**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

**Отчет**

**по контрольному домашнему заданию**

по теме **алгоритмы сжатия данных без потерь**

по направлению подготовки бакалавров 09.03.04 «Программная инженерия»

|  |  |
| --- | --- |
|  | Выполнил  студент группы БПИ181  образовательной программы 09.03.04 «Программная инженерия»  А.А. Матевосян\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись, Дата |

Принял

Доцент ДПИ, к.т.н

Р.З. Ахметсафина

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись, Дата

**Москва 2020**

# Оглавление

[Оглавление 2](#_Toc36914147)

[Постановка задачи 3](#_Toc36914148)

[Описание реализации алгоритмов и использованных структур данных 4](#_Toc36914149)

[bitbuf, ibitbuf, obitbuf 4](#_Toc36914150)

[Timer 4](#_Toc36914151)

[Archiver 4](#_Toc36914152)

[Node 4](#_Toc36914153)

[Triplet 4](#_Toc36914154)

[compare 4](#_Toc36914155)

[ext\_char 4](#_Toc36914156)

[Huffman 4](#_Toc36914157)

[LZ-77 5](#_Toc36914158)

[LZW 6](#_Toc36914159)

[План эксперимента 7](#_Toc36914160)

[Использованные аппаратные средства для проведения эксперимента 7](#_Toc36914161)

[Результаты экспериментов - таблицы и графики 8](#_Toc36914162)

[Сравнительный анализ алгоритмов по эффективности сжатия файлов разных типов, по скорости работы 16](#_Toc36914163)

[Заключение 17](#_Toc36914164)

[Использованные источники 18](#_Toc36914165)

# Постановка задачи

Разработать на языке C++ программу, реализующую алгоритмы сжатия данных без потерь, получение архивного файла из исходного и разархивированного файла из архивного (упаковка файла и распаковка архива).

Список реализуемых алгоритмов сжатия

1. алгоритм Хаффмана
2. алгоритм Лемпеля-Зива LZ77 (со скользящим окном),
3. алгоритм Лемпеля-Зива-Велча LZW

Провести вычислительный эксперимент для исследования эффективности реализованных алгоритмов сжатия без потерь (упаковка и распаковка) для файлов разного типа. Для проведения эксперимента с алгоритмами сжатия без потерь необходимо использовать набор из N файлов различных типов с именами 1.\* … N.\*.

В наборе присутствуют следующие файлы с размером 2 мегабайт:

* видео 1.mp4
* документ Word 2.docx
* исполняемый файл 3.exe
* документ 4.pdf
* презентация 5.pptx
* текстовый 6.txt
* изображение черно-белое 7.bmp
* изображение черно-белое 8.jpg
* цветное изображение 9.bmp
* цветное изображение 10.jpg

# Описание реализации алгоритмов и использованных структур данных

## bitbuf, ibitbuf, obitbuf

Представляют собой обертки для побитовой записи и чтения. Данная структура дает возможность побитого манипулировать байтами, то есть читать и записывать не байтами, а битами.

Сами данные в обертке хранятся в виде последовательности байтов, а доступ к конкретному биту в байте поддерживается структурой bitset. При чтении одного бита происходит чтение данного бита и передвигается указатель на следующий бит. Точно так же работает запись отдельного бита.

Есть возможность чтения битов в обычном и обратном порядке, то есть если мы читаем последовательность “0101”, то оно либо равно 5, либо 10, смотря с какой стороны мы считаем. Преобразование двоичного кода к десятичному числу происходит с помощью побитовых операции.

Можно записать несколько битов данного числа или сразу написать конкретные биты. Поддерживается запись как в прямом, так и обратном порядке. Каждый кусок битов, соответствующий одному байту, записывается в последовательность. При записи в выходной поток проверятся остаточные биты, и они записываются в поток с добавлением не достающих нулей.

## Timer

Класс, который дает возможность для измерения времени в наносекундах. В качестве инструмента замера используется high\_resolution\_clock из библиотеки chrono.

## Archiver

Абстрактная структура для распаковки и упаковки файлов и потоков. От него наследуются все классы архиваторы.

## Node

Структура данных, для хранения узла для алгоритма Хаффмана.

