Курсовой проект по курсу дискретного анализа: Архиватор. Кодирование по Хаффману

Выполнил студент группы М8О-301Б-21 МАИ *Тулин Иван.*

Условие

Формат запуска должен быть аналогичен формату запуска программы gzip. Должны быть поддерживаться следующие ключи: -c, -d, -k, -l, -r, -t, -1, -9. Должно поддерживаться указание символа дефиса в качестве стандартного ввода.

Метод решения

Идея кодирования по Хаффману заключается в том, чтобы задать символы таким образом, чтобы наиболее часто встречающиеся из них имели более короткий код, что позволит сжимать файлы для экономии места на диске. Для того, чтобы декодирование было однозначным, необходимо, чтобы код одного символа не являлся префиксом кода другого. Для обеспечения выполнения обоих этих условий используется двоичное дерево, в листьях которого хранятся символы. Другой особенностью дерева является его свойство, что каждый узел имеет параметр частоты, полученный путем суммирования частот детей этого узла. Листья дерева, содержащие в себе символы исходного текста, имеют параметр частоты равный частоте вхождений этих символов в текст. В программе реализован полустатический вариант алгоритма, где посредством двух массивов линейным образом строится дерево частот, такое что вершины упорядочены по убыванию частоты, что позволяет коротким кодам соответствовать символам с наибольшей частотой.

Описание программы

Программа состоит из 3ех файлов.
Файл *huffman.hpp* является реализацией описанного выше алгоритма. Он сожержит основной класс *HuffmanTree*, который имеет 2 конструктора. Первый конструктор используется для архивации файлов, он строит дерево на основе исходного текста. В это время второй конструктор уже нужен при разархивации данных, он получает сериализованное дерево и десериализует его. Так или иначе в результате работы обоих конструкторов мы получаем дерево частот и на его основе генерируем коды для каждого из символов. Прямая и обратная архивация в программе происходит посредством функций *huffmanIn* и *huffmanOut* соответственно. Для работы достаточно вызвать нужную функцию с названием обрабатываемого файла в качестве аргумента.
Во втором файле *lz77.hpp* была перенесена реализация алгоритма lz77 из первой части курсовой.
В файле *arch.cpp* производится манипуляция представленнми алгоритмами и описывается реализация ключей утилиты *gzip*.

Флаги

1. Без флагов — по умолчанию ко входному файлу применяется только сжатие с помощью кодов Хаффмана:
   ./arch test.txt
   При этом в директории появится сжатый файл *text.arc* и файл с ключами *text.keys*.

2. Флаг -1 — установлен по умолчанию, если нет флага -9.

3. Флаг -9 — входной файл сначала кодируется в триплеты алгоритмом LZ77, потом к полученному файлу применяется сжатие с помощью кодов Хаффмана:
   ./arch -9 test.txt

4. Флаг -c — вместо сохранения в новый файл результат сжатия посимвольно выводится в стандартный поток вывода.

5. Флаг -d — распаковать входной файл (если при сжатии указывался флаг -9, его необходимо указать)
   ./arch -d text.arc или ./arch -9 -d text_lz77.arc

6. Флаг -k — сохранить входной файл (без этого флага входной файл удаляется):
   ./arch -k text.txt

7. Флаг -l — вывести в стандартный поток вывода информацию о сжатии: размер сжатого файла, размер исходного файла, степень сжатия и название файла.

8. Флаг -r — на вход подаётся название директории, после чего эта директория рекурсивно обходится, и сжатие применяется к каждому найденному в ней файлу. Вместе с флагом -d эту директорию можно точно также рекурсивно распаковать.

9. Флаг -t — проверить на корректность сжатый файл:
   ./arch -t text.arc

Дневник отладки

1. Сначала был написан и протестирован класс *HuffmanTree* и прилагающиеся к нему функции *huffmanIn* и *huffmanOut*

2. При тестировании было обнаружено, что чтение программой бинарного файла со сжатым тексом вызывало следущие баги: если длина закодированного текста не была кратна размеру байта, в конце его оставались лишние нули. Кроме того при чтении пропадали байты, коды которых совпадали с символами-разделителями в тексте.

3. Для исправления первой ошибки в начало файла пришлось записывать длину в битах закодированного файла, чтобы обрезать лишние биты. Вторая ошибка была исправлена путем ипользования метода get() библиотеки fsrteam. После исправлений ошибок первая часть программы стала работать корректно.

