НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Дисциплина: «Алгоритмы и структуры данных»

Контрольное Домашнее Задание

Исследование алгоритмов архивации Хаффмана, Шеннона-Фано и LZ77

Пояснительная записка

Выполнил: Шакин Кирилл,
студент БПИ163.

Преподаватель: Мицюк А.А.,
старший преподаватель департамента
программной инженерии
факультета компьютерных наук

Оглавление

Постановка задачи

(1)

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)

Алгоритмы Хаффмана и Шеннона-Фано работают в два прохода. Сначала строится таблица частот встречаемости символов в конкретном упаковываемом файле. Затем строится кодовое дерево. По нему определяются коды символов и с их помощью упаковывается файл.Для распаковки алгоритмам требуется знать таблицу частот встречаемости символов/ кодовое дерево, которые использовались при упаковке. Как вариант: соответствующая таблица должна сохраняться в начале упакованного файла и использоваться при распаковке. В начале пишется количество различных символов n, имеющихся в сжимаемом файле, а затем символы по убыванию частоты встречаемости символа в сжимаемом файле.

Алгоритм LZ77 работает в один проход. Используется скользящее окно для динамического построения словаря, который, в свою очередь, используется для кодирования содержимого упаковываемого файла.

(2)

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.

Вычислить:

- энтропию исходных фаий лов; определяется общее количество различных

энтропия файла по формуле $H=\sum_{i=1} w_i log_m w_i$;

- коэффициент сжатия.

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).

(3)

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 (коэффициент сжатия = отношение размеров выходного и входного файлов), данные о времени работы каждого алгоритма с каждым файлом из тестового набора.

Описание алгоритмов и использованных структур данных

Программа состоит из 5 основных классов:

class huffman

Осуществляет получение отображения байт в двоичный код по Алгоритму Хаффмана(простому). Итоговая сложность — O(nlogn) — n – количество различных байт.
Структуры данных:

1. Очередь с приоритетом (priority_queue<Node*, vector<Node*>, Comparator> queue;). Хранит ноды в виде minheap, чтобы на каждом шаге выбирать 2 минимальных нода. Сравнение элементов основывается на частотах. Сложность работы O(nlogn), где n — количество различных байт в файле
2. Отображение байта в двоичный код (map<uchar, string> codes;). Отображает код байта в строку, состоящую из 0 и 1. Ключ выбран как unsigned char, так как байт максимум принимает 256 значений, а unsigned выбран для удобства. Сложность доступа: O(logn) — n – количество различных байт.

class fano

Осуществляет получение отображения байт в двоичный код по Алгоритму Шеннона-Фано (простому). Итоговая сложность — O(nlogn) — n – количество различных байт.
Структуры данных:

1. Отсортированный(требуется для корректной работы алгоритма) массив нодов (Node* nodes;) Сложность работы O(nlogn), где n — количество различных байт в файле, так как сортировка массива осуществляется за O(nlogn)
2. Отображение байта в двоичный код (map<uchar, string> codes;). Отображает код байта в строку, состоящую из 0 и 1. Ключ выбран как unsigned char, так как байт максимум принимает 256 значений, а unsigned выбран для удобства. Сложность доступа: O(logn) — n – количество различных байт.

class lz77

Осуществляет получение отображения байт в тройки (length, offset, char), с помощью алгоритма LZ77(со скользящим окном, заданного размера). Итоговая сложность — $O(n^3)$ — n – количество символов в файле. (цикл по файлу * поиск большего совпадения [$O(n^2)$])
Структуры данных:

1. Vector<Node> хранит выходные байты. Сложность: вставка в конец — амортизированная O(1)

class Utility

Предоставляет (как некий контроллер) вспомогательные классы и методы для работы с файлами и алгоритмами.
Классы:

1. class CSVWriter — осуществляет построчную запись в .csv файл массива с элементами типа string указанной длины.
2. class comp_coef_row — представляет строку в таблице коэффициентов сжатия
3. class alg_time_row — представляет строку в таблице времени сжатия и разжатия.

Методы:

1. map<uchar, int> get_frequency(string path) — функция получения словаря частотности для заданного файла. Функция возвращает map<uchar, int>, объект которого заполняется во время посимвольного прохода по файлу (выбрана из за быстрого доступа по ключу – O(logn), n — количество различных байт). Итоговая сложность для метода — O(m), m — количество символов в файле.
2. map<uchar, string> create_huffman_codes(string path) — функция получения кодов байт по алгоритму Хаффмана, на основании мапа частотности файла. map<uchar,

string> — отображение байта в строку из 0 и 1. Общая сложность алгоритма — O(nlogn).

