Arithmetic & LZ Coding

by Erol Seke

For the course “**Data Compression**”

ESKİŞEHİR OSMANGAZİ UNIVERSITY

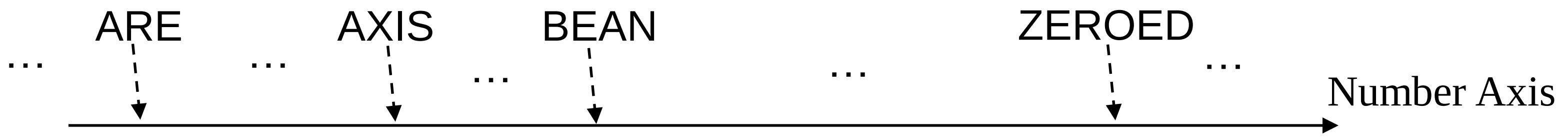
Arithmetic Coding

Any stream constructed using symbols from a fixed symbol set can be represented by a real number within a certain range. The number itself can be coded in binary.

Alphabet :

A={A, B, C, D, … , Z}

Think of all the words that can be derived using this alphabet.
They all can be represented by a number within (0, 1).



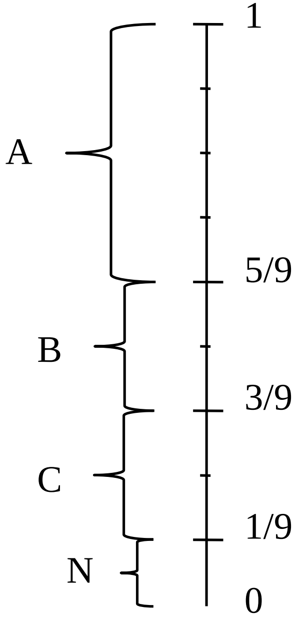
Example

Example stream : BABACANCA

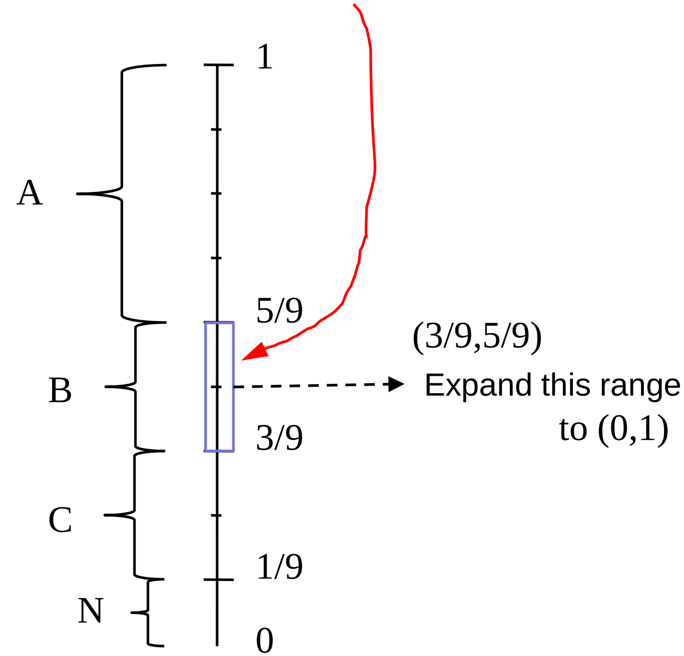
Alphabet :
A={N, C, B, A}

Probabilities of symbols :
z ={1/9, 2/9, 2/9, 4/9}

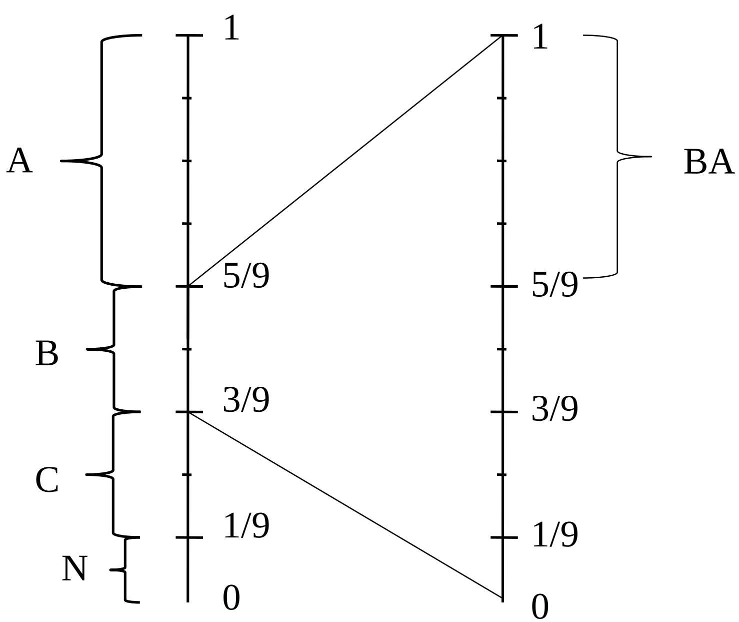
Assign probability sub-ranges to each symbol on (0,1) range



