Universidade Federal do Maranhão
DEINF - Departamento de Informática
Disciplina – Estrutura de Dados 2006.1
Professor – MsC. Carlos de Salles
Monitor – Bruno Seabra Nogueira Mendonça Lima – CP04120-55

Terceira Avaliação – Códigos de Huffman X Lempel-Ziv-Welch (LZW)

1.Introdução

O objetivo desta avaliação é fazer a implementação dos 2 algoritmos de compactação Huffman e LZW e confrontá-los em um mini-relátorio.

Neste relatório deve constar:

O que o seu programa é capaz de fazer, o que não conseguiu implementar a tempo, as dificuldades encontradas, gráficos confrontando a porcentagem de compactação entre Huffman e LZW em arquivos: texto, imagem e mp3.

A seguir segue uma breve explicação sobre cada algoritmo assim como um exemplo passo a passo para codificação e decodificação usando Huffman e LZW.

2. Código de Huffman

Uma aplicação interessante de árvores binárias em codificação de informações a serem transmitidas e compactação de arquivos são os códigos de Huffman.

Para enviar uma mensagem através de um cabo de transmissão, os caracteres da mensagem são enviados um a um, modificados através de alguma tabela de codificação. Em geral, este código forma um número binário. Como a velocidade de transmissão é importante, é interessante tornar a mensagem tão curta quanto possível sem perder a capacidade de decodificar o texto enviado. Um algoritmo de determinação de códigos binários de comprimentos variados para caracteres (não é o caso que cada caractere seja representado com o mesmo número de bits), baseado na freqüência de uso destes caracteres, foi sugerido por Shannon [5] (veja também [6]) Fano [3] e logo depois esta idéia foi aperfeiçoada por D.A. Huffman [4]. A idéia é associar números binários com menos bits aos caracteres mais usados num texto. Desta maneira espera-se que o número de bits economizados para codificar os caracteres que ocorrem com maior freqüência em um texto seja mais que o suficiente para cobrir o déficit que ocorre ao codificarmos os caracteres que ocorrem raramente com cadeias longas de bits.

O algoritmo de Huffman que será implementado para codificar um arquivo consistirá de três fases:

(1) Na primeira fase a freqüência de cada caractere que ocorre no texto deverá ser calculada.

(2) A segunda fase do algoritmo de Huffman consiste em construir uma árvore binária baseada na freqüência de uso destas letras de modo que as mais freqüentemente usadas apareçam mais perto da raiz que as menos freqüentemente usadas. Chamaremos esta árvore binária de árvore de Huffman.

Em cada passo desta fase teremos uma coleção de árvores binárias (ou seja, uma floresta formada por árvores binárias). As folhas de cada uma destas árvores correspondem a um conjunto de caracteres que ocorrem no texto. À raiz de cada uma destas árvores será associado um número que corresponde à freqüência com que os caracteres associados às folhas desta árvore ocorrem no texto. Escolheremos então as duas árvores desta floresta com a menor freqüência e as transformaremos em uma única árvore, ligando-as a uma nova raiz cujo valor será dado pela soma dos valores das freqüências das duas

subárvores (veja um exemplo na próxima seção).

(3) Finalmente na terceira fase a árvore de Huffman será usada para codificar e decodificar o texto. Nas Seções 4 e 5 é mostrado um exemplo de como usar a árvore para codificar e decodificar.

A árvore de Huffman construída durante a codificação será utilizada na decodificação (logo, a árvore de Huffman deverá ser armazenada junto com o texto que foi codificado).

3. Construção da árvore de Huffman: Um exemplo

Suponha que uma mensagem seja composta pelos caracteres A, B, C, D, E, R e que a freqüência destas letras na mensagem seja, respectivamente, 22, 9, 10, 12, 16, 8.

A	B	C	D	E	R
22	9	10	12	16	8

O algoritmo de Huffman contrói uma árvore binária (a árvore de Huffman) baseada na freqüência de uso destas letras de modo que as mais freqüentemente usadas {A, E} apareçam mais perto da raiz que as menos freqüentemente usadas {B, R}.

