**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

**Алгоритмы и структуры данных**

**КДЗ 1**

Выполнил: Чуев Иван,

студент гр. БПИ163.

Преподаватель:

Мицюк Алексей Александрович

Москва 2018

**Постановка задачи**

Разработать с использованием языка C++ программу, реализующую алгоритмы сжатия данных без потерь, для упаковки и распаковки файлов различного типа:

1. алгоритм Хаффмана (простой),
2. алгоритм Шеннона-Фано (простой),
3. алгоритм Лемпеля-Зива 77 года LZ77 (со скользящим окном)

Провести вычислительный эксперимент с целью оценки реализованных алгоритмов сжатия без потерь (упаковка/распаковка). Для проведения эксперимента с алгоритмами сжатия без потерь необходимо использовать набор из 36 файлов с именами “1”...”36”, выданных вместе с заданием. Рекомендуется для упаковки рассматривать исходный файл как поток байт.

Вычислить:

1. энтропию исходных файлов
2. коэффициент сжатия.
3. время упаковки;
4. время распаковки

Подготовить отчет по итогам работы, содержащий постановку задачи, описание алгоритмов и задействованных структур данных, описание реализации, обобщенные результаты измерения эффективности алгоритмов, описание использованных инструментов (например, если использовались скрипты автоматизации), оценку соответствия результатов экспериментальной проверки теоретическим оценкам эффективности исследуемых алгоритмов.

**Описание алгоритмов и использованных структур данных**

Кодирование алгоритмом Хаффмана:

1. Подсчитываем количество символов в кодируемом файле (используя отображение std::map<char, std::size\_t> count)
2. Строится таблица кодов

2.1. По отображению count создаются HuffmanNode-ы

2.2. Если вершина одна, построение таблицы окончено

2.3. Соединяя 2 вершины, дописываем к элементам одной 1, к элементам другой 0, переходим к шагу 2.2

1. Записываем таблицу частот(count), далее, используя таблицу отображений, создаем сжатый файл

Кодирование алгоритмом Хаффмана:

1. Считываем таблицу частот, создаем отображение код – буква.
2. Декодируем файл

Кодирование алгоритмом Фано:

1. Подсчитываем количество символов в кодируемом файле (используя отображение std::map<char, std::size\_t> count)
2. Строится таблица кодов

2.1. count сортируется по убыванию частоты

2.2. Если элементов 2 или более, ищется медиана, левой части приписываем 0 в начало, правой 1 в начало

2.3. повторяем действия 2.2. для левой и правой части

1. Записываем таблицу частот(count), далее, используя таблицу отображений, создаем сжатый файл

Кодирование алгоритмом Фано:

