НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: Алгоритмы и структуры данных

Контрольное домашнее задание

Выполнила:
студентка II курса
факультета компьютерных наук
образовательной программы
«Программная инженерия»
группы БПИ162
Игнатьева Мария Михайловна

Москва 2018

Оглавление

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:

- энтропию исходных файлов; определяется общее количество различных символов m, вычисляется их частотная встречаемость w_i в файле, и энтропия файла по формуле $H = \sum_{i=1}^{m} w_i \log_m w_i$;
- коэффициент сжатия, как отношение размера выходного файла и к размеру входного файла.

Измерить для каждого файла и для каждого алгоритма:

- время упаковки;
- время распаковки.

Время измерять в тактах ЦП или в наносекундах. Для получения достоверных результатов упаковку и распаковку каждого файла каждым методом выполнить не менее 20 раз, после чего вычислить среднее время работы алгоритма. Количество экспериментальных измерений времени не менее (20 раз * 10 (или 12) алгоритмов 36 файлов) = 720 (учтены алгоритмы упаковки/распаковки LZ77 с разным размером окна).

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. Отчет также должен содержать описание аппаратных средств и показатели качества архивации, данные о времени работы каждого алгоритма с каждым файлом из тестового набора.

Описание алгоритмов

Во всех нижеперечисленных алгоритмах файлы считываются и записываются по байтам. В качестве байта используется тип данных char.

Алгоритм Хаффмана

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана.

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который равен количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге (левой), выходящей из родителя, ставится в соответствие бит 1, другой (правой) — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Затем в сжатое сообщение сначала записываются следующие элементы:

- размер таблицы встречаемости символов
- таблица встречаемости символов
- количество незначащих нулей в конце закодированного сообщения (нули дополняющие последние биты закодированного сообщения до байта)
- закодированное сообщение

Таблица встречаемости символов записывается как последовательность следующих байт: символ, a = n / 256^3, b = (n – a * 256^3) / 256^2, c = (n – a * 256^3 – b * 256^2) / 256, d = n – a * 256^3 – b * 256^2 – c * 256; где / - деление нацело, ^ - возведение в степень.

При декодировании сначала восстанавливается таблица встречаемости символов, а потом уже коды для каждого символа.

Алгоритм Шеннона-Фано

Основные этапы:

1. Символы первичного алфавита выписывают по убыванию количества встречаемости в сообщении.
2. Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
4. Полученные части рекурсивно делятся и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Когда размер подалфавита становится равен нулю или единице, то дальнейшего удлинения префиксного кода для соответствующих ему символов первичного алфавита не происходит, таким образом, алгоритм присваивает различным символам префиксные коды разной длины.

Затем в сжатое сообщение записываются элементы, такие же, как и в алгоритме Хаффмана. Декодирование происходит аналогично.

Алгоритм LZ77

В кодируемых строках часто содержатся совпадающие длинные подстроки. Идея, лежащая в основе LZ77, заключается в замене повторений на ссылки на позиции в тексте, где такие подстроки уже встречались.

Информацию о повторении можно закодировать парой чисел — смещением назад от текущей позиции (offset) и длиной совпадающей подстроки (length).

Алгоритм LZ77 кодирует ссылки блоками из трёх элементов ⟨offset, length, next⟩. В дополнение к двум уже описанным элементам, новый параметр next означает первый символ после найденного совпадающего фрагмента. Если LZ77 не удалось найти совпадение, то считается, что offset = length = 0.

Для эффективного поиска повторов в LZ77 применяется метод «скользящего окна» — совпадения ищутся не на всём обработанном префиксе, а в небольшом буфере, состоящем из последних обработанных символов. Таким образом, при больших объемах ввода алгоритм тратит меньше времени за счет того, что просматривается не вся исходная строка.

Для декодирования LZ77 необходимо пройти по уже раскодированной строке назад, вывести необходимую последовательность, затем следующий символ.

В закодированный файл значения offset и length записываются как последовательность из 2 байт каждое по аналогии с записью таблицы частот встречаемости в алгоритме Хаффмана.

Описание использованных структур данных

std::map

std::map — отсортированный ассоциативный контейнер, который содержит пары ключ-значение с неповторяющимися ключами. Порядок ключей задаётся функцией сравнения Compare. Операции поиска, удаления и вставки имеют логарифмическую сложность. Данный тип, как правило, реализуется как красно-чёрное дерево.

Используется для доступа к коду символа, и для доступа к символу по его коду.

std::priority_queue

Очередь с приоритетом – это тип контейнера, который позволяет достичь константной скорости доступа к максимальному (или минимальному, в зависимости от реализации Compare – в данной задаче сортировка производилась по убыванию количества вхождений символов в исходное сообщение) элементу, за счет увеличения скорости вставки элементов в контейнер до логарифмической. Работа с priority_queue похоже на работу с кучей в контейнерах случайного доступа, но имеет преимущество в виде невозможности случайного повреждения кучи.

Используется для построения дерева алгоритмом Хаффмана.

std::vector

std::vector — последовательный контейнер, инкапсулирующий массивы переменного размера.

Используется для хранения последовательности некоторых объектов.

Описание плана эксперимента

В ходе эксперимента, который проводился на всех 36 тестируемых файлах, использовались следующие алгоритмы сжатия без потерь:

- алгоритм Хаффмана
- алгоритм Шеннона-Фано
- алгоритм LZ77 с размером окна 5 Кб и размером словаря 4 Кб
- алгоритм LZ77 с размером окна 10 Кб и размером словаря 8 Кб
- алгоритм LZ77 с размером окна 20 Кб и размером словаря 16 Кб

Эксперименты проводились последовательно на каждом файле. Для каждого файла применялись следующие шаги:

1) записываются частоты встречаемости символов в исходном файле;
2) вычисляется энтропия исходного файла,
3) для каждого алгоритма вычисляется среднее время (за 20 раз) упаковки и распаковки,
4) для каждого алгоритма вычисляется коэффициент сжатия файла,
5) полученные данные записываются в таблицу.

