**Explaination&Instruction***Arithmetic Coding with Scaling*

*By Nguyen Thai Son – 931924*

**EN**

**A, Explaination**

***1, About the main Idea***

This Program uses Arithmetic Coding Algorithm to compress data. Because of the “Floating point” data type, the encoded will be a floating point type with small error due to the limit of memory and struct. That will lead to a big error when we decode, when multiply a float/double some time, the error will be bigger and bigger. To solve this problem, I used the Scaling Metod to return a string of 1 and 0, and that string will be convert to bit type.

We have 3 type of scaling: Low, High and Mid. But in my program, I only used Low and High:

+ if [low;high) is in [0;0.5) then it will be Low scaling. We will expand [low;high) to [2\*low;2\*high) and return a 0 in the tag string. (Tag is the return of encoding phase)

+if[low;high) is in [0;5,1) then it will be High scaling. We will Expand [low;high) to [2\*low-1;2\*high-1) and return a 1 in the tag string.

We will keep doing this Scaling as more as possible in every time we **Encode** a letter of the plain text.

**Example**: We have a string contains only a,b and c: “abc……” to encode. p(a)=0.3; p(b)=0.5; p(c)=0.2. Tag string = “”

Set low:high=0:1;

Then, (0;1) doesn’t need scaling. Encode ‘a’ -> low:high=0:0.3

[0;0.3) is a low scaling, then tag string= “0”, low:high=0:0.6

0:0.6 doesn’t need scaling. Encode ‘b’ -> low:high=0.18:0.48

0.18:0.48 need low scaling, then tag string= “00”, low:high=0.36:0.96

Then encode ‘c’ -> low:high=0.84:0.96, call High scaling, low:high then equal 0:68:0.92, scaling again ->low:high=0.36:0.84, the tag string is “0011”.

Then keep doing until we encode all letters in the string.

The tag string will be converted to bit type(each letter in tag string equal a bit, 8 bits(8 chars in tag string) equal 1 char in the return string(called “tag”). Then write tag to output.

About **Header**, it is the same with Header mode 1 in CanonicalHuffman. I will save bit\_len, the number of time of each char, and the char.

For example, there are 30 ‘a’, 50 ‘b’, and 20 ‘c’. To save 50, we need 6 bits, then 6 is bit\_len, then (011110(30)110010(50)­010100(20)-converted), then “abc”

To **Decode**, first read Header to rebuild Frequency&Interval Table. Then we read the tag, re-convert it to string of ‘0’ and ‘1’.

To start Decode, low:high will be set as 0:1

Then the tag string (for example: “0011”) is equal to a number

(0.00112 = 0.187510), then if that number is in the Interval of letter x, we will add x to return string, remove first element in tag string (left shift), and re-calculate low:high and **ALL** the interval(I used a copy table of Interval table to save the re-calculated). Then of course call Scaling if needed, the return of scaling is put to the end of tag string.

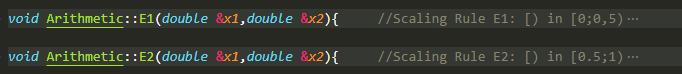
We keep doing all the step until we Decode all the string.

***2, Code explaination***

We will have the class **Arithmetic**, which contain Encode and Decode functions. 

return log2(x)

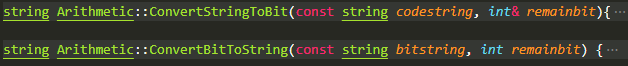
return decimal value of tag string(see **Decode)**



E1: Low, E2: High

Read and save plain data, save number and then calculate Frequency.

Create Table by calculate all [low:high) Interval.

Same as in Huffman class

Call EncodeInput, call CreateTable, PrintTable, create and output Header, encode Data (metod in “About the main Idea). Convert to bit type, save to output)

Read Header, rebuild table, calculate Number of element in Plain text, read Encoded string.

Call DecodeInput, reconvert data string into tag string(string of 0,1), Decode data(metod in “About the main Idea”), save plain text in to ofname file

**B. Instruction**

In the main fuction, you will be asked that you want to Encode or not. Enter Y or y to Encode. Then it’s will asked you to enter the plaintext filename. And then the output filename. It will automatically add **.arth** extension to output filename.

In decode phase, enter Y or y to decode, you have to enter the inputfile (full name, include **.arth)**  
Then enter the file you want to save the result.

**РУ**

**A, Объяснение**

***1, О Главной Идее***

Эта программа использует алгоритм арифметического кодирования для сжатия данных. Из-за типа данных «с плавающей запятой» кодируемый будет тип с плавающей запятой с небольшой ошибкой из-за ограничения памяти и структуры. Это приведет к большой ошибке при декодировании, при умножении числа с плавающей запятой / двойного некоторое время ошибка будет все больше и больше. Чтобы решить эту проблему, я использовал метод масштабирования, чтобы вернуть строку из 1 и 0, и эта строка будет преобразована в битовый тип.

