Algoritmos para compressão de URLs

Ronan Loschi Rodrigues Ferreira, Fabrício Benevenuto
PPGCC - Programa de Pós-Graduação em Ciência da Computação
UFOP - Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais, Brasil
email: ronan.loschi@gmail.com, benevenuto@gmail.com

***Resumo*—Uma grande quantidade de informações é gerada pelo uso da tecnologia. Além dos bancos de dados locais a Internet tem proporcionado uma explosão de informação disponível *on-line*. Hoje a Web possui bilhões de páginas estáticas disponíveis. Cada bilhão ocupa aproximadamente 10 *terabytes* de texto corrido. Um *terabyte* tem pouco mais de um milhão de *bytes*, espaço suficiente para armazenar o texto de um milhão de livros. Em 2003 a ferramenta de busca Google (www.google.com.br) dizia ter mais de 3,5 bilhões de páginas estáticas em seu banco de dados. O armazenamento e o acesso a tal quantidade de texto apresentam um grande desafio. Portanto neste artigo vamos dar ênfase à compressão de textos e apresentar o método de compressão de Huffman, considerado. O método de compressão de Huffman, usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser compactado para determinar palavras de código binário, de tamanho variável, para representar cada símbolo. Apresentaremos ainda a ordem de complexidade dos algoritmos e os experimentos realizados.**

***Keywords*-Algoritmos, complexidade, compressão, Huffman.**

I. Introdução

Uma grande quantidade de informações é gerada pelo uso da tecnologia. Além dos bancos de dados locais, a Internet tem proporcionado uma explosão de informação disponível *on-line*. Hoje a Web possui bilhões de páginas estáticas disponíveis. Cada bilhão ocupa aproximadamente 10 *terabytes* de texto corrido. Um *terabyte* tem pouco mais de um milhão de *bytes*, espaço suficiente para armazenar o texto de um milhão de livros. Em 2003, a ferramenta de busca Google (www.google.com.br) dizia ter mais de 3,5 bilhões de páginas estáticas em seu banco de dados. O armazenamento e o acesso a tal quantidade de texto apresentam um grande desafio. Portanto neste artigo vamos dar ênfase à compressão de textos.

A compressão de textos está relacionada a maneira de representar o texto original em menos espaço. Para isto basta substituir os símbolos do texto por outros que possam ser representados usando um número menor de *bits* ou *bytes*. O ganho obtido com a compressão de textos é que o texto comprimido ocupa menos espaço de armazenamento, leva menos tempo para ser lido do disco, ser transmitido ou ser pesquisado. O preço a pagar é o custo computacional para codificar e decodificar o texto. Este custo, entretanto, está se tornando cada vez menos significativo à media que a tecnologia avança. De acordo com Patterson & Hennessy(1995), em 20 anos, o tempo de acesso a discos magnéticos aumentou aproximadamente 2 mil vezes. À medida que o tempo passa, investir mais poder computacional para alcançar a compressão de dados, torna uma opção extremamente vantajosa [1]. Além da economia de espaço, existem outros importantes aspectos a considerar:

- Velocidade de compressão e de descompressão. Em muitas situações a velocidade de descompressão é mais importante do que a velocidade de compressão. esse é o caso de banco de dados textuais e sistemas de documentação, nos quais é comum comprimir o texto uma vez e fazer muitas leituras do disco.
- Possibilidade de realizar casamento de cadeias diretamente no texto comprimido. Nesse caso, a busca sequencial pode ser mais eficiente por meio da compressão da cadeia e ser pesquisada em vez de descomprimir o texto a ser pesquisado.

Consequentemente, a pesquisa no texto comprimido é muito mais rápida porque menos *bytes* têm de ser lidos. O método de compressão de Huffman usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser compactado para determinar palavras de código binário, de tamanho variável, para representar cada símbolo. Os métodos de Huffman apresentados neste artigo atendem aos requisitos citados [2].

O artigo está organizado da seguinte forma. A Seção II apresenta o método pesquisado: Compressão de Huffman por caractere. A análise de complexidade é apresentada na Seção III, os experimentos na Seção IV e a conclusão na Seção V.