Для запуска программы необходимо добавить в проект все исходные файлы и запустить функцию main в файле main.cpp.

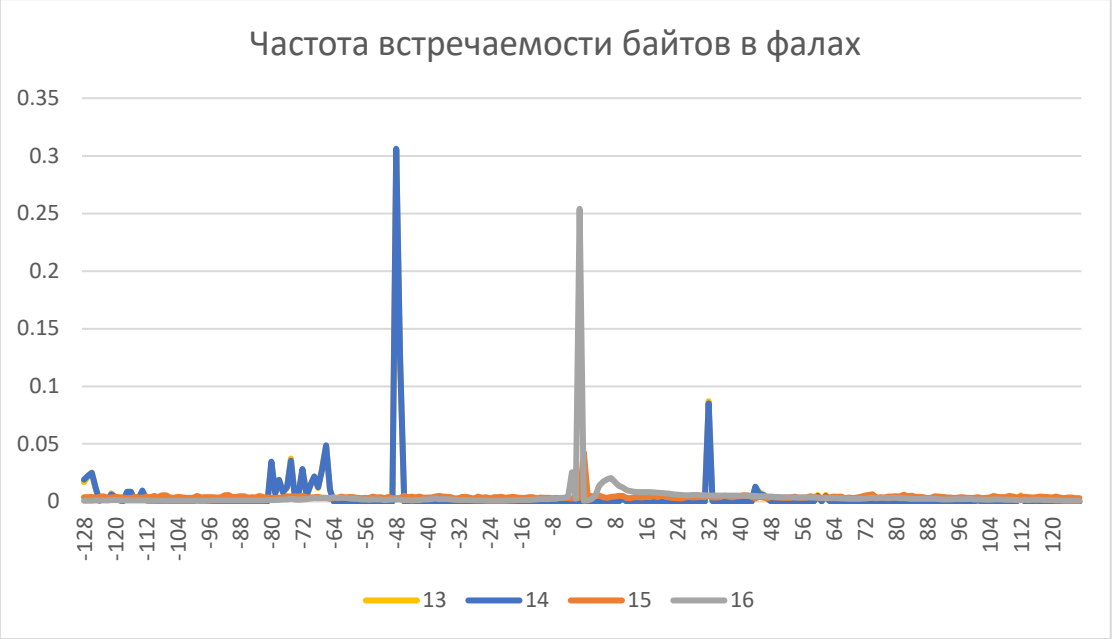
Результаты экспериментов

		Huffman			Shannon-Fano			LZ77_5			LZ77_10			LZ77_20		
File	H	K	tp	tu	K	tp	tu	K	tp	tu	K	tp	tu	K	tp	tu
1	0,36	1,00	1152064950	10929892650	1,00	1118977350	10723028700	2,37	8172602150	74578650	2,30	15804465800	77707350	2,20	30486611200	76202100
2	0,21	0,58	627589850	4161702150	0,58	576031850	3944496350	0,82	5824997650	65674250	0,74	10696521100	63165950	0,68	18561381750	63668600
3	0,44	0,96	38619100	312816300	0,97	40628300	329912150	2,01	209057850	7018700	1,99	371989450	8540150	2,06	641206100	7019550
4	0,44	1,00	98260100	650732950	1,01	73206050	641631450	2,40	506363950	19837050	2,36	889349200	9558450	2,33	1643856300	9795250
5	0,26	0,57	24903500	139285650	0,57	26554900	138491850	1,04	213060000	11565650	1,07	360312200	11525350	1,24	605109200	11029600
6	0,23	0,58	207516150	1286423500	0,58	202163200	1172125350	1,04	2013166300	19632700	0,96	3633684350	20075550	0,91	6544489150	18040600
7	0,39	0,97	188850100	1722079550	0,97	188618150	1729619250	1,96	1161106500	14136100	1,91	2187828600	14530650	1,87	4106512350	14354300
8	0,36	1,00	1263363250	10772967050	1,00	1329172100	12718314050	2,38	9568957850	79211000	2,30	18547343850	88234950	2,19	35679408850	79705200
9	0,31	1,00	11800034800	102192000000	1,00	13332750700	125333000000	2,38	140425000000	2491194000	2,30	255254000000	2276352300	2,18	356108000000	2412555400
10	0,39	0,89	66342900	729741000	0,89	67366000	504153200	1,48	287834200	17979800	1,49	532415600	11028500	1,53	965574800	12025600
11	0,37	0,96	366892700	3005015900	0,96	314836300	2887761800	1,97	2015708200	22447400	1,89	3745968500	21262900	1,81	7000616100	24064000
12	0,28	0,61	26809200	207814200	0,63	27236800	203362300	0,57	117179800	8156600	0,66	190513200	7069500	0,87	358953700	8021300
13	0,19	0,52	446186200	2799774100	0,52	365761500	2587402400	0,67	5184793300	81345300	0,61	9348023300	79146400	0,55	16277303700	79212400
14	0,19	0,52	418073100	2859077800	0,52	397057200	2969902500	0,68	5872825900	103072100	0,61	10413989100	87951000	0,56	18608708800	90037000
15	0,45	1,00	47126200	389034500	1,00	50188400	436169800	2,41	301276800	7018300	2,45	550577500	6906100	2,47	1000691600	7992800
16	0,30	0,80	465204500	4229726500	0,80	451200800	3845806800	0,92	2068669500	28910100	0,87	3779041900	29079100	0,82	6870277800	27073700
17	0,36	0,95	492258200	3924486700	0,95	403653100	3763141300	0,85	1463895500	28442400	0,79	2509699400	25477600	0,73	4470915000	26151200
18	0,30	0,80	541440500	4652076100	0,80	516374200	4459864700	1,21	2988990700	35053700	1,13	5452504700	34091100	1,06	9774478300	34824200
19	0,43	1,00	92242800	855313800	1,00	96257800	818206200	2,42	619645100	8029900	2,37	1164905900	7016100	2,32	2233963700	8106300
20	0,47	1,01	41766400	325210300	1,01	43435100	328556200	2,47	247867500	7997900	2,48	550926100	8430900	2,53	814164300	8022700
21	0,47	1,01	42078700	330880700	1,01	44136600	327042800	2,47	248498100	11115100	2,47	456211200	12089600	2,52	868370200	10029200
22	0,39	1,00	337474000	3146773900	1,00	378090500	3056733500	2,41	2374264200	20998400	2,33	4514510500	19550300	2,23	8476552100	21055200
23	0,36	0,94	477268100	4130142400	0,95	455773800	3911120600	1,53	2643775200	31342100	1,43	4848900500	34090700	1,33	8775876000	29030000
24	0,42	0,98	94249900	837227400	0,98	101298100	827199400	2,27	598595100	10023700	2,21	1125993900	9025300	2,15	2145673100	10031800
25	0,00	0,13	77250200	387695200	0,13	107735500	342374300	0,01	6811119700	16092100	0,02	8742942500	18383300	0,04	9000023500	13974600
26	0,70	1,53	6015600	8021700	1,53	7019100	8021400	5,00	11028100	6016800	5,00	5472800	2998700	5,00	6992200	3008800
27	0,04	0,20	4980500	4042600	0,20	6056900	5975900	4,40	11066400	6985000	5,00	9024400	3008500	5,00	10046700	3190700
28	0,31	0,58	39022600	6013400	0,61	6783600	6022400	5,00	6010000	4075600	5,00	5952400	3328400	5,00	6728100	3228200
29	0,28	0,89	5816836900	49890950300	0,89	5497530900	49521471600	0,62	19982237000	648084600	0,55	34651188100	795893200	0,39	39576390100	794142500
30	0,30	0,95	6755157500	56814593100	0,95	6320986300	56659413500	0,97	23002353000	1098040400	0,88	40308126100	868290400	0,72	65652673500	1169594600
31	0,26	0,82	8652359800	72577612800	0,82	7749859600	63805764100	1,46	52371852200	1613709800	1,39	92310433200	1718215000	1,32	168703000000	1415410700
32	0,23	0,77	35499717600	288612000000	0,78	34553107300	272878000000	1,28	364739045900	5603889100	1,19	662995196169	5120607163	1,08	924952765940	5426993205
33	0,28	0,81	1587285000	15007884000	0,82	1557636300	14731847500	1,26	10375724300	106880100	1,07	17388073300	98770500	1,00	31905070000	97638900
34	0,27	0,82	3962897500	36971095500	0,83	3969550100	39098608700	1,20	28348818200	451022900	1,12	53692983600	383366800	1,06	92917150700	465372500
35	0,26	0,73	707660100	6389413900	0,73	716753300	6521428900	1,02	7150046000	101857800	0,95	13604050000	96797100	0,91	25876774700	100706200
36	0,29	0,75	243278700	2160347900	0,75	232511800	2136428900	1,04	1470860400	55832400	1,01	2681699500	41702000	0,99	4676767300	41022300

