Codificação aritmética

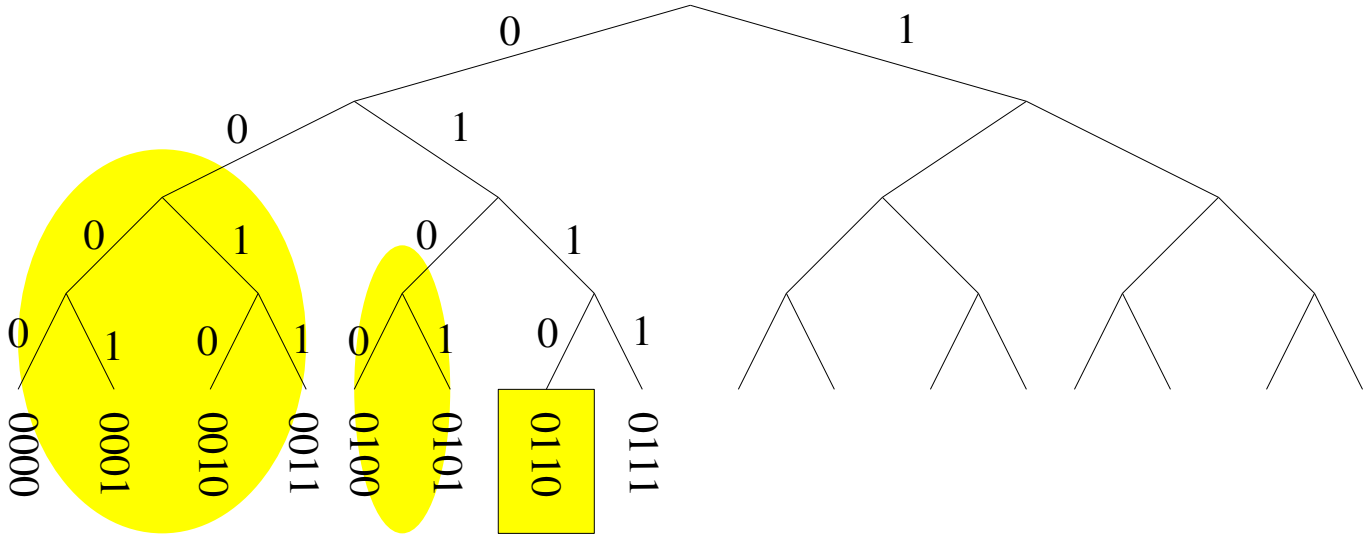
É uma extensão directa da codificação de Shannon-Fano-Elias onde, em vez de atribuir palavras de código a símbolos individuais, se atribui uma única palavra à mensagem inteira.

Ideia:
Substituir cada sequência de *n* símbolos por um único número no intervalo [0,1[. O número tem tanto mais casas decimais quanto menos provável é o bloco a codificar (=Shannon-Fano-Elias).
Calcular de forma eficiente a probabilidade conjunta do bloco $p(x^n)$ e a acumulada $F(x^n)$ que permite determinar o código.

Vantagens:
- Não é necessário construir/manipular árvores.
- Desempenho muito bom (melhor que o Huffman) uma vez que não necessita de um número inteiro de bits para codificar cada símbolo individualmente.

Considere-se duas strings binárias x^n e y^n de comprimento n.
Define-se uma ordem tal que $x^n > y^n$ se $0.x^n > 0.y^n$.

Podemos colocar as strings como folhas de uma árvore com n níveis.
Se $x^n > y^n$ então x^n está à direita de y^n.
Assim, $F(x^n)$ é a soma das probabilidades de todos os blocos à esquerda de x^n.



$F(x^n) = ?$

$F(x^n)$ pode ser calculado eficientemente somando as probabilidades das árvores à esquerda de **0110**.

Codifica-se o bloco x^n com a parte decimal de um número no intervalo [$F(x^n)$-$p(x^n)$, $F(x^n)$ [truncado a $l(x^n)$ casas decimais com

$$l(x^n)=\lceil -\log p(x^n)\rceil+1$$

Exemplo: Fonte binária i.i.d. com x~Bern(0.15).
A fonte gera a string **111011111101011111111**. Para codificar esta mensagem é necessário calcular $F(x^n)$ e $p(x^n)$.

$p(x^n) = p(\mathbf{111011111101011111111}) = 0.15^3 \times 0.85^{18} \cong 0.0001810566$
$l(x^n)$ = 14 bits (o original tem 21 bits!)
$F(x^n) = p(\mathbf{0}) + p(\mathbf{10}) + p(\mathbf{110}) + p(\mathbf{11100}) + p(\mathbf{111010}) + p(\mathbf{1110110}) +$
$+ p(\mathbf{11101110}) + p(\mathbf{111011110}) + p(\mathbf{1110111110}) +$
$+ p(\mathbf{111011111100}) + p(\mathbf{11101111110100}) + ...$

$C(x^n) = \mathbf{01110001110100}$, $L(C) \cong 0.6667$ bits, $H(X) \cong 0.6098$ bits

À medida que se vai actualizando o valor $F(x^n)$ são necessárias progressivemente mais casas decimais.

A utilização da unidade de vírgula flutuante dos CPUs actuais está fora de causa, uma vez que tem uma precisão muito limitada (52 bits).

A dificuldade deste algoritmo está em escrever rotinas em aritmética inteira que efectuem os cálculos com a precisão necessária.
Actualmente estas rotinas já existem.