



RELATÓRIO

Concepção e análise de algoritmos

Faculdade de Engenharia da Universidade do Porto

28 de Maio de 2013

Duarte Nuno Pereira Duarte

ei11101@fe.up.pt

Miguel Rui Pereira Marques

ei11099@fe.up.pt

Ruben Fernando Pinto Cordeiro

ei11097@fe.up.pt

Conteúdos

Resumo	2
Introdução	3
Requisitos	4
Funcionalidade	4
Usabilidade	4
Estrutura	5
Modelação de dados	5
Diagrama UML	5
Algoritmos implementados	6
<i>Huffman codes</i>	6
Codificação constante	6
Codificação dinâmica	7
<i>Keyword encoding</i>	8
<i>Lempel-Ziv-Welch (LZW)</i>	9
Compressão	9
Descompressão	9
<i>Run-length encoding (RLE)</i>	9
Resultados	10
Medidas de desempenho	10
Conclusões	11
Análise comparativa de dados	11
Limitações	12
Dificuldades encontradas	13
Sugestões para trabalho futuro	14
Distribuição de tarefas	15
Apêndice	16
Referências	19

Resumo

Apesar da relativa abundância de espaço para armazenamento digital e de largura de banda para comunicação, o armazenamento e transferência de dados são dois aspetos importantes no âmbito das tecnologias de comunicação e informação. O facto de grandes volumes de dados serem constantemente processados em computadores pessoais ou na *internet* afirmam a necessidade da sua compressão.

Foram implementados cinco algoritmos distintos:

- Codificação de *Huffman* (constante e dinâmica);
- *Keyword encoding*;
- RLE (*Run-length encoding*);
- LZW (*Lempel-Ziv-Welch*).

As duas versões do algoritmo de codificação de *huffman* inserem-se na categoria de métodos estatísticos, ao passo que os restantes se inserem na categoria de métodos baseados em dicionários. Todos os algoritmos implementados foram avaliados a nível da taxa de espaço libertado, assim como dos tempos de compressão e descompressão de dados. Para este efeito, foi utilizada uma amostra baseada num texto real.

Com base na análise comparativa dos algoritmos, verifica-se que o algoritmo LZW atinge as maiores taxas de espaço libertado em detrimento de tempos de compressão e descompressão maiores, seguidos dos algoritmos de *Huffman*. O algoritmo de *keyword encoding* não teve a capacidade de comprimir a amostra em tempo útil. O algoritmo de RLE apresentou taxas de espaço libertado virtualmente nulas.

Os algoritmos de *Huffman* apresentam tempos de execução mais curtos relativamente ao LZW, no entanto, as taxas de espaço libertado não atingem valores tão elevados.

Introdução

Na Teoria da Informação, compressão de dados corresponde ao acto de codificar informação utilizando menos bits do que o necessário para representá-la na sua forma original, com ou sem perda de conteúdo. No âmbito dos ficheiros de texto, a compressão de dados é conseguida através da minimização da representação de dados repetitivos, tais como caracteres (espaços, vogais) ou palavras repetidas.

A compressão de dados tem aplicações importantes nas áreas de armazenamento e transmissão de informação. Muitas aplicações de processamento de dados exigem o armazenamento de grandes volumes de informação, pelo que a sua compressão reduz os custos operacionais envolvidos na sua manutenção.

Este trabalho envolve uma análise comparativa do desempenho dos diferentes algoritmos de compressão implementados, que são:

- Codificação de *Huffman* (constante e dinâmica);
- *Keyword encoding*;
- RLE (*Run-length encoding*);
- LZW (*Lempel-Ziv-Welch*).

Requisitos

Funcionalidade

Mediante um ficheiro especificado, a aplicação deve ser capaz de:

- Comprimir o ficheiro através dos algoritmos especificados, assim como restaurar o ficheiro original;
- Permitir a análise comparativa dos métodos de compressão.