1. Считываем таблицу частот, создаем отображение код – буква.
2. Декодируем файл

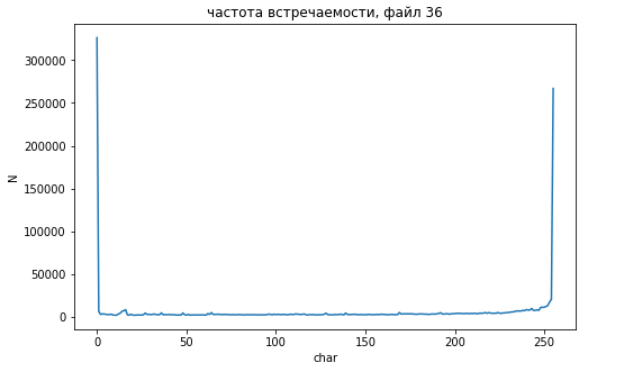
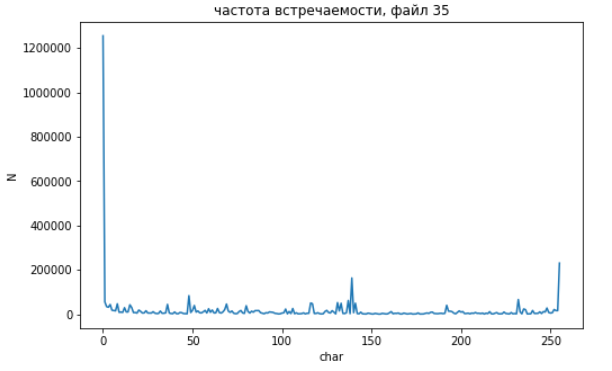
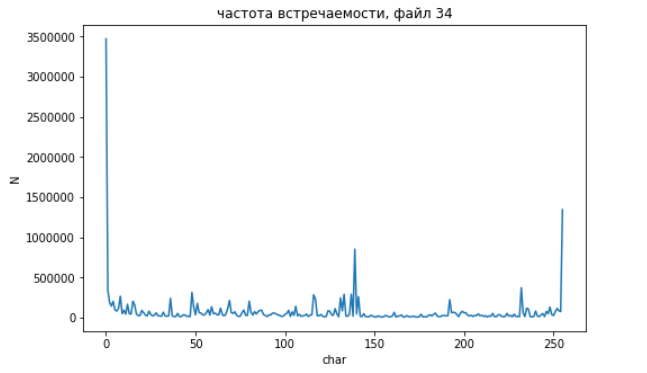
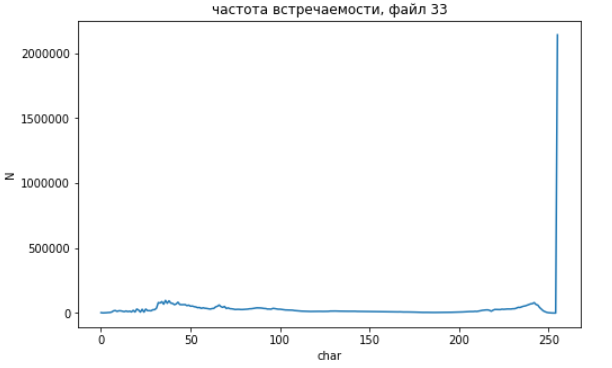
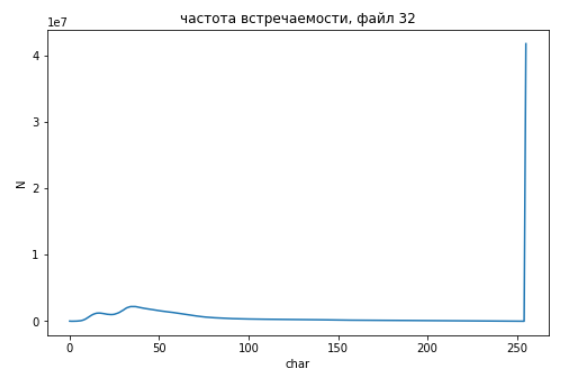
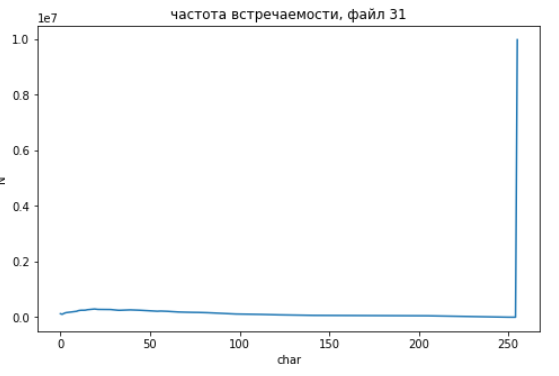
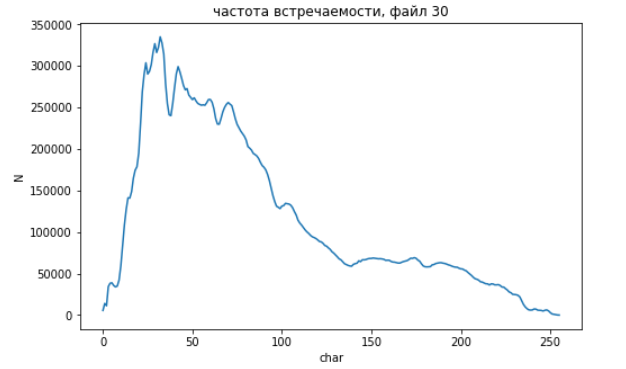
