МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет

им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки: «Фундаментальная информатика и информационные технологии»

ОТЧЕТ

по проекту

**Симуляция выполнения алгоритма Шора**

**на квантовом компьютере**

Выполнил: студент группы 381806-2

Куландин Денис Сергеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Научный руководитель: заведующий лабораторией интернета вещей

Линев Алексей Владимирович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Нижний Новгород

2020 г.

Содержание

[1 Введение 4](#_Toc39753235)

[2 Постановка задачи 5](#_Toc39753236)

[3 Теоретические основы 6](#_Toc39753237)

[3.1 Факторизация целых чисел 6](#_Toc39753238)

[3.2 Идентификация простого числа 6](#_Toc39753239)

[3.2.1 Перебор всех возможных делителей 6](#_Toc39753240)

[3.2.2 Тест Ферма 6](#_Toc39753241)

[3.2.3 Тест Рабина-Миллера 6](#_Toc39753242)

[3.3 Алгоритмы факторизации чисел 7](#_Toc39753243)

[3.3.1 Перебор всех возможных делителей 7](#_Toc39753244)

[3.3.2 Ро-алгоритм Полларда 7](#_Toc39753245)

[3.4 Квантовый компьютер 7](#_Toc39753246)

[3.4.1 Квантовая природа 7](#_Toc39753247)

[3.4.2 Квантовый бит 8](#_Toc39753248)

[3.4.3 Квантовые гейты 9](#_Toc39753249)

[3.5 Метод шифрования RSA 11](#_Toc39753250)

[3.5.1 Описание работы метода шифрования RSA 11](#_Toc39753251)

[3.6 Алгоритм Миллера 12](#_Toc39753252)

[3.6.1 Описание работы алгоритма Миллера 13](#_Toc39753253)

[3.6.2 Анализ алгоритма Миллера 13](#_Toc39753254)

[3.7 Алгоритм Шора 14](#_Toc39753255)

[3.7.1 Переход к квантовому компьютеру 14](#_Toc39753256)

[3.7.2 Описание алгоритма Шора 14](#_Toc39753257)

[3.7.3 Анализ алгоритма Шора 15](#_Toc39753258)

[3.7.4 Квантовое преобразование Фурье 15](#_Toc39753259)

[3.7.5 Возведение в степень по модулю на квантовом компьютере 18](#_Toc39753260)

[3.7.6 Оценка сложности квантовой части алгоритма 23](#_Toc39753261)

[4 Структура проекта 25](#_Toc39753262)

[4.1 Схема вычислений 25](#_Toc39753263)

[4.2 Модули и компоненты 26](#_Toc39753264)

[4.2.1 Класс Quantum 26](#_Toc39753265)

[4.2.2 Библиотека функций работы с простыми и составными числами 27](#_Toc39753266)

[4.3 Описание реализаций 27](#_Toc39753267)

[5 Способ использования 27](#_Toc39753268)

[6 Результаты экспериментов 28](#_Toc39753269)

[6.1 Корректность программы 28](#_Toc39753270)

[6.2 Производительность 29](#_Toc39753271)

[7 Заключение 30](#_Toc39753272)

[8 Литература 31](#_Toc39753273)

# Введение

В наши дни люди создают огромное количество новых изобретений. Одним из таких изобретение является квантовый компьютер. Он помогает работать с огромными количествами данных за короткое время. Сложность заключается в создании алгоритмов для квантового компьютера, так называемых квантовых алгоритмов. Одним из квантовых алгоритмов является алгоритм Шора, который позволяет осуществлять быструю факторизацию целых чисел.

Проблема факторизации целых чисел существовала очень долгое время. Ею занимались такие великие умы, как Лежандр, Эйлер, Гаусс. Все существующие алгоритмы для обычного компьютера решают задачу либо за экспоненциальное время, либо за субэкспоненциальное время. Квантовый компьютер с использованием алгоритма Шора решает эту задачу быстрее, чем любой из существующих алгоритмов факторизации числа для классического компьютера.

Факторизация целых чисел используется в криптографии, в алгоритмах, основанных на вычислительной сложности разложения числа на простые множители (RSA).

Сложность темы заключается в том, что алгоритмы для классического компьютера не применимы для квантового. Поэтому приходится разрабатывать новые алгоритмы. Это очень важно и актуально, ведь квантовый компьютер может давать невероятно большую скорость вычислений. Тот, кто первый адаптирует алгоритмы под квантовый компьютер, захватит весь мир.

# Постановка задачи

В данном проекте необходимо реализовать программу, которая выполняет факторизацию заданного числа при помощи симуляции квантового алгоритма Шора на обычном компьютере. Задача делится на 3 этапа:

1. Изучение теоретического материала: что такое факторизация, анализ алгоритмов проверки чисел на простоту, анализ алгоритмов факторизации чисел, суть квантового компьютера и его преимущества по сравнению с обычным компьютером, квантовые алгоритмы.
2. Реализация алгоритма Шора на классическом компьютере.
3. Реализация симулятора квантового компьютера, выполняющего ограниченный набор базовых операций.
4. Разработка программы факторизации числа, использующей симуляцию квантовой части алгоритма Шора.
5. Проведение экспериментов по оценке корректности и производительности.

# Теоретические основы

## Факторизация целых чисел

Факторизация целого числа – это разложение числа на произведение степеней его простых множителей. Число называется простым, если оно делится только на себя и на единицу, то есть имеет всего два делителя (единицу не принято считать простым числом). Из основной теоремы арифметики следует существование и единственность разложения числа на простые множители с точностью до порядка следования множителей. В отличие от задачи идентификации простоты числа, факторизация является вычислительно сложной задачей.

## Идентификация простого числа

Для того чтобы идентифицировать простое число нужно проверить существуют ли делители этого числа отличные от единицы и его самого. Существует несколько тестов на простоту, некоторые из них являются истинными, некоторые – вероятностными. Рассмотрим некоторые из них. Далее *N* – раскладываемое число.

### Перебор всех возможных делителей

Сложность:

Один из самых простых и очевидных алгоритмов проверки числа на простоту. Заключается в переборе всех возможных делителей числа из отрезка .

### Тест Ферма

Данный тест не случайно назван в честь великого математика Пьера Ферма, который в 17 веке открыл и доказал теорему. Малая теорема Ферма: если n – простое число и a – целое число, которое не делится на n, то . То есть, если найдется хотя бы одно , то n – составное.

Рассмотрим данный алгоритм с точки зрения теории вероятностей. - это реальная вероятность выполнения равенства при конкретном числе а. То есть после одного измерения вероятность того, что число n – простое равна .

Рассмотрим тест числа n на простоту с использованием *K* различных чисел *а*, которые не делятся на *n*. В случае прохождения всех *K* испытаний вероятность того, что наше число простое равна . Из формулы можно понять, что 50 различных чисел *а*, которые не делятся на *n*, достаточно для идентификации простоты числа *n*.

Поскольку для одного испытания нужно возвести число в степень по модулю, для реализации возведения числа в степень по модулю можно воспользоваться бинарным возведением числа в степень по модулю, которое работает за , где *B* – степень, в которую мы возводим число. Таким образом, сложность проверки числа на простоту равна . Данный тест не является точным, даже не смотря на полученную вероятностную оценку, так как существуют числа Кармайкла – числа, которые удовлетворяют сравнению для всех *а*, взаимно простых с *n*. Такие числа относительно редки, но доказано, что их существует бесконечно много. Наименьшее из чисел Кармайкла – 561.

### Тест Рабина-Миллера

Пусть *N*–простое число и , где d–нечётно. Тогда для любого , выполняется хотя бы одно из условий:

* существует целое число *r* < *s* такое, что .