II. Métodos

A. Huffman binário

O método de codificação de Huffman (1952) é um dos mais conhecidos para compressão de dados. Dependendo das características dos dados, mantém a razão de compressão entre 20 a 90%, definida pela porcentagem que o arquivo comprimido representa em relação ao tamanho do arquivo não comprimido. Como exemplo, se um arquivo não comprimido possuir 100 bytes e o arquivo após a compressão

possuir 30 bytes, então a razão de compressão é de 30%. Em sua versão tradicional, Huffman binário, considera caracteres como símbolos. Através de uma tabela com o cálculo das frequências de ocorrência, o método representa de modo ótimo cada caractere; para isso atribui códigos mais curtos a símbolos com frequências altas e códigos mais longos a símbolos com frequências curtas. Um código único, de tamanho variável, é atribuído a cada símbolo diferente do texto e esses códigos são cadeias binárias que representam cada caractere.

Há outras maneiras de representar os caracteres, por exemplo, usando um código binário de comprimento fixo. ASCII (*Americam Standart Code for information interchange*) é um padrão de codificação de caracteres, onde cada caractere é codificado com o mesmo número de *bits* por caractere, por exemplo 8 ***bits*** , contudo esta maneira é menos eficiente que a proposta por Huffman (1952), que utiliza um código binário de comprimento variável. Por exemplo: podemos representar um texto com o alfabeto $C = abcd$ com 4 *bits*, como sendo $a = 0000$, $b = 0001$, $c = 0010$ e $d = 0011$. A tabela I apresenta as frequências de cada caractere e neste caso o arquivo de dados antes da compressão possui (35 + 18 + 23 + 9)*1000 = 85.000 caracteres ou 680.000 bits. O cálculo, para encontrar a quantidade de bits necessários para representar textos, pelo método de Huffman, é expresso pela Fómula 1. Utilizando o código binário de comprimento fixo são necessários: (35*3 + 18*3 + 23*3 + 9*3)*1000 = 255.000 bits para codificar o arquivo. Se utilizarmos o código binário de comprimento variável são necessários: (35*1 + 18*3 + 23*2 + 9*3)*1000 = 162.000 bits para codificar o arquivo. Uma economia de aproximadamente 63%, o que mostra que este é um método ótimo para representar os caracteres deste texto [3].

$$B(T) = \sum_{c \in C} f(c) d_t(c) \tag{1}$$

Onde: |C| = alfabeto c = caracteres do alfabeto $f(c)$ = frequência do caractere c no alfabeto C; $d_t(c)$= profundidade de folha ou comprimento da palavra de código em bits.

Para codificar o texto, de forma simples, é importante

Tabela I
TABELA DE FREQUÊNCIAS

Caractere:	a	b	c	d
Frequência (milhares)	35	18	23	9
Código fixo	000	001	010	011
Código variável	0	111	10	110

eliminar a ambiguidade dos códigos binários. Portanto, consideramos que nenhuma das palavras de código binário (Código fixo ou de Código variável), obtidas pelo método de Huffman, é também um prefixo de alguma outra. É comum encontrar na literatura o termo "código de prefixo"para identificar essas palavras de código binário. Por exemplo, utilizando o código de comprimento fixo da Tabela I e um arquivo de 4 caracteres $abcd$, obtemos a codificação 000.001.010.011 com 12 bits. Utilizando o código de comprimento variável da Tabela I e um arquivo de 4 caracteres $abcd$, obtemos a codificação 0.111.10.110 com 9 bits, o que mostra, mais uma vez, que este é um método ótimo para representar os caracteres deste texto. O símbolo (.) denota a concatenação das palavras de código que representam cada caractere do arquivo.

Para decodificar o texto, considerando os códigos de prefixo, sabemos que a palavra de código binário que inicia o arquivo codificado não é ambígua. Identificamos o primeiro código de prefixo, o traduzimos de volta para o caractere correspondente e o removemos do arquivo codificado. Repetimos o processo de decodificação até o final do arquivo. Utilizando o código variável da Tabela I, podemos exemplificar a decodificação da cadeia de bits 00111010011001110 como 0.111.0.10.0.110.0.111.0, que decodificamos como $abacadaba$. Para o sucesso do processo de codificação e decodificação, é necessária uma boa representação para as palavras de código binário, de forma que a palavra seja extraída com facilidade. Uma árvore de codificação, construída pelo método de Huffman, fornece tal representação, onde consideramos as folhas como os caracteres de dados e a palavra de código binário como o caminho desde a raiz até este caractere. Porém não são árvores de pesquisa binária, pois as folhas não precisam estar ordenadas em uma sequência e os nós internos não possuem chaves de caracteres.

