МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ

по учебной практике

Тема: Квантовые вычисления. Алгоритм Шора.

Студент гр. 9308 _____ Яловега Н.В.

Руководитель _____ Фамилия И.О.

Санкт-Петербург
2021

Задание
на учебную практику

Студент Яловега Н.В.
Группа 9308
Тема практики: Квантовые вычисления. Алгоритм Шора.

Задание на практику:
Изучить основы квантовых вычислений и алгоритмов. Понять принцип работы и реализовать алгоритма Шора.

Сроки прохождения практики: 01.07.2021-14.07.2021
Дата сдачи отчета: 14.07.2021
Дата защиты отчета: 14.07.2021

Студент гр. 9308	_____	Яловега.Н.В.
Руководитель	_____	Фамилия И.О.

Аннотация

Цель практического задания – добиться понимания принципа работы одного из квантовых алгоритмов и осуществить его реализацию на эмуляторе квантовой системы. В моём случае необходимо проработать квантовый алгоритм факторизации чисел, придуманный в 1994 году Питером Шором.

Для достижения этой цели мне было необходимо усвоить принципы квантовых вычислений, изучить понятия кубита, системы кубитов, и операций над ними. Затем перейти к изучению квантовых алгоритмов и их отличий от классических. Финальным этапом работы стала реализация алгоритма Шора.

Содержание

Введение

Любое устройство, выполняющее вычисления, основано на каком-либо физическом процессе. До ЭВМ вычисления выполнялись механическими системами. Потом появились устройства, основанные на электромагнитных реле, радиолампах, транзисторах. Каждый последующий тип устройства был эффективнее предыдущего, поэтому естественно было бы ожидать появления новых, еще более эффективных ЭВМ. Вместе с устройствами эволюционировали и алгоритмы. В 1936 году Алан Тьюринг предложил математическую модель вычислений, названную впоследствии машиной Тьюринга. Модель Тьюринга формализовала понятие алгоритма (способа вычисления некоторой функции на физическом устройстве), что позволило ввести понятие вычислимости и в дальнейшем - понятие сложности вычислений. Тогда же, в 1936 году, Тьюринг показал, что существуют невычислимые (undecidable) функции - такие функции, которые можно формально определить, но для вычисления которых невозможно предложить алгоритм. Понятие сложности вычислений (количества ресурсов, необходимых для выполнения алгоритма) позволило выделить классы задач, названных неразрешимыми (intractable). В отличие от невычислимых функций, для них существуют алгоритмы, работающие за конечное время. Однако ресурсы, необходимые для вы полнения этих алгоритмов, растут слишком быстро (например, экспоненциально) с увеличением размера входных данных. С появлением неразрешимых задач возникла потребность в физических устройствах, вычислительные возможности которых зависели бы экспоненциально от размеров (ресурсов) устройства. В 1981 году нобелевский лауреат по физике Ричард Фейнман предложил построить вычислительное устройство на основе квантовой системы. Оптимизм Фейнмана был основан на том, что простейшая квантовая система намного сложнее простейшей классической системы. Кроме того, линейным ростом квантовой системы вызывается экспоненциальный рост сложности ее описания. Например, для классического описания состояния 10-кубитной системы потребуется 1024 комплексных числа, а для 30 кубит - более миллиарда комплексных чисел. Уже в 1985 году Дэвид Дойч разработал математическую модель универсального квантового компьютера и показал некоторые ее преимущества перед классической моделью. А в 1994-м Питер Шор взорвал

научную общественность, опубликовав полиномиальный квантовый алгоритм разложения составного числа на множители. Результат Шора поставил под угрозу широко используемый алгоритм ассиметричного шифрования RSA. RSA - ассиметричный алгоритм шифрования, названный по первым буквам фамилий его авторов - Rivest, Shamir, Adleman. Таким образом, квантовые вычисления имеют все шансы превзойти современные классические компьютеры сразу по двум направлениям: 1) как более совершенный, быстрый и информационно емкий физический процесс, 2) как более перспективная в теоретическом смысле модель. Репертуар квантового компьютера шире, чем у классической машины Тьюринга (которая не может, например, генерировать случайные числа), а для многих сложных для классических вычислений задач уже существуют эффективные квантовые алгоритмы.

