Compactação de Arquivos

Desiree Silva de Araujo[1], **Fernanda Oliveira da Costa**[2]
Orientador: Leando G.M. Alvim[3]

[1]Departamento de Tecnologias e Linguagens –
Universidade Federal Rural do Rio de Janeiro (UFRRJ)
R. Governador Roberto Silveira S/N – Nova Iguaçu –
Rio de Janeiro – RJ – Brasil ,

alvim.lgm@gmail.com, desiree_silva02@hotmail.com, nannda-oliveira@hotmail.com

Abstract. *This project deals with the implementation of a data compression program , specifically for text files using the Huffman coding method, with the aid of data structures seen in class . That is, using this technique can represent a large file to a smaller , reducing the amount of bytes to represent a given, without losses and possible to recover the original file after compression . To demonstrate its operation, they have been some experiments that will be reported later in this report.*

Resumo. *Esse projeto se trata da implementação de um programa de compactação de dados, especificamente para arquivos de texto, utilizando o método de Codificação de Huffman, com o auxílio das estruturas de dados vistas em sala de aula. Ou seja, utilizando essa técnica é possível representar um arquivo grande por outro menor, reduzindo a quantidade de bytes para representar um dado, sendo possível recuperar o arquivo original após a compactação, sem perdas. Para demonstrar o seu funcionamento, foram feitos alguns experimentos, que serão relatados no decorrer deste relatório.*

Sumário

A. Introdução

Com o crescimento da quantidade de dados gerados e transmitidos, a compactação desses se torna cada dia mais essencial. Um método de compactação bem conhecido é o código de Huffman, utilizado neste trabalho, que segundo [David A. Huffman] é um algoritmo que codifica os símbolos, que aparecem com maior frequência, com uma menor quantidade de bits. E, a compressão se divide em dois processos:

- Codificação: a redução do arquivo em si;
- Decodificação: o caminho de volta até o arquivo original.

O objetivo principal deste trabalho é comprimir dados, que se destina também a retirar a redundância, baseando-se que muitos dados contêm informações redundantes que podem ou precisam ser eliminadas de alguma forma. Essa forma é através de uma regra, que, quando seguida, elimina os bits redundantes de informações, de modo a diminuir seu tamanho nos ficheiros.

Existem diversas formas de se criar a árvore de Huffman, porém, utilizamos a abordagem mais comum, que é descrita na seção B. Na seção C, é descrito o objetivo. Para a conclusão deste trabalho foram usadas referências de livros e alguns artigos, que serão descritos na seção G.

B. Problema e Motivação

A codificação de Huffman se divide em dois processos, como descrito acima. Para entendê-los melhor devemos seguir alguns passos:

- Descobrir a frequência de cada caractere, ou seja, o número de vezes que cada caractere aparece no texto;
- No processo de compactação, atribuímos um código binário para cada caractere.

Para fazer essa atribuição do código binário aos caracteres, de tal forma que o texto compactado seja o menor possível, Huffman propôs um algoritmo guloso para atribuição desses códigos, onde não pode ocorrer ambiguidade na descompactação, ou seja, os códigos devem ser livres de prefixo. Onde, dados dois caracteres quaisquer i e j representados pela codificação, a sequencia de bits associada a i nao é um prefixo da sequência associada a j. A vantagem dos códigos livres de prefixo se torna evidente quando vamos decodificar o arquivo comprimido. Para representar de maneira conveniente uma codificação livre de prefixo de modo a facilitar o processo de decodificação, utilizamos uma árvore binária.

Quando criada, ela é percorrida, atribuindo-se os códigos binários a cada caractere que aparece no arquivo. Nesta árvore, os nós folhas possuem os caracteres do arquivo, logo, é necessário que nenhum código seja prefixo de outro e quanto mais alta a frequência de um termo, mais alto ele aparecerá na árvore.

O passo geral do algoritmo é:

- Escolher um par de valores mínimos f e f (do conjunto de frequências;
- Substituir f e f (por f + f (no conjunto));
- Repetir este processo até que um único elemento reste no conjunto;

Para decodificar a mensagem, é criada uma tabela que mapeia cada caractere de entrada em um código definido pela árvore, onde cada linha armazena um caracter e seu respectivo código binário (os bits da codificação dos caracteres). O processo para decodificação funciona da seguinte maneira, para cada símbolo de entrada (bit):

- Se aparecer um bit 0, desce para a esquerda
- Se aparecer um bit 1, desce para a direita
- Quando a folha é atingida a codificação foi encontrada.

Esse processo se repete até o término da árvore. A motivação deste projeto é conseguir representar os dados de uma forma mais simplificada e economizar espaço no disco.

C. Objetivo

Construir um programa que compacte e descompacte arquivos de texto, sem perda de informações, gerando um arquivo binário, onde através dele deverá retornar ao arquivo original sem perdas/alterações dos dados.