3. map<uchar, string> create_fano_codes(string path) — функция получения кодов байт по алгоритму Шеннона-Фано, на основании мапа частотности файла(мап переводится в вектор и сортируется, так как алгоритм работает только с отсортированными частотами). map<uchar, string> — отображение байта в строку из 0 и 1. Общая сложность алгоритма — O(nlogn).
4. vector<Node> create_lz77_codes(string path, int histBufMax, int prevBufMax, double& bytes_num)— функция получения кодов байт, представляемых в виде троек (length, offset, char) = Node. Общая сложность алгоритма — $O(n^3)$.

class EncoderDecoder

1. double encode_file(string source_path, string destination_path, map<uchar, string> codes) — функция архивации файла на основании map <uchar, string>, следовательно данная функция подходит как для архивации через алгоритм Хаффмана, так и через алгоритм Шеннона-Фано. Кодирование входного map и запись в файл: вначале записывается байт отображающий количество ключей в map, далее записываются пары <ключ, значение> (строка из 0 и 1), в конце строки записывается разделяющий байт — «\n». Для того, чтобы разбить битовое представление закодированного контента в виде целого числа байт, строка дополняется незначащими нулями и их количество записывается после закодированного отображения. После этого при помощи строкового буфера, который служит для временного хранения битового представления символов, которые получаются при кодировании байт исходного файла, при помощи map, при втором проходе по нему. После каждого очередного увеличения буфера, проверяется можно ли разбить его по 8 бит, чтобы получить целый байт, если такая возможность есть, 8 бит трансформируются в результирующий байт и удаляются из строки буфера, таким образом длина строки постоянно небольшая, и следовательно это эффективно по памяти. Итоговая сложность O(k), где k- сумма количества символов в исходном файле и количества уникальных символов в этом файле.
2. double decode_file(string source_path, string destination_path) — функция разархивирования также подходит для обоих алгоритмов. В начале заполняется map с байтами и их кодами(элементы map сортируются, с помощью переданного компаратора, так чтобы самые короткие коды были вначале). Потом сохраняется число незначащих нулей. Дальше обрабатывается оставшаяся часть файла — биты записываются в буфер-строку. Потом перебираются элементы map до первого совпадения, первое совпадение будет самым коротким кодом, так как они отсортированы, плюс самые короткие коды чаще всего встречаются. Каждый код проверяется с началом буфера, если совпало, то указатель положения в буфере смещается на длину разархивированного символа, значение которого получается из map. Итоговая сложность O(n*log(m)), где n - количество символов в исходном файле, m - количество уникальных символов в этом файле. Произведение потому что перебираются элемента map, однако, за счет отсортированности map, можно считать, что для файлов с неравномерным распределением частот — m << n и имеет уже не столь большое влияние на сложность алгоритма.

3. void write_lz77(vector<Node>& v, string destination_path) — метод архивации троек (Node), полученных после работы алгоритма lz77. Сначала записывается количество троек – 4 байта (потому что может быть велико для файлов с неповторяющимися комбинациями байт). Дальше каждая тройка записывается как 5 байт — пусть тройка (int len, int offset, char ch), тогда len и offset можно

разложить как 256 * k + b и хранить только k и b, которые помещаются в 1 байт. Итог k и b для len = 2 байта, k и b для offset = 2 байта и char ch = 1 байт. Максимальное значение такого разложения 256 * k + b = 65536, а максимальный размер окна 20 * 1024 = 20480 => более чем достаточно и 5 байт на тройку. Итоговая сложно — O(n), где n – количество троек.

4. double decode_lz77(string source_path, string destination_path) — функция разархивации файла на основе алгоритма lz77. Сначала читается 4 байта — количество записаных троек. Далее в цикле читаются 5 байт и трансформируются обратно в тройки ($256 * k_l + b_l$, $256 * k_o + b_o$, ch) и записываются в массив vector<Node>, который потом декодируется с помощью алгоритма lz77. Итоговая сложность – O(n), n – количество троек.

Описание плана эксперимента

Для проведения экспериментов был создан class experiments.

Столбцы .csv файлов

1) Замеры времени: “Тип алгоритма”, “Имя файла”, “Время архивации”, “Время разархивации”.
2) Замеры коэфициентов сжатия: “Тип алгоритма”, “Имя файла”, “Коэфициент сжатия”
3) Такой формат дает гибкость и удобство при обработки этих данных в дальнейшем, в том числе возможность группировки по значениям столбцов.