1. Математическая модель квантовых вычислений

1.1. Кубит

Ключевым понятием всей теории квантовых вычислений является «кубит» — сокращение термина «квантовый бит» (quantum bit) — минимальная информационная единица квантового мира. Так же как бит является частичкой информации, содержащейся в простейшем содержательном классическом вычислительном процессе, кубит является описанием простейшей содержательной квантовой системы.

Кубит — это вектор единичной длинны в двумерном гильбертовом пространстве над полем комплексных чисел.

$$|\phi\rangle \in H, ||\phi|| = 1, dimH = 2;$$

В гильбертовых пространствах определено скалярное произведение векторов, а значит в можно определить понятие угла θ между векторами:

$$\theta = arccos\frac{|\langle x|y\rangle|}{||x|| \cdot ||y||}, \theta \in [0, \frac{\pi}{2}];$$

Выделим в этом пространстве некоторый базис, вектора которого обозначим как $|0\rangle$ и $|1\rangle$. Внутри скобок записывается индекс базисного вектора в двоичной системе счисления, начиная с нуля без дополнительных символов.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix};$$

Тогда произвольный вектор ϕ можно выразить следующим образом:

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

где α и β - некоторые комплексные числа, такие что $|\alpha|^2 + |\beta|^2 = 1$

1.2. Система кубитов

В подавляющем большинстве случаев для вычислений требуется более одного бита. Система, состоящая из нескольких кубитов, описывается тензорным произведением составляющих ее систем. Например, квантовый регистр из 2 кубитов будет описываться пространством H^4 и иметь 4 базисных состояния.

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Таким образом, система из 3 кубитов описывается вектором в 8-мерном пространстве, а размерность пространства системы из 10 кубитов равна 1024. Для 1000 кубитов мы получаем пространство, размерность которого описывается числом с 300 нулями. Построив компьютер на всего лишь 1000 атомов, мы получаем в свое распоряжение машину, состояние которой описывается 1030 комплексными числами.

1.3. Эволюция квантовой системы

Эволюция квантовой системы унитарна, т.е. любое изменение ее состояния выражается действием унитарного оператора. Поскольку изменение состояния и есть вычисление, то и программа для квантового компьютера представляет собой последовательное применение различных унитарных операторов к вектору состояния. Оператор U унитарен, если его сопряженный оператор совпадает с обратным:

$$UU^* = U^*U = I$$

Матрица сопряженного оператора представляет собой транспонированную матрицу исходного оператора, все числа в которой заменены на сопряженные. Необходимым и достаточным условием унитарности оператора является сохранение им при отображении длин векторов и углов между ними:

$$\forall \phi \in H \ ||U|\phi\rangle|| = ||\phi\rangle||,$$
$$\forall \phi, \psi \in H \ |\langle U|\phi\rangle|U|\psi\rangle\rangle| = |\langle\phi|\psi\rangle|,$$

Несколько примеров наиболее встречающихся операторов:
Оператор Адамара:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Гейт X:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Гейт CNOT:

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Также для алгоритма Шора нам потребуется квантовый оператор — квантовое преобразование Фурье (Quantum Fourier Transform, QFT):

Квантовое преобразование Фурье

$$N := 2^n$$

$$QFT|x\rangle = \frac{||x||}{2^{n/2}} \sum_{y=0}^{2^n-1} e^{2\pi i \frac{xy}{N}} |y\rangle$$

1.4. Алгоритм квантовой оценки фазы

Фурье-преобразование является основой для общей процедуры, известной как оценка фазы (phase estimation), которая является ключевой для многих квантовых алгоритмов. Пусть есть унитарный оператор U, имеющий собственный вектор $|U\rangle$ и собственное число $exp(2\pi i\phi)$, где ϕ - неизвестно. Задача алгоритма оценки фазы состоит в вычислении ϕ.

Квантовая процедура оценки фазы использует два регистра. Первый регистр содержит t кубит первоначально находящихся в состоянии $|0\rangle$. Второй регистр начинается с состояния $|U\rangle$ и содержит столько кубит, сколько необходимо чтобы собрать $|U\rangle$.

Алгоритм состоит из трех частей.

Первая часть состоит в использовании гейта Адамара с последующим применением controlled-U - операций на второй регистр, с U увеличенным до степени 2.

Второй шаг вычисления фазы состоит в применении обратного квантового Фурье-преобразования для первого регистра. Оно образуется путем обращения цепи квантового Фурье-преобразования.

