Trabalho prático 1 da disciplina Algoritmos 2

Matheus Farnese Lacerda Senna – 2021031939

1. Introdução

Essa documentação trata da implementação de um compressor de arquivos texto com base no algoritmo LZ78. Tal algoritmo consiste em criar uma árvore de prefixos para servir como um dicionário que substitui sequências de texto já lidas por um inteiro, reduzindo assim o número de bytes necessários para armazenar toda a informação do arquivo.

2. Método

O código foi dividido em 2 arquivos: “trie.cpp”, que contém a implementação da árvore trie de prefixos, bem como funções relacionadas e “main.cpp”, que contém a implementação da compressão e descompressão, bem como da geração dos arquivos de saída relacionados.

2.1) trie.cpp: Implementa duas classes. A primeira é “Node”, cujos atributos são definidos em “trie.hpp”: index, que armazena o inteiro associado ao nó, esse que serve como código para a substituição de sequências já lidas; token e token_sec para guardar o caractere associado ao nó, onde token_sec é utilizado para guardar caracteres acentuados, pois esses não são codificados diretamente pela tabela ascii; children é um vetor para armazenar os filhos do respectivo nó e children_size é um inteiro para controlar o tamanho do vetor mencionado.

A segunda classe, também definida no header, é “Tree”, que implementa a árvore de prefixos trie que servirá como dicionário. Essa classe posui apenas dois atributos: root, que á a raiz da árvore e, obrigatoriamente está associada ao código (index) 0 e seu caractere é ‘’, ou seja, a string vazia, e size, que guarda a quantidade de nós presentes na árvore. Os métodos dessa classe são create_trie, para inicializar a árvore, match_prefix para a compressão e get_sequence para a descompressão.

Além disso, também há uma função para a impressão de caracteres acentuados (“print_special”)

2.2) main.cpp: Nesse arquivo, há a implementação da leitura e escrita de arquivos, bem como o controle das operações de compressão e descompressão, ao invocar corretamente as funções match_prefix e get_sequence com os dados lidos dos respectivos arquivos de entrada.

O algoritmo para a compressão lê o texto de entrada e, para cada caractere lido, avança na árvore de prefixos, caso o nó pai possua um filho com o tal caractere ou escreve na saída um par de inteiro e caractere, sendo que o inteiro representa uma sequência já lida e o caractere é o que acabou de ser lido.

Para a descompressão, o algoritmo lê um par inteiro e caractere do arquivo e faz um caminhamento na árvore trie para achar o inteiro (um nó com index = número lido) e adicionar o caractere lido como seu filho, bem como atribuir um index a ele, a saber, 1 a mais do que o maior index presente na árvore. Esse valor é controlado com a variável que indica o tamanho da árvore

3. Análise de complexidade

Tal análise deve ser feita em duas partes: uma para a compressão e outra para a descompressão.

3.1) Compressão: Essa operação envolve diversas inserções de nós em uma árvore trie. Para cada símbolo lido no texto, há uma inserção ou caminhamento de um pai para um filho na árvore. Como tais operações são executadas em tempo constante, a complexidade de tempo da compressão é O(n).

3.2) Descompressão: Essa operação envolve caminhamento em árvores, pois para cada código (o inteiro do par inteiro-caractere) do arquivo binário é feito um caminhamento para encontrá-lo na árvore. Como tal caminhamento é feito em O(n), a complexidade de tempo da descompressão é O(nk), onde k é o número de códigos presentes no arquivo binário. Essa complexidade de aproxima de $O(n^2)$ para k grandes (próximos de n).

A complexidade de espaço de ambas operações é O(k), pois é necessário armazenar k nós na árvore trie, além de outras variáveis de tamanho constante. Essa complexidade de aproxima de O(n) para k grandes (próximos de n).

4. Taxa de compressão – exemplos

1) Os Lusíadas: Nesse exemplo, foi utilizado o tipo unsigned short int para a indexação da árvore, pois quere-se analisar o efeito da utilização desse tipo no tamanho do arquivo comprimido. Assim sendo, o arquivo txt tem 337 kb e o arquivo comprimido tem 191 kb, ou seja, 56% do tamanho original.

2) Constituição de 1988: Tamanho do txt → 637 kb / Tamanho do z78 → 432 kb / Taxa de compressão → 68% do tamanho original

3) Dom Casmurro: Tamanho do txt → 401 kb / Tamanho do z78 → 360 kb / Taxa de compressão → 89% do tamanho original

4) The Fellowship of the Ring : Tamanho do txt → 1002 kb / Tamanho do z78 → 767 kb / Taxa de compressão → 76% do tamanho original

5) The Two Towers: Tamanho do txt → 819 kb / Tamanho do z78 → 644 kb / Taxa de compressão → 78% do tamanho original

6) The Return of the King: Tamanho do txt → 710 kb / Tamanho do z78 → 564 kb / Taxa de compressão → 79% do tamanho original

7) The Hobbit: Tamanho do txt → 526 kb / Tamanho do z78 → 444 kb / Taxa de compressão → 84% do tamanho original

8) Especificação desse trabalho prático: Tamanho do txt → 6,4 kb / Tamanho do z78 → 9,7 kb / Taxa de compressão → 150% do tamanho original. Observa-se que, para arquivos pequenos, a compressão utilizando o tipo int para indexar aumenta o tamanho do arquivo. Entretanto, ao utilizar unsigned short int para essa indexação, o tamanho do arquivo comprimido passa a ser 5,8 kb, ou seja 90% do tamanho original.

9) The Philosopher's Stone: Tamanho do txt → 481 kb / Tamanho do z78 → 395 kb / Taxa de compressão → 82% do tamanho original

10) The Chamber of Secrets: Tamanho do txt → 539 kb / Tamanho do z78 → 438 kb / Taxa de compressão → 81% do tamanho original

5. Conclusão

O algoritmo foi executado com sucesso, tanto na compressão como na descompressão. Observa-se que utilizar um tipo que ocupa menos bytes para a indexação (unsigned short int) possui um impacto significativo no tamanho do arquivo comprimido e deve ser utiliazado sempre que o arquivo txt de entrada for pequeno o suficiente, ou seja, sempre que a árvore associada ao texto possuir menos do que 2^{16} nós.