COTUCA - Colégio Técnico de Campinas
Curso Técnico em Desenvolvimento de Sistemas
Estrutura de Dados II

Compactação de arquivos utilizando o método de Huffman

Como sabemos, cada caracter que utilizamos para compor um arquivo é representado dentro do computador por uma sequência de 8 (oito) bits. Cada número representável nesses oito bits (256 números) é associado a um símbolo através da tabela ASCII (American Standard Code for Information Interchange).

Na tabela ASCII, por exemplo, a letra 'A' é representada pelo número 97 que pode ser expresso pela sequência de bits '01100001' que é a sequência utilizada internamente pelo computador a todo momento que desejamos representar (em tela, em uma impressora ou em um arquivo) a letra 'A'. O mesmo ocorre para a letra 'a' (minúscula) e para todas as outras letras do alfabeto, para os digitos que representam os número, para os símbolos existentes em seu teclado e para todos os caracteres de controle (ENTER, Fim de linha, Fim de arquivo, Null) utilizados pelo computador. Cada um desses símbolos tem uma representação distinta.

Ao analisarmos um arquivo texto, podemos perceber que alguns símbolos da tabela ASCII aparecem em maior frequência do que outros. Conte quantas vezes a letra 'e' aparece na frase anterior...

Sabendo que todos os símbolos são representados utilizando oito bits e sabendo que alguns símbolos aparecem com maior frequência nos textos, podemos deduzir que se conseguíssemos representar esses símbolos mais frequentes com uma quantidade menor de bits, conseguiríamos uma diminuição no tamanho final do arquivo.

Em 1952, Davis A. Huffman, em resposta a um desafio de um de seus professores à sua classe no Massachussets Institute of Technology, desenvolveu um algoritmo de compressão de dados com base nessa observação. O algoritmo ficou conhecido como algoritmo de compressão de Huffman ou simplesmente algoritmo de Huffman e será ele a base deste nosso projeto.

Funcionamento do Algotirmo

O primeiro passo para a execução do algoritmo de Huffman é construir uma tabela de ocorrência de cada caracter no texto a ser compactado. Para ilustrar a explicação, utilizaremos a seguinte mensagem original:

Batatinha quando nasce, esparrama pelo chão

Para a frase acima teremos a seguinte tabela de ocorrência:

Caracter	Ocorrência	Caracter	Ocorrência
B	1	a	8
t	2	i	1
n	3	h	2
q	1	u	1
d	1	o	3
s	2	c	2
e	3	p	2
r	2	m	1
l	1	ã	1
,	1	branco	5

Para sabermos o tamanho total de um arquivo cujo conteúdo fosse a frase acima, teríamos apenas que somar as ocorrências obtendo o total de caracteres do texto e depois multiplicar este valor por 8, que é a

quantidade de bits utilizada para representar cada caracter. Fazendo as contas obtemos o número 344 (43 caracteres * 8 bits).

Pela tabela acima podemos perceber que o texto tem 20 caracteres distintos. Perceba que para representar 20 valores distintos, precisamos de apenas 5 bits e não de 8. Com essa informação podemos pensar que o tamanho do nosso arquivo pode ser reduzido a 215 (43 caracteres * 5 bits) o que já nos dá uma redução de aproximadamente 37% no tamanho original.

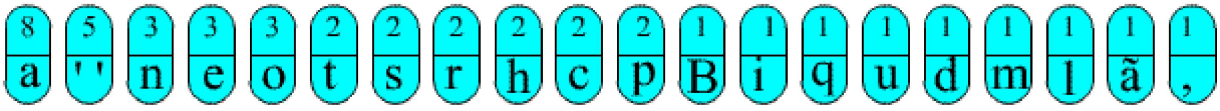
Mas a ideia é reduzir ainda mais o tamanho final do arquivo aproveitando o fato já percebido por nós de que alguns caracteres aparecem com maior frequência no texto e, representar esses caracteres com uma quantidade menor ainda de bits.

Para seguir na execução do algoritmo, vamos ordenar a tabela acima por ocorrência.

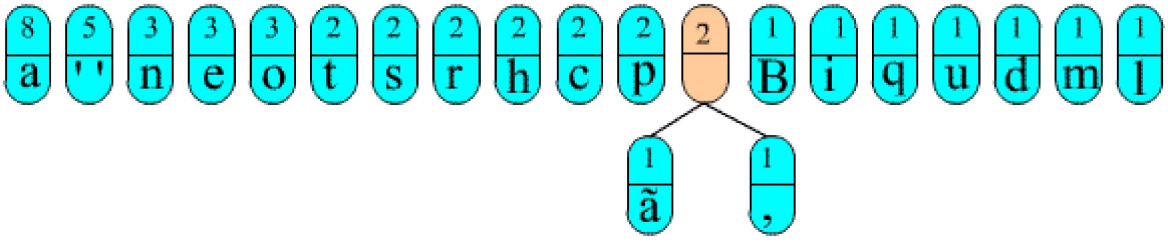
Caracter	Ocorrência	Caracter	Ocorrência
a	8	p	2
branco	5	B	1
n	3	i	1
e	3	q	1
o	3	u	1
t	2	d	1
s	2	m	1
r	2	l	1
h	2	ã	1
c	2	,	1

O próximo passo é montar uma árvore binária onde os nós-folhas representarão os caracteres do nosso texto original e o caminho a ser percorrido desde o nó-raiz até o nó-folha será utilizado para criar a representação binária de cada caracter.

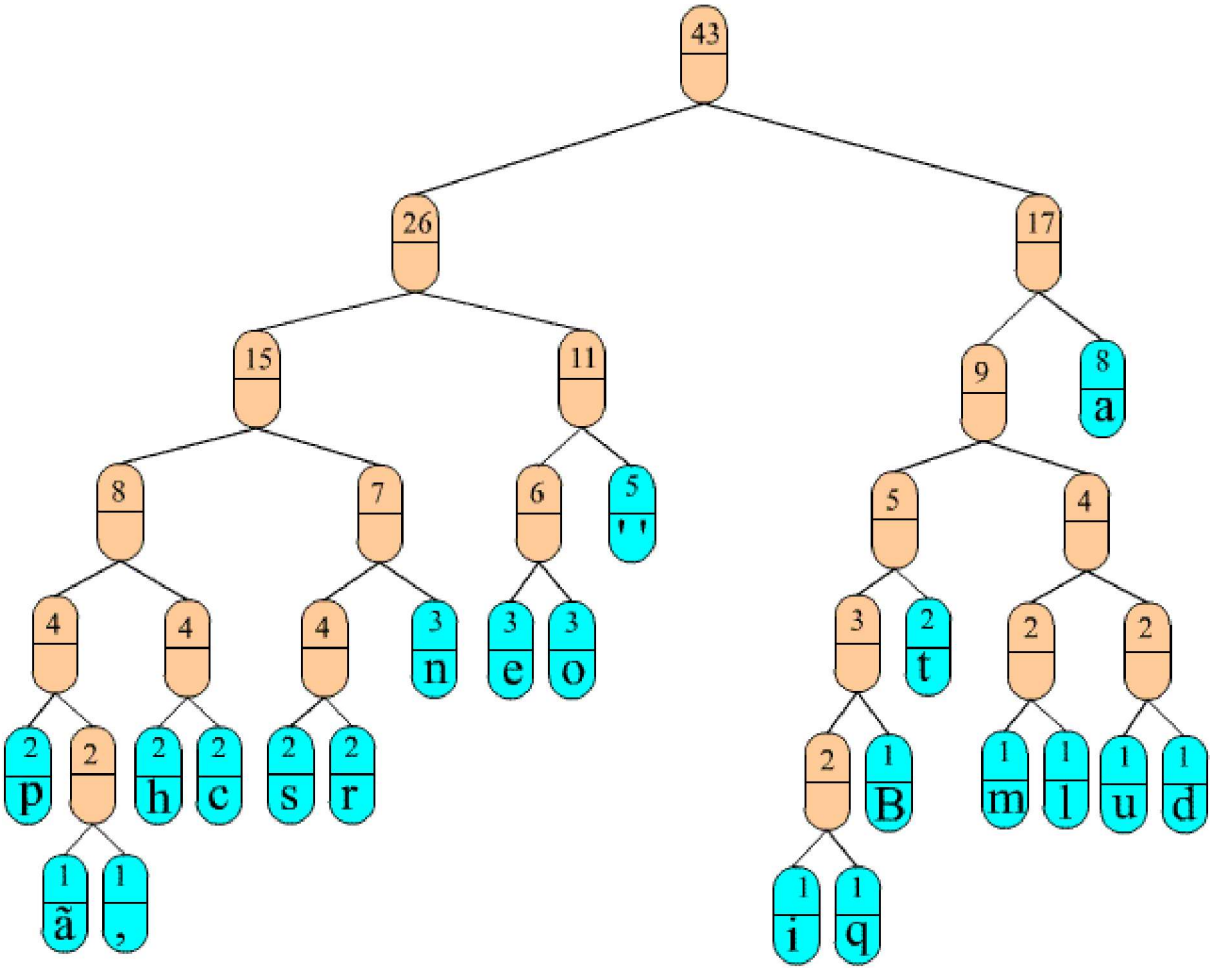
Iniciaremos o trabalho gerando um nó para cada par caracter/ocorrência:



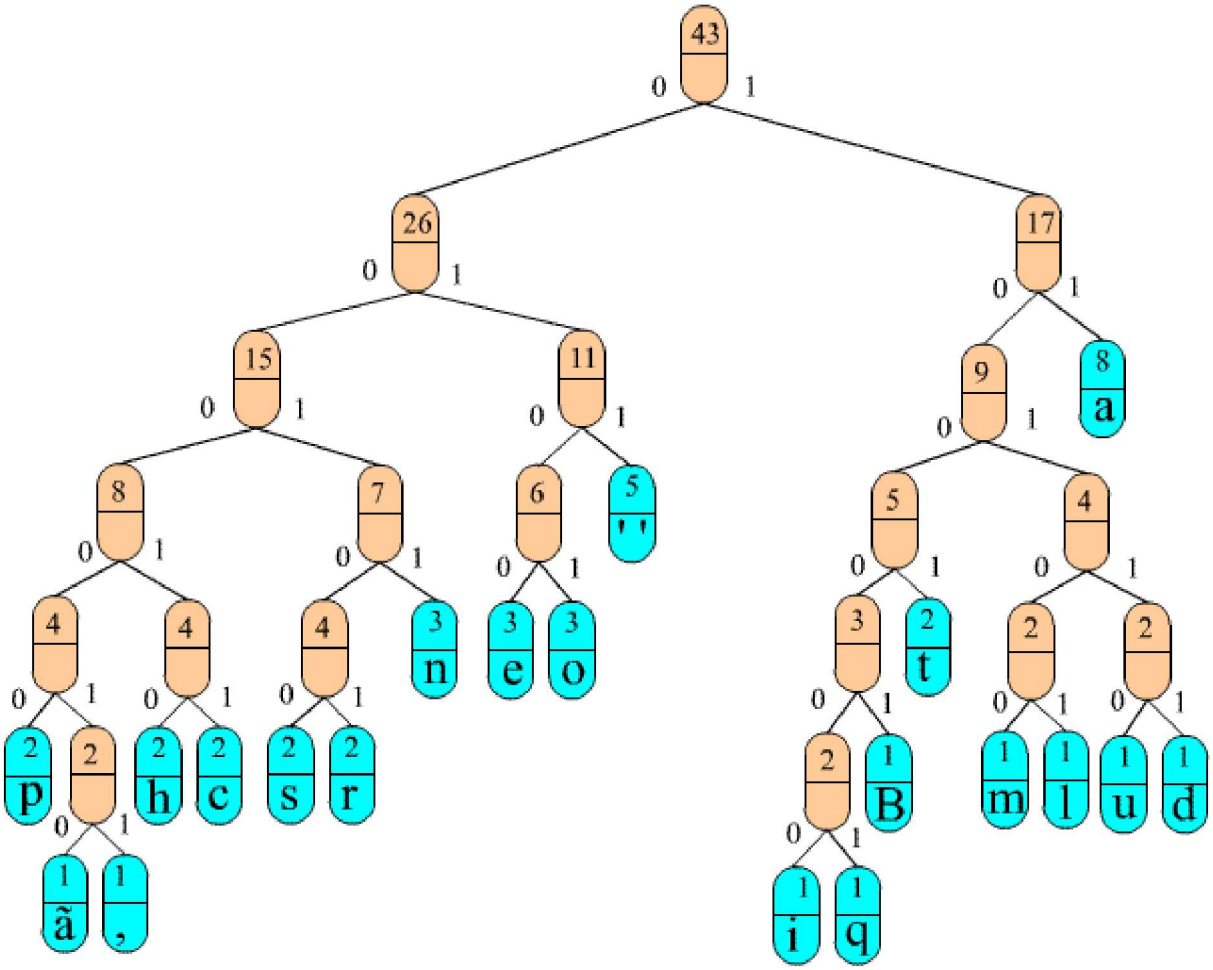
Neste ponto, o caracter já não será mais trabalhado e nosso pensamento se restringirá às ocorrências deles. O próximo passo é pegar os dois nós de menor ocorrência e montar uma pequena árvore onde eles serão as folhas e a raiz será um novo nó cujo caracter não existirá e a ocorrência será a soma das ocorrências dos dois nós-filhos. Esta árvore será inserida na sequência dos nós logo após o último número de ocorrência menor do que o de sua raiz. Realizando esta operação teremos a seguinte sequência de árvores:



Fazendo isso com todo par de arvores de menor ocorrência teremos:



Perceba que a ocorrência registrada agora no nó-raiz da árvore é exatamente a quantidade de caracteres que temos no texto original. Agora fazemos o seguinte: Daremos a cada deslocamento para um nó à direita, o bit 1 e a cada deslocamento para um nó é esquerda, o bit zero. Nossa árvore ficará assim:



Basta agora identificarmos o código de Huffman para cada um dos caracteres deste texto. Para isso basta percorrer a árvore até atingir um caracter e montar o conjunto de bits que o representa. A representação do caracter 'a', o de maior ocorrência, será '11' e a representação do caracter 'u', um dos de menor ocorrência será '10110'. Perceba que os de maior ocorrência são representados por uma sequência menor de bits e os de menor ocorrência são representados por uma sequência maior de bits.

A tabela de Huffman para os caracteres do texto original seria então:

Caracter	Código de Huffman
a	11
branco	011
n	0011
e	0100
o	0101
p	00000
t	1001
s	00100
r	00101
h	00010
c	00011
i	100000
B	10001
q	100001
u	10110
d	10111
m	10100
l	10101
ã	000010
,	000011

Nos resta agora apenas percorrer o texto original codificando os seus caracteres em bits de acordo com a tabela acima até que tenhamos um conjunto de 8 bits que serão enviados para o texto compactado.

Para efetuar a descompactação de um arquivo compactado, basta ler bit a bit do arquivo compactado e seguir a árvore montada até atingir uma folha, que será o caracter representado pela sequência de bits lida, e assim seguir até atingir o final do arquivo.

Com base no algoritmo de Huffman desenvolva classes em java que permitam a compactação e descompactação de arquivos.

Especificações Técnicas

- Uso OBRIGATÓRIO da classe BitSet para implementação do Projeto.
- Procure seguir os conceitos de OO adquiridos ao longo das aulas de programação do curso de Desenvolvimento de Sistemas.
- Desenvolva classes prestadoras de serviços separadas da classe principal do projeto, que deve apenas utilizá-las para gerenciar a execução da compactação/descompactação.
- Todos os arquivos utilizados para desenvolvimento de código devem conter ao seu início um comentário com o nome do seu curso, da disciplina e do desenvolvedor do projeto.
- Comente os blocos de código das suas classes assim como os métodos nela desenvolvidos.

BOM TRABALHO!!