МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

*ИНСТИТУТ Информационных технологий и компьютерных наук*

*КАФЕДРА Инженерной кибернетики*

**Лабораторная работа №3 Распределенный поиск**

**по дисциплине  
 «Глобальные сети»**

*Студент Голованов Н.С.*

*Группа БПМ-17-1*

*Принял с оценкой*

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_2021 г.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(подпись) (Фамилия И.О.)

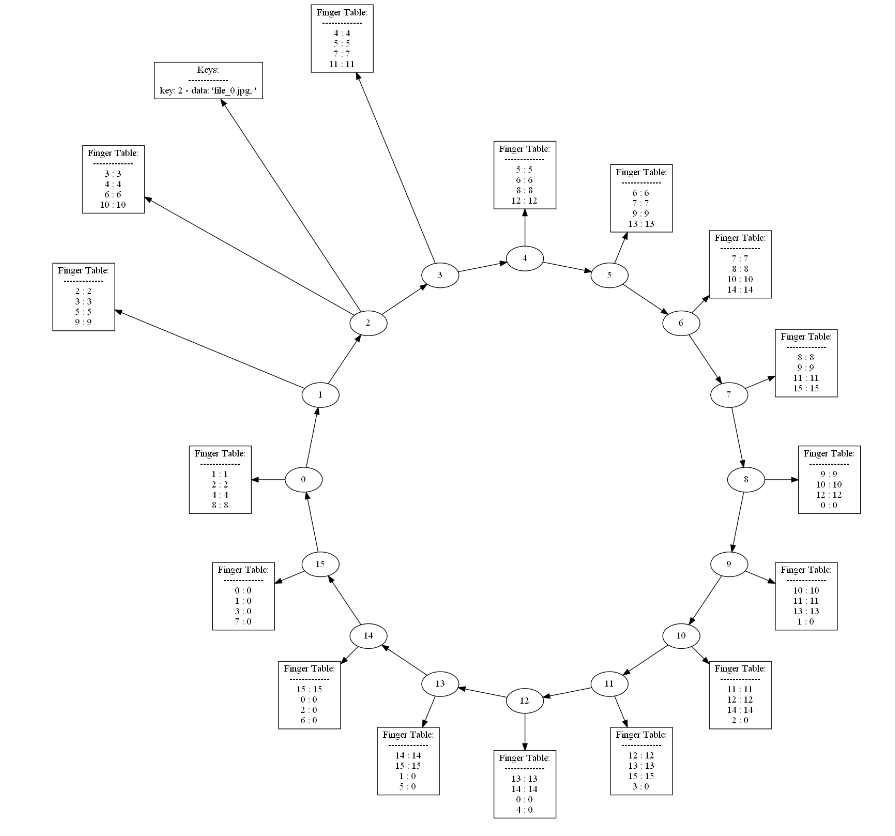
Москва 2021

Описание реализованного алгоритма поиска.

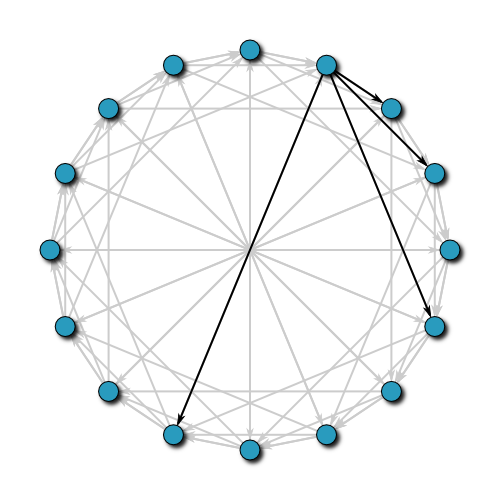
**Общая информация**

Распределенная хеш-таблица (DHT) — это класс децентрализованной распределенной системы, которая предоставляет службу поиска, аналогичную хеш-таблице: пары (ключ, значение) хранятся в DHT, и любой участвующий узел может эффективно извлекать значение, связанное с данным ключом. Ответственность за поддержание сопоставления от ключей к значениям распределяется между узлами таким образом, что изменение набора участников вызывает минимальные нарушения. Это позволяет DHT масштабироваться до чрезвычайно большого количества узлов и обрабатывать постоянные приходы, уходы и отказы узлов.

