Análise de complexidade de um algoritmo para visualização da árvore de frequência da codificação de Huffman

Allan Felipe Assis Moreira[1]

[1]Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – Partenon – Porto Alegre – RS – Brasil

Resumo. *A codificação de Huffman é um dos primeiros algoritmos para compressão de dados. A codificação utiliza códigos de diferentes comprimentos, atribuindo códigos mais curtos para caracteres de maior frequência. Para calcular a frequência de caracteres é utilizada uma árvore binária. O objetivo deste trabalho é determinar a complexidade de um algoritmo que permite a visualização da árvore utilizada na compressão. A conclusão é de que a complexidade depende da quantidade de caracteres diferente com frequências diferentes na palavra a ser comprimida e pode ser descrita como O(n).*

1. Introdução

A criação do algoritmo de Huffman é atribuída por [Wayner 2000] à David Huffman em um artigo de 1952 [Huffman 1952]. A área do algoritmo é classificada como compressão de dados, por método estatístico. A codificação utiliza códigos de diferentes comprimentos, atribuindo códigos mais curtos para caracteres de maior frequência. Desta forma, é gerada uma mensagem mais curta que a mensagem original, sem perda de informação. Para calcular a frequência de caracteres é utilizada uma árvore binária.

O objetivo deste trabalho é determinar a complexidade de um algoritmo que permite a visualização da árvore utilizada na compressão. A tarefa proposta apresenta três etapas: (a) localizar ou desenvolver uma implementação do algoritmo de Huffman, (b) exportar a árvore utilizada pela implementação em um arquivo texto no formato utilizado pelo sistema graphviz e (c) determinar a complexidade do algoritmo utilizado na exportação.

A conclusão é de que a complexidade depende da quantidade de caracteres diferente com frequências diferentes na palavra a ser comprimida e pode ser descrita como O(n).

2. Metodologia

A metodologia aplicada para o desenvolvimento do algoritmo foi a da criação a partir do zero, sem a utilização de algoritmos como base ou outros tipos de consultas. Decidiu-se, então, partir do interesse pelo desafio proposto assim como por ter-se chegado à conclusão de que a complexidade do problema era grande, porém não tão grande que o impedisse de ser resolvido por completo.

3. Implementação da codificação de Huffman

A implementação foi desenvolvida na linguagem de programação Java. Para a codificação da String, foi utilizado uma tabela de hashing que possui todos os caracteres com as suas

respectivas frequências de repetição. Cada par (caractere/frequência) foi colocado em uma árvore com somente a raiz, e estas foram inseridas em um vetor. A cada nova árvore inserida no vetor, utilizou-se um método de ordenção por inserção para garantir de que o vetor sempre estaria em ordem de frequência do caractere.

As duas árvores com menores frequências eram então removidas do vetor e unidas gerando uma nova árvore com as raízes e a soma das frequências dos nodos da árvore e esta nova árvore era então adicionada ao vetor. Depois de adicionada, o método de ordenção por inserção era então realizado. Este processo de validação das duas árvores com menor frequência se repete até se chegar a somente uma árvore dentro do vetor, que é a árvore final esperada.

A arvore final é então percorrida em pré-ordem. Cada vez que percorre o próximo nodo filho, adiciona-se 0 ao código do caracter se for pra esquerda do pai, e 1 à direita. Quando chegar na folha, esse código é salvo em uma tabela de hashing com o seu respectivo caractere. A cada retorno da recursão o último bit do código gerado é removido para poder prosseguir.

No final, a cada caractere da String é consultado na tabela de códigos o código referente a mesma, gerando um vetor de inteiros como representação da codificação final. Esse vetor é então transformado em um vetor de bytes, que é salvo no formato de um arquivo binário.

Para a decodificação, lê-se então o arquivo binário codificado obtendo-se um vetor de bytes. Esse vetor é então transformado em um vetor de inteiros que é decodificado a partir da tabela de código/frequência, validando bit a bit até ser encontrado uma codificação existente, e assim sucessivamente até a formação da String original.

4. Exportação no formato graphviz

Para a exportação no formato graphviz, foi utilizado o algoritmo para percorrer a árvore em largura (Breadth First Traversal), montando a árvore nível a nível. Após percorrê-la, os dados obtidos são salvos em um arquivo.

4.1. Algoritmo

O algoritmo utilizado na exportação é como segue:

Algoritmo 1: Algoritmo de percurso em largura para exportação no formato Graphviz

início

1 $fila \leftarrow$ cria fila vazia;
2 $fila \leftarrow$ adiciona $raiz$ na fila;
3 $sb \leftarrow$ "digraph G {";
4 $nAtual \leftarrow$ inicializa variável vazia;
enquanto tamanho $fila \neq \emptyset$ **faça**
5 $nó \leftarrow$ elemento removido da $fila$;
6 $nAtual \leftarrow$ freq do $nó$;
se filho esquerdo do $nó \neq NULL$ **então**
7 $sb \leftarrow sb$ + $nAtual$ + char e freq do filho esquerdo do $nó$;
8 $fila \leftarrow$ adiciona filho esquerdo de $nó$ na fila;
fim
se filho direito do $nó \neq NULL$ **então**
9 $sb \leftarrow sb$ + $nAtual$ + char e freq do filho direito do $nó$;
10 $fila \leftarrow$ adiciona filho direito de $nó$ na fila;
fim
fim
11 $arquivo \leftarrow$ escreve sb em arquivo;
fim

4.2. Exemplos

Foi realizado um teste fazendo uso da String "aaaaabbbbcccdde", conforme a imagem (Figura 1).

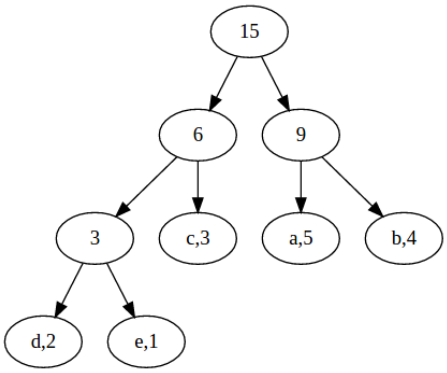

Figura 1. Árvore gerada pelo Graphviz com a String "aaaaabbbbcccdde"

4.3. Análise

O algortimo de geração do arquivo do graphviz teve comportamento linear, percorrendo a árvore somente uma vez, em largura. Com isso, sua complexidade será, no pior caso, O(n).

O algoritmo de Huffman também teve comportamento linear, sendo que, para a codificação, a sua complexidade é, no melhor caso, O(n), e no pior caso, O(n) também. Pois mesmo que se percorra K vezes outros vetores de tamanho N, e aumente a quantidade de caracteres diferentes com frequências diferentes, esses mesmos K vetores serão percorridos K vezes, e continuarão tendo tamanho N.

5. Conclusão

Uma dificuldade encontrada foi de que o algoritmo de huffman implementado só consegue fazer a codificação de String's que tenham mais de um tipo de carcatere.

O trabalho se mostrou desafiador, assim como uma excelente oportunidade para aprofundamento do conhecimento da API da linguagem de programação Java, principalmente no uso de leitura e escrita de arquivos em bytes, a conversão de bytes em inteiros e vice-versa, de Hashings, de criação, escrita e leitura de arquivos entre outros. O domínio sobre como percorrer uma árvore de diferentes maneiras, de ordenção de vetores também se mostraram importantes.

Por fim, pode-se dizer que houve ainda mais interesse do que játinha em algoritmos e estruturas de dados e assuntos correlatos.

Referências

Huffman, D. A. (1952). A method for the construction of minimum reduncy codes. In *Proceedings of IRE*.

Wayner, P. (2000). *Compression Algorithms for Real Programmers*. Morgan Kaufmann.

Código-fonte do algoritmo de Huffman implementado, Allan Moreira, *https://github.com/allanmoreira/*