D. Proposta

D.1. O algoritmo

O algoritmo de Huffman, muito utilizado para a compressão de dados, possui como ideia básica utilizar a menor codificação possível para representar o caractere mais frequente no arquivo. Tem como entrada, um arquivo de texto e a partir dele faz a contagem de caracteres e constrói a tabela ordenada dos caracteres. Após isso, os dois nós de menor frequência são pegos a cada iteração para formar uma árvore inteira, até que o tamanho da lista seja igual a 1. Depois da árvore formada, ela é percorrida até o nó folha para obter o caractere correspondente a cada símbolo do texto, formando assim uma tabela. Utilizando essa tabela substitui-se cada caractere pelo seu código binário (0 ou 1). Para a decodificação, basta fazer ao contrário, olhar o código binário e descobrir seu caractere no texto.

D.2. Prova de corretude

O algoritmo de Huffman atribui códigos binários aos caracteres de forma ótima, ele determina um prefixo ótimo, sem ambiguidade, utilizando a propriedade de ”escolha gulosa”.

Para provar sua corretude, demonstra-se inicialmente que:

- Um código prefixo ótimo exibe uma escolha gulosa;
- Um código prefixo ótimo exibe a propriedade da subestrutura ótima.

O lema que garante a propriedade da escolha gulosa, é o seguinte:

Segundo, [Cid Carvalho], seja C um alfabeto onde cada caractere tem uma frequência. Sejam x e y dois caracteres em C que tenham as menores frequências. Então existe um código prefixo ótimo para C no qual as palavras-código para x e y tem o mesmo comprimento e diferem apenas no último bit. A prova desse lema mostra que o prefixo ótimo possui a propriedade de escolha gulosa.

Já o lema que garante a propriedade da subestrutura ótima, é o seguinte:

Segundo, [Cid Carvalho], seja C um alfabeto com frequência definida para cada caractere. Sejam x e y dois caracteres de C com as menores frequências. Seja C”o alfabeto obtido

pela remoção de x e y e pela inclusão de um novo caractere z. As frequências dos caracteres em C"interseção C são as mesmas que em C e z, definida como sendo f[z] = f[x] + f[y]. Seja T"uma árvore binária representado um código ótimo livre de prefixo para C". Então a árvore binária T obtida de T"substituindo-se a folha z por um nó interno tendo x e y como filhos, representa um código ótimo livre de prefixo para C. Na prova deste lema, temos a propriedade de subestrutura ótima provada.

D.3. Análise da complexidade

A complexidade da codificação de Huffman, é O (log n), pois é usada uma Heap para armazenar a frequência de cada árvore e cada iteração requer O (log n) para determinar a menor frequência e inserir a nova frequência, e, existem O (n) iterações, uma para cada item. Logo, concluímos que a complexidade do algoritmo de codificação de Huffman é O (n log n).

E. Experimentos

O trabalho foi implementado em linguagem de programação C e foram realizados alguns experimentos com o programa de compactação e descompactação. A tabela abaixo, demonstra qual o tempo de compactação, o tamanho dos arquivos originais e a taxa de compactação.

Nome do arquivo	Tamanho original do arquivo	Tempo em segundos	Taxa de compressão
teste1.txt	220 bytes	0.000093	45,95%
teste3.txt	50 bytes	0.000088	51.82%
teste2.txt	28 bytes	0.000083	49.03%

Figura 1. Informações do arquivo

Para calcular o tempo de compressão do arquivo, foi utilizado uma biblioteca padrão do C, como auxílio, para poder calcular o tempo que o programa levou para comprimir desde o início da sua execução até o fim.

F. Conclusão

O objetivo do trabalho foi de implementar um compactador de arquivos de texto sem perdas de dados na compactação e na descompactação, utilizando a codificação de Huffman.

Obtivemos bons resultados nos testes feitos durante a compactação e descompactação, não houveram perdas de dados e conseguimos uma boa redução em bytes no tamanho dos arquivos. Os resultados foram satisfatórios, como apresentado na figura 1. Para a conclusão dele, obtivemos o auxílio de colegas da turma, professores e livros didáticos, que serão mencionados posteriormente. Através dos experimentos e com auxílio da tabela, nota-se que quanto menor o arquivo, menor é o tempo usado para compactá-lo.

G. Referências

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. [*Algoritmos: Teoria e Prática.*] 3ª Ed., Elsevier, 2012.

Szwarcfiter, J.L., Markenzon, L. [*Estruturas de Dados e Seus Algoritmos.*] 2ª Ed., LTC, Rio de Janeiro, 2004.

Arrigoni, L. [*Compactação sem perda pelo método de Huffman associado à transformada de Wavelet.*] Universidade Vila Velha, 2012.

Cid Carvalho de Souza, Cândida Nunes da Silva, Orlando Lee. [Algoritmos Gulosos] https://www.passeidireto.com/arquivo/2739509/algoritmos-gulosos/3