В данной работе был реализован алгоритм поиска в децентрализованной пиринговой сети **Chord**. Отличительная особенность данного алгоритма в том, что каждый узел хранит таблицу пальцев, для того чтобы избежать линейного поиска по кольцу.



*Рисунок 1. Chord-NET с визуализации “пальцев” в виде таблиц.*



*Рисунок 2. Визуализация с “пальцами” (подсвечено черным для одного узла)*

Максимальная вместимость сети с алгоритмом поиска Chord задается числом **2^m**. Таким образом мы получаем кольцо от 0 до 2^m-1. Число m должно быть достаточно большим, чтобы избежать коллизии. В идеальном случае кольцо должно быть полностью заполнено, но такое бывает далеко не часто. Поэтому некоторая информация ввиду отсутствия подходящего ключа по hash-коду будет лежать в *successor* или узле-предке.

**Формирование таблиц пальцев**

Таблица пальцев является изюминкой данного алгоритма. Она позволяет ускорить процесс поиска. Алгоритм ее формирования приведен ниже.

i-ая запись узла n будет содержать преемника ((n + 2 ^ {i-1}) mod 2 ^ m). Первая запись в таблице “пальцев” фактически является непосредственным **преемником узла**. Каждый раз, когда узел хочет найти ключ k, он будет передавать запрос ближайшему преемнику k в своей таблице “пальцев” («самый большой» в круге, идентификатор которого меньше, чем k), пока узел не узнает, что ключ хранится в его непосредственном преемнике.

**Распределение файлов**

Вся информация, необходимая для хранения и быстрого поиска, хешируется и располагается в специально отведенным ему узле. В моем случае я беру имя файла, хеширую, переводим в hex и из нее получаем последовательность байт. И получаем целочисленный ключ (отвечающий за номер узла, где будет расположен данный файл) по следующем алгоритму:

num\_bits = m #m заданный параметр сети

bt = hashlib.sha1(str.encode(filename)).digest()

# число нужного количество байт для id

req\_bytes = (num\_bits + 7) // 8

# Получаем ключ в виде int

hashed\_id = int.from\_bytes(bt[:req\_bytes], 'big')

if num\_bits % 8:

hashed\_id >>= 8 - num\_bits % 8

Таким образом, мы записываем файл в соответствующий ему по полученному хеш-коду узел. Для уникальности мы записываем полностью hash-значение, но могут возникнуть проблемы, если у нас одинаковые называния файлов. Тогда уже необходимо еще и хешировать по содержанию файла. В данной работе это отмечается, но не решается.

**Алгоритм поиска**

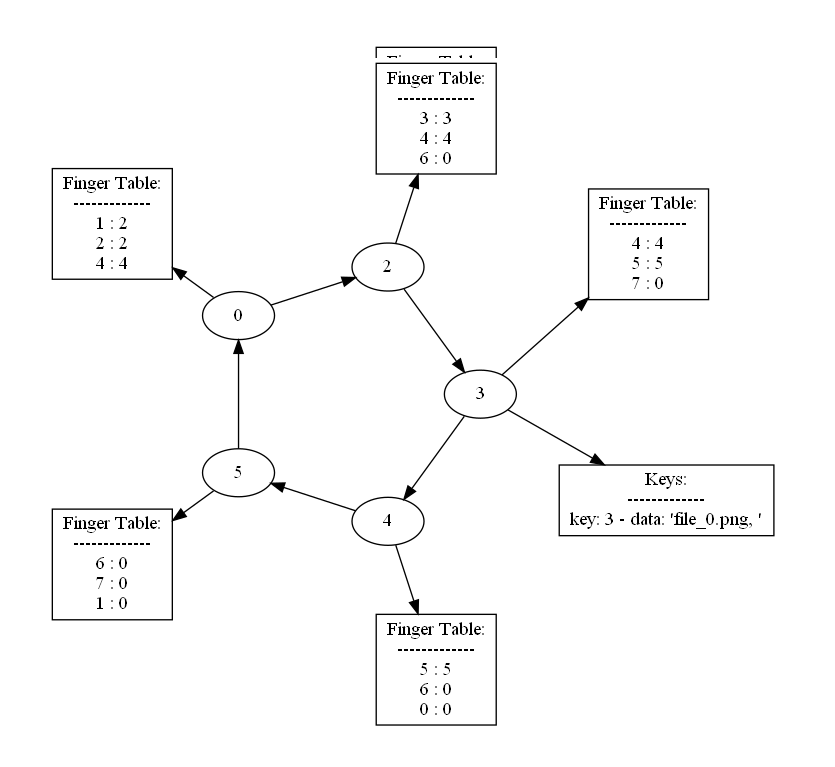
После того, как мы создали сеть, сделали записи в таблицах пальцев и распределили файлы внутри сети, можно искать файлы достаточно быстро. На вход функции **find\_successor** мы подаем ключ, который получили в результате хеширования имени файла и преобразованный специальным образом (см. распределение файлов).

