Программирование на языке Си

Практикум: Компрессор по методу Хаффмана (Часть 1)

Штанюк А.А.

30 января 2014 г.

Введение

Завершающие практикумы курса посвящены разработке файлового компрессора по методу Хаффмана. Задача разбита на три части:

I) Разработка блока частотного анализа входного файла и блока генератора кодов.

II) Разработка блока кодирования и упаковки (сжатия) данных.

III) Разработка формата заголовка сжатого файла, тестирование программы, написание документации.

Построение компрессора предполагает создание программы, выполняющей обратную операцию - декомпрессию сжатого файла. Во многих случаях декомпрессор является функцией компрессора, а не отдельной программой.

Сжатие информации

Сжатие файлов основано на использовании более экономного способа кодирования информации.

Обычно, содержимое файла представлено в кодировке **ASCII**:

Символ	**ASCII-код**
Символ 1	00000000
Символ 2	00000001
Символ 3	00000010
...	...
Символ 256	11111111

В результате, каждый символ занимает одно и то же пространство памяти, независимо от частоты появления в файле. Такой способ необходим, если все символы встречаются одинаковое количество раз (поровну).

Энтропия

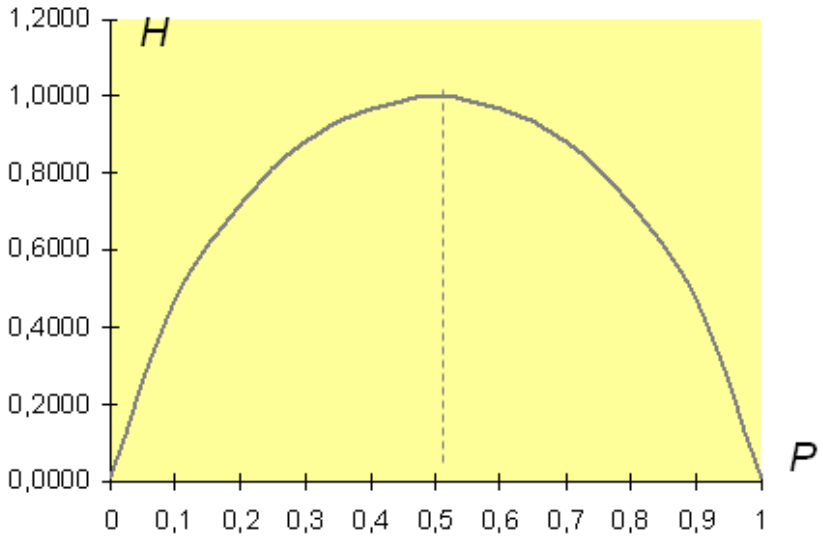
Для числовой характеристики содержимого файла используется *математическое ожидание* количества информации в отдельных сообщениях (символах). Это математическое ожидание называется энтропия.

$$H(U) = \sum_{i=1}^{N} P(u_i) log \frac{1}{P(u_i)}$$

где $P(u_i)$ - вероятность появления символа u_i в анализируемом файле.

Чем больше энтропия файла, тем большее количество информации содержится в одном символе и тем больше информации во всём файле.

Свойства энтропии



1. Энтропия неотрицательна. Она равна нулю, если файл содержит только один символ с вероятностью 1.
2. Максимально возможное значение энтропии файла с объемом алфавита N равно $logN$ и достигается в том случае, когда все символы равновероятны.
3. Энтропия объединения нескольких независимых файлов равна сумме энтропии файлов - свойство *аддитивности* энтропии.

Избыточность

Таким образом, энтропия файла максимальна в том случае, когда выполняются 2 условия:

1. все символы независимы (что справедливо для произвольного файла).
2. все символы равновероятны.

Невыполнение любого из этих требований уменьшает энтропию и является причиной избыточности. Избыточностью файла с энтропией H и объемом алфавита N называется величина

$$\mu = \frac{H_{max} - H}{H_{max}} = 1 - \frac{H}{H_{max}} = 1 - \frac{H}{logN}$$

Вероятности появления русских букв

В реальных условиях, символы, содержащиеся в файле, имеют разные вероятности появления (частоты встречаемости). Например, существует таблица частот для русских букв:

пробел 0.175	О 0.090	Е, Ё 0.072	А 0.062
И 0.062	Т 0.053	Н 0.053	С 0.045
Р 0.040	В 0.038	Л 0.035	К 0.028
М 0.026	Д 0.025	П 0.023	У 0.021
Я 0.018	Ы 0.016	З 0.016	Ь, Ъ 0.014
Б 0.014	Г 0.013	Ч 0.012	Й 0.010
Х 0.009	Ж 0.007	Ю 0.006	Ш 0.006
Ц 0.004	Щ 0.003	Э 0.003	Ф 0.002

Из таблицы видно, что, например, самой популярной буквой русского языка является **О**, а самой редкой **Ф**.