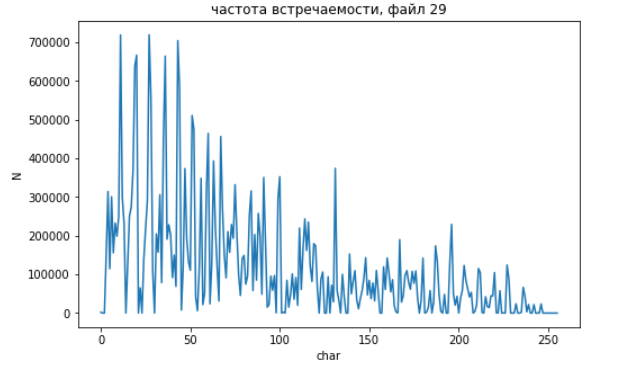
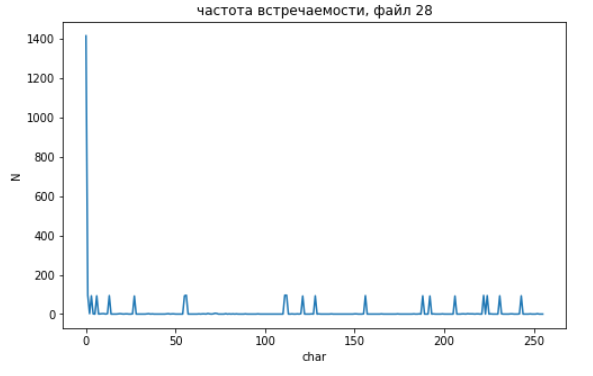
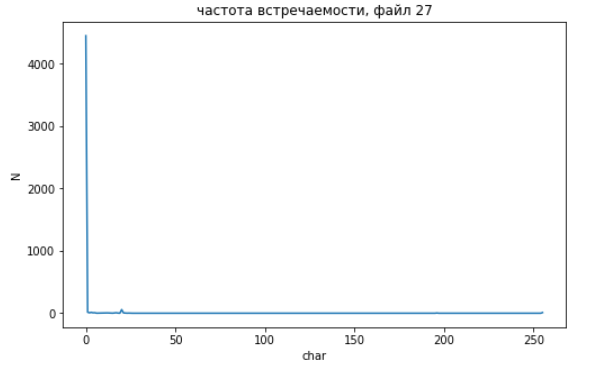
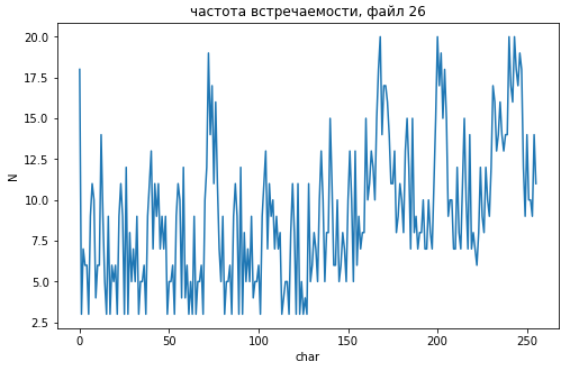
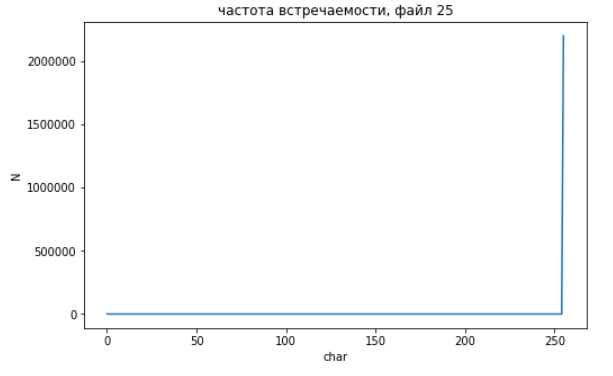
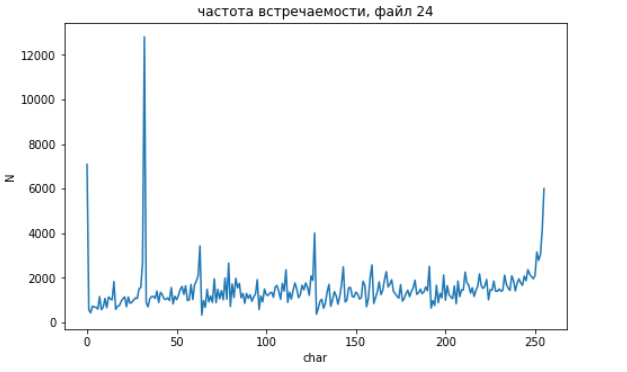
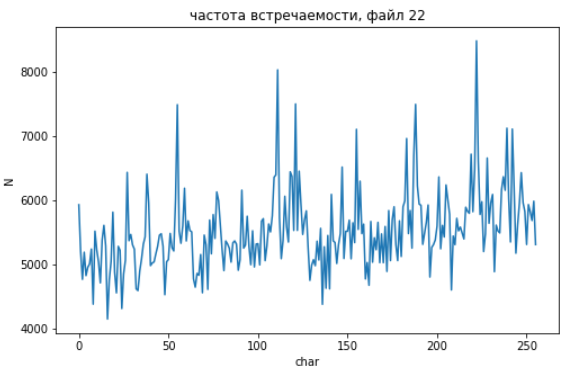
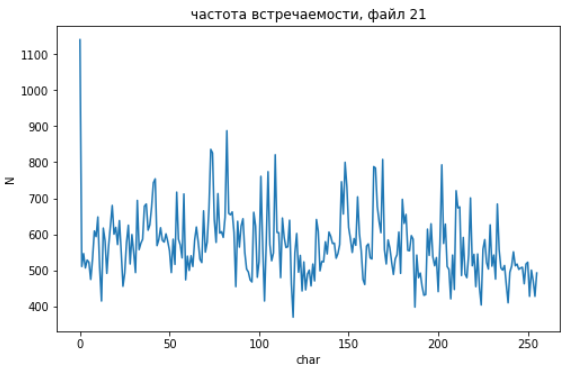
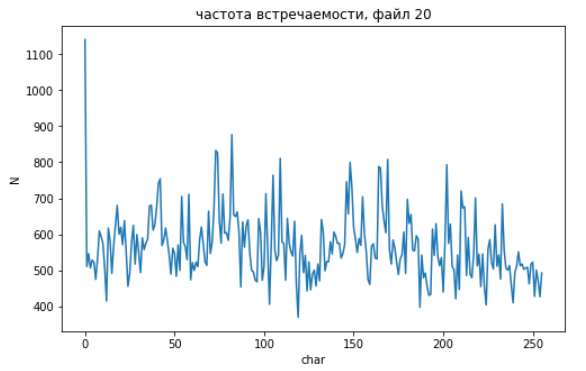
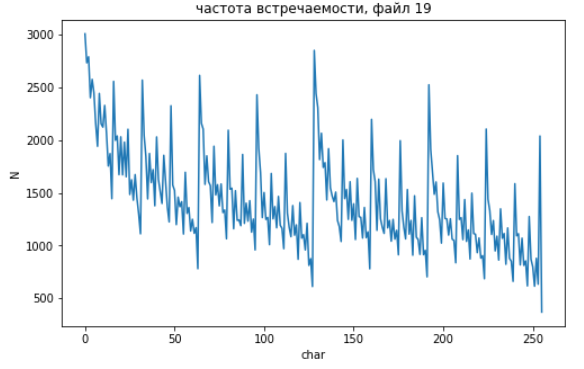