Число *a* называется свидетелем простоты, и алгоритм проверки заключается в нахождении целого числа *S*, целого нечётного числа *d* и последующей выборке и проверке случайных чисел . Доказательство работы с точки зрения теории вероятностей схоже с доказательством для теста Ферма.

Сложность теста: .

## Алгоритмы факторизации чисел

Рассмотрим некоторые «простые» алгоритмы факторизации целых чисел. Далее *N* – раскладываемое число.

### Перебор всех возможных делителей

Сложность:

Простой и очевидный алгоритм факторизации чисел, соответствующий теореме о факторизации, которая гласит, что число может быть представлено произведением степеней его простых множителей. Алгоритм заключается в последовательном делении факторизуемого числа на числа из отрезка .

### Ро-алгоритм Полларда

Сложность:

Является вероятностным алгоритмом, позволяющим находить делитель составного числа *N*.

Шаг 1. Выбираем небольшое число и строим последовательность чисел

, определяя каждое следующее .

Шаг 2. Одновременно на каждом шаге вычисляем , где - всевозможные разности.

Шаг 3. Если , то вычисление заканчивается, и найденное на предыдущем шаге число d является делителем n. Если не является простым числом, то процедура поиска делителей продолжается с шага 1, взяв в качестве n число .

## Квантовый компьютер

### Квантовая природа

Квантовый компьютер – вычислительное устройство, которое использует явления квантовой механики (квантовая суперпозиция и квантовая запутанность) для передачи и обработки данных. Классическая машина Тьюринга эквивалентна универсальному квантовому компьютеру с точки зрения выполнения вычислений. Поскольку квантовый компьютер может использовать несколько входных состояний одновременно и одновременно выполнять программу в разных состояниях, классическая машина Тьюринга может рассматриваться как особый тип квантового компьютера, который может одновременно запускать программу только в одном начальном состоянии. Иными словами, классическая машина Тьюринга – это универсальный квантовый компьютер, в котором нет квантовой запутанности, помех, неопределенности вычислительных состояний в конце каждого элементарного вычислительного шага.

### Квантовый бит

В отличие от классического компьютера, квантовый компьютер оперирует не битами, а кубитами. Кубит – наименьший элемент хранения информации на квантовом компьютере. Допускает два собственных состояния и , но при этом может находиться и в их суперпозиции, то есть в состоянии , где – комплексные числа, удовлетворяющие соотношению . При любом измерении состояния кубита он случайно переходит в одно из своих собственных состояний. Вероятности перехода в то или иное состояние соответственно равны и . Кубиты могут быть запутаны друг с другом, то есть на них может быть наложена ненаблюдаемая связь, выражающаяся в том, что при всяком изменении над одним из нескольких кубитов остальные меняются согласованно с ним. Одним из состояний, которое может демонстрировать суть квантовой запутанности, называется состояние Белла.

Предположим, что у нас есть два независимых кубита, которые мы хотим объединить в одну систему:

|  |  |
| --- | --- |
| первый кубит: | , где |
| второй кубит: | , где |

Для того чтобы получить состояния кубитов в одной системе воспользуемся тензорным произведением состояний первого и второго кубитов, получим

Преобразуем: , где

Теперь рассмотрим состояние Белла:

Как можно заметить, такое состояние невозможно получить путем тензорного произведения двух независимых кубитов, но данное состояние можно получить путём применения двух квантовых гейтов (преобразование Адамара и Controlled NOT), которые будут рассмотрены ниже. Таким образом, показанный здесь пример демонстрирует то, что квантовая суперпозиция и квантовая запутанность кубитов в квантовом компьютере обеспечивает массовую параллельную обработку, экспоненциально превосходящую классические компьютеры.

Например, Питер Шор в 1994 году представил чрезвычайно мощный алгоритм, который делит полупростое число (число, являющееся произведением двух простых чисел) на два простых множителя, используя собственный параллелизм, присутствующий в квантовом компьютере, который возникает из суперпозиции и запутанности квантовой физики. Подробнее данный алгоритм будет описан ниже. По сравнению с наиболее эффективным алгоритмом целочисленной факторизации классического компьютера, известным нам, алгоритм Шора экспоненциально быстрее.

Как мы увидим далее, алгоритм Шора является особенно интересным, но все же опасным алгоритмом, потому что метод шифрования, используемый в защищенной сети, например, алгоритм шифрования RSA основано на практической невозможности взлома большого полупростого числа на два множителя. Последствия падения метода шифрования RSA варьируются от неисправности систем онлайн-банкинга до серьезной угрозы национальной безопасности.

### Квантовые гейты

Квантовый гейт – логическая операция над кубитом или системой кубитов. Один гейт над системой из *n* кубитов эквивалентен одновременному изменению комплексных чисел. Для того, чтобы описать и реализовать в последствии алгоритм Шора для квантового компьютера необходимо изучить и понять работу описанных ниже гейтов. В симуляторе реализованы все ниже перечисленные гейты.

#### NOT

Однокубитный гейт, который переводит кубит из состояния в состояние . Данную операцию над кубитом можно записать в матричной форме следующим образом:

, получим .

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: | https://upload.wikimedia.org/wikipedia/commons/thumb/2/23/Qcircuit_NOT.svg/150px-Qcircuit_NOT.svg.png |

#### Преобразование Адамара

Однокубитный гейт, который переводит кубит в равновероятные состояния.

Из состояния в состояние . В матричной форме:

или . Нетрудно заметить, что повторное применение этого гейта к этому же кубиту переведет его в изначальное состояние.

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: | https://upload.wikimedia.org/wikipedia/commons/thumb/1/1a/Hadamard_gate.svg/150px-Hadamard_gate.svg.png |

#### Phase Shift

Однокубитный гейт, который изменяет только состояние таким образом, что при измерении изменённого кубита, измерения будут такими же как и до применения гейта, хотя этот гейт изменит фазу состояния кубита. Пусть изначальное состояние: , после применения гейта: . В матричной форме: .

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: | File:Phase shift gate.png - Wikimedia Commons |

#### Reverse Phase Shift

Однокубитный гейт, который является обратным к описанному выше гейту Phase Shift. В матричной форме добавится всего лишь минус в степени экспоненты: .

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: |  |

#### CNOT(Controlled NOT)

Двукубитный гейт выполняет гейт NOT для второго кубита только если первый кубит находится в состоянии . В матричном виде:

. Из матричного вида становится понятно, как происходят изменения после применения гейта CNOT.

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: |  |

#### Controlled Phase Shift

Двукубитный гейт, выполняющий преобразование состояния второго кубита, когда первый кубит находится в состоянии . В матричном виде: . Нетрудно заметить, что ко второму кубиту применяется гейт Phase Shift, когда первый кубит в состоянии .

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: |  |

#### Reverse Controlled Phase Shift

Данный гейт является обратным к описанному выше гейту Controlled . В степени экспоненты появится минус. В матричном виде: .

|  |  |
| --- | --- |
| В квантовых схемах обозначается следующим образом: |  |

#### Fredkin gate (гейт Фредкина)

Трехкубитный гейт, выполняющий перестановку местами двух управляемых кубитов, только когда управляющий кубит в состоянии . Первый кубит – управляющий, второй и третий – управляемые. С помощью данного гейта возможна реализация любой обратимой логической функции. В матричном виде:

.

|  |  |
| --- | --- |
| В квантовых схемах обозначается: |  |

## Метод шифрования RSA

С развитием информационных технологий в последние десятилетия трудно представить себе жизнь без удобства доступа в Интернет. Информационные технологии лежат в основе современного образа жизни: начиная от электронной почты и социальных сетей и заканчивая банковскими онлайн-транзакциями на личном уровне, от управления местными офисами до международных разведывательных операций. Чтобы сохранить все эти транзакции в секрете, необходимо использовать быстрый, но безопасный способ шифрования информации. Наиболее популярным из используемых в настоящее время методов шифрования является метод шифрования RSA.

