Отчет по дисциплине «Алгоритмы и структуры данных» КДЗ №1

Выполнил:

Студент БПИ154 ФКН группы

Истомин Иван Викторович

Оглавление

Постановка задачи

Реализовать с использованием языка C++ программы для архивирования и разархивирования текстовых файлов. При этом использовать два известных алгоритма кодирования информации:

1. Хаффмана,
2. Шеннона-Фано.

Описание алгоритмов и использованных структур данных

Алгоритм Хаффмана

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (H-дерево).

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Алгоритм Шеннона-Фано

Алгоритм префиксного неоднородного кодирования. Относится к вероятностным методам сжатия (точнее, методам контекстного моделирования нулевого порядка).
Подобно алгоритму Хаффмана, алгоритм Шеннона — Фано использует избыточность сообщения, заключённую в неоднородном распределении частот символов его (первичного) алфавита, то есть заменяет коды более частых символов короткими двоичными последовательностями, а коды более редких символов — более длинными двоичными последовательностями.

1. Символы первичного алфавита m1 выписывают по убыванию вероятностей.
2. Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
4. Полученные части рекурсивно делятся и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Структуры данных

При реализации приложения были использованы следующие структуры данных:

1. Из стандратной библиотеки STL:
    1. `vector`
    2. `map`
    3. `priority_queue`
2. А также были реализованы собственные структуры:

1.
```
struct Letter {
        char32_t letter;
        int freq;
        string code = "";

        Letter(char32_t letter, int freq) {
            this->letter = letter;
            this->freq = freq;
        }
};
```

2.
```
struct Node {
        char32_t data;
        int freq;
        Node* left, * right;

        Node(char32_t data, int freq) {
            left = right = nullptr;
            this->data = data;
            this->freq = freq;
        }

        Node(char32_t data, Node* left, Node* right) {
            this->data = data;
            this->left = left;
            this->right = right;
        }

        bool isLeafNode() {
            return left == nullptr;
        };
};
```

Описание плана эксперимента

В качестве тестирования автоматизации тестирования, был написать bash скрипт :

```
#!/bin/bash
#
# Just write "sh testing.sh" in console
#

sizes=( 20 40 60 80 100 1 2 3 )
sub_size=( 1 2 3 )
sets=( FirstSet SecondSet ThirdSet )
work_directories=( archive extract logs )
```

```
# Delete all garbage
echo "Delete all garbage";
for dir in "${work_directories[@]}"
do
    for set in "${sets[@]}"
    do
        echo "Delete all files in ${dir}/${set}"
        rm -r ${dir}/${set}/*
    done
done

# Huffman Archive
for set in "${sets[@]}"
do
    echo "Huffman Archive ${set}."

    for size in "${sizes[@]}"
    do
        for sub in "${sub_size[@]}"
        do
            echo "Starting ${size}_${sub} test..."
            ./ArchiveAlgorithm_HW -h sets/${set}/${size}_${sub}.txt >>
logs/${set}/${size}_${sub}_h_log.txt

            echo "Move ${size}_${sub}.haff to
archive/${set}/${size}_${sub}.haff"
            mv ${size}_${sub}.haff archive/${set}/
        done
    done
done

# Huffman Extract
for set in "${sets[@]}"
do
    echo "Huffman Extract ${set}:"

    for size in "${sizes[@]}"
    do
        for sub in "${sub_size[@]}"
        do
            echo "Starting ${size}_${sub} test..."
            ./ArchiveAlgorithm_HW -a archive/${set}/${size}_${sub}.haff >>
logs/${set}/${size}_${sub}_unz_h_log.txt

            echo "Move ${size}_${sub}-unz-h.txt to extract/${set}/"
            mv ${size}_${sub}-unz-h.txt extract/${set}/
        done
    done
done
```

```
# Shannon-Fano Archive
for set in "${sets[@]}"
do
    echo "Shannon-Fano Archive ${set}."

    for size in "${sizes[@]}"
    do
        for sub in "${sub_size[@]}"
        do
            echo "Starting ${size}_${sub} test..."
            ./ArchiveAlgorithm_HW -s sets/${set}/${size}_${sub}.txt >>
logs/${set}/${size}_${sub}_s_log.txt

            echo "Move ${size}_${sub}.shan to
archive/${set}/${size}_${sub}.shan"
            mv ${size}_${sub}.shan archive/${set}/
        done
    done
done

# Shannon-Fano Extract
for set in "${sets[@]}"
do
    echo "Shannon-Fano Extract ${set}:"

    for size in "${sizes[@]}"
    do
        for sub in "${sub_size[@]}"
            do
            echo "Starting ${size}_${sub} test..."
            ./ArchiveAlgorithm_HW -u archive/${set}/${size}_${sub}.shan >>
logs/${set}/${size}_${sub}_unz_s_log.txt

            echo "Move ${size}_${sub}-unz-s.txt to extract/${set}/"
            mv ${size}_${sub}-unz-s.txt extract/${set}/
        done
    done
done
```