Третий и окончательный шаг есть чтение состояния первого регистра.

Схематично весь алгоритм определяется квантовой цепью:

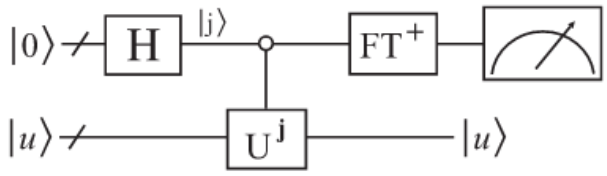

Рис. 1: Схема алгоритма квантовой оценки фазы

Процедура оценки фазы может быть использована для решения ряда задач, в число которых входит поиск периода.

2. Алгоритм Шора

2.1. Проблема факторизации

Факторизацией натурального числа называется его разложение в произведение простых множителей. Существование и единственность (с точностью до порядка следования множителей) такого разложения следует из основной теоремы арифметики.

В отличие от задачи распознавания простоты числа, факторизация предположительно является вычислительно сложной задачей. В настоящее время неизвестно, существует ли эффективный алгоритм факторизации целых чисел. Даже самым современным алгоритмам требуется экспоненциальное время, чтобы найти коэффициенты целого числа. Это означает, что с достаточно большими числами решить эту задачу практически невозможно, несмотря на огромную вычислительную мощность. Однако доказательства того, что не существует решения этой задачи за полиномиальное время, также нет.

Предположение о том, что для больших чисел задача факторизации является вычислительно сложной, лежит в основе широко используемых алгоритмов (например, RSA).

В 1994 году Питер Шор теоретизировал алгоритм квантовых вычислений с полиномиальным временем для целочисленной факторизации. Алгоритм Шора использует как классические, так и квантовые (определение периода) вычисления. Однако преобладающая оптимизация алгоритма Шора обусловлена эффективностью квантового преобразования Фурье и модульным возведением в степень путем многократного возведения в квадрат.

Алгоритм Шора, используя возможности квантовых компьютеров, способен произвести факторизацию числа не просто за полиномиальное время, а за время, не намного превосходящее время умножения целых чисел (то есть практически так же быстро, как происходит само шифрование).Таким образом, реализация масштабируемого квантового компьютера поставит крест на большей части современной криптографической защиты. Речь не только о схеме RSA, прямо опирающейся на сложности факторизации, но и о других сходных схемах, которые квантовый компьютер способен взломать аналогичным образом.

Важно отметить, что алгоритм Шора имеет вероятностный характер.

Первый источник случайности встроен в классическое вероятностное сведение разложения на множители к нахождению периода некоторой функции. Второй источник появляется из необходимости наблюдения квантовой памяти, которое также даёт случайные результаты.

2.2. Поиск периода и факторизация

Алгоритм Шора предназначен для поиска неизвестного периода некоторой периодической функции. Как это позволит раскладывать числа на множители?

Пусть p и q - различные простые числа и $N = pq$. Для некоторого числа $a < N : (a, N) = 1$ определим функцию $f_a(x)$:

$$f_a(x) = a^x \pmod N$$

Функция f_a переодическая, и ее период r является порядком числа a в кольце $\mathbb{Z}_{\mathbb{N}}$:

$$a^r = 1 \pmod N$$

$$\forall r_1 < r \; a^{r_1} \neq 1 \pmod N$$

Если в нашем распоряжении есть устройство, умеющее находить период любой периодической функции, то при помощи этого устройства, выбрав случайным образом число a, мы можем найти период r функции f_a.

Допустим, число r - четное. Тогда выражение

$$a^r - 1 = 0 \pmod N$$

мы можем представить в виде

$$(a^{r/2} - 1)(a^{r/2} + 1) = Nk$$

Число $(a^{r/2} - 1)$ не делится на N, так как иначе $a^{r/2}$ было бы сравнимо с 0 по модулю N, и число $r/2$ было бы периодом.

Допустим также, что

$$(a^{r/2} + 1) \neq 0 \pmod N$$

Тогда произведение $(a^{r/2} - 1)(a^{r/2} + 1)$ делится на N, но ни один из множителей $(a^{r/2} - 1)$, $(a^{r/2} + 1)$ не делится на N целиком. Следовательно, числа $(a^{r/2} - 1)$, $(a^{r/2} + 1)$ не взаимно просты с N, и мы можем найти p и q

при помощи алгоритма Евклида:

$$p, q = GCD(a^{r/2} \pm 1, N);$$

В ходе рассуждения мы сделали два допущения. С какой вероятностью для наугад выбранного числа a его порядок r удовлетворяет этим условиям? Оказывается, что вероятность неудачи сведения задачи факторизации к задаче поиска периода при выборе числа a небольше $1/2$.

