МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра многопроцессорных систем и сетей

БАРСКИЙ АНТОН ЮРЬЕВИЧ

ОЦЕНКИ ПО ПАМЯТИ И БЫСТРОДЕЙСТВИЮ ПРОЦЕССОВ ИЗМЕНЕНИЯ ФАЙЛОВ В РЕЗУЛЬТАТЕ ИХ СЖАТИЯ И ВОССТАНОВЛЕНИЯ РАЗЛИЧНЫМИ МЕТОДАМИ С ПОТЕРЯМИ И БЕЗ ПОТЕРЬ

Задание по архитектуре компьютеров
студента 3 курса 3 группы

Преподаватель
Буза Михаил Константинович
Профессор кафедры МСС

Минск, 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Для хранения и передачи данных используются различные алгоритмы сжатия, позволяющие сократить размер требуемой для хранения данных памяти, а также позволяющие ускорить передачу данных по сети.

Сжатие данных основано на устранении избыточности, содержащейся в исходных данных. Пример такой избыточности - часто повторяющиеся слова в тексте, повторяющиеся части кадров. Сжатие достигается с помощью ссылок на уже закодированные части данных.

Также есть такой вид сжатия, как энтропийное кодирование - кодирование, использующее знания о том, что какие-то данные встречаются чаще других. В таком случае, часто встречающиеся данные кодируются короткими кодовыми словами, а редко встречающиеся - длинными.

Сжатие данных, не обладающих свойством избыточности (например, случайный сигнал или белый шум, зашифрованные сообщения), принципиально невозможно без потерь.

Для сжатия данных используются некоторые сведения о том, какого рода данные сжимаются. Не обладая такими сведениями об источнике, невозможно сделать никаких предположений о преобразовании, которое позволило бы уменьшить объём сообщения. Модель избыточности может быть неизменной для всего сжимаемого сообщения, либо строиться на этапе сжатия (и восстановления). Методы, позволяющие на основе входных данных изменять модель избыточности информации, называются адаптивными. Неадаптивными являются обычно узкоспециализированные алгоритмы, применяемые для работы с данными, обладающими хорошо определёнными и неизменными характеристиками. Подавляющая часть достаточно универсальных алгоритмов являются в той или иной мере адаптивными.

Все методы сжатия данных делятся на два основных класса:

- Сжатие без потерь
- Сжатие с потерями

В этой работе будут рассмотрены и сравнены алгоритмы обоих видов сжатия.

ГЛАВА 1
АЛГОРИТМЫ СЖАТИЯ БЕЗ ПОТЕРЬ

Сжатие без потерь позволяет полностью восстановить исходное сообщение, так как не уменьшает в нем количество информации, несмотря на уменьшение длины. Такая возможность возникает только если распределение вероятностей на множестве сообщений неравномерное, например часть теоретически возможных в прежней кодировке сообщений на практике не встречается. Как уже описывалось в введении, данные алгоритмы используют кодирование с помощью словарей и ссылок на уже закодированные части данных. Историю развития данных алгоритмов можно показать в виде дерева (*Приложение 1*).

Все стандартные алгоритмы сжатия данных без потерь можно протестировать используя проект с открытым кодом: *lzbench*.

Для тестирования будем использовать стандартный набор файлов для тестирования алгоритмов сжатия: *Silesia compression corpus.*

Некоторые элементы из данного набора:

- собрание сочинений Чарльза Диккенса
- медицинское магнитно-резонансное изображение
- пример базы данных в формате MySQL из Open Source Database Benchmark
- рентгенологическая картина

После запуска *lzbench* на *Silesia compression corpus*, получили следующие результаты (указаны: наименование алгоритма, скорость сжатия, скорость восстановления, размер сжатого файла, размер сжатого файла в процентах относительно размера исходного) (*Приложение 2)*

Как можно заметить из результатов, алгоритмы делятся на: быстрые на этапе сжатия; быстрые на этапе восстановления; медленные, но с хорошим ratio.

Как уже оговорено выше, алгоритмы сжатия используются в большинстве случаев либо для хранения, либо для передачи данных.

Для хранения, как несложно понять, стоит использовать алгоритмы с наилучшим ratio. Среди таких алгоритмов - *zstd, bzip2, zlib.*

Для того, чтобы выяснить, какие алгоритмы лучше всего использовать для передачи по сети, требуется анализировать все три параметра, так как передача по сети включает в себя:

- сжатие данных для передачи
- передача сжатых данных по сети
- восстановление принятых по сети данных

А значит наилучшим алгоритмом будем считать тот, который выдает наименьшее значение выражения:
compression_time+(compressed_size/network_speed)+decompression_time

Для конкретного файла переменные *compression_time*, *compressed_size*, *decompression_time* - фиксированные. *network_speed* может меняться даже во время передачи файла по сети, поэтому требуется провести исследование времени передачи файла в зависимости от скорости сети для каждого алгоритма.