Usabilidade

Sendo um programa utilitário, a especificação dos parâmetros da aplicação é feita através da linha de comandos. As opções da linha de comandos são as seguintes:

<code>-v [--version]</code>	print version string
<code>-h [--help]</code>	produce help message
Generators:	
<code>--generator arg</code>	Data generator, can be 'words', 'chars' or 'file'.
<code>--gen-min arg (=1)</code>	Minimum number of words or chars.
<code>--gen-max arg (=100)</code>	Maximum number of words or chars.
<code>--gen-count arg (=10)</code>	Number of files to create.
Available algorithms:	
<code>--lzw</code>	Run Lempel-Ziv-Welch
<code>--rle</code>	Run Run-length encoding
<code>--key</code>	Run Keyword encoding
<code>--huffdyn</code>	Run dynamic Huffman coding
<code>--huffsta</code>	Run static Huffman coding
<code>--all</code>	Run all the above
Configuration:	
<code>-c [--compress]</code>	Compress file
<code>-d [--decompress]</code>	Decompress file
<code>-p [--plot]</code>	Create charts and plots comparing the selected algorithms
<code>-s [--stats]</code>	Write simple statistics about compression and/or decompression

Algoritmos implementados

Os algoritmos contemplados neste documento inserem-se em duas categorias principais: métodos estatísticos e métodos baseados em dicionários.

As técnicas de compressão estatística baseiam-se na probabilidade da ocorrência repetida dos símbolos de entrada dos dados a comprimir: símbolos frequentes devem ter uma representação codificada mais compacta relativamente aos símbolos menos frequentes.

As técnicas de compressão baseada em dicionários utilizam estruturas para armazenar segmentos (palavras ou expressões no caso dos ficheiros de texto) repetidos no ficheiro original. No processo de compressão, os segmentos repetidos são substituídos pelo índice respetivo do dicionário.

Devido à construção do dicionário no processo de codificação dos segmentos, os métodos baseados em dicionários têm normalmente tempos de compressão maiores e tempos de descompressão ligeiramente mais rápidos. Pelo contrário, os métodos estatísticos tendem a ser igualmente rápidos tanto no processo de compressão como no processo de descompressão.

Huffman codes

Termos:

- $freq(c) \leftarrow$ número de ocorrências do carácter c no ficheiro original
- $peso(t) \leftarrow$ peso atribuído a uma árvore binária t
- **Código instantâneo:** Conjunto de palavras no qual cada palavra não é um prefixo de outra palavra contida nesse conjunto. Um código instantâneo é representado por uma árvore de codificação, sendo também denominado por código prefixo.
- **Prefixo:** Uma palavra $u \in \Sigma^*$ é um prefixo de uma palavra $w \in \Sigma^*$ sse $w = uz, z \in \Sigma^*$

Codificação constante

O método de codificação constante de *Huffman* transforma o código de representação dos caracteres do ficheiro fonte. Os caracteres podem ocupar um número indeterminado de *bits* consoante a representação de dados. De forma a simplificar a descrição do algoritmo, assume-se que os dados de entrada são ficheiros de texto cujos caracteres são representados em *ASCII*¹ a 8 *bits*.

A noção de **código instantâneo**ⁱ é fundamental para a operação do algoritmo. Este código representa um conjunto de palavras cuja representação não seja um **prefixo**ⁱⁱ de outra palavra desse conjunto. Esta restrição de codificação assegura um processo de descodificação imediato e livre de ambiguidades.

Para o alfabeto binário $\{0, 1\}$, a representação do código prefixo corresponde a uma árvore binária na qual os ramos esquerdos e os ramos direitos de cada nó são representados pelos símbolos 0 e 1, respetivamente.

¹ American Standard Code for Information Interchange

O algoritmo de compressão envolve três operações essenciais:

1. Cálculo do número de ocorrências de cada carácter no ficheiro original;
2. Construção da árvore binária para representar o código instantâneo para o alfabeto do ficheiro;
3. Codificação de cada palavra do ficheiro original para o ficheiro comprimido através da árvore construída no passo anterior.

No passo 2 do algoritmo, a construção da árvore tem em conta a frequência relativa de cada carácter. Sendo $freq(c)$ a frequência da ocorrência do carácter c no texto original, a construção da árvore binária é feita da seguinte forma:

- Criar uma árvore t para cada carácter c do alfabeto do ficheiro original. O peso de cada árvore será igual à frequência relativa do carácter correspondente, $peso(t) = freq(c)$;
- Seleccionar e extrair duas árvores $t1$ e $t2$ com o menor peso e torná-las sub-árvores de uma nova árvore $t3$, em que $peso(t3) = peso(t1) + peso(t2)$;
- Repetir a rotina anterior até sobrar uma única árvore com a codificação de todos os caracteres.

