НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Отчет по контрольному домашнему заданию по дисциплине:

“*Алгоритмы и структуры данных*”

Выполнил: студент группы БПИ154 (1)

Исаков А.Э.

2016

1. Постановка задачи

Реализовать с использование языка C++ программы для архивирования и разархивирования текстовых файлов (кодировка UTF-8). При этом использовать два известных алгоритма кодирования информации:

1) Хаффмана
2) Шеннона-Фано.

Провести вычислительный эксперимент с целью оценки реализованных алгоритмов архивации / разархивации. Оценить количество элементарных операций каждого алгоритма.

Для этого:

1. Подготовить тестовый набор из нескольких текстовых файлов разного объема (20, 40, 60, 80, 100 Кб; 1, 2, 3 Мб — всего 8 файлов) на разных языках (ru, en - кодировка UTF-8) с разным набором символов в каждом файле, а именно: 1. первый набор: символы латинского алфавита и пробел 2. второй набор: символы из первого набора + символы русского алфавита 3. третий набор: символы из второго набора + следующие знаки и спецсимволы: знаки арифметики „+ - * / =“, знаки препинания „. , ; : ? !“, символы „% @ # $ & ~‘’, скобки разных типов „( ) [ ] { } < >“ , кавычки „“„“),

2. Измерить (экспериментально) количество операций (в рамках модели RAM (см. лекционный материал)), выполняемых за время работы (архивирования, разархивирования) каждого алгоритма на нескольких различных (не менее трех) файлах для каждого размера входного файла и набора символов (итого получается 8 * 3 * 3 = 72 эксперимента по архивированию и 72 по разархивированию для каждого алгоритма, т. е. Всего минимум 144 * 2 = 288, имеет смысл задуматься об автоматизации). Для повышения достоверности результатов каждый эксперимент можно повторить несколько (5-10) раз на различных файлах (с одним возможным набором символов) одного размера с последующим усреднением результата.

2. Описание алгоритмов и использование структур данных

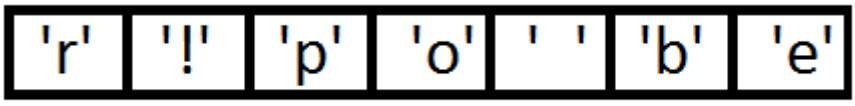
1. Алгоритм Хаффмана-алгоритм оптимального префиксного кодирования алфавита.

   Возьмем для пример строку : “beep boop beer!”. Чтобы получить код для каждого символа на основе частотности, нам надо построить бинарное дерево, такое, что каждый лист этого дерева будет содержать символ (печатный знак из строки). Дерево будет строиться от листьев к корню, в том смысле, что символы с меньшей частотой будут дальше от корня, чем символы с большей.

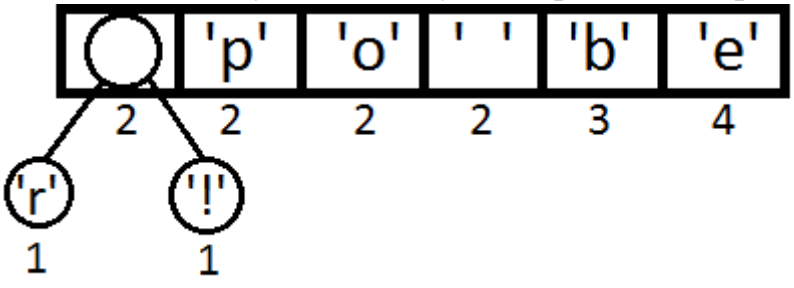
   Для начала посчитаем частоты всех символов:

Символ	Частота
'b'	3
'e'	4
'p'	2
' '	2
'o'	2
'r'	1
'!'	1

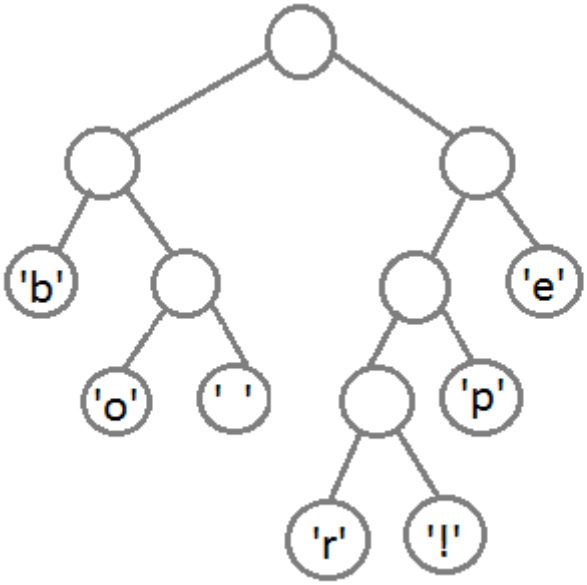
   После вычисления частот мы создадим узлы бинарного дерева для каждого знака и добавим их в таблицу, используя частоту в качестве приоритета (т.е. для каждого узла в таблице есть частота и она отсортирована по возрастанию)



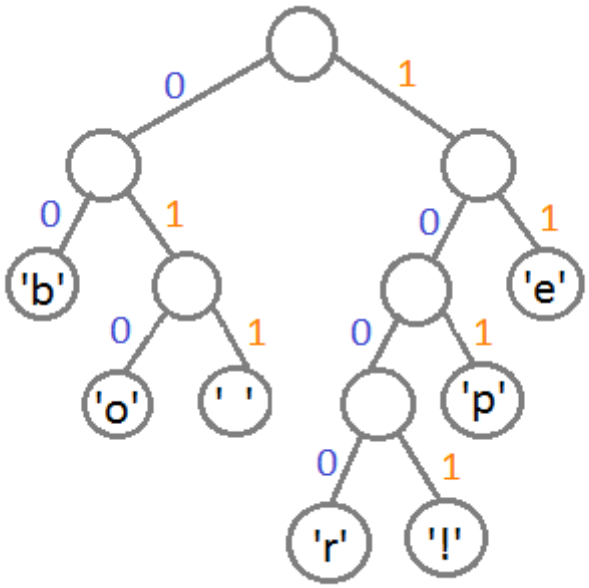
   Теперь мы достаем два первых элемента из таблицы и связываем их, создавая новый узел дерева, в котором они оба будут потомками, а приоритет нового узла будет равен сумме их приоритетов. После этого мы добавим получившийся узел обратно в очередь.