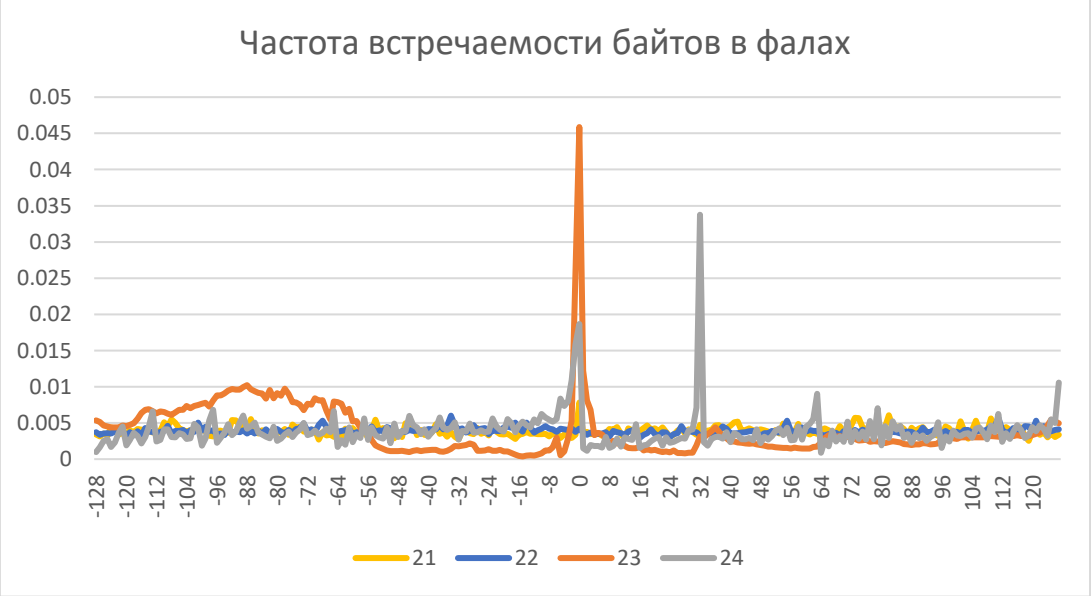

Частота встречаемости байтов в фалах
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
1
2
3
4