A construção desta árvore binária é feita de baixo para cima (das folhas para a raiz), começando a partir das letras menos usadas até atingir a raiz. Nesta árvore, as letras serão representadas nas folhas e os seus vértices internos conterão um número correspondente à soma das freqüências dos seus descendentes.

Vejamos a construção da árvore de Huffman para o alfabeto e freqüências do exemplo acima. Em cada passo do algoritmo existe uma coleção de árvores de Huffman. São escolhidas duas com as menores freqüências associadas e elas são transformadas em uma só árvore cujo valor é a soma dos valores dessas duas. Inicialmente, cada uma das letras será uma árvore composta apenas pela raiz e cujo conteúdo é a freqüência com que esta letra ocorre no texto (veja Figura 1).

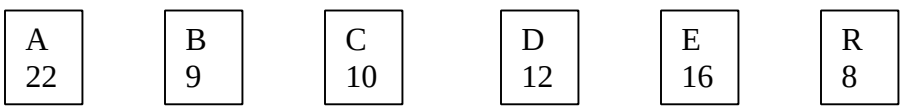


Figura 1. Floresta no início do algoritmo.

Escolha as duas raízes de menor freqüência (árvores com R e B com freqüências 8 e 9, respectivamente) e junte essas duas árvores, formando uma nova árvore, que fará parte da coleção (veja Figura 2).

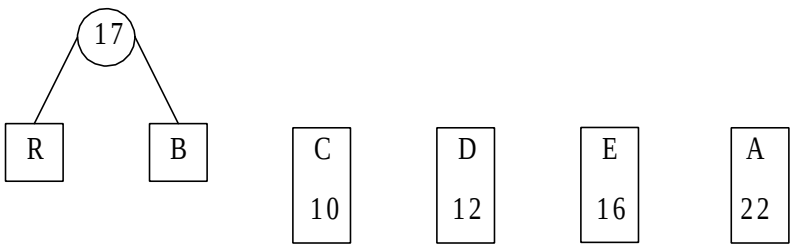


Figura 2. Floresta após termos juntado duas árvores (das freqüências 8 e 9).

Repetindo o processo, escolha as árvores com raízes 10 e 12, e junte-as. A coleção de árvores tem a forma mostrada na Figura 3:

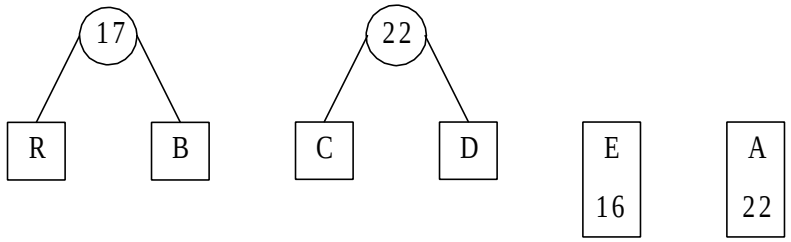


Figura 3. Floresta após termos juntado as árvores com freqüências 10 e 12, formando a árvore com freqüência associada 22.

Continuando o mesmo processo, escolha as árvores com freqüências 16 e 17, formando a floresta que pode ser vista na Figura 4.

Depois, juntamos as árvores com freqüências 22 e 22 formando a floresta exibida na Figura 5.

E, finalmente juntamos as duas árvores restantes para obtermos a árvore mostrada na Figura 6. Assim, a árvore de Huffman está completamente construída.

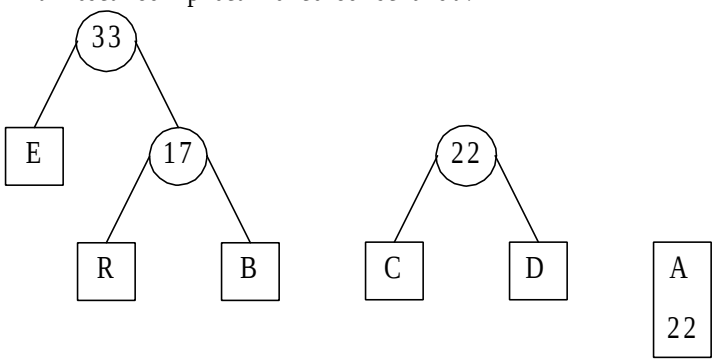


Figura 4. Floresta após termos juntado as árvores com freqüências 16 e 17.

