Universidade do Minho - Dep.to Informática
Mestrado em Engenharia de Telecomunicações & Informática

Serviços de Rede & Aplicações Multimédia

Edição 2023-2024

Trabalho Prático N°1 – Compressão LZWdR

v1

A compressão LZWdR é uma variante da compressão por padrões (ou por dicionário) conhecida como LZW (ver material de apoio) que, por sua vez, é uma variante do algoritmo LZ78. Dependendo do ficheiro, este tipo de algoritmos permite obter taxas de compressão e rendimentos virtuais superiores a 100%, o que é impossível com algoritmos baseados em compressão estatística como o Shannon-Fano, Huffman ou codificação aritmética. Existem muitas variantes do algoritmo original LZW e que são usadas nos programas populares de compressão como o ZIP ou o RAR ou na construção dos formatos de ficheiros do tipo GIF ou PDF. A utilização deste tipo de algoritmo faz com que deixe de ser necessário usar o algoritmo RLE em ficheiros com muitas repetições consecutivas de símbolos pois estas sequências são casos especiais de padrões e são facilmente comprimidos pelos algoritmos baseados em LZW.

O algoritmo LZW tem como principal característica a substituição de sequências de símbolos originais (ou padrões) pelos códigos respetivos. Como não se sabe à partida quais vão ser os padrões que vão ser encontrados num ficheiro (ou bloco de bytes em memória), e em que local do ficheiro, o algoritmo inclui um mecanismo elegante em que se vai registando apenas os padrões que vão sendo encontrados na fonte de informação à medida que os dados vão sendo processados. Além disso, os novos padrões são sempre definidos à custa de acrescentar símbolos a padrões já existentes. O algoritmo LZW começa com um conjunto de 256 padrões, que são os 256 símbolos individuais possíveis num *byte*. O registo dos padrões é feito numa tabela (ou estrutura de dados equivalente) que é inicializada com 256 entradas, uma para cada símbolo individual de 8 bits. O algoritmo vai identificando novos padrões a partir destes padrões básicos com símbolos individuais e acrescentando-os na tabela até se atingir um tamanho máximo definido para a tabela (para algoritmos baseados no LZW o valor costuma estar entre $4096=2^{12}$ e 2^{24}). Em geral, quando este tamanho máximo é atingido o algoritmo reinicia a tabela outra vez com apenas os 256 padrões de símbolos individuais. O algoritmo apresenta níveis de compressão cada vez maiores à medida que a tabela vai crescendo e os padrões vão ficando maiores, mas a a taxa de processamento de símbolos/bytes por segundo mantem-se constante. As componentes mais exigentes/lentas do algoritmo são a procura e registo/inserção de padrões. Embora o algoritmo não seja óbvio, é simples e fácil de implementar em linguagem C, C++ ou Rust.

O principal objetivo deste trabalho é implementar uma aplicação codificadora de ficheiros que implemente o algoritmo LZWdR (ver definição deste algoritmo no tutorial fornecido no material de apoio da UC e no anexo a este enunciado) e que calcule alguns dados estatísticos sobre o processo de codificação.

A grande diferença entre o LZW e o LZWdR é que este último acrescenta, em cada iteração, um ou mais novos padrões formados à custa de dois padrões já conhecidos, enquanto o LZW acrescenta, em cada iteração, um único novo

padrão formado por um padrão já conhecido e o símbolo seguinte. Isto leva a que o dicionário do LZWdR cresça mais rapidamente que o dicionário do LZW, eventualmente aumentando o rendimento, mas necessitando de mais memória na execução. No entanto, as diferenças reais de rendimento são dependentes da distribuição dos símbolos pelo ficheiro a codificar.

Parâmetros da ferramenta de codificação LZWdR

Os seguintes parâmetros de funcionamento da ferramenta devem ser implementados:

