**Universidade do Vale de Itajaí**

**Ciência da Computação**

**Relatório sobre o Algoritmo de Huffman**

**Professor: Marcos Carrard**

**Alunos: Diogo Morgado Viana, Gabriel Schaldach Morgado**

**Data: 15 de maio de 2024**

**Sumário**

1. **Introdução**
2. **Fundamentos Teóricos do Algoritmo de Huffman**
3. **Processo de Codificação de Huffman**
4. **Processo de Descodificação de Huffman**
5. **Complexidade do Algoritmo de Huffman**
6. **Aplicações Práticas e Desafios**
7. **Conclusão**
8. **Referências**

**1. Introdução**

Este relatório apresenta uma análise detalhada do algoritmo de Huffman, uma técnica fundamental de compressão de dados amplamente utilizada na área de ciência da computação. Desenvolvido por David A. Huffman em 1952, o algoritmo de Huffman é uma abordagem eficaz e eficiente para compressão de dados sem perdas, utilizando frequências de ocorrência de símbolos na fonte de dados. Exploraremos os princípios teóricos do algoritmo, seu processo de codificação e descodificação, complexidade e aplicações práticas.

**2. Fundamentos Teóricos do Algoritmo de Huffman**

O algoritmo de Huffman é baseado na ideia de que os símbolos mais comuns em um conjunto de dados devem ser representados por códigos binários mais curtos, enquanto os símbolos menos comuns devem ser representados por códigos mais longos. Isso permite uma representação mais eficiente dos dados, reduzindo o número total de bits necessários para armazenar ou transmitir as informações.

> Assim, a Codificação de Huffman funcionará para qualquer tipo de dado onde haja repetição de símbolos, sendo o principal exemplo de aplicação as “strings”.

**3. Processo de Codificação de Huffman**

O processo de codificação de Huffman envolve várias etapas, incluindo o cálculo da frequência de ocorrência de cada símbolo, a ordenação dos símbolos por frequência e a construção de uma árvore binária de codificação. > Assumiremos que a codificação será feita em cima de uma string que possui somente caracteres da tabela ASCII para melhor exemplificação, mas ela poderia ser também feita para outros tipos de dados que apresentam repetição de símbolos.

A primeira etapa consiste no cálculo da frequência de cada caractere na string, o que gerará uma lista com os caracteres e as suas frequências. Após isso, devemos ordenar essa lista por ordem crescente em relação ao número de frequências, deixando os caracteres com menor frequência nas primeiras posições da lista.

Essa ordenação da lista servirá para criar os nós folha da árvore binária que será montada. Desta forma, cada um dos elementos da lista até este momento, que representam os caracteres que existem na string original e as suas frequências, representarão nós folha desta árvore.

A segunda etapa consistirá em, usando essa lista ordenada, montar a árvore inteira que nos mostrará os códigos binários de cada caractere. Para fazer isso, a partir da lista ordenada, começamos pegando os 2 primeiros elementos da lista, que no começo representarão os 2 caracteres com menor número de ocorrência na string. A partir desses 2 elementos, criaremos um nó na árvore, onde o filho esquerdo será o número com menor frequência entre os 2, e o filho direito será o número com maior frequência. O valor deste novo nó consistirá na soma das 2 frequências de seus filhos, e ele não representa nenhum caractere em específico.

Após criar um nó, retiraremos os 2 nós usados da lista e adicionaremos o novo nó criado na lista, a fim de repetir o processo até que só sobre um nó na lista. Este novo nó adicionado funcionará exatamente da mesma forma que os demais nós, com a diferença de que não representará nenhum caractere, e o processo seguirá exatamente da mesma forma, retirando 2 nós e adicionando 1 nó na lista de cada vez. Ao final, existirá somente um nó.

Com a árvore montada, o nó raiz (último restante da lista) terá o número total de caracteres da string, e sempre seguirá a lógica que o filho da esquerda representa o menor número de frequência, e o filho da direita representa o maior número. Ao percorrer a árvore até chegar em um nó folha, que representa um caractere da string, devemos atribuir 0 para cada vez que o filho está à esquerda, e 1 para cada vez que o filho está à direita. Logo, seguindo o caminho desde o nó raiz, teremos para cada nó folha um caminho representado por zeros e uns, que também representará o código que o caractere assumirá na nova string que será comprimida.

Assim, basta substituir na string original cada caractere por seu respectivo código designado pela árvore, e além da string agora codificada, será necessário também para a decodificação ou a tabela que associa os códigos com os caracteres, ou a própria árvore utilizada. Com ambos em mãos, a string estará completamente codificada e será possível, quando necessário, fazer o processo da decodificação.

**4. Processo de Decodificação de Huffman**

A decodificação de Huffman é o processo inverso da codificação e envolve a utilização da árvore binária de codificação para recuperar os dados originais. O algoritmo percorre a árvore binária bit a bit, navegando da raíz até as folhas correspondentes aos símbolos originais.

A codificação de Huffman produzirá 2 resultados, que serão usados para a parte da decodificação. Um deles é a árvore de Huffman, com todos os nós, e o outro é a própria string codificada, que usa os códigos referentes aos caracteres ao invés dos próprios caracteres.

No centro da decodificação de Huffman está o processo de transformar os códigos binários em caracteres originais. Esses códigos, chamados de códigos de Huffman, são criados com base em quantas vezes cada caractere aparece na informação original. Durante a codificação, uma árvore chamada de árvore de Huffman é feita. Esta árvore é importante para a decodificação.

A árvore de Huffman é uma árvore com dois caminhos em cada nó, onde cada nó guarda a soma das frequências dos caracteres que ele contém. As folhas da árvore são os caracteres originais, e os caminhos da raiz até as folhas representam os códigos de Huffman desses caracteres. Durante a decodificação, a árvore é usada para transformar os códigos binários de volta aos caracteres originais.

Para decodificar, começamos na raiz da árvore de Huffman. A cada "0" na sequência compactada, seguimos para o nó filho à esquerda, e a cada "1", para o nó filho à direita. Isso continua até chegarmos a uma folha, quando o caractere correspondente é emitido. Então, o processo recomeça para decodificar o próximo caractere.

Uma parte importante da codificação de Huffman é que os códigos são únicos, garantindo que nenhum código seja parte de outro. Isso faz com que a decodificação seja clara e precisa, sem confusões.

A decodificação de Huffman é usada em muitas áreas, desde compactar arquivos até enviar dados pela internet. Sua habilidade de reduzir o tamanho dos dados sem perder informações importantes a torna essencial em muitas situações.

**5. Complexidade do Algoritmo de Huffman**

A complexidade do algoritmo de Huffman varia dependendo do tamanho do conjunto de dados de entrada. Em geral, a complexidade de tempo do algoritmo de Huffman para codificação é O(n log n), onde n é o número de símbolos no conjunto de dados. No entanto, a descodificação de Huffman tem uma complexidade de tempo de O(log n), onde n é o número de símbolos na árvore de codificação.

**6. Aplicações Práticas e Desafios**

O algoritmo de Huffman é amplamente utilizado em uma variedade de aplicações, incluindo compressão de arquivos de texto, imagens, áudio e vídeo. No entanto, ele apresenta algumas limitações, como a necessidade de modelar probabilidades antes da codificação e a variação no comprimento dos códigos.

**7. Conclusão**

O algoritmo de Huffman continua sendo uma técnica fundamental e amplamente utilizada em compressão de dados, mesmo após mais de seis décadas de seu desenvolvimento. Sua simplicidade e trivialidade o tornam uma escolha popular em uma variedade de contextos. No entanto, é importante considerar suas limitações e explorar abordagens alternativas em situações em que elas possam ser mais adequadas.

Este relatório fornece uma visão abrangente do algoritmo de Huffman, desde seus fundamentos teóricos até suas aplicações práticas e desafios, incluindo uma análise da complexidade do algoritmo.

**8.** **Referências**

**Algoritmo de Huffman**. Disponível em: <http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/algoritmo-de-huffman/>. Acesso em: 15 maio. 2024.

DISCRETO, U. **Como a compressão de dados funciona? (Árvore de Huffman)**. Disponível em: <https://www.youtube.com/watch?v=-TonlL3vcGk>. Acesso em: 15 maio. 2024.

FEOFILOFF, P. **Código de Huffman**. Disponível em: <https://www.ime.usp.br/~pf/analise\_de\_algoritmos/aulas/huffman.html>. Acesso em: 15 maio. 2024.

**Huffman coding**. Disponível em: <https://www.programiz.com/dsa/huffman-coding>. Acesso em: 15 maio. 2024.