Для удобства будем рассматривать не все алгоритмы. Разобьем все алгоритмы на классы и из каждого класса рассмотрим представителя. Результаты покажем на графике (*Приложение 3*).

По графику видно, что LZ4 остаётся лидером на скоростях выше 50 Мб/сек, а ZSTD демонстрирует лучшие результаты на скоростях от 0.5 Мб/сек до 50 Мб/сек.

Однако, нельзя делать поспешных выводов, проанализировав алгоритмы на одной машине: в зависимости от архитектуры процессора, количества ядер, размера оперативной памяти, результаты скорости работы алгоритмов могут различаться. Поэтому перед тем, как начинать использовать тот или иной алгоритм в своем продукте стоит протестировать его на соответствующей машине. Однако это может стать проблемой: протестировать все алгоритмы, на всех доступных компьютерах. Для того, чтобы сравнить различные форматы сжатия данных без потерь на разных типах данных при выполнении на различных процессорах, можно использовать следующий онлайн инструмент: [2]

Таким образом, для хранения данных на внешнем носителе, эффективнее всего будет использование алгоритмов: ZLIB, ZSTD, BZIP2. Для передачи данных по мобильной сети: ZSTD. Для передачи данных по локальной сети: LZ4.

ГЛАВА 2
АЛГОРИТМЫ СЖАТИЯ С ПОТЕРЯМИ

Сжатие с потерями позволяет достичь степени сжатия, недоступной для алгоритмов без потерь, однако его целесообразно использовать только в случаях, когда незначительное искажение данных не принципиально. Такие алгоритмы чаще всего применяются при сжатии изображений, звука, видео.

В данной работе рассмотрим самые популярные из алгоритмов сжатия изображений. Таковыми являются PNG, JPEG, GIF, TIFF.

При рассматривании алгоритмов сжатия с потерями, проблемой становится оценка качества сжатия. Чаще всего при оценках ошибок используют значение среднеквадратичной ошибки. Однако в случае изображений эта оценка не всегда дает приемлемые результаты (так, например, к изображению можно добавить незначительный шум, невидимый для человеческого глаза, но оценка выдаст результат, информирующий о том, что эти изображения сильно отличаются). Единственным надежным способом проверки качества сжатия остается проверка тестируемых изображений своими глазами.

Также, почти все алгоритмы сжатия с потерями дают возможность настроить степень сжатия. Таким образом, с любым алгоритмом сжатия можно добиться практически безграничного уменьшения размера сжатого файла, однако, естественно, с соответствующим ухудшением качества.

Итак, рассмотрим плюсы и минусы каждого алгоритма сжатия:

JPEG:

Плюсы:

- малый размер сжатого файла
- поддерживает 24-битные цвета

Минусы:

- быстрая потеря качества после нескольких сохранений.
- не поддерживает анимацию

GIF:

Плюсы:

- поддерживает анимацию
- поддерживает альфа канал (1 бит)
- поддерживает сжатие без потерь

Минусы:

- поддерживает только 256 цветов
- имеет большой размер, относительно конкурентов

TIFF:

Плюсы:

- предоставляет высокое качество изображений
- поддерживает разноуровневые изображения

Минусы:

- большой размер сжатого файла
- много времени на восстановление
- много времени на передачу по сети

PNG:

Плюсы:

- без потерь
- поддерживает альфа канал (1 байт)
- сжатые файлы меньше по размеру, чем GIF

Минусы:

- не поддерживает анимацию
- не эффективен для сжатия больших файлов (порой после сжатия PNG, они становятся только больше)

Таким образом, при выборе типа сжатия изображения, стоит опираться на то, какую цель Вы преследуете. Если требуется хранить полноцветные фотографии в наименьшем размере - JPEG. Если нужно обеспечить высокое качество изображений, загружаемых в Интернет - PNG. Если нужен логотип бренда или другое изображение, которое требует прозрачности - PNG или GIF. Небольшая анимация - GIF.

Так как хранение и передача анимаций - не так часто возникающая проблема, как передача обычных фото, либо скриншотов, рассмотрим подробнее алгоритмы PNG и JPEG.

PNG: Итак, рассмотрим первый вариант. Алгоритмы, опирающиеся на наличие повторяющихся фрагментов - словарные алгоритмы (LZ*). Один из таких - LZ77, который в совокупности с алгоритмом Хаффмана используется в алгоритме сжатия Deflate. Deflate в свою же очередь

используется в PNG. Таким образом PNG будет эффективно сжимать повторяющиеся фрагменты, что приемлемо при передаче изображений скриншотов окон, панелей, изображений с небольшой цветовой гаммой, схем и других искусственных изображений. Чаще всего (к примеру, в web) для достижения приемлемого качества достаточно 8-битного цвета, для этого можно использовать PNG-8.
Формат PNG-8 применяется при верстке сайтов, при передаче графики по сети, передаче больших изображений, таких как скриншоты, в целом, в случаях когда нужно сильно сжать изображение, не теряя его качество.[4]