## Triplet

Структура данных, для хранения тройки для алгоритма LZ77.

## compare

Структура данных, которая реализует операцию сравниванию двух узлов.

## ext\_char

Тип символа для алгоритма Хаффмана. Так как в алгоритме Хаффмана нужно как-то отметить конец файла не существующим символом из алфавита, был определен специальный тип ext\_char. Этот тип сам по себе является type alias-ом для int-а, но для избегания от путаницы было решено использовать его в место int-а.

## Huffman

Структура данных, которая реализует алгоритм Хаффмана.

Упаковка файла:

1. Построение таблицу частот для выбранного файла
2. Запись таблицы частот в выходной файл
3. Построение дерева узлов из таблицы частот
4. Замена каждого символа из входного файла кодом из построенного дерева
5. Запись полученного кода в выходной файл

Распаковка файла:

1. Чтение таблицы частот из выбранного файла
2. Построение дерева узлов из таблицы частот
3. Замена каждого кода из входного файла символом из построенного дерева
4. Запись полученного символа в выходной файл

При упаковке каждый код соответствующего символа записывается с помощью обертки. При распаковке обертка побитого считает файл и смотрит под соответствующий символ, который соответствуют данному коду.

## LZ-77

Структура данных, которая реализует алгоритм Лемпеля-Зива LZ77 (со скользящим окном).

Упаковка файла:

1. Поиск самой длинной подстроки из буфера поиска, который является префиксом буфера предпросмотра
2. Сохранения первого индекса полученной подстроки начиная нумерацию с правого конца буфера поиска
3. Сохранение длины подстроки и следующего символа следующего после префикса буфера предпросмотра
4. Получение тройки из индекса, длины и следующего символа
5. Запись тройки в выходной файл
6. Передвинуть окно буферов на право на одну единицу больше, чем длина тройки

Распаковка файла:

1. Чтение тройки из входного файла
2. Восстановление подстроки из оскользавшего окна использую подстроку, которая начинается с индекс тройки и имеет соответствующую длину из тройки
3. Запись символа из тройки в выходной файл
4. Извлечение последнего символа при случаи избыточной записи в выходной файл

Для каждой реализации архиватора под LZ77 нужно явно указать размер буфера поиска и размер буфера предпросмотра. Сама структура считает эти значения и задает максимальный размер для хранения троек побитого. Также недостатком выбранных размеров из условия является наличие остаточного одного бита. Например, при длине буфера предпросмотра 1024 бит, для его хранения нужно хранить 11 бит, так как оно имеет 1025 возможных значений. Для решения той же проблемы для буфера поиска используется некоторый трюк, тройки вида “0 0 \*”, хранятся в виде “1 0 \*”, что избавляет от хранения нулевого значения для индекс найденной подстроки в буфере поиска. При записи индекс уменьшается на единицу, а при чтении увеличивается на единицу.

Перед записью элементы троек записываются в одно число посредством построения числа из их бинарной записи при вставке их сразу за друг другом использую максимальную длину из самого класса. Затем при чтении считаются заданные биты из закодированного числа и получается исходная тройка. При чтении последней тройки, последний добавленный символ отбрасывается из выходного файла, если он является избыточным.

## LZW

Структура данных, которая реализует алгоритм Лемпеля-Зива-Велча LZW

Упаковка файла:

1. Добавление всех доступных значений байтов в словарь
2. Чтение очередного символа из входного файла
3. Добавление данного символа во временную строку
4. Поиск временной строки в словаре
5. При отсутствии строки в словаре, добавляется временная строка в словарь и записываются в файл индекс записи словаря
6. Если размер словаря уже переполнен, то новое слово не добавляется
7. Временная строка принимает значение очередного символа
8. Если после чтения файла, временная строка не пустая, то записывается значение словаря в файл

Распаковка файла:

1. Добавление всех доступных значений байтов в словарь
2. Чтение очередного индекс и просмотр индекса в словаре
3. Если текущая длина словаря равна индексу, то в словарь записывается текущая временная строка и его первый символ
4. Запись значения словаря для текущего индекса
5. Если текущий индекс не перевешивает допустимый размер словаря и временная строка не пустая, то в словарь записывается временная строка и первый символ, который соответствует значению словаря для текущего индекса
6. Временная строка принимает значение, соответствующее значению словаря для текущего индекса

Для каждой реализации данного архиватора нужно явно указать максимальное количество бит словаря. Если размер словаря переполняется, то запись в словарь больше не происходит. Тут реализуется классический вариант алгоритма LZW, без динамически расширяемого словаря, с константной длинной индексов словаря, которые записываются и считываются с помощью обертки.