Сила метода шифрования RSA основана на теории чисел: на данный момент не существует быстрого алгоритма, который мог бы раскладывать огромное (больше 100 знаков) полупростое число в произведение двух простых его множителей. Мы спокойно можем получить произведение двух простых чисел, но не можем получить произведение двух простых чисел из полупростого числа. Вот почему данный метод шифрования является самым востребованным и невзламываемым.

### Описание работы метода шифрования RSA

Предположим, что у нас есть два длинных простых числа: *p* и *q*. Число является полупростым. Далее для числа *n* вычислим функцию Эйлера , которая отражает количество взаимно простых чисел с *n* от 1 до *n*. По свойству «мультипликативности» функции Эйлера получим: . Свойство функции Эйлера: если *p*–простое, то .

Из свойства следует, что .

Далее необходимо выбрать число e, которое впоследствии будет описывать открытый ключ. Число е такое, что:

Далее получаем число *d*, которое описывает закрытый ключ: . Говоря языком теории чисел, получим обратный к числу *е* по модулю *n*. Данное число легко найти с помощью расширенного алгоритма Евклида. Данный алгоритм позволяет найти решение диофантова уравнения второго порядка.

Пусть у нас есть диофантово уравнение второго порядка: , решение существует, так как мы выбрали такое *e*, что оно взаимно просто с . Запишем уравнение (1) по модулю:

*,* как видно из уравнения *x* – это как раз то самое, число которое нам нужно найти.

Итак, у нас есть два числа: *e* и *d*. Предположим, что есть два человека Алиса и Боб, которые хотят общаться в интернете абсолютно приватно. Алиса у себя локально сгенерирует два числа *e* и *d*, и пришлет Бобу пару чисел, которую мы будем называть открытым ключом: (*e, n*). Пару чисел (*d, n*) будем называть закрытым ключом, его Алиса оставит у себя. Допустим, что Боб хочет отправить приватное сообщение Алисе, для этого Боб возьмет сообщение, которое он хочет отправить, пусть *m*–это сообщение. Бобу необходимо возвести его сообщение в степень: . Затем, полученное число «*с*» он отправит Алисе. Алиса, при получении сообщения Боба сможет легко декодировать сообщение, которое отправил ей Боб, следующим образом: . Таким образом, ребята смогли приватно пообщаться в интернете.

Доказательство работы.

Поскольку числа *e* и *d* объединены следующим равенством: можем произвести анализ того, что происходит с исходным сообщением Боба во время шифрования и дешифрования. Зашифрованное сообщение: Сообщение, которое декодирует Алиса: (2). Подставим «» в (2), получим:

. Таким образом, , что является исходным сообщением Боба.

Предположим, что успешной приватной переписке Алисы и Боба захочет помешать ревнивая Ева. Единственное, что Ева может увидеть в канале связи, по которому общаются Алиса и Боб - это открытый ключ (*e, n*) и закодированное письмо Боба Алисе – *с*. Но для того, чтобы понять, что пишет Боб Алисе, Еве необходимо узнать закрытый ключ, которого нигде она найти не может. Единственным вариантом получить закрытый ключ будет получить те самые простые множители полупростого числа *n* – *p* и *q*. Так как длина числа n порядка 100 знаков, Ева будет искать числа *p* и *q* очень долго.

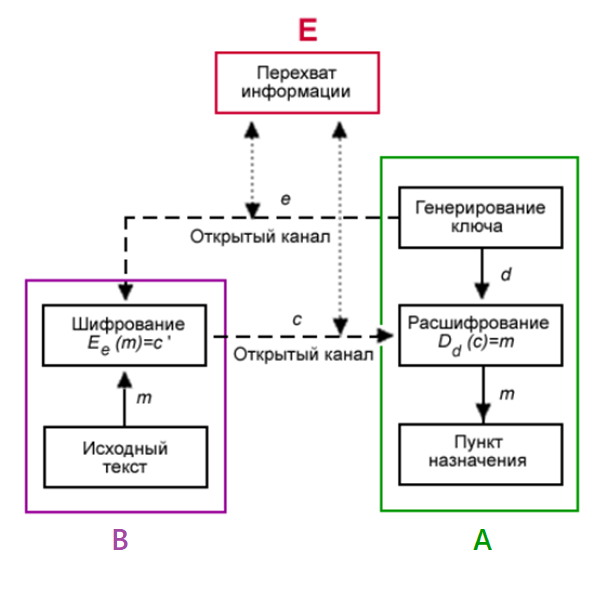


Рисунок 1. Описание метода шифрования RSA

## Алгоритм Миллера

Одним из классических алгоритмов, который может быть использован для того, чтобы помочь Еве факторизовать полупростое число, является алгоритм Миллера. Данный алгоритм интересен для нас тем, что квантовой версией этого алгоритма является алгоритм Шора, симуляцию которого необходимо осуществить в рамках данного проекта. Следовательно, понимание алгоритма Миллера имеет огромное значение при исследовании квантового алгоритма Шора. Несмотря на то, что в качестве классического алгоритма алгоритм Миллера неэффективен при разложении полупростого числа, детали, обсуждаемые в этом разделе, послужат вспомогательной ступенькой к пониманию алгоритму Шора.

### Описание работы алгоритма Миллера

Предположим, что мы хотим получить делитель числа *n*:

1. Если *n* четное, то вернуть делитель 2.
2. С помощью бинарного поиска на обычном компьютере определим, может ли *n* представлено как , где . Если так, возвращаем делитель *p*.
3. Выбираем случайное число *а*, такое что: . Если , возвращаем полученный НОД в качестве делителя.
4. Вычисляем период *w* числа *а* по модулю *n*. Период *w-* минимальное положительное число такое, что .
5. Если w нечетное или , то вернуться на шаг 3, так как число *а* нам не подходит. Иначе вычисляем и . Проверяем, является ли одно из полученных чисел делителем числа *n* и возвращаем его.

### Анализ алгоритма Миллера

Для начала введем понятие нетривиального квадратного корня из *n*.

Пусть *x* – нетривиальный квадратный корень из *n*,значит, выполняется следующее: .

Лемма. Пусть *x* – нетривиальный квадратный корень из *n*. Тогда число или является делителем *n*, отличным от 1 и *n*.

Доказательство. Рассмотрим первое равенство . Перенесем единицу влево и разложим как разность квадратов, получим: . Из второго условия нетривиального делителя получаем, что ни один из сомножителей и по условию на n не делится, поэтому некоторые простые множители n должны войти в один сомножитель, а другие – в другой. В таком случае и будет содержать только некоторые сомножители *n*.

Таким образом, если в алгоритме Миллера находится четный период для числа а, то число будет являться нетривиальным квадратным корнем из *n*, тем самым из этого мы сможем получить один из делителей *n*.

Нетрудно заметить, что самым трудоёмким в этом алгоритме является поиск числа периода числа. Сложность алгоритма Миллера возрастает экспоненциально в зависимости от увеличения полупростого числа. На данный момент мы дошли до основной задачи во взломе метода шифрования RSA. В следующем разделе мы увидим, как квантовый компьютер позволяет экстремально быстро найти период числа по сравнению с другими существующими способами.