First symbol : B

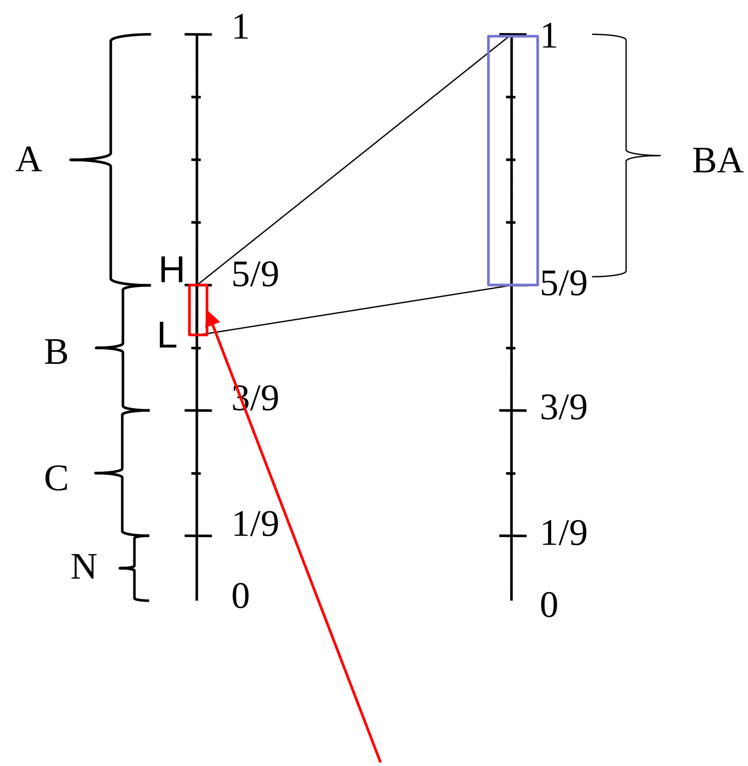


Second symbol : A



Expanded range

mapping of the range for BA to the first range



That is, BA is a number within the range shown as red

H= $\frac{5}{9}$

L= $\frac{3}{9}+\left(\frac{5}{9}-\frac{3}{9}\right)\frac{5}{9}$

Continuing this operation for the rest of the symbols in the stream, we find

$$L=0.504218425577384 \quad \text{and} \quad H=0.504228998069330$$

Any number within this (L, H) range is the number representation of BABACANCA

For example 0.50421.

In order to recover the stream BABACANCA from the number 0.50421

one needs

1- the number itself
2- Alphabet A
3- Probability set z
4- the number of symbols to generate

Compression

```
L = 0.0
H = 1.0
For Each Character(k) In Stream
  R = H - L
  H = L + R * H(z(k))
  L = L + R * L(z(k))
Next
Output = Any number between L and H
```

Decompression

```
X = the coded number
Until all characters are generated
  Output symbol C from X
  R = H(X) - L(X)
  X = (X - L(z(C))) / R
Next Character
```

Problem : Long streams require higher resolution numbers. No computer can retain the desired resolution/accuracy in numbers and perform arithmetic on them.

Solution : J. G. Witten, in 1987, has proven that floating point numbers are not actually required and that binary numbers are sufficient to perform the arithmetic coding algorithm(s).

Lempel-Ziv Coding

Advantage : Does not require preparation of a dictionary beforehand. Dictionary is prepared along the compression process.

Therefore : Compression is fast but not optimal for smaller input files. It theoretically approaches entropy-limit with larger input files.

Method : Each new symbol block refers to a previously seen block and creates a symbol block by appending a new symbol to it.

Example :

previous entries : 1. D, 2. DA, 3. DAT, 4. C, 5. CO, 6. COM, …
new entry : **DAT**A

in this example the new word DATA is expressed by referencing a previous entry and appending the A character afterwards.
3A : means that the 3rd entry and A

Example :

Input stream : `0001100001100111000010010000010`

symbol blocks: `0 00 1 10 000 11 001 110 0001 0010 00010`

no block is seen previously

These separated blocks forms a dictionary (reference table)

No	A
0	0
1	00
2	1
3	10
4	000
5	11
6	001
7	110
8	0001
9	0010
10	00010

Since these are already in the file, location of a previously seen block in the file is a reference actually.

