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

Выполнил студент группы М8О-308Б-20 МАИ *Попов Матвей*.

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

Основная идея кодирования по Хаффману — закодировать символы так, чтобы наиболее часто встречающиеся символы имели самый короткий код, ведь так сжатый файл будет занимать меньше всего места на диске. Также для того, чтобы декодирование было однозначным, необходимо, чтобы код любого символа не являлся префиксом кода для другого символа. Чтобы соблюсти оба эти условия, было реализовано двоичное дерево, в листах которого хранятся символы (это обеспечивает выполнение второго условия), а также у каждого символа есть параметр частоты, причём после прочтения каждого символа дерево перестраивается так, чтобы при обходе в ширину все вершины были расположены в порядке невозрастания частоты. Это необходимо, чтобы символы с наибольшей частотой располагались сверху, путь от корня к ним будет короче, а значит их код будет наиболее коротким.

Описание программы

В программе реализованы 2 основных класса *TCompressTree* и *TDecompressTree*, первый класс отвечает за построение дерева, сжатие исходного файла и сохранение файла с парами «символ-код» для последующего декодирования, второй класс с помощью файла с парами строит дерево для декодирования сжатого файла, а потом распаковывает этот файл. Для работы обоих классов достаточно вызвать лишь их конструктор. При этом необходимо, чтобы и сжатый файл, и файл с ключами находились в одной директории с запускаемой программой. Также в программу из предыдущей работы была перенесена реализация алгоритма LZ77.

Флаги

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

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:
./main.o -9 test.txt
При этом в директории появится сжатый файл *text.txt.lz77.huffman* и флаг с ключами *text.txt.lz77.key*.

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

5. Флаг -d — распаковать входной файл, помимо этого необходимо указать тот же флаг (-1 или -9), который указывался при сжатии:
./main.o -d text.txt.huffman
При этом в директории появится распакованный файл *decompressed_ text.txt*.
Важно: для работы программы необходимо, чтобы в этой же директории находился файл с ключами *text.txt.key*.

6. Флаг -k — сохранить входной файл (без этого флага входной файл удаляется):
./main.o -k text.txt
При этом в директории останется файл *text.txt* и появятся файлы *text.txt.huffman* и *text.txt.key*.

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

8. Флаг -r — на вход подаётся название директории, после чего эта директория рекурсивно обходится, и сжатие применяется к каждому найденному в ней файлу.

9. Флаг -t — проверить на корректность файл с кодами Хаффмана:
./main.o -t text.txt.key

Флаги -d, -l, -k, -1, -9 поддерживают работу с несколькими входными файлами.

Дневник отладки

1. Сначала был написан и протестирован класс *TCompressTree*.

2. Затем был написан класс *TDecompressTree*, но в их работе была ошибка.

3. Выяснилось, что ошибка возникала из-за некорректной работы функции перевода из двоичной системы счисления в десятичную, после исправления функции программа работает корректно.

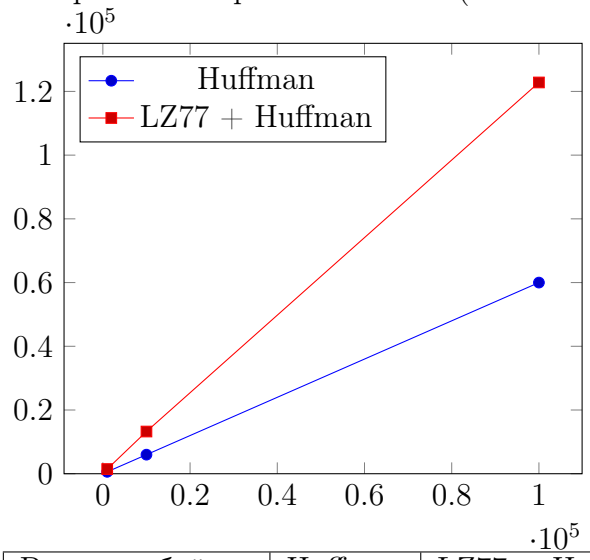
4. Добавлена реализация алгоритма LZ77.

5. Добавлена поддержка всех флагов.

Тестирование

Тест сжатия текстового файла

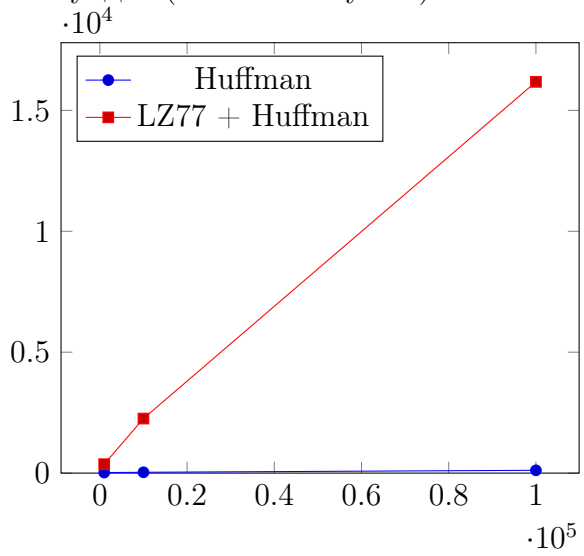
Протестируем эффективность сжатия текстовых файлов различного размера, содержащих только латинские буквы. По оси X — размер исходного файла в байтах, по оси Y — размер сжатого файла в байтах (меньше — лучше).