Сначала смотрим, совпадает ли ключ данных с текущем узлом? Если да, то сразу проверяем наличие нужного нам файла по полному ключу в этому узле. Если ключи не совпали, мы подключаем функцию дистанции (file\_key(n2) -node\_key(n1) if n1 <= n2 else m - n1 + n2). Т.е. мы расстояние до ключа файла меньше от текущего узла, чем от нашего преемника тоже до ключа, значит файл находится у приемника. Иначе, определяем ближайшего соседа из числа преемников в таблице “пальцев” и повторяем алгоритм с этим соседом, до тех пор, пока мы не найдем нужный ключ.

Графовое представление

Алгоритмом заложено, что у нас связанный ориентированный граф (Рисунок 2). Степень вершины фиксирована, если у нас совпадает значение количества узлов с максимально возможным. Тогда степень равна числу m. Если у нас есть пропуски например пропущены узлы 4, 5 и 6, тогда значения ключей для узлов 4, 5 и 6 будут лежать в предке 3-его, а именно в 7-мом узле. Что означает, что таблица пальцев для 3-его будет следующая: 4:7 5:7 7:7 (для m = 3). Вот еще один пример (Рисунок 3). Для вершины 4 выполняется правило степень = m. Так как узел 4 не принимает на себя другие ключи и отвечает только за хеш ключ 4. В то время как 0 узел помимо ключа 0 берет на себя берет ключи 6 и 7.

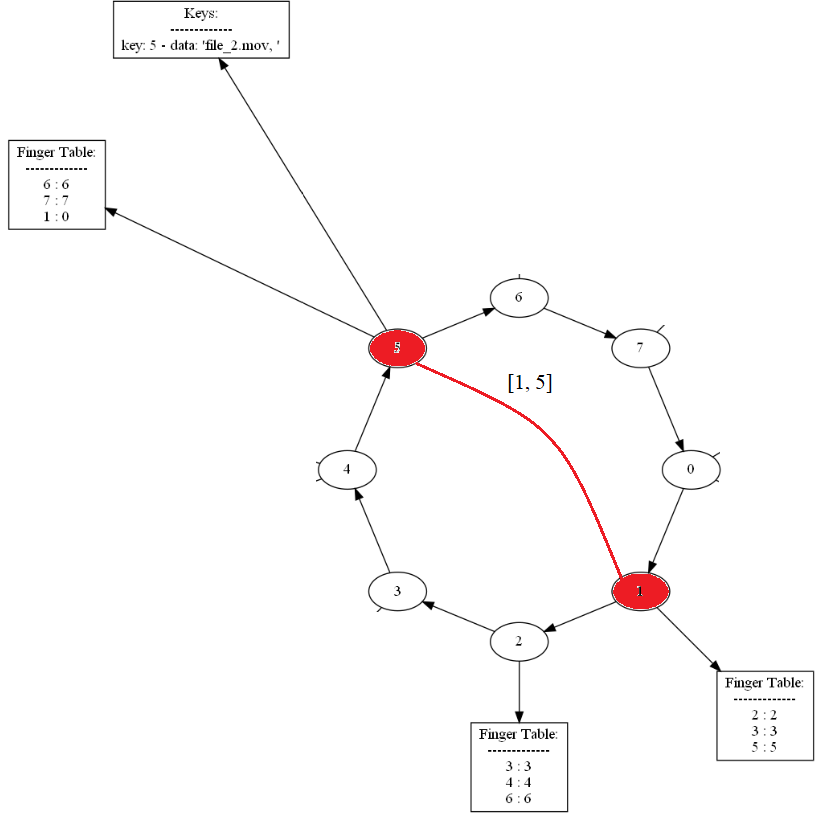
Визуализация построена с помощью программы Graphviz.



