil$ and $N^2 \leq 2^t < 2N^2$, and initializes the state to $|\psi_0\rangle = |0, 0\rangle$.

We represent the memory by explicitly saving all the 2^{t+n} possible states and their corresponding amplitudes, equivalent to the representation of the quantum state as a linear vector combination $|\psi\rangle = \sum_{j=0}^{t+n} a_j \, |j\rangle$.

3.2 Hadamard Gates

The algorithm then applies Hadamard gates to the first t qubits. This creates a quantum superposition where the amplitudes are equidistributed between the first t bits. The state becomes $|\psi_1\rangle = H^{\otimes t} \, |\psi_0\rangle = 2^{-t/2} \sum_{j=0}^{2^t-1} |j\rangle \, |0\rangle$.

We simulate this step by explicitly reaching for the states where the last n qubits are $|0\rangle$ and setting their amplitudes to $2^{-t/2}$.

3.3 Modular Exponentiation

In the next step, the operator $|j, k\rangle \mapsto |j, k + x^j \pmod N\rangle$ is applied to all the qubits, giving the state $|\psi_2\rangle = 2^{-t/2} \sum_{j=0}^{2^t-1} |j\rangle \, |x^b\rangle$.

This step is fast because it generates all the powers simultaneously by quantum parallelism.

Here we take advantage of Python's built-in modular exponentiation function and simulate this by applying the operator to all the states sequentially.

3.4 Quantum Fourier Transform

The discrete Fourier transform is then applied to the first t qubits. This step is $O(n2^n)$ if done classically, but can be done polynomially with a quantum computer.

We simulate this step by sequentially applying the formula $|k\rangle \mapsto 2^{-t/2} \sum_{j=0}^{2^t-1} \omega^{jk} |j\rangle$, where $\omega^{jk} = e^{2\pi ijk/N}$, to all the possible states, i.e., for each state $|\phi\rangle = |k\rangle$, its amplitude is changed to $2^{-t/2} \sum_{j=0}^{2^t-1} \omega^{jk}$.

After the quantum state of the system is $2^{-t/2} \sum_{j=0}^{2^t-1} \sum_{k=0}^{2^t-1} \omega^{jk} |k\rangle |x^j\rangle$

3.5 Obtaining the Order

Finally, a measurement is taken, leaving the state to collapse to one vector of the computational basis. After the application of the previous operations we are left with an approximation of a number $a/r, a \in \mathbb{Z}$ with high probability.

We use a known efficient classical algorithm [2] for extracting r based on the best approximation property of the convergents of continued fractions. If we succeed to find r, we return it, else the algorithm is restarted.

4 Execution

We can run the program by issuing the command

```
$ ./shor.py N
```

4.1 Examples

```
$ ./shor.py 15
| picked random a = 3
| got lucky, 15 = 3 * 5, trying again...
|------------------------------------------------
| picked random a = 8
| measured 59, approximation for 0.23046875 is 3/13
| 8^13 mod 15 = 8
| failed, trying again ...
| measured 246, approximation for 0.9609375 is 1/1
| 8^1 mod 15 = 8
| failed, trying again ...
| measured 109, approximation for 0.42578125 is 3/7
| 8^7 mod 15 = 2
| failed, trying again ...
| measured 222, approximation for 0.8671875 is 7/8
| 8^8 mod 15 = 1
| got 8
```

```
| found factor: 15 = 5 * 3
5

$ ./shor.py 21
| picked random a = 10
| measured 152, approximation for 0.296875 is 3/10
| 10^10 mod 21 = 4
| failed, trying again ...
| measured 342, approximation for 0.66796875 is 2/3
| 10^3 mod 21 = 13
| failed, trying again ...
| measured 37, approximation for 0.072265625 is 1/14
| 10^14 mod 21 = 16
| failed, trying again ...
| measured 53, approximation for 0.103515625 is 2/19
| 10^19 mod 21 = 10
| failed, trying again ...
| measured 42, approximation for 0.08203125 is 1/12
| 10^12 mod 21 = 1
| got 12
| found factor: 21 = 7 * 3
7
```

References

[1] Peter Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

[2] G. H. Hardy, E. M. Wright, *Introduction to Theory of Numbers*, Oxford University Press, 4th Edition, 1975.