Ferramenta IPMT

Valdemir de Andrade - vabj@cin.ufpe.br
Mateus de Freitas Leite - mfl3@cin.ufpe.br

1. Introdução

O *ipmt* é um programa utilizado para indexação, compressão e buscas de padrões em arquivos de textos comprimidos. Possui as seguintes interfaces de uso:

$ *ipmt* <*mode*> [*options*] *pattern indexfile* [*indexfile*...]

Aqui, <*mode*> corresponde a uma funcionalidade da ferramenta. Existem dois modos de execução: *index* (indexação e compressão de texto) e *search* (busca em textos previamente comprimidos por esta ferramenta). O parâmetro *pattern* representa o padrão a ser procurado no texto (apenas válido para o modo de busca). Por fim, *indexfile* representa, no modo de indexação, o arquivo de texto a ser comprimido, e, no modo de busca, o arquivo comprimido que contém o texto a ser processado à procura do padrão especificado. Mais de um arquivo de texto pode ser especificado na entrada. A ferramenta também dá suporte à utilização de *wildcards*.

As opções do modo de indexação são as seguintes:

- *-c --compression*: determina o algoritmo de compressão utilizado nessa etapa. Há dois algoritmos de compressão implementados por essa ferramenta; as opções disponíveis são “huffman” (Algoritmo de Huffman) e “lz78” (Algoritmo de Lempel-Ziv, 1978).
- *-i --indexfile*: especifica qual a estrutura de indexação a ser utilizada. Há apenas uma estrutura implementada nesta ferramenta: o argumento disponível para esse modo é “sa” (Vetor de Sufixos, Manber e Myers, 1991).

As opções do modo de busca, por sua vez, são essas:

- *-c --count*: caso essa opção esteja habilitada, a ferramenta imprime apenas o número de ocorrências do padrão no texto.
- *-p --pattern*: caso essa opção seja selecionada, a ferramenta interpretará o parâmetro *pattern* como um arquivo de texto contendo todos os padrões a serem procurados no texto.

1.1. Divisão de tarefas

Inicialmente, os módulos foram divididos da seguinte maneira: a implementação da estrutura de vetor de sufixos ficou por parte de Valdemir, enquanto o algoritmo de compressão ficou sob a responsabilidade de Mateus. No entanto, devido ao fator tempo para trabalhar nos projetos do final do período, a dupla precisou trabalhar em conjunto na implementação da estrutura de índice. Os testes foram executados também foram executados por ambos.

2. Implementação

2.1. Algoritmos de indexação

Para a indexação, foi utilizado o algoritmo de vetor de sufixos. Esse algoritmo faz uma varredura no texto, incluindo todos os sufixos do texto e depois ordenando-os. O algoritmo utilizado para a construção do vetor de sufixos é o de Manber e Myers (1991), mas sem a utilização do array de LCPs (*Longest Common Prefixes*). Desta maneira, há uma perda de eficiência na funcionalidade de busca de padrões no texto, visto que é necessário utilizar uma busca binária de complexidade $O(m \lg n)$, onde m, n são os tamanhos do padrão e do texto, respectivamente.

2.2. Algoritmos de compressão

Para a compressão, foram utilizados os algoritmos de Huffman e Lempel-Ziv (1978). Por padrão, a ferramenta utiliza o algoritmo de Huffman para compressão, visto que a diferença entre as taxas de compressão obtidas nos testes não foi significativa em relação ao tempo de execução para arquivos de texto grandes, além de esse algoritmo não necessitar de uma quantidade muito grande de memória em relação ao LZ78. Além do mais, o dicionário utilizado no LZ78 não é muito eficiente para compressão de textos pequenos, visto que não há muita repetição de padrões, o que pode resultar num aumento do texto codificado. Entretanto, para todos os arquivos utilizados nos testes, o LZ78 apresentou uma taxa de compressão maior que a do algoritmo de Huffman. Também foi possível notar que o tempo de descompressão do LZ78 era, em geral, menor que o do algoritmo de Huffman.