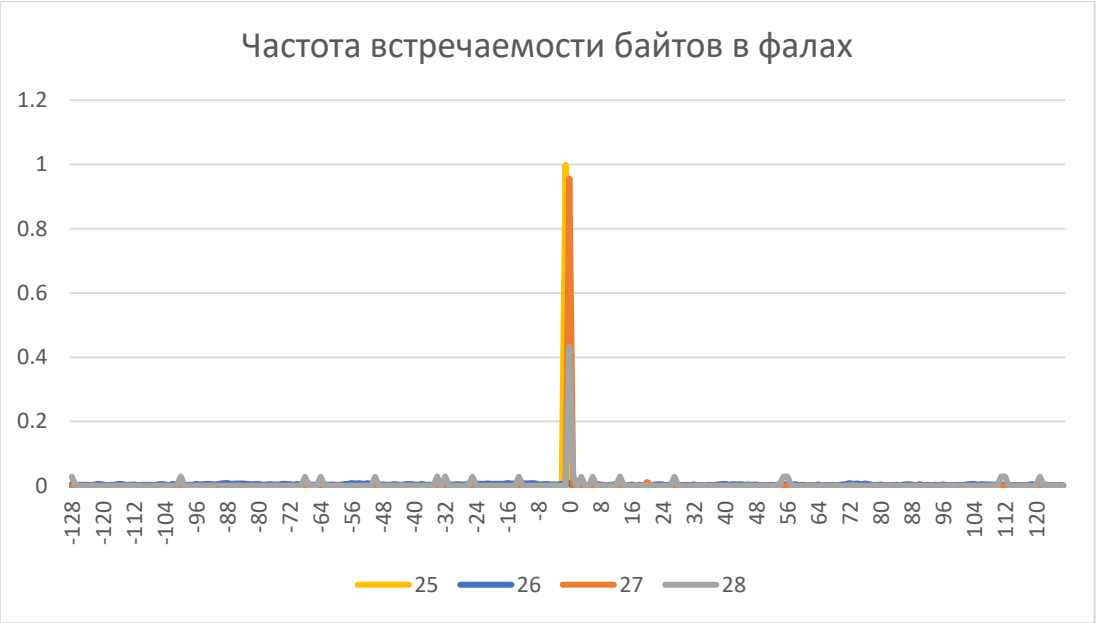

Частота встречаемости байтов в фалах
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
5
6
7
8


Частота встречаемости байтов в фалах
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
9
10
11
12

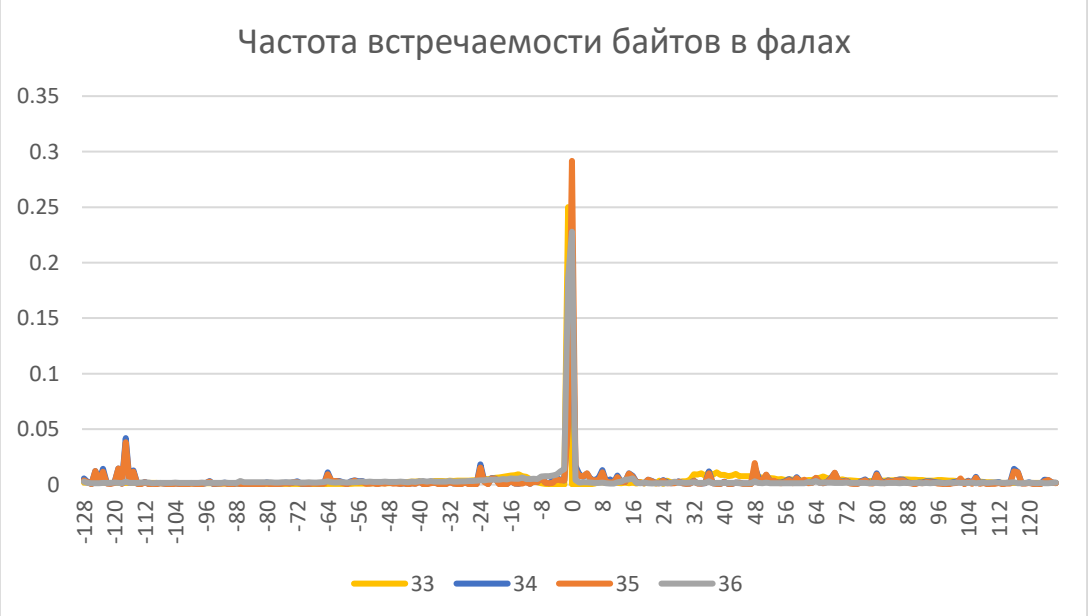

Частота встречаемости байтов в фалах
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
13
14
15
16


Частота встречаемости байтов в фалах
0.3
0.25
0.2
0.15
0.1
0.05
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
17
18
19
20


Частота встречаемости байтов в фалах
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
21
22
23
24