A construção da árvore de codificação, pelo método de Huffman, utiliza uma abordagem gulosa partindo de baixo para cima. De acordo com os elementos da estratégia gulosa [3], "um algoritmo guloso obtém uma solução ótima para um problema fazendo uma sequência de escolhas. Para cada ponto de decisão no algoritmo, a opção que parece melhor no momento é escolhida". Também de acordo com a propriedade de escolha gulosa [3], "uma solução globalmente ótima pode ser alcançada fazendo-se uma escolha localmente ótima (gulosa). E ainda de acordo com a propriedade da subestrutura ótima [3], "um problema exibe subestrutura ótima se uma solução ótima para o problema contém dentro dela soluções ótimas para os subproblemas". Portanto o método de Huffman considera um conjunto de n folhas representando os caracteres que formam o vocabulário do arquivo e suas respectivas frequências. A cada etapa, de forma gulosa, as duas árvores com frequências menores são combinadas em uma única árvore e a soma de suas frequências é associada ao nó raiz. Ao final das $n-1$ etapas, temos a árvore de codificação ótima para o arquivo em questão, na qual as palavras de código binário são identificados pelos rótulos das arestas que compõem o caminho da raiz até a folha que representa o caractere. Seguindo o método de

Huffman, as arestas da esquerda são rotuladas com o *bit* 0 e as arestas da direita com o *bit* 1, com isso o *bit* zero significa caminhe para o filho da esquerda e o *bit* 1 caminhe para o filho da direita, na árvore. E para ser um código ótimo, a árvore dever estar sempre completa, ou seja, cada nó que não é uma folha deve ter dois filhos. Uma codificação ótima possui um alfabeto |C| com |C| folhas, sendo uma folha para cada caractere do alfabeto, e |C|-1 nós internos, além disso todas as frequências são positivas. A Figura 1 mostra a árvore para os códigos de comprimento fixo, que não é ótimo, e a Figura 2 mostra as atapas da construção da árvore para os códigos ótimo, de comprimento variável, de acordo com a Tabela I. As folhas são identificadas, internamente, com um caracter e externamente com sua frequência de ocorrência. Na árvore da Figura 1 temos o dicionário de código a=000, b=001, c=010, d=011 e na Figura 2 temos o dicionário de código a=0, b=111, c=10 e d=110. Os dicionários, para os caracteres, são obtidos pelo caminho desde a raiz até este caractere. Neste ponto podemos concluir que o custo da árvore, também pode ser calculado pela fórmula por 1. No nosso exemplo da Tabela I, o custo da árvore com o código de comprimento fixo é (35*3 + 18*3 + 23*3 + 9*3) = 255 bits e o custo da árvore com o código de comprimento variável é (35*1 + 18*3 + 23*2 + 9*3) = 162 bits. Além disso a representação da palavra de código, na forma de árvore, ajuda a facilitar a visualização do método de Huffman e por isso é comum encontrar na literatura [3] referência à árvore de Huffman. De acordo com [2], existem outras árvores que produzem a mesma compressão, mesmo com uma codificação alternativa, porém na maioria das aplicações são utilizadas árvores de Huffman canônicas. Uma árvore de Huffman é dita canônica quando a altura da subárvore à direita de qualquer nó nunca é menor que a altura da subárvore à esquerda. A árvore da figura 2(d) é uma árvore de Huffman canônica.

O método de Huffman binário

David A. Huffman desenvolveu em 1952 o método de Huffman, um código de paradigma guloso que produz palavras de código de prefixo ótimo. De acordo com [3], no método a seguir temos que C é um alfabeto qualquer de n caracteres c e todo $c \in C$ possui uma frequência $f[c]$. O método constrói de baixo para cima a árvore de Huffman que corresponde ao dicionário com palavras de código ótimas, para cada um dos caracteres do alfabeto C. No início o método possui um conjunto de |C| folhas e executa uma sequência de |C|-1 operações de "intercalação"para criar a árvore de Huffman final. Uma fila de prioridades Q é gerada em função das frequências $f[c]$ em ordem crescente, a análise das frequências é utilizada para selecionar os dois elementos de menor frequência. Após a seleção tem início a operação de intercalação, onde a soma das frequências dos elementos selecionados, resulta em um novo elemento. O novo elemento é inserido, ordenado, na árvore e uma nova seleção é iniciada. No exemplo da Tabela I, o alfabeto é