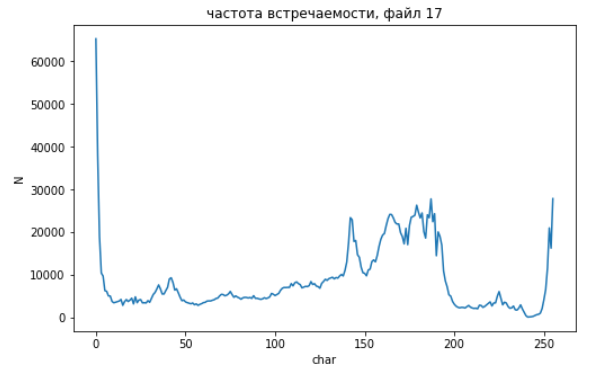
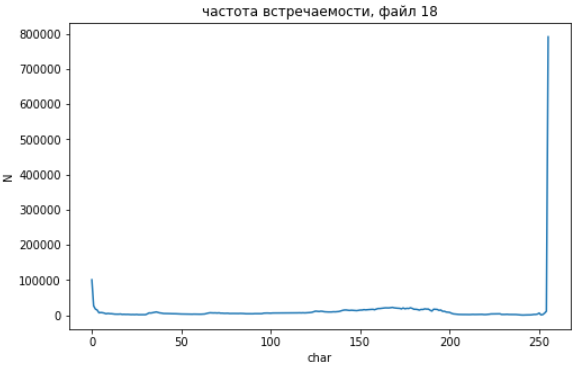
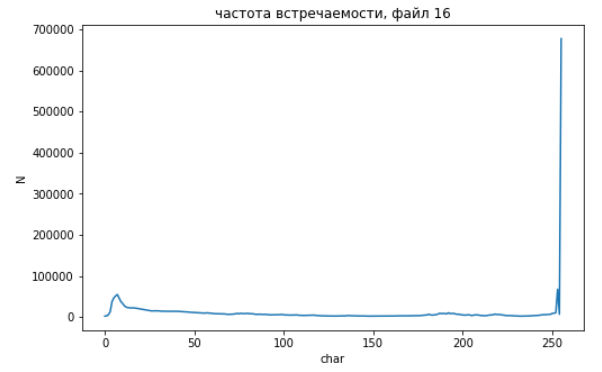
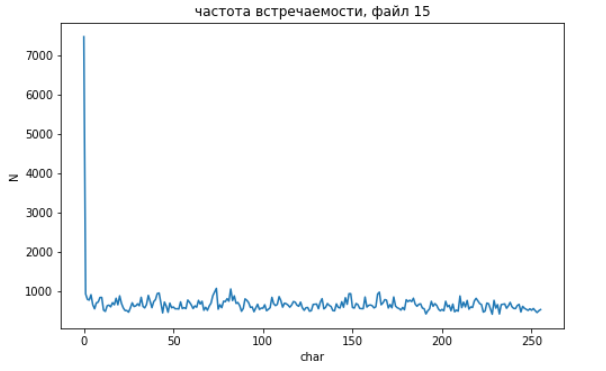
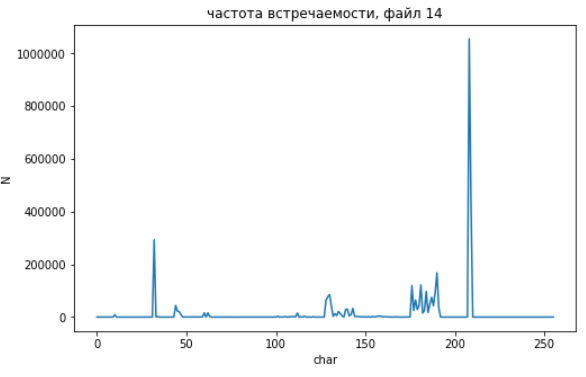
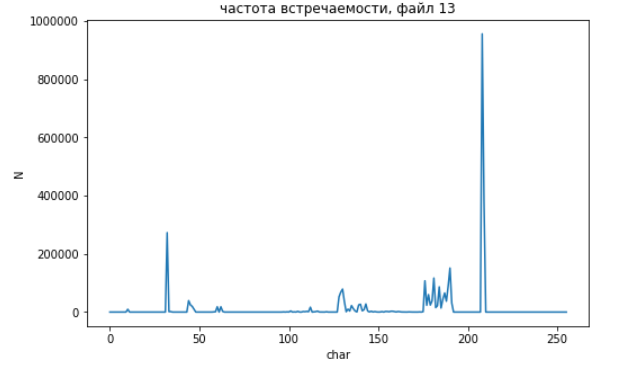
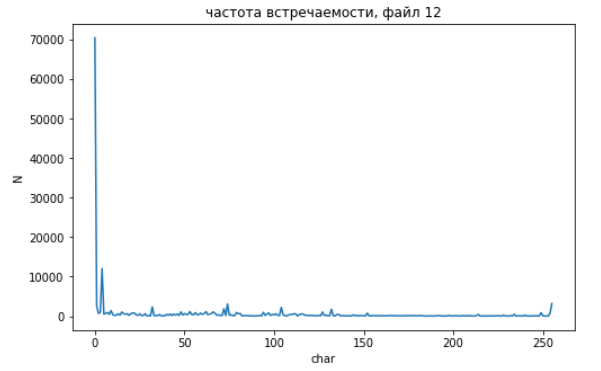
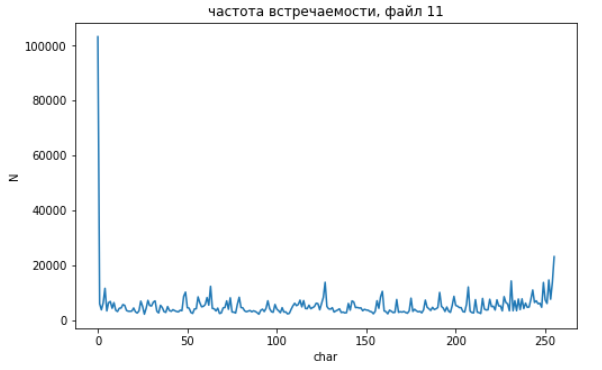
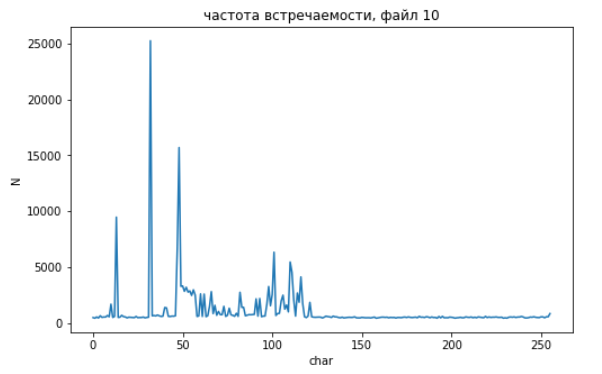