*Рисунок 3. Пример неравномерности степеней вершин.*

Результаты работы

Визуализация для 5 шагов (расстояние по кольцу) до файла **file\_2.mov**. Сразу переход в 5 узел, где лежит файл, за **два хопа**. Первое значение массива – откуда стартуем.



*Рисунок 4. Визуализация 2 хопов.*

Вывод:

---поиск файла за: 0.0 сек ---

>>> Поиск 'file\_2.mov' по короткому ключу 5

>>> Найден файл 'file\_2.mov' в узле 5 по короткому ключу 5 и длинному 1063458784768643030156490012575531990011123169140

Путь - [1, 5]

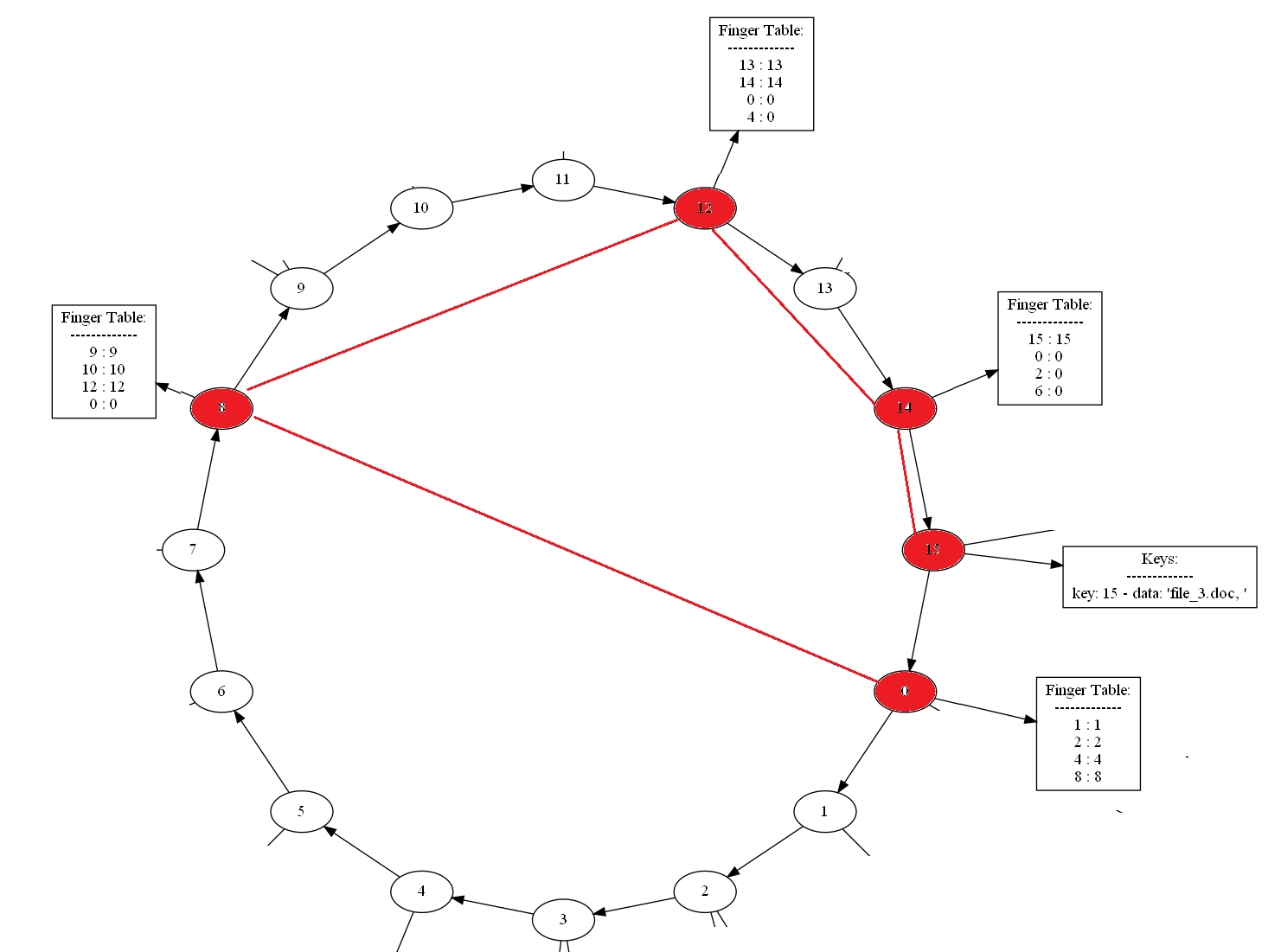
**Для 5-ти хопов**.