- Tamanho dos blocos para processamento do ficheiro de entrada – este tamanho também define indiretamente o tamanho dos blocos da informação codificada e que tem depois de ser gravada no ficheiro de saída; por defeito, o valor deve ser de 64 KByte; não é obrigatório implementar a possibilidade de se utilizar outros valores;
- Tamanho máximo do dicionário, em número de entradas/padrões – este valor deve incluir os padrões do dicionário inicial (normalmente 256 padrões iniciais equivalentes a todos os símbolos individuais com um byte); por defeito, o valor deve ser 2^{16}; não é obrigatório implementar a possibilidade de se utilizar outros valores;
- Tipo de dicionário inicial – este parâmetro define qual é o grupo de padrões iniciais; por defeito, o dicionário inicial é o mesmo que é definido para o LZW, ou seja, os 256 padrões equivalentes às 256 combinações dum byte; não é obrigatório implementar a possibilidade de se utilizar outros tipos de dicionários iniciais;
- Tipo de limpeza do dicionário – este parâmetro define o tipo de limpeza que é feito quando é atingido o número máximo de padrões que o dicionário pode conter; por defeito, o processo de limpeza não faz nada, i.e., quando se atinge o número máximo de entradas é usado o dicionário completo sem se acrescentar mais padrões novos; não é obrigatório implementar a possibilidade de se utilizar outros tipos de limpeza do dicionário (*reset* total, inserção condicionada, etc.);
- Informações estatísticas – este parâmetro define quais as estatísticas sobre o processo de codificação que são calculadas; por defeito, deve ser calculado o número de bytes processados, o número de vezes que os padrões são encontrados, o número de códigos de padrões enviados para o *output*, o tamanho médio dos padrões inseridos no dicionário e o número de vezes que o tamanho máximo do dicionário foi atingido; não é obrigatório implementar a possibilidade de se calcularem estatísticas adicionais;
- Informação sobre os padrões – este parâmetro define quais as informações que devem ser impressas durante o processo de codificação; por defeito, devem ser impressos todos os padrões que vão sendo encontrados na sequência de entrada; não é obrigatório implementar a possibilidade de se imprimirem informações adicionais.

Formato do ficheiro de saída

A sintaxe dos ficheiros de saída **lzwdr** para este trabalho é muito simples: uma sequência de números que representam os códigos (índices) dos padrões, cada um guardado como uma sequência de dígitos decimais, separados por vírgulas.

Informações da aplicação

A aplicação deve imprimir as seguintes informações durante a sua execução:

- Autoria e data de criação do código (no início);
- Nome do ficheiro de entrada (no início);
- Nome do ficheiro de saída (no início);
- Tamanho máximo do dicionário e tipo de dicionário inicial (no início);
- Tipo de limpeza de dicionário definido (no início);
- Informações sobre os padrões encontrados/inseridos (durante);
- Tamanho dos blocos processados (todos + último, no final);
- Número de blocos processados e tamanho do ficheiro processado (no final);
- Lista dos valores estatísticos calculados (no final);
- Tempo de execução total (no final).

Processamento por blocos

Numa ferramenta que precise processar os dados dum ficheiro completo é natural que os dados sejam lidos do ficheiro para processamento em memória uma vez que o processamento em ficheiro é mais complexo e muito menos eficiente.