# План эксперимента

Для проведения данного эксперимента была написана программа, которая выполняет сжатые и восстановление файлов и записывает соответствующие результаты. Перед там как запустить программу нужно выполнить следующие шаги:

1. Задать пути к входным данным, к выходным сжатым данным, к выходным восстановленным данным, к выходным результирующим файлам
2. Задать доступные архиваторы с их настройками
3. Задать имена входных файлом, расширения для сжатых и восстановленных файлов для каждого архиватора

После запуска программа выполняет следующие действия:

1. Получение размера, таблицы частот и энтропии для каждого входного файла и сохранение данных в виде файлов
2. Запуск каждого архиватора для сжатия и восстановления входных данных
3. Игнорирование некоторых результатов для более точной оценки времени
4. Измерение запуска архиватора с одним и тем же параметром для измерения среднего времени сжатия и восстановления файлов
5. Проверка на равенство восстановленного файла и входного файла и сохранения результата в файл
6. Запись исходного размера сжатого файла

После завершения работы программы нужно выполнить следующие шаги:

1. Сбор и анализ полученных результатов, проверка алгоритмов на корректность, постройка графиков и таблиц для более наглядного и легкого анализа результатов
2. Сбор всех материалов и написание отчета

Таблицы и графики были созданы с использованием программы MS Excel. Все файлы имеют почти одинаковы размер и соответствуют условиям задания. Все реализации алгоритмов работают корректно.

# Использованные аппаратные средства для проведения эксперимента

1. Операционная система - Windows 10 Домашняя
2. Процессор - Intel® Core™ i7-8565U, 1,8 ГГц, 4 ядра, кеш 8 МБ
3. Видеокарта - NVIDIA® GeForce® GTX 1050 Max-Q, Intel® UHD Graphics 620
4. Дисплей - 15,6” Full-HD (1920x1080) 16:9, глянцевый
5. Оперативная память - 16 ГБ DDR4 2400 МГц, встроенная
6. Хранение данных - 512 ГБ PCIe® 3.0 x2 SSD