JPEG: Любой источник скажет, что JPEG предназначен для хранения изображений окружающего мира, фотоизображений, причем один из самых популярных в своем роде.
Ему на замену приходит JPEG2000, предоставляющий возможность сохранения изображений так, что изображение получается более гладким и чётким, а размер файла по сравнению с JPEG при одинаковом качестве оказывается меньшим. Однако из недостатков - сложность реализации и требовательность к использованию памяти. На смену ему и как новый стандарт сжатия изображений приходит формат JPEG XS, который позиционируется как легковесная система кодирования изображений, обеспечивающая минимальные задержки при кодировании и декодировании, и ориентированная на оптимизацию передачи последовательностей изображений очень высокого качества (до 8K). *«Впервые за всю историю кодирования изображений мы сжимаем меньше, чтобы сохранить качество, и мы делаем процесс быстрее, используя меньше энергии,* — объяснил глава JPEG Турадж Ибрахими — *Идея в том, чтобы использовать меньше ресурсов, и использовать их более обдуманно. Это реальный сдвиг парадигмы» [5]*. Таким образом, JPEG XS отлично подходит для быстрой передачи по высокоскоростным сетям.
Можно сказать, что для передачи рисунков и фото по сети при низкой скорости интернета целесообразно использовать JPEG2000, при высокой (например, 5G, локальной) - JPEG XS.
Итак, подытожим. PNG хорошо работает с изображениями, состоящими из линий на однородном фоне. Для обычных фотографий, где можно

допустить частичные потери, лучше подходит JPEG. Для медицинских изображений, где потери недопустимы, подойдет JPEG 2000.

Так как не всегда можно заранее узнать, какой формат изображения будет сжиматься (естественный снимок, двухцветная фотография, изображение рабочего стола), требуется формат, который будет давать хорошие результаты на любых типах изображений.
Такой формат - FLIF, уже набирает популярность. Этот алгоритм использует арифметическое кодирование. Таким образом, уже первые несколько байт дают приемлемое представление изображения.[7] Также FLIF обладает полезным свойством - он не теряет качества при пересохранении изображений.[6] Из недостатков FLIF выделяют только то, что он пока не поддерживается браузерами и редакторами изображений. Результаты работы FLIF - в приложениях.
Как уже оговаривалось, пока не существует числового критерия хорошего качества сжатия. Для тестирования работы того или иного формата сжатия удобно использовать следующий онлайн ресурс: [8]

Для сжатия видео чаще всего используются форматы MJPEG и H264.

MJPEG - формат сжатия, концепция которого основана на выборе опорных кадров, а для остальных кадров передаются только отличия их от ближайшего опорного кадра.

H264 - последователь MJPEG, однако в нем для описания следующего кадра используется не один предыдущий кадр, а целый набор предыдущих кадров. Что позволяет эффективно сжимать циклические действия, а также движение объектов на статическом фоне. Однако, так как этот алгоритм использует много информации о предыдущих кадрах, он требователен к памяти и к характеристикам процессора. Но дает намного лучшие показатели сжатия относительно MJPEG.
При рассмотрении вопроса качества, сталкиваемся с той же проблемой, как и при работе с изображениями. Поэтому опять же предлагаю онлайн ресурс, для наглядного тестирования качества сжатия видео кодеков: [9]

Таким образом, при работе с видео, если сжатие происходит на маломощной машине, стоит безоговорочно использовать MJPEG. Если же

процессор достаточно “мощный”, стоит рассматривать вариант использования H264. Особенно нужно обратить внимание на H264, если сжатие требуется для передачи видео по сети, так как H264 сжимает видео намного лучше, чем MJPEG.

ЗАКЛЮЧЕНИЕ

Таким образом, в данной работе были рассмотрены основные алгоритмы сжатия без потерь и сжатия с потерями.
Были сделаны следующие выводы:

При использовании сжатия без потерь:
Для хранения данных на внешнем носителе, эффективнее всего будет использование алгоритмов: ZLIB, ZSTD, BZIP2. Для передачи данных по мобильной сети: ZSTD. Для передачи данных по локальной сети: LZ4.