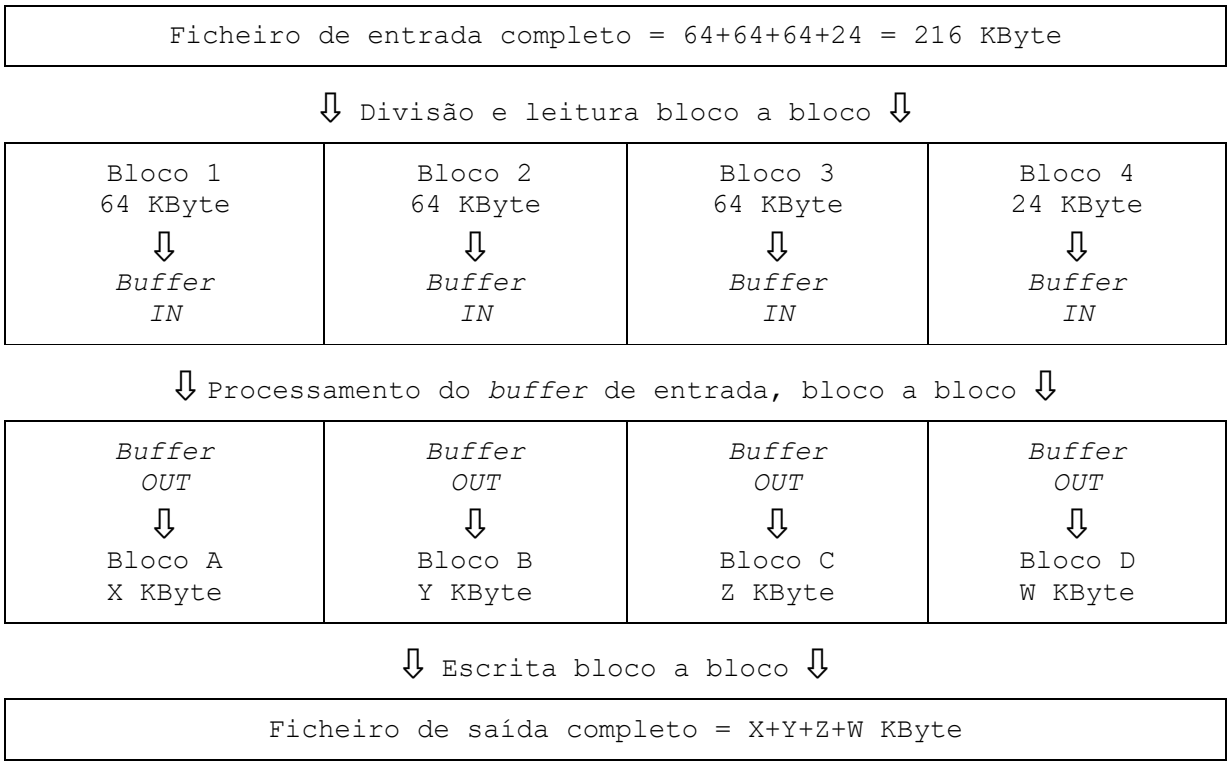

Figura 1: Exemplo de processamento por blocos num ficheiro de 216 KByte.

Assim, esta ferramenta deve ler os dados do ficheiro de entrada em blocos para um *buffer* em memória onde depois irão ser processados, um bloco de cada vez. Os resultados do processamento de cada bloco também devem ser guardados num outro *buffer* em memória que só deve ser processado ou gravado num ficheiro de saída quando o bloco completo tiver sido processado. Ou

seja, a utilização de *buffers* em memória deve ser feita tanto para armazenamento temporário dos dados do ficheiro de entrada a processar como para armazenamento temporário dos resultados desse processamento.

O tamanho de cada bloco define-se, por defeito, como 64 KByte. Outros tamanhos maiores (no máximo até 64 MByte) podem ser definidos pelo utilizador.

De notar que o tamanho do *buffer* de saída (que vai conter os resultados do processamento dos dados no *buffer* de entrada) não será igual ao tamanho do *buffer* de entrada. O seu tamanho depende da compressão específica implementada no processamento.

A Figura 1 apresenta um exemplo esquemático da divisão em blocos de 64 KByte dum ficheiro de entrada de 216 KByte realizada no módulo da primeira fase.

Funcionalidades e otimizações opcionais

Não é necessário a ferramenta desenvolvida saber fazer a descodificação dos ficheiros **lzwdr** para se obter os ficheiros de entrada originais.

Não sendo obrigatório, o trabalho pode ser valorizado implementando otimizações nos mecanismos de procura e gestão dos padrões.

Também é possível melhorar o trabalho desenvolvido incluindo as seguintes funcionalidades adicionais, entre outras:

- O tamanho dos blocos para processamento do ficheiro de entrada pode ser configurado para ser diferente de 64 KByte;
- O tamanho máximo do dicionário pode ser configurado para ser diferente de 2^{16}, por exemplo, valores entre 2^{12} até 2^{24} são admissíveis;
- O tipo de dicionário inicial pode ser diferente do usado pelo LZW; pode, por exemplo, ser nulo ou ter um conjunto inicial de entradas que inclua padrões que já se sabe à partida terem grande probabilidade de ser encontrados dependendo do tipo de ficheiro a codificar;
- O tipo de limpeza do dicionário pode incluir uma limpeza completa do dicionário (*reset*), i.e., voltar ao dicionário inicial quando o número máximo de entradas é atingido, ou uma limpeza eliminando apenas os padrões menos usados, etc.;
- Podem ser calculadas algumas informações estatísticas adicionais como, por exemplo, o tamanho médio dos padrões usados nos códigos de saída, o número médio de bytes processados por cada código de saída, o número de vezes que cada código de saída é repetido, o número total de vezes que os códigos de saída mais comuns são usados, a distância média e a distância máxima entre os valores dos códigos de saída consecutivos, o número total de códigos de saída únicos, etc.;
- Durante a execução, em cada iteração, podem ser impressas informações adicionais sobre os padrões como, por exemplo, os padrões novos encontrados e inseridos no dicionário, a distância para o valor do código de saída da iteração anterior, etc.

Quaisquer otimizações e funcionalidades extra devem ser bem documentadas

por forma a avaliação poder refletir a sua inclusão.

Relatório e outras recomendações

O trabalho deve ser desenvolvido em grupos de, no máximo, dois alunos.

A ferramenta criada deve ser testada com ficheiros de teste de vários tamanhos e de vários tipos (ficheiros de texto, imagem, vídeo, PDF, etc.). O docente também fornecerá alguns ficheiros ou indicará apontadores para esses recursos *online*.

O código deve utilizar apenas funções normalizadas da linguagem C, de preferência da norma 217 ISO (STD 17), e ou funções desenvolvidas pelo grupo de trabalho. Não use APIs, funções ou excertos de código de terceiros que não tenham sido fornecidos ou aprovados pelo docente.

O código deve ser claro e usar convenções de nomeação de variáveis, tipos, funções e constantes. O código deve ser estruturado duma forma o mais modular possível, sem complexidades desnecessárias, e permitir reutilização sempre que possível.

A qualidade e correção do código não se mede pelo seu tamanho.

Os alunos devem documentar/explicar o código criado através de comentários nas secções mais relevantes e no relatório. Todos os ficheiros do código devem ter um cabeçalho com a informação relevante que identifique os seus autores e explique as principais funções criadas, tipos de dados e variáveis usadas, etc.

O relatório deve incluir:

- Na primeira página, o título do trabalho e a identificação dos autores (incluindo fotografia), universidade, curso, unidade curricular e data de entrega;
- Um índice do conteúdo;
- Uma secção com a discussão das estratégias escolhidas, as opções tomadas e os mecanismos adotados, incluindo eventuais otimizações;
- Uma secção com a explicação e análise crítica (apenas) das principais funções implementadas, os seus principais méritos e a suas limitações mais importantes;
- Uma secção com a análise crítica dos resultados dos testes efetuados, tentando explicar o mais detalhadamente as razões lógicas para os resultados encontrados;
- Uma secção de conclusões que inclua uma eventual discussão sobre o que gostava de ter feito melhor ou de ter acrescentado e não conseguiu;
- Uma lista de eventuais referências bibliográficas, artigos científicos ou recursos informais na web e que tenham sido úteis.

O relatório é para ser avaliado pelo docente por isso não se deve incluir informação genérica e irrelevante que o docente já conhece. Deve aspirar-se concisão e clareza. Sempre que se incluir uma afirmação importante no contexto do relatório e que seja de autoria de terceiros, ou que seja baseada diretamente em afirmações de terceiros ou concluída de informação retirada de recursos alheios, deve referenciar-se corretamente essas

autorias ou proveniências e acrescenta-las na lista das referências.

O relatório pode incluir algumas partes relevantes do código quando estas ajudam às análises e justificações apresentadas. Não se deve integrar código desnecessário no texto do relatório. Sempre que possível, comente-se antes o próprio código.

Todo o material entregue deve juntar-se num único ficheiro zip. Este ficheiro zip deve conter um ficheiro PDF com o relatório e todos os ficheiros do código do projeto. O nome do ficheiro zip deve ser igual a SRAM-2023-2024-*Número_Aluno_A-Número_Aluno_B*.zip, como, por exemplo, SRAM-2023-2024-83974-87766.zip. O ficheiro principal de código deve chamar-se **lzwdr.c**.

Por fim, é recomendável que, durante a defesa do trabalho, se responda honesta e concisamente apenas às questões colocadas por forma a que as sessões de apresentação não se arrastem para além dos 20-30 minutos.

ANEXO – Especificação do algoritmo LZWdR

Considere-se a sequência de símbolos $S=S_1S_2...S_N$ na entrada de dados, em que cada símbolo S_i pode assumir um de K valores possíveis dum alfabeto $A=\{X_1,X_2,...,X_K\}$. Por conveniência, $S[i]=S_i$ e $A[i]=X_i$.
Defina-se um dicionário com um máximo de T padrões tal que $D=\{P_1,P_2,...\}$ em que $D[i]=P_i$ é o padrão identificado pelo código/índice i. P_i^* representa a sequência de símbolos invertida de P_i, da direita para a esquerda. Valores típicos: $K=2^8$, $T=2^{20}$.
Inicia-se o dicionário de padrões D com os K padrões de símbolos individuais de A.
Processar S em pares consecutivos de padrões conhecidos $P_a|P_b$ (os maiores já existentes em D), acrescentando ao dicionário: i) os padrões novos formados por concatenação de P_a com todos os padrões que estão em P_b e ii) todos os padrões novos formados ao inverter os padrões obtidos em i).
Enviar para a saída o código/índice de P_a.
Voltar a processar S em pares consecutivos de padrões a partir do primeiro símbolo de P_b, i.e., $P_a=P_b$, repetindo os passos anteriores até não haver mais símbolos em S para processar depois de P_b.
Terminar enviando para a saída o código/índice de P_b.

LZWdR non-optimized algorithm

Let's assume there's:

i) a search(P,D) function that returns the code/index of pattern P in D (if P is not in D the function returns zero);
ii) an insert(P,D) function that inserts the pattern P in D returning its code/index (if P is already in D*, or if D is full**, the function inserts nothing & returns zero*, or T**);
iii) an output(c) function that sends the code c to the output sequence iv) the sequence S must be longer than one symbol (N>1);
iv) a concat(Pa,Pb) function that returns the concatenation of the two input patterns, i.e., Pa|Pb;
v) a reverse(P) function that returns the inverted pattern of P (P in reverse order);

```
vi)  S[i,j] or P[i,j] represents the sequence/pattern beginning on the
     ith symbol up to the jth symbol in S or P;
vii) the first element of a sequence or pattern is S[1,1] or P[1,1].

1.-- Inserting all symbols from A as patterns in D.
  -- The first Pa is just the first symbol of S and it is already in D.
for (i=1 up to K) insert(A[i],D)
Pa=S[1,1]; ind=2

2.-- Processing S to find the bigger pattern Pb after Pa already in D.
code=search(Pa,D); Pb=S[ind,1]; i=1
while (ind+i≤N & search(concat(Pb,S[ind+i,1]),D))
     Pb=concat(Pb,S[ind+i,1]); i=i+1

3.-- Send the code of Pa to the output.
output(code)

4.-- Insert in D all the new patterns while D is not full
  --(D maxes out at T entries).
j=1; t=T-1
while (j<=i & t<T)
     t=insert(concat(Pa,Pb[1,j]),D)
     if (t<T) t=insert(reverse(concat(Pa,Pb[1,j])),D)
     j=j+1

5.-- Update variables and return to 3. if it's not the end of S,
  -- otherwise it will end by sending to the output the code of Pb;
  -- even if D is full, it lets keep processing S…
if (ind+i>N) output(search(Pb,D))
else
     ind=ind+i; Pa=Pb; go to 2.
```

Exemplo codificação LZWdR

Sequência a codificar (K=2): "ABABABBABABAABBABBAB"

Passo 1a: Pa="A"=D.1 e Pb="B"=D.2
Passo 1b: Novos padrões D.3="AB"
Passo 1c: Novos padrões D.4="BA"
Passo 1d: Enviar para a saída código de Pa (1); Pa=Pb

Passo 2a: Pa="B"=D.2 e Pb="AB"=D.3
Passo 2b: Novo padrão D.5="BAB"
Passo 2c: Enviar para a saída código de Pa (2); Pa=Pb

Passo 3a: Pa="AB"=D.3 e Pb="AB"=D.3
Passo 3b: Novos padrões D.6="ABA", D.7="ABAB"
Passo 3c: Novo padrão D.8="BABA"
Passo 3d: Enviar para a saída código de Pa (3); Pa=Pb

Passo 4a: Pa="AB"=D.3 e Pb="BABA"=D.8
Passo 4b: Novos D.9="ABB", D.10="ABBA", D.11="ABBAB" e D.12="ABBABA"
Passo 4c: Novos D.13="BBA", D.14="BABBA" e D.15="ABABBA"
Passo 4d: Enviar para a saída código de Pa (3); Pa=Pb

Passo 5a: Pa=”BABA”=D.8 e Pb=”BA”=D.4
Passo 5b: Novos padrões D.16=“BABAB” e D.17=“BABABA”
Passo 5c: Novos padrões D.18=“ABABAB”
Passo 5d: Enviar para a saída código de Pa (8); Pa=Pb

Passo 6a: Pa=”BA”=D.4 e Pb=”ABBAB”=D.11
Passo 6b: D.19/20/21/22/23=”BAA/BAAB/BAABB/BAABBA/BAABBAB”
Passo 6c: D.24/25/26/27/28=”AAB/BBAAB/BBAABA/ABBAAB/BABBAAB”
Passo 6d: Enviar para a saída código de Pa (4); Pa=Pb

Passo 7a: Pa=”ABBAB”=D.11 e Pb=”BAB”=D.5
Passo 7b: Novos padrões D.29=“ABBABB”, D.30=“ABBABBA”, D.31=“ABBABBAB”
Passo 7c: Novos padrões D.32=“BBABBA”, D.33=“BABBABBA”
Passo 7d: Enviar para a saída código de Pa (11); Pa=Pb

Passo 8a: Pa=”BAB”=D.5 e já não existem dados para definir um novo Pb…
Passo 8b: Enviar para a saída código de Pa (5)
Passo 8c: *Fim de dados*

> Resultado final na saída (8+1 índices): 1, 2, 3, 3, 8, 4, 11, 5, 0*

Nota: 0* = valor reservado que significa “Fim de dados”