Universidade Federal do Estado do Rio de Janeiro
Centro de Ciências Exatas e Tecnologia
Escola de Informática Aplicada

Algoritmo de Huffman

Beatriz Queiroz e Maria Jordana Cavalcante

Rio de Janeiro, RJ – Brasil
agosto de 2022

Índice

1 Introdução

Devido à quantidade exorbitante de informações guardadas e compartilhadas em basicamente todo e qualquer setor da computação atual, é necessário desenvolver e aplicar algoritmos que façam uma compressão desses dados em prol da otimização do tempo e do espaço.

A compressão de dados é um processo que reduz o tamanho do arquivo em alguns bits podendo ou não haver perda de informações [5]. Considerando que o melhor cenário seria uma compressão sem perda de dados, o algoritmo de Huffman é uma opção eficiente e simples de implementar e que pode ser aplicado a uma grande diversidade de tipos de arquivo.

O algoritmo de Huffman é uma codificação livre de prefixo e de tamanho variável que leva em consideração o símbolo de maior frequência e atribui a este símbolo um único bit, aos símbolos menos comuns são atribuídos mais bits [2]. Essa codificação é feita utilizando a estrutura de dados de Heap Mínima, e pode ser representada por uma árvore binária cheia, onde os símbolos são os nós folha e cada palavra-código é gerada percorrendo um caminho da raiz até uma folha atribuindo “0” para a esquerda e “1” para a direita [2].

Sendo assim, é aparente que o algoritmo de Huffman é de grande importância na compressão dos dados e sua aplicação poderá ser observada no código escrito pela nossa dupla.

2 Métodos

Classe Arvbin

public void mostraCodigo(Deque<Character>cod)

Método recursivo que imprime os códigos de Huffman associados a cada símbolo.
Ao receber um objeto do tipo Deque<Character>, o método vai percorrer a árvore, inserindo um "0" no deque quando encontrar um nó esquerdo e um "1" no deque quando encontrar um nó direito. Quando ele chega a um nó folha, ele chama o método auxiliar imprimeCod que imprime o símbolo correspondente e o conteúdo do deque. O deque é zerado no retorno à raiz e todo o processo é feito novamente até encontrar todos os nós folha e printar seus respectivos símbolos e códigos binários. [1]

private void imprimeCod(char simbolo, Deque<Character> cod)

Método auxiliar do método mostraCodigo(Deque<Character>cod) que recebe uma variável do tipo char e um objeto do tipo Deque<Character> e imprime ambos seguindo um modelo.

public char getSimbolo()

Retorna a variável símbolo.

public int getFrequencia()

Retorna a variável frequência.

Classe MinBinaryHeap

public void carregaDados()

Método que gera randomicamente as entradas necessárias para testar o algoritmo. Em um loop que roda *n* vezes, sendo *n* o número de símbolos determinado, ele atribui um símbolo aleatório, de uma série de símbolos pré-definidos, à variável símbolo, e um número até 100 à variável frequência, determinando a frequência do símbolo. Essa dupla é instanciada como

um objeto nó folha da classe Arvbin e o objeto é inserido na posição atual de um vetor do tipo Arvbin. A posição é incrementada e o contador do loop é incrementado. Após o fim do loop é chamado o método constroiHeap, que vai organizar o vetor preenchido com os nós folha para que a propriedade de ordem heap seja respeitada e o vetor seja então considerado uma heap.

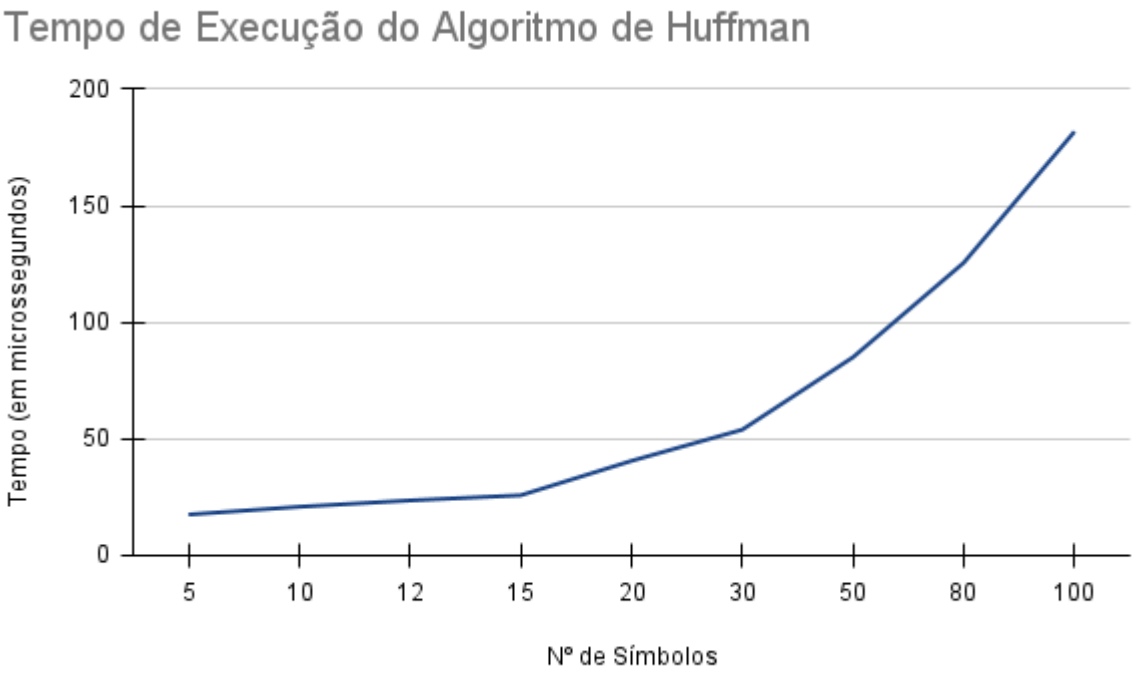
public void aplicaHuffman()

Método de implementação do algoritmo de Huffman que remove os dois menores valores da heap e instancia uma árvore onde o nó pai é a soma dos valores removidos, o nó filho esquerdo é o primeiro valor removido e o nó filho direito é o segundo valor removido. Esse nó produto da soma é chamado nó interno e logo é inserido na heap (o método insere chama o método que ordena o vetor de acordo com a propriedade heap). Esse processo se encontra dentro de um loop que irá funcionar até restar apenas um único nó dentro da heap. [1][3]
Complexidade: while O(n - 1) + 2*removeMin O(log n) + insere O(log n) = O(n log n)

public void mostraCodigos()

Método auxiliar que instancia um objeto do tipo Deque<Character> e chama o método mostraCodigo(Deque<Character>cod), passando esse objeto por parâmetro.

3 Gráfico



O tempo de execução do algoritmo de Huffman foi calculado utilizando o método System.nanoTime() [4]. Antes da execução do algoritmo, uma variável recebia o tempo inicial retornado pelo método e, após a execução do algoritmo, era calculado a diferença entre o tempo final e o tempo inicial. Como o computador não consegue retornar o horário em nanossegundos com muita precisão, há grandes variações no tempo de execução de um método que é executado várias vezes. Sendo assim, a abordagem escolhida para lidar com tais variações foi executar 10 vezes cada entrada e tirar a média dos tempos retornados. Para melhor visualização do gráfico, os resultados foram convertidos de nanossegundos para microssegundos.

4 Referências Bibliográficas

[1] BARNWAL, Aashish. Huffman Coding | Greedy Algo-3. Geeks for Geeks, 2022. Disponível em: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/. Acesso em: 05 de Agosto de 2022.

[2] DASGUPTA, Sanjoy; PAPADIMITRIOU, Christos; VAZIRANI, Umesh. Algoritmos. São Paulo: McGraw-Hill Education, 2009.

[3] GeeksforGeeks. Huffman Coding | GeeksforGeeks. Youtube, 22 de maio de 2017. Disponível em: https://www.youtube.com/watch?v=0kNXhFIEd_w. Acesso em: 05 de Agosto de 2022.

[4] Class System. Oracle, 2020. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html. Acesso em: 08 de Agosto de 2022.

[5] PUJAR, Jagadish; KADLASKAR, Lohit. A New Lossless Method Of Image Compression And Decompression Using Huffman Coding Techniques. Journal of Theoretical and Applied Information Technology. 15 (1): 18–23, 2010.