Função de Custo e Complexidade em um Algoritmo de Compressão de Huffman

Neste artigo, discutimos a função de custo e complexidade de um algoritmo para construir uma árvore de Huffman, com base em uma implementação em C. O código busca calcular a frequência dos caracteres em um arquivo de texto, construir a árvore de Huffman, e gerar um arquivo .dot para visualização gráfica. A análise de custo considera diversas variáveis que influenciam a eficiência do algoritmo. Abaixo, apresentamos as principais funções, seguidas pela análise da complexidade.

Estruturas de Dados Utilizadas

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.

Funções Principais e Complexidade

Criação do Nó e Min-Heap

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.

Operações no 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).

Construção da Árvore de Huffman

A árvore de Huffman é construída através da função buildHuffmanTree, que utiliza a fila de prioridade para combinar os nós de menor frequência. A complexidade total dessa operação

é dada por O(n log n), considerando que em cada iteração os nós são extraídos e reinseridos na fila de prioridade, até que reste apenas um nó, a raiz da árvore.

Função de Custo

Após uma análise considerando muitos 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

Simplificação da Função de Custo

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

Teste de Performance

Foram realizados dois testes com diferentes arquivos HTML para medir o tempo de execução do algoritmo. Os valores obtidos e a função de custo aproximada são apresentados abaixo.

Teste 1: pagina_testado.html

- n = 93, m = 25.172

T(93,25172) = $\frac{29*93}{2}log_2$ 93+$\frac{29}{2}log_2$ 93+ 88log_2 93 +37*93+4*25172+826
T(93,25172) = $1.348,5$ *log_2 93+$14,5$ *log_2 93+ 88* log_2 93 +3.441+100.688+826

log_2 93 é aproximadamente 6.5.

T(93,25172) = $1.348,5$ *6,5+$14,5$ *6,5+ 88*6,5+3.441+100.688+826

T(93,25172) = 8.765,25 +94,25+ 572+3.441+100.688+826

T(93,25172) é aproximadamente 114.386 instruções.

Convertendo para tempo de execução:

Regra de Três Simples:
1000 instruções - 333 µs
114.386 instruções - x

x=333*114386/1000≈38.090 µs

convertendo para segundos : 0,03809 s

O tempo medido no primeiro computador foi 0,039 s, e no segundo, 0,009 s, confirmando a estimativa.

Teste 2: benstombados.html

- n = 96, m = 2.637.040

T(96,2637040) = $\frac{29*96}{2} log_2$ 96+$\frac{29}{2} log_2$ 96+ 88log_2 96+ 37*96+4*2637040+826
T(96,2637040) = $1.392 log_2$ 96+$14,5 log_2$ 96+ 88log_2 96+ 37*96+4*2637040+826

log_2 96 é aproximadamente 6,58
1.392 *6,58+$14,5$ *6,58+ 88*6,58+ 37*96+4*2637040+826

T(96,2637040) é aproximadamente 10.562.371 instruções

1000 instruções - 333 µs
10.562.371 instruções - x

x=333*10.562.371/1000≈3.517.269µs

convertendo para segundos : 3,517269 s
tempo de execução computado pelo algoritmo (computador1)≈ 2,4300 s

tempo computador2 = 0.426000s

No primeiro computador, o tempo medido foi aproximadamente 2,43 s. No segundo, o tempo medido foi aproximadamente 0.426000s

.Complexidade:Verificando o valor mair alto na função de custo e fazendo algumas abstrações obtemos a complexidade de O(n*log n).

Considerações Finais

A função de custo desenvolvida para o algoritmo de Huffman mostra que o comportamento do min-heap é o principal fator de complexidade, com a manipulação de n elementos resultando em termos dominantes O(n*log n) . O impacto do tamanho do arquivo e do número de caracteres únicos também é notável, mas menor em comparação ao custo da reorganização do heap.

Os resultados obtidos com base nos testes de tempo mostraram-se próximos das estimativas teóricas, validando a função de custo desenvolvida.