Modelos de ordem superior

- Até aqui, trabalhámos essencialmente com fontes sem memória S dadas, embora admitíssemos proceder à codificação das suas extensões S^n.
- Num *modelo de ordem k*, são dadas as probabilidades conjuntas $P(s_{i1}s_{i2}\cdots s_{i,k+1})$ de todas as mensagens possíveis de comprimento $k+1$. Note-se que então
  - como $P(s_{i_1}s_{i_2}\cdots s_{i_m}) = \sum_{i_{m+1}}\sum_{i_{m+2}}\cdots\sum_{i_{k+1}} P(s_{i_1}s_{i_2}\cdots s_{i_m}s_{i_{m+1}}\cdots s_{i_{k+1}})$, os modelos de uma ordem são dedutíveis dos modelos de ordem superior;
  - como $P(s_{i,m+1}|s_{i1}s_{i2}\cdots s_{im}) = \frac{P(s_{i1}s_{i2}\cdots s_{im}s_{i,m+1})}{P(s_{i1}s_{i2}\cdots s_{im})}$, podemos construir modelos de Markov de ordens $m < k+1$, conhecidas que fixam as probabilidades para cada estado e cada transição.
- Vejamos como as codificações de Huffman e aritmética são afetadas pela passagem a modelos de ordem superior.

Codificação de Huffman de ordem superior

- A questão básica que desde logo se coloca é como é que as probabilidades para S^n são obtidas, conhecido um modelo de ordem k.
  - para $n \leqslant k+1$, podemos usar a fórmula acima.
  - Para $n > k+1$, podemos trabalhar como se a memória se perdesse.
- Mas, na realidade, como é obtido um modelo para começar? Naturalmente pela estimativa das probabilidades face às observações das mensagens produzidas pela fonte.
  Assim, as probabilidades são estimadas pela fórmula

$$P(s_{i1}s_{i2}\cdots s_{i,k+1}) = \frac{C(s_{i1}s_{i2}\cdots s_{i,k+1})}{\sum_{j_1,j_2,\ldots,j_{k+1}} C(s_{j_1}s_{j_2}\cdots s_{j_{k+1}})}$$

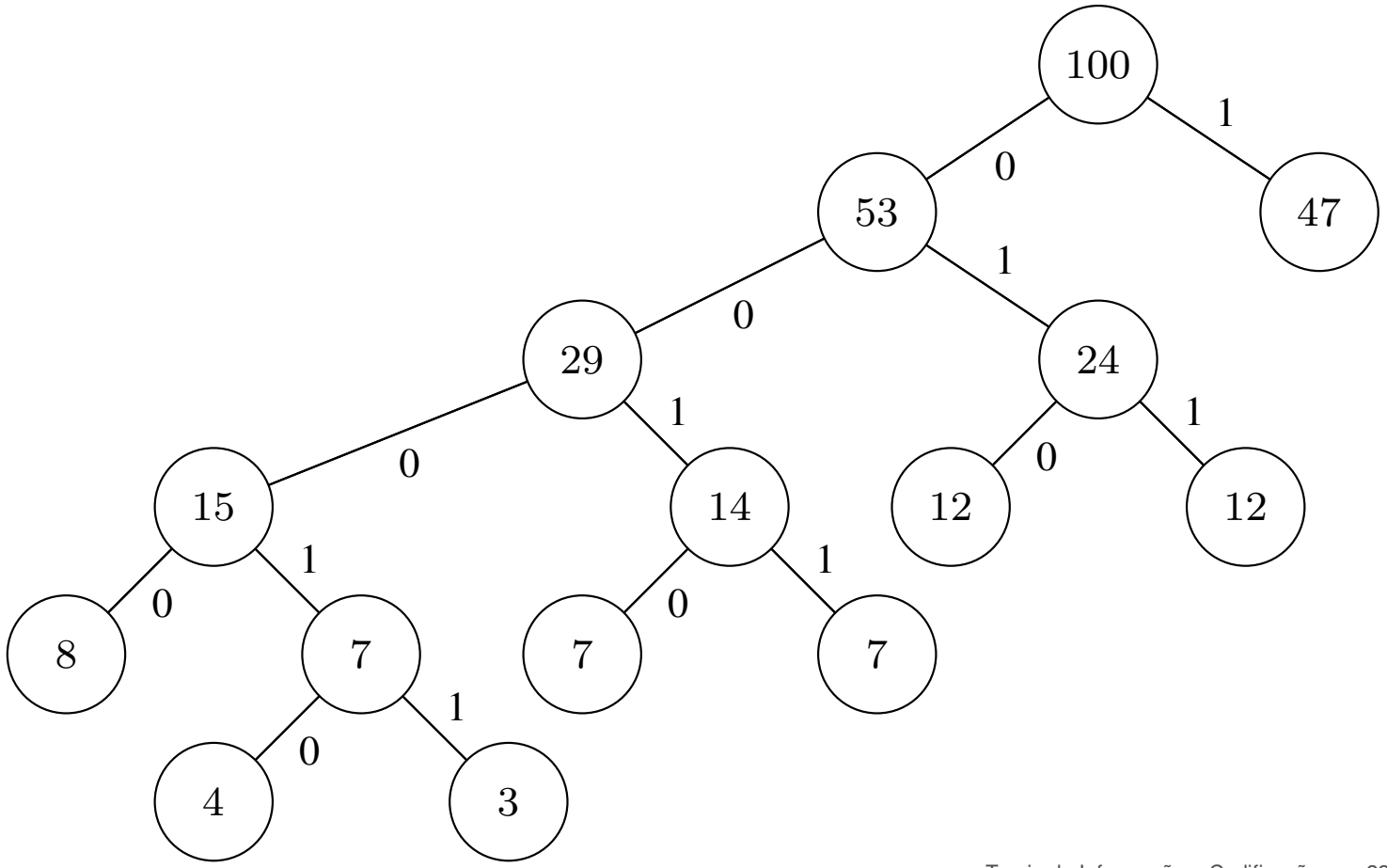
onde $C(s_{i1}s_{i2}\cdots s_{i,k+1})$ representa o número de vezes que a mensagem $s_{i1}s_{i2}\cdots s_{i,k+1}$ foi observada.

Exemplo

A menos de escala, podemos então trabalhar com os inteiros $C(s_{i1}s_{i2}\cdots s_{i,k+1})$ em vez das probabilidades.

Por exemplo se, para uma fonte binária, na mensagem de comprimento 100 se observa as contagens indicadas, então o código de Huffman binário é dado pela seguinte árvore:

palavra	C
000	47
001	12
010	7
011	8
100	12
101	3
110	7
111	4



Codificação aritmética de ordem superior

- Para a codificação arimética de uma mensagem de comprimento n precisamos das probabilidades condicionadas $P(s_{ij}|s_{i1}\cdots s_{i,j-1})$ para $j = 1, \ldots, n$.
  Conhecido o modelo k-ário,
  - para $j \leqslant k+1$, $P(s_{ij}|s_{i1}\cdots s_{i,j-1}) = \frac{P(s_{i1}\cdots s_{i,j-1}s_{ij})}{P(s_{i1}\cdots s_{ij})}$;
  - para $j > k+1$, toma-se
    $P(s_{ij}|s_{i1}\cdots s_{i,j-1}) = P(s_{ij}|s_{i,j-k}\cdots s_{i,j-1})$.
- Para a codificação aritmética, poder-se-á também trabalhar com estimativas das probabilidades pela frequências observadas nas mensagens produzidas pela fonte.
  Ao contrário do que vimos para os códigos de Huffman, além dos valores de $C(s_{i1}\cdots s_{i,k+1})$, precisamos também da soma $\sum_{i_1,i_2,\ldots,i_{k+1}} C(s_{i_1}s_{i_2}\cdots s_{i_{k+1}})$.

Exemplo

Consideremos a codificação aritmética binária da mensagem 01011 baseada no modelo de segunda ordem resultante da contagem do exemplo anterior (de terceira ordem):

palavra	000	001	010	011	100	101	110	111
C	47	12	7	8	12	3	7	4

As probabilidades resultantes até à terceira ordem são:

palavra	000	001	010	011	100	101	110	111
prob.	0.47	0.12	0.07	0.08	0.12	0.03	0.07	0.04

palavra	00	01	10	11
prob.	0.59	0.15	0.15	0.11

palavra	0	1
prob.	0.74	0.26

Aplicando o algoritmo de codificação aritmética à mensagem 01011, obtemos:

$s_{ij}\|s_{i,j-2}s_{i,j-1}$	$P(s_{ij}\|s_{i1}\cdots s_{i,j-1})$	$[m, M[$
$-$	$-$	$[0, 1[$
$0\|-$	$P(0) = .74$	$[0, .74[$
$1\|0$	$P(1\|0) = \frac{P(01)}{P(0)} = .203$	$[.59, .74[$
$0\|01$	$P(0\|01) = \frac{P(010)}{P(01)} = .467$	$[.59, .66[$
$1\|10$	$P(1\|10) = \frac{P(101)}{P(10)} = .2$	$[.646, .660[$
$1\|01$	$P(1\|01) = \frac{P(011)}{P(01)} = .533$	$[.6525, .6600[$

Obtemos portanto o subintervalo $[.6525, .6600[$ associado à mensagem inicial 01011.

Na base 2, o intervalo é $[0.101001110000, 0.101010001111[$ pelo que a codificação da mensagem inicial é 10101.

4 Compressão de dados

- Recorde-se que a entropia é uma medida da dificuldade de prever ou da falta de redundância, i.e., do conteúdo de informação. Aumentar a entropia corresponde portanto a aumentar a informação e reduzir a redundância.
- Como vimos, a entropia pode ser alterada por codificação. Quando esta via é seguida para aumentar a entropia, falamos em *compressão de dados*.
- Naturalmente, para que a compressão dos dados seja eficaz, deve-se explorar a estrutura dos próprios dados, processo este que exige cálculo adicional.

Exemplo

- Dada uma mensagem nas letras A, B, C, D, uma medida da informação, em bits, contida na mensagem é o comprimento dessa mensagem codificada em código binário.
- No caso das letras ocorrerem aleatoriamente, o melhor que podemos alcançar é a codificação por blocos de dois bits para cada letra da mensagem, ou seja a medida da informação é 2 bits por letra.
- Suponhamos pelo contrário que as letras não ocorrem aleatoriamente mas que se verifica uma restrição por exemplo como a seguinte: cada A é seguido de B ou C, cada B é seguido de C ou D, cada C é seguido de D ou A, e cada D é seguido de A ou B. Numa tal mensagem, conhecida uma letra, a seguinte tem uma de duas possibilidades, pelo que basta uma alternativa binária para a descrever. Podemos portanto descrever a primeira letra por dois bits acrescentando em seguida um bit por letra, ou seja, para uma mensagem de comprimento n bastam $n + 1$ bits.

Conceitos básicos

- Estudamos anteriormente métodos de codificação que visavam reduzir o comprimento médio da mensagem codificada tendo em atenção simplesmente a frequência de ocorrência das letras na mensagem original.
- Na mensagem $00000000000000000001111111111111111111$, ambas as letras aparecem com a mesma frequência, mas a mensagem contém muito pouca informação; na linguagem da Matemática, poderíamos simplesmente comprimi-la escrevendo $0^{20}1^{20}$. Este tipo de codificação diz-se *codificação por comprimento de segmentos numa letra* (em inglês: *run-length coding*).
- Em certas situações, há segmentos de letras (*sub-palavras*) ou, mais geralmente, padrões, que se repetem frequentemente, pelo que a sua representação por palavras mais curtas pode comprimir a informação. Fala-se aqui em *codificação por dicionário* (*dictionary coding*).

- Em alguns casos, são conhecidos modelos estatísticos aproximados que podem ser usados para prever a informação e portanto comprimi-la na descrição do modelo—*técnicas de compressão estática*. Se o modelo não for exacto, as previsões por vezes falharão. Nas *técnicas semi-adaptativas de compressão* constrói-se um modelo estatístico que se acrescenta à informação comprimida. Os melhores algoritmos de compressão usam *técnicas adaptativas* nas quais o modelo é construído tanto pelo compressor como pelo descompressor à medida que decorre o processo de codificação ou descodificação.
- Em certas situações, como por exemplo na compressão de ficheiros de computador, em geral estamos interessados em recuperar fidedignamente a informação da sua versão comprimida. Esperamos portanto que a compressão seja feita *sem perda de informação*.
- Noutras situações, como na compressão de som ou imagem (ou suas sequências), o limiar de percepção pelos seres humanos é relativamente baixo, pelo que a redução do conteúdo de informação original pode não ser percetível. É então aceitável que a compressão se faça *com perda de informação*. O tratamento digital do som e da imagem, como por exemplo a fotografia digital, explora esta faceta.

A norma CCITT para transmissão de facsimile (fax)

- A CCITT (*Comité Consultatif International Téléphonique et Télégraphique*) desenvolveu um método de compressão de informação adequado à transmissão de informação, lida linha a linha, por *fax*. O método é tão eficaz que este meio de comunicação se tornou tão sistemático como o uso do correio ou do telefone e em muitos casos suplantando-os.
- A norma *CCITT grupo 3* destina-se à transmissão de facsimile de páginas A4 (210mm $\times$ 297mm) a preto e branco por linhas analógicas. A imagem em causa é decomposta em linhas horizontais que são varridas por um dispositivo de leitura (*scanner*), cada linha tendo 215mm e contendo 1728 *pixels*. Cada pixel é digitalizado pela representação por 0 ou 1 para indicar se é branco ou preto. Linhas de varrimento consecutivas distam 0.26mm ou 0.13mm conforme se utilize a resolução normal ou melhorada, o que dá cerca de 2×10^6 ou 4×10^6 bits por página.

- A transmissão da informação nesta forma seria demasiado cara para que este método de comunicação fosse utilizado tão sistematicamente, pelo que é essencial proceder à compressão da informação antes de a transmitir. Por exemplo, numa carta há muitas linhas de varrimento que são completamente brancas, pelo que bastaria dizer que a linha tem 1728 0's.
- Assim, usa-se um sistema de codificação que comprime a informação. Cada linha é tratada não como uma sequência de 0's e 1's mas como sequências alternadas de segmentos de 0's e 1's, que podem ser vistas como duas palavras cujas letras são inteiros positivos ($\leqslant 1728$). Usa-se uma versão modificada dos códigos de Huffman para proceder à codificação desta informação. De facto os blocos são partidos módulo 64: comprimentos de 0 a 63 e múltiplos de 64 (até $1728 = 27 \times 64$) são representados por palavras-código. Assim, blocos de 64 pixels consecutivos iguais, o que corresponde a $\frac{64}{1728} 215\text{mm} \simeq 8\text{mm}$ são codificados por uma palavra-código.

- Na passagem de uma linha para a seguinte descreve-se somente as diferenças em relação à linha anterior, o que permite reduzir o comprimento de uma palavra descrevendo a informação contida na nova linha. No caso de se verificarem erros de transmissão os erros locais poderiam desta forma propagar-se desastrosamente, pelo que este método só é usado de duas em duas linhas.
- Também existe uma norma *CCITT grupo 4* para a transmissão de fax por linhas telefónicas digitais, mais eficientes que as analógicas, o que permite codificar imagens com tons de cinzento ou a cores.
- Este método de compressão é usado no programa de compressão *bzip2* (além dos códigos de Huffman). Consiste em decompor a mensagem em blocos e reordenar as letras em cada bloco, por permutação cíclica, de forma a maximizar a compressão por blocos de letras iguais.

Compressão por ordenação de blocos

- Vejamos num exemplo como esta ideia funciona: consideremos a mensagem "`the fat cat sat`". As sucessivas permutações circulares produzem a primeira coluna de palavras. eordenando por ordem lexicográfica da direita para a esquerda, a menos do último caráter, obtemos:

the_fat_cat_sa	t	at_cat_satthe_	f
he_fat_cat_sat	t	atthe_fat_cat_	s
e_fat_cat_satt	h	at_satthe_fat_	c
_fat_cat_satth	e	_satthe_fat_ca	t
fat_cat_satthe	_	_cat_satthe_fa	t
at_cat_satthe_	f	the_fat_cat_sa	t
t_cat_satthe_f	a	t_satthe_fat_c	a
_cat_satthe_fa	t	fat_cat_satthe	_
cat_satthe_fat	_	t_cat_satthe_f	a
at_satthe_fat_	c	_fat_cat_satth	e
t_satthe_fat_c	a	tthe_fat_cat_s	a
_satthe_fat_ca	t	satthe_fat_cat	_
satthe_fat_cat	_	cat_satthe_fat	_
atthe_fat_cat_	s	he_fat_cat_sat	t
tthe_fat_cat_s	a	e_fat_cat_satt	h

o que produz a mensagem modificada "`fscttta_aea__th`", onde encontramos os blocos de letras repetidas "`ttt`" e "`__`".

- Como recuperar a mensagem inicial da mensagem modificada "`fscttta_aea__th`"?
- Reordenando os carateres, obtemos "`___aaacefhstttt`". Juntando estas duas sequências, obtemos:

_	f
_	s
_	c
a	t
a	t
a	t
c	a
e	_
f	a
h	e
s	a
t	_
t	_
t	t
t	h

Conhecendo qual das letras é a primeira (o que tem de ser registado), começamos por a ler na segunda coluna, no caso concreto o último `t`. A seguir, localizamos o último `t` na primeira coluna, a que corresponde a leitura do `h` ao fundo da segunda coluna. Vamos sucessivamente decifrando a letra seguinte notando qual é aquela que aparece na segunda coluna da linha onde aparece na primeira coluna a mesma ésima ocorrência da letra anterior que aquela que acabamos de encontrar na segunda coluna.

Codificação por dicionário

- Na compressão de texto podemos fazer uma lista das palavras que ocorrem, substituindo cada palavra por uma referência para essa lista.
- Ziv e Lempel (1977) propuseram um método que permite a compressor e descompressor trabalhar sem conhecer a lista completa.
- O algoritmo de codificação *LZ77* produz uma sequência de ternos $(m, n, \texttt{c})$, onde m e n são inteiros (positivos) e `c` é um caráter. Tal terno indica que os próximos $n+1$ carateres são os n carateres que se encontram m posições à esquerda seguidos do caráter `c`.
- Por exemplo, consideremos o texto "`the_fat_cat_sat_on_the_mat.`". Até ao segundo "`t`" não há repetições pelo que a codificação correspondente é $(0,0,\texttt{t})(0,0,\texttt{h})(0,0,\texttt{e})(0,0,\texttt{\_})(0,0,\texttt{f})(0,0,\texttt{a})$. Reconhecida a repetição `t`, a seguir encontramos o terno $(6,1,\texttt{\_})$. Segue-se a nova letra `c`, que produz o terno $(0,0,\texttt{c})$ e as repetições `at_`, seguidas das letras `s` e `o`, que produzem os ternos $(4,3,\texttt{s})(4,3,\texttt{o})$. Prosseguimos com $(0,0,\texttt{n})(0,0,\texttt{\_})$ e, finalmente, $(19,4,\texttt{m})(11,2,.)$.
- Até onde levar a procura de segmentos do texto idênticos aos já encontrados? Normalmente é fixado um majorante para essa procura. O programa *gzip* cria uma tabela *hash* para acelerar a procura de ocorrências anteriores de segmentos.

LZ78

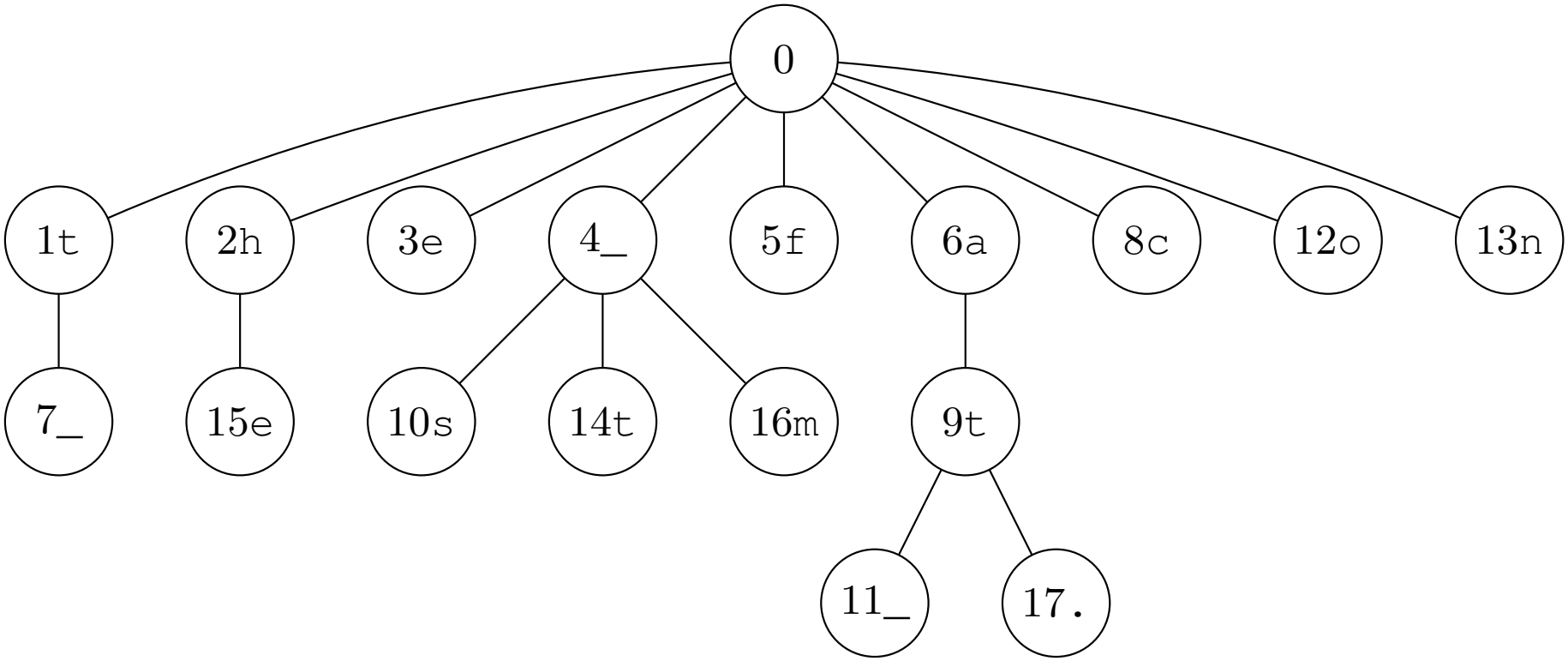
- No algoritmo *LZ78*, é construído sucessivamente um dicionário, tanto no processo de compressão como de descompressão e o que efetivamente ocorre no texto comprimido são referências a este dicionário. O dicionário em vez de ser tomado como uma lista de palavras é tratado como uma árvore, o que facilita a sua localização. Mais precisamente o texto comprimido consiste da árvore em causa e de pares $(i, \texttt{c})$ onde i é o i-ésimo termo no dicionário (num percurso natural da árvore) que deve ser seguido do caráter `c`.
- No exemplo anterior, "`the_fat_cat_sat_on_the_mat.`", começamos com $(0,t)(0,h)(0,e)(0,\_)(0,f)(0,a)$ que vai sucessivamente conduzindo às seguintes entradas do dicionário:

$1:$ `t`
$2:$ `h`
$3:$ `e`
$4:$ `_`
$5:$ `f`
$6:$ `a`

A seguir, encontramos um caráter que já consta do dicionário e vamos lendo o texto ao longo da árvore até que surja uma discrepância, ou seja `t_`, o que produz a codificação $(1, \_)$ e um novo item, $7:$ `t_`, para o dicionário. Segue-se $(0, \texttt{c})$ e o novo item $8:$ `c`. A seguir vem o segmento `at`, onde o último caráter é o que o faz sair da árvore, produzindo $(6, \texttt{t})$ e $9:$ `at`. Note-se que, representando efetivamente a árvore, os números correspondem à numeração dos vértices à medida que eles vão aparecendo.

the_fat_cat_sat_on_the_mat.

$(0, \texttt{t})(0, \texttt{h})(0, \texttt{e})(0, \_)(0, \texttt{f})(0, \texttt{a})(1, \_)(0, \texttt{c})(6, \texttt{t})(4, \texttt{s})(9, \_)(0, \texttt{o})(0, \texttt{n})(4, \texttt{t})(2, \texttt{e})(4, \texttt{m})(9, .)$



Outros algoritmos

- Note-se que a árvore pode ser construída igualmente a partir do texto original e a partir do texto comprimido. A árvore por sua vez determina completamente ambos. Ou seja, qualquer destes dados permite recuperar os outros dois, e além disso de forma eficiente.
- Na versão *LZW* (Lempel-Ziv-Welch), começa-se com os códigos ASCII dos carateres, correspondendo ao início da árvore, a qual vai crescendo de forma análoga a partir daí. O velho comando *compress* do sistema *Unix* usa este algoritmo.
- O algoritmo **PPM** (*Prediction by partial matching*) constrói sucessivamente para o texto a comprimir modelos de Markov que lhe permitam prever qual o caráter seguinte e os próprios modelos vão sendo actualizados no processo.