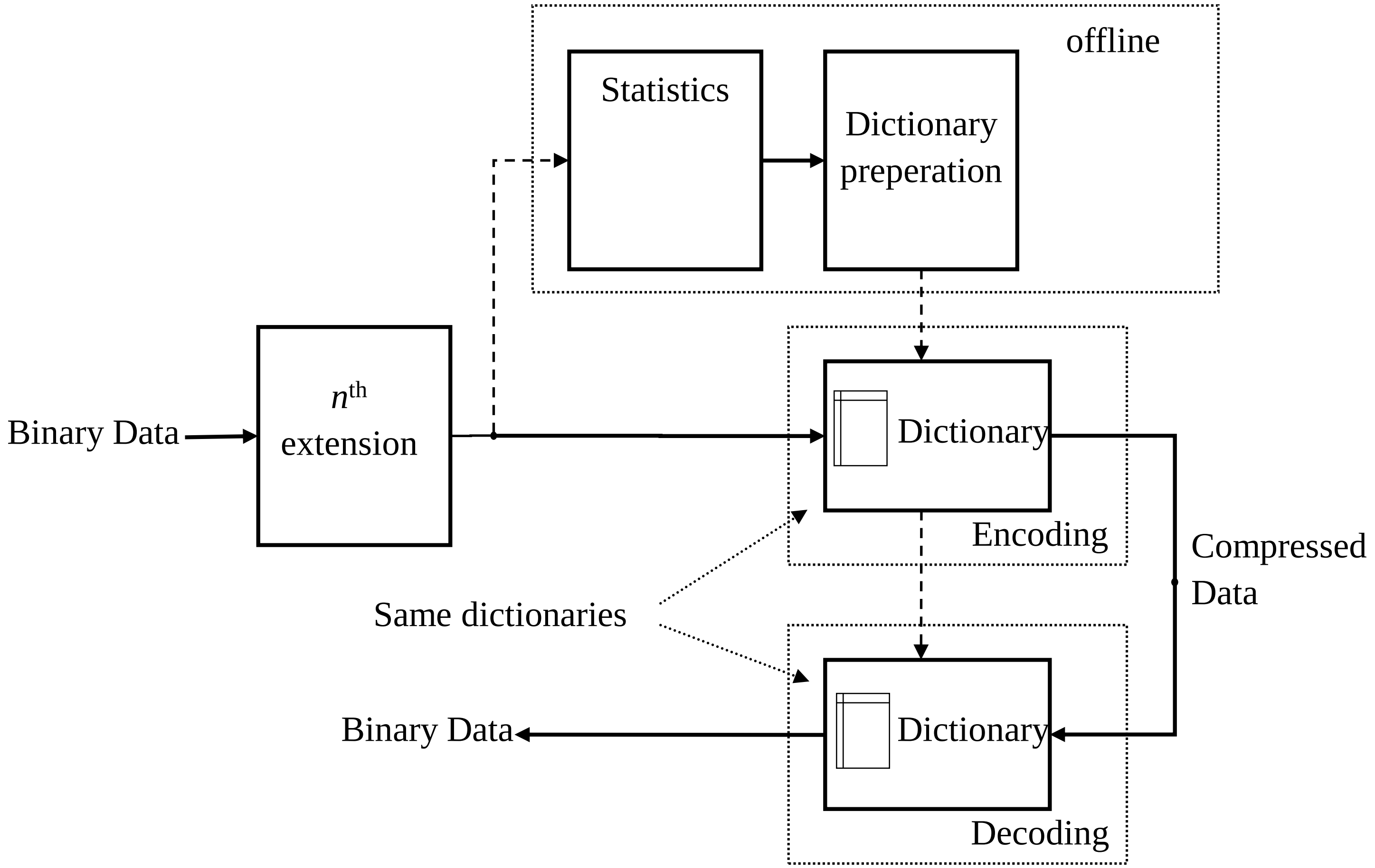
Output (compressed file) is created during back-searching

(,0) (1,0) (,1) (2,0) (1,0) (2,1) (1,1) (5,0) (4,1) (6,0) (8,0)

It might seem a very inefficient coding initially but note that larger and larger blocks are getting represented by a small number.

Flow of Dictionary Coding

(Compression/Decompression)



END