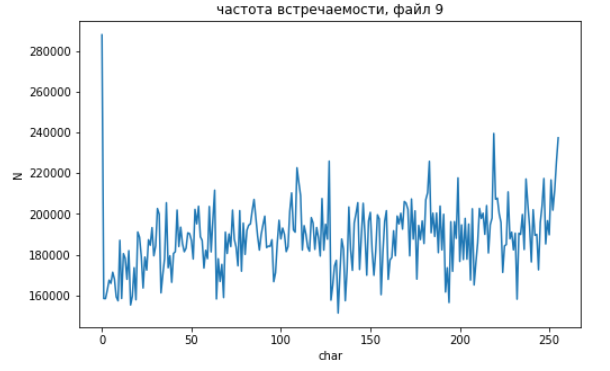
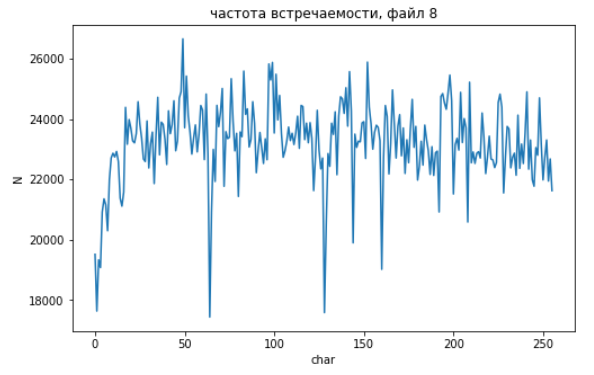
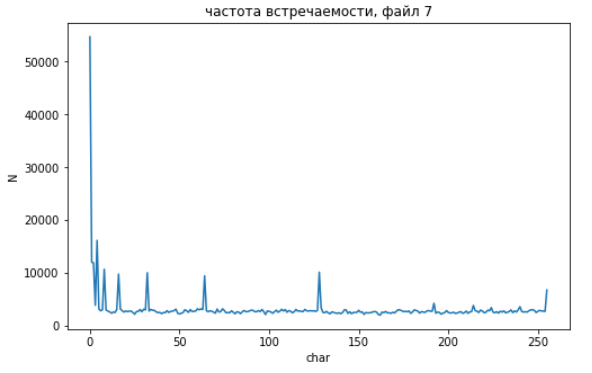
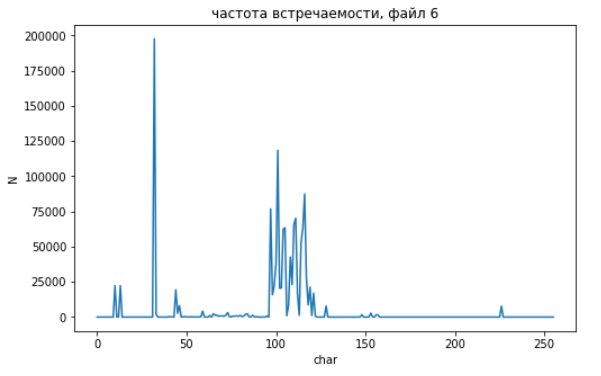
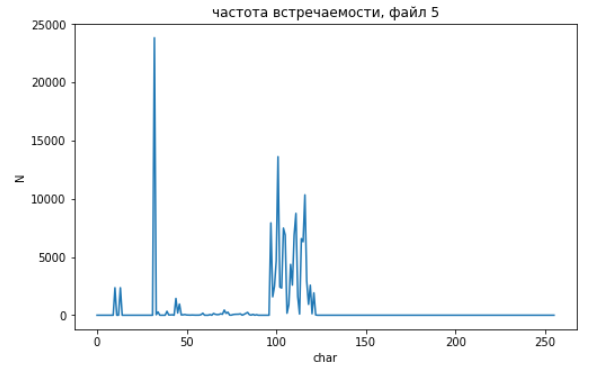
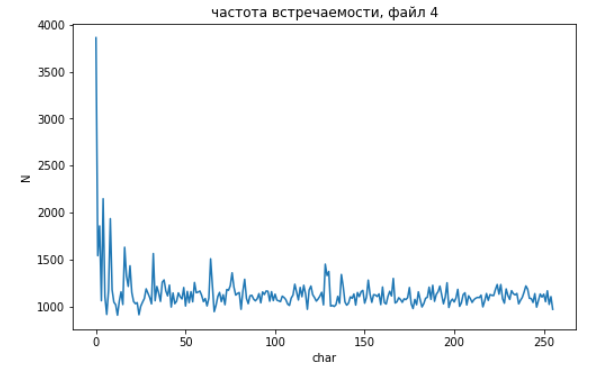
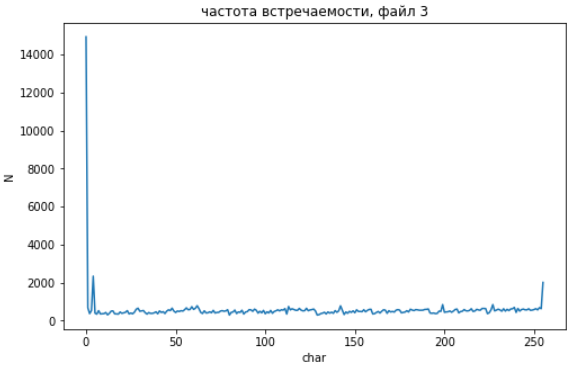
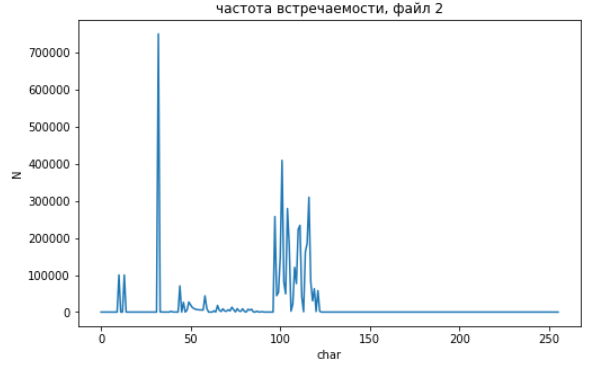
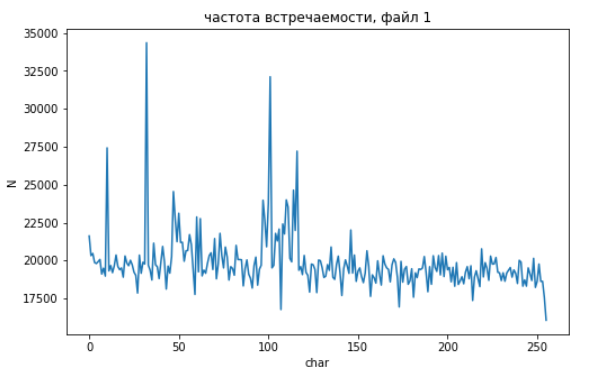
Кодирование алгоритмом LZ77:

1. Используем структуру циклический буфер, делим его на предысторию и на предпросмотр, заполняем секцию предпросмотра
2. Ищем совпадения в предыстории из предпросмотра, записываем найденную позицию, длину и следующий символ
3. Дописываем в буфер символы, равные максимальной найденной длине совпадения, повторяем шаг 2, пока секция предпросмотра не пуста

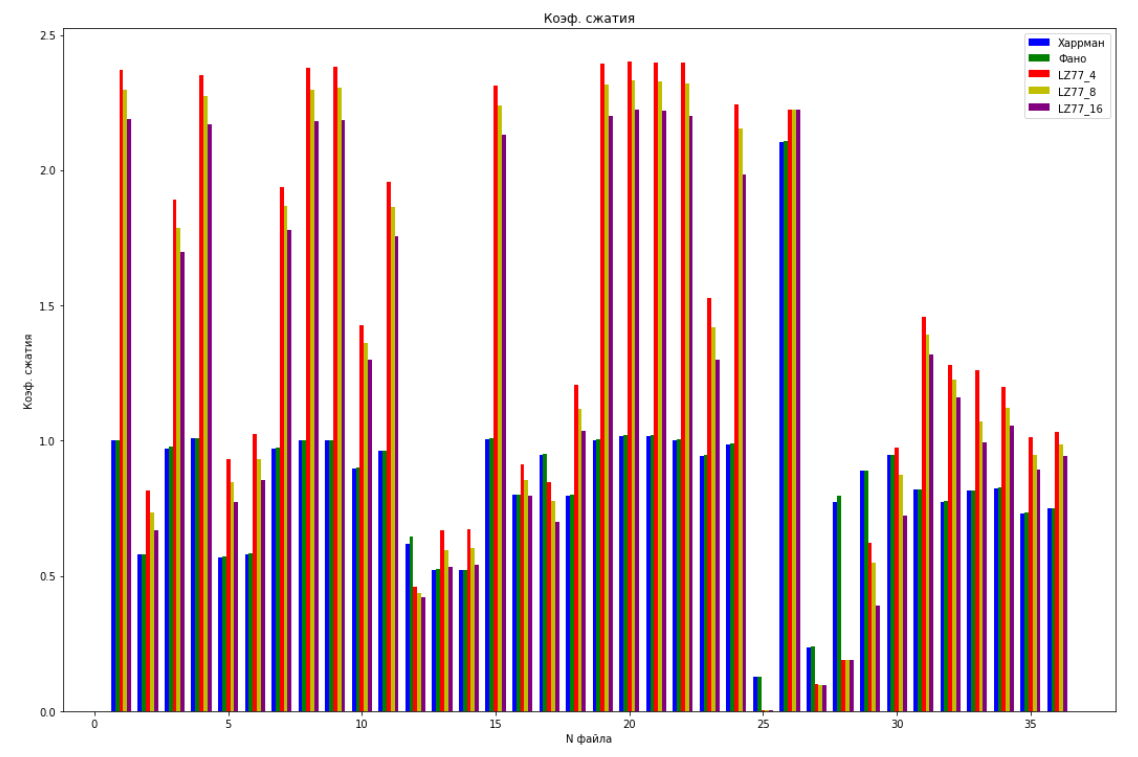
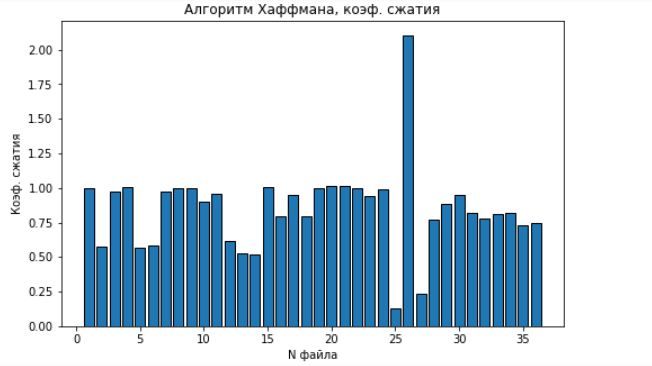
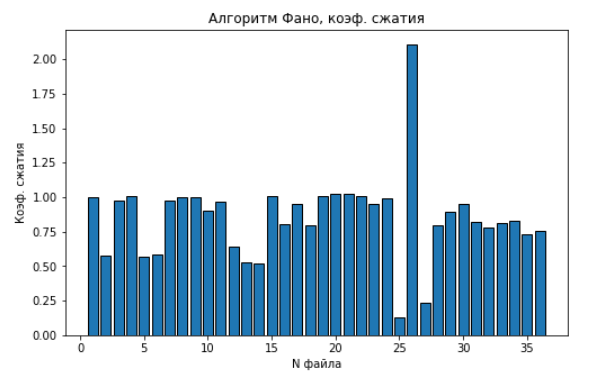
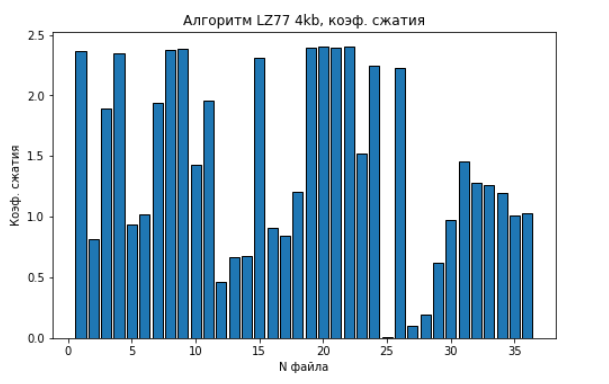
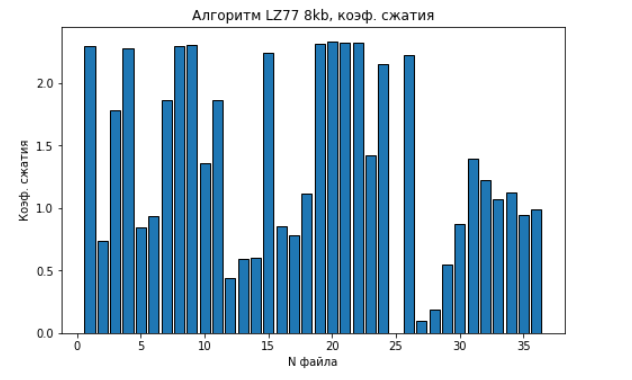
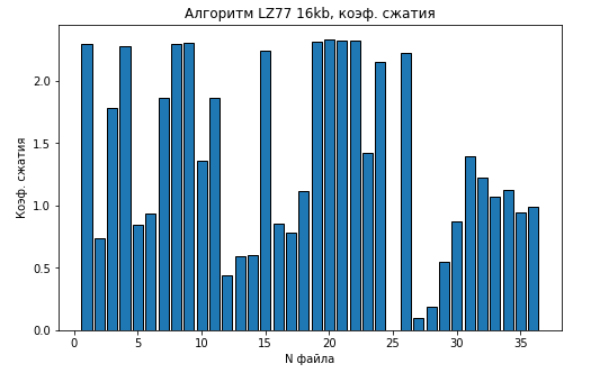
Кодирование алгоритмом Фано:

1. Заводим буфер предыстории
2. Считываем тройку: позиция, длина, следующая буква, обращаясь к буферу записываем совпадения и следующую букву
3. Дописываем в буфер символы равные длине, повторяем шаг 2, буфер не пуст

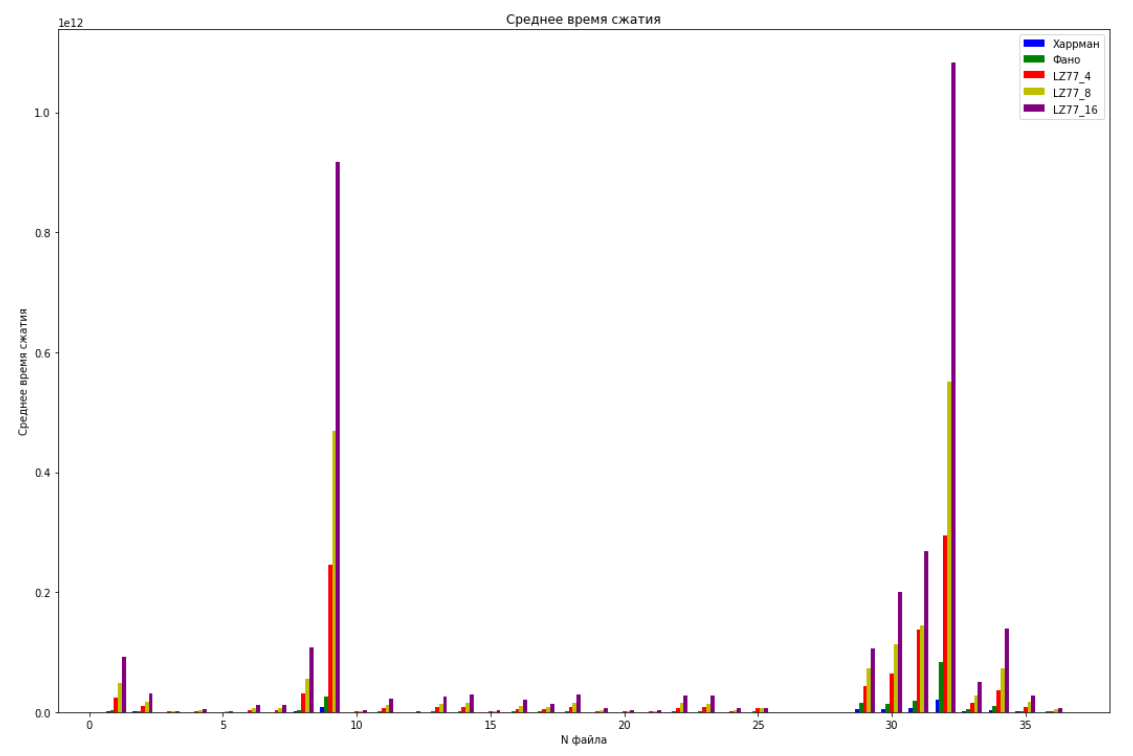
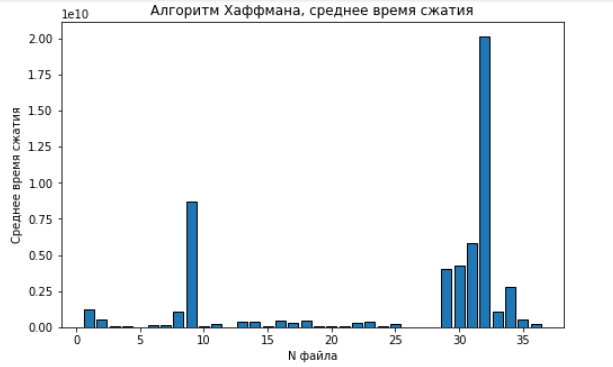
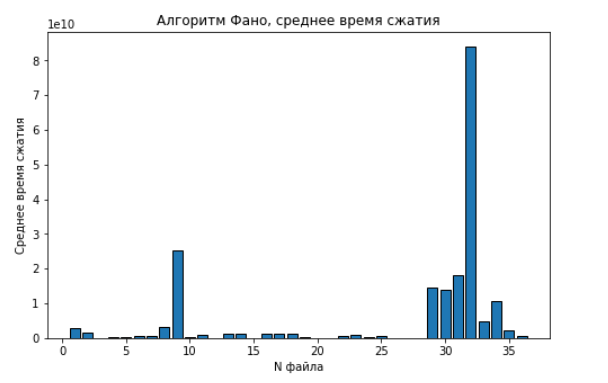
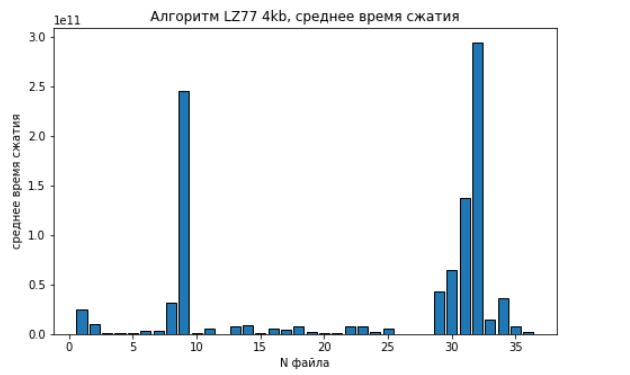
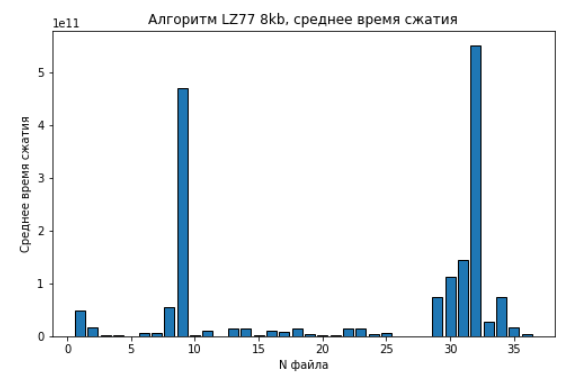
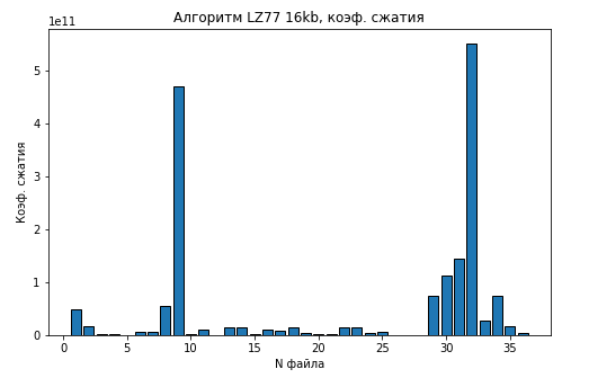
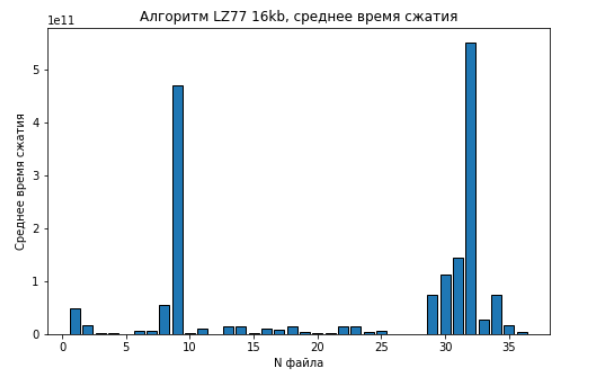
**Диаграммы распределения частот символов в файлах**

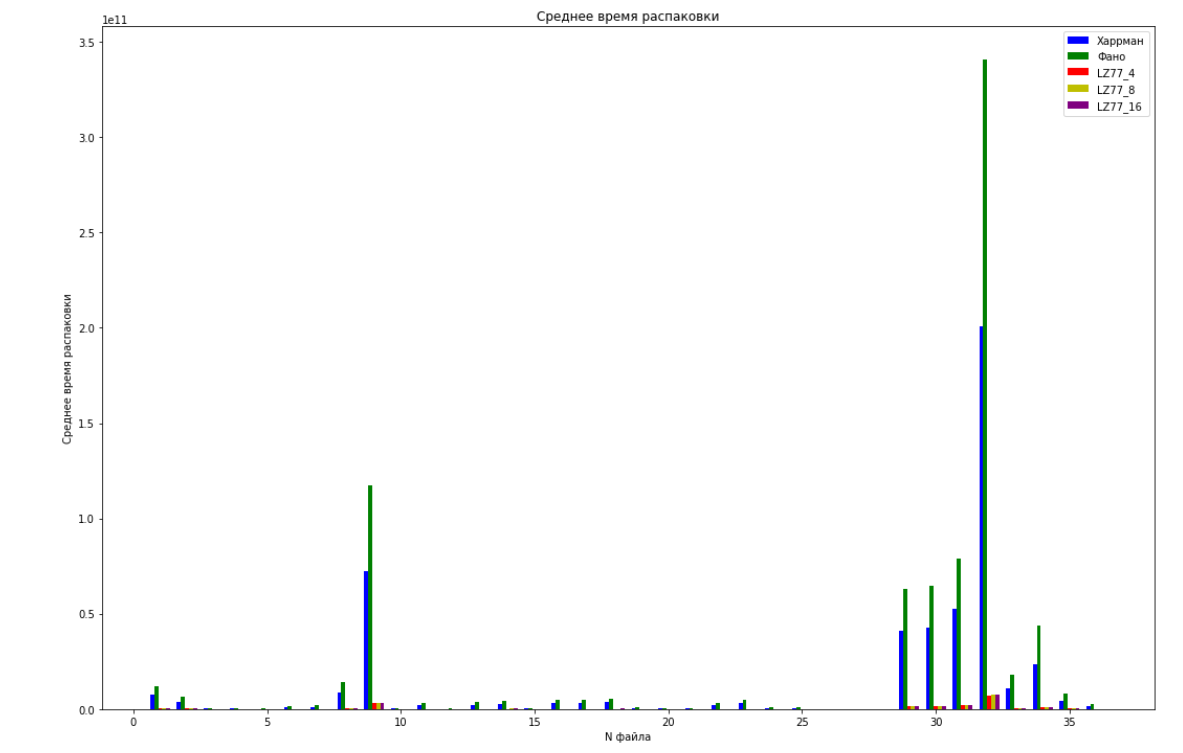
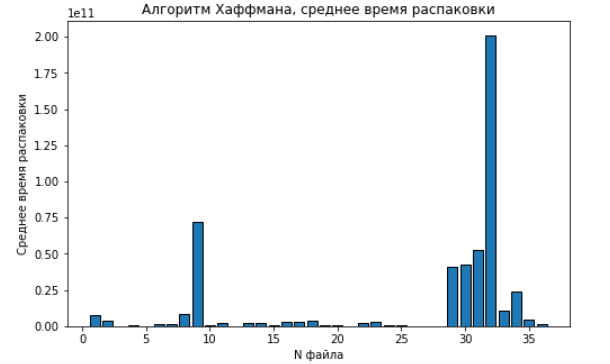
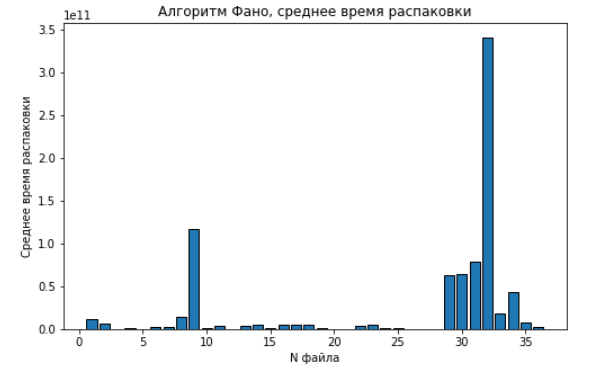
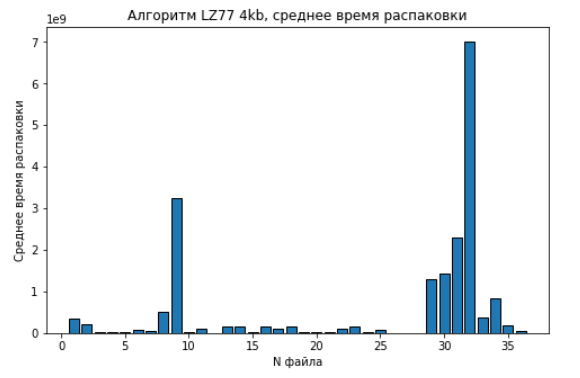
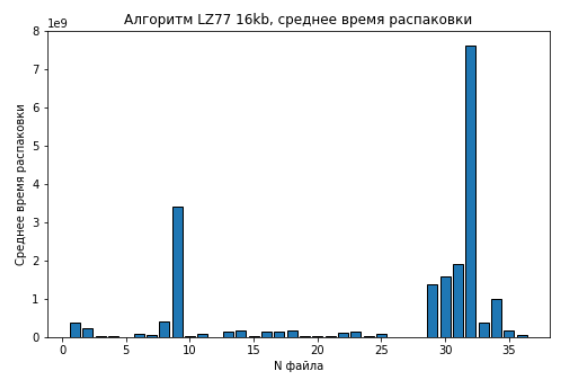


**Результаты экспериментов**

**Коэффициенты сжатия:**     

**Среднее время кодирования:**

**Среднее время распаковки:**      

**Сравнительный анализ алгоритмов**

Сравним теоретическую сложность через O асимптотику:

Сложность кодирования:

1. Алгоритм Хаффмана: время - O(n\* log|Σ|), память - O(|Σ|)
2. Алгоритм Шеннона-Фано: время - O(n\* log|Σ|), память - O(|Σ|)
3. LZ77 Алгоритм: время - O(n^3), память – O(1)

Сложность декодирования:

1. Алгоритм Хаффмана: время - O(n\* log|Σ|), память - O(|Σ|)
2. Алгоритм Шеннона-Фано: время - O(n\* log|Σ|), память - O(|Σ|)
3. LZ77 Алгоритм: время - O(n), память – O(1)

По результатам эксперимента легко заметить, что алгоритм кодирования Хаффмана является универсальным алгоритмом, который может успешно применяться к практически любым файлам. В некоторых случаях LZ77 показывает себя намного лучше, но таких файлов не так много и надо тщательно анализировать файлы, перед кодированием.  
  
В алгоритмах Хаффмана и Шенона-Фано кодирование равнозначно декодированию по времени, что не скажешь про ZL77. Алгоритм ZL77 достаточно долго кодирует информацию, зато декодирование очень быстрое.

**Выводы**

Алгоритмы Хаффмана и Шеннона-Фано почти универсальны и работают довольно хорошо на любых входных данных. Даже если символы в кодируемом файле будут и меть одинаковое распределение, алгоритмы Хаффмана и Шеннона-Фано сведутся к равномерному, и полученный файл увеличится только на размер таблицы.

В отличии от алгоритмов Хаффмана и Шеннона-Фано, ZL77 более чувствительный, при кодировании файлов с высокой энтропией они кодируют очень плохо, но при небольшой энтропии и высокой частоте повторения схожих символов показывает себя много лучше чем алгоритм Хаффмана и Шеннона-Фано.