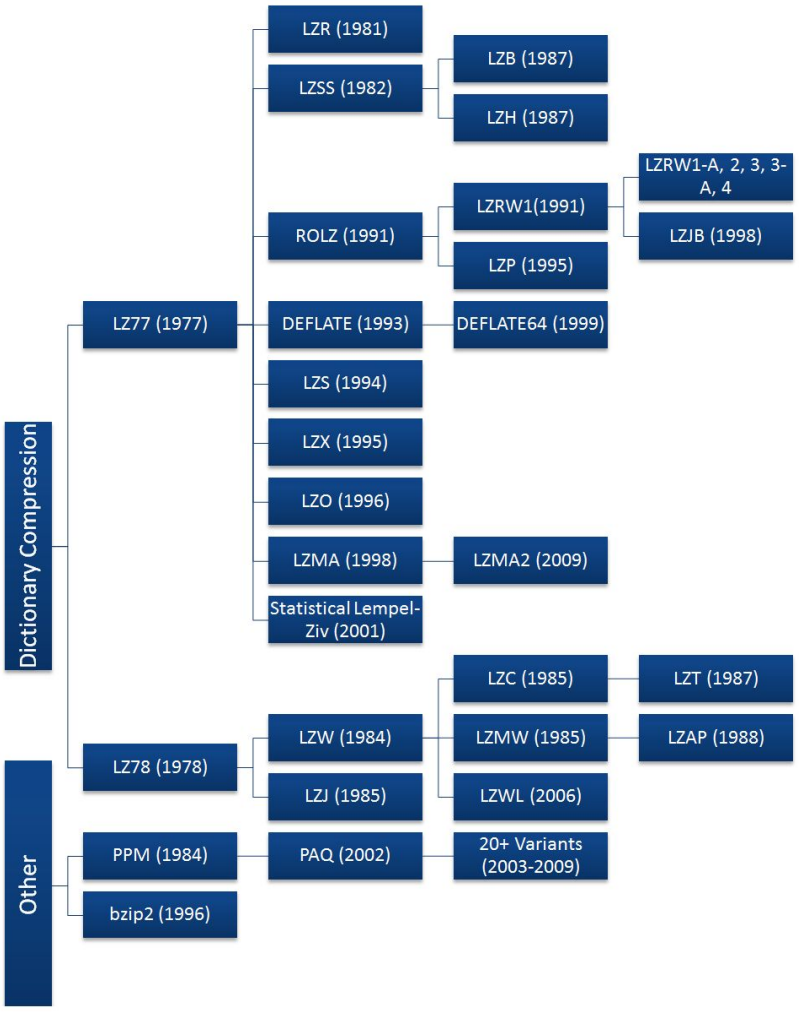
При сжатии изображений:
PNG хорошо работает с изображениями, состоящими из линий на однородном фоне. Для обычных фотографий, где можно допустить частичные потери, лучше подходит JPEG. Для медицинских изображений, где потери недопустимы, подойдет JPEG2000.

При сжатии видео:
Если сжатие происходит на маломощной машине, стоит безоговорочно использовать MJPEG. Если же процессор достаточно “мощный”, стоит рассматривать вариант использования H264. Особенно нужно обратить внимание на H264, если сжатие требуется для передачи видео по сети, так как H264 сжимает видео намного лучше, чем MJPEG.

Однако скорость и эффективность каждого алгоритма сжатия зависит от машины, на которой происходит его выполнение, поэтому перед выбором определенного алгоритма сжатия удобно пользоваться существующими и постоянно обновляемыми онлайн ресурсами для тестирования эффективности алгоритмов под разными архитектурами компьютеров.

Для просмотра тестируемых изображений в хорошем качестве, используйте электронную версию данной работы: [12]

