МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ П.М. МАШЕРОВА»

Факультет математики и информационных технологий

Кафедра информатики и информационных технологий

КУРСОВОЙ ПРОЕКТ  
по дисциплине «Программирование численных методов»

РАЗРАБОТКА И РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ САЙТА-МАГАЗИНА

Моисейченков Дмитрий Юрьевич

4 курс, 44 группа

Руководитель:

Шедько Василий Викторович

старший преподаватель

Витебск, 2019

**Реферат**

Курсовой проект 26 с., 5 рис., 6 источников.

PHP СЕРВЕР БАЗЫ ДАННЫХ ИНТЕРНЕТ-МАГАЗИН ВЗАИМОДЕЙСТВИЕ

Объект исследования – методы и технологии создания распределённых приложений.

Предмет исследования − реализация в программировании методов и технологий технологии создания распределённых приложений.

Цель проекта – ознакомиться с различными методами, технологиями по созданию распределённых приложений, сделать сравнительную характеристику данных методов и технологий, организовать практическое представление объекта исследования с помощью современных языков программирования с помощью языка программирования Python.

Методы исследования: анализ, синтез, изучение литературы, практическая реализация.

СОДЕРЖАНИЕ

[Введение 4](#_Toc7706407)

[1. Простые числа 6](#_Toc7706408)

[1.1 Определение простого числа 6](#_Toc7706409)

[1.2 Постановка проблемы определения простоты числа 6](#_Toc7706410)

[2. Алгоритмы определения простоты числа 8](#_Toc7706411)

[2.1 Перебор делителей 8](#_Toc7706412)

[2.2 Теорема Вильсона 9](#_Toc7706413)

[2.3 Тест Ферма 10](#_Toc7706414)

[2.4 Тест Миллера - Рабина 11](#_Toc7706415)

[2.5 Решето Эратосфена 13](#_Toc7706416)

[2.6 Решето Сундарама 15](#_Toc7706417)

[2.7 Решето Аткина 15](#_Toc7706418)

[2.8 Сравнение алгоритмов 16](#_Toc7706419)

[3. Практическая реализация алгоритма Диффи-Хеллмана в криптографии 17](#_Toc7706420)

[Заключение 25](#_Toc7706421)

[Список использованных источников 26](#_Toc7706422)

[ПРИЛОЖЕНИЕ А 27](#_Toc7706423)

# Введение

Выделение простых чисел является сложнейшей задачей для математиков. Ученые на протяжении многих веков пытаются найти закон, по которому можно определить n-ое простое число. Первый, кто занимался этой задачей, был великий математик древности, по совместительству ещё и географ, Эратосфен.

Первую известную всем нам таблицу простых чисел составил выдающийся итальянский математик Пьетро Антонио Катальди в 1603 г. Она охватывала все простые числа от 2 до 743. В 1750 г. Л. Эйлер установил, что число 231 - 1 является простым. Оно оставалось самым большим из известных простых чисел более ста лет [1].

В 1770 г. Немецкий математик Иоганн Генрих Ламберт опубликовал таблицу наименьших делителей всех чисел, не превосходящих 102000 и не делящихся на 2, 3, 5. Вложив в этот труд большие усилия, И. Г. Ламберт гарантировал бессмертие тому, кто доведет таблицу делителей до миллиона. На его призыв откликнулись многие вычислители.

К середине 19 века уже были составлены таблицы наименьших делителей не только первого миллиона, но и следующих, вплоть до 9.

В это же время в прессе появились сообщения, которые представлялись абсолютно фантастическими: в Венскую академию поступило 7 больших томов рукописных таблиц «Великий канон делителей всех чисел, которые не делятся на 2, 3 и 5, и простых чисел между ними до 100330201». Автором этого труда был Якуб Филипп Кулик, профессор высшей математики Пражского университета.