Após a construção da árvore, é possível determinar o código binário de cada carácter através de uma simples pesquisa em profundidade

Uma vez que o código de cada carácter depende do texto original, toda a estrutura da árvore terá que ser guardada no ficheiro comprimido.

Na implementação do projeto, a estrutura da árvore foi substituída por uma tabela de dispersão cujas chaves são os caracteres e um vetor de cujas posições correspondem aos índices das chaves da tabela, uma vez que os códigos representativos de cada caracter são sequenciais. Esta implementação assegura tempos de inserção e acesso constantes.

Codificação dinâmica

As principais desvantagens da variante estática do algoritmo de *huffman* são:

- O cálculo das frequências dos caracteres para a respetiva construção da árvore de codificação e respetiva escrita para o ficheiro comprimido implicam uma leitura dupla do ficheiro original;
- A informação da árvore de codificação tem que obrigatoriamente ser escrita no ficheiro comprimido.

Estas limitações são ultrapassadas através da implementação dinâmica do algoritmo de *huffman*: a árvore de codificação é atualizada a cada símbolo que é lido do ficheiro original. A eficiência deste método é assegurada na *siblings property* (Cormack & Morspool, 1984).

Siblings property: Sendo T uma árvore de *huffman* (árvore binária) com n folhas. Os nós de T podem ser ordenados numa sequência $(x_0, x_1, \dots, x_{2n-2})$ de forma a que:

1. a sequência de pesos $(p(x_0), p(x_1), \dots, p(x_{2n-2}))$ está disposta por ordem decrescente;
2. Para qualquer i , $(0 \leq i \leq n-2)$, os nós consecutivos x_{2i+1}, x_{2i+2} são irmãos (têm o mesmo nó como raiz).

Esta propriedade assegura a eficiência do algoritmo e a criação de uma árvore de codificação o mais compacta possível.

Keyword encoding

O processo de compressão deste algoritmo consiste em substituir palavras muito comuns por uma sequência composta por um ou mais caracteres especiais. As palavras são substituídas de acordo com uma tabela de frequências: consoante o número de caracteres especiais a atribuir, as palavras mais frequentes tem prioridade em relação às palavras menos frequentes. O texto com as palavras codificadas e a tabela de frequências são escritas para o ficheiro comprimido. O processo decodificador reverte a decodificação de acordo com a tabela de frequências. Os caracteres especiais não podem constar no ficheiro original.

Lempel-Ziv-Welch (LZW)

Sendo um método baseado em dicionários, LZW é um algoritmo de compressão de dados que tira partido da repetição de padrões de dados. Todas as subcadeias são armazenadas no dicionário de forma a que no processo de codificação sejam apenas escritos inteiros que referenciam a posição das subcadeias no mesmo. Como qualquer método de compressão dinâmico, a estrutura do dicionário é inicializada numa primeira instância, sendo posteriormente modificada à medida que os blocos de dados do ficheiro original são lidos. Apesar do algoritmo ser explicado no contexto da compressão de ficheiros de texto, pode também ser usado para qualquer tipo de ficheiro, embora este seja mais eficiente em ficheiros com subcadeias repetidas.

Compressão

O dicionário é inicializado com os 256 caracteres da tabela da representação *ASCII*. Posteriormente, o algoritmo resume-se às seguintes operações:

- Ler um carácter do ficheiro original. Caso a exista uma entrada para a cadeia de caracteres a processar, é escrito um inteiro correspondente ao índice dessa entrada para o ficheiro comprimido. Cada vez que uma nova subcadeia é detectada, é adicionada uma nova entrada para a mesma, caso a subcadeia já exista no dicionário, esta é concatenada com o próximo carácter a ler;
- Repetir o processo anterior até não existirem mais dados a comprimir.

De forma a que o processo não esgote a memória disponível, é definido um número máximo de entradas para a estrutura do dicionário.

Descompressão

O processo de descompressão implica a reconstituição do dicionário usado para a codificação dos dados; para este efeito, os dicionários iniciais de ambos os métodos têm que ser iguais (256 caracteres em *ASCII* neste caso).