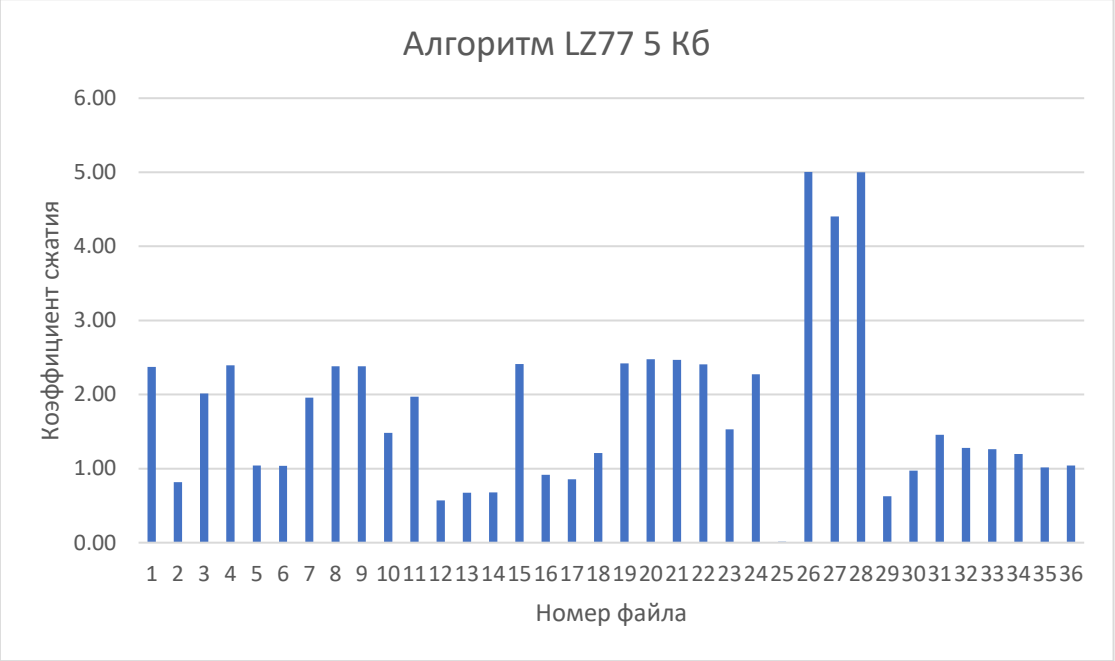

Частота встречаемости байтов в фалах
1.2
1
0.8
0.6
0.4
0.2
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
25
26
27
28


Частота встречаемости байтов в фалах
0.3
0.25
0.2
0.15
0.1
0.05
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
29
30
31
32

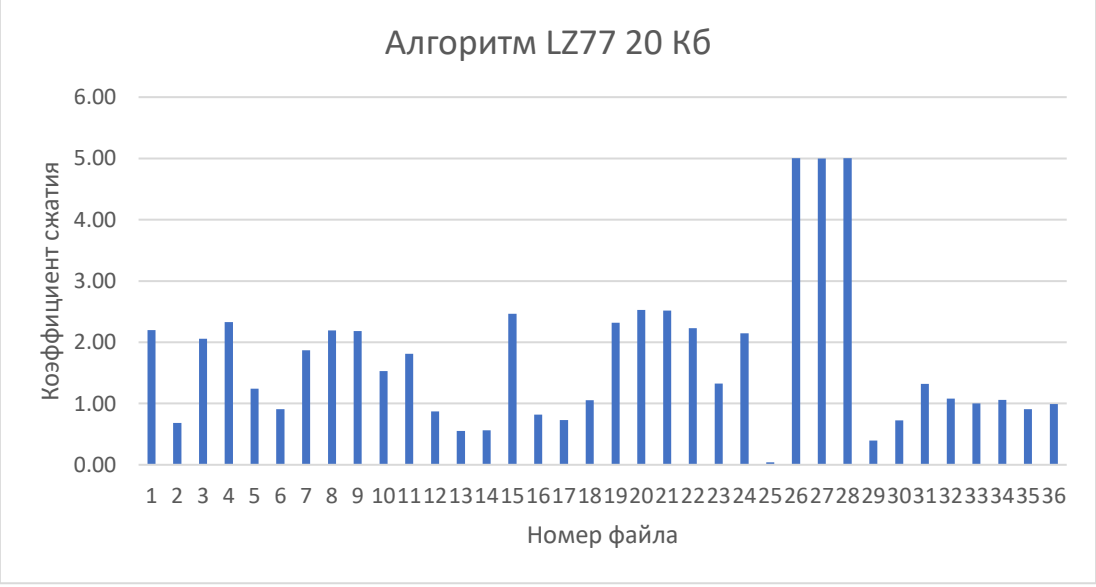

Частота встречаемости байтов в фалах
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-128 -120 -112 -104 -96 -88 -80 -72 -64 -56 -48 -40 -32 -24 -16 -8 0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
33
34
35
36


Алгоритм Хаффмана
Коэффициент сжатия
1.80
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла


Алгоритм Шеннона-Фано
Коэффициент сжатия
1.80
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла

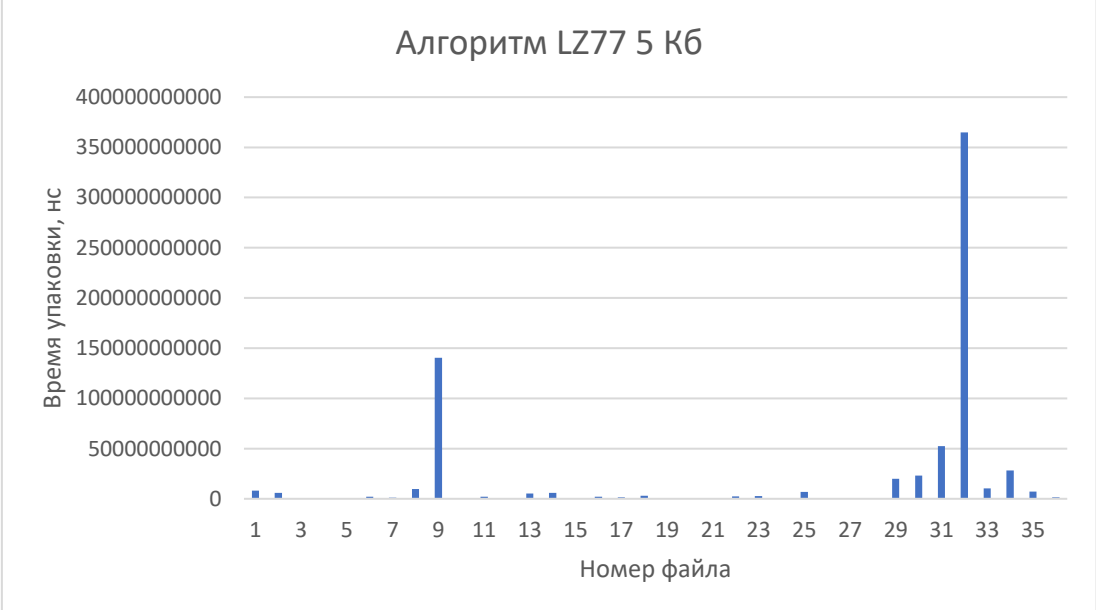

Алгоритм LZ77 5 Кб
Коэффициент сжатия
6.00
5.00
4.00
3.00
2.00
1.00
0.00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла

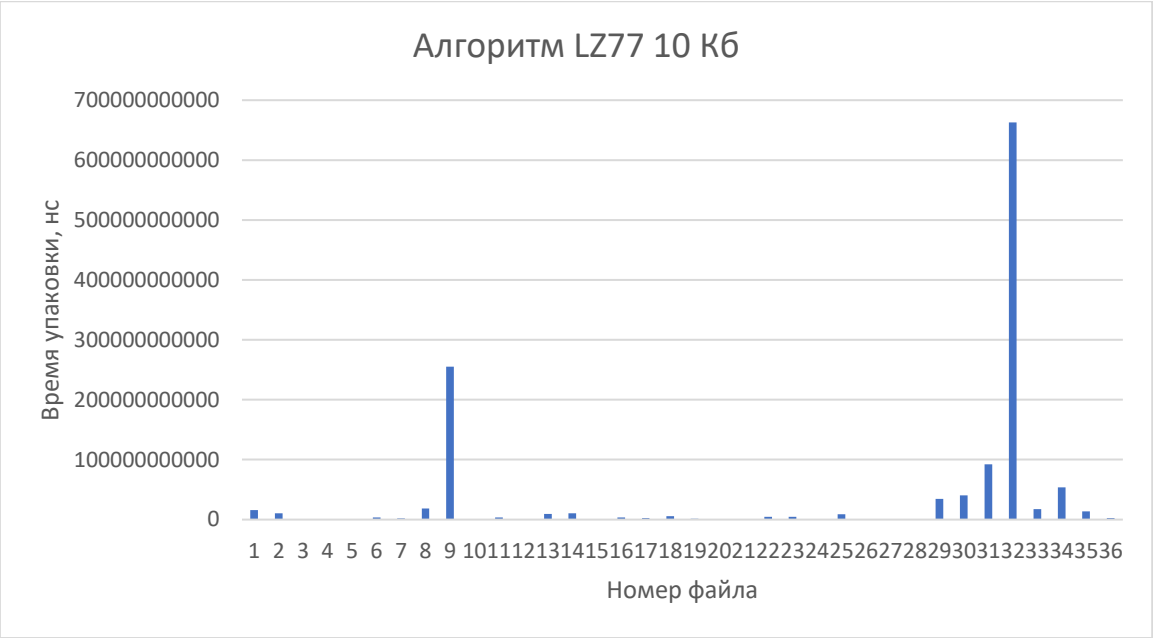

Алгоритм LZ77 10 Кб
Коэффициент сжатия
6.00
5.00
4.00
3.00
2.00
1.00
0.00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла

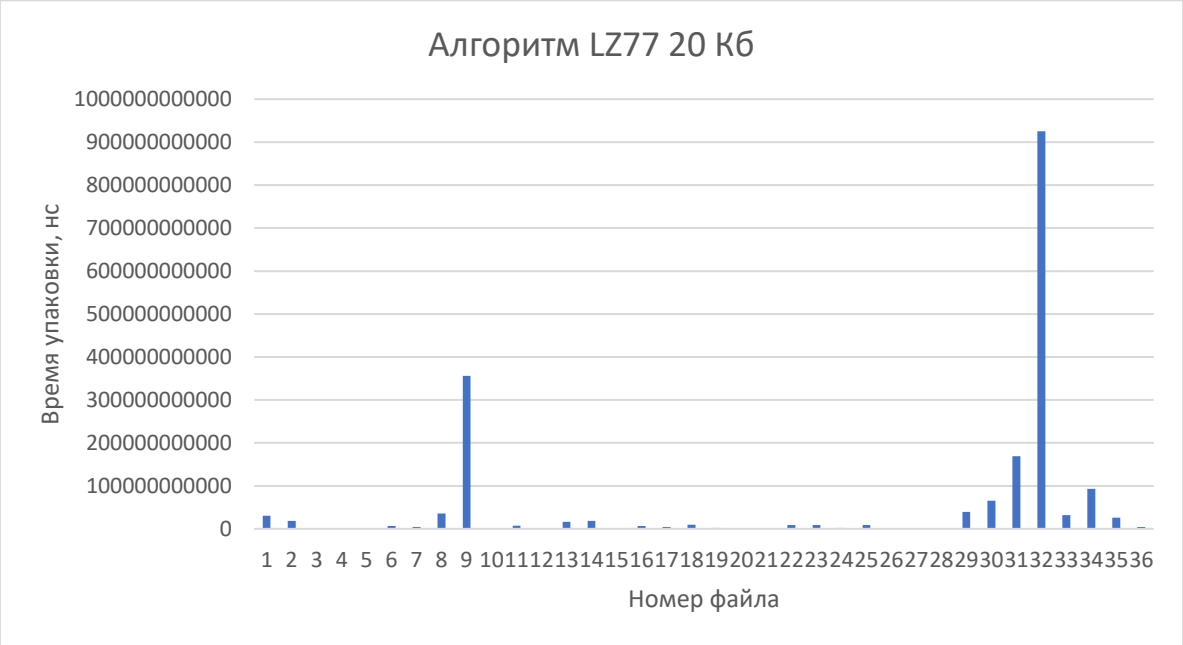

Алгоритм LZ77 20 Кб
Коэффициент сжатия
6.00
5.00
4.00
3.00
2.00
1.00
0.00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла


Алгоритм Хаффмана
40000000000
35000000000
30000000000
25000000000
20000000000
15000000000
10000000000
5000000000
0
Время упаковки, нс
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
Номер файла


Алгоритм Шеннона-Фано
40000000000
35000000000
30000000000
25000000000
20000000000
15000000000
10000000000
5000000000
0
Время упаковки, нс
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
Номер файла

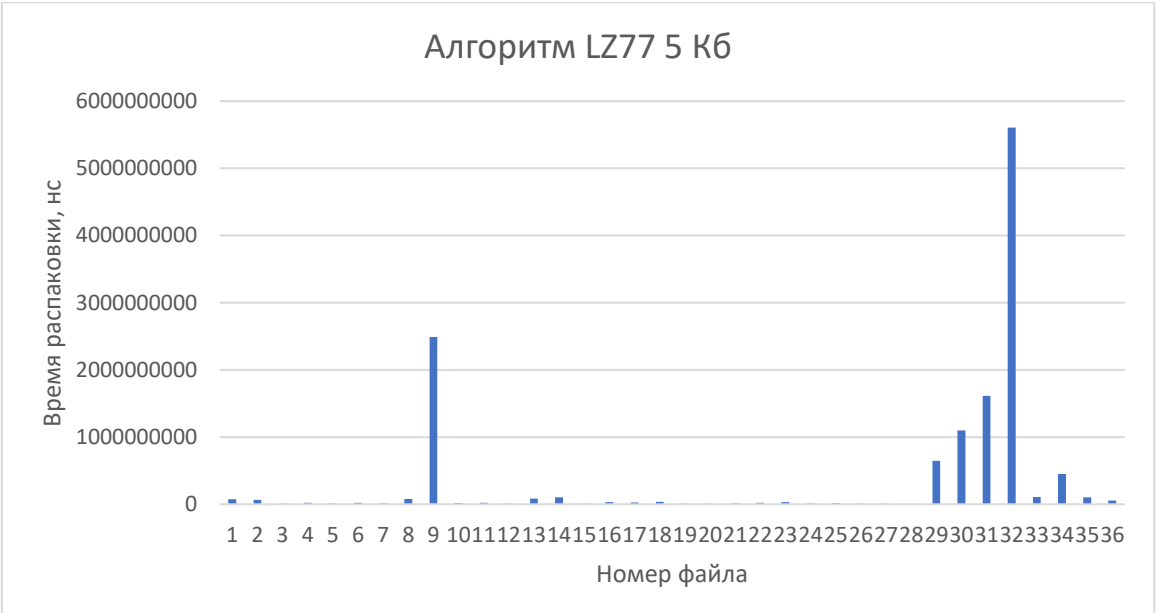

Алгоритм LZ77 5 Кб
400000000000
350000000000
300000000000
250000000000
200000000000
150000000000
100000000000
50000000000
0
Время упаковки, нс
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
Номер файла

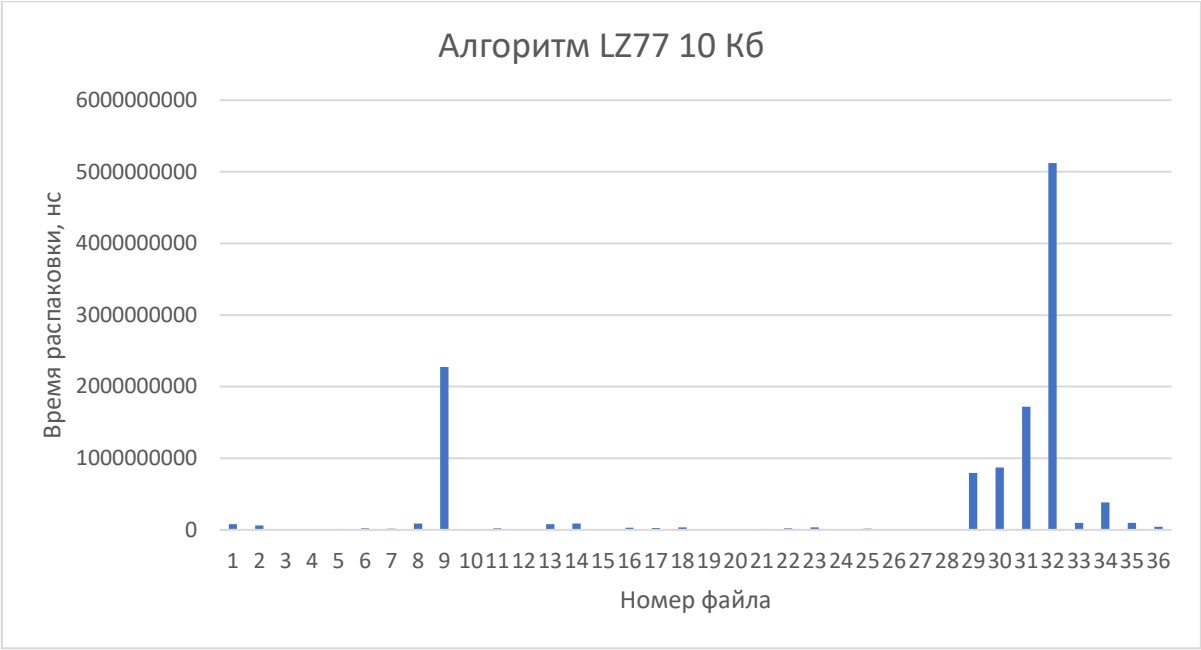

Алгоритм LZ77 10 Кб
700000000000
600000000000
500000000000
400000000000
300000000000
200000000000
100000000000
0
Время упаковки, нс
1 2 3 4 5 6 7 8 9 10111213141516171819202122232425262728293031323334353 6
Номер файла

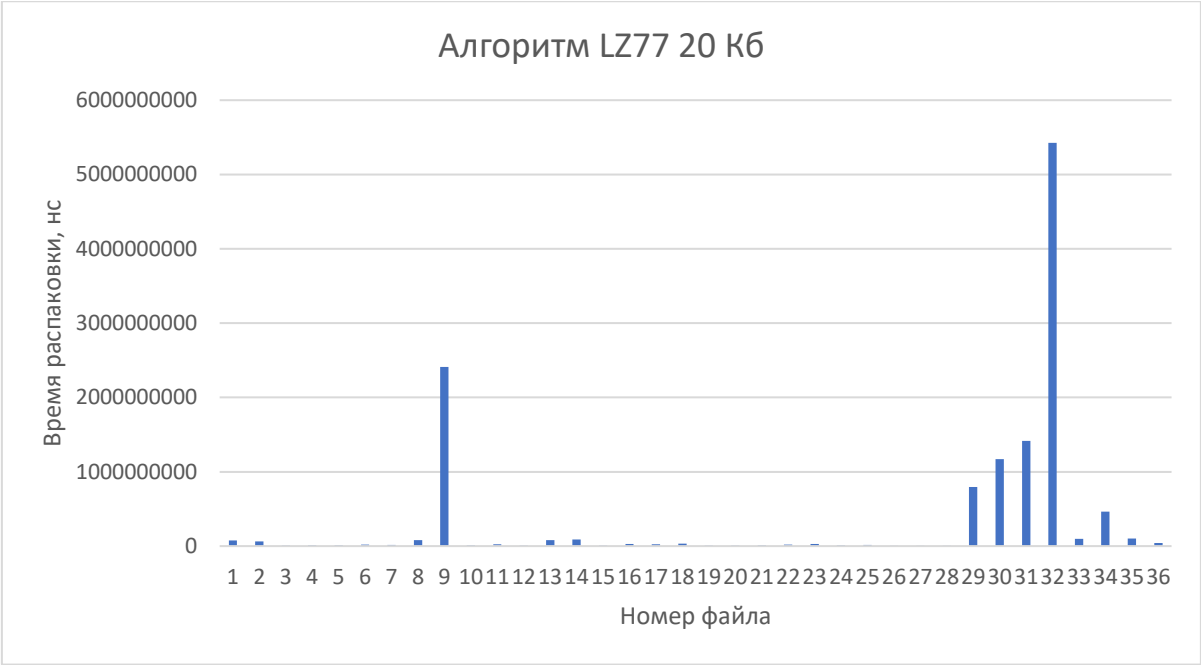

Алгоритм LZ77 20 Кб
1000000000000
900000000000
800000000000
700000000000
600000000000
500000000000
400000000000
300000000000
200000000000
100000000000
0
Время упаковки, нс
1 2 3 4 5 6 7 8 9 10111213141516171819202122232425262728293031323334353 6
Номер файла


Алгоритм Хаффмана
350000000000
300000000000
250000000000
200000000000
150000000000
100000000000
50000000000
0
Время распаковки, нс
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла


Алгоритм Шеннона-Фано
300000000000
250000000000
200000000000
150000000000
100000000000
50000000000
0
Время распаковки, нс
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла


Алгоритм LZ77 5 Кб
6000000000
5000000000
4000000000
3000000000
2000000000
1000000000
0
Время распаковки, нс
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла


Алгоритм LZ77 10 Кб
Время распаковки, нс
6000000000
5000000000
4000000000
3000000000
2000000000
1000000000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла


Алгоритм LZ77 20 Кб
Время распаковки, нс
6000000000
5000000000
4000000000
3000000000
2000000000
1000000000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Номер файла

Сравнительный анализ алгоритмов

Алгоритмы Шенонна-Фано и Хаффмана - довольно схожие алгоритмы, явное их отличие в механизме построения кодового дерева, однако все равно коды для одного и того же алфавита не сильно отличаются. Именно поэтому показатели кодирования и декодирования файлов для этих двух алгоритмов практически одинаковые.

Кроме того, алгоритмы Хаффмана и Шеннона-Фано очень чувствительны к частоте встречаемости символов в сообщении. Чем выше его энтропия, то есть все символы в файле почти равновероятно встречаемы, тем хуже происходит сжатие, так как длина кодов этих символов получается примерно одинаковой и почти максимальной, следовательно, сжатие получается неэффективным.

В то же время можно заметить, что алгоритм LZ77 не зависит напрямую от энтропии сжимаемого сообщения. На хорошее сжатие файла этим алгоритмом влияет повторение подстрок.

В алгоритме кодирования LZ77 большую роль играет размер скользящего окна, особенно это заметно на сжатии файлов больших размеров. При маленьком размере окна длина подстроки, которую можно закодировать, и расстояние между подстроками, которые можно закодировать, уменьшаются. Однако при большем размере окна увеличивается время поиска совпадающей подстроки в буфере истории. Из всего этого следует то, что при увеличении размера окна уменьшается коэффициент сжатия, то есть файлы сжимаются лучше, а время кодирования, наоборот, увеличивается. Время же распаковки файла существенно не отличается.

Также важной особенностью кодирования файлов является то, что некоторые файлы после сжатия больше по размеру исходных. Это связано с тем, что в начале файлов записана таблица частот встречаемости символов. Если файл занимает немного памяти и немного символов в нем повторяется, то таблица получается большой. Аналогично с записью закодированных троек в LZ77, при маленьком количестве повторяющихся символов одна тройка занимает место целого символа.

Заключение

1. Алгоритмы Хаффмана и Шеннона-Фано очень эффективно сжимают файл, когда в исходном сообщении некоторые символы встречаются особенно часто.
2. Алгоритм LZ77 эффективно сжимает файл, когда в исходном сообщении много повторяющихся подстрок.
3. Чем больше размер скользящего окна в алгоритме LZ77, тем меньше коэффициент сжатия и тем больше тратится времени на кодирование.

Источники

1. Код Хаффмана [Электронный ресурс]: https://ru.wikipedia.org/wiki/Код_Хаффмана

2. Алгоритм Шеннона — Фано [Электронный ресурс]: https://ru.wikipedia.org/wiki/Алгоритм_Шеннона_—_Фано

3. Алгоритм LZ77 [Электронный ресурс]: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B_LZ77_%D0%B8_LZ78

4. std::map [Электронный ресурс]: http://ru.cppreference.com/w/cpp/container/map

5. std::priority_queue [Электронный ресурс]: http://ru.cppreference.com/w/cpp/container/priority_queue

6. std::vector [Электронный ресурс]: http://ru.cppreference.com/w/cpp/container/vector