В дальнейшем поиски простых чисел уже не носили характера массовой охоты, с которой можно сравнить составление таблиц, а превратились в целенаправленный отбор отдельных представителей. У охотников за числами популярны простые числа французского ученого Марена Мерсенна – числа вида Mn = 2n -1, где n — натуральное число. Числа такого вида замечательны, в том числе тем, что некоторые из них являются простыми числами [2].

Некоторые представления о распределения простых чисел имели уже древние греки. Из доказательства Евклида следует, например, что они не собраны вместе, а разбросаны по всей числовой оси. В 1845 г. французский математик Жозеф Луи Франсуа Бертран, исследуя таблицу простых чисел в промежутке от 1 до 6000000, обнаружил, что между числами n и 2•n – 2, где n > 3, содержится одно простое число. Это свойство получило название постулата Ж. Л. Ф. Бертрана, хотя ему обосновать его так и не удалось. Доказал постулат в 1852 г. русский математик Пафнутий Львович Чебышев. Из результата П. Л. Чебышева следовала и более точная оценка.

Таким образом, даже среди очень больших чисел простые числа не так уж редки. С другой стороны, существуют промежутки, включающие тысячи, миллионы, миллиарды и вообще какое угодно большое количество подряд стоящих натуральных чисел, среди которых нельзя найти ни одного простого.

И только в 1770 г. немецкий математик Иоганн Генрих Ламберт опубликовал таблицу наименьших делителей всех чисел, не превосходящих 102000 и не делящихся на 2, 3, 5. Также первым значительным успехом в определении простоты целых чисел явились теорема Вильсона и теорема Ферма, которые являются центральными теоремами в данной проблеме и на сегодняшний день.

Почему же так важно знать простое число или нет? Поиск больших простых чисел имеет, действительно, важное значение для математики и не только. Например, в криптографии большие простые числа используются в алгоритмах шифрования с открытым ключом.

В данном курсовом проекте будут представлены основные алгоритмы проверки чисел на простоту. Цель данного курсового проекта - познакомиться с основными алгоритмами проверки чисел на простоту и реализовать на языке программирования Python некоторые из них.

# 1. Простые числа

## 1.1 Определение простого числа

Простые числа – это натуральные числа, большие единицы, которые имеют только два делителя: единицу и само это число.

Примеры простых чисел: 2 , 3, 5, 7, 11, 13…

(Единица не является простым числом!)

Существует множество задач, связанных с простыми числами, и хотя формулируются они достаточно просто, решить их бывает очень трудно. Некоторые свойства простых чисел еще не открыты. Это побудило немецкого математика Германа Вейля (Wayl, 1885-1955) так охарактеризовать простые числа: «Простые числа – это такие существа, которые всегда склонны прятаться от исследователя».

Во все времена люди хотели найти как можно большее простое число. Пока люди считали только при помощи карандаша и бумаги, им нечасто удавалось обнаружить новые простые числа. До 1952 г. самое большое известное простое число состояло из 39 цифр. Теперь поиском все больших простых чисел занимаются компьютеры. Это может представлять интерес для любителей рекордов.

Не будем гнаться за рекордами, а рассмотрим несколько алгоритмов нахождения простых чисел [3].

## 1.2 Постановка проблемы определения простоты числа

Простые числа находят широкое применение в современном мире, например, в криптографии. Несмотря на то что простые числа изучаются на протяжении длительного времени, проблема поиска простых чисел в наше время наиболее актуальна, в связи с потребностью получать большие простые числа для криптографических алгоритмов.

Сегодня существует огромное количество методов по определению простоты числа. Однако в каждом из них есть свои положительные моменты, но также и недостатки, что более существенно. Именно поэтому данный курсовой проект нацелен на вывод сравнительной характеристики всевозможных методов по определению простоты больших чисел.

Сегодня можно выделить два вида методов по определению простоты больших чисел:

* Классические
* Современные

В данном курсовом проекте произойдёт представление и будет показан пример работы как классических, так и современных методов по определению простоты больших чисел.