Operações:

- Ler um inteiro correspondente a um índice do dicionário. A subcadeia associada ao índice é escrita no ficheiro decodificado. O primeiro carácter desta subcadeia é concatenado à cadeia acumulada até à altura. Esta nova concatenação é adicionada ao dicionário, de forma a simular o processo de codificação das subcadeias;
- Repetir a operação anterior até não restarem índices codificados no ficheiro comprimido.

Run-length encoding (RLE)

Técnica de compressão de dados através da supressão de sequências longas de caracteres repetidos. Este algoritmo é pouco eficiente para compressão de texto, uma vez que sequências longas de caracteres repetidos (3 ou mais símbolos) são muito raras.

Resultados

Medidas de desempenho

O principal objetivo por detrás da compressão de dados consiste em maximizar o fator de compressão num intervalo de tempo o mais breve possível. Frequentemente, as métricas usadas para avaliação do desempenho de cada algoritmo são a taxa de compressão, assim como o tempo de codificação e decodificação.

Sendo T_c a taxa de compressão:

$$T_c = \frac{\text{Tamanho do ficheiro comprimido}}{\text{Tamanho do ficheiro original}}, 0 \leq T_c \leq 1$$

Os tempos de execução dos métodos de compressão e decodificação para cada algoritmo englobam o tempo de CPU² assim como o tempo das operações de escrita e leitura em disco.

De forma a tornar a análise comparativa de dados mais inteligível, usar-se-á o espaço libertado ao invés da taxa de compressão para a avaliação do desempenho dos algoritmos a nível de espaço.

Sendo E_L o espaço libertado pelo algoritmo:

$$E_L = 1 - T_c, 0 \leq E_L \leq 1$$

Uma vez que grande parte dos métodos implementados nesta aplicação são baseados em dicionários, é importante que as amostras de teste não sejam geradas de forma aleatória, caso contrário os resultados correm o risco de serem uma aproximação grosseira dos casos de utilização reais.

Para efeitos de teste, foi utilizado um ficheiro de texto de nome *kjv10.txt*, disponibilizado como anexo na pasta do projeto.

Os dados comparativos para os tempos de compressão/descompressão, assim como espaço libertado para os vários algoritmos podem ser visualizados no [apêndice](#).

² Central processing unit

Conclusões

Análise comparativa de dados

A partir da análise das figuras 1 e 2 do apêndice pode-se constatar que o algoritmo mais eficiente ao nível da percentagem espaço libertado é o *Lempel-Ziv-Welch* (60%), seguindo-se as versões dinâmica e estática de *Huffman* (42% e 12%, respectivamente). O método de RLE apresenta uma percentagem de espaço libertado virtualmente nula. Ao contrário dos restantes algoritmos, o método de *keyword encoding* não tem a capacidade de comprimir a amostra em tempo útil, sendo por isso impossível gerar quaisquer dados de análise do mesmo, algo que é indicador das enormes limitações do algoritmo a nível da sua eficiência temporal.

A nível dos tempos de compressão/descompressão, verifica-se que o algoritmo mais rápido é o RLE, sendo seguido das suas versões do algoritmo de *Huffman*. Apesar de gerar o ficheiro mais compacto, o algoritmo de LZW é o mais lento tanto a nível do processo de compressão como no processo de descompressão.

É imediatamente perceptível que os melhores algoritmos são efectivamente as duas versões de *Huffman*, assim como o LZW. A discrepância entre estes dois últimos pode ser facilmente explicada através das categorias a que estes dois métodos pertencem. O algoritmo de LZW, sendo um método baseado em dicionários, apresenta tempos de compressão e descompressão superiores ao algoritmo de *Huffman*, que por sua vez pertence à categoria de métodos estatísticos.

A nível da evolução dos tempos de compressão/descompressão assim como da taxa de espaço libertado em função do número de caracteres para a mesma amostra (figuras 3, 4, 5, e 6), verifica-se que os tempos de execução de todos métodos crescem de forma linear, ao passo que a taxa de espaço de libertado estabiliza sensivelmente a partir dos 886660 caracteres. Importa referir que a taxa de espaço libertado do *Run-length encoding* atinge valores muito reduzidos a partir dessa marca.