Диаграмма частот

На основании таблицы можно построить диаграмму частот символов

Идея компрессора

Идея компрессора состоит в том, чтобы заменить стандартные равномерные ASCII-коды на неравномерные таким образом, чтобы часто всречающимся символам соответствовали короткие кодовые последовательности, а редко встречающимся - более длинные. Если средняя длина кода будет меньше 8 бит, то файл сжать удасться.

Неравномерный код должен быть префиксным, то есть ни один код не может быть началом другого, иначе при декодировании возникнет неоднозначность.

Пример плохого кода:

Символ	**Код**
Символ 1	0
Символ 2	1
Символ 3	00
...	...

Пример префиксного кода

Самым простейшим примером префиксного кода является следующий:

Символ	**Код**
Символ 1	0
Символ 2	10
Символ 3	110
...	...

Недостаток его очевиден: длина кодовой последовательности линейно возрастает с позицией символа в таблице.

Поэтому, требуются специальные алгоритмы, которые позволяют по таблице частот получить префиксные коды, которые наиболее оптимальны для заданного частотного распределения. Одним из самых известным является алгоритм Хаффмана.

Схема алгоритма

Рассмотрим работу алгоритма на примере:

Пусть необходимо сжать файл следующего содержания (в файле встречаются только 4 символа: **'x'**, **'y'**, **'z'** и **'.'**)

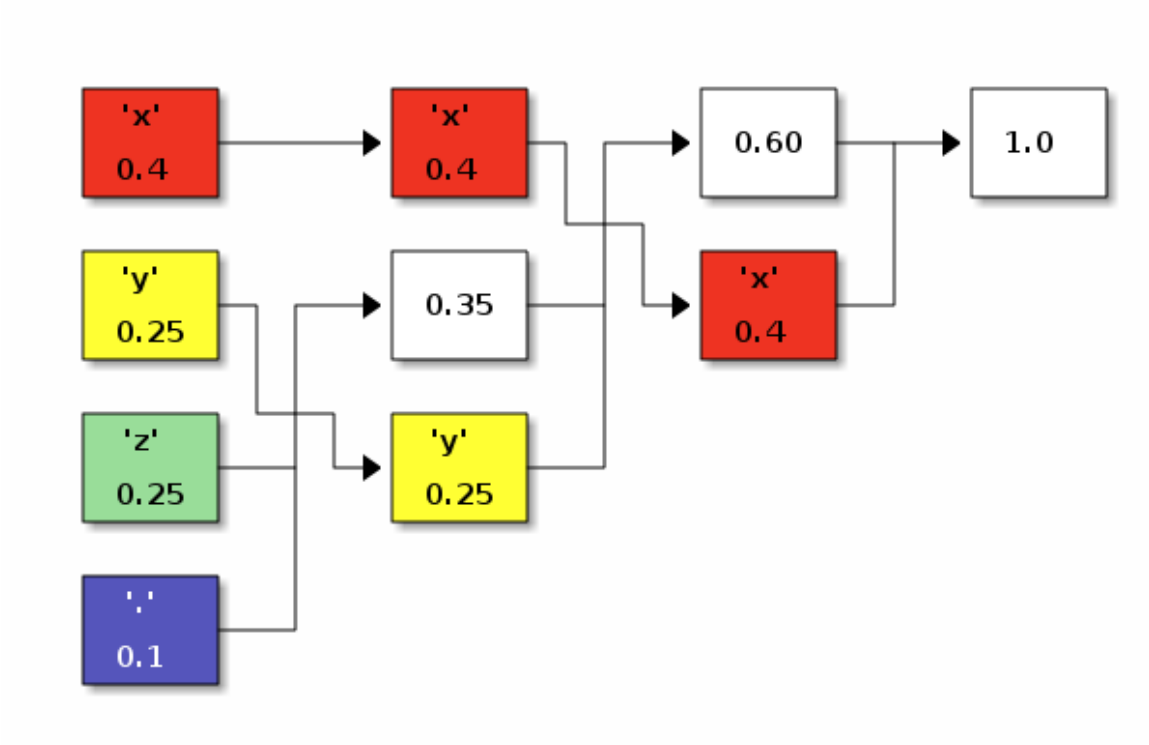
```
zzxz.yzyxxzxy.yzxxxzy.yx.
zxyyxzyx.xyzyxxzzyyz.xzxy
.xxzxyxxzxyyz.yxxzxyyzyxx
zx.zzxyyyxzxxxyxx.zxxxzxx
```