---поиск файла за: 0.0 сек ---

>>> Поиск 'file\_3.doc' по короткому ключу 15

>>> Найден файл 'file\_3.doc' в узле 15 по короткому ключу 15 и длинному 1400268798265846286555301240757088200043282823104

Путь - [0, 8, 12, 14, 15]



*Рисунок 5. Визуализация 10 хопов.*

**Для 10 хопов** и больше уже сложно визуализировать. Так как, как правило, количество хопов редко превышает m.

**Результаты для фиксированного графа (1000 вершин m = 10). 5000 файлов.**

Всего файлов было сгенерировано: 5000

Минимальное количество файлов в узле: 0

Максимальное количество файлов в узле: **14**

Среднее количество файлов в узле: **5.0**

Количество затраченных временных ресурсов, чтобы разнести все файлы в правильные узлы (в секундах): 0.0003

--------------------------------------------------------------------------------

Минимальное количество hops: 0

Максимальное количество hops: **10**

Среднее количество hops: **5.1576**

Количество затраченных временных ресурсов, чтобы найти все файлы (в секундах): **4.4916**

**Результат для фиксированного графа (4000 вершин m = 12). 8000 файлов**

Всего файлов было сгенерировано: 8000

Минимальное количество файлов в узле: 0

Максимальное количество файлов в узле: 10

Среднее количество файлов в узле: 2.0

Количество затраченных временных ресурсов, чтобы разнести все файлы в правильные узлы (в секундах): 0.0003

--------------------------------------------------------------------------------

Минимальное количество hops: **1**

Максимальное количество hops: **12**

Среднее количество hops: **6.269**

Количество затраченных временных ресурсов, чтобы найти все файлы (в секундах): **7.2967**

**Результаты для фиксированного графа (250 вершин m = 8). 700 файлов.**

Всего файлов было сгенерировано: 700

Минимальное количество файлов в узле: 0

Максимальное количество файлов в узле: **8**

Среднее количество файлов в узле: **2.8**

Количество затраченных временных ресурсов, чтобы разнести все файлы в правильные узлы (в секундах): 0.0002

--------------------------------------------------------------------------------

Минимальное количество hops: 0

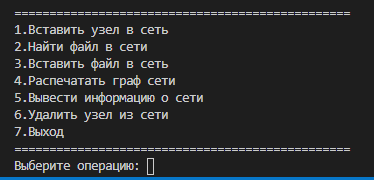
Максимальное количество hops: **8**

Среднее количество hops: **3.9629**

Количество затраченных временных ресурсов, чтобы найти все файлы (в секундах): **0.5298**

Дополнительная работа. Скриншоты.

Для удобства была разработана меню с командами, для более удобного тестирования.



*Рисунок 6. Меню*

Также для удобства по поиску высвечивается путь при поиске.

---поиск файла за: 0.0009996891021728516 сек ---

>>> Поиск 'file\_699.py' по короткому ключу 195

>>> Найден файл 'file\_699.py' в узле 195 по короткому ключу 195 и длинному 1116755970891120144935039863898825333224477188403

[77, 141, 173, 189, 193, 195]

Если файл отсутствует, то это также обрабатывается программой.

>>> Введите имя файла, который хотите найти: пау

>>> Поиск 'пау' по короткому ключу 81

>>> 'пау' нет в данной сети

Путь до файла - [77, 81]

---Поиск файла за: 0.0009996891021728516 сек ---

Github: