Atividade Acadêmica de Estrutura de Dados I: Usando a Codificação de Huffman para compactação de arquivos

Lívia de Azevedo da Silva[1], **William Anderson de Brito Gomes**[2]
Orientador: Leandro G. M. Alvim[4]

[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, livia_de_azevedo@yahoo.com.br, wabgomes.ufrrj@gmail.com

Abstract. *In this paper, we will use the Huffman algorithm for the Huffman compression in text files. This is a problem with how we can reduce the memory file with preserve the original text and to resolve this problem we have used one of the most algorithms that community use: Huffman algorithm. It consists that we should construct a binary tree to obtain the bits sequencies for the characters of text. After, we use the binary tree to unzip the compressed file. We have done experiments with our tests that features various cases of texts and we have exposed their results.*

Resumo. *Neste trabalho, irá se usar o algoritmo de Huffman para a compactação de arquivos, que é um problema tipicamente de como se pode reduzir o tamanho de um arquivo sem alterar seu conteúdo original e para resolvê-lo foi usado um dos algoritmos mais recomendados, o algoritmo de Huffman, que consite em definir uma árvore binária para obter as sequências de bits para os caracteres do texto para depois ser possível a descompactação. Foram feitos experimentos com testes que englobam os diversos casos de textos existentes e foram expostos seus resultados.*

1. Introdução

Há um problema existente na computação: a compactação de dados, a qual objetiva diminuir o espaço de memória que um determinado arquivo ocupa. Para resolver este problema, existe o algoritmo de Huffman que realiza esta tarefa de forma otimizada e com ele este trabalho irá se basear. Ao final, tem-se um algoritmo de compactação feito com base no algoritmo de Huffman que apresentou resultados satisfatórios e esperados com relação ao objetivo que se tinha.

Para explicar como foi o decorrer deste trabalho, o relatório é dividido 5 seções: Problema e Motivação, que descreve o problema da compactação; Objetivos, que descreve o nosso objetivo com este trabalho; Proposta, que descreve qual a ideia implementada do algoritmo de Huffman e mostra a complexidade da mesma; Experimentos, que descreve como foi feito os testes para validar o algoritmo feito; Conclusões, que descreve resumidamente nossos resultados e conclusões a respeito do trabalho.

2. Problema e Motivação

Suponha que se deseje armazenar um arquivo de grande porte em algum tipo de memória, primária ou secundária. Para melhor utilizar os recursos disponíveis, deseja-se também minimizar, de alguma forma e na medida do possível, o espaço de memória utilizado. Uma forma de tentar resolver esse problema consiste em codificar o conteúdo do arquivo de maneira apropriada. Se o arquivo codificado for menor do que o original, pode-se armazenar a versão codificada em vez do arquivo propriamente dito. Isto representaria um ganho de memória. Naturalmente, uma tabela de códigos seria também armazenada, para permitir a decodificação do arquivo. Essa tabela seria utilizada pelos algoritmos de codificação e decodificação, os quais cumpririam a tarefa de realizar tais operações de forma automática. Este problema é conhecido como compactação de dados [Jayme Luiz e Lilian 1994].

A este problema, há diversos tipos de soluções possíveis, entre elas a que terá enfoque neste trabalho, a Codificação de Huffman. Ela é uma das soluções mais usadas para compactação de dados. Por isso, é interessante tratar este tipo de solução existente.

3. Objetivo

O objetivo deste trabalho foi apresentar uma forma de implementação em linguagem C da Codificação de Huffman, que é uma das soluções existentes para este problema, além de demonstrar como se pode manipular os dados para obter as informações necessárias para a realização da compressão.

4. Proposta

4.1. O algoritmo de Huffman

De acordo com [Jayme Luiz e Lilian 1994], se descreve como é o algoritmo de Huffman, que resumidamente será explicado a seguir.

Considere um texto a ser codificado. Este texto possui caracteres que futuramente serão substituídos por uma determina sequência de bits referente àquele caracter. Cada caracter terá sua frequência, que consta da quantidade de vezes que ele aparece no texto. Todas as frequências de todos os caracteres serão colocados cada um em um nó e estes nós colocados em uma lista ordenada de forma crescente. Os dois primeiros elementos dessa lista terão suas frequências somadas e colocada em um novo nó que receberá o primeiro elemento da lista como filho esquerdo e o segundo como filho direito. Este nó será adicionado a lista e os outros dois retirados. Isto é feito até que a lista tenha apenas um único nó: que terá a frequência total dos caracteres(total de caracteres presentes no texto). No final, teremos uma árvore com raiz sendo este único nó restante da lista ordenada, sendo chamada de *árvore de prefixos*. Ela irá ter todas as informações a respeito das sequência de cada caracter. Para determinarmos uma sequência, temos que, a partir da raiz da árvore, ir para esquerda, que representa o valor 0, ou direita, que representa o valor 1; caso encontramos um nó folha, ou seja, um nó que tenha um caracter, utilizaremos a sequência que obtivemos no caminho para representá-lo. Obtendo todas essas informações, montamos a tabela de codificação e realizamos a codificação do texto em si.

Na decodificação será necessário reconstruir a árvore original para reconstituir o texto original. Nesse caso, deveremos ler bit a bit da sequência codificada enquanto ”andamos”na árvore(ir para esquerda ou direita) de acordo com o bit lido. Quando se chega

a um nó folha, o caracter contido no mesmo irá ser escrito no arquivo descompactado e voltamos a nossa busca da árvore ao início, que corresponde a raiz da árvore de Huffman. Esse processo é realizado até que todos os caracteres do texto sejam lidos.

Esta é ideia geral do algoritmo de Huffman. Este trabalho seguiu esta ideia.

A ideia implementada foi ler o texto original e armazenar a quantidade total de caracteres do texto, os caracteres usados no texto e a frequência de cada caracter. A partir disto, constrói-se a árvore de Huffman e obtém as sequências através da busca por pós-ordem na árvore de Huffman. Para armazenar as sequências de bits dos caracteres, foi usado variáveis do tipo *unsigned char* denominadas *buffers*(que possui 8 bits), formando um vetor de *unsigned char* caso a quantidade de bits da sequência fosse maior que 8 bits e, para tratar do caso de bits adicionais(se por exemplo um sequência possuir apenas 3 bits, teríamos que escrever estes 3 bits no buffer de 8 bits, pois o mínimo de armazenamento é de 8 bits), salvamos também a quantidade de bits válidos daquela sequência salva, que irá dizer quantos bits aquela sequência realmente possui. Tendo estas informações, é salvo, primeiramente no arquivo codificado, três informações: total de caracteres do texto, quantidade de tipos de caracteres usados e o quantos buffers necessários para armazenar a maior sequência binária da tabela e depois é salvo a tabela de prefixos, que consiste no caracter, sua sequência e a quantidade de bits válidos da sequência e logo depois e salva o texto codificado. Para decodificar, lemos as três primeiras informações já descritas e a tabela de prefixos; com isto disponível, é feita a reconstrução da árvore de Huffman e a decodificação da mensagem usando a árvore. Ao final, ter-se-á o texto original decodificado salvo e disponível.

4.2. Análise da Complexidade

O algoritmo implementado varia sua complexidade baseada em dois fatores, o número de caracteres do texto e a quantidade de caracteres que nele existem e algumas constantes, entretanto o algoritmo se mostra, surpreendentemente linear em sua grande parte, mas em partes onde é necessário percorrer a arvore toda em algumas buscas (por exemplo, o valor da altura, ou então a definição das sequencias de bit de cada nó folha) ele toma passos de $n * \log_2 n$ devido à necessidade de se percorrer a árvore toda.

Logo no pior dos casos o algoritmo é O(m) ou O($n \log_2 n$), onde m é a quantidade de caracteres escritos no arquivo, e 'n' são quantos tipos de caracteres diferentes existem no arquivo. Essa variação ocorre para caso o arquivo tenha muitos caracteres repetidos (onde $m >= n \log_2 n$), neste caso a complexidade do algoritmo seria O(m), caso contrário, com poucos caracteres repetidos, a complexidade seria O($n \log_2 n$).

É um algoritmo eficiente mesmo para instâncias muito grandes, embora seja gasto um bom espaço de memoria em sua execução, devido o grande número de variáveis de controle necessárias para o processamento da informação, em instâncias relativamente grandes o algoritmo se sai muito bem com um tempo linear, o que é excelente.

5. Experimentos

A metodologia usada para os testes foi a de usar instâncias que puderem nos trazer algum tipo de comparação: testes com características diferentes entre si. O objetivo é analisar se realmente conseguimos alcançar o principal objetivo da compressão: um arquivo compactado com tamanho menor.

Alguns conjuntos de dados foram obtidos independentemente pelos criadores desse trabalho e outros obtidos de textos aleatórios da internet. Os denominados ”Teste 1”, “Teste 2”, “Teste 4” e “Teste 5” foram obtidos independentemente, já o “Teste 4” foi retirado da internet.

A seguir, irá se apresentar os textos contidos em cada teste:

- Teste 1: Todos os caracteres são diferentes entre si e cada um só esta presente uma única vez.
  Mensagem:

  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%”&*()_-+={}[]—\/?çá><:;˜ˆ

- Teste 2: Repedida 714 vezes e cada uma em uma linha diferente. Um arquivo de texto grande, mas muito repetido.
  Mensagem:

  Eu estou pronto para ser codificado e depois ser decodificado. Por favor, faça-me este favor!

- Teste 3: Texto retirado da *Wikipédia* no artigo “Codificação de Huffman”. Texto de dissertação normal que simularia um texto comum do cotidiano.
  Mensagem:

  A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. Ele foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutorado no MIT, e foi publicado no artigo ”A Method for the Construction of Minimum-Redundancy Codes”. Uma árvore binária completa, chamada de árvore de Huffman é construída recursivamente a partir da junção dos dois símbolos de menor probabilidade, que são então somados em símbolos auxiliares e estes símbolos auxiliares recolocados no conjunto de símbolos. O processo termina quando todos os símbolos foram unidos em símbolos auxiliares, formando uma árvore binária. A árvore é então percorrida, atribuindo-se valores binários de 1 ou 0 para cada aresta, e os códigos são gerados a partir desse percurso. A codificação de Huffman tem desempenho ótimo quando as probabilidades dos símbolos são potências negativas de dois. A codificação gerada tem também a garantia de ser não ambígua, pois nenhum código pode ser o prefixo de outro código. Para atribuir aos caracteres mais frequentes os códigos binários de menor comprimento, constrói-se uma árvore binária baseada nas probabilidades de ocorrência de cada símbolo. Nesta árvore as folhas representam os símbolos presentes nos dados, associados com suas respectivas probabilidades de ocorrência. Os nós intermediários representam a soma das probabilidades de ocorrência de todos os símbolos presentes em suas ramificações e a raiz representa a soma da probabilidade de todos os símbolos no conjunto

de dados. O processo se inicia pela junção dos dois símbolos de menor probabilidade, que são então unidos em um nó ao qual é atribuída a soma de suas probabilidades. Este novo nó é então tratado como se fosse uma folha da árvore, isto é, um dos símbolos do alfabeto, e comparado com os demais de acordo com sua probabilidade. O processo se repete até que todos os símbolos estejam unidos sob o nó raiz. A cada aresta da árvore é associado um dos dígitos binários (0 ou 1). O código correspondente a cada símbolo é então determinado percorrendo-se a árvore e anotando-se os dígitos das arestas percorridas desde a raiz até a folha que corresponde ao símbolo desejado. Desta forma, o caractere de maior frequência possui o código de menor comprimento que é exatamente a meta da compressão estatística. Assim acontece com os demais caracteres, em ordem proporcionalmente crescente do número de bits, conforme a propriedade: O tamanho médio dos caracteres (TMC) é dado pelo somatório dos produtos do tamanho de cada caracter (T(c)) pela sua probabilidade de limitações (P(c)). (O mesmo resultado pode ser obtido pelo simples somatório das probabilidades de cada nó interno na árvore de Huffman). De posse dos símbolos construídos pelo algoritmo acima, a compressão dos dados é a simples substituição de cada símbolo por seu código correspondente.

- Teste 4: É a união dos textos do “Teste 1”,”Teste 2” e ”Teste 3” juntamente com a seguinte mensagem:

  aojdapmfceiofjiewongijuepnrgjekmçklm,vpoasmvkmweoaqnhfvONSAC
  PAM[EJGP[A] kwoewof,wmpomviwejfiowmpcld,mvs,vopamoejfirghuthgipmvv[7a
  aofmspmslmeopmpwmfsfkoekfoekofçx,cvmijfwiohf238y758023yijrolkfeld
  ncwp sofnksv,mksdj,vklsdm,s,fklopiuytçopiuytrçpoiuytredfcvgbhjklçopityghjklç
  []jkhabcansmc achaoc29564613186e,fop,few,w,ldç,vksnvwiefgiwj0-
  23u832y572372yu9wdiojnsjvnsçvmsdçlvm

  afmskpnfapogmkermgçlrg,moefkg,el,g.fdç,georkpew,
  çsvsaçlwjopegpemgódmgjeiorhu9abijefdskml,çvkmnjaiuojklmajseuijgoakesl

  Esta união repetida duas vezes no arquivo.

- Teste 5: Teste de tamanho bem pequeno.
  Mensagem:

  abc

Os resultados foram exatamente os esperados. Como irá mostrar a tabela, a conclusão que se chega já é a mesma que muitos que trabalham com este algoritmo já saibam: para textos pequenos, não vale a pena aplicar o algoritmo, pois o tamanho do arquivo compactado é maior, devido ao fato que devemos salvar o adicional da tabela de codificação

para decodificarmos depois a mensagem. Para os testes que possuem um tamanho consideravelmente grande, o aproveitamento de memória chega a quase metade e, tendo isso, vale a pena usarmos o arquivo codificado. Além disso, para o "Teste 1" que demonstra nenhuma repetição dos caracteres no texto, foi o que apresentou a menor taxa de compressão, o que é esperado: a intenção do algoritmo é compactar os dados de acordo com os caracteres que possuem repetições no texto e, com isso, tentar diminuir o tamanho se beneficiando com isto. A taxa de compressão foi calculado da seguinte forma:

Taxa de compressão = 1 – (bits usados para compactar o texto) / (bits usados para salvar o texto original)

A seguir, temos a seguinte tabela que mostra os resultados obtidos:

Testes	Tamanho arquivo	Tamanho compactado	% de compressão	% economia de memória
1	82 bytes	319 bytes	19.58	-289.02
2	67117 bytes	34352 bytes	48,94	48,73
3	2991 bytes	1906 bytes	44,82	36,67
4	143316 bytes	75896 bytes	47,37	47,03
5	4 bytes	20 bytes	75,00	-400,00

Tabela 1. Tabela de resultados dos testes

6. Conclusões

O problema da compactação de dados é algo decorrente da computação que merece ser levado mais para frente na questão de se trabalhar soluções mais inteligentes e eficientes que possam resolver o problema. A questão de se economizar memória é algo muito importante, pois, quanto mais memória se ter, mais otimizações computacionais poderão ser efetivadas e o desperdício minimizado.

Em suma, o problema foi atacado com o intuito de proporcionar uma forma de implementar ao algoritmo de Huffman e apresentar resultados que possam ser interessantes de se ter. O trabalho foi feito de forma a tentar economizar o máximo possível de informações na parte da compactação, que na realidade é o maior desafio. Além disso, a forma de como foi tratada a manipulação das sequências binárias(usando buffers de unsigned char) foi uma das estratégias mais interessantes de se trabalhar e muito eficiente.

Os resultados obtidos foram aquelas já esperados pelos criados desse trabalho, demonstrando que este trabalho pode ser usado como uma representação de implementação do algoritmo de Huffman. A análise dos resultados mostrou a economia de memória para os teste de tamanho grande e desperdícios para os teste de tamanho pequeno.

Espera-se que este trabalho sirva como um exemplo de como este problema da compactação e o algoritmo de Huffman possam ser tratados. De maneira inteligente e eficiente, foi implementado um programa que possam compactar arquivos de textos e descompactá-los em sua forma original.

Possivelmente, se existir alguma ideia, melhorias neste trabalho serão feitos. Otimizar os meios já existentes é um dos passos que se tem para se ter a evolução dos métodos.

Referências

Jayme Luiz, S. e Lilian, M. (1994). *Estrutura de Dados e Seus Algoritmos*. Editora, primeira edição.