2.3. Limitações da ferramenta

Como já mencionado na Seção 2.1, não foi possível implementar a tempo a estrutura de array de LCPs, a fim de otimizar a busca de padrões no texto utilizando o vetor de sufixos. Além disso, embora os textos originais sejam comprimidos, seus índices (ou seja, seus

respectivos vetores de sufixos) não são, pois ocorreu um *bug* na leitura dos índices comprimidos na etapa de descompressão do arquivo comprimido. Por questão de tempo, não conseguimos solucionar a tempo esse problema.

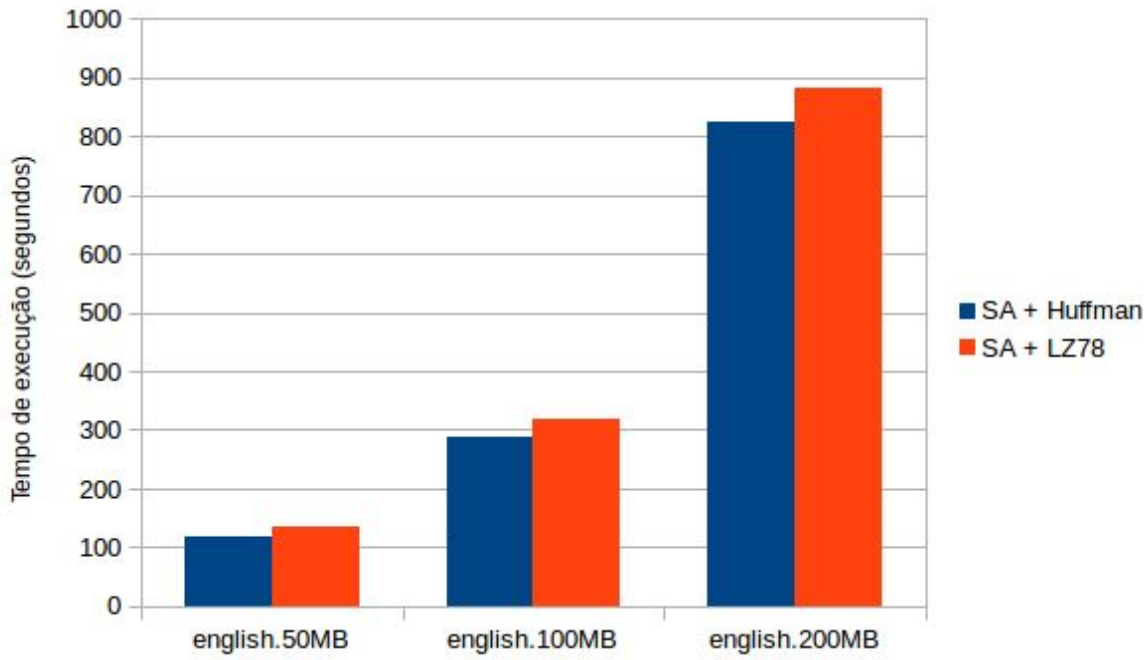
3. Testes e Resultados

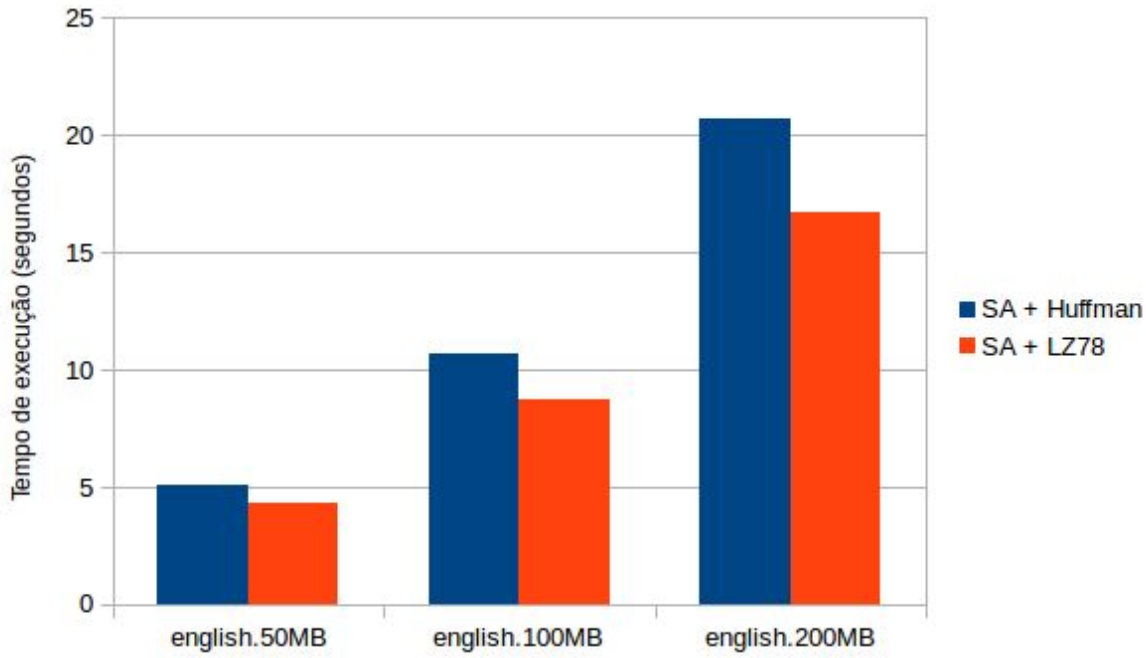
Inicialmente, foram executados testes com arquivos de texto em inglês, a fim de medir o tempo de execução dos algoritmos de indexação e compressão implementados nesta ferramenta. Os arquivos possuíam, respectivamente, 50 MB, 100 MB e 200 MB, e foram extraídos do *corpus* do *Pizza&Chili*. Os testes foram executados em uma máquina com processador Intel Core i7-4500U (1.8 GHz), sistema operacional Ubuntu 16.04 e 8 GB de memória RAM. Em seguida, foram analisadas as taxas de compressão dos algoritmos implementados nesta ferramenta. Como *benchmark*, utilizou-se o *gzip* e calculou-se também sua taxa de compressão. Os arquivos utilizados foram os mesmos do testes iniciais. De maneira mais formal, sejam n e n_c os tamanhos dos textos original e comprimido, respectivamente. O cálculo da taxa de compressão é dado da seguinte forma:

$$T = 1 - n_c / n$$

3.1. Desempenho

Nos gráficos abaixo, o eixo horizontal contém os nomes dos arquivos utilizados nessa etapa de testes, enquanto o eixo vertical representa o tempo de execução da ferramenta em cada um dos modos de execução. O primeiro gráfico diz respeito ao desempenho do modo de indexação da ferramenta, e o segundo, ao do modo de busca.