Limitações

A nível de funcionalidade da aplicação, a enorme ineficiência temporal do algoritmo de *keyword encoding* inviabiliza a compressão de grande parte dos ficheiros de entrada. No âmbito da usabilidade da mesma, e apesar desta ser uma aplicação utilitária, a utilização interface de linha de comandos pode eventualmente ser demasiado técnica para uma certa percentagem de utilizadores.

Dificuldades encontradas

A geração de gráficos para a análise comparativa de dados por parte da aplicação constituiu um desafio.

Sugestões para trabalho futuro

No decorrer do trabalho experimental foram feitas opções que determinaram um rumo a seguir, outras opções poderiam ter sido tomadas e certamente outros resultados seriam encontrados e novas perspectivas se abririam. Assim como esta análise dos dados obtidos não representa o fim do projeto, mas antes apenas uma parte do mesmo, são feitas algumas sugestões para trabalho a desenvolver que visam não só complementar o trabalho realizado como abrir novos percursos de a no campo da teoria da informação e compressão de dados.

Para os algoritmos inseridos na categoria dos métodos estatísticos, sugere-se que se avalie a eficiência da utilização de tabelas de frequências pré-estabelecidas, a fim de avaliar até que ponto a análise sintática e semântica das linguagens beneficiaria os tempos de execução dos mesmos.

Sugerem-se também investigações sobre o tópico da compressão em cadeia, averiguando as melhores configurações e combinações de métodos de forma a maximizar as taxas de espaço libertado dentro de tempos de execução aceitáveis. Um algoritmo não implementado que faz uso deste conceito é o *Lempel-Ziv-Markov* (LZMA).

Distribuição de tarefas

As tarefas de desenvolvimento da aplicação foram distribuídas de forma equitativa por todos os elementos do grupo.