Основные методы

1. void write_frequency_to_csv(string output_path) — метод, который проходит по всем 36 экспериментальным файлам и считает для каждого файла частоты байт и энтропию файла (в энтропии логарифм по основанию m, m – количество различных байт в файле, что дает значение энтропии для каждого файла от 0 до 1 (могу доказать более менее формально)). Результат записывается в .csv файл в формате: столбцы — имя файла, байт, частотность. Итоговая сложность O(n), n – количество байт во всех файлах.
2. void write_huff(fano/lz77)_to_csv(string compress_path, string coef_path) — метод, который проходит по всем 36 экспериментальным файлам и для каждого по 20 раз вызывает алгоритм архивации и разархивации Хаффмана/Шеннона Фано/ LZ77, замеряет время(в наносекундах) как для архивации так и для разархивации, и потому усредняет по всем 20ти замерам. Также совместно замеряется коэффициент сжатия — архив / исходный файл. Полученные данные записываются в .csv файлы.

Использованные инструменты и технологии

1. Среда разработки CLion 2017.2 (C++ 11)
2. Эксперименты запускались на удаленном сервере в облаке Microsoft Azure
3. Для анализа полученных результатов использовался Python 3 в комбинации с Jupyter Notebooks
4. Библиотеки:
    a. Pandas — построение таблицы с отчетом
    b. Matplotlib + Seaborn — построение графиков

Результаты экспериментов — таблицы и графики

Таблицы и графики находятся в файле GraphsAndTable.pdf.

Сравнительный анализ алгоритмов

Анализ поведения алгоритмов на основании полученных графиков

1. Коэффициент сжатия:
    a. Хаффман и Шеннон-Фано показывают себя более устойчивыми к изменению (увеличению) энтропии файла. Когда энтропия файла стремиться к 1, их коэффициент сжатия тоже стремится к 1 и не выходит за эти пределы, потому что тогда эти алгоритмы просто равномерно кодируют все байты и их количество сохраняется.
    b. LZ77 намного сильнее зависит от энтропии файла, при энтропии стремящейся к 1, он может увеличивать размеры файла в 2-2.5 раза. Причем LZ775 показывает себя хуже всех, из за окна в 5 байт, то есть он не может ухватить даже малейшие последовательности повторяющихся символом в файле с равномерным распределением.
    Однако при энтропии стремящейся к 0, LZ77 намного сильнее сжимает файлы по сравнению с Хаффманом и Шенноном-Фано, потому что количество троек, записываемых в файл, уменьшается пропорционально длине окна.
    c. Также было выяснено, что средний по всем алгоритмам коэффициент сжатия имеет почти линейную зависимость от энтропии файла.
2. Время сжатия
    a. Хаффман и Фано тут работает намного(в 10ки раз) быстрее, чем LZ77, что подтверждает их теоретические сложности
    nlog(n) — у Хаффмана и Фано
    n^3 — у LZ77
    b. При этом Хаффман и Фано работают примерно за одинаковое время, так как для записи кодов файл используется один и тот же метод.
    c. А в случае LZ77 медленнее всего работает LZ7720, потому что ему приходится намного дальше просматривать предыдущие байты.
    d. Также было выяснено, что среднее по алгоритмам время сжатия не зависит ни от энтропии, ни от коэффициента сжатия, но зависит от размера файла.
3. Время разжатия
    a. При разжатии Хаффман и Фано работают в разы дольше, чем LZ77, потому что им приходится перебирать map кодов для поиска подходящего — то есть сложность в худшем случае O(n * log(m)), где m – количество различных байт, n – количество байт в файле (Но так как используется map с сортировкой по длине кодов, в файлах с малой энтропией алгоритмы могут работать значительно быстрее). А у LZ77 сложность в худшем случае O(n), n – количество байт в файле.
    b. Время разжатия как у Хаффмана и Фано, так и LZ77XX в своей категории примерно равное, потому что для Хаффмана и Фано используется один и тот же метод разархивации, и для всех LZ77XX используется единый метод разархивации.

Заключение

1. Была разработана программа на C++, которая реализует 5 алгоритмов сжатия данных без потерь:
    a. Алгоритм Хаффмана (простой)
    b. Алгоритм Шеннона-Фано (простой)
    c. Алгоритмы LZ775, LZ7710, LZ7720
2. Была подготовлена система автоматического тестирования и записи данных и запущена для тестирования на удаленном сервере.
3. Было проведено сравнение алгоритмов:
    a. По коэффициенту сжатия — у LZ77 сжатие сильно зависит от энтропии файла, а Хаффман и Шеннон-Фано являются более устойчивыми.
    b. Времени сжатия — как и в теории Хаффман и Шеннон-Фано(O(nlog(n))) работают быстрее, чем LZ77($O(n^3)$)
    c. Времени разжатия — Хаффман и Шеннон-Фано работают медленнее, чем LZ77. Однако была проведена оптимизация процесса за счет отсортированного map.