2.3. Квантовый алгоритм Шора

Квантовый алгоритм Шора решает проблему нахождения периода следующей функции:

$$f(x) = a^x \pmod N$$

Решение Шора заключалось в использовании квантовой оценки фазы унитарного оператора:

$$U|y\rangle \equiv |ay \ mod \ N\rangle$$

Чтобы увидеть, насколько это полезно, давайте разберемся, как может выглядеть собственное состояние U. Если мы начали в состоянии $|1\rangle$, мы увидим, что каждое последующее применение U будет умножать состояние нашего регистра на $a \pmod N$, и после r применений мы снова придем к состоянию $|1\rangle$. Таким образом, суперпозиция состояний в этом цикле ($|u_0\rangle$) будет собственным состоянием U:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \ mod \ N\rangle$$

Это собственное состояние имеет собственное значение 1, что не очень интересно. Более интересным собственным состоянием могло бы быть такое, в котором фаза различна для каждого из этих вычислительных базисных состояний. В частности, давайте рассмотрим случай, когда фаза k-го состояния пропорциональна k:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} |a^k \ mod \ N\rangle$$

$$U|u_1\rangle = e^{\frac{2\pi i}{r}} |u_1\rangle$$

Это особенно интересное собственное значение, поскольку оно содержит r. Это не единственное собственное состояние с таким поведением; чтобы обобщить это дальше, мы можем умножить целое число s на эту разность фаз, которая будет отображаться в нашем собственном значении:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k s}{r}} |a^k \ mod \ N\rangle$$

$$U|u_s\rangle = e^{\frac{2\pi i s}{r}}|u_s\rangle$$

Теперь у нас есть уникальное собственное состояние для каждого целого значения $0 \leq s \leq r-1$.

Если мы суммируем все эти собственные состояния, разные фазы сокращают все вычислительные базисные состояния, кроме $|1\rangle$

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = |1\rangle$$

Поскольку вычислительное базисное состояние $|1\rangle$ является суперпозицией этих собственных состояний, это означает, что если мы выполняем QPE на U, используя состояние $|1\rangle$, мы будем измерять фазу:

$$\phi = \frac{s}{r},$$

где s случайное число между 0 и $r-1$. Наконец, мы используем алгоритм цепных дробей на ϕ, чтобы найти r. Принципиальная схема выглядит следующим образом:

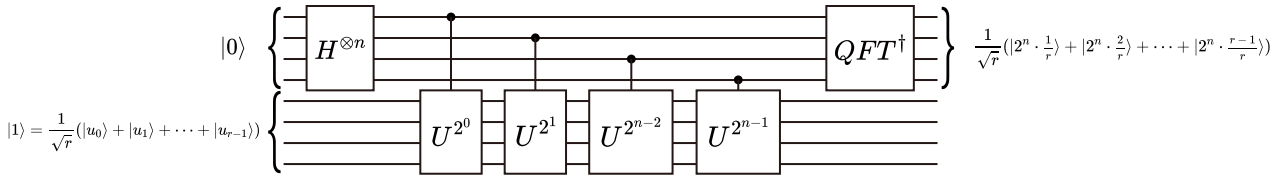

Рис. 2: Схема алгоритма Шора

2.4. Реализация алгоритма

Алгоритм был реализован на языке python с использованием библиотеки qiskit.Рассмотрим некоторые части.