Повторим те же шаги и получим последовательно:



Теперь, чтобы получить код для каждого символа, надо просто пройтись по дереву, и для каждого перехода добавлять 0, если мы идем влево, и 1 -если направо:



После этого мы получим следующие коды символов:

Символ	Код
'b'	00
'e'	11
'p'	101
' '	011
'o'	010
'r'	1000
'!'	1001

Чтобы расшифровать закодированную строку, нам надо, соответственно, просто идти по дереву, сворачиваю в соответствующую каждому биту сторону до тех пор, пока мы не достигнем листа. И еще так как каждый код не является префиксом для кода другого символа, мы не можем получить конфликт, т.е. к примеру- если 00 код для ‘b’, то 000 не может оказаться чьим-либо кодом, т.к. а иначе мы никогда бы не достигли бы этого символа в дереве, потому что останавливались бы еще на ‘b’.

На практике, при реализации данного алгоритма сразу после построения дерева я строю таблицу Хаффмана. Она содержит каждый символ и его код (это делает кодирование более эффективным). А для декодирования я использую дерево Хаффмана, так как довольно затратно каждый раз искать символ и одновременно вычислять его код, так как мы не знаем где он находится, и придется обходить все дерево целиком. Я помещаю созданное дерево Хаффмана в закодированную строку, и чтобы получатель знал, как его интерпретировать, чтобы раскодировать сообщение, я использую способ прохода по дереву и конкантенирую 1 для каждого узла и 0 для листа с битами, представляющими оригинальный символ. Это представление я добавляю почти в самое начало закодированной строки (в начале закодированной строки я храню количество символов которое было в незакодированной)

2. Алгоритм Шеннона-Фано
   Алгоритм почти полностью совпадает с алгоритмом Хаффмана, за исключением построения дерева.
   Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева; такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины. Ветви кодового дерева размечаем символами 1 и 0, как в случае кода Хаффмана.

3. Использованные структуры данных:
   Бинарное дерево.

4. Описание плана эксперимента

Для генерации файлов была написана программа на C#.

Программа создает файлы размеров 20, 40, 60, 80, 100 Кб; 1, 2 и 3 Мб. Для каждого размера создает 3 файла с разными наборами символов (итого 24 файла):

0) символы латинского алфавита и пробел

1) символы из первого набора + символы русского алфавита

2) символы из второго набора + следующие знаки и спецсимволы: знаки арифметики „**+ - * / =**“, знаки препинания „**. , ; : ? !**“, символы„**%@#$&~**‘’,скобки разных типов „**()[]{}**<>“,кавычки„**„“**“

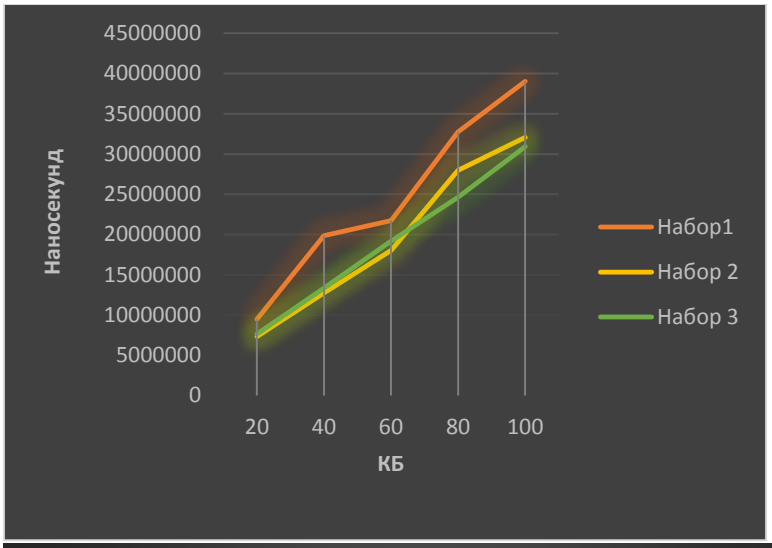
Для каждого файла измеряется время архивирования \ разархивирования в c++ и записывается в res.csv. в наносекундах.

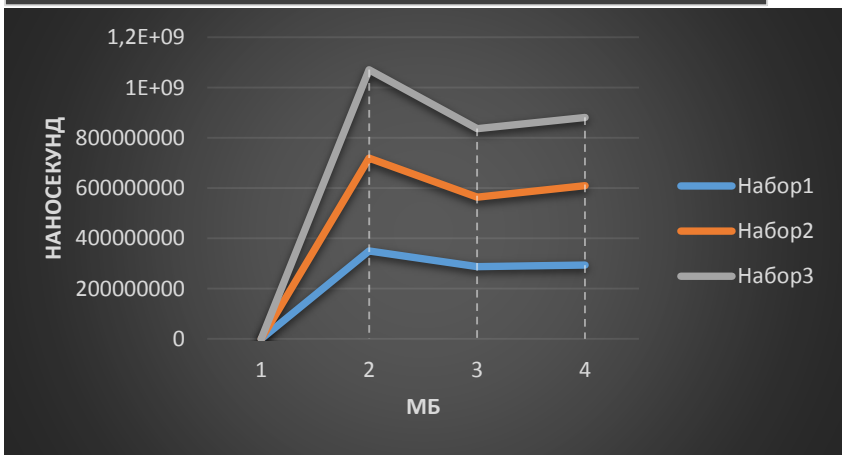
5.Результаты экспериментов

Следующая таблица содержит полную информацию обо всех экспериментах:

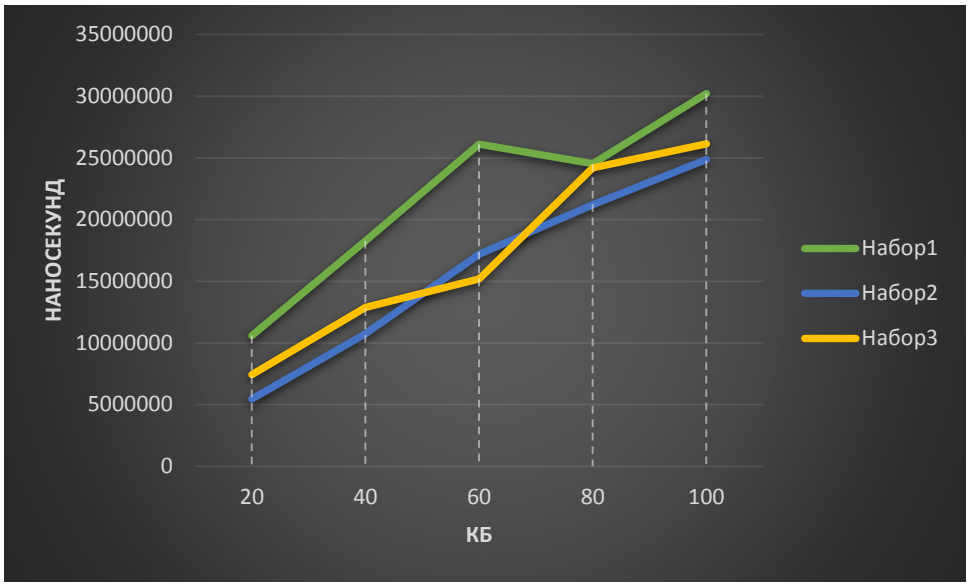
Имя файла	Размер	Арх:Хафф	Деарх:Хафф	Размер после арх:Хафф	Арх:Шен	Деарх:Шен	Размер после арх:Шен
0_20480	20	9498463	10595718	15	8588968	9242281	15
0_40960	40	19863886	18280153	29	15255745	17110428	29
0_61440	60	21751240	26113576	44	26988461	22027835	44
0_81920	80	32763818	24527334	58	28968583	29539619	58
0_102400	100	39027192	30232559	72	36500039	32023325	72
0_1048576	1	349940501	339265242	737	396432894	255211134	737
0_2097152	2	719148367	486577270	1473	701086022	453936895	1472
0_3145728	3	1071414349	700872376	2209	1042567214	669009328	2209
1_20480	20	7350708	5447104	11	7149149	6471661	11
1_40960	40	12736574	10750237	22	13095104	11549345	22
1_61440	60	18055046	17186262	32	19353974	16373584	32
1_81920	80	27978180	21221258	43	24287117	23970610	43
1_102400	100	32086386	24886434	53	31376170	26536707	53
1_1048576	1	287689292	224886719	538	301418494	208067767	539
1_2097152	2	564041176	426399978	1076	599118786	388685397	1078
1_3145728	3	836850015	598681172	1614	1004451056	602200953	1616
2_20480	20	7646517	7427226	12	7796931	4969178	12
2_40960	40	13363602	12879918	23	14768526	9787659	23
2_61440	60	19181038	15196732	35	21711270	12059314	35
2_81920	80	24627230	24190928	46	25176770	16645104	46
2_102400	100	30931258	26158221	57	33291063	20207591	57
2_1048576	1	294659177	245890624	581	320466624	228028812	582
2_2097152	2	609375304	418519971	1163	619037809	426565273	1164
2_3145728	3	881309658	686189091	1744	931111321	691900987	1746

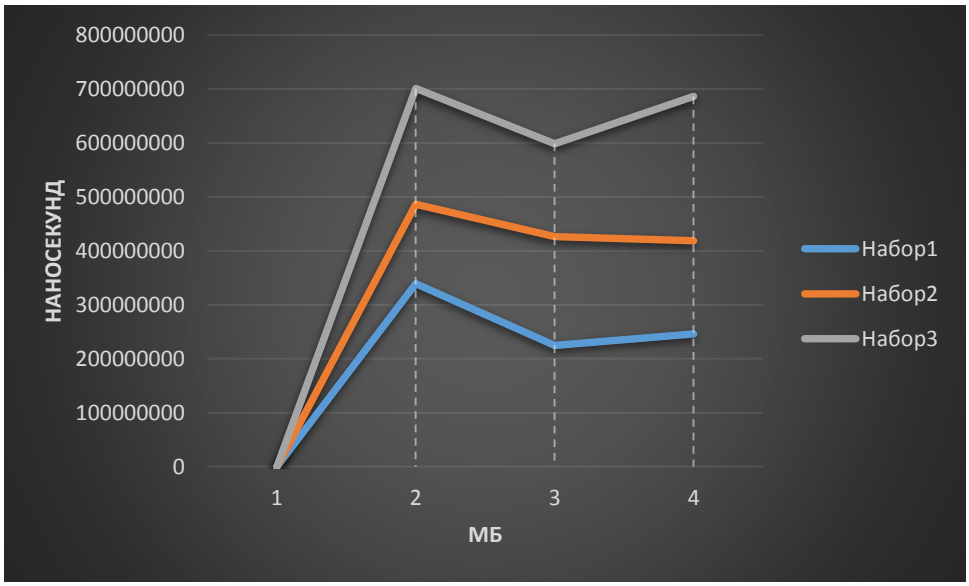
Время архивирования алгоритмом Хаффмана трех разных наборов символов:



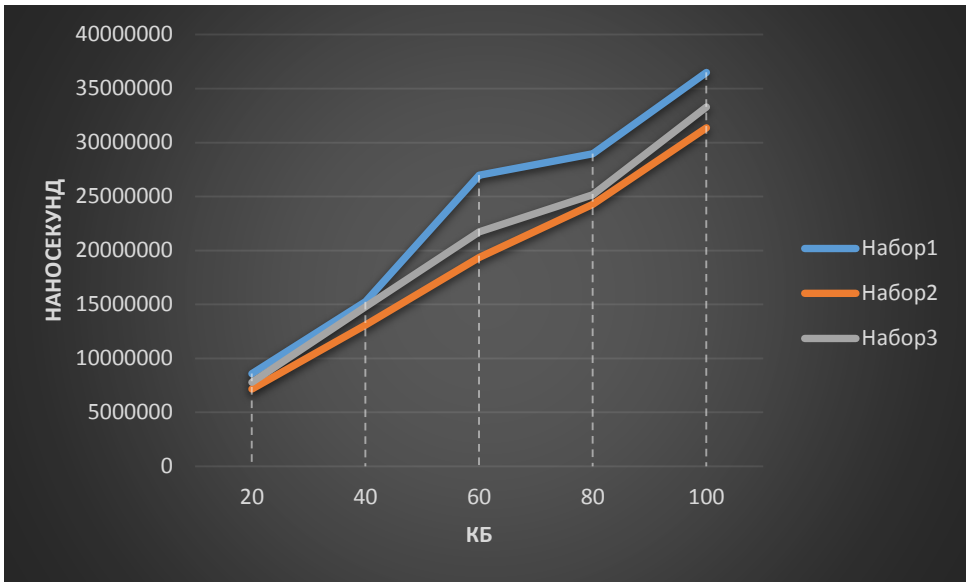


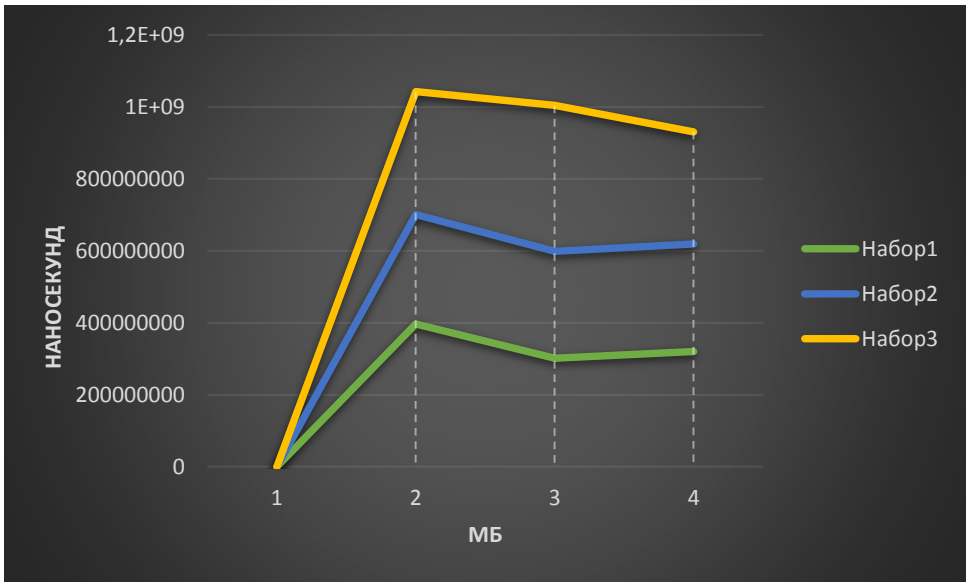
Время разархивирования алгоритмом Хаффмана трех разных наборов символов:



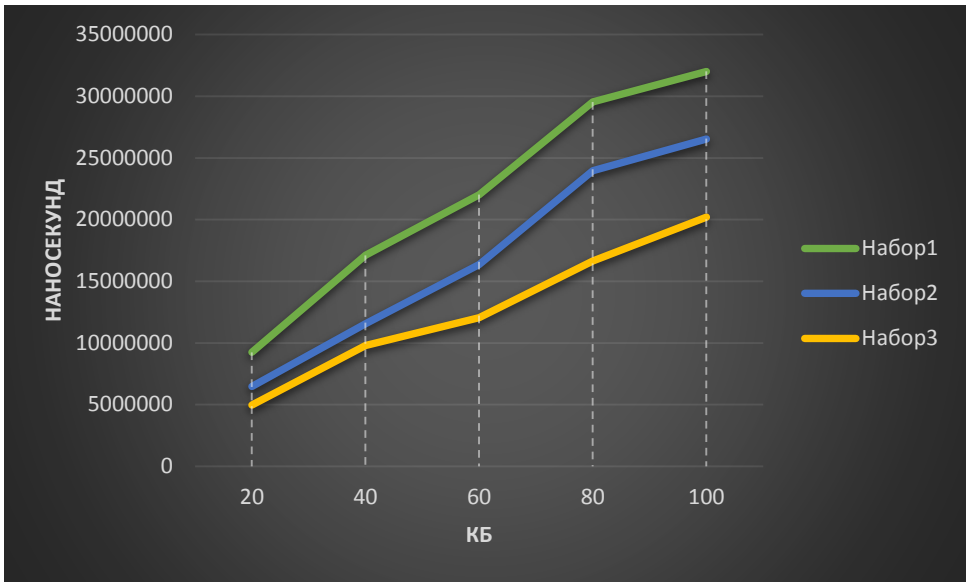


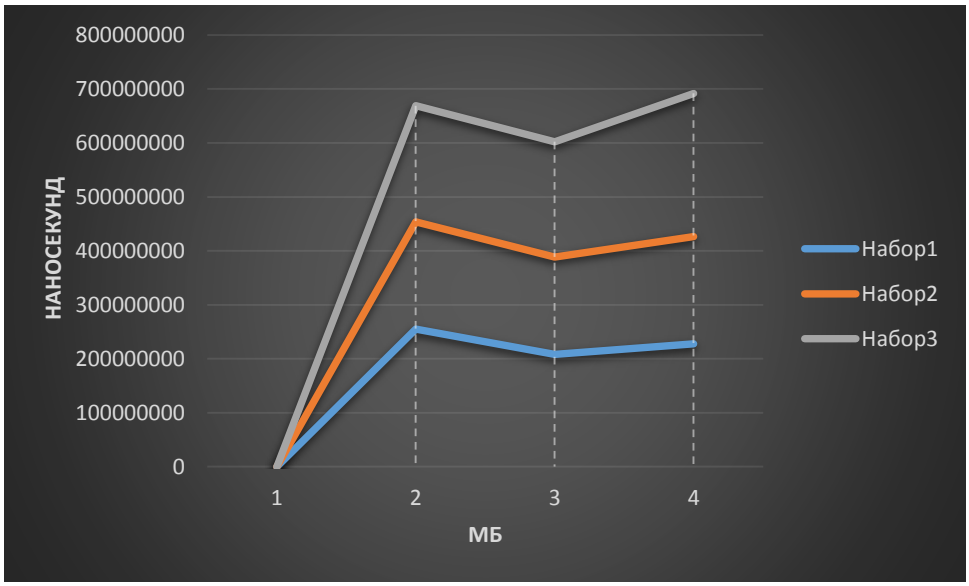
Время архивирования алгоритмом Шеннона-Фано трех разных наборов символов:



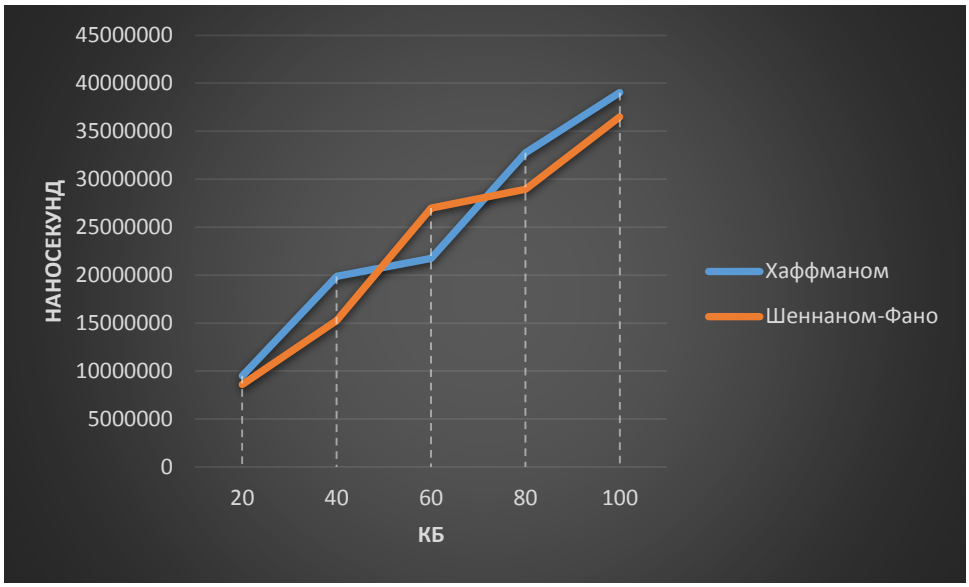


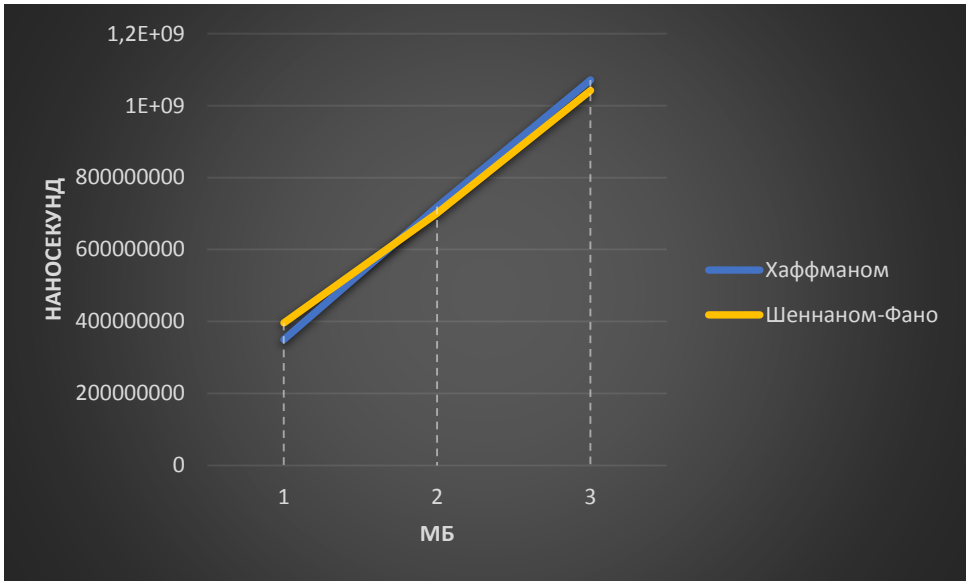
Время разархивирования алгоритмом Шеннона-Фано трех разных наборов символов:



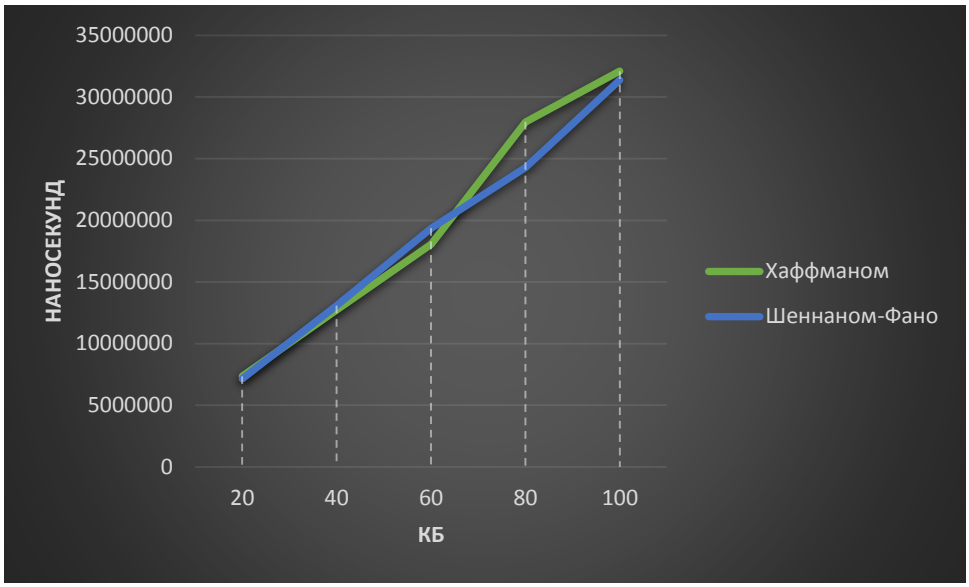


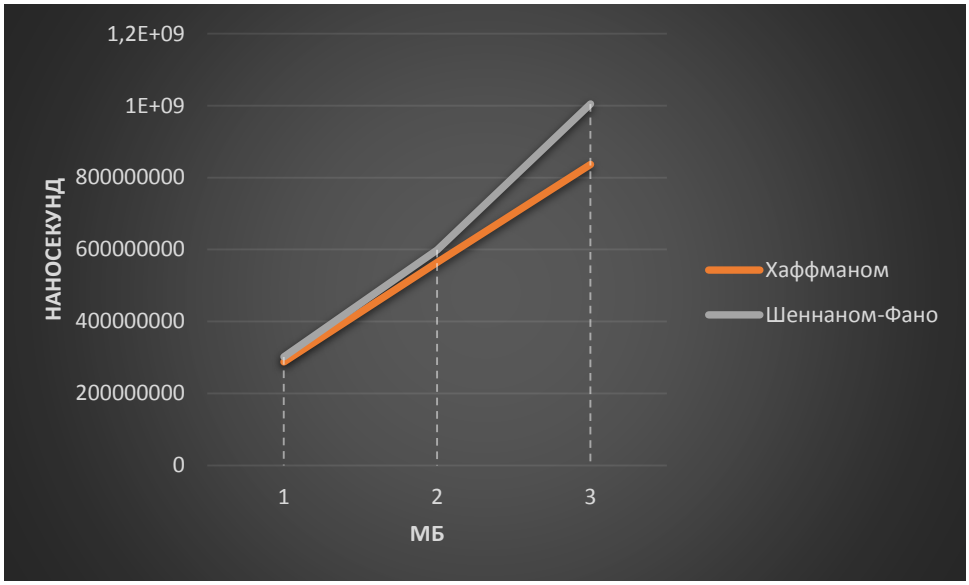
Время архивирования для набора символов 1:



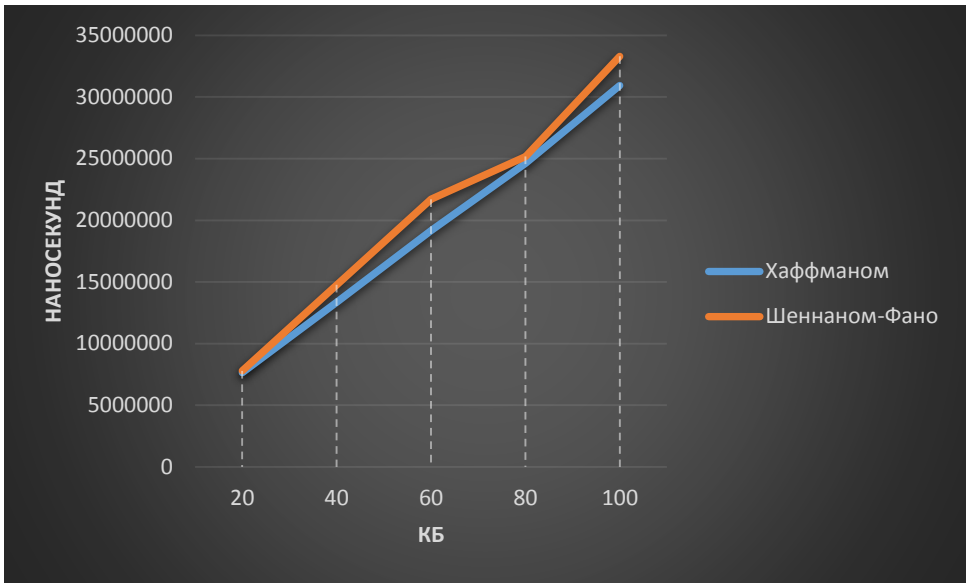


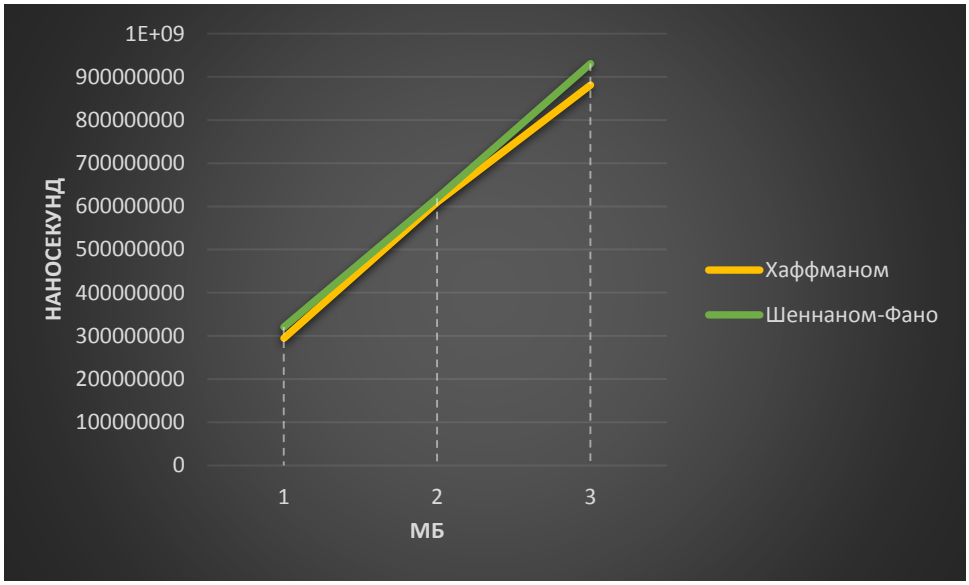
Время архивирования для набора символов 2:



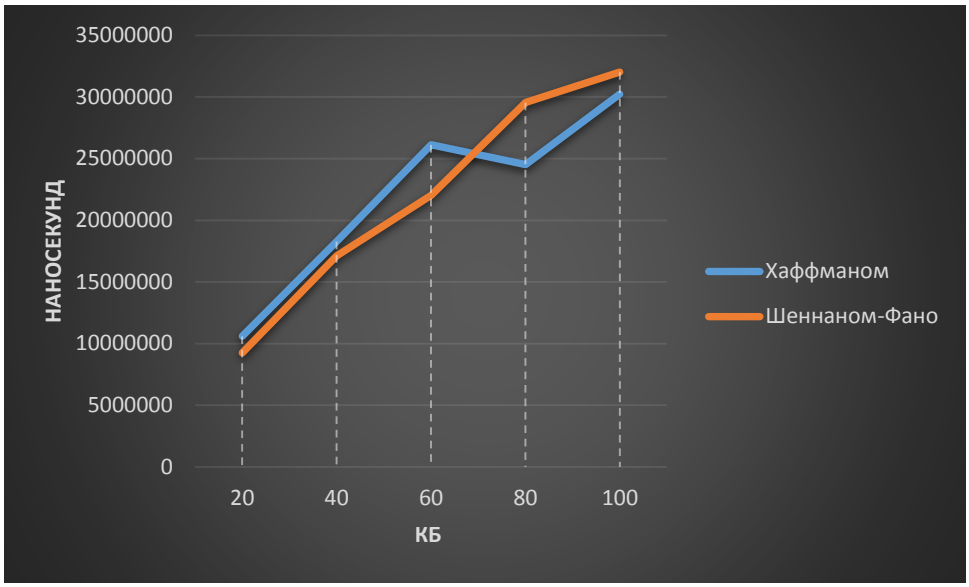


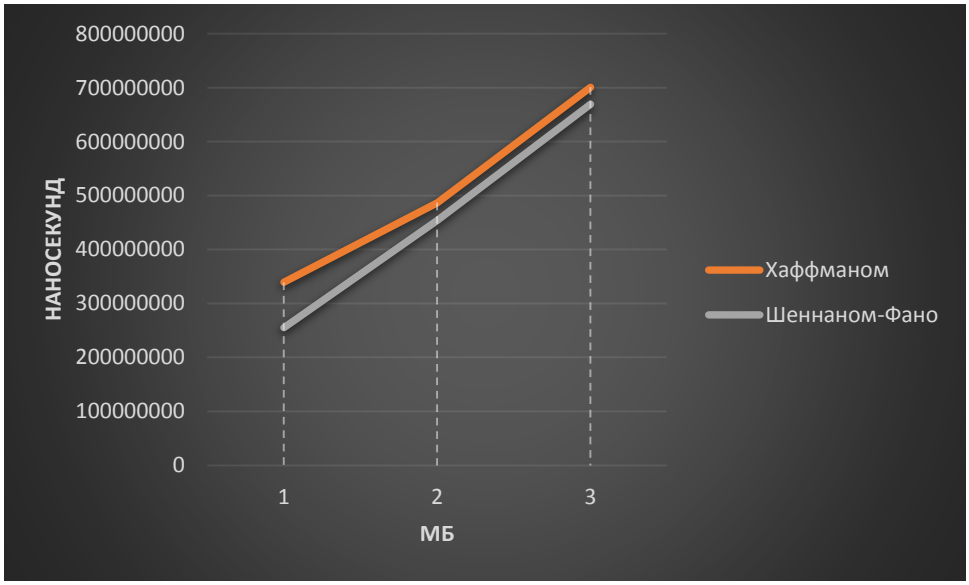
Время архивирования для набора символов 3:



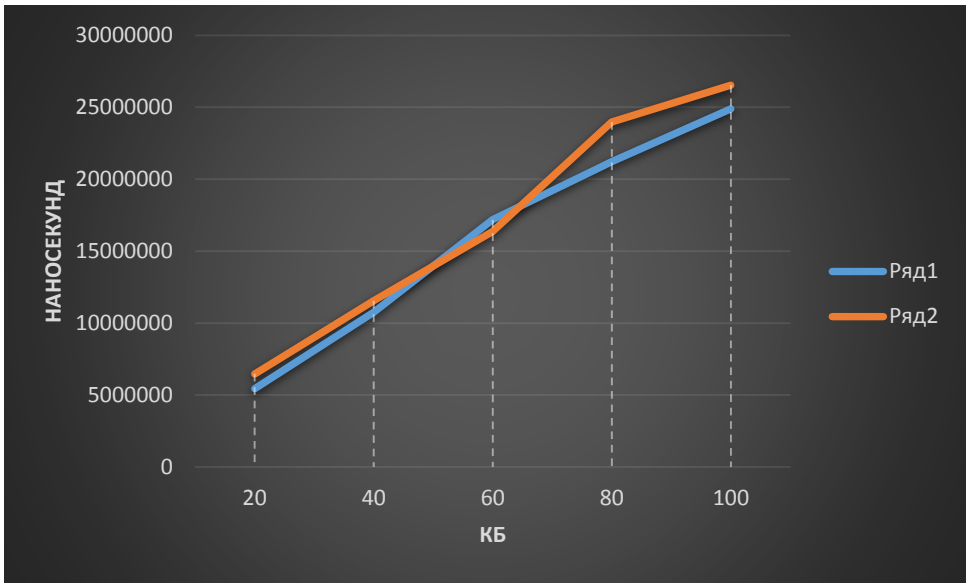


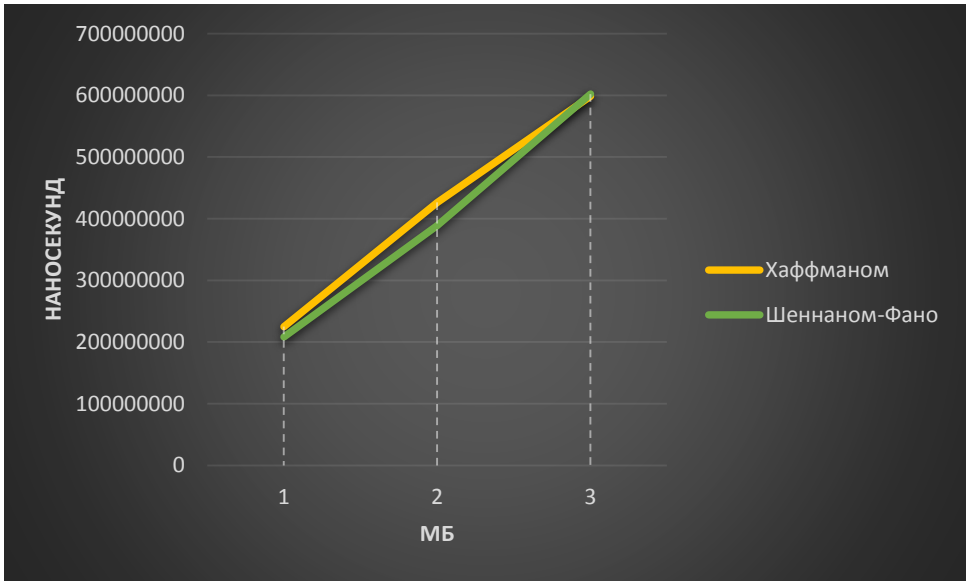
Время разархивирования для набора символов 1:



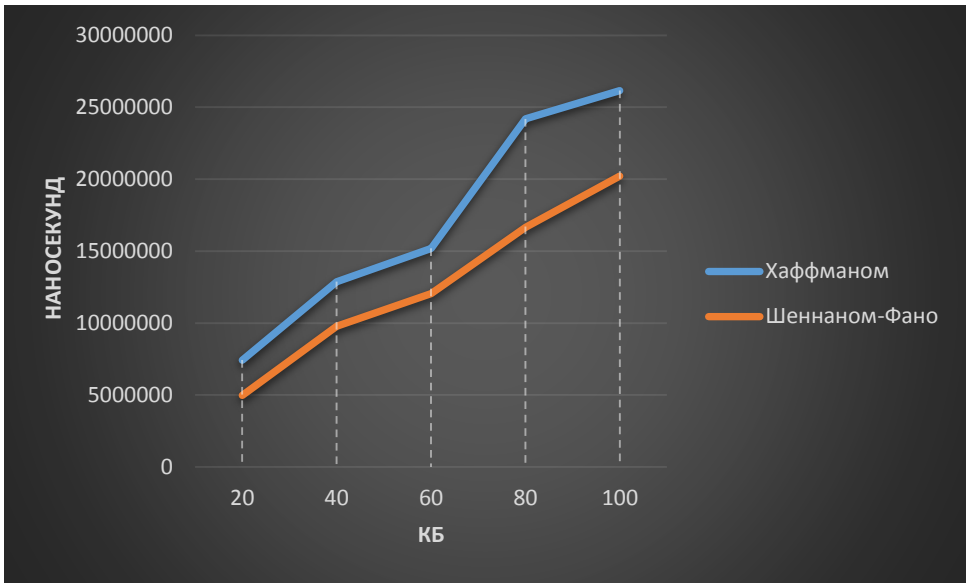


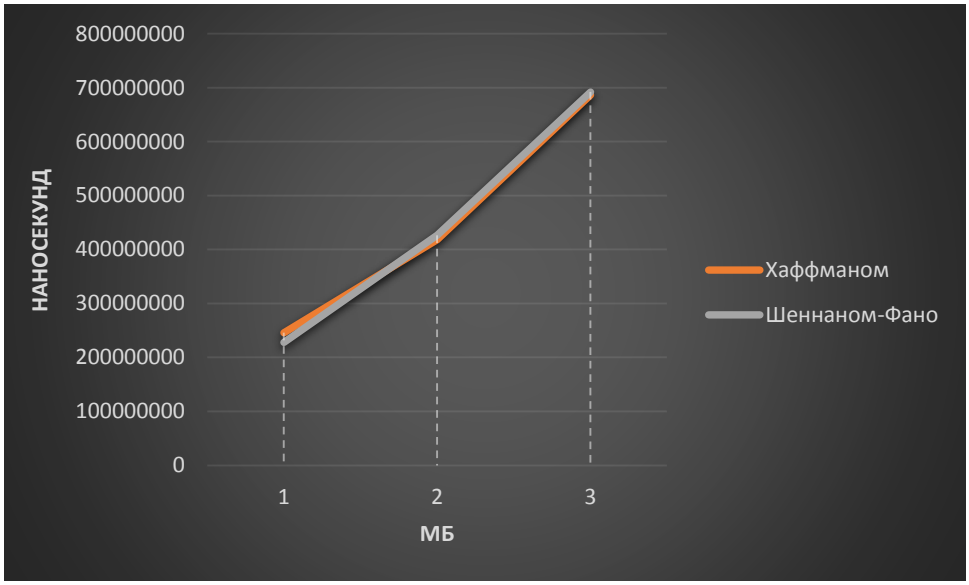
Время разархивирования для набора символов 2:





Время разархивирования для набора символов 3:





6.Сравнительный анализ методов

Хаффман:

Сложность: O(N*logN)

Достоинства: 1) в отличие от алгоритма Шеннона-Фано остается всегда оптимальным и для вторичных алфавитов m_2 с более чем двумя символами.

2)Кодирование Хаффмана широко применяется при сжатии данных, в том числе при сжатии фото- и видеоизображений (JPEG, MPEG), в популярных архиваторах (PKZIP, LZH и др.), в протоколах передачи данных HTTP (Deflate), MNP5 и MNP7 и других.

3) Метод Хаффмана производит идеальное сжатие (то есть, сжимает данные до их энтропии), если вероятности символов точно равны отрицательным степеням числа 2

Недостатки:

1) При энтропии=1 (алфавит состоит из 0 и 1) Хаффман не сжимает файл.
2) Пользователь должен знать, как интерпретировать дерево для разархивирования.

Шеннон-Фано:

Сложность: O(N*logN)

Достоинства: 1) Придуман немного раньше

Недостатки: 1) Кодирование Шеннона — Фано является достаточно старым методом сжатия, и на сегодняшний день оно не представляет особого практического интереса

2) Методика Шеннона–Фано не всегда приводит к однозначному построению кода. Ведь при разбиении на подгруппы на 1-й итерации можно сделать большей по вероятности как верхнюю, так и нижнюю подгруппу. В результате среднее число символов на букву окажется другим.
Таким образом, построенный код может оказаться не самым лучшим.

3)А также имеет такие же недостатки, как и Хаффман.

7.Заключение(вывод)

В данном отчете были описаны два реализованных алгоритма, был проведен их сравнительный анализ, было проведено тестирование, на основе которого были сделаны графики и таблица с данными. После всего этого можно сделать несколько выводов:

1) Алгоритмы работают почти за одинаковое линейное время
2) Хаффман является более эффективным алгоритмом, чем Шеннон-Фано и используется в некоторых существующих архиваторах.
3) Русские символы сжимаются лучше латинских, так как они всегда занимают два байта.
4) Получены полезные знания и потрачено много времени!!

8.Использованные источники

https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0

https://habrahabr.ru/post/144200/

https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0

https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%B5%D0%BD%D0%BD%D0%BE%D0%BD%D0%B0_%E2%80%94_%D0%A4%D0%B0%D0%BD%D0%BE

https://en.wikipedia.org/wiki/UTF-8

https://ru.wikipedia.org/wiki/UTF-8

http://cppstudio.com/post/446/

https://habrahabr.ru/post/137766/

http://fkn.ktu10.com/?q=node/3324

2016