Далее, в директории `logs` получим файлы с количеством операций, для дальнейших вычислений.

Результаты экспериментов

Процент сжатия ~45%.

Количество операций (**H** — Huffman, **SF** — Shannon-Fano, Архивирование/Разархивирование):

	First Set	Second Set	Third Set
H 20 кб	948 177 / 1 136 006	748 185 / 892 227	813 646 / 969 842
H 40 кб	1 875 233 / 2 254 383	1 441 846 / 1 734 585	1 564 230 / 1 882 884
H 60кб	2 760 538 / 3 322 394	2 141 729 / 2 584 519	2 309 997 / 2 790 200
H 80	3 682 981 / 4 435 215	2 832 585 / 3 423 487	3 069 996 / 3 714 732
H 100	4 609 985 / 5 553 519	3 517 799 / 4 255 610	3 819 503 / 4 626 434
H 1	46 375 121 / 55 938 087	34 954 470 / 42 431 797	39 288 173 / 47 777 294
H 2	92 729 431 / 111 858 943	68 035 343 / 82 604 807	77 541 113 / 94 316 069
H 3	139 082 695 / 167 778 469	106 110 392 / 128 842 873	115 496 305 / 140 492 154
SF 20	1 653 513 / 1 137 072	1 646 281 / 894 324	1 923 888 / 971 856
SF 40	3 302 851 / 2 257 173	3 216 030 / 1 739 916	3 785 110 / 1 887 209
SF 60	4 868 035 / 3 322 395	4 861 623 / 2 588 258	5 666 670 / 2 795 019
SF 80	6 499 880 / 4 435 216	6 467 104 / 3 428 685	7 559 299 / 3 721 275
SF 100	8 141 596 / 5 553 520	8 054 034 / 4 262 054	9 394 094 / 4 635 204
SF 1	82 398 739 / 55 938 088	80 647 992 / 42 460 492	98 674 205 / 47 806 961
SF 2	164 862 688 / 111 858 944	157 430 048 / 82 662 326	195 107 768 / 94 353 105
SF 3	247 326 505 / 167 778 470	246 238 247 / 128 842 874	290 734 699 / 140 545 829

На количество операций не влияет порядок одних и тех же символов в файле, что, в общем-то логично и доказывается между попытками тестов 1 и 2, где символы просто перемешаны, но их количество и порядок сохранены.

Сравнительный анализ методов

Для начала проверим размеры заархивированных файлов первой подборки:

```
➜  FirstSet ls -l
-rw-r--r--  1 ivanistomin  staff    15376 20_1.haff
-rw-r--r--  1 ivanistomin  staff    15392 20_1.shan (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    20199 20_1.txt

-rw-r--r--  1 ivanistomin  staff    30485 40_1.haff
-rw-r--r--  1 ivanistomin  staff    30527 40_1.shan (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    40399 40_1.txt

-rw-r--r--  1 ivanistomin  staff    44913 60_1.haff
-rw-r--r--  1 ivanistomin  staff    44913 60_1.shan
-rw-r--r--  1 ivanistomin  staff    59690 60_1.txt

-rw-r--r--  1 ivanistomin  staff    59947 80_1.haff
-rw-r--r--  1 ivanistomin  staff    59947 80_1.shan
-rw-r--r--  1 ivanistomin  staff    79789 80_1.txt

-rw-r--r--  1 ivanistomin  staff    75055 100_1.haff
-rw-r--r--  1 ivanistomin  staff    75055 100_1.shan
-rw-r--r--  1 ivanistomin  staff    99989 100_1.txt

-rw-r--r--  1 ivanistomin  staff   755729 1_1.haff
-rw-r--r--  1 ivanistomin  staff   755729 1_1.shan
-rw-r--r--  1 ivanistomin  staff  1009999 1_1.txt

-rw-r--r--  1 ivanistomin  staff  1511196 2_1.haff
-rw-r--r--  1 ivanistomin  staff  1511196 2_1.shan
-rw-r--r--  1 ivanistomin  staff  2019999 2_1.txt

-rw-r--r--  1 ivanistomin  staff  2266644 3_1.haff
-rw-r--r--  1 ivanistomin  staff  2266644 3_1.shan
-rw-r--r--  1 ivanistomin  staff  3029999 3_1.txt
```