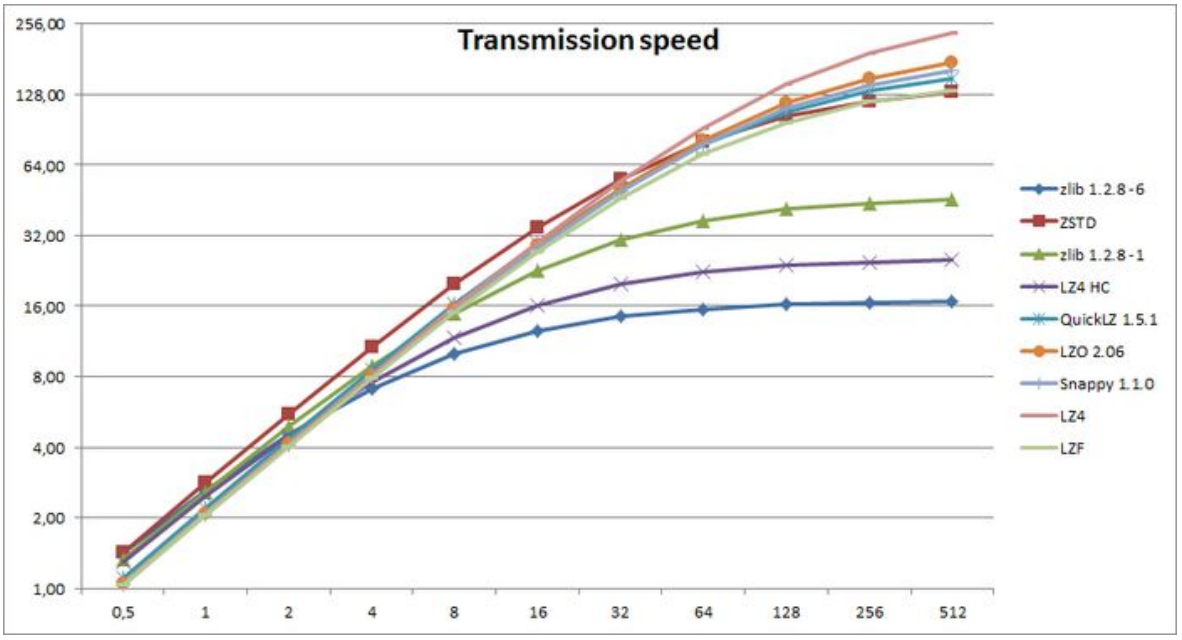
ПРИЛОЖЕНИЯ



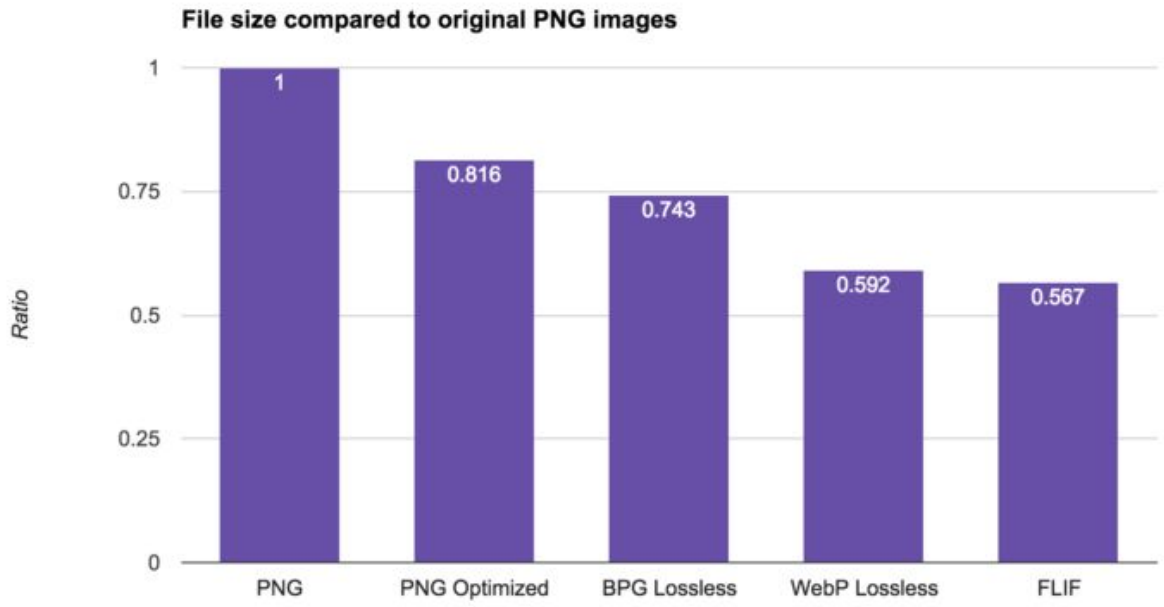
Приложение 1

Compressor name	Compress.	Decompress.	Compr. size	Ratio
memcpy	10362 MB/s	10790 MB/s	211947520	100.00
blosclz 2.0.0 -1	6485 MB/s	7959 MB/s	211947520	100.00
blosclz 2.0.0 -3	1073 MB/s	5909 MB/s	199437330	94.10
blosclz 2.0.0 -6	412 MB/s	1083 MB/s	137571765	64.91
blosclz 2.0.0 -9	403 MB/s	1037 MB/s	135557850	63.96
brieflz 1.2.0 -1	197 MB/s	431 MB/s	81138803	38.28
brieflz 1.2.0 -3	108 MB/s	436 MB/s	75550736	35.65
brieflz 1.2.0 -6	19 MB/s	468 MB/s	67208420	31.71
brieflz 1.2.0 -8	0.46 MB/s	473 MB/s	64912139	30.63
brotli 2019-10-01 -0	420 MB/s	419 MB/s	78433298	37.01
brotli 2019-10-01 -2	154 MB/s	485 MB/s	68060686	32.11
brotli 2019-10-01 -5	35 MB/s	520 MB/s	59568603	28.11
brotli 2019-10-01 -8	10 MB/s	533 MB/s	57140168	26.96
brotli 2019-10-01 -11	0.63 MB/s	451 MB/s	50412404	23.79
bzip2 1.0.8 -1	18 MB/s	52 MB/s	60484813	28.54
bzip2 1.0.8 -5	16 MB/s	44 MB/s	55724395	26.29
bzip2 1.0.8 -9	15 MB/s	41 MB/s	54572811	25.75
crush 1.0 -0	53 MB/s	413 MB/s	73064603	34.47
crush 1.0 -1	6.11 MB/s	455 MB/s	66494412	31.37
crush 1.0 -2	0.82 MB/s	468 MB/s	63746223	30.08
csc 2016-10-13 -1	21 MB/s	73 MB/s	56201092	26.52
csc 2016-10-13 -3	9.38 MB/s	71 MB/s	53477914	25.23
csc 2016-10-13 -5	3.86 MB/s	77 MB/s	49801577	23.50
density 0.14.2 -1	2214 MB/s	2677 MB/s	133042166	62.77
density 0.14.2 -2	933 MB/s	1433 MB/s	101651444	47.96