## Алгоритм Шора

### Переход к квантовому компьютеру

Для того чтобы преобразовать алгоритм Миллера в квантовую версию, необходимо заметить следующее: . Исходя из введенной нами периодической функции, видно, что периодом этой функции является период *w* числа *x* (поэтому так и называется). Поэтому, задача нахождение периода числа переходит в задачу поиска периода некой периодической функции.

У кого-то может возникнуть вопрос: «Зачем нам квантовый компьютер, если период функции можно найти с помощью обычного компьютера?». Предположим, что у нас есть число длиной 100 знаков, которое мы хотим факторизовать. Тогда функция будет неотличима от случайной функции с точек. Если мы попытаемся найти периодичность данной функции, то нас ожидает провал. С поставленной задачей нам поможет справиться преобразование Фурье. На практике с обычным компьютером невозможно выполнить преобразование Фурье для точек данных. Для загрузки и выполнения преобразования Фурье, а затем для хранения преобразованных данных требуется огромное количество времени, которое на порядки больше возраста нашей вселенной.

Однако квантовый компьютер может работать с несколькими входными состояниями одновременно и выполнять преобразование Фурье для всех чисел одновременно, используя массивный параллелизм, возникающий в результате квантовой суперпозиции и квантового запутывания. Именно в этом моменте квантовый компьютер побеждает классический компьютер и, в конечном итоге, приводит к полиномиальному масштабированию числа вычислительных шагов.

### Описание алгоритма Шора

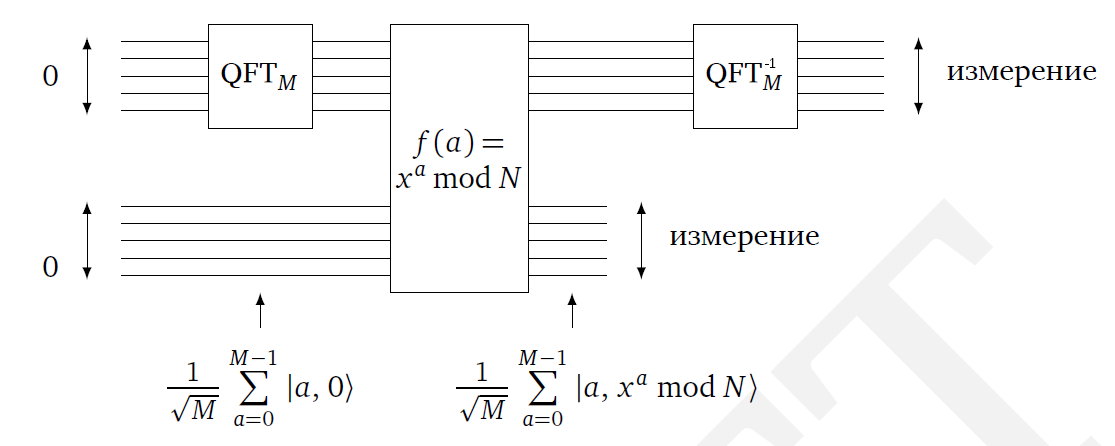
На вход алгоритму подается числа *x* и *n*, на выходе алгоритм должен выдавать период числа *x* по модулю *n*. Далее будет описываться квантовая схема алгоритма, в ней система из кубитов состоит из двух регистров: первый содержит кубитов, второй – кубитов, где *M*–ближайшая к *n* сверху степень двойки.

Рисунок 2. Квантовая схема алгоритма Шора

Алгоритм состоит из нескольких шагов:

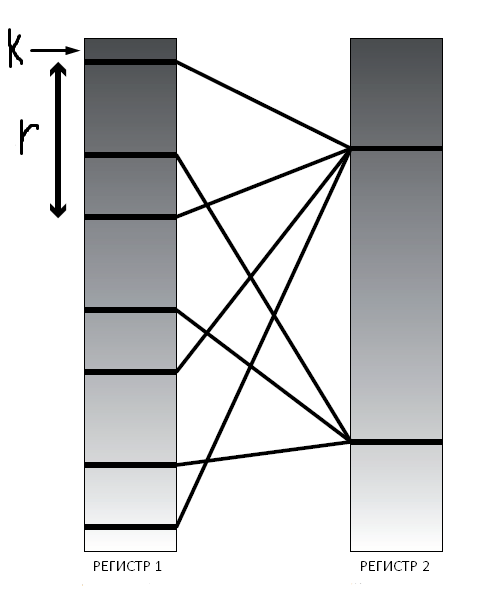
1. Слова из нулевых битов помещаем в два квантовых регистра, первый из которых вмещает в себя остаток по модулю *M*, а второй – по модулю *N*.
2. Применяем квантовое преобразование Фурье к первому регистру и получим в нем суперпозицию . На данном шаге в первом регистре все кубиты переводятся в равновероятные состояние, это эквивалентно тому, если бы мы к каждом кубиту первого регистра применили преобразование Адамара отдельно.
3. Вычисляем с использованием квантовой схемы, получаем суперпозицию .
4. Измеряем содержимое второго регистра. Так как при измерении любого кубита или регистра квантовой схемы кубиты изменяют своё состояние в зависимости от их прошлого состояния, то после измерения первый регистр содержит периодическую суперпозицию , где *k* – некоторый сдвиг между 0 и *r*–1, а *r* – искомый период *x* по модулю *n*. Для того чтобы понять, что происходит, лучше взглянуть на рисунок 3.
5. Применяем обратное преобразование Фурье к первому регистру, тем самым избавляясь от *k* и получая в первом регистре финальную периодическую функцию.
6. Производим измерение первого регистра, получим индексы кубитов, которые находятся в ненулевых состояниях. Пусть *g* – наибольший общий делитель полученных индексов. Таким образом, получаем период функции равный .

Рисунок 3. Иллюстрация 4-го шага алгоритма Шора.

### Анализ алгоритма Шора

Покажем как использовать периодичность функции для получения суперпозиции с периодическими коэффициентами.

* Возьмем два квантовых регистра, в которых изначально все кубиты находятся в нулевом состоянии.
* Применим квантовое преобразование Фурье к первому регистру. В результате в нём появится суперпозиция всех значений от 0 до с коэффициентами при каждом из них.
* Применяем квантовую схему, соответствующую классическому вычислению функции . При этом в качестве входа используем первый регистр, а результат помещаем во второй регистр (изначально содержит все нули с амплитудой 1). В результате получаем состояние двух регистров, равное .
* Производим измерение второго регистра. В результате его значение будет фиксировано, а в первом регистре окажется суперпозиция состояний с периодическими амплитудами. Раз функция имеет период , то значением второго регистра после измерения будет одно из чисел при *k* от 0 до . А в первом регистре останутся с равными коэффициентами те значения a, которые совместимы с измеренным состоянием второго регистра. Поскольку в последовательности повторения случаются только через шагов, то в первом регистре останется периодическая последовательность (одинаковые коэффициенты, которые разделены нулями), и обратное преобразование Фурье позволит найти период , что нам и требовалось.

### Квантовое преобразование Фурье

Квантовое преобразование Фурье выполняет ту же самую задачу, что и классическое дискретное преобразование Фурье, только на квантовом компьютере. Классическое дискретное преобразование Фурье – это операция, сопоставляющая одной функции вещественной переменной другую функцию вещественной переменной (переводит изначальную функцию в пространство Фурье). Эта новая функция описывает коэффициенты (амплитуды) при разложении исходной функции на элементарные составляющие – гармонические функции с разными частотами (лично для меня, оказалось проще понять преобразование Фурье на примере музыкального аккорда, который может быть выражен в виде суммы музыкальных звуков, которые его составляют). Таким образом, с полученной функцией мы можем производить любые операции, которые мы могли бы производить с исходной функцией, но эти операции окажутся проще выполнимыми с полученной функцией. После применения всех операций необходимо использовать обратное преобразование Фурье для того, чтобы вернуться к функции в привычном для нас виде.