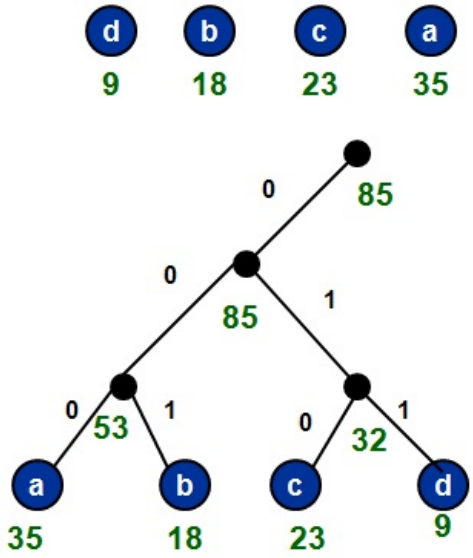

Figura 1. Árvore correspondente ao código de comprimento fixo

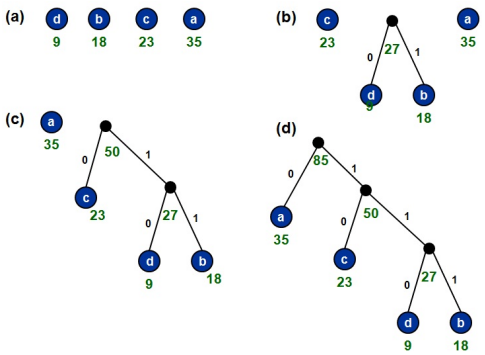

Figura 2. Passos de construção da árvore correspondente ao código de comprimento variável pelo método de Huffman

|C|=abcd e possui n=4 caracteres, 4 passos de intercalação são necessários para construir a árvore de Huffman 2, que representa o dicionário com palavras de código binário ótimas para cada caractere. A palavra de código é obitida pelas etiquetas das arestas no caminho entre a raiz e o caractere [3]. O código resultante desse processo é apresentado no Programa 1.

HUFFMAN(C) [3]

Listing 1. Timer

```
1) n <- |C|
2) Q <- C
3) for i <- 1 to n-1
4)     do alocar um novo nó z
5)     esquerda[z] <- x <- EXTRACT-MIN(Q)
6)     direita [z] <- y <- EXTRACT-MIN(Q)
```

```
7)      f[z] <- f[x] + f[y]
8)      INSERT (Q,z)
9) return EXTRACT–MIN (Q)
```

Na linha 2 inicializa a fila de prioridades Q com os caracteres C; o loop nas linhas 3 a 8 extrai repetidamente os dois nós x e y de frequência mais baixa da fila e os substitui na fila por um novo nó z representando sua intercalação. A frequencia de z é calculada como a soma das frequências de x e y na linha 7. O nó z tem x como seu filho da esquerda e y como seu filho da direita. Depois de n-1 intercalações, o único nó da esquerda na fila, a raiz da árvore de Huffman, é retornado na linha 9 [3]. O Algoritmo de Huffman apresentado, foi utilizado para orientar a implementação do código utilizado nos testes realizados. Uma vez que determinar a palavra de código ótima utiliza as propriedades de paradigma guloso, como já mencionado no início deste artigo, o método de Huffman é correto e portanto deve ser seguido. Os lemas **1** e **2** foram extraídos de [3] e mostram que a propriedade de escolha gulosa é válida e que construir códigos de prefixo ótimos tem a propriedade de subestrutura ótima e que o método de Huffman produz código de prefixo ótimo.

Lema 1: Seja ***C*** um alfabeto em que cada caractere c tem frequência $f[c]$. Sejam x e y dois caracteres em ***C*** que têm as frequências mais baixas. Então, existe um código de prefixo ótimo para ***C*** no qual as palavras de código para x e y têm o mesmo comprimento e diferem apenas no último *bit*.

Lema 2: Seja ***C*** um dado alfabeto com frequência$f[c]$ definida para cada caractere c $\in$ ***C***. Sejam dois caracteres x e y em ***C*** com frequência mínima. Seja ***C'*** o afabeto ***C*** com os caracteres x e y removidos e o (novo) caractere z adicionado, de forma que ***C'***=C-$\{x,y\}\bigcup\{z\}$; defina f para ***C'*** como para **C**, exceto pelo fato de que **f**[z]=**f**[x]+**f**[y]. Seta ***T'*** qualquer árvore representando um código de prefixo ótimo para o alfabeto ***C'***. Então a árvore T, obtida a partir de ***T'*** pela substituição do nó de folha para z por um nó interno que tem x e y como filhos, representa um código de prefixo ótimo para o alfabeto ***C***.

B. Huffman usando palavras

O método de compressão Huffman pelo considera palavras como símbolos a serem codificados e não caracteres. Considerar palavras como símbolos significa que a tabela de símbolos do codificador é exatamente o vocabulário do texto, o que permite uma integração natural entre o método de compressão e o arquivo. Este método conta a frequência das palavras e gera um código de Huffman para as palavras, em seguida comprime o texto substituindo cada palavra pelo seu código. Um texto em linguagem natural é constituído de palavras e separadores. separadores são caracteres que aparecem entre as palavras, tais como espaço, vírgula, ponto e vírgula, interrogação e assim por diante. Como a maioria dos separadores é o espaço simples entre palavras, uma forma eficiente de lidar com palavras e separadores é representar o espaço simples de forma implícita no texto comprimido. Nesse modelo, se uma palavra é seguida de um espaço, então, somente a palavra é codificada. senão, a palavra e o separador são codificados separadamente. No momento da decodificação, supõe-se que um espaço simples segue cada palavra, a não ser que o próximo símbolo corresponda a um separador.

O métoto de huffman usando palavras (Moura (1999); Moura, Navarro, Ziviani e Baeza-Yates, 2000; Ziviani, Moura, Navarro e Baeza-Yates, 2000) é também uma abordagem gulosa que constrói uma árvore de codificação partindo de baixo para cima. No início há um conjunto de n folhas representando as palavras do vocabulário e suas respectivas frequências. A cada iteração, as duas árvores com menores frequêncuias são combinadas em uma árvore única, e a soma de suas frequências é associada ao ní raiz. Ao final das n-1 iterações, obtém a árvore de codificação, na qual os códigos associados a cada palavra são representados pela sequência dos rótulos das arestas que levam da raiz à folha que a representa. Da mesma maneira que acontece no método de Huffman binário II-A. Como exemplo do funcionamento do método de Huffman usando palavras, consideramos a frase "'‘para cada rosa rosa, uma rosa é uma rosa"', na qual o conjunto de símbolos representando o alfabeto |C| é dado por {"'‘para"', "'‘cada"', "'‘rosa"', "'‘, espaço"', "'‘uma"', "'‘é"'}, e as frequências são 1, 1, 4, 1, 2, 1, respectivamente, representado na figura 3. A palavra mais frequênte, no caso "'‘rosa"', recebe o código mais curto, no caso "'‘0"', com isso o método de Huffman produz a árvore de codificação que diminui o tamanho do arquivo comprimido. A figura 4 apresenta os passos seguintes para a construção da árvore de Huffman usando o método de codificação por palavra [2]. **O Método de Huffman usando palavras:**

O algoritmo para o método de Huffman usando palavras, atribuído a Moffat e Katajainen (1995) e descrito em Moffat e Turpin (2002), e descrito em [2], calcula os comprimentos dos códigos em vez dos códigos propriamente ditos, uma vez que a compressão atingida é a mesma, independentemente dos códigos utilizados. Além disso é possível gerar a palavra de código binário, de uma palavra, a partir dos comprimentos dos códigos obitidos. O mesmo é válido para os algoritmos de codificação e decodificação. A entrada do algoritmo é um vetor A contendo as frequências das palavras em ordem não crescente, como apresentado no exemplo da figura