И сразу третьей подборки:

```
➜  ThirdSet ls -l
-rw-r--r--  1 ivanistomin  staff    13421 20_1.haff
-rw-r--r--  1 ivanistomin  staff    13451 20_1.shan  (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    20205 20_1.txt

-rw-r--r--  1 ivanistomin  staff    26006 40_1.haff
-rw-r--r--  1 ivanistomin  staff    26072 40_1.shan  (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    40269 40_1.txt

-rw-r--r--  1 ivanistomin  staff    38517 60_1.haff
-rw-r--r--  1 ivanistomin  staff    38590 60_1.shan  (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    59713 60_1.txt

-rw-r--r--  1 ivanistomin  staff    51262 80_1.haff
-rw-r--r--  1 ivanistomin  staff    51361 80_1.shan  (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    79917 80_1.txt

-rw-r--r--  1 ivanistomin  staff    63827 100_1.haff
-rw-r--r--  1 ivanistomin  staff    63960 100_1.shan (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff    99747 100_1.txt

-rw-r--r--  1 ivanistomin  staff   658787 1_1.haff
-rw-r--r--  1 ivanistomin  staff   659237 1_1.shan   (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff  1036168 1_1.txt

-rw-r--r--  1 ivanistomin  staff  1300479 2_1.haff
-rw-r--r--  1 ivanistomin  staff  1301041 2_1.shan   (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff  2046541 2_1.txt

-rw-r--r--  1 ivanistomin  staff  1937156 3_1.haff
-rw-r--r--  1 ivanistomin  staff  1937969 3_1.shan   (<- here .shan > .haff)
-rw-r--r--  1 ivanistomin  staff  3049417 3_1.txt
```

Стоит сразу обратить внимание на то, что отличия в архивации возрастают с количество символов в файле. Это происходит потому, что алгоритм Шеннона-Фано генерирует префиксные коды длинней префиксных кодов Хаффмана.

Сложность *алгоритма Хаффмана* $O(n \log n)$, из-за использования кучи

Сложность *алгоритма Шеннона-Фано* $O(n \log n)$, из-за сортировки.

Заключение

Алгоритм Шеннона-Фано является частным алгоритма Хаффмана. Как мы смогли убедиться, с помощью алгоритма Хаффмана можно получить практически идеальное префиксное кодирование, по сравнению с его предшественником, что дает нам возможность лучше сжимать информацию.

Источники

[1] Новиков Ф. А. Дискретная математика для программистов: Учебник для вузов. 3-е изд. —
СПб.: Питер, 2009. ISBN 978-5-91180-759-7

[2] ifstream - Read file by bits c++ - Stack Overflow:
http://stackoverflow.com/questions/22390641/read-file-by-bits-c

[3] binaryfiles - writing bits into a c++ file - Stack Overflow:
http://stackoverflow.com/questions/28500047/writing-bits-into-a-c-file

[4] A quick tutorial on generating a huffman tree:
https://www.siggraph.org/education/materials/HyperGraph/video/mpeg/mpegfaq/huffman_tutorial.html

[5] Алгоритм Хаффмана — Викиконспекты: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0

[6] Код Хаффмана — Википедия:
https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0

[7] Метод Шеннона-Фано: http://www.compression.ru/download/articles/huff/tiger_shannon-fano.html

[8] Алгоритм Шеннона — Фано — Википедия:
https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%B5%D0%BD%D0%BD%D0%BE%D0%BD%D0%B0_%E2%80%94_%D0%A4%D0%B0%D0%BD%D0%BE