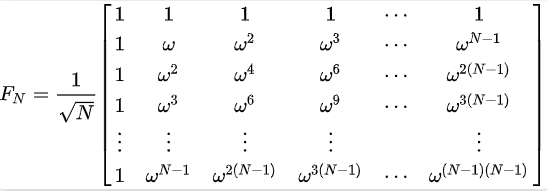
Классическое преобразование Фурье переводит вектор в вектор по следующей формуле:

Аналогично, квантовое преобразование Фурье переводит квантовое состояние в квантовое состояние по следующей формуле:

Так как это поворот по комплексной плоскости на определенный угол, то обратное квантовое преобразование Фурье будет вычисляться по следующей формуле:

Таким образом, изначальное состояние после применения квантового преобразования Фурье перейдет в состояние:

Эквивалентно формуле, квантовое преобразование Фурье может быть рассмотрено в виде матрицы унитарного преобразования:



где.

Для того чтобы получше понять приведу пример для и , тогда матрица преобразования будет выглядеть следующим образом:

Для того чтобы выполнить квантовое преобразование Фурье на квантовом компьютере необходимо собрать квантовую схему, которая будет выполнять преобразование. Такая схема есть, она состоит из двух описанных выше гейтов: преобразование Адамара и Controlled Phase Shift . Квантовая схема выглядит следующим образом:

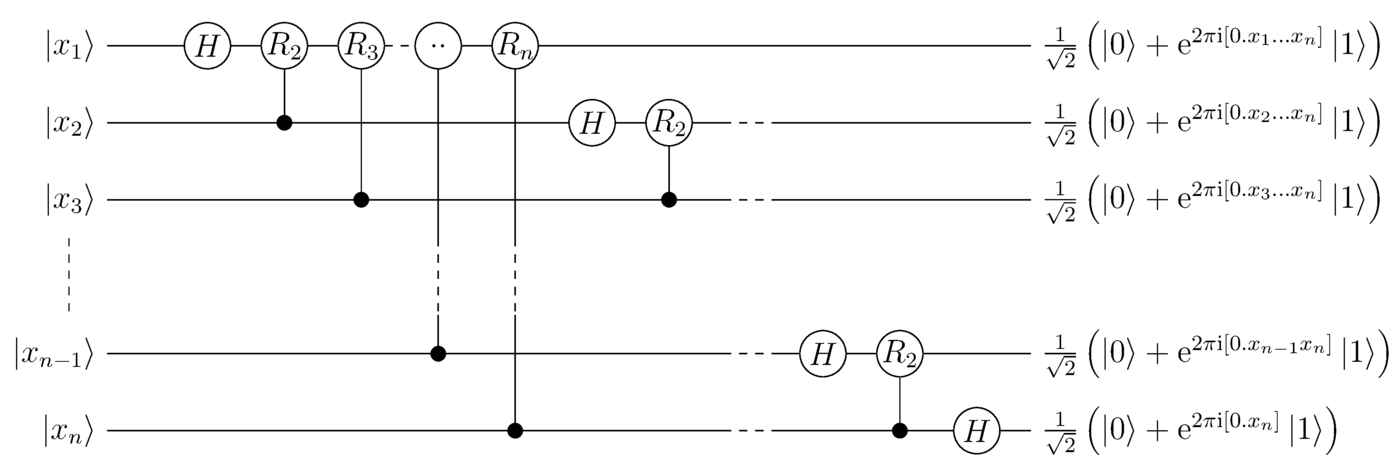
**

Рисунок 3. Схема квантового преобразования Фурье

Доказательство работы квантовой схемы для квантового преобразования Фурье. **В рамках доказательства операция (умножить) - будет обозначать тензорное произведение.**

Квантовое преобразование Фурье, предположительно, может быть реализовано для любого *N*, но в рамках доказательства будем считать .

У нас есть ортонормированный базис, состоящий из векторов: .

Базисные состояния перечисляют все возможные состояния кубитов:

, где показывает в каком состоянии находится кубит , в 0 или 1. Будем считать, что квантовое преобразование Фурье индекса базисного состояния упорядочивает лексикографически возможные состояния кубитов, то есть конвертируя из двоичной системы счисления в десятичную получим:

Для краткости будем обозначать в виде .

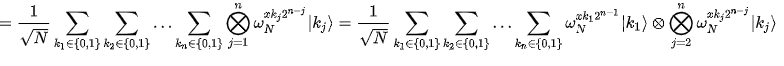
Например, или .

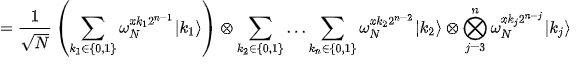
С таким обозначением процесс квантового преобразования Фурье можно кратко записать следующим образом:

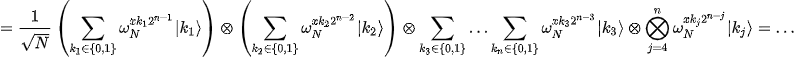
(1)

Данная формула может быть получена, если мы запишем преобразование Фурье в двоичном виде:











Где

Пусть

Заметим, что и

Исходя из этого, получим:

Перепишем, получим представление квантового преобразования Фурье с учетом того, что сейчас получено:

Если сравнить полученное выражение с (1), то можно сделать вывод, что это одно и тоже, значит квантовое преобразование Фурье имеет место быть.

Рассмотрим квантовую схему преобразования Фурье (рисунок 3).

Операция над *n* кубитами может преобразована в тензорное произведение *n* однокубитных операций. Фактически, каждая из этих операций с одним кубитом может быть эффективно реализована с использованием гейтов преобразования Адамара и Controlled Phase Shift. Первому кубиту необходимо использовать одно преобразование Адамара и (*n* – 1) Controlled Phase Shift, второму – одно преобразование Адамара и (*n*– 2) Controlled Phase Shift и т.д. Таким образом, получим необходимых гейтов, каждый из которых работает за *O*(1). Значит квантовое преобразование Фурье может быть реализовано за полиномиальное время на квантовом компьютере.

### Возведение в степень по модулю на квантовом компьютере

На шаге 3 в алгоритме Шора упоминается квантовый гейт, который производит возведение *x* в степень *a* по модулю *n*. В данном разделе содержится объяснение того, как устроен и работает данный гейт.

Прежде всего, давайте разберемся как работает возведение в степень по модулю. Возведение в степень по модулю отличается от обычного возведения в степень лишь тем, что в конце вычислений мы берем остаток от деления нашего ответа на некоторый модуль. Само возведение в степень может быть описано циклическое повторение одного и того же действия некоторое количество раз. Данным действием является умножение. Таким образом, задача возведения в степень сводится к задаче умножения чисел. Рассмотрим умножение двух чисел. Умножение двух чисел может быть описано как циклическое повторение сложения некоторое количество раз. Таким образом, получаем, что если мы научимся складывать два числа по модулю на квантовом компьютере, то задача возведения в степень по модулю будет частично решена.