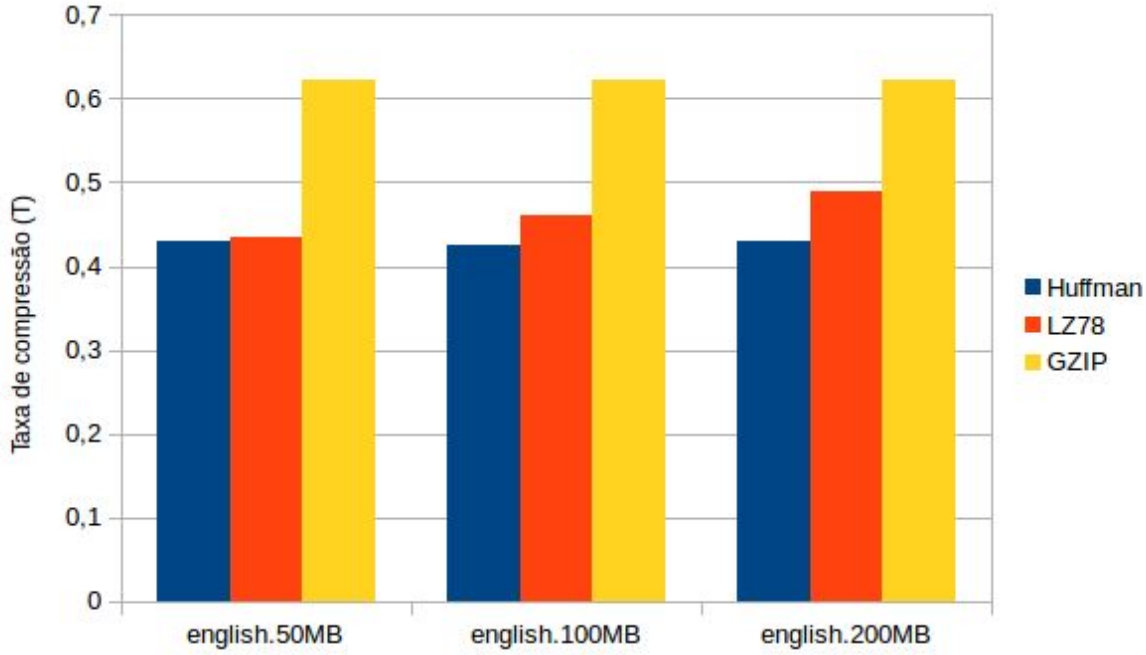




O critério principal para a escolha do algoritmo de compressão padrão da ferramenta foi seu tempo de execução. Embora seja possível perceber no segundo gráfico que, durante a etapa de busca (e, por conseguinte, de descompressão do texto), o LZ78 obteve um melhor desempenho que o algoritmo de Huffman, esse cenário não se repete na etapa de indexação e compressão. Como a ordem de grandeza do tempo de execução da busca é menor que a da indexação (especialmente por causa da construção do vetor de sufixos), o algoritmo de Huffman foi eleito o método padrão da ferramenta.

3.2. Taxas de compressão

Embora o LZ78 não tenha obtido um bom desempenho na etapa de compressão comparado ao algoritmo de Huffman, ele apresentou uma taxa de compressão maior. Isso se deve pelo de que, em arquivos de texto grandes, é mais provável que ocorra repetições de padrões, o que aumenta a eficiência do dicionário usado no LZ78. O algoritmo de Huffman, por sua vez, só considera a frequência dos *caracteres*, logo o conjunto de códigos utilizados possui, no máximo, a mesma cardinalidade do alfabeto utilizado para representar o texto. O resultado é que não se consegue aproveitar da melhor maneira a disposição dos padrões de *cadeias de caracteres* no texto. O gráfico abaixo mostra essa diferença.



Em contrapartida, nenhum dos dois algoritmos conseguiu uma taxa de compressão superior à ferramenta de *benchmark* utilizada nos testes, o *gzip*. Enquanto a maior taxa obtida pelo *ipmt* foi de 48,95% (LZ78, no arquivo de 200 MB), as taxas do *gzip* ficaram acima dos 62%. Vale ressaltar que, enquanto o LZ78 foi mais sensível à mudança no tamanho do texto, tanto o GZIP quanto o algoritmo de Huffman não tiveram mudanças significativas.

4. Conclusão

No que diz respeito aos métodos de compressão, apesar de os algoritmos implementados nesta ferramenta se comportarem de maneira bastante eficiente nos casos de teste, os experimentos mostraram que eles não são as melhores opções para compressão de textos razoáveis, ou seja, que não sejam gerados aleatoriamente, ou que não possuam um alfabeto muito restrito.

Quanto ao módulo de indexação, embora a etapa de busca não tenha sido feita de uma maneira eficiente, percebeu-se que o tempo de processamento não chegava a ser muito grande para arquivos de tamanhos razoáveis, o que mostra a importância das estruturas de indexação para efetuar *string matching* de padrões relativamente pequenos em textos muito grandes. Otimizações para o algoritmo de Manber e Myers também podem ser realizadas, como a proposta por Kärkäinnen e Sanders (2003), para a construção de vetores de sufixos, o que pode eliminar o visível gargalo notado nos testes de desempenho para o modo de indexação.