У нас есть 3 типа масштабирования: низкий, высокий и средний. Но в своей программе я использовал только Low и High:

+ если [low; high) находится в [0; 0,5), то будет масштабирование Low. Мы расширим [low; high) до [2 \* low; 2 \* high) и вернем 0 в строке тега. (Тег - возврат фазы кодирования)

+ если [low; high) находится в [0; 5,1), тогда будет масштабирование High. Мы расширим [low; high) до [2 \* low-1; 2 \* high-1) и вернем 1 в строке тега.

Мы будем продолжать делать это масштабирование как можно чаще каждый раз, когда мы кодируем букву простого текста.

Пример: у нас есть строка, содержащая только a, b и c: «abc ……» для кодирования. p (a) = 0,3; p (b) = 0,5; р (с) = 0,2. Строка тега = «»

Установить низкий: высокий = 0: 1;

Тогда (0; 1) не требует масштабирования. Кодировать «а» -> низкий: высокий = 0: 0,3

[0; 0,3) - низкое масштабирование, тогда строка тега = «0», low: high = 0: 0,6

0: 0,6 не требует масштабирования. Кодировать «b» -> низкий: высокий = 0,18: 0,48

0,18: 0,48 необходимо низкое масштабирование, тогда строка тега = «00», low: high = 0,36: 0,96.

Затем закодируйте «c» -> low: high = 0,84: 0,96, вызовите масштабирование High, low: high, затем равное 0: 68: 0,92, снова масштабирование -> low: high = 0,36: 0,84, строка тега будет «0011».

Затем продолжайте, пока мы не закодируем все буквы в строке.

Строка тега будет преобразована в битовый тип (каждая буква в строке тега равна биту, 8 бит (8 символов в строке тега) равны 1 символу в строке возврата (называемой «тег»). Затем напишите тег для вывода.

Что касается заголовка, то это то же самое, что и режим заголовка 1 в CanonicalHuffman. Я сохраню bit\_len, количество раз для каждого символа и символ.

Например, есть 30 «a», 50 «b» и 20 «c». Чтобы сохранить 50, нам нужно 6 бит, затем 6 - это bit\_len, затем (011110 (30) 110010 (50) ¬010100 (20) -конвертировано), затем «abc»

Для декодирования сначала прочтите заголовок, чтобы восстановить таблицу частот и интервалов. Затем мы читаем тег, преобразовываем его в строку «0» и «1».

Чтобы начать декодирование, низкий: высокий будет установлен как 0: 1.

Тогда строка тега (например: «0011») равна числу

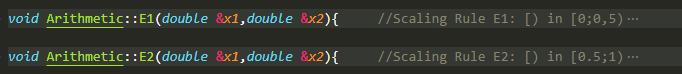
(0,00112 = 0,187510), тогда, если это число находится в интервале буквы x, мы добавим x к возвращаемой строке, удалим первый элемент в строке тега (сдвиг влево) и повторно вычислим low: high и ВСЕ интервал (I использовал таблицу копирования таблицы интервалов для сохранения пересчитанного значения). Затем, конечно, вызовите масштабирование, если необходимо, возврат масштабирования помещается в конец строки тега.

Мы продолжаем делать весь шаг, пока не декодируем всю строку.

***2, Объяснение Кода***

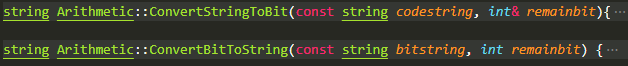
У нас будет класс Arithmetic, который содержит функции кодирования и декодирования.

return log2(x)

 вернуть десятичное значение строки тега (см. Декодирование) 

E1: Low, E2: High

 Прочтите и сохраните простые данные, сохраните число, а затем вычислите частоту. Создайте таблицу, вычислив все [low: high) Interval.

Same as in Huffman class

 Вызвать EncodeInput, вызвать CreateTable, PrintTable, создать и вывести заголовок, закодировать данные (метод в «Об основной идее»). Преобразовать в битовый тип, сохранить для вывода) Прочитать заголовок, перестроить таблицу, вычислить количество элементов в обычном тексте, прочитать закодированную строку. Вызвать ввод декодирования, преобразовать строку данных в строку тега (строка 0,1), декодировать данные (метод в «Об основной идее»), сохранить простой текст в файл имени.

***Б. Инструкция***

В основной функции вас спросят, хотите вы кодировать или нет. Введите Y или y для кодирования. Затем вам будет предложено ввести имя файла в виде открытого текста. И затем имя выходного файла. Он автоматически добавит расширение .arth к имени выходного файла.

На этапе декодирования введите Y или y для декодирования, вы должны ввести входной файл (полное имя, включая .arth)

Затем введите файл, в который хотите сохранить результат.