#### Сложение по модулю

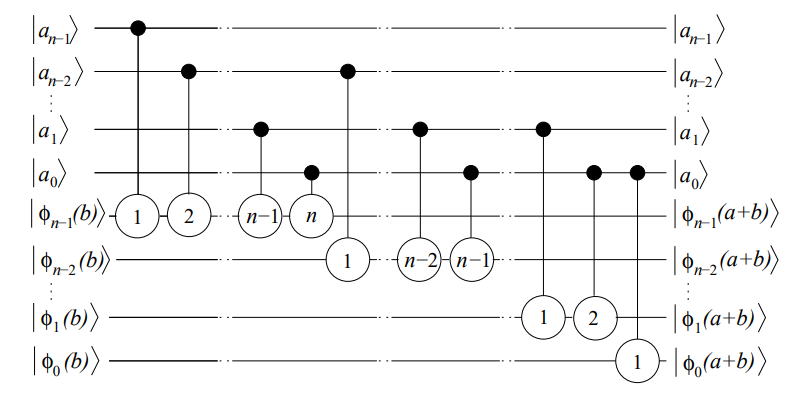
Для реализации квантового гейта, который выполняет сложение двух чисел по модулю нам пригодится знание лишь об одном стандартном гейте – Controlled Phase Shift. Предположим, что мы хотим получим .

Рисунок 4. Иллюстрация квантового гейта сложения по модулю.

Рассмотрим схему. На вход данной схеме идет два регистра, первый представляет из себя число *а*, второе – число *b*, надо которым уже сделали квантовое преобразование Фурье. Прошу заметить, что количество кубитов в каждом из регистров должно быть равно , это сделано для того, чтобы идентифицировать переполнения (увидим далее). После использования гейта первый регистр не изменится, во втором регистре будет результат в пространстве Фурье.

Кратко данный гейт будем обозначать следующим образом:

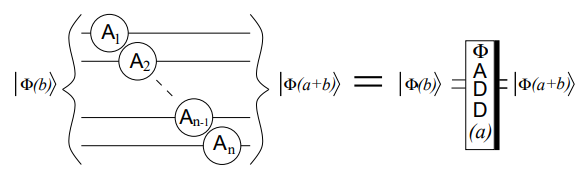
**

Рисунок 5. Иллюстрация квантового гейта сложения по модулю в краткой форме.

Гейты представляют из себя вычисленные комбинации гейтов Controlled Phase Shift.Черная полоса справа от гейта ФADD(a) обозначает, что происходит сложение. Если полоса будет слева, то гейт будет производить обратное действие, в нашем случае вычитание.

#### Вычитание по модулю

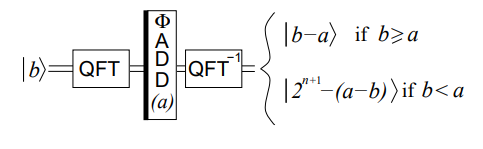


Рисунок 6. Иллюстрация квантового гейта вычитания по модулю.

Как и говорилось выше, черная полоса слева обозначает вычитание. Здесь необходимо рассмотреть два случая:

* , в этом случае получим положительное число, которое и будет финальным ответом
* , в этом случае нам пригодится тот запасной (*n* + 1) кубит, который предназначен как раз для этого случая. В результате получим .

#### Сложение по модулю

Заранее будем считать, что . В этом случае схема будет выглядеть немного сложнее:

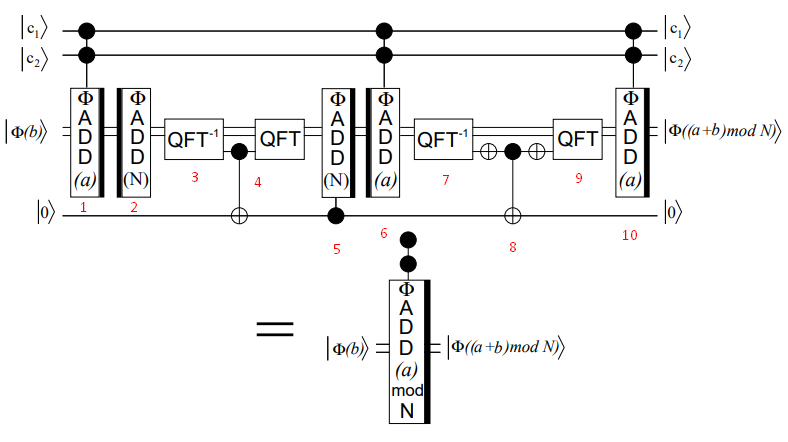


Рисунок 7. Иллюстрация квантового гейта (a + b) mod N.

Для работы данного гейта необходимо 4 регистра: первый, второй и четвертый представляют из себя регистр, состоящий из одного кубита, третий – число *b*, над которым уже сделано квантовое преобразование Фурье. Первый и второй регистры называются контролирующими, если они оба находятся в состоянии ,то ФADD будет выполняться, иначе – нет. Четвертый регистр представляет из себя кубит, который будет нести ответственность за переполнение и прочие ошибки во время выполнения гейта.

Рассмотрим поведение регистров во время выполнения гейтов по шагам. Будем считать, что контролирующие кубиты находятся в состоянии .

1. Складываем число *b* с числом *a*. Получим
2. Вычитаем из полученного числа *N*. Получим . В данном случае логично будет рассмотреть два варианта:
   1. , в таком случае кубит будет в состоянии .
   2. , в этом случае кубит будет в состоянии , это значит, что мы получили переполнение и наш результат равен .
3. Производим обратное преобразование Фурье
4. Если последний бит в полученном числе равен , то в таком случае мы знаем, что произошло переполнение. Ставим 4 регистр в состояние .Производим квантовое преобразование Фурье для дальнейших арифметических операций.
5. Прибавим к имеющемуся результату *N*, только если 4 регистр находится в состоянии .В данном случае результат из пункта 2.1 не изменится, а вот к результату из пункта 2.2 прибавится *N*, получим: .

Дальнейшие шаги необходимы для того, чтобы «почистить» 4 регистр, то есть перевести его в состояние . На данном этапе мы уже получили результат .

1. Вычтем из результата *a*.
   1. , такое возможно только если пункт 2.2 имел место быть. В таком случае получим в последнем кубите состояние .
   2. , последний кубит в состоянии .
2. Производим обратное преобразование Фурье
3. Применяем к последнему кубиту третьего регистра гейт NOT, затем Conrolled NOT совместно с четвертым регистром, затем вновь NOT.
   1. Последний кубит в состоянии . После гейта NOT, последний кубит станет , в данном случае четвертый регистр сейчас в состоянии (из шага 2.2), Controlled NOT переведет четвертый регистр в состояние , повторный гейт NOT переведет последний кубит третьего регистра в исходное состояние.
   2. Последний кубит в состоянии . Так как четвертый регистр в состоянии , то использование трех гейтов ничего не изменит.
4. Производим квантовое преобразование Фурье
5. Прибавляем к результату *а*, чтобы получить желаемый результат.

В итоге после применения гейта получаем в пространстве Фурье. Четвертый регистр используется для идентификаций ошибок во время выполнения гейтов, это позволяет быстро понять, где произошла ошибка.

#### Умножение по модулю

Для того чтобы научиться перемножать числа в рамках квантового компьютера прибегнем к некоторой хитрости:

Переводом числа в двоичную систему счисления мы свели операцию умножения к операции сложения (а это мы уже умеем из прошлого раздела).

Исходя из этого, квантовая схема примет вид:

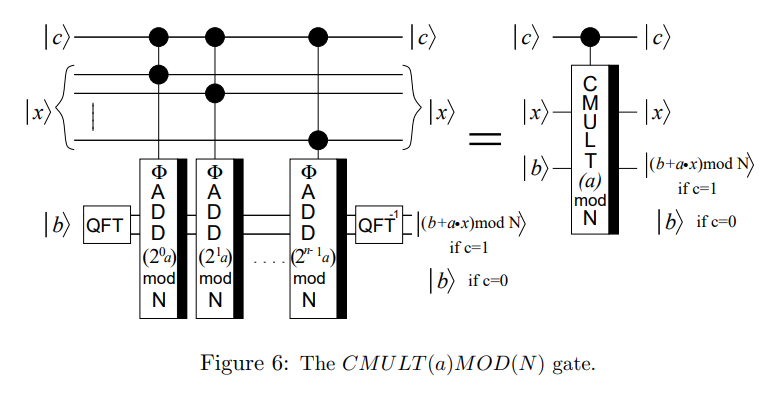


Рисунок 8. Иллюстрация квантового гейта (b + ax) mod N.