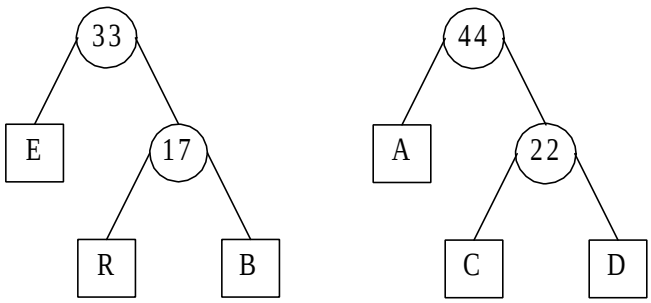


Figura 5. Floresta após termos juntado as árvores com freqüências 22 e 22.

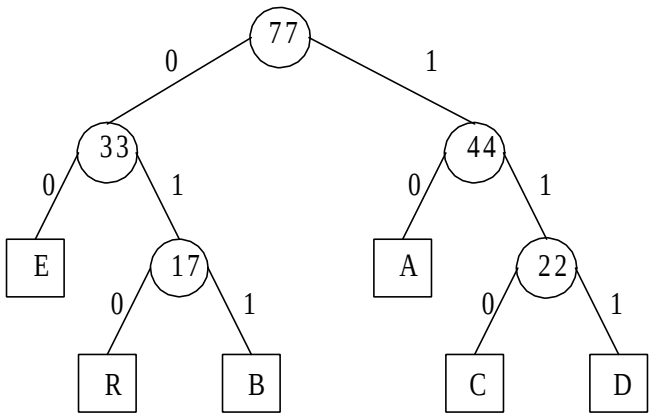


Figura 6. Árvore de Huffman.

4. Codificação

Associe 0 às arestas da árvore de Huffman que ligam um nó com seu filho esquerdo e 1, às arestas que ligam um nó com seu filho direito. O código correspondente a cada letra será formado pelo número binário associado ao caminho da raiz até a folha correspondente. Com isso, a tabela de códigos resultante da árvore de Huffman da Figura 6 é:

A	B	C	D	E	R
10	011	110	111	00	010

Repare que na codificação alguns caracteres diminuem de tamanho (menos de 8 bits), mas outros podem aumentar (até 256 bits).

5. Decodificação

Para decodificar uma mensagem obtida através da tabela acima, por exemplo:

1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1 0 1 0 1 0,

basta ir utilizando cada bit da mensagem para percorrer a árvore de Huffman desde a raiz até alguma folha, quando se obtém o símbolo decodificado. Volte então para a raiz e continue a percorrer a árvore para decodificar o próximo símbolo. Para o exemplo da seqüência de bits acima, o texto correspondente é ABRACADABRA. Observe como este texto foi compactado de 11 bytes (88 bits) para 28 bits.

6. Especificações da implementação

Faça um programa que permita codificar e decodificar arquivos, usando códigos de Huffman.

Codificação: O programa deve ler um arquivo, compactar..
Existem duas maneiras para isso ser feito:
1º Gerar os arquivos huffman.cod (arquivo de inteiros contendo a codificação do arquivo aberto) e huffman.arv (arquivo contendo a árvore de Huffman). Dessa maneira mesmo depois de fechado seu programa poderá descompactar o arquivo.
2º Deixar os arquivos discutidos na memória, assim seu programa só poderá descompactar o arquivo atual na execução atual do programa.
Ambas as maneiras são válidas.
OBS:
A árvore de Huffman deve ser representada utilizando-se alocação dinâmica. Em cada passo da construção da árvore de Huffman o algoritmo escolhe, dentre uma floresta de árvores com freqüências, as duas árvores