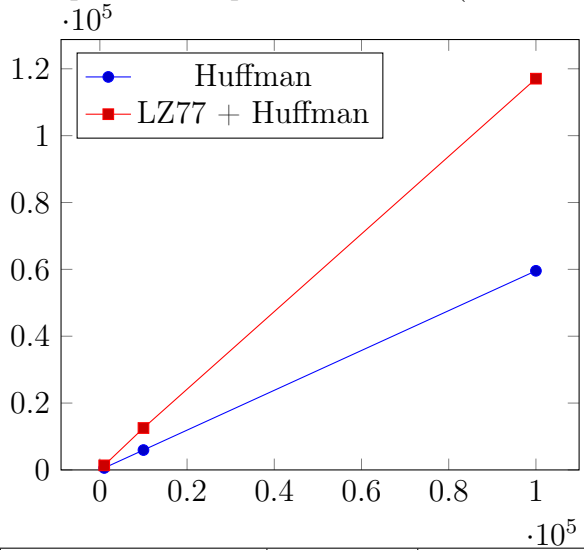
4. Была добавлена и адаптирована реализация алгоритма LZ77.

5. Добавлена поддержка всех флагов (при реализации флагов -l и -t пригодился размер файлов, записанный в их начале, кроме того пришлось записывать размеры файлов до архивации).

Тестирование

Тест сжатия текстового файла

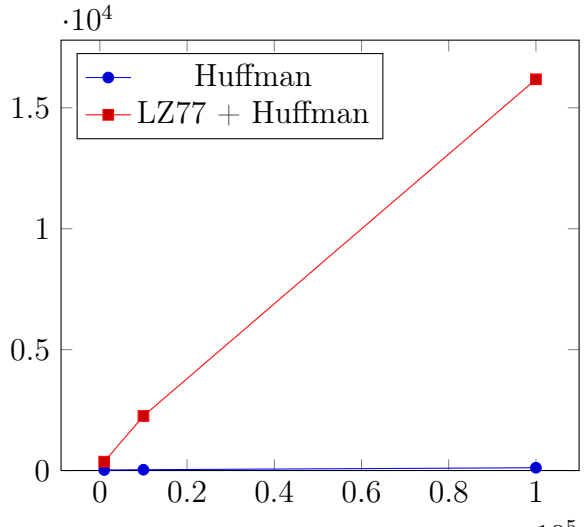
Протестируем эффективность сжатия текстовых файлов различного размера, содержащих только латинские буквы. По оси X — размер исходного файла в байтах, по оси Y — размер сжатого файла в байтах (меньше — лучше).



Размер в байтах	Huffman	LZ77 + Huffman
1000	588	1370
10000	5947	12538
100000	59565	117046

Тест времени сжатия текстового файла

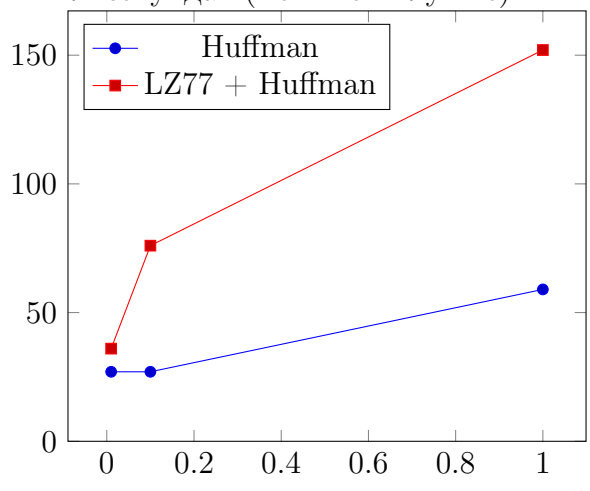
По оси X — размер исходного файла в байтах, по оси Y — время работы программы в милисекундах (меньше — лучше).



Размер в байтах	Huffman	LZ77 + Huffman
1000	22	368
10000	34	2256
100000	116	16181

Тест времени распаковки текстового файла

По оси X — размер распакованного файла в байтах, по оси Y — время работы программы в милисекундах (меньше — лучше).



Размер в байтах	Huffman	LZ77 + Huffman
1000	27	36
10000	27	76
100000	59	152

Из проведённых выше тестов видно, что алгоритм LZ77 мало того, что намного увеличивает время выполнения сжатия, так ещё и не даёт никакого сжатия. Это связано с неэффективным хранением триплетов в файле. Тем не менее удалось реализовать эффективный алгоритм сжатия текстовых файлов с помощью кодов Хаффмана, сжатые

с помощью него файлы занимают в среднем 60% места от размера исходных.

Временные сложности

1. Сложность построения дерева кодов Хаффмана: $O(k^2)$, где k — количество различных символов во входном файле (в реальных тестах можно считать константной, так как $k \leq 256$).

2. Сложность архивации/деархивации с помощью кодов Хаффмана: $O(n)$ (поиск кода символа считаем константным).

3. Сложность архивации с помощью алгоритма LZ77: $O(n^3)$.

4. Сложность деархивации с помощью алгоритма LZ77: $O(n)$, где n — количество символов в исходном тексте.

Недочёты

Не удалось реализовать эффективный способ хранения триплетов для алгоритма LZ77, из-за чего использование флага -9 не имеет никакого смысла.

Выводы

В ходе выполнения работы я реализовал алгоритм архивации с помощью кодов Хаффмана и алгоритм LZ77, протестировал их проанализировал их преимущества и недостатки.