Из данной схемы можно заметить, что тот второй контролирующий кубит, который использовался в сложении по модулю (прошлом разделе) сейчас отражает в каком состоянии находится кубит регистра, отвечающий за число . То есть сложение будет выполняться только если кубит из регистра, который отвечает за число, находится в состоянии . На выходе регистр числа не изменится, изменится регистр числа . В этом гейте получаем на выходе в пространстве обычных чисел, так как преобразование Фурье было выполнено внутри гейта.

#### Возведение в степень по модулю

Прибегнем к похожей хитрости из прошлого раздела только для использования в возведении в степень:

Исходя из этого, квантовая схема примет вид:

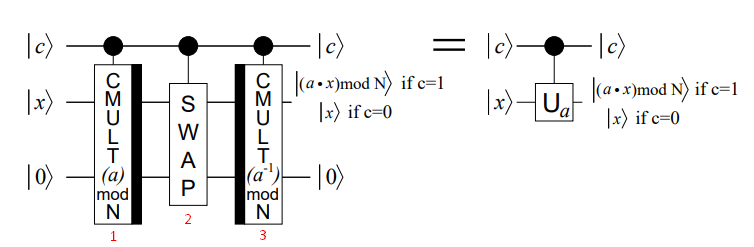


Рисунок 9. Иллюстрация квантового гейта UnitCMULT.

На вход данному гейту идут три квантовых регистра: первый – контролирующий, второй и третий – регистры, представляющие числа (*x* и *b*).В нашем случае *b* = 0, так как нас интересует только чистое (без прибавления иных чисел) умножение числа по модулю.

Рассмотрим работу гейта по шагам. Считаем, что контролирующий первый регистр в состоянии .

1. Выполняем умножение чисел. На выходе из второго регистра получим , а из третьего – .
2. Меняем местами второй и третий кубит.
3. Выполняем гейт, обратный к умножению по модулю (не деление!). (Пусть на вход этому гейту будут числа *x* и 0, гейт выполняет умножение относительно числа *а*. Таким образом на выходе гейта получим .) На вход данному гейту во втором регистре будет , в третьем регистре - . На выходе из данного гейта во втором регистре получим , в третьем регистре – .

В качестве гейта Controlled SWAP в данной схеме, будем использовать гейт Фредкина (Fredkin gate), описанный в разделе 1.4.3.8.

Таким образом, мы научились выполнять возведение в степень по модулю на квантовом компьютере. Итоговая квантовая схема для алгоритма Шора будет выглядеть следующим образом:

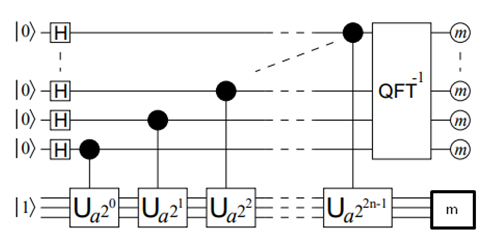


Рисунок 10. Квантовая схема алгоритма Шора

### Оценка сложности квантовой части алгоритма

Проанализируем сложность данной квантовой схемы для факторизации *n*-битного числа *N*. Анализ отслеживает количество кубитов, количество гейтов и временную сложность квантовой схемы. Для подсчета временной сложности будем считать, что можно будет одновременно применять разные квантовые гейты, которые действуют на разные кубиты квантового компьютера. Однако будем считать невозможным использование одного кубита в качестве управляющего для нескольких операций за один шаг. В квантовой схеме используются однокубитные квантовые гейты, двукубитные гейты (CNOT, Controlled Phase Shift), трехкубитный гейт (Fredkin gate). Эти гейты могут быть реализованы с использованием постоянного числа однокубитных гейтов и гейтов CNOT, поэтому будем считать и рассматривать их в качестве элементарных квантовых гейтов.

Гейт ФADD требует кубитов, имеет постоянное количество операций , количество используемых квантовых гейтов равно . Использование кубитов необходимо для того, чтобы избежать переполнений.

Гейт ФADD mod *N*, который имеет два контролирующих кубита, требует кубита. Количество операций гейта , так как в нем используется квантовое преобразование Фурье, квантовая схема которого была описана в разделе 1.7.4. Количество используемых гейтов равно .

Гейт CMULT требует кубита. В процессе выполнения гейта используется *n* раз гейт ФADD *mod N*, поэтому количество операций гейта составит . Количество используемых гейтов составляет .

Гейт UnitCMULT, в процессе выполнения которого дважды используется гейт CMULT и гейт Controlled SWAP, поэтому количество используемых кубитов , количество операций останется . Количество используемых гейтов составляет .

Для выполнения полной квантовой схемы необходимо кубитов. Так как в процессе выполнения раз используется гейт UnitCMULT, то итоговое количество операций схемы составит . Количество используемых гейтов во всей квантовой схеме равно .

# Структура проекта

## Схема вычислений

Нет

Число N

Число простое?

Да

Число простое

Генерация числа «а»

Конструирование квантовой схемы

Симуляция квантовой части алгоритма Шора

Вычисление наибольших общих делителей

Одно из полученных чисел является делителем числа N?

Нет

Да

Делитель числа

## Модули и компоненты

Класс Quantum

Библиотека функций работы с простыми и составными числами

### Класс Quantum

В данном классе реализована симуляция квантовой части алгоритма.

Поля класса:

* Vector<complex<double>> firstRegister - первый регистр (2n кубитов)
* Vector<complex<double>> secondRegister - второй регистр (n + 1 кубит)
* Vector<complex<double>> firstRegister - третий регистр (n + 1 кубит)
* Complex<double> fourthRegister- четвертый регистр (1 кубит)
* Int a – сгенерированное число
* Int N – число, делители которого хотим найти

Методы класса:

* int reverseNumber(int x, int len); - переворот двоичного представления числа х длины len
* string perevod(int x, int k); - перевод числа из десятичной системы счисления в k-ую
* vector< Complex<double> > QFFT(const vector< Complex<double> > & input, int cnt\_qubits); - прямое квантовое преобразование Фурье
* vector< Complex<double> > ReverseQFFT(const vector< Complex<double> > & input, int cnt\_qubits); - обратное преобразование Фурье
* vector< Complex<double> > Hadamar(const vector< Complex<double> > & a, int ind); - гейт преобразования Адамара
* vector< Complex<double> > CNOT(const vector< Complex<double> > & a, int x, int y); - гейт CNOT
* vector< Complex<double> > PhaseShift(const vector< Complex<double> > & a, int ind, int m); - прямой гейт Phase Shift
* vector< Complex<double> > ReversePhaseShift(const vector< Complex<double> > & a, int ind, int m); - обратный гейт Phase Shift
* vector< Complex<double> > Controlled\_Rm(const vector< Complex<double> > & a, int x, int y, int m); - прямой гейт Controlled Phase Shift
* vector< Complex<double> > ReverseControlled\_Rm(const vector< Complex<double> > & a, int x, int y, int m); - обратный гейт Controlled Phase Shift
* vector< Complex<double> > Fredkin(const vector< Complex<double> > &a, int x, int y, int z); - гейт Фредкина
* vector< Complex<double> > FADD(vector< Complex<double> > b, int a); - прямой гейт ФADD
* vector< Complex<double> > ReverseFADD(vector< Complex<double> > b, int a); - обратный гейт ФADD
* void FADD\_modN(Complex<double> &\_1, Complex<double> &\_2, vector< Complex<double> > &b, Complex<double> &\_4, int a, int N, int cnt\_qub); - прямой гейт ФADD mod N
* void ReverseFADD\_modN(Complex<double> &\_1, Complex<double> &\_2, vector< Complex<double> > &b, Complex<double> &\_4, int a, int N, int cnt\_qub); - обратный гейт ФADD mod N
* void CMULT\_modN(Complex<double> &\_1, vector< Complex<double> > & x, vector< Complex<double> > &b, Complex<double> &\_4, int a, int N, int cnt\_qub); - прямой гейт CMULT
* void ReverseCMULT\_modN(Complex<double> &\_1, vector< Complex<double> > & x, vector< Complex<double> > &b, Complex<double> &\_4, int a, int N, int cnt\_qub); - обратный гейт CMULT
* void UnitCMULT\_modN(Complex<double> &\_1, vector< Complex<double> > & x, vector< Complex<double> > & b, Complex<double> & \_4, int a, int N, int cnt\_qub); - гейт UnitCMULT
* int Shor(); - алгоритм Шора
* vector<cd> measure\_cubit(const vector< Complex<double> > & a, int index); - измерение системы кубитов