# Результаты экспериментов - таблицы и графики

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Файл | S1 | H | Алгоритм Хаффмана | | Алгоритм LZ77, окно 5 Кб | | Алгоритм LZ77, окно 10 Кб | | Алгоритм LZ77, окно 20 Кб | | Алгоритм LZW | |
|
|
| S2 | K | S2 | K | S2 | K | S2 | K | S2 | K |
| 1 | 2107842 | 0.999656 | 2110330 | 1.00 | 3942894 | 1.87 | 4077155 | 1.93 | 4117917 | 1.95 | 3032688 | 1.44 |
| 2 | 2101546 | 0.995383 | 2059649 | 0.98 | 205399 | 0.10 | 178275 | 0.08 | 168372 | 0.08 | 3179392 | 1.51 |
| 3 | 2072295 | 0.989118 | 2142109 | 1.03 | 4002802 | 1.93 | 4146900 | 2.00 | 4192576 | 2.02 | 3173650 | 1.53 |
| 4 | 2139566 | 0.999861 | 2061589 | 0.96 | 3385111 | 1.58 | 3492527 | 1.63 | 3536357 | 1.65 | 2940568 | 1.37 |
| 5 | 2066451 | 0.994086 | 2059366 | 1.00 | 3620084 | 1.75 | 3720841 | 1.80 | 3741750 | 1.81 | 2967814 | 1.44 |
| 6 | 2058866 | 0.998141 | 1099011 | 0.53 | 1100834 | 0.53 | 1047227 | 0.51 | 1022771 | 0.50 | 544424 | 0.26 |
| 7 | 2047469 | 0.532654 | 392175 | 0.19 | 128348 | 0.06 | 127447 | 0.06 | 128061 | 0.06 | 98918 | 0.05 |
| 8 | 2088062 | 0.152225 | 2016836 | 0.97 | 2983096 | 1.43 | 3019744 | 1.45 | 2985728 | 1.43 | 2251886 | 1.08 |
| 9 | 2143474 | 0.93666 | 1993766 | 0.93 | 2196928 | 1.02 | 2178405 | 1.02 | 2174183 | 1.01 | 3212612 | 1.50 |
| 10 | 2054646 | 0.965519 | 2100985 | 1.02 | 3870362 | 1.88 | 3961523 | 1.93 | 3957700 | 1.93 | 2856990 | 1.39 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Файл | S1 | H | Алгоритм Хаффмана | Алгоритм LZ77, окно 5 Кб | Алгоритм LZ77, окно 10 Кб | Алгоритм LZ77, окно 20 Кб | Алгоритм LZW |
|
|
| tu | tu | tu | tu | tu |
| 1 | 2107842 | 0.999656 | 2.29E+08 | 2.01E+09 | 4.01E+09 | 6.74E+09 | 6.96E+08 |
| 2 | 2101546 | 0.995383 | 2.17E+08 | 1.23E+08 | 1.94E+08 | 2.90E+08 | 6.39E+08 |
| 3 | 2072295 | 0.989118 | 2.25E+08 | 2.04E+09 | 4.03E+09 | 6.86E+09 | 6.73E+08 |
| 4 | 2139566 | 0.999861 | 2.15E+08 | 1.73E+09 | 3.55E+09 | 5.79E+09 | 6.76E+08 |
| 5 | 2066451 | 0.994086 | 2.19E+08 | 1.83E+09 | 3.68E+09 | 6.62E+09 | 6.33E+08 |
| 6 | 2058866 | 0.998141 | 1.38E+08 | 1.38E+09 | 2.46E+09 | 4.39E+09 | 2.34E+08 |
| 7 | 2047469 | 0.532654 | 5.90E+07 | 1.02E+09 | 1.70E+09 | 2.93E+09 | 1.74E+08 |
| 8 | 2088062 | 0.152225 | 2.07E+08 | 1.62E+09 | 3.12E+09 | 5.23E+09 | 5.20E+08 |
| 9 | 2143474 | 0.93666 | 1.86E+08 | 1.53E+09 | 2.81E+09 | 4.90E+09 | 5.76E+08 |
| 10 | 2054646 | 0.965519 | 2.25E+08 | 1.97E+09 | 3.92E+09 | 6.54E+09 | 6.07E+08 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Файл | S1 | H | Алгоритм Хаффмана | Алгоритм LZ77, окно 5 Кб | Алгоритм LZ77, окно 10 Кб | Алгоритм LZ77, окно 20 Кб | Алгоритм LZW |
|
|
| tp | tp | tp | tp | tp |
| 1 | 2107842 | 0.999656 | 1.66E+08 | 1.31E+08 | 1.40E+08 | 1.36E+08 | 2.42E+08 |
| 2 | 2101546 | 0.995383 | 1.56E+08 | 3.58E+07 | 3.80E+07 | 3.60E+07 | 2.29E+08 |
| 3 | 2072295 | 0.989118 | 1.55E+08 | 1.31E+08 | 1.45E+08 | 1.38E+08 | 2.49E+08 |
| 4 | 2139566 | 0.999861 | 1.49E+08 | 1.18E+08 | 1.19E+08 | 1.25E+08 | 2.38E+08 |
| 5 | 2066451 | 0.994086 | 1.57E+08 | 1.20E+08 | 1.30E+08 | 1.79E+08 | 2.30E+08 |
| 6 | 2058866 | 0.998141 | 1.04E+08 | 5.90E+07 | 5.60E+07 | 5.71E+07 | 8.10E+07 |
| 7 | 2047469 | 0.532654 | 5.71E+07 | 3.40E+07 | 3.40E+07 | 3.40E+07 | 4.80E+07 |
| 8 | 2088062 | 0.152225 | 1.44E+08 | 1.15E+08 | 1.10E+08 | 1.09E+08 | 2.01E+08 |
| 9 | 2143474 | 0.93666 | 1.29E+08 | 9.10E+07 | 8.80E+07 | 8.75E+07 | 2.09E+08 |
| 10 | 2054646 | 0.965519 | 1.65E+08 | 1.30E+08 | 1.34E+08 | 1.35E+08 | 2.42E+08 |

