НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Алгоритмы и структуры данных»

Контрольное домашнее задание

Архиваторы

Выполнила: Уварова Анастасия,

студент гр. БПИ181_1

Москва 2020

Оглавление

1. Постановка задачи.....3

2. Описание алгоритмов и структур данных.....4

3. Описание реализации алгоритмов.....5

4. План эксперимента.....6

5. Аппаратные средства.....6

6. Результаты эксперимента.....7

7. Сравнительный анализ алгоритмов.....7

8. Заключение.....8

1. Постановка задачи

1) Разработать на языке C++ программу, реализующую следующие алгоритмы сжатия без потерь:

а) алгоритм Шеннона-Фано
б) алгоритм Лемпеля-Зива LZ77 (3 размера окна: 5кб, 10кб, 20 кб)

в) алгоритм Лемпеля-Зива-Велча LZW (бонус-задача)

В данном домашнем задании реализованы пункты а) и б).

2) Провести вычислительный эксперимент для исследования эффективности алгоритмов для файлов разных типов. Вычислить энтропию файлов, коэффициент сжатия как отношение размера сжатого файла к размеру исходного. Измерить время распаковки и упаковки файлов.

Исследования проведены для следующих типов файлов (нумерация соответствует названиям файлов):

1. docx

2. jpg (цвет)

3. bmp (цвет)

4. bmp (ч.б)

5. pdf

6. djvu

7. jpg (ч.б.)

8. pptx

9. dll

10. txt

Проведены необходимые вычисления, результаты отображены в таблицах и графиках.

3) Подготовить отчет и загрузить работу в LMS

2. Описание алгоритмов и структур данных

1) Алгоритм Шеннона-Фано

Алгоритм реализован при помощи рекурсии. На каждом этапе частоты появления байтов в тексте (отсортированные по убыванию) разбиваются на две части, затем алгоритм вызывается от левой и правой частей. При этом к ко всем кодам символов в правой части прибавляется «1», а в левой - «0». Разбиение производится по медиане — такому элементу, что суммы элементов слева и справа от него предельно близки.

При кодировании используется неявное построение дерева, чтобы сразу получить коды всех символов в map. При декодировании дерево строится явно для упрощения расшифровки. В остальном кодирование и декодирование идентичны.

Использованные структуры данных:

1) std::map для хранения частот символов, итоговых кодов

2) std::vector для хранения частот символов и сумм частот

3) std::pair для реализации пары символ-частоты

4) класс Node — реализация узла дерева

2) Алгоритм LZ77

Кодирование происходит тройками вида (стартовый индекс в строке, длина, новый символ).

Шифрование — прохождение в цикле по всей входной строке. На каждом этапе рассматривается текущий символ, в буфере ищется (простым проходом) самое длинное совпадение, записывается новая тройка (в вектор). Затем действие повторяется (начиная с символа, который следует через один после конца совпадающего участка).

При дешифровке последовательно считываются все тройки, текст восстанавливается тривиально: в уже записанной строке ищется стартовый индекс, записывается участок нужной длины, затем дописывается новый символ.

Структуры данных:

1) std::vector для хранения троек при кодировании

2) класс Triple — реализация кодирующей тройки

3) std::bitset — для получения двоичного представления числа

3. Описание реализации алгоритмов

1) Форматы всех файлов указаны в п.1. Краткое описание реализации — в п.2.

2) Методы работы с битами

а) Шеннон-Фано

- Сжатие

В начало файла записываются длина строки и число различных байтов, которые в ней встречаются, отсортированный вектор вероятностей символов. Затем считывается исходная строка, и код каждого символа записывается побитово следующим образом:

1) хранится переменная текущего байта и текущий сдвиг в байте (от 0 до 7)

2) так как код символа состоит и 0 и 1, делается простой проход по его битам в цикле

3) каждый бит кода добавляется к байту (используется текущий сдвиг), сдвиг увеличивается. Если байт заполняется (сдвиг становится 8), байт пишется в файл, затем байт и сдвиг обнуляются.

4) после прохода по всей строке остаток байта пишется в файл.

-Распаковка

1) Считываются количество символов в строке и число различных символов

2) Считываются вероятности символов, по ним строится декодирующее дерево

3) В цикле считывается текущий бит закодированной строки, (из предварительно считанного байта), если это 1 — идем по дереву вправо, иначе — влево.

4) У узлов дерева есть пометка, соответствует ли его код (путь от корня) какому-то символу. Проверяется метка, если код — символ, делается запись в файл, сбрасывается байт и сдвиг.

5) Если текущий сдвиг 8, считывается новый байт из файла

б) LZ77

- Упаковка

1) В файл записывается число троек, а также верхняя граница в битах для первых двух элементов тройки (для нового символа это, очевидно, 8).