### Библиотека функций работы с простыми и составными числами

Реализованы следующие функции:

* Бинарное умножение по модулю
* Бинарное возведение в степень
* Алгоритм Евклида
* Расширенный алгоритм Евклида
* Генерация случайного числа в указанном промежутке
* Классическая проверка на простоту числа
* Тест Ферма проверки на простоту числа
* Тест Рабина-Миллера проверки на простоту числа

## Описание реализаций

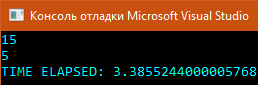
Проект реализован в MS Visual Studio 2017. Включает в себя следующие файлы:

* Main.cpp
* solve.h, solve.cpp
* Quantum.h, Quantum.cpp – квантовая часть
* algoNumbers.h, algoNumbers.cpp – библиотека работы с числами

Сборка проекта выполняется в Visual Studio x64/Release. Исходные коды проекта находятся здесь – <https://github.com/OnlyDeniko/Shor-algorithm>

# Способ использования

* Консольное приложение.
* Компиляцию рекомендуется проводить с использованием x64/Release
* Запуск может осуществляться из командной строки или из проводника.
* После запуска необходимо ввести число (от 2 до 127), для которого хотим найти делитель.
* Программа выводит результат.

Пример вывода программы

# Результаты экспериментов

Проведены эксперименты двух типов: корректность программы и производительность.

Тестовая платформа:

* OS Windows 10
* CPU: Intel Core i5-7500, теоретическая мощность 217,6 Gflops
* RAM: 24 гб, 2133 МГц, DDR4, пиковая скорость передачи данных 17066,67 Мб/с
* Компилятор: VS 2017

## Корректность программы

|  |  |  |
| --- | --- | --- |
| Число N | На каком шаге найден свидетель комплексности | Найденный делитель |
| 15 | 1 | 5 |
| 21 | 1 | 3 |
| 33 | 1 | 3 |
| 35 | 1 | 5 |
| 39 | 1 | 3 |
| 45 | 1 | 15 |
| 51 | 1 | 17 |
| 57 | 1 | 3 |
| 63 | 1 | 3 |
| 65 | 1 | 5 |

Вывод: Во всех случаях симуляция работает верно.

## Производительность

|  |  |
| --- | --- |
| Число N | Время, сек |
| 15 | 3.3855 |
| 21 | 32.126 |
| 33 | 282.0811 |
| 35 | 371.085 |
| 45 | 326.999 |
| 51 | 282.712 |
| 65 | 3006,052 |

Алгоритм Шора также был реализован для библиотеки симуляции квантовых вычислений QuEST (один из наиболее известных пакетов). Производительность моей реализации в 10 раз ниже, чем при использовании QuEST. Если оценивать производительность с точки зрения flops, то моя реализация достигает 1% от теоретически возможной производительности тестовой системы.

# Заключение

В рамках выполнения проекта выполнено следующее:

1. Реализованы следующие алгоритмы
   * Алгоритм Миллера (С++)
   * Симуляция квантового алгоритма Шора (С++, с использованием OpenMP)
   * Тест Ферма
   * Тест Рабина-Миллера
   * Алгоритм шифрования RSA
   * Операции модульной арифметики
   * Исходный код реализации: <https://github.com/OnlyDeniko/Shor-algorithm>
2. Прочитано и изучено большое количество материала по следующим темам:
   * Факторизация целых чисел
   * Теория чисел
   * Методы шифрования информации
   * Квантовые компьютеры
   * Квантовые кубиты, гейты
   * Преобразование Фурье
   * Алгоритмы Миллера и Шора

Для того чтобы запускать симуляцию квантового алгоритма Шора на обычном компьютере необходимо понимать, что будет потрачено огромное количество ресурсов компьютера. А именно, при попытке факторизовать *n*-битное число будет байт оперативной памяти и, теоретически, будет составлять количество операций. На практике время выполнения существенно меньше, так как реализация использует библиотеку OpenMP, позволяющая распараллеливать выполнение программы. В связи с ограниченностью ресурсов удалось провести эксперименты только для 7 и меньше битных чисел. На данный момент существуют квантовые компьютеры (IBM), на которых возможно «честно» реализовать квантовый алгоритм Шора для чисел до 15.

Дальнейшие направления работы:

1. Реализация варианта алгоритма Шора, использующего кубитов (найдена и разобрана статья с его описанием).
2. Повысить производительность симулятора.
3. Разработка, реализация и симуляция алгоритма Шора с использованием библиотеки QuEST на обычном компьютере.
4. Реализация алгоритма плотного кодирования для обычного и квантового компьютеров.
5. Использование и тестирование алгоритмов на квантовом компьютере (IBM Experience).

# Литература

1. Shor P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer // Foundations of Computer Science : Conference Publications, 1997, pp. 1484–1509.
2. Bressoud, D. M. Factorization and Primality Testing, N. Y.: Springer-Verlag, 1989,pp. 153
3. Bakhitiari M., Maarof M. A. Serious Security Weakness in RSA Cryptosystem // IJCSI, 2012, pp. 175–178.
4. H. Cohen, H. W. Lenstra, Jr. Primality Testing and Jacobi Sums // Mathematics of Computation, 1984, pp 165.
5. A. M. Steane, E. G. Rieffel. Beyond Bits: The future of Quantum Information Processing // IEEE Computer, 2000, pp. 38-45.
6. T.D. Ladd, Y. Nakamura. Quantum computing // Nature, 2010, pp 45 – 63.
7. Nam Y. Running Shor's Algorithm on a complete, gate-by-gate implementation of a virtual, universal quantum computer. – 2012.
8. Stephane Beauregard: Circuit for Shor’s algorithm using 2n+3 qubits // Quantum Information and Computation, Vol. 3, No. 2 (2003) pp. 175-185
9. Draper T. G. Addition on a quantum computer //arXiv preprint quant-ph/0008033. – 2000.
10. Pavlidis A., Gizopoulos D. Fast Quantum Modular Exponentiation Architecture for Shor's Factorization Algorithm //arXiv preprint arXiv:1207.0511. – 2012.
11. Jones T. et al. Quest and high performance simulation of quantum computers //Scientific reports. – 2019. – Т. 9. – №. 1. – С. 1-11.