# Сравнительный анализ алгоритмов по эффективности сжатия файлов разных типов, по скорости работы

Во-первых, нужно подчеркнуть тот факт, что не всегда сжатие работает положительно. Иногда может получиться так, что выходной файл больше исходного. При этом этот исход в основном связен не с алгоритмом, а с исходный файлом.

Алгоритм Хаффмана почти всегда имеет коэффициент сжатия близкий к единице, что говорит, а неэффективности данного алгоритма для большинства файлов. Эти файлы имеют практически равномерное распределение частот символов.

Алгоритм LZ77 работает для некоторых файлов намного хуже, чем алгоритм Хаффмана, но при этом он отлично работает в файле 2, который является документом Word, получая коэффициент сжатия 0.1.

Алгоритм LZW работает в среднем плохо, он почти всегда имеет коэффициент 1.5, правда для большинства файлов коэффициент лучше, чем у LZ77.

Очень хорошо сжимается файлы номер 6 и 7. Первый является текстовым файлом со случайным набором букв из английского алфавита, а второй является RAW чёрно-белой фотографией. И в том, и в другом есть сильный разброс в относительных частотах файла. Для этих файлах все алгоритмы дают коэффициент сжатия меньше 0.54, что говорит о том, что такие типы файлов очень хорошо сжимаются.

Самыми медленными алгоритмами сжатия являются алгоритмы LZ77, при этом чем больше размер окна тем медленнее она работает, но это не касается алгоритма распаковки, оно работает почти одинаково для всех размеров окон и является самым быстрым из всех алгоритмов.

Далее идет алгоритм LZW, который является средним для упаковки, но является самым медленным при распаковке.

Алгоритм Хаффмана является самым быстрым по упаковке и средним при распаковке, слегка уступая алгоритмам LZ77.

# Заключение

В данном наборе файлов в основном находятся файлы, которые внутри имеют сжатую структуру. Не всегда алгоритмы сжатия дают хороший результат, иногда выходной файл может быть намного больше, чем исходный, а иногда может превзойти исходный в несколько раз. Но главной задачей алгоритмов сжатия является сокращение исходного размера файла. В основном это связано с тем, что исходный файл сам является сжатым файлом или имеет худшие характеристики для данного алгоритма. Например, алгоритм Хаффмана работает хорошо, когда некоторые символы встречаются намного чаще чем другие или для LZ77, когда встречаются часто некоторые фрагменты, а не символы. Из-за таких различии LZ77 в основном работает намного лучше, чем другие алгоритмы, это можно явно увидеть в файле 3.

Часто в реализации алгоритма нужно выбрать между минимизацией скорости работы и максимизацией коэффициента сжатия. Также нужно реализовать обертки для работы с файлами в побитовым режиме. Важно еще учесть то, что некоторые алгоритмы имеют настраиваемые параметры, как например размеры окон буфера поиска и буфера предпросмотра для LZ77 или, например длины словаря для LZW. В зависимости от этих параметров может измениться время выполнение и коэффициент сжатия.

Из-за таких сложностей нельзя выбрать лучший алгоритм, а можно в ходе разных тестов, дать рекомендации по использованию алгоритмов. Эти рекомендации помогут при выборе правильного алгоритма для сжатия определенных файлов.

# Использованные источники

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 3-е издание. — Пер. с англ. — М.; СПб.; Киев: ООО «И.Д. Вильямс», 2013. — 1328 с.
2. Клейнберг Дж., Тардос Е. Алгоритмы: разработка и применение. Классика Computer Science / Пер. с англ. Е. Матвеева. - СПб.: Питер, 2018.- 800 с.
3. Сэломон Д. Сжатие данных, изображений и звука. - Москва: Техносфера, 2006. - 368 с.
4. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. - М.: ДИАЛОГ-МИФИ, 2003.- 384 с.