# 2. Алгоритмы определения простоты числа

Существует огромное количество алгоритмов, созданных или придуманных именно для того, чтобы качественно определить простоту числа. Однако в каждом алгоритме есть свои плюсы и минусы. Поэтому следует внимательно разобраться в работе каждого алгоритма, выделить все недостатки алгоритма, попробовать предположить, что можно добавить в алгоритм, чтобы определение простоты больших чисел происходило намного быстрее.

Этапы определения простоты больших чисел:

1. Происходит ввод числа;

2. Происходит выполнения алгоритма;

3. Происходит выдача ответа о простоте.

## 2.1 Перебор делителей

Перебор делителей — алгоритм [факторизации](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) или [тестирования простоты](http://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D1%82%D1%8B) числа путем [полного перебора](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80) всех возможных потенциальных [делителей](http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C).

Обычно перебор делителей заключается в переборе всех [целых](http://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) (как вариант: [простых](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE)) чисел от 2 до [квадратного корня](http://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D1%80%D0%B5%D0%BD%D1%8C) из факторизуемого числа **n** и в вычислении [остатка от деления](http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%BE%D0%BC) исходного **n** на каждое из этих чисел. Если остаток от деления на некоторое число **m** равен нулю, то **m** является делителем числа **n**. В этом случае либо n объявляется составным, и алгоритм заканчивает работу (если тестируется простота n), либо n сокращается на m и процедура повторяется (если осуществляется факторизация n). По достижении квадратного корня из n и невозможности сократить n ни на одно из меньших чисел, n объявляется простым.  
Примечание: если у n есть некоторый делитель p, то n/p так же будет делителем, причем одно из этих чисел не превосходит

Реализация данного алгоритма на языке Python представлен в листинге 2.1.1

Листинг 2.1.1

def divider(n):

i = 2

j = 0 # флаг

while i\*\*2 <= n and j != 1:

if n%i == 0:

j = 1

i += 1

if j == 1:

print ("Это составное число")

else:

print ("Это простое число")

## 2.2 Теорема Вильсона

Возможно, что самым знаменитым условием простоты числа является теорема Вильсона:

Число n является делителем числа (n – 1)! + 1 тогда и только тогда, когда n – простое число.

Теорема Вильсона в значительной мере имеет теоретическое значение, поскольку довольно не просто вычислить (n–1)!. Проще найти an–1, поэтому элементарные тесты, определяющие является ли число простым, основаны не на теореме Вильсона, а на малой теореме Ферма:

Если **n** – простое число и **a** – целое число, не делящееся на**n** , то an–1– 1 делится на **n** (другими словами an–1при делении нацело на **n** даёт в остатке 1).

Например, наибольшее простое число, найденное используя теорему Вильсона, – 1099511628401, и даже с умным подходом к расчету n!, потребуется около суток вычислений на процессорах SPARC. Но числа с десятками тысяч цифр, проходят тест на простоту с использованием теоремы Ферма, меньше чем за час.

Однако, в отличие от малой теоремы Ферма, теорема Вильсона является одновременно необходимым и достаточным условием простоты числа.

Теоремы Ферма и Вильсона можно соединить в одну следующую теорему:

Если n – простое число, то для любого целого числа а число an+ a · (n – 1)! делится на n.

Теорема Вильсона впервые была сформулирована  в 1770 году известным английским математиком XVIII века, членом Лондонского королевского общества, доктором медицины и профессором математики Кембриджского университета Эдвардом Варингом.

Реализацию данной теоремы можно увидеть в листинге 2.2.1

Листинг 2.2.1

def prime(n):

if (math.factorial(n-1)+1) % n!=0:

print ("Это составное число")

else:

print ("Это простое число")

## 2.3 Тест Ферма

Первый вероятностный метод, который мы обсуждаем, — испытание простоты чисел тестом Ферма [4].

Если n — простое число, то an-1http://www.intuit.ru/img/symbols/equiv.gif 1 mod n.

