BST and Heap: Huffman coding and decoding

Felipe Rubens de Sousa Borges, *Universidade Federal de Roraima, Boa Vista, RR, Brasil*

Wanderson Morais de Sousa, *Universidade Federal de Roraima, Boa Vista, RR, Brasil*

Gilberto Alexsandro Almeida Pessoa, *Universidade Federal de Roraima, Boa Vista, RR, Brasil*

Abstract*— This report explores the concepts of binary search trees (BST) and heaps in the context of Huffman encoding and decoding. Huffman encoding is an efficient data compression technique that uses binary trees to represent symbols based on their frequencies. The Huffman algorithm constructs a binary tree to minimize the cost of encoding, where more frequent symbols receive shorter codes. For the implementation, heaps are used to optimize insertion and extraction operations, ensuring O(n log n) performance. The report discusses in detail the costs involved in building the Huffman tree and the operations performed, highlighting the importance of data structures in the algorithm's efficiency. The analysis addresses the advantages and disadvantages of these approaches, providing insights into their application in data compression scenarios.

A codificação de Huffman é uma técnica de compressão de dados amplamente utilizada devido à sua eficiência em reduzir o tamanho de arquivos sem perda de informações. Seu princípio básico é atribuir códigos binários mais curtos a caracteres de alta frequência e códigos mais longos a caracteres de baixa frequência, resultando em uma representação compacta dos dados. Para realizar isso, uma árvore de Huffman é construída, na qual os nós internos representam combinações de frequências e as folhas representam os caracteres individuais.

Neste programa, o processo começa com a leitura de um arquivo HTML, onde a frequência de cada caractere presente no conteúdo é calculada. Com essas frequências em mãos, o programa constrói uma estrutura de min-heap que facilita a priorização dos caracteres com menor frequência. A partir daí, os nós da heap são combinados para formar uma árvore binária, onde os dois nós de menor frequência são unidos em um nó interno, cuja frequência é a soma dos nós filhos. Esse processo é repetido até que reste apenas um nó, que se torna a raiz da árvore de Huffman.

Após a construção da árvore, o algoritmo percorre a estrutura para gerar os códigos binários de cada caractere. Cada vez que o percurso segue para a esquerda, um 0 é adicionado ao código, e cada vez que segue para a direita, um 1 é adicionado, criando assim uma tabela de códigos única para cada caractere com base em sua frequência.

OBJETIVO

O objetivo principal deste projeto é implementar um sistema de codificação e decodificação de Huffman, utilizando como entrada um arquivo contendo uma página HTML, como um artigo da Wikipedia. Para atingir esse objetivo, o sistema construirá uma árvore binária de Huffman, que será fundamental para a compressão eficiente dos dados. Durante a construção da árvore, utilizaremos uma fila de prioridade para garantir que os nós com as menores frequências sejam selecionados de forma otimizada. Isso não apenas demonstrará a eficiência do algoritmo de Huffman, mas também proporcionará uma compreensão prática de como a estrutura de dados contribui para o desempenho do algoritmo.

Além disso, o projeto se propõe a criar um dicionário de códigos binários, mapeando cada letra a seu respectivo código gerado pela árvore de Huffman. Após a construção do dicionário, o sistema permitirá a conversão de novas frases em seus códigos Huffman correspondentes, seguido

da decodificação dessas sequências de volta para a sentença original. Para garantir a integridade do aprendizado, o projeto requer que as implementações da árvore binária de busca (BST) e do heap sejam realizadas manualmente, sem a utilização de bibliotecas específicas da linguagem de programação. No entanto, bibliotecas externas, como o GraphViz, poderão ser utilizadas para visualização da árvore Huffman, facilitando a compreensão do processo de codificação.

VISUALIZAÇÃO

A visualização dos resultados da implementação se dá através da geração de um arquivo .dot contendo o dicionário de códigos gerados pela árvore de Huffman, então é renderizado no GraphViz, podendo produzir uma representação em imagem PNG com a árvore.

ESTRUTURA DO PROGRAMA

O programa é composto por várias seções principais, cada uma cumprindo uma função específica no processo de codificação Huffman. Abaixo está uma visão geral da estrutura do programa:

Estruturas de Dados:

- MinHeapNode: Um nó na árvore de Huffman, contendo o caractere (data), sua frequência (freq) e ponteiros para os nós filhos esquerdo e direito.
- MinHeap: Uma fila de prioridade (min-heap) usada para construir a árvore de Huffman combinando continuamente os nós com as menores frequências.

Funções Auxiliares:

- newNode(): Aloca memória para um novo nó da árvore com um caractere e frequência específicos.
- createMinHeap(): Inicializa uma min-heap vazia com uma capacidade dada.
- swapMinHeapNode(): Troca dois nós na min-heap para manter a propriedade de heap.
- minHeapify(): Garante que a heap mantenha sua propriedade de heap mínima reorganizando a árvore, se necessário.
- isLeaf(): Verifica se um nó é uma folha (não possui filhos).

Algoritmos Principais:

- buildMinHeap(): Constrói a min-heap ajustando repetidamente a estrutura da heap.
- extractMin(): Remove e retorna o nó com a menor frequência da heap.
- insertMinHeap(): Insere um novo nó na heap, garantindo que a propriedade de heap seja mantida.
- buildHuffmanTree(): O coração do programa, essa função constrói a árvore de Huffman combinando os dois nós com as menores frequências até que reste apenas uma árvore.

Manipulação de Arquivos:

- HuffmanCodesFromHTML(): Lê um arquivo HTML, calcula a frequência de cada caractere e constrói a árvore de Huffman.
- writeDotFile(): Gera a árvore de Huffman no formato DOT, permitindo a visualização com ferramentas externas.

Visualização:

- A estrutura da árvore é exportada como um arquivo DOT, que representa os nós e conexões da árvore de forma estruturada. Esse arquivo foi renderizado usando o Graphviz para produzir uma representação gráfica da árvore de Huffman.

IMPLEMENTAÇÃO E TESTES

Na implementação do código foi necessário a instalação do GraphViz para a renderização da árvore. Inicialmente o procedimento padrão para compilar e executar um arquivo na linguagem C: “gcc -o huffman huffman.c” para compilar e “./huffman” para gerar o arquivo executável, ao mesmo tempo que foi implementado no código que fosse gerado também o arquivo DOT, de mesmo nome.

Foram feitas 5 (cinco) implementações para diferentes páginas HTML que resultaram cinco arquivos DOts contendo o dicionário de códigos para cada caractere da página.

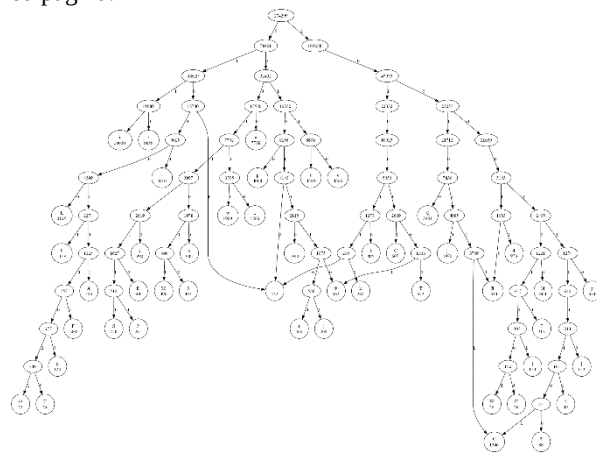

FIGURA 1. Note que essa árvore foi criada a partir da leitura do HTML da página https://pt.wikipedia.org/wiki/Tartarugas_Ninja, como primeiro teste de implementação.