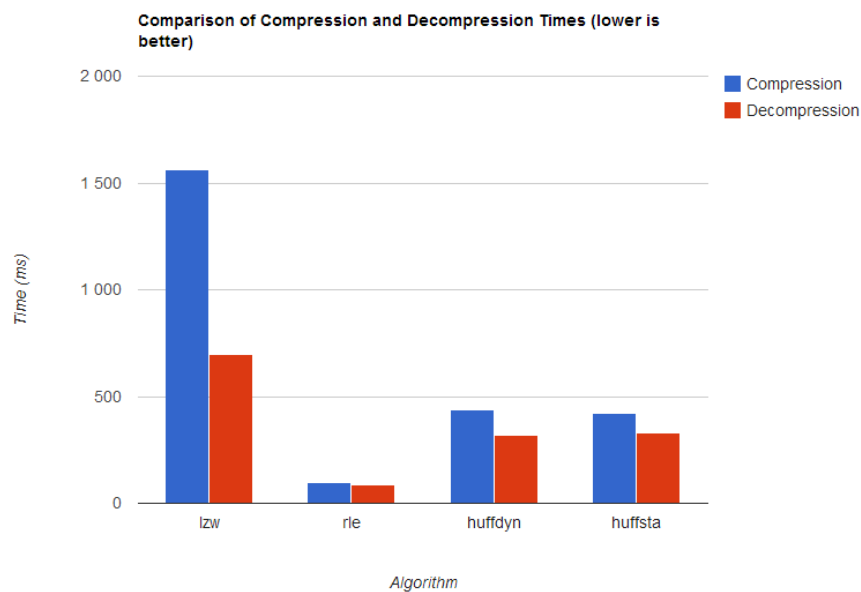


Figura 1: Comparação entre tempos de compressão e descompressão

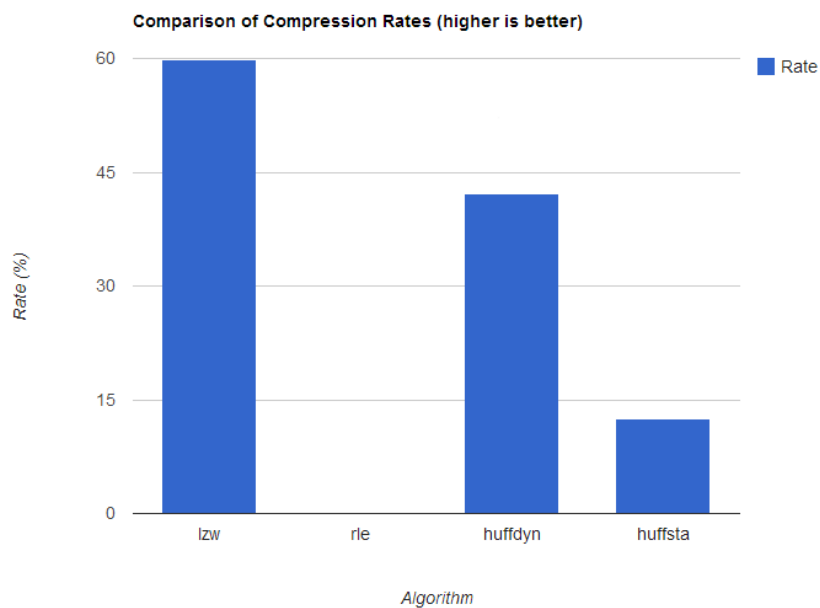


Figura 2: Comparação entre o espaço libertado de cada algoritmo

Generator: file
File: kv10.txt
Size: 3.8 MB
Min: 10000 - Max: 4000000 - Count: 10

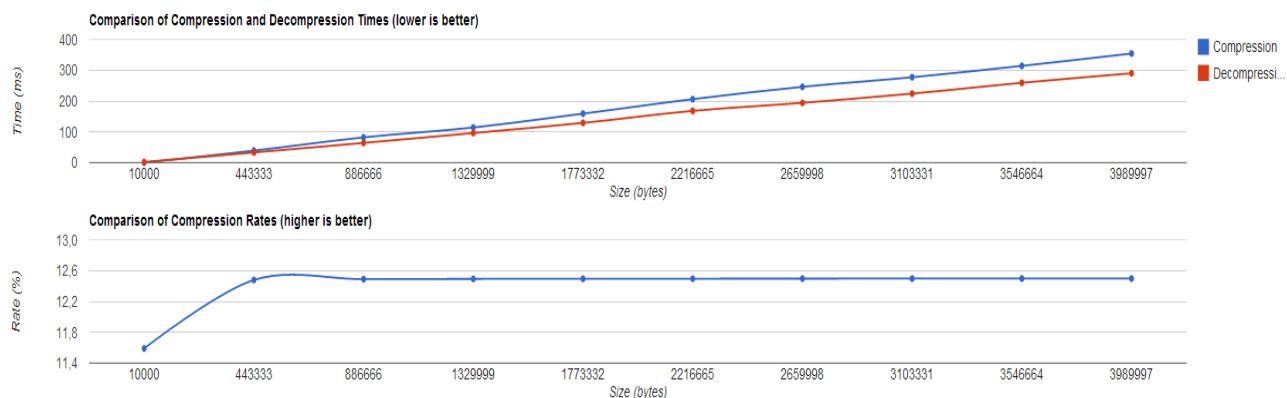


Figura 3: Evolução dos tempos de compressão/descompressão e espaço libertado em função do número de caracteres para a versão estática de Huffman

Generator: file
File: kv10.txt
Size: 3.8 MB
Min: 10000 - Max: 4000000 - Count: 10