Следует заметить, что если n — простое число, то сравнение справедливо. Это не означает, что если сравнение справедливо, то n — простое число. Целое число может быть простым числом или составным объектом. Мы можем определить следующие положения как тест Ферма:

Если n — простое число, то an-1 http://www.intuit.ru/img/symbols/equiv.gif 1 mod n

Если n — составной объект, то возможно, что an-1 http://www.intuit.ru/img/symbols/equiv.gif 1 mod n

Подводя итог сказанному, реализуем алгоритм теста Ферма в листинге 2.3.1.

Листинг 2.3.1  
def primFerma(a,n):

if a\*\*(n-1)%n==1:

print "правдоподобно простое"

else:

print "составное"

Если в результате работы теста мы получим сообщение: «правдоподобно простое», то можно заключить: **n** — составное с вероятностью ≤ существуют составные числа, относительно которых тест Ферма выдаст ответ правдоподобно простое, для всех взаимно простых с **n** оснований **а**. Такие числа называются [числами Кармайкла](http://ru.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE). К сожалению, их бесконечное количество. Первые из них — это 561, 1105 и 1729. Числа Кармайкла обладают следующими свойствами:

* они нечетны,
* имеют по крайней мере три простых делителя,
* свободны от квадратов1,
* если р делит число Кармайкла N, то р — 1 делит N — 1.

Следовательно, они встречаются не очень часто, но и не настолько редко, чтобы их можно было игнорировать.

## 2.4 Тест Миллера - Рабина

[Вероятностный полиномиальный](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_BPP) [тест простоты](http://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D1%82%D1%8B). Тест Миллера — Рабина позволяет эффективно определять, является ли данное число составным. Однако, с его помощью нельзя строго доказать простоту числа. Тем не менее, тест Миллера — Рабина часто используется и в [криптографии](http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) для получения больших [случайных простых чисел](http://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE). Данный алгоритм был разработан [Гари Миллером](http://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B8%D0%BB%D0%BB%D0%B5%D1%80,_%D0%93%D0%B0%D1%80%D0%B8&action=edit&redlink=1) в [1976 году](http://ru.wikipedia.org/wiki/1976_%D0%B3%D0%BE%D0%B4), а затем модифицирован [Майклом Рабином](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B1%D0%B8%D0%BD,_%D0%9C%D0%B0%D0%B9%D0%BA%D0%BB_%D0%9E%D0%B7%D0%B5%D1%80) в [1980 году](http://ru.wikipedia.org/wiki/1980_%D0%B3%D0%BE%D0%B4).

Пусть m — нечётное число большее 1. Число m-1  однозначно представляется в виде , где t нечётно. Целое число a, 1 < a < m, называется свидетелем простоты числа m, если выполняется одно из условий: или существует целое число k, , такое, что.

 Теорема Рабина утверждает, что составное нечётное число m имеет не более  различных свидетелей простоты, где  — [функция Эйлера](http://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0).  
Алгоритм Миллера — Рабина параметризируется количеством раундов **r**. Рекомендуется брать **r** порядка величины log2(m), где **m** — проверяемое число.

Для данного числа m находится такое целое число s и целое нечётное число t, что m − 1 = 2st. Выбирается случайное число a, 1 < a < m. Если a не является свидетелем простоты числа m, то выдается ответ «m составное», и алгоритм завершается. Иначе, выбирается новое случайное число a и процедура проверки повторяется. После нахождения r свидетелей простоты, выдается ответ «m, вероятно, простое», и алгоритм завершается. Как и для теста Ферма, все числа n > 1, которые не проходят этот тест – составные, а числа, которые проходят, могут быть простыми. И, что важно, для этого теста нет аналогов чисел Кармайкла.

В 1980 году было доказано, что вероятность ошибки теста Рабина-Миллера не превышает 1/4. Таким образом, применяя тест Рабина-Миллера t раз для разных оснований, мы получаем вероятность ошибки 2-2t [5].

Реализация данного метода представлена в листинге 3.4.1.

Листинг 3.4.1  
 def toBinary(n):

r = []

while (n > 0):

r.append(n % 2)

n = n / 2

return r

def MillerRabin(n, s = 50):

for j in xrange(1, s + 1):

a = random.randint(1, n - 1)

b = toBinary(n - 1)

d = 1

for i in xrange(len(b) - 1, -1, -1):

x = d

d = (d \* d) % n

if d == 1 and x != 1 and x != n - 1:

return True # Составное

if b[i] == 1:

d = (d \* a) % n

if d != 1:

return True # Составное

return False # Простое

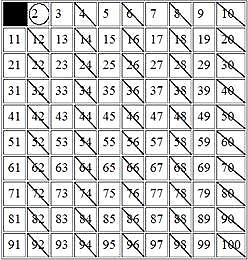
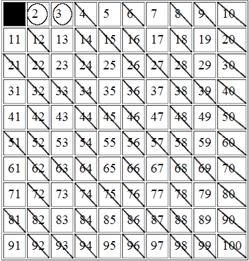
## 2.5 Решето Эратосфена

Вычислительная сложность алгоритма:

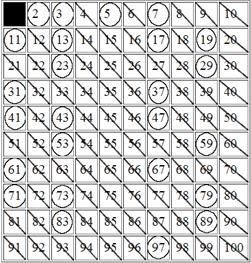
Пространственная сложность алгоритма:

Описание: Детерминированный алгоритм поиска простых чисел на отрезке от единицы до заданного целого числа **n**.

Алгоритм: Выпишем последовательно все целые числа из отрезка [2,N], где N — заданное число. Возьмем первое число x1 из этого списка и вычеркнем все числа кратные (2x1, 3x1, 4x1...) ему. Найдем первое не зачеркнутое число x2 в списке, большее чем x1 и повторим операцию вычеркивания кратных чисел. Будем продолжать эти шаги до тех пор, пока это возможно. Таким образом, все оставшиеся (не зачеркнутые) числа также будут являться простыми.

  
**Рисунок 2.5.1 – Описание работы алгоритма. Исключение чисел кратных 2**  


**Рисунок 2.5.2 Описание работы алгоритма. Исключение чисел кратных 2 и 3.**



**Рисунок 2.5.3 Результат работы алгоритма**.

## 2.6 Решето Сундарама

Вычислительная сложность алгоритма:

Пространственная сложность алгоритма:

Описание: Детерминированный алгоритм поиска простых чисел на отрезке от единицы до 2N+1, где N - заданное число. Основная идея алгоритма заключается в вычеркивании всех составных чисел из списка, таким образом, в нем останутся только простые числа.

Алгоритм: Выпишем последовательно все целые числа из отрезка [2,N]. Вычеркнем из списка все целые числа вида i+j+2ij<=N, где i, j – всевозможные комбинации целых чисел от 1 до N, причем i<=j. Каждое оставшееся (не зачеркнутое число) умножается на 2 и увеличивается на 1. Таким образом будет получена последовательность простых чисел на отрезке [1, 2N+1].

## 2.7 Решето Аткина

Вычислительная сложность алгоритма:

Пространственная сложность алгоритма:

Описание: Алгоритм поиска простых чисел на отрезке от единицы до заданного числа n. Алгоритм основан на следующих теоремах:

Теорема 1: Пусть n – натуральное число, свободное от квадратов (т.е. не делящееся ни на какой квадрат простого числа) и которое можно представить в виде 1+4**Z**. Тогда, n – просто тогда и только тогда, когда выражение  – нечетно.

Теорема 2: Пусть n–натуральное число, свободное от квадратов (т.е. не делящееся ни на какой квадрат простого числа) и которое можно представить в виде 1+6**Z**. Тогда, n – просто тогда и только тогда, когда выражение  – нечетно.

Теорема 3: Пусть N – натуральное число, свободное от квадратов (т.е. не делящееся ни на какой квадрат простого числа) и которое можно представить в виде 11+12**Z**. Тогда, n – просто тогда и только тогда, когда выражение   – нечетно.

Алгоритм: Выпишем последовательно все целые числа из отрезка [2,N], где N – заданное число. Для каждой пары (x,y):  вычислим значения уравнений  (по теореме 1), (по теореме 2) и если x > y: (по теореме 3). Если полученное значение нечетно и его можно представить в виде, то число либо простое, либо кратно квадрату простого числа. Таким образом, чтобы убедиться, что число простое в заключительном этапе необходимо вычеркнуть числа кратные их квадратам.

## https://sibac.info/files/2016_08_22_studmeghdis/Shveikin.files/image017.jpg2.8 Сравнение алгоритмов

**Рисунок 2.8.1 Сравнительный график трёх теорем**

Был проведен сравнительный анализ различных алгоритмов поиска простых чисел, по результатам которого можно построить график. Легко заметить, что среди тестов, которые подверглись проверке на быстродействие, наиболее эффективным оказалось решето Аткина.

# 3. Практическая реализация алгоритма Диффи-Хеллмана в криптографии

Протокол Ди́ффи — Хе́ллмана ([англ.](https://wiki2.org/ru/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Diffie–Hellman*, *DH*) — [криптографический протокол](https://wiki2.org/ru/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB), позволяющий двум и более сторонам получить общий секретный [ключ](https://wiki2.org/ru/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов [симметричного шифрования](https://wiki2.org/ru/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [6].

Схема открытого распределения ключей, предложенная Диффи и Хеллманом, произвела настоящую революцию в мире шифрования, так как снимала основную проблему классической криптографии — проблему распределения ключей.

В чистом виде алгоритм Диффи — Хеллмана уязвим для модификации данных в канале связи, в том числе для атаки «[Man-in-the-middle (человек посередине)](https://wiki2.org/ru/%D0%A7%D0%B5%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA_%D0%BF%D0%BE%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5" \o "Человек посередине)», поэтому схемы с его использованием применяют дополнительные методы односторонней или двусторонней аутентификации.

Передача ключа по открытым каналам была большой проблемой криптографии XX века. Но эту проблему удалось решить после появления алгоритма Диффи — Хеллмана. Данный алгоритм позволил дать ответ на главный вопрос: «Как при обмене зашифрованными посланиями уйти от необходимости передачи секретного кода расшифровки, который, как правило, не меньше самого послания?» Открытое распространение ключей Диффи — Хеллмана позволяет паре пользователей системы выработать общий секретный ключ, не обмениваясь секретными данными.

Предположим, существует два абонента: Алиса и Боб. Обоим абонентам известны некоторые два числа **g** и **p**, которые не являются секретными и могут быть известны также другим заинтересованным лицам. Для того, чтобы создать неизвестный более никому секретный ключ, оба абонента генерируют большие случайные числа: Алиса — число **a**, Боб — число **b**. Затем Алиса вычисляет [остаток от деления](https://wiki2.org/ru/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%BE%D0%BC) (1):

 (1)

и пересылает его Бобу, а Боб вычисляет [остаток от деления](https://wiki2.org/ru/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%BE%D0%BC) (2):

  (2)

и передаёт затем Алисе. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (то есть, у него нет возможности вмешаться в процесс передачи).

На втором этапе Алиса на основе имеющегося у неё **a** и полученного по сети **B** вычисляет значение (3):

  (3)

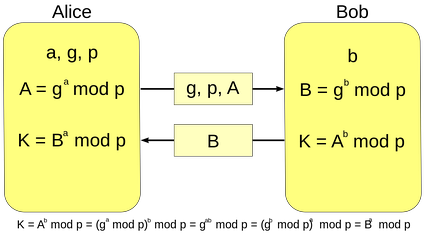
Боб на основе имеющегося у него **b** и полученного по сети **A** вычисляет значение (4):

   (4)

Как нетрудно заметить, у Алисы и Боба получилось одно и то же число (5):

   (5)

Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник встретится с практически неразрешимой (за разумное время) проблемой вычисления (3) или (4) по перехваченным  и , если числа **p, a, b** выбраны достаточно большими. Наглядная работа алгоритма показана на рисунке 3.1.