Размер в байтах	Huffman	LZ77 + Huffman
1000	597	1468
10000	5980	13244
100000	59996	122810

Тест времени сжатия текстового файла

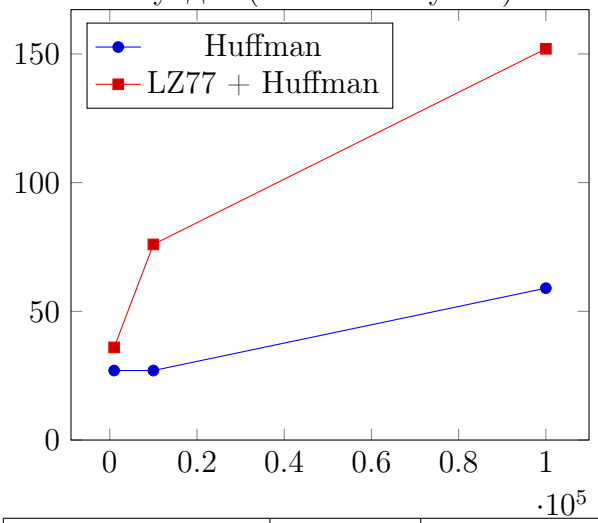
По оси X — размер исходного файла в байтах, по оси Y — время работы программы в милисекундах (меньше — лучше).



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).



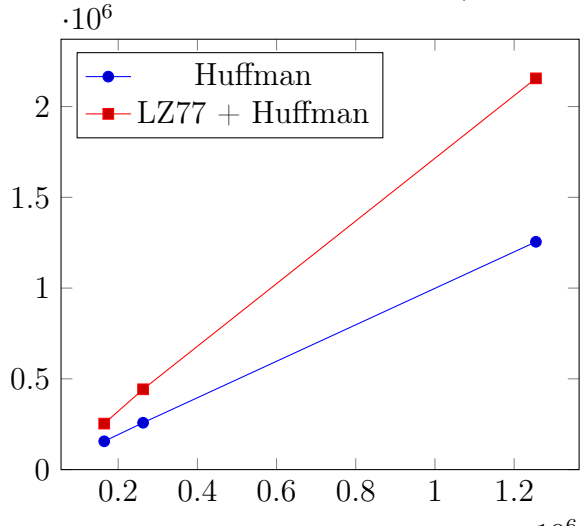
Размер в байтах	Huffman	LZ77 + Huffman
1000	27	36
10000	27	76
100000	59	152

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.

Теперь проведём тестирование сжатия фотографий. Для тестов использовалась приведённая ниже фотография:

Тест сжатия фотографии

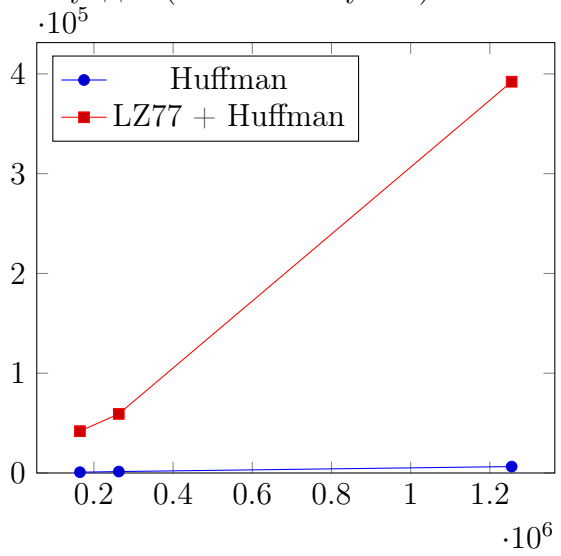
Протестируем эффективность сжатия текстовых файлов различного размера, содержащих только латинские буквы. По оси X — размер исходного файла в байтах, по оси Y — размер сжатого файла в байтах (меньше — лучше).



Размер в байтах	Huffman	LZ77 + Huffman
164703	156025	253494
262827	258117	442526
1255046	1254869	2155391

Тест времени сжатия фотографии

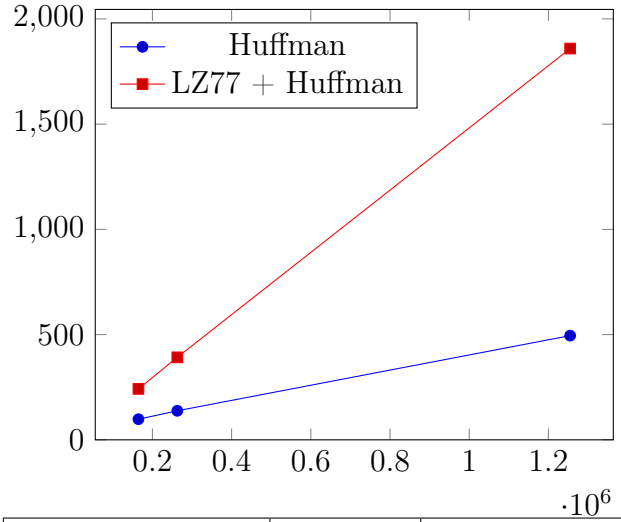
По оси X — размер исходного файла в байтах, по оси Y — время работы программы в милисекундах (меньше — лучше).



Размер в байтах	Huffman	LZ77 + Huffman
164703	805	41938
262827	1436	59222
1255046	6438	392153

Тест времени распаковки фотографии

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



Размер в байтах	Huffman	LZ77 + Huffman
164703	98	242
262827	138	392
1254869	495	1859

Реализация алгоритма LZ77 всё ещё неэффективна, также при сжатии фотографии был обнаружен недостаток кодов Хаффмана: если все символы в файле встречаются примерно с одинаковой частотой, средняя длина кодов Хаффмана будет стремиться к 8, а значит сжатие будет менее эффективным. Тем не менее, сжатые с помощью кодов Хаффмана файлы занимают всё ещё меньше места, чем исходные.

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.