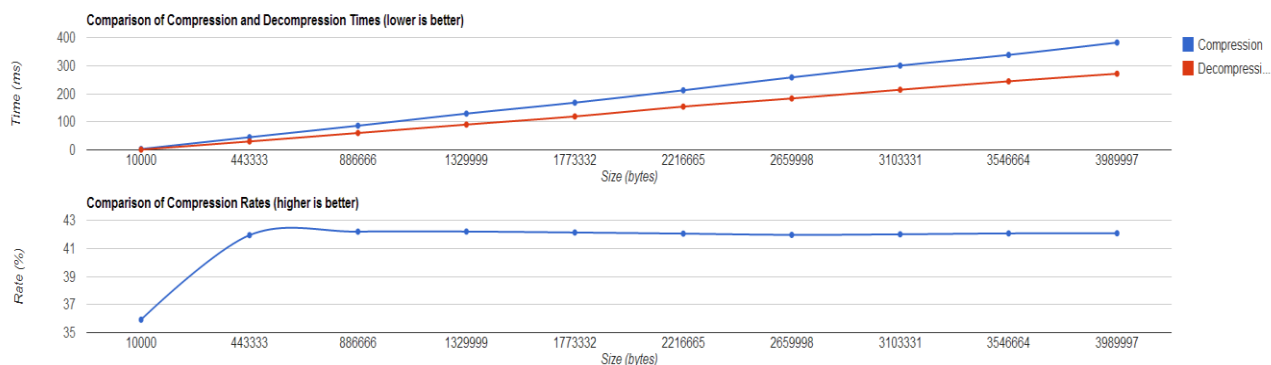



Figura 4: Evolução dos tempos de compressão/descompressão e espaço libertado em função do número de caracteres para a versão dinâmica de Huffman

Generator: 
 File: kjv10.txt
 Size: 3.8 MB
 Min: 10000 - Max: 4000000 - Count: 10

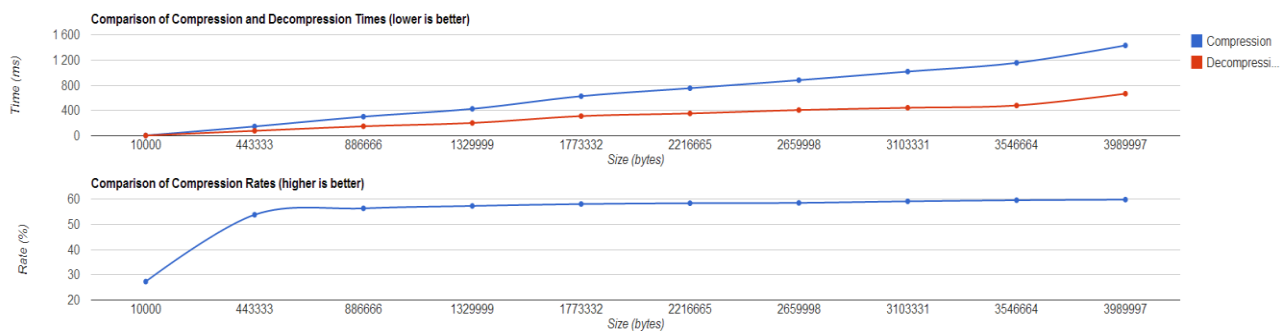



Figura 5: Evolução dos tempos de compressão/descompressão e espaço libertado em função do número de caracteres para o algoritmo de *Lempel-Ziv-Welch* (LZW)

Generator: 
 File: kjv10.txt
 Size: 3.8 MB
 Min: 10000 - Max: 4000000 - Count: 10

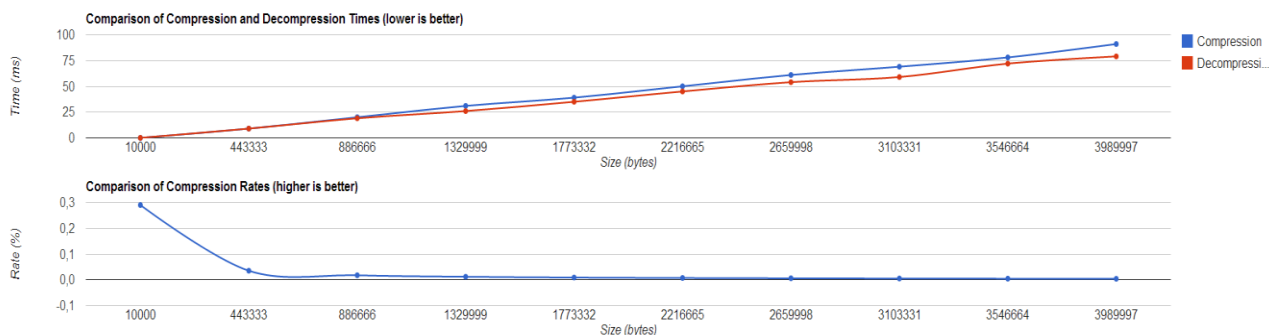


Figura 6: Evolução dos tempos de compressão/descompressão e espaço libertado em função do número de caracteres para o algoritmo de *Run-length encoding* (RLE)

Referências

Cormack, G. V., & Morspool, R. N. (1984). Algorithms for adaptive Huffman Codes. *Info. Process. Lett.*

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms*. The MIT Press.