A primeira implementação foi a partir de uma página escolhida de forma aleatória, então para que os testes fossem mais eficientes, foram selecionadas páginas maiores (em quantidades de bytes) para implementação.

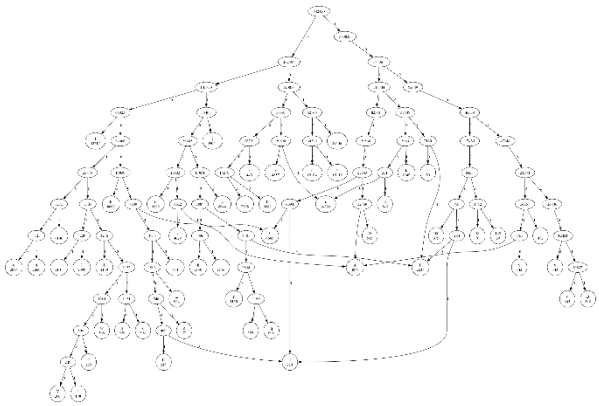

FIGURA 2. Note que essa árvore foi criada a partir da leitura do HTML da página https://pt.wikipedia.org/wiki/Bíblia, identificada como uma página maior para realização de um teste.

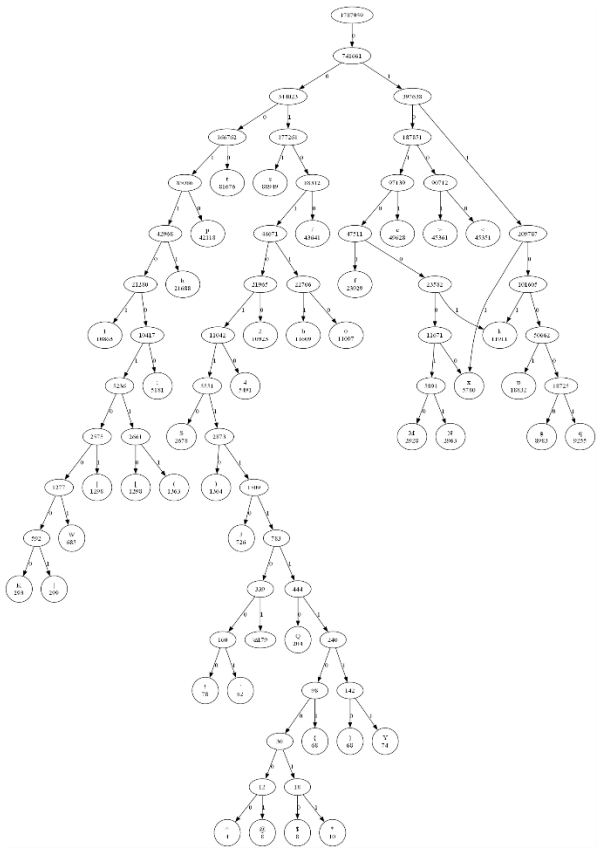

FIGURA 3. Note que essa árvore foi criada a partir da leitura do HTML da página https://pt.wikipedia.org/wiki/História_da_humanidade, identificada como uma página maior para realização de um teste, contendo 612 645 bytes.

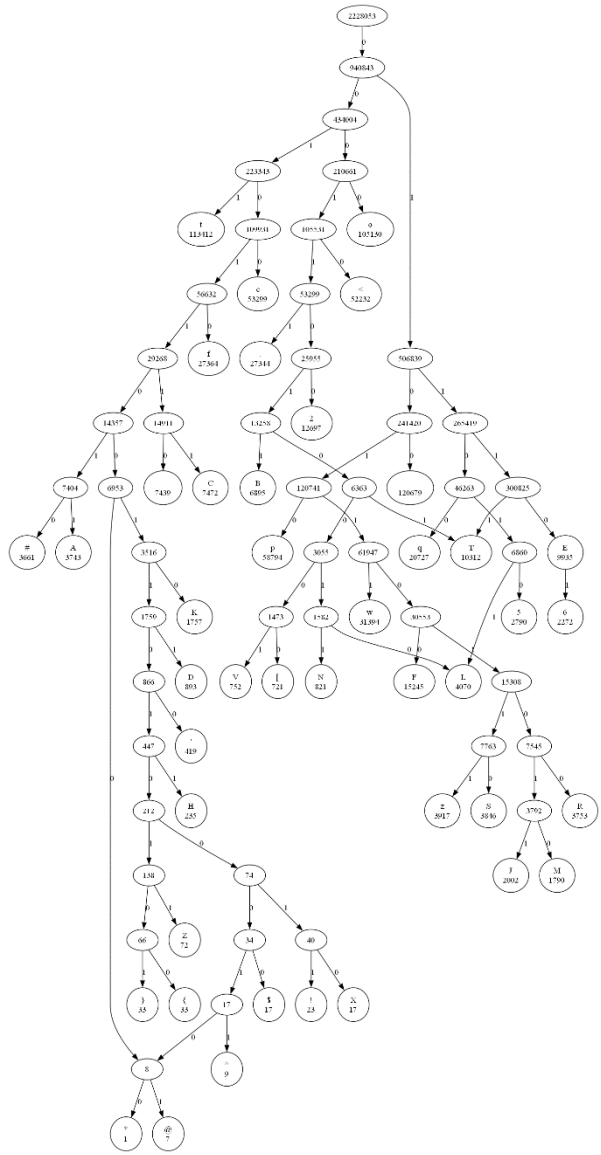

FIGURA 4. Note que essa árvore foi criada a partir da leitura do HTML da página https://pt.wikipedia.org/wiki/Temporada_do_Clube_de_Regatas_do_Flamengo_de_2017, identificada sendo menor que a página anterior, contendo 486 867 bytes.

Apesar do quarto teste utilizando uma página consideravelmente menor que a página do terceiro teste, notou-se que a frequência dos caracteres afetou a execução, justificando um teste notável. Então para a

realização do último teste foi feita uma busca da maior página (em bytes) fornecida pelo site https://pt.wikipedia.org/wiki/, com o idioma português para o Brasil.

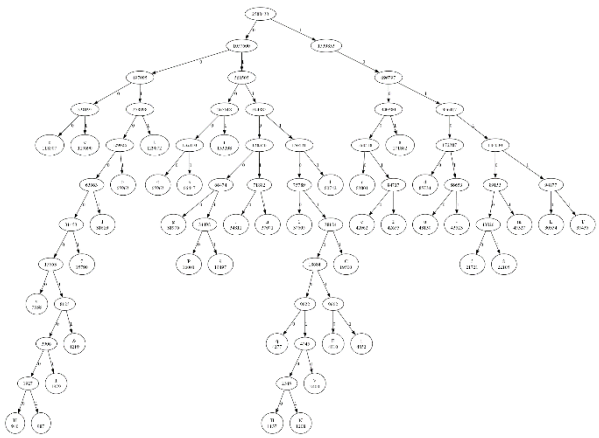

FIGURA 5. Note que essa árvore foi criada a partir da leitura do HTML da página https://pt.wikipedia.org/wiki/Lista_de_bens_tombados_na_cidade_de_São_Paulo, identificada sendo a maior página do site em questão para o idioma Português, contendo.

ANÁLISE DE CUSTO E COMPLEXIDADE

A análise de custo considera diversas variáveis que influenciam a eficiência do algoritmo. Abaixo, apresentamos as principais funções, seguidas pela análise da complexidade.

BST (Binary Search Tree) e heaps são estruturas de dados essenciais em algoritmos como a codificação de Huffman. O BST permite operações de busca, inserção e exclusão em O(log n) em média, enquanto o heap oferece inserção e remoção em O(log n).

A construção da árvore de Huffman utiliza um min-heap, e o custo dessa fase é O(nlogn), onde n é o número de caracteres únicos. A complexidade total do algoritmo é O(nlogn), sendo eficiente para compressão de dados.

A árvore de Huffman é construída utilizando uma fila de prioridade (min-heap) para ordenar os nós baseados nas frequências dos caracteres. Cada nó da árvore contém:

- data: caractere (custo O(1))
- freq: frequência do caractere (custo O(1))
- left e right: nós filhos esquerdo e direito (custo O(1))

A estrutura da fila de prioridade (min-heap) é composta por um array de nós da árvore. A criação, manipulação e reorganização desse min-heap são fundamentais para a construção eficiente da árvore de Huffman.

A função newNode(char data, unsigned freq) cria um novo nó para a árvore de Huffman, com custo constante O(1) para inicializar cada nó. Da mesma forma, a função createMinHeap(unsigned capacity) constrói uma fila de prioridade com complexidade O(d), onde d é a capacidade do min-heap.

Funções de manipulação da fila de prioridade incluem:

- swapMinHeapNode: troca dois nós da fila, com custo O(1).
- minHeapify: ajusta a posição dos nós para manter a propriedade do min-heap. A complexidade desta função é O(log n) devido à reorganização dos elementos no heap.
- insertMinHeap: insere um novo nó no min-heap com complexidade O(log n), conforme a inserção ocorre no último nível do heap e a estrutura é reorganizada.
- buildMinHeap: constrói o min-heap com base nas frequências dos caracteres, com custo O(n log n).

Após uma análise considerando muitas instruções para extrair o custo. Foi encontrado um valor para a função de custo:

$$T(n,m,k,d,a) = \frac{29n}{2}log_2\ n+\frac{29}{2}log_2\ n+\ 88log_2\ n\ +30n+2*m+6k+d+2*a+826$$

A função de custo total para a execução do algoritmo de Huffman é composta por várias etapas, levando em consideração variáveis que influenciam o desempenho:

- n: número de elementos no min-heap
- m: tamanho total do arquivo HTML em bytes
- k: número de caracteres únicos no arquivo
- d: capacidade do min-heap
- a: número de caracteres no arquivo.

Após a verificação dos valores das funções foi possível perceber que algumas delas possuíam o mesmo valor. Então de cinco variáveis passou a ser duas, m e n. A função de custo aproximada pode ser expressa da seguinte forma:

$$T(n,m) = \frac{29n}{2}log_2\ n+\frac{29}{2}log_2\ n+\ 88log_2\ n\ +37n+4*m+826$$

RESULTADOS DOS TESTES

Os resultados dos testes de execução do Heap, BST e criação da árvore de Huffman foram feitos implementando a biblioteca time.h para extrair o tempo de execução, para que fosse gerado um gráfico de comparação de execução para diferentes entradas.

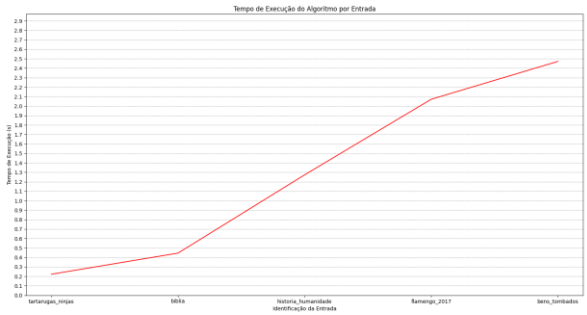

FIGURA 6. Note que esse gráfico possui 5 identificações de entradas no eixo x, sendo respectivamente as páginas HTML da Wikipédia: Tartarugas Ninjas, Bíblia, história da humanidade, Elenco 2017 do Flamengo e, por fim Bens tombados da cidade de São Paulo. Enquanto no eixo y, é identificado o tempo de execução em uma escala de 0,1 segundos.

Os tempos de execução variam significativamente entre as diferentes entradas, indo de 0.223 segundos até 2.472 segundos. Isso sugere que o tamanho ou a complexidade dos textos das páginas HTML influencia diretamente o tempo de construção da árvore de Huffman.

O tempo de execução aumenta de maneira relativamente consistente conforme a complexidade do texto aumenta, mas não segue uma progressão linear. Isso é comum em algoritmos de compressão e construção de árvores, pois a quantidade de caracteres únicos e frequência de ocorrência podem impactar o processo de criação da árvore de Huffman. A análise de tempo de execução revela a eficácia do algoritmo de Huffman na compressão de dados de diferentes tamanhos. Os tempos de execução, embora variem entre as páginas, ainda são bastante rápidos, considerando que o processo de construção da árvore de Huffman pode ser computacionalmente intensivo. Para um uso prático, como compressão de grandes volumes de dados ou de análise de páginas HTML extensas, esses tempos são aceitáveis, sugerindo que o algoritmo pode ser utilizado de maneira eficiente em diversas aplicações.

APLICAÇÃO PRÁTICA

Uma aplicação prática muito comum da codificação Huffman é no processo de compactação de imagens no formato JPEG. O algoritmo de Huffman é usado em uma das etapas de compressão para reduzir o tamanho da imagem sem perda significativa de qualidade.

Como o Huffman é aplicado no JPEG:

Conversão da Imagem: Uma imagem é convertida em diferentes blocos, onde cada bloco representa as cores ou luminosidade.

Transformada Discreta do Cosseno (DCT): O JPEG usa a DCT para transformar os dados de espaço de cores em coeficientes de frequência. Isso separa os detalhes importantes (baixas frequências) dos menos relevantes (altas frequências).

Quantização: O processo de quantização reduz a precisão dos coeficientes, descartando as informações de alta frequência (menos perceptíveis ao olho humano). Nesse ponto, muitos dos coeficientes se tornam zeros ou valores pequenos.

Codificação Huffman: Finalmente, os coeficientes quantizados são codificados usando o algoritmo de Huffman. A ideia é gerar códigos de menor comprimento para os valores que aparecem com mais frequência, e códigos mais longos para valores raros. Isso resulta em uma compactação eficiente da imagem.

CONCLUSÃO

O projeto teve como tema principal a construção e análise de uma árvore de Huffman para compressão de páginas HTML. A escolha das páginas HTML como entrada, incluindo textos de diferentes complexidades e tamanhos, foi fundamental para demonstrar como o algoritmo de Huffman se comporta em cenários variados, avaliando seu desempenho em termos de tempo de execução e eficiência de compressão.

A utilização dos conceitos fundamentais de estrutura de dados, como árvores binárias e heaps, que desempenham papel crucial na organização dos elementos de forma hierárquica e na priorização das frequências dos caracteres.

A função de custo e a complexidade do algoritmo de Huffman foram analisadas durante o projeto. A criação da árvore de Huffman tem uma complexidade de tempo O(n log n), onde *n* é o número de caracteres distintos no texto. Isso ocorre devido à necessidade de construir a fila de prioridade (heap) e realizar operações de extração e inserção de nós na árvore. A análise dos testes mostrou que, embora o tempo de execução aumente com a complexidade do texto, o algoritmo se comporta de maneira previsível, seguindo o comportamento assintótico esperado.

Na prática, isso significa que o algoritmo pode ser aplicado eficientemente em aplicações que envolvem compressão de grandes volumes de dados, como páginas HTML ou arquivos de texto extensos.

REFERÊNCIAS

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Algoritmos: Teoria e Prática, 3rd ed. Rio de Janeiro, Brasil: Elsevier, 2012.

LINK REPOSITÓRIO

https://github.com/F3lipeB0rges/FelipeBorgesGilbertoAlexsandroWandersonMorais_FinalProject_AA_RR_2024