density 0.14.2 -3	432 MB/s	529 MB/s	87649866	41.35
fastlz 0.1 -1	341 MB/s	806 MB/s	104628084	49.37
lizard 1.0 -49	1.95 MB/s	1729 MB/s	60679215	28.63
lz4 1.9.2	737 MB/s	4448 MB/s	100880800	47.60
lz4fast 1.9.2 -3	838 MB/s	4423 MB/s	107066190	50.52
lz4fast 1.9.2 -17	1201 MB/s	4632 MB/s	131732802	62.15
lz4hc 1.9.2 -1	131 MB/s	4071 MB/s	83803769	39.54
lz4hc 1.9.2 -4	81 MB/s	4210 MB/s	79807909	37.65
lz4hc 1.9.2 -9	33 MB/s	4378 MB/s	77884448	36.75
lz4hc 1.9.2 -12	11 MB/s	4427 MB/s	77262620	36.45
lzf 3.6 -0	400 MB/s	869 MB/s	105682088	49.86
lzf 3.6 -1	398 MB/s	914 MB/s	102041092	48.14
lzfse 2017-03-08	90 MB/s	934 MB/s	67624281	31.91
lzg 1.0.10 -1	91 MB/s	653 MB/s	108553667	51.22
lzg 1.0.10 -4	53 MB/s	655 MB/s	95930551	45.26
lzg 1.0.10 -6	29 MB/s	702 MB/s	89490220	42.22
zlib 1.2.11 -1	119 MB/s	383 MB/s	77259029	36.45
zlib 1.2.11 -6	35 MB/s	407 MB/s	68228431	32.19
zlib 1.2.11 -9	14 MB/s	404 MB/s	67644548	31.92
zling 2018-10-12 -0	75 MB/s	216 MB/s	62990590	29.72
zling 2018-10-12 -1	67 MB/s	221 MB/s	62022546	29.26
zling 2018-10-12 -2	60 MB/s	225 MB/s	61503093	29.02
zling 2018-10-12 -3	53 MB/s	226 MB/s	60999828	28.78
zling 2018-10-12 -4	46 MB/s	226 MB/s	60626768	28.60
zstd 1.4.3 -1	480 MB/s	1203 MB/s	73508823	34.68
zstd 1.4.3 -2	356 MB/s	1067 MB/s	69594511	32.84
zstd 1.4.3 -5	104 MB/s	932 MB/s	63993747	30.19
zstd 1.4.3 -8	46 MB/s	1055 MB/s	60757793	28.67
zstd 1.4.3 -11	20 MB/s	1001 MB/s	59239357	27.95
zstd 1.4.3 -15	7.12 MB/s	1024 MB/s	57167422	26.97