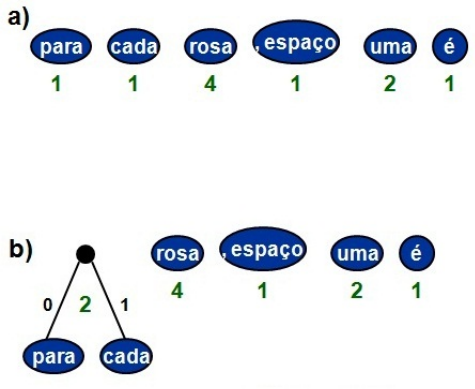

Figura 3. Passos de construção da árvore correspondente ao código de comprimento variável pelo método de Huffman usando palavras

figura 3. O algoritmo divide-se em três fases distintas, na primeira é feita a construção da árvore de Huffman, como apresentado na figura 4, na segunda fase são calculadas as profundidades dos nós internos da árvore e na terceira fase, as profundidades dos nós folhas da árvore, a partir da profundidade dos nós internos. As profundidades dos nós folhas são então utilizados para a obtenção do código de Huffman para cada palavra. Métodos de Huffman baseados em caracteres comprimem o texto para aproximadamente 60%, enquanto métodos de Huffman baseado em palavras comprimem o texto para valores pouco acima de 25%. [2]

III. Análise de Complexidade

Método de Huffman binário:
A análise do tempo de execução do método de Huffman supõe que $\boldsymbol{Q}$ é implementada como um *heap* binário (*a estrutura de dados heap binário é um objeto arranjo que pode ser visto como uma árvore binária quase completa, onde cada nó da árvore corresponde a um elemento do arranjo que armazena o valor do nó. As operações básicas sobre heaps são executadas em um tempo máximo proporcional à altura da árvore, e assim demoram um tempo* O(lg$n$)). Para um alfabeto $\boldsymbol{C}$ de $\boldsymbol{n}$ caracteres, a inicialização de $\boldsymbol{Q}$ na linha 2 pode ser executada com ordem de complexidade, em relação tempo O(n); complexidade alcançada através da Análise Armoortizada. O loop das linhas 3 a 8 é executado exatamente $\boldsymbol{n}$-1 vezes e, como cada operação de heap exige o tempo O(lg$n$), o loop contribui com O($n$lg$n$) para o tempo de execução. Desse modo, o tempo de execução total do método de Huffman, em um conjunto de n caracteres é $\mathbf{O}(\boldsymbol{n}\mathbf{lg}\boldsymbol{n})$. [3]

Método de Huffman usando palavras:
O algoritmo, para o Método de Huffman usando palavras, calcula os comprimentos dos códigos a partir do vetor A contendo as frequências das palavras, em ordem não crescente, a um custo $\mathbf{O}(\boldsymbol{n})$ em tempo e espaço. [2]

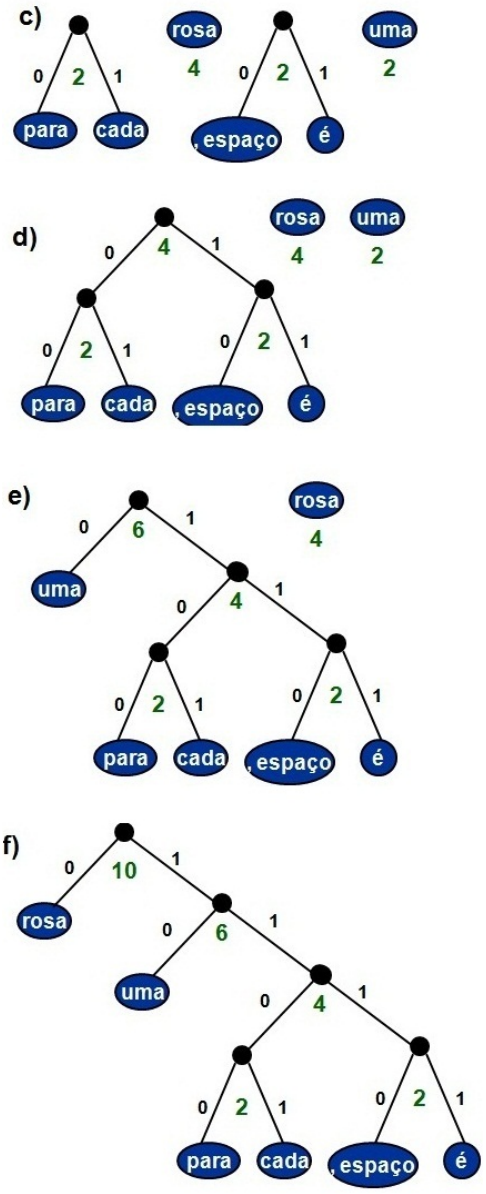

Figura 4. Passos de construção da árvore correspondente ao código de comprimento variável pelo método de Huffman usando palavras