2) В цикле делается проход по всем тройкам, записываются все три элемента следующим образом:

- с помощью структуры данных bitset получается битовое представление числа

- из него берется только число битов, равное вычисленной ранее верхней границе

- биты записываются в файл при помощи схемы, использованной в предыдущем методе

- Распаковка

1) Считывается количество троек, обе границы
2) В цикле побитово считываются элементы тройки, используется известный размер каждого числа в битах

3) По считанной тройке сразу производится декодирование, символы записываются в файл

4. План эксперимента

1) Сбор файлов разных форматов примерно одинакового размера (около 2.2 МБ)

2) Запись информации о файлах в таблицу (подсчет частот символов, энтропия)

3) Прогон сжатия и разжатия каждого файла каждым алгоритмом по 10 раз

4) Измерение времени каждого сжатия с помощью библиотеки chrono. Использовался класс system_clock и duration_cast для хранения времени (в

миллисекундах, так как наносекунды оказались слишком мелкой единицей измерения), запись результатов в таблицу

5) Построение графиков по полученным данным, выводы по проведенному эксперименту

5. Аппаратные средства

Ноутбук: Lenovo - Thinkpad X1 Carbon 6th Gen

Среда разработки: Clion

Компилятор: Clang

6. Результаты эксперимента

Таблицы (и соответствующие им графики) находятся в файлах coef.ods, freq.ods, time.ods в папке Results.

7. Сравнительный анализ алгоритмов

а) Эффективность

Для алгоритма Шеннона-Фано лучшие результаты (коэффициент сжатия около 0.4 - 0.5) для файлов 3 и 4 (.bmp), неплохой результат для dll и txt (0.6 — 0.7). Для остальных файлов результат в районе единицы, то есть, видимого улучшения нет (тем не менее, чем больше, к примеру, jpg файлы, тем лучший результат дает алгоритм).

По сравнению с алгоритмом Шеннона-Фано алгоритмы LZ77 могут существенно увеличить файлы в размере, что хорошо видно на графиках. В 2 раза увеличились размеры jpg файлов (вероятно, потому что это уже сжатый формат), docx (не очень удачно был подобран файл, в нем в основном картинки), pdf, djvu, pptx. Тем не менее, алгоритмы показали очень хороший результат для bmp файлов: коэффициент сжатия достиг минимума в 0,24 для LZ775 на черно-белой картинке, для цветной — примерно 0.375. Лучший результат для черно-белого варианта обусловлен большим количеством совпадающих участков. Для dll файла результат незначительно лучше первого алгоритма. Для текстового файла сжатие оказалось менее эффективным (коэффициент около 0.8). Примечательно, что текстовый файл — единственный, для которого увеличение размера скользящего окна дало прирост в качестве (если у

LZ775 сжатие 0,844, то у LZ7720 уже 0,774). Вероятно, при увеличении размера окна можно было бы добиться от алгоритма лучших результатов.

б) Скорость

Самым эффективным по скорости упаковки оказался алгоритм Шеннона-Фано — для различных файлов скорость находится в пределах от 100 до 310 мс. Это гораздо меньше чем для алгоритма LZ77. Для него скорость сильно падает с увеличением окна (в алгоритме использовался обычный поиск, результат можно было бы улучшить, воспользовавшись, к примеру, алгоритмом Кнута-Морриса-Пратта). Упаковка для окна 5кб занимает от 13 до 30 секунд, для 10 — от 24 до 61, для 20 - от 43 до 123. То есть, время исполнения практически прямо пропорционально размеру скользящего окна (при этом, как было сказано ранее, размер окна положительно повлиял только на txt файл). Также стоит заметить, что, чем лучше сжался файл, тем меньше времени было потрачено на его упаковку.

По скорости упаковки лучшим, опять же, оказался алгоритм Шеннона-Фано. Распаковка заняла гораздо меньше времени для всех алгоритмов, особенно хорошо видна разница для алгоритмов LZ77. Все распаковки не вышли за пределы интервала 100-550 мс, а быстрее, опять же, оказались распаковки для более сжатых файлов.

8. Заключение

По результату проделанного эксперимента можно сделать выводы как по «качеству сжатия» разных форматов файлов, так и по работе конкретных алгоритмов. Лучше всего (и быстрее всего) сжимаются форматы bmp, dll, txt. С jpg, djvu, pdf не смог справиться ни один из алгоритмов, вероятно, из-за особенности форматов (для Фано, возможно, результат был бы на более больших файлах). Файл формата docx также не был сжат из-за неудачного содержимого (картинки). Сжатие для pdf и pptx получилось только для алгоритма Шеннона-Фано, но вышло совсем незначительным.

Алгоритм LZ77 затрачивал слишком много времени на упаковку в сравнении с Шенноном-Фано, а также для «неудачных» форматов выдавал прирост размера файла в 2 раза. Тем не менее, он дал достаточно хороший выигрыш для формата bmp.

Самым удачным решением для архивации с помощью данных алгоритмов была бы их комбинация: использовать Шеннона-Фано для форматов, где LZ77 точно

бесполезен, выбирать лучший результат из двух для остальных (и не использовать LZ77 для слишком больших файлов).