Algoritmo de compactação LZ77

O algoritmo de compactação LZ77 é uma técnica baseada em dicionário concebido em 1977 por *Ziv* e *Lempel*, e tem vindo a tomar sucessivas variantes tendentes a melhorar tanto a taxa de compactação, como o tempo de execução dos métodos de compactação.

Versões comerciais de compactadores (*ARC*, *PKZIP* e *ARJ*) utilizam esta técnica, também conhecida por "Janela deslizante".

Baseia-se, em princípio, na representação de uma frase de texto pela sua dimensão e posição em que se encontre situada num dicionário de frases que seja adoptado como referência, tanto para a compactação como para a descompactação. Ocorre compactação caso o número de bytes necessários para representar a dimensão e posicionamento da frase seja menor que o necessário para exprimir a frase.

Para que tal aconteça, impõe-se definir limites, quer para a dimensão máxima das frases a codificar, quer para a dimensão do dicionário.

Se, por exemplo, adoptarmos para dimensão máxima das frases 16 caracteres e para dimensão do dicionário 4096 caracteres, e pretendermos codificar a frase "ENCONTRE SITUADA", que exigiria 128 bits de representação em *ASCII*, se encontrarmos na posição 175 do dicionário, não exactamente essa frase, mas a sequência dos seus primeiros 10 caracteres "ENCONTRE S", esta frase seria codificada por "175 10", exigindo 12 bits para codificar a posição (no domínio 0 a 4095) e 4 bits para codificar o número das coincidências com o dicionário (no domínio 0 a 15). A relação entre o número de bits da frase original a codificar (80 bits) e a sua codificação com este critério (12+4 = 16 bits) traduziria a excelente compactação de 16/80 = 20/100.

O algoritmo LZ77 adopta um tipo de dicionário em janela deslizante, evoluindo ao longo da acção de compactação, constituído pelos últimos *N* caracteres que já tenham sido compactados.

A estrutura básica de suporte consiste num contentor circular (`RingBuffer`), no qual (aquando da compactação) vamos distinguir duas zonas, conforme se ilustra na figura 1.

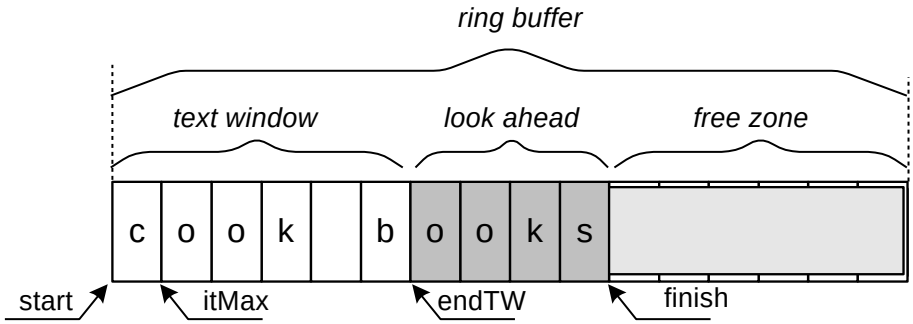


Figura 1 — Janela deslizante para compactação

Na figura, pressupõe-se uma dimensão máxima de 4 para o *look ahead buffer* (que contém a sequente frase a codificar), uma dimensão máxima de 12 para o *text window* (número máximo de caracteres que pode comportar). Considera-se que cada carácter é início de uma frase de dimensão 4, pelo que conceptualmente consideramos que este *text window* pode conter 12 frases de dimensão igual ao do *look ahead*. Nestas condições, a posição no dicionário pode ser representada a 4 bits e o número de coincidências pode ser representado a 2 bits.