IV. Experimentos

Nos experimentos realizamos, veja Tabela II e Tabela III a compressão dos arquivos de dados através do método de Huffman binário implementado. Com base na afirmação de que métodos de Huffman baseados em caracteres comprimem o texto para aproximadamente 60%, do tamanho original, enquanto métodos de Huffman baseado em palavras comprimem o texto para valores pouco acima de 25%, do tamanho original, encontrada em [2], deduzimos que existe uma diferença de aproximadamente 35% na comparação

entre estes dois métodos.
Como não implementamos o método de Huffman usando palavras, vamos aplicar a diferença de 35% sobre os valores dos experimentos com o método de Huffman usando caracteres. Com isso pretendemos simular na Tabela III a execução do método não implementado e realizar a comparação entre as taxas de compressão, dos arquivos utilizados nos experimentos.

V. Conclusão

O método de compressão de Huffman pesquisado é uma boa opção para realizar a compressão de dados, principalmente quando comparado com o método ASCII que atribui um tamanho fixo para as palavras de código. Entre os métodos de compressão por caractere e por palavra, fica claro a vantagem do segundo, que possui uma taxa de compressão melhor e ordem de complexidade também melhor. No caso dos dois métodos, a construção da árvore de Huffman é um passo importante para a compreensão das diferentes soluções propostas. Os experimentos realizados, embora simples, demonstram a utilidade e a vantagem de se comprimir dados com o objetivo de ganhar espaço em disco. Embora não explorados neste artigo, outros métodos de compressão como os da família de Ziv-Lempel, podem ser pesquisados como trabalhos futuros, além da implementação do método de Huffman usando palavras e experimentos demonstrando a pesquisa direta no texto comprimido e acesso direto a qualquer parte do texto comprimido, sem necessidade de descomprimir.

Este trabalho de pesquisa contribui diretamente com o tema de dissertação "Novos métodos para compressão de URLs no Twitter" [4], [5], [6]. [1]. pois revela detalhes sobre o método de Huffman que servem como parte da revisão bibliográfica necessária. Além disso, demonstra a importância e contribuição da disciplina de Projeto e Analise de Algoritmos em uma aplicação real. A Tabela II apresenta valores.

Referências

[1] Wikipedia, "Compressão de dados," 2011, http://pt.wikipedia.org/wiki/Algoritmodecompressao, Acesso em 06/06/2011.

[2] N. Ziviani, *Projeto de Algoritmos com Implementações em Pascal e C*, 3rd ed. Cengage Learning, 2011, iSBN: 978-85-221-1050-6.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009, iSBN-13: 978-0-262-53305-8.

[4] D. Antoniades, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E. Markatos, and T. Karagiannis, "we.b: The web of short urls," in *Int'l Conference on World Wide Web.*, 2011, pp. 715–724.

[5] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS), Redmond, Washington, USA. July*, 2010, pp. 1–9.

[6] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@spam: The underground on 140 characters or less," in *ACM conference on Computer and communications security (CCS)*, 2010, pp. 27–37.

Tabela II
TABELA DE EXPERIMENTOS REAIS - MÉTODO HUFFMAN BINÁRIO

Método:	Tamanho inicial	Tamanho final	Razão de compressão	Caracteres
Huffman binário	116KB	67KB	57,2%	120.739
Huffman binário	2,5MB	1,9MB	76%	2.657.033
Huffman binário	5,2MB	4,2MB	81%	5.458.215
Huffman binário	10,4MB	8,4MB	81%	10.916.432
Huffman binário	25MB	20,2MB	81%	209.447.656

Tabela III
TABELA DE EXPERIMENTOS SIMULADOS - MÉTODO HUFFMAN USANDO PALAVRAS, COM BASE NA TABELA II

Método:	Tamanho inicial	Tamanho final	Razão de compressão	Caracteres
Huffman binário	116KB	25,7KB	22,2%	120.739
Huffman binário	2,5MB	1,0MB	41%	2.657.033
Huffman binário	5,2MB	2,4MB	46%	5.458.215
Huffman binário	10,4MB	4,8MB	46%	10.916.432
Huffman binário	25MB	11,5MB	46%	209.447.656