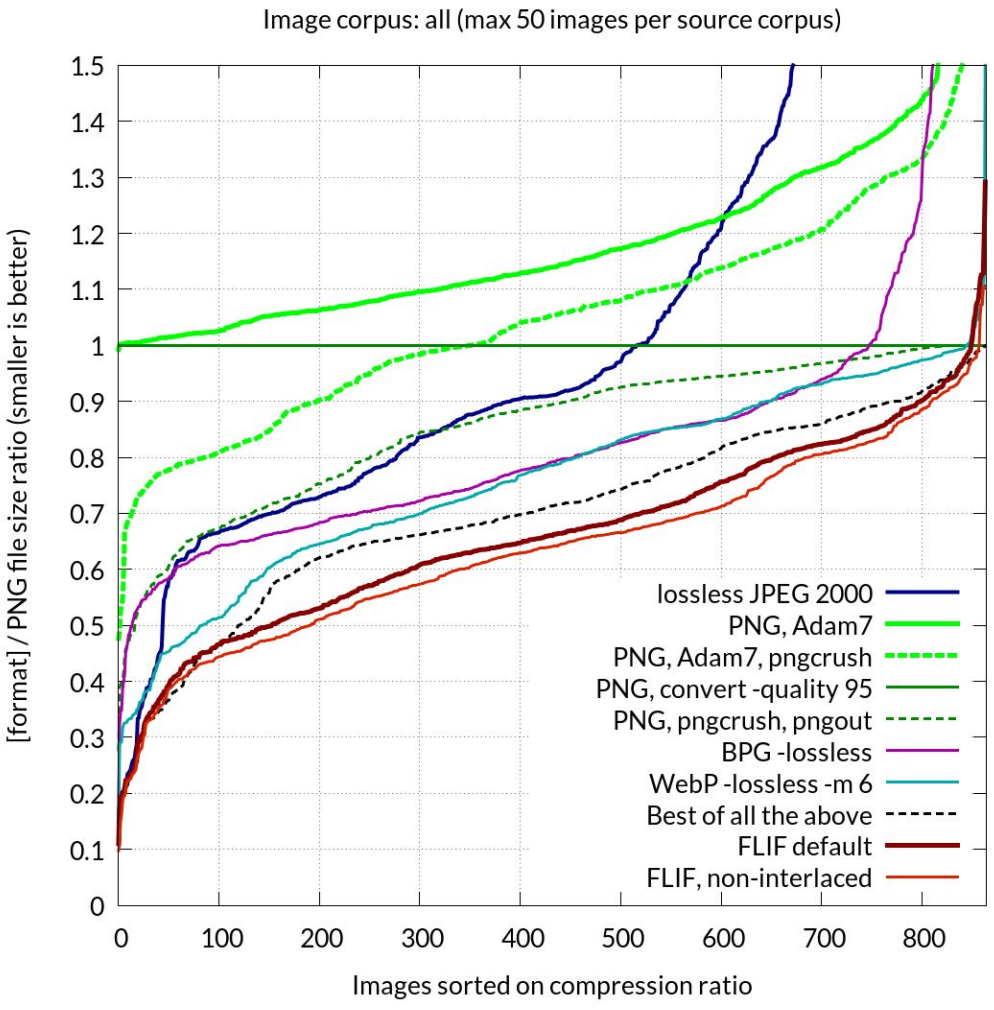
Приложение 2



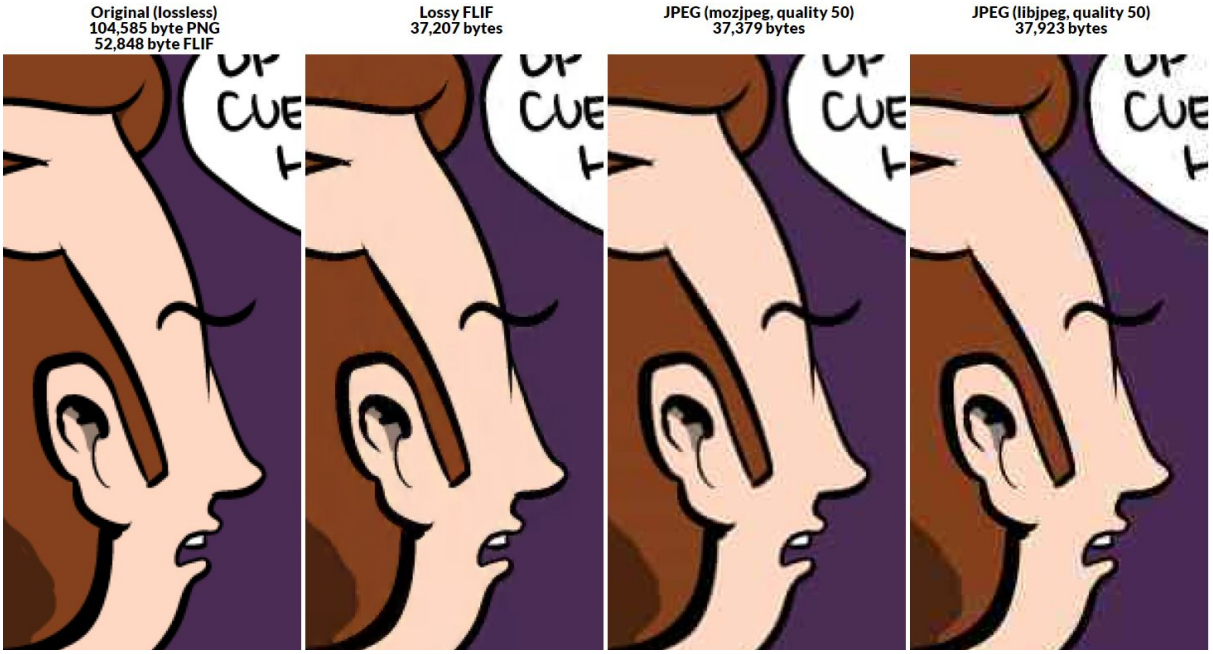
Приложение 3



FLIF comparison 1



FLIF comparison 2



FLIF comparison 3

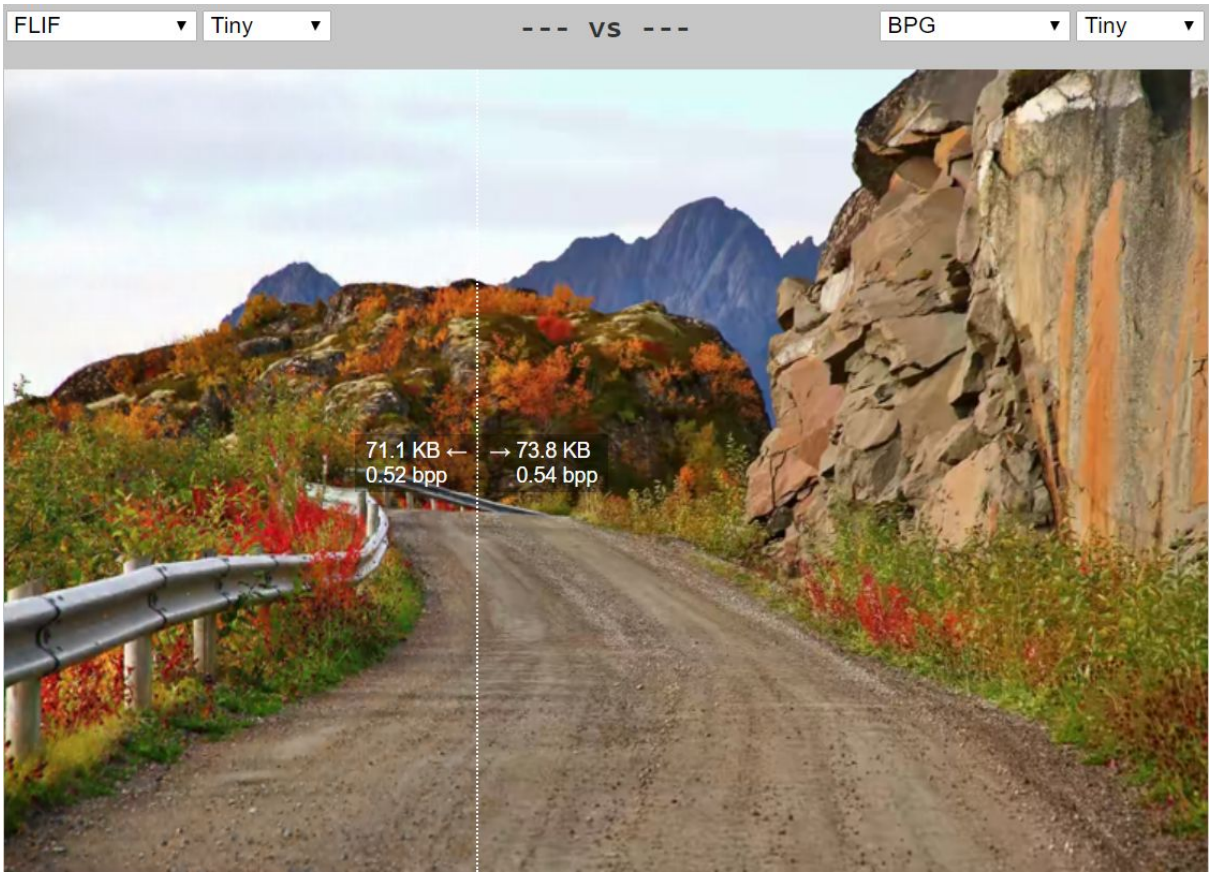


Image compressions comparison

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы сжатия данных без потерь [Режим доступа] https://habr.com/ru/post/231177/ [Дата доступа] 23.12.19
2. Сайт для сравнения алгоритмов сжатия без потерь [Режим доступа] https://quixdb.github.io/squash-benchmark/ [Дата доступа] 23.12.19
3. FLIF [Режим доступа] https://flif.info/ [Дата доступа] 23.12.19
4. PNG [Режим доступа] http://psand.ru/format-png-8-i-png-24-v-chem-razlichiya-kakoy-format-luchshe-vyibrat/ [Дата доступа] 23.12.19
5. JPEG XS[Режим доступа] *https://3dnews.ru/968254* [Дата доступа] 23.12.19
6. Generation loss [Режим доступа] https://www.youtube.com/watch?time_continue=20&v=_h5gC3EzlJg&feature=emb_logo [Дата доступа] 23.12.19
7. FLIF example [Режим доступа] https://www.youtube.com/watch?v=ByH7RMsMxBY [Дата доступа] 23.12.19
8. Image compressions comparison [Режим доступа] https://wyohknott.github.io/image-formats-comparison/ [Дата доступа] 23.12.19
9. Video compressions comparison [Режим доступа] http://video.1ko.ch/codec-comparison/ [Дата доступа] 23.12.19
10. FLIF sources [Режим доступа] https://github.com/FLIF-hub/FLIF [Дата доступа] 23.12.19
11. lzbench sources [Режим доступа] https://github.com/inikep/lzbench [Дата доступа] 23.12.19
12. Электронная версия данной работы [Режим доступа] https://docs.google.com/document/d/1gSuGqcJ-S_OkwdgjZZjETqAeuC10MwCFMggNIeIVKFA/edit?usp=sharing [Дата доступа] 23.12.19