Функция `c_amod15` - контроллирующий гейт преобразования $a^x mod N$. В примере мы использовали $a = 7$ и $N = 15$.

```
def c_amod15(power):
    U = QuantumCircuit(4)
    for iteration in range(power):
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
            for q in range(4):
                U.x(q)
    U = U.to_gate()
    U.name = "7^%i mod 15" % power
    c_U = U.control()
    return c_U
```

Функция `inversed_qft` - обраное квантовое преобразование Фурье.

```
def inversed_qft(n):
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT-1"
    return qc
```

При помощи данных функций мы уже можем собрать схему квантовую схему нахождения периода. Функция `shor_period` находит период квантовым алгоритмом Шора.

```
n_count = 8

qc = QuantumCircuit(QuantumRegister(8,'x'),
                    QuantumRegister(4,'f(x)'),
                    ClassicalRegister(8))

for q in range(n_count):
    qc.h(q)

qc.x(3+n_count)

for q in range(n_count):
    qc.append(c_amod15(2**q), [q] + [i+n_count for i in range(4)])

qc.append(inversed_qft(n_count), range(n_count))
qc.measure(range(n_count), range(n_count))
print(qc)
```

Функция `factorisation` реализует классическую часть алгоритма факторизации.

Полный исходный код программы находится в приложении А.

Заключение

В ходе практики мы усвоили принципы квантовых вычислений, изучили один из квантовых алгоритмов и рассмотрели его реализацию на языке python с использованием библиотеки qiskit.

Список литературы

[1] Сысоев С. С. Введение в квантовые вычисления. Издательство Санкт-Петербургского государственного университета, 2019. 7–109 с.

[2] Курс "Квантовые вычисления (Quantum computing)"[Электронный ресурс] URL: https://www.coursera.org/learn/kvantovyye-vychisleniya

[3] Документация IBM Quantum [Электронный ресурс] URL: https://quantum-computing.ibm.com/lab/docs/

[4] Документация к библиотеке Qiskit [Электронный ресурс] URL: https://qiskit.org/documentation/

Приложение А
Исходный код программы

```
import numpy as np
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister,
execute, circuit, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()

from fractions import Fraction
from math import gcd

def c_amod15(power):
    '''
        Преобразование a^2^power mod 15
        :param a: Случайное число, взаимно простое с N
        :param power: Степень
        :return: Вентиль преобразования
    '''
    U = QuantumCircuit(4)
    for iteration in range(power):
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
            for q in range(4):
                U.x(q)
    U = U.to_gate()
    U.name = "7^%i mod 15" % power
    c_U = U.control()
    return c_U
```

```
def inversed_qft(n):
    '''
        Обратное квантовое преобразование фурье
        :param n: Количество входов квантового вентиля
        :return: Вентиль обратного преобразования фурье
    '''
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT-1"
    return qc

def shor_period(a, N):
    '''
        Квантовый алгоритм Шора для нахождения
        периода функции a^x mod N
        :param a: Параметр функции a
        :param N: Параметр функции N
    '''
    n_count = 8

    qc = QuantumCircuit(QuantumRegister(8,'x'),
                        QuantumRegister(4,'f(x)'),
                        ClassicalRegister(8))

    # инициализация расчетных кубитов в состоянии |+>
    for q in range(n_count):
        qc.h(q)
```

```
    # вспомогательные регистры в состоянии |1>
    qc.x(3+n_count)

    # выполнение controlled-U операции
    for q in range(n_count):
        qc.append(c_amod15(2**q), [q] + [i+n_count for i in range(4)])

    # выполнение QFT-1 операции
    qc.append(inversed_qft(n_count), range(n_count))

    # измерение рассчетных кубитов
    qc.measure(range(n_count), range(n_count))

    # вывод схемы на экран
    print(qc)

    # симуляция
    qasm_sim = Aer.get_backend('qasm_simulator')
    t_qc = transpile(qc, qasm_sim)
    obj = assemble(t_qc, shots=2048)
    result = qasm_sim.run(obj, memory=True).result()
    readings = result.get_memory()

    # вычисление периода
    phase = int(readings[0], 2)/(2**n_count)
    frac = Fraction(phase).limit_denominator(N)
    r = frac.denominator
    return r


def factorization(N):
    '''
        Алгоритм факторизации чисел (поиск простых делителей числа)
        :param N: Число, необходимое факторизовать
```

```
    '''
    a = 7
    factor_found = False
    attempt = 0
    while not factor_found:
        attempt += 1
        print("\nПопытка %i:" % attempt)
        phase = shor_period(a, N)
        frac = Fraction(phase).limit_denominator(N)
        r = frac.denominator
        print("Период: r = %i" % r)
        if phase != 0:
            guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
            for guess in guesses:
                if guess not in [1, N] and (N % guess) == 0:
                    print('Простые делители числа', N, ':', guess)
                    factor_found = True
                else:
                    factor_found = False

N = 15
factorization(N)
```