Static Arithmetic Compression and Decompression

- Varul Srivastava

Abstract

*This article covers Arithmetic Compression, in its **static version.** It is a little less efficient in time than its adaptive counterpart.*

Static Arithmetic Compression is a model of compression that involves mapping the probability of each character occouring from 0 to 1.

We can see that in this mode of compression, the output is an infinite precision floating point number, that provides the information about the entire file, and a stored metadata of probabilities of each character.

I. The Algorithm

In static arithmetic compression, we divide [0,1) range in parts, the lengths of which are in proportion of the probability of occourance of that character. We take 2 pointers, H (abbrv. High) and L (abbrv. Low) that are used to determine the range in which the information lies.

We then read character by character, the file, and move in the range of that character. For the next step, we consider, the new L, H to be our 0, 1 and update the L, H accordingly.

In C++ the updation code looks like:

```
high := low + (range * p[ch].max)
low  := low + (range*p[ch].min)
```

II. Infinite Precision Implementation

This implementation is for infinite precision, that is very efficient when compared to the finite precision floaing point.

The final output is a high precision floating point number, f which is between the last calculated H and L. For saftey, it is outputted as L but in our implementation, because finite memory data-types, we continuously write the output byte stream even when our final range is not calculated fully yet.

We output a bit in the output stream under 2 conditions :

1. Lower limit’s most signifiacnt bit is 1 : In this case, the range is lying in between L, and H for all following cases, so MSB will remain 1. Thus, we remove a bit OUTPUT 1 and right shift, to increase the presicison by 1 bit.

2. Upper Limit’s Most significant bit is 0 : In this case, we are sure that since following ranges R lies in [L,H), the MSB will always be 0. We make use of this fact, output 0 and increase our presicion by 1 bit.

Consider, **unsigned int** data type in c++ for storing the high and low values. The code looks like :

```
if( L & 0x80000000 != 0 )
        output(1);
else if ( H & 0x80000000 == 0)
        output(0);
```

III. Decompression

Decompression is also handled in similar fashion. We keep on reading bits, unless we find a confirmed range in which a unique character lies.
Suppose we have read 1100110100... , notice that we have read a decimal point and it is actually 0.1100110100... and hence we search in 0 to 1 range.
The most optimal search will be binary search but that also falls insufficient :

<u>Optimizing Range search to O(1)</u> :

Notice binary search will give O(n * log(256)) which is o(n*8) which is a very large range. I thus plan to implement an O(1) search to make the complexity O(n). It will take the value of the smallest interval, and then mark the character that belongs to the given range.

Ex. Given frequency distribution is :
a = 0 to 0.4
b = 0.4 to 0.6
q = 0.6 to 0.7
n = 0.7 to 0.8
z = 0.8 – 1.0

Initialize

ar[0] = a, ar[0.1] = a , ar [0.2] = a, ar[0.3] = a, ar[0.4] = b, ar[0.5] = b, ar[0.6] = q, ar[0.7] = n, ar[0.8] = z, and ar[0.9] = z;

Search

now, suppose the evaluated value is 0.56... then we look up 0.5 and output that character,
look up time for ar[0.5] is O(1).

The representation of floating number by array indices is done using hash functions.

IV.Optimal use of Hardware and Kernel

As a general rule of hand, we plan to use hardware well by :

1. Using bitwise operators instead of arithmetic.
2. Perform huge buffer read/write instead of character by character.
3. Perform block-wise file read-write. The block size is less enough to reduce load on RAM and prevent swapping of process due to high RAM utilization, and large enough to reduce the average number of Reads and Writes from Disc.
4. A possible implementation for the future could be to use seperate threads, for reading and writing, so that we can increase the throughput of the process.

Other modes of optimization are being explored

V. Refrences

Motivations and guidance for the material was developed via the following refrences :

[1] Youtube - Compressor Head [Arithmetic Compression]
[2] Hitchhiker's Guide to compression - Arithmetic Compression
[3] Wikipedia - Arithmetic Coding

============================