Размер анализируемого файла $100x8 = 800$ бит. Таблица встречаемости принимает вид:

Символ	**Частота**
'x'	0.4
'y'	0.25
'z'	0.25
'.'	0.1

Схема алгоритма

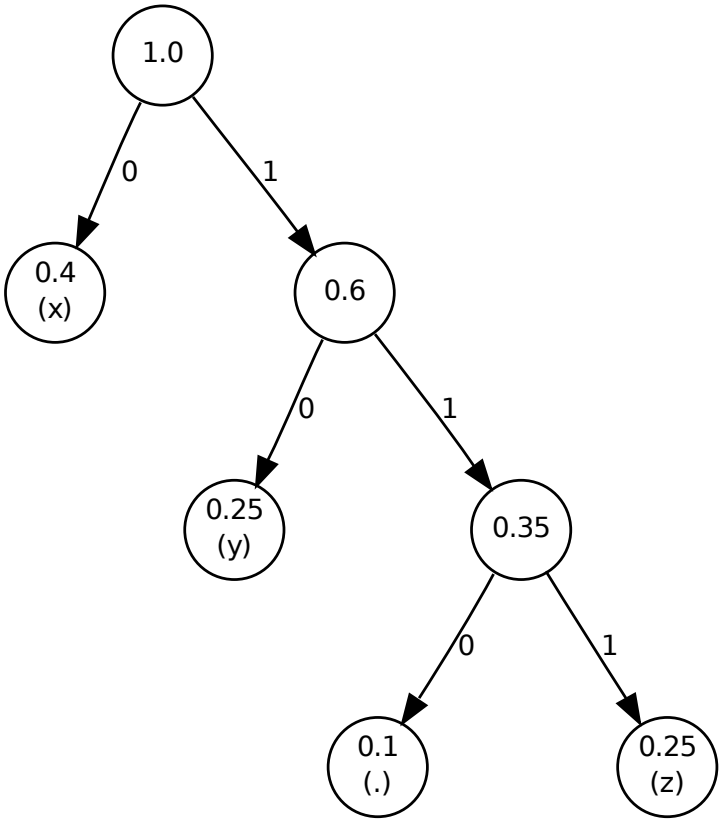
На вход алгоритма подаётся таблица встречаемости символов файла, отсортированная по убыванию частоты.



Работа алгоритма заключается в том, что на основе двух последних элементов таблицы создаётся новый элемент с их суммарной частотой и затем он помещается в таблицу так, чтобы сохранялось убывание по частоте элементов. Алгоритм работает до появления единственного элемента с частотой 1.0 (на схеме справа).

Дерево Хаффмана

В результате работы алгоритма мы получаем бинарное дерево, в котором листьями выступают исходные структуры с символами, а узлы с суммарной частотой связывают их. Двигаясь в дереве от корня к листам и используя '0' и '1' для обозначения перехода, получим префиксные коды для каждого символа.



Результат работы алгоритма

Полученное бинарное дерево позволяет получить префиксный код, наиболее подходящий для заданного распределения частот:

Символ	Код
'x'	0
'y'	10
'z'	111
'.'	110

Результат кодирования

В результате кодирования исходного файла получаем закодированную последовательность:

```
11111101111101011110001110101101011100011110110100110111
10101001111001100101111000111111101011110011101011000111
11010001110101011111010001110101011110001110110111111101
01010011100010001101110001110
```

Длина кодовой последовательности - 195 бит, что меньше исходной (800) в 4,1 раза. Это и есть коэффициент сжатия, которого нам удалось достич. На самом деле, этот коэффициент будет немного меньше за счёт добавления в сжатый файл заголовка с таблицей встречемости, необходимой для распаковки сжатого файла.

Блок-схема компрессора

Обозначения:

1 - исходный (сжимаемый) файл;

2 - таблица встречаемости символов, отсортированная в порядке убывания частоты;

3 - таблица символов и префиксных кодов;

4 - закодированный файл, состоящий из символов '0' и '1' (.101);

5 - результирующий (сжатый) файл.

Описание работы компрессора

- Входной файл подвергается статистическому анализу, в результате чего образуется таблица встречаемости символов
- Таблица встречаемости, отсортированная по частоте подаётся на вход генератора кода. Генератор кода строит дерево Хафманна и определяет префиксный код для представления всех символов в таблице встречаемости
- Исходный файл читается побайтно и на основании таблицы с кодами формируется текстовый файл с символами '0' и '1', который представляет собой закодированную версию исходного файла
- Файл с '0' и '1' читается по 8 символов, затем по этим значениям формируется 1 байт, разряды которого равны либо 0 либо 1. данная операция называется упаковкой

Описание работы компрессора

- Формируется заголовок сжатого файла, в котором располагается информация, необходимая для восстановления. Заголовок записывается первым в сжатый файл
- Вслед за заголовком в сжатый файл дописываются байты, возникшие в результате упаковки
- Если размер файла с '0' и '1' не кратен 8, то рзряды в последнем байте дополняются 0 и в заголовок записывается количество значащих разрядов последнего байта (длина "хвоста")

Представление символа

Наиболее важной структурой данных в компрессоре является структура **SYM**, которая используется при построении дерева Хафманна

```
struct SYM // представление символа
{
   unsigned char  ch; // ASCII-код
   float         freq; // частота встречаемости
   char     code[256]; // массив для нового кода
   struct SYM  *left; // левый потомок в дереве
   struct SYM  *right; // правый потомок в дереве
};
```

ch - поле, в котором хранится ASCII-код символа. Диапазон ведичин от 0 до 255
freq - частота встречаемости данного символа в файле как результат деления количества данного символа на суммарное количество символов. Диапазон от 0.0 до 1.0
code - символьный массив для хранения префиксного кода в виде строки из '0' и '1'
left,right - указатели для связи узла с нижележащими в дереве

Блок анализа входного файла

Блок анализа входного файла (БАВФ) строит таблицу встречаемости символов и сортирует её по частоте. Информация о символах хранится в массиве структур типа **SYM**. Максимальный размер этого массива - 256 (по максимальному количеству символов согласно ASCII-таблице).

При создании массива структур **SYM** необходимо позаботиться о том, чтобы указатели **left** и **right** были равны нулю, а в массиве **code** содержалась пустая строка.

Массив структур **SYM** может быть статическим или динамическим.

Кроме массива структур **SYM** в БАВФ необходимо создать массив из указателей на структуру **SYM** и заполнить его адресами структур **SYM**

- **syms** - массив структур
- **psysms** - массив указателей на структуры

Процедура построения дерева

Данная процедура реализует алгоритм Хаффмана и использует рекурсию

```
struct SYM* buildTree(struct SYM *psym[], int N)
{
   // создаём временный узел
   struct SYM *temp=(struct SYM*)malloc(sizeof(struct SYM));
   // в поле частоты записывается сумма частот
   // последнего и предпоследнего элементов массива psym
   temp->freq=psym[N-2]->freq+psym[N-1]->freq;
   // связываем созданный узел с двумя последними узлами
   temp->left=psym[N-1];
   temp->right=psym[N-2];
   temp->code[0]=0;
   if(N==2) // мы сформировали корневой элемент с частотой 1.0
     return temp;
   ..
   // добавляем temp в нужную позицию psym,
   // сохраняя порядок убывания частоты
   ..
   return buildTree(psym,N-1);
}
```

Процедура построения дерева

После того, как дерево построено, можно вызывать процедуру получения префиксного кода

```
void makeCodes(struct SYM *root)
{
   if(root->left)
   {
      strcpy(root->left->code,root->code);
      strcat(root->left->code,"0");
      makeCodes(root->left);
   }
   if(root->right)
   {
      strcpy(root->right->code,root->code);
      strcat(root->right->code,"1");
      makeCodes(root->right);
   }
}
```

Процедура обходит дерево, копируя кодовые комбинации от верхних узлов нижним и добавляя "0" или "1" в зависимости от левой или правой ветви.