[](https://wiki2.org/ru/%D0%A4%D0%B0%D0%B9%D0%BB:Diffie-Hellman-Schl%C3%BCsselaustausch_svg)

**Рисунок 3.1 Алгоритм Диффи-Хеллмана**

При работе алгоритма каждая сторона:

1. Генерирует случайное [натуральное число](https://wiki2.org/ru/%D0%9D%D0%B0%D1%82%D1%83%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) a — закрытый ключ
2. Совместно с удалённой стороной устанавливает открытые параметры p и g (обычно значения p и g генерируются на одной стороне и передаются другой), где p является [случайным простым числом](https://wiki2.org/ru/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) (p-1)/2 также должно быть [случайным простым числом](https://wiki2.org/ru/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) (для повышения безопасности) g является [первообразным корнем](https://wiki2.org/ru/%D0%9F%D0%B5%D1%80%D0%B2%D0%BE%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D1%80%D0%B5%D0%BD%D1%8C_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%87%D0%B8%D1%81%D0%B5%D0%BB)) [по модулю](https://wiki2.org/ru/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E) p (также является простым числом)
3. Вычисляет открытый ключ A, используя преобразование над закрытым ключом A = ga mod p
4. Обменивается открытыми ключами с удалённой стороной
5. Вычисляет общий секретный ключ K, используя открытый ключ удаленной стороны B и свой закрытый ключ a

K = Ba mod p

К получается равным с обеих сторон, потому что:

Ba mod p = (gb mod p)a mod p = **gab mod p** = (ga mod p)b mod p = Ab mod p

В практических реализациях для a и b используются числа порядка 10100 и p порядка 10300. Число g не обязано быть большим и обычно имеет значение в пределах первого десятка.

Реализация данного алгоритма на языке программирования Python представлена в листинге 3.1.

Листинг 3.1

# -\*- coding: utf-8 -\*-

# This program

#will be used

#to demonstrate

#basic Diffie-Hellman

#Key Exchange

import numpy as np

import pandas as pd

import random

from fractions import gcd

import math

def soe(n):

store = []

prime = []

z = n

r = range(z+1)[2:len(range(z+1))]

r = r[1::2]

for i in range(len(r)):

for j in range(len(r)):

prime.append(r[i] % r[j])

chunks = [prime[x:x+len(r)] for x in xrange(0, len(prime), len(r))]

for i in range(len(chunks)):

if chunks[i].count(0) == 1:

store.append(r[i])

store.insert(0,2)

return store

#primes = soe(20000)

def first\_primitive\_root(p):

primes = soe(int(round(math.sqrt(p))))

tot = p-1

divisor = []

more = []

for i in primes:

if tot % i == 0:

divisor.append(i)

for i in divisor:

more.append(tot/i)

prim\_check = []

for i in range(2,p):

prim\_check = []

for j in more:

prim\_check.append((i\*\*j % p))

if any(z == 1 for z in prim\_check):

continue

else:

break

return i

def is\_prime(n):

return all([(n%j) for j in range(2, int(n\*\*0.5)+1)]) and n>1

def all\_roots(p):

if is\_prime(p) == True:

roots = [first\_primitive\_root(p)]

a = first\_primitive\_root(p)

for i in range(2,p):

if gcd(i,p-1) == 1:

roots.append(a \*\* i % p)

return roots

else:

return "Use a prime please"

def invmodp(a, p):

for d in xrange(1, p):

r = (d \* a) % p

if r == 1:

break

else:

raise ValueError('%d has no inverse mod %d' % (a, p))

return d

any\_in = lambda a, b: any(i in b for i in a)

def baby\_step\_giant\_step(h,g,p):

m = int(math.ceil(math.sqrt(p)))

group = []

for i in range(m+1):

group.append(pow(g,i) % p)

z = pow(invmodp(g,p),m) % p

store = []

more = 0

k = 0

q = m

for i in range(0,q):

if any\_in(group,store) == True:

more += i-1

break

else:

store.append(h\*pow(z,i) % p)

for j in range(len(group)):

if group[j]== store[more]:

k = k+j

final\_number = more\*m+k

return final\_number

return more,group,store,k,final\_number,z

# return m

## return final\_number #z,store,more,jj,final\_number

# return more

#

def DH(p,g,a,b):

# p = 761

# g = 6

## Alice ##

# a = 12

p = p

A = pow(g,a) % p

## Bob ##

# b = 15

B = pow(g,b) % p

## Alice

s\_a = pow(B,a) % p

## Bob

s\_b = pow(A,b) % p

return {'secret':"The common secret is " + str(s\_a), 'A':A,'B':B ,'p':p,'g':g}

# print s\_a

# print s\_b

D\_H = DH(13,6,4,8)

for i in range(1,4):

#print pow(D\_H['g'],i) % 13

if pow(D\_H['g'],i) % 13 == D\_H['A']:

print "Your secret is not secret, it's " + str(D\_H['B']\*\*i % 13)

break

else: print "I don't know your secret"

print D\_H['secret']

#D\_H = DH(23,5,7,28)

#D\_H1 = DH(23,5,7,6)

#print "Eve has got your secret " + str(baby\_step\_giant\_step(A,g,p))

# Заключение

В результате выполнения курсового проекта были получены навыки работы и поиска учебной литературы, улучшены навыки изучения и анализа различных источников, были подробно описаны все этапы работы и использовании представленных алгоритмов.

Также можно сделать вывод, что скорость работы современных методов позволяет быстро находить результат посредством программирования, однако появляется проблема тестирования программы, которая остаётся весь период эксплуатации. Для классических методов характерны простые программные решения, но программные тесты значительно сложнее самой программы.

Сегодня современный технологии позволяют хранить очень большой объем информации, например сетевая база данных простых чисел. В ней можно легко отыскать или не отыскать нужное нам число, а также легко пополнить и оставить в памяти навсегда. Также, если ещё попытаться задействовать нейронную сеть, то результат выполнения поиска будет увеличен в разы. Тем самым, решение проблемы тестирования будет решено единожды и навсегда.

Подводя итоги, можно констатировать, что в данном проекте были изучены различные алгоритмы поиска простых чисел и осуществлён их сравнительный анализ. Проведённые исследования показывают, что решето Аткина является наиболее эффективным из представленных алгоритмов.

Также посредством языка программирования Python была реализована программа по определению простого числа в криптографии, в основе которой лежал алгоритм Диффи-Хеллмана. После тестирования, можно сделать вывод, что алгоритм работает правильно.

Следовательно, можно сделать вывод, что поставленная нами цель в начале курсового проекта была достигнута.

# Список использованных источников

1. Тест простоты − Википедия [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Тест_простоты> − Дата доступа: 29.04.2019

2. Кормен, Т. Проверка чисел на простоту/ Т. Кормен, Ч. Лейзер. − М.: МЦНМО, 2002.− 772 с.

3. Шнирельман, Л.Г.  Простые числа/ Л.Г. Шнирельман, − М.: 1940.− 60 с.

4. Додонова, Н.Л.  Конспект лекций по дисциплине алгебраические структуры и теория чисел/ Н.Л. Додонова, − Самара: 2016.− 140 с.

5. Шнайер, Б.М. Прикладная криптография/ Б.М.Шнайер - ТРИУМФ 2002. – 816 с.

6. Протокол Диффи-Хеллмана − Википедия [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Протокол_Диффи-Хеллмана> − Дата доступа: 29.04.2019

# ПРИЛОЖЕНИЕ А

**Содержание электронного носителя**

На электронном носителе расположены следующие директории и файлы:

− Исходный код программы

* Файл «Курсовой проект «Современные методы определения простоты больших чисел» с документацией к курсовому проекту

− Презентация « Определение простоты больших чисел Моисейченков Д.Ю.»