Códigos de Huffman:

Outro problema que pode ser resolvido através de uma abordagem gulosa é a codificação através dos Códigos de Huffman. A ideia principal por detrás desse método de codificação é atribuir códigos de representação menores (ou seja, que utilizem menos bits) para caracteres com maior frequência de presença. Considere a seguinte figura:

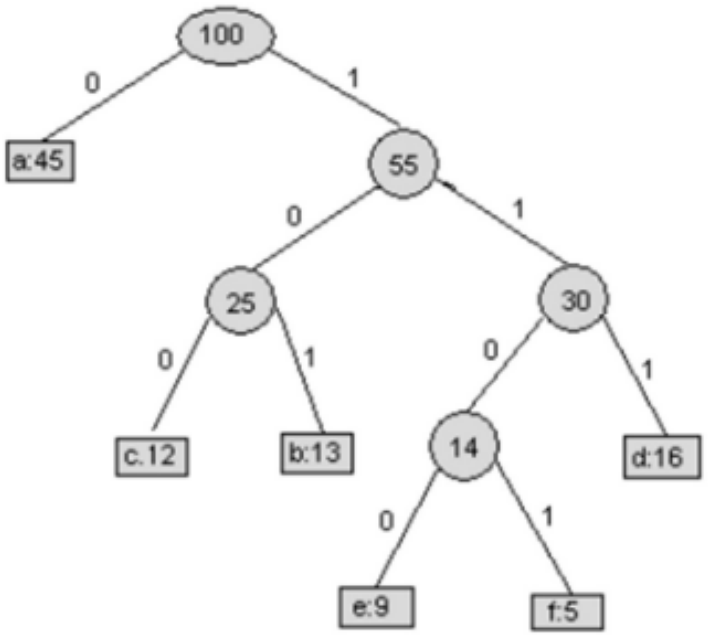
	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5
Palavra de código de comprimento fixo	000	001	010	011	100	101
Palavra de código de tamanho variável	0	101	100	111	1101	1100

Caso optássemos por utilizar o código de comprimento fixo, teríamos um uso de memória igual a (45*3)+(13*3)+(12*3)+(16*3)+(9*3)+(5*3)=300 milhares de bits. Ao utilizar o código de comprimento variável, temos (45*1)+(13*3)+(12*3)+(16*3)+(9*4)+(5*4)=224 milhares de bits. Isso nos dá uma redução de aproximadamente 25% na quantidade de bits utilizada, diminuindo o custo de armazenamento.

Códigos de prefixo:

Um código de caracteres binário é obtido através da concatenação das palavras de código correspondentes aos caracteres usados, ou seja, um trecho abc em código de tamanho variável seria representado por 0+101+100 = 0101100. Utilizar códigos de prefixo simplificam a codificação, uma vez que nenhuma palavra de código é prefixo de outra, evitando comparações desnecessárias e possíveis confusões. Com isso, pode-se simplesmente identificar a palavra de código inicial, traduzi-la, removê-la do arquivo e repetir o processo até que se tenha decodificado todo o arquivo.

Entretanto, para tal é necessário um esquema de codificação conveniente, o qual pode ser obtido através de árvores binárias, onde as folhas representam os caracteres a serem codificados.



A figura acima representa a árvore de codificação de tamanho variável. Uma árvore binária cheia é a forma ótima de se representar códigos de prefixo por não deixar lacunas no código (como combinações não utilizadas de binário, o que complicaria a decodificação caso surgissem). Desse modo, nota-se que para atingir determinado caractere, basta seguir pelos filhos (caminho para a esquerda representa o binário 0 e para a direita, o binário 1). Assim, para o caractere ‘a’ temos ‘0’, para o caractere ‘c’ temos ‘100’, e assim segue.

Considerando C como o número de caracteres a serem codificados (no caso atual, 6), a árvore de codificação terá C folhas e C-1 nós internos. O tamanho de cada palavra de código pode ser determinado pela profundidade da folha em questão, uma vez que cada nível que se leva para chegar até a folha desejada representa um binário que será adicionado à palavra código. Como exemplo, pode-se observar o caractere ‘e’. Este está localizado no nível 4 da árvore, e seu código é ‘1100’, tendo exatamente 4 bits.

Sendo T a árvore binária correspondente à codificação desejada, f(c) a frequência do caractere c e d(c) a profundidade da folha correspondente ao caractere c, o custo de armazenamento da árvore será de:

$$B(T) = \sum_{c \in C} \mathrm{f(c)d(c)}$$

No caso da árvore apresentada na figura acima, seu custo é de 224 bits.

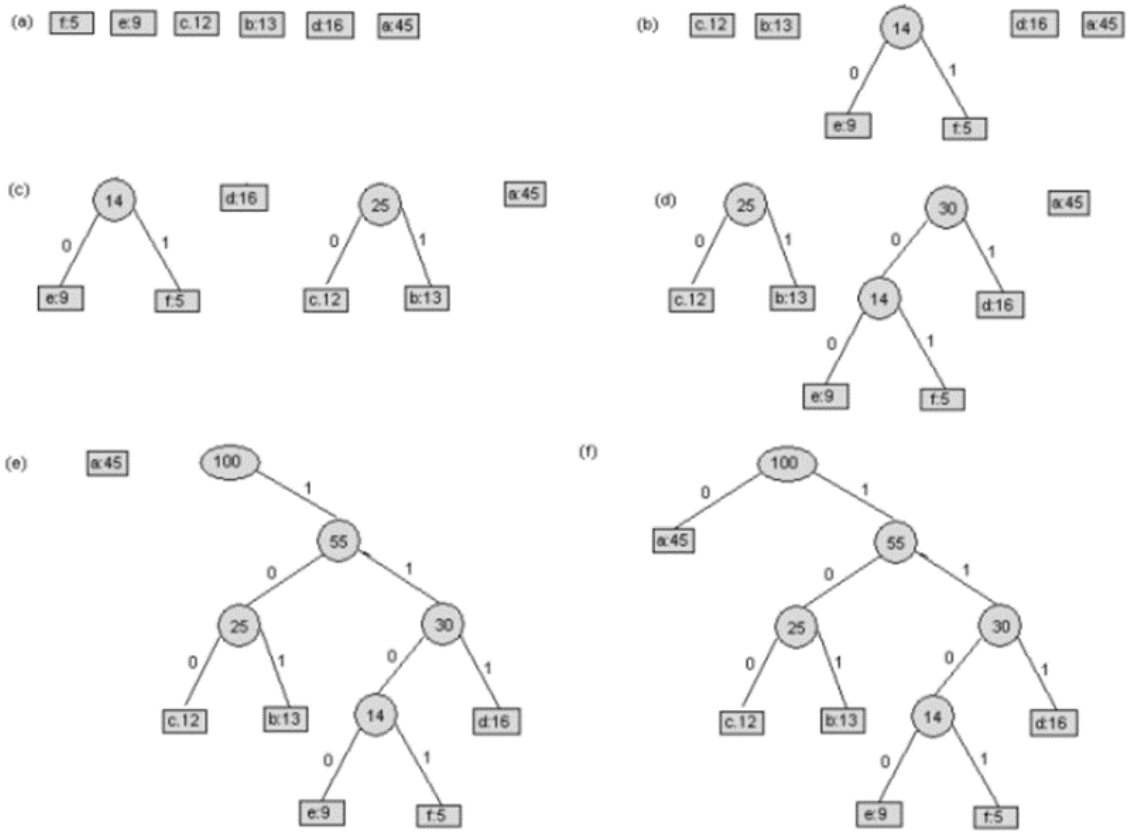
Algoritmo guloso de Huffman

A figura abaixo apresenta um algoritmo guloso para produção de um código de prefixo ótimo.

```
1:  função HUFFMAN(C)
2:      n ← |C|
3:      Q ← C
4:      para todo i = 1 até n − 1 faça
5:          alocar um novo nó z
6:          esquerda[z] ← x ← EXTRAI_MIN(Q)
7:          direita[z] ← y ← EXTRAI_MIN(Q)
8:          f[z] ← f[x] + f[y]
9:          INSIRA(Q, z)
10:     fim para
11:     retorne EXTRAI_MIN(Q)
12: fim função
```

A execução do algoritmo se dá inicializando a fila de prioridade Q, atribuindo à ela os caracteres do alfabeto C, que são os desejados para a codificação. Após essa etapa, um laço é executado a fim de substituir, na fila de prioridade, os dois nós com mais baixa prioridade por um nó intermediário, de modo que esse novo nó será o pai dos dois nós com menor frequência, criando assim uma subárvore. Após esse processo ser executado n-1 vezes, sendo n o número de caracteres em C, a raiz da árvore binária é retornada.

A figura a seguir ilustra visualmente a execução do algoritmo para o exemplo desta seção.



Considerando que a fila de prioridade Q pode ser implementada como um heap, a inicialização da mesma possui complexidade O(n). O laço para construção da árvore contribui com um custo O(n logn), uma vez que cada operação de heap contribui com O(logn) e são realizadas n-1 dessas operações. Logo, pela regra da soma, a complexidade total do algoritmo para uma entrada de tamanho n é de **O(n log n)**.

Referência das imagens:

Rocha, A., & Dorini, L. B. (2004). Algoritmos gulosos: definições e aplicações. Campinas, SP, 53.