Algoritmos 2 - Compressão LZ78

Jorge Augusto de Lima e Silva - 2021032005 - UFMG - DCC207

Maio de 2023

1 Introdução

A proposta do trabalho consistia em implementar o algoritmo de compressão LZ78, no contexto de compactação de textos.

O algoritmo foi desenvolvido em 1978 por Abraham Lempel e Jacob Ziv [2]. Uma das grandes vantagens deste algoritmo é sua simplicidade, i.e., a facilidade que se encontra em implementar este algoritmo e, por mais que não seja o Estado da Arte no tópico de compressão de dados, ainda apresenta resultados satisfatórios.

2 Implementação

O funcionamento do algoritmo consiste em encontrar sequencias cada vez maiores no texto, a medida que se caminha nele, e gerar um código que irá representá-la no arquivo. Mais especificamente, queremos encontrar o maior prefixo de uma certa string e substituí-lo por um valor único e que ocupa menos memória.

Esta seção trata de uma implementação em mais alto nível do algoritmo, detalhes em nível do código de como o algoritmo foi implementado podem ser verificados no respositório do github: `https://github.com/jorgesilva2407/LZ78`

2.1 A Estrutura de Dados

Como forma de auxiliar na busca dos prefixos, foi implementada uma Trie Simples, de modo que o caminhamento feito por uma string na árvore, de acordo com seus caracteres, até que chegue a uma folha, representa o maior prefixo desta string que já foi encontrado em algum outro momento do texto.

2.2 A Compressão

Com a estrutura de dados criada, a geração dos códigos foi feita da seguinte forma: iniciamos inserindo um nó raiz na Trie, que representará a string vazia e será codificado pelo inteiro 0. Em seguida, caminha-se sequencialmente pelo arquivo, fazendo a leitura de byte em byte, e verifica se o caractere representado pelo byte encontrado possui um nó associado a ele que é filho do nó que se está agora. Caso este nó não exista, então ele é criado e um código é associado a ele (mais especificamente a string que representa o caminho na árvore até alcançar este nó), retorna-se para a raiz e, no arquivo de saída é escrito o código associado ao nó que se estava e o caractere que acaba de ser inserido. Repita este processo até que o texto acabe e insira o código do nó atual ao final do texto. Desta forma, a compressão está completa.

2.2.1 A Codificação

A escrita dos códigos no arquivo foi feita da seguinte forma: computa-se o maior código usado durante o processo de compressão e utiliza-se a menor quantidade de bytes possível para representá-lo. Desta forma, todos os códigos escritos no arquivo possuem o mesmo tamanho, sendo, no geral, 3 bytes, que são capazes de representar 16,777,216 strings diferentes. Porém, por mais que todas estas strings possam ser representadas, em nenhum dos caso mais de 100,000 strings precisaram ser codificadas, de modo que uma escolha do tamanho do código que fosse pensada na quantidade de bits ao invés de bytes seria algo mais interessante do ponto de vista de melhorar a compressão, mas estaria associada a um custo de tornar a implementação do método muito mais complexa.

2.3 A Descompressão

Podemos interpretar o arquivo de saída do processo de compressão como um registro das inserções na árvore e, como foi codificado que o código do k-ésimo nó que foi inserido na árvore é k, podemos reconstruir o texto com facilidade, sabendo que cada tupla código-caractere (N,C) significa que a sequência representada pelo código N foi lida do texto e ao final foi inserido o caractere C. Assim, para reconstruir a string representada pela tupla (N,C), basta verificar qual foi a string que gerou a N-ésima inserção na árvore; i.e., a N-ésima tupla escrita no arquivo; e concatenar o caractere C ao final dela.

Por fim, se repetirmos o processo para todas as tuplas do arquivo e concatenarmos suas strings associadas, obteremos o arquivo que havia sido comprimido.

Um detalhe que deve ser observado é que ao final do arquivo existe um código N que não possui um caractere ao seu lado, e este pode ser tratado como simplesmente a string associada ao código N, sem a necessidade de concatenar nenhum caractere ao seu lado.