No estado em que se mostra o *ring buffer* na figura, considera-se que já anteriormente foi codificado (compactado) o segmento de texto "**`cook b`**" presente no *text window* e que pretendemos codificar o segmento de texto que se lhe segue "**`ooks`**", presente no *look ahead*. Para tal, vamos comparar essa frase com as 6 frases já presentes no dicionário (*text window*), de forma a detectar qual dessas frases apresenta o maior número de coincidências com ela. E constata-se que, neste caso, só a frase apontada por `itMax` apresenta coincidências com a do *look ahead*, nomeadamente três coincidências ("**`ook`**"). Assim, o segmento de mensagem "**`ook`**" irá ser codificado (compactado por um *token* constituído pela representação da **posição** e **coincidências**). Para representar o `itMax`, com um número de bits que nunca exceda 4 (valores entre 0 e 10), vamos adoptar a distância entre `endTW` (apontador para o fim do *text window*) e `itMax` (que aponta para o início da frase com maior número de coincidências). Neste caso, o *token* representativo da frase "**`ook`**" será "5 3" (0101 11), compactando-se assim em 6 bits uma informação que exigiria 24 bits.

Na descompactação, utilizando um *ring buffer* da mesma dimensão mas onde já não existe a região *look ahead* (o *text window* ocupa todo o *ring buffer*), frente ao *token* gerado será reconstruído no *ring buffer* o texto original (descompactado), como se mostra na figura 2.

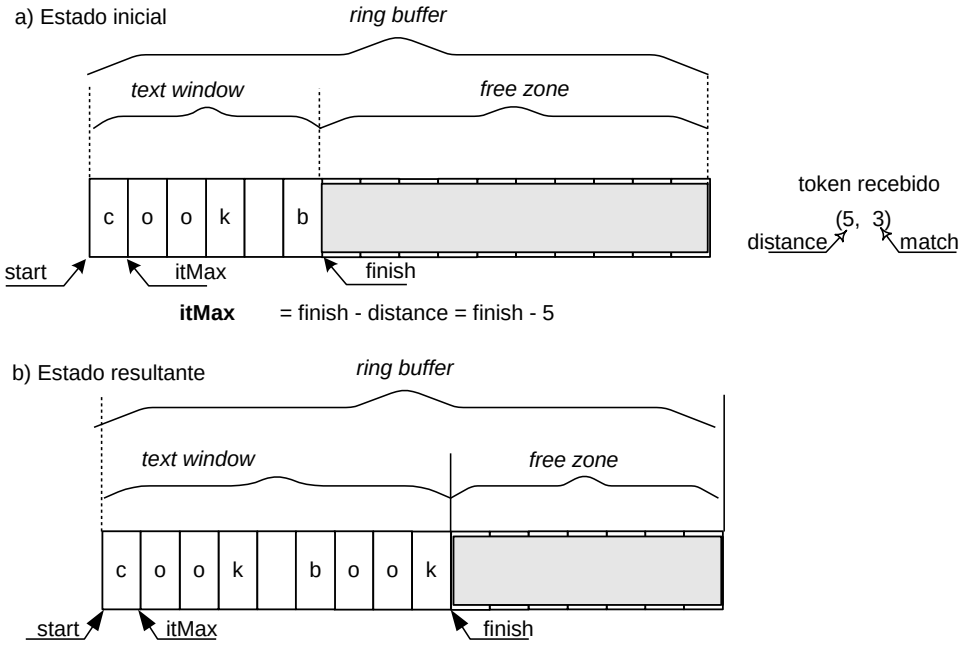


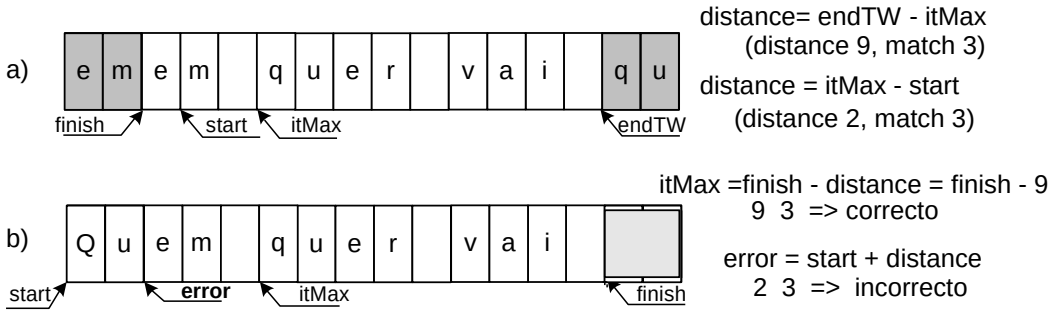
Figura 2 — Janela deslizante para descompactação

Estando o *ring buffer* no estado mostrado em a), ao ler o *token* "**5 3**", posiciona-se **`itMax = finish - 5`** como se mostra, e insere-se no *ring buffer*, em sequência, os três caracteres ("**`ook`**") apontados por `itMax`, enriquecendo assim o dicionário com mais três frases de comprimento 4 ("**`k bo`**", "**` boo`**" e "**`book`**"), como mostrado em b).

Pode parecer estranho, à primeira vista, adoptarmos como componente distância do *token* a distância entre `endTW` e `itMax` e não a distância entre `itMax` e `start`. Tal opção não garantia coerência de representação nalguns estados do `RingBuffer`, como se mostra na figura 3.

Com efeito, enquanto o *ring buffer* de compactação inclui o *look ahead buffer* com a próxima frase a compactar e, por esse facto, já esmagou os dois primeiros caracteres que

lhe tinham sido inseridos e o `itMax` tomou a distância 2 relativamente ao `start` quando gerou o *token*, o da descompactação, ao descodificar esse *token* ainda não está plenamente ocupado e o `itMax` correcto situa-se a uma distância 5 da posição que deveria corresponder ao componente **`distance`** desse *token*, o que é totalmente incoerente.



a) Estado do buffer de compactação aquando da geração do token

b) Estado do buffer de descompactação aquando da interpretação do token

Figura 3 — Razão da adopção de `position=endTW−itMax`

Exemplo ilustrativo da compactação

O algoritmo suporta-se num *ring buffer* organizado em dois campos, durante a compactação:

- Um dicionário *text window* que retém um número fixo de símbolos (os últimos *N* símbolos) do texto codificado até ao momento.
- Um *look ahead buffer*, também de comprimento fixo, que contém um número fixo de símbolos lidos do ficheiro a compactar e ainda não codificados.

O algoritmo do processo de compactação toma como chave de pesquisa a frase situada no *look ahead buffer* e procura no *text window* a sequência que tenha com ela o maior número de coincidências de símbolos.

Adoptam-se dois tipos de *tokens*: de **símbolo** e de **frase**.

Caso não conste do dicionário nenhuma frase que coincida sequer com o primeiro carácter da frase presente no *look ahead*, codifica-se esse carácter com um *token* de símbolo.

Os *tokens* de símbolo isolados tomam a dimensão de 9 bits: um bit *flag* a 0 (indicando que se trata de um *token* de símbolo), seguido dos 8 bits do próprio símbolo que codifica.

Os *tokens* de frase usam o critério que temos vindo a descrever e a sua dimensão em bits depende da dimensão adoptada para o *look ahead buffer* e para o *text window*.

Neste exemplo, para facilidade de exposição, adopta-se a dimensão 4 para o *look ahead* e 16 para o *ring buffer* total.

Assim, bastam 2 bits para codificar de 1 a 4 coincidências (atribuindo o código (00) a uma coincidência e (11) a quatro coincidências) e 4 bits para codificar a posição no dicionário onde ocorre esse máximo de coincidências.

A representação dessa posição no *token*, pelas razões atrás expostas, toma o valor da distância entre `endTW` e `itMax`.

Assim, os *tokens* de frase terão a dimensão de 7 bits: um bit *flag* a 1 (significando que

se trata de um *token* de frase), seguido da posição onde se situa a frase no dicionário (codificado a 4 bits), seguido do número de coincidências com a frase a codificar (codificado a 2 bits).

No caso geral, com dimensões bastante maiores das que agora se propõem, a dimensão dos *tokens* de frase serão diferentes e só serão codificados como *token* de frase casos em que o número de coincidências conseguidas, codificadas por *tokens* de símbolos separados, atingisse uma dimensão maior que a de um *token* de frase. No caso presente, justifica-se gerar um *token* de frase, mesmo que só ocorra coincidência de um único carácter, dado que a dimensão dos *tokens* de frase é menor que a dimensão dos *tokens* de carácter.

A codificação dos sucessivos caracteres recolhidos do ficheiro a compactar correspondente à mensagem tomada como exemplo:

"**`coookbooookokokokxy`**"

e os *tokens* resultantes que vão sendo inscritos no ficheiro compactado ilustram-se na Figura .

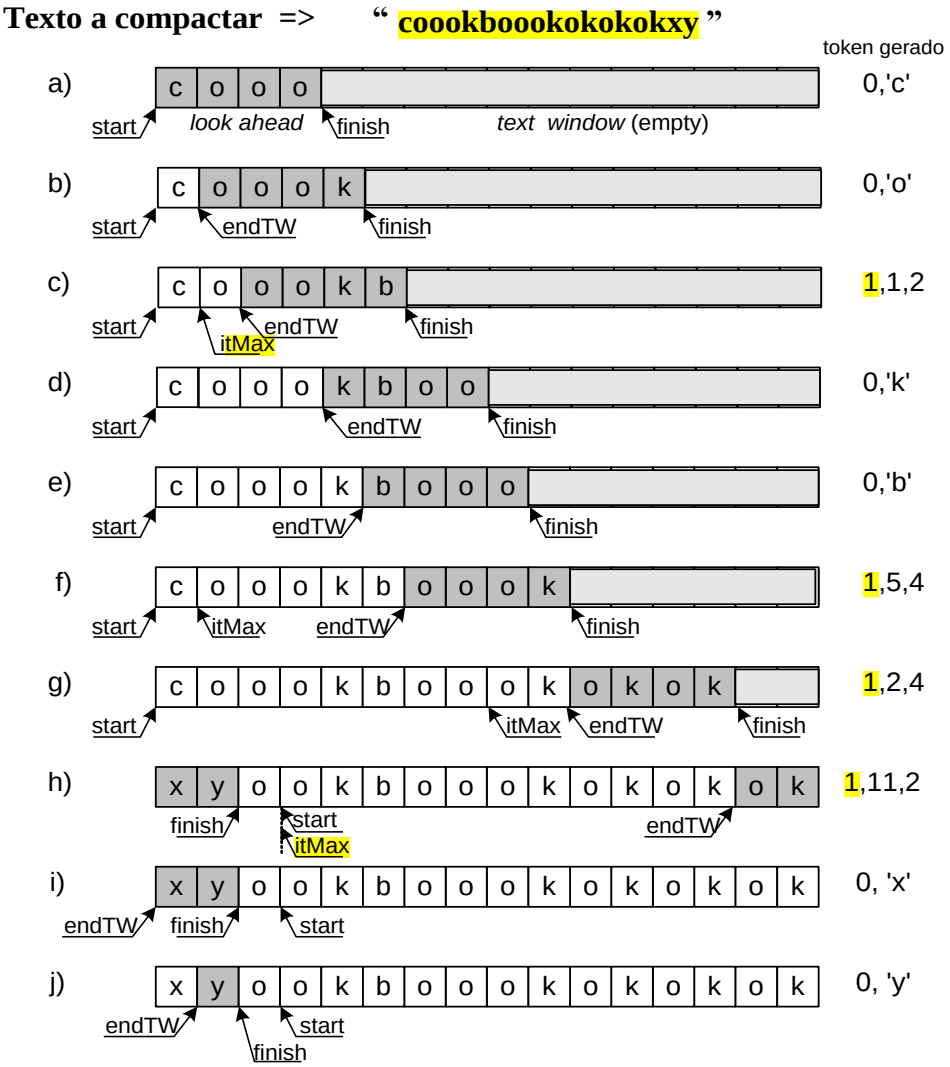


Figura 4 — *Tokens* resultantes da compactação de uma mensagem

Em a) procede-se à recolha no *look ahead buffer*, da primeira frase de 4 caracteres a codificar. Desta frase, só irá ser codificado o primeiro carácter com um *token* de carácter, dado que o *text window* está vazio. No ficheiro compactado, será portanto inscrito o *token* **'0 01100011'**, em que **01100011** é o código *ASCII* do carácter '**c**' em binário.

Em b) passa para o *text window* o carácter codificado e é completado o preenchimento do *look ahead* com o próximo carácter. A frase do *look ahead* a codificar ("**oook**") não tem nenhuma coincidência com a única frase presente até ao momento no *text window* ("**cooo**"). Assim, só poderá ser codificado o primeiro carácter do *look ahead* ('**o**') com um *token* do carácter '**o**' (**0 01101111**).

Em c) foi passada a nova frase "**oook**" para o *text window* (índice 1) e preenchido o *look ahead* com o carácter seguinte. Neste caso, já ocorrem duas coincidências entre a frase a codificar "**ookb**" e a frase da posição `itMax` do *text window* "**oook**" que se encontra a uma distancia de 1. Assim, será gerado um *token* de frase 1, 1, 2 (1 0001 01) — duas coincidências codificam-se com 01 a dois bits – dado que nunca se pretende codificar 0 coincidências.

Em d) preencheu-se o *look ahead* com os dois caracteres seguintes do ficheiro a compactar e transferiu-se para o *text window* as duas novas frases correspondentes aos dois caracteres codificados no passo anterior. No entanto, dado que nenhuma das frases situada no *text window* começa com o carácter '**k**', só podemos codificar o primeiro carácter do *look ahead* ('**k**') com um *token* de carácter (**0 01101011**).

Em e) preenche-se o *look ahead* com o carácter '**o**', transfere-se para o *text window* a nova frase "**kboo**" e (pelas mesma razões da alínea anterior) codifica-se o carácter '**b**' com um *token* de carácter (**0 01100010**).

Em f) completa-se com o carácter '**k**' o *look ahead* e codifica-se a frase de 4 caracteres nele situada com um *token* de frase 1,5,4 (**1 0101 11**), dada a total coincidência entre a frase do *look ahead* e a situada no *text window*, distância 5.

Em g) completa-se o *look ahead* com os caracteres "**okok**" e gera-se um *token* de frase com distância 2 e com 4 coincidências (**1 0010 11**).

Em h) completa-se o *look ahead* com "**okxy**" e gera-se um *token* de frase com a distância 11 e com 2 coincidências (**1 1011 01**).

No passo seguinte teríamos que ler (se fosse possível) mais dois caracteres do ficheiro a compactar. No entanto, o ficheiro já não tem mais caracteres. Detectado este facto, é posto em execução o algoritmo da "fase final do processo", que se limita a codificar os caracteres que restarem no *look ahead*. Neste caso, são gerados os *tokens* correspondentes aos caracteres '**x**' e '**y**'.

Notar que em h) o *text window* ficou totalmente preenchido, tendo já sido esmagados dois dos caracteres anteriormente inseridos; `start` avançou três posições e, até que se esgote o ficheiro a compactar, mantém-se constante daí em diante a relação (`finish+1 == start`).

A descodificação dos sucessivos *tokens* recolhidos do ficheiro compactado processa-se como se mostra na Figura .

Por forma a reconstituir o ficheiro original através do ficheiro compactado (ficheiro de *tokens*), terá de se instanciar um *text window* com a mesma dimensão do que foi usado para a compactação.

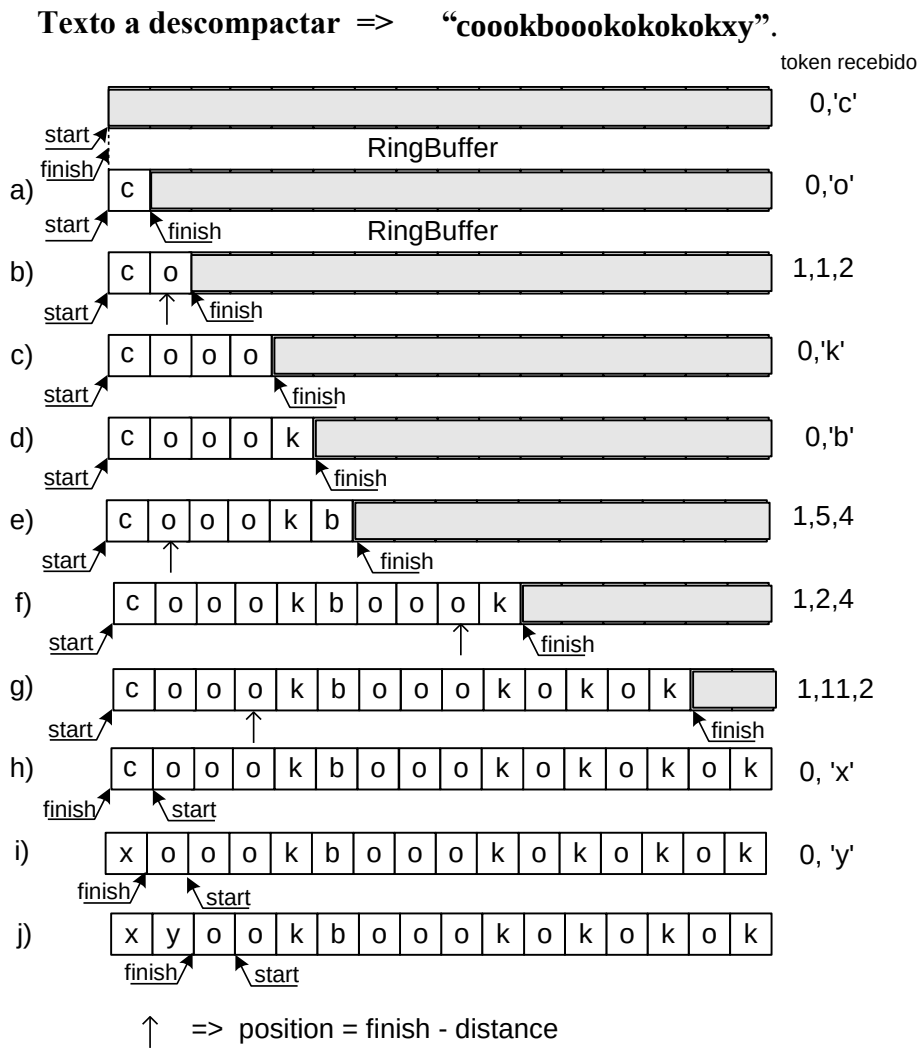


Figura 5 — Descompactação da mensagem a partir dos *tokens*

O processo de descompactação insere no *text window* os caracteres correspondentes aos *tokens* lidos do ficheiro compactado (reconstituição do dicionário) e, simultaneamente, escreve no ficheiro descompactado esses caracteres.

Pode parecer estranho que, por exemplo, na alínea f) seja imposto pelo *token* copiar do dicionário quatro caracteres a partir de `position`, quando na realidade só existem dois caracteres para copiar. Acontece, no entanto, que, ao copiar o primeiro carácter, inseriu-se simultaneamente esse carácter no dicionário e ficará por esse facto disponível para vir a ser copiado, o mesmo acontecendo com os caracteres seguintes.

NOTA: Texto extraído e adaptado do livro "Programação em C++ - Algoritmos e Estruturas de Dados"