3 Resultados

Como conjunto de testes foram escolhidos 20 dos top 100 livros do Projeto Gutenberg [1], no qual o maior livro tinha 1.8 MB e enquanto o menor tinha 22 KB. A partir disso, foi usado um script bash para compilar o programa, comprimir e descomprimir os 20 arquivos e conferir se existia alguma diferença entre o arquivo antes da compressão e após a reconstrução, e não houve nenhuma em nenhum dos arquivos, mostrando que não houve nenhuma perda de informação durante o método.

Os resultados obtidos estão mostrados na figura da página seguinte. Como podemos ver na figura 2, atingimos uma taxa de compressão média de 1.60, com o pior caso sendo uma taxa de compressão de 1.42 e o melhor de 1.82. Portanto, concluímos que o método apresenta uma grande estabilidade, uma vez que o desvio padrão do conjunto de taxas de compressão é de apenas 0.12, e essa baixa variância é fortemente observada na figura 1, onde é visível que existe uma relação quase linear entre o tamanho do arquivo original e o tamanho do arquivo após a compressão.

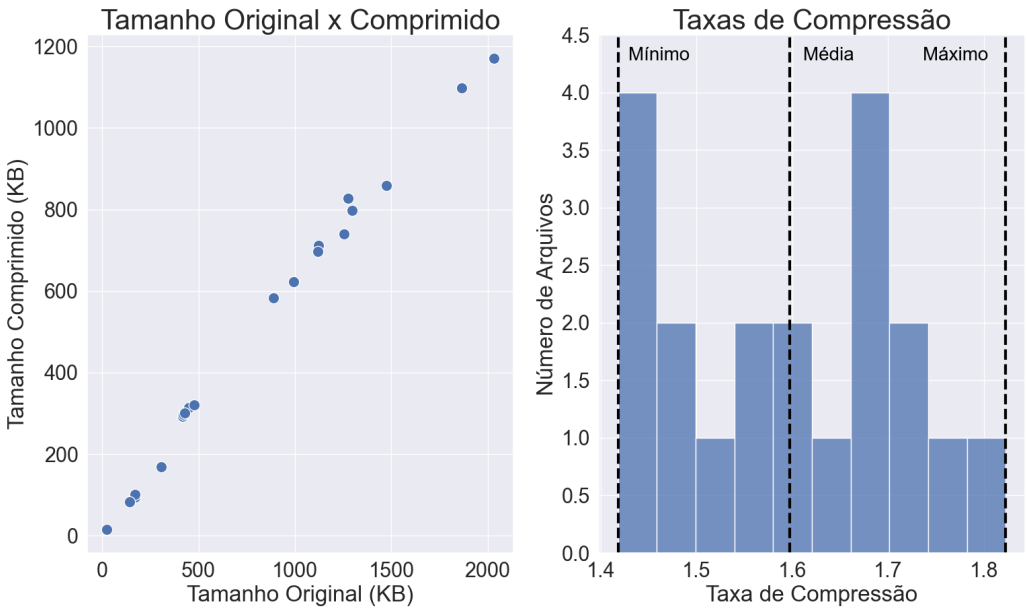

Figura 1:

4 Conclusão

Concluímos, portanto, que por mais que a taxa de compressão ainda seja baixa, a simplicidade do algoritmo é algo que compensa sua implementação. Além do mais, várias otimizações podem ser aplicadas para tornar o método ainda melhor, por exemplo, os tamanhos dos códigos podem ser definidos de uma forma dinâmica ao serem escritos no arquivo, i.e., podemos ter códigos que possuem uma diferente quantidade de bits, o que tornaria o algoritmo ainda mais eficiente ao custo de tornar sua implementação mais complexa.

Referências

[1] *Top 100 EBooks yesterday*. URL: `https://www.gutenberg.org/browse/scores/top`. (accessed: 06.05.2023).

[2] J. Ziv e A. Lempel. «Compression of individual sequences via variable-rate coding». Em: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 530–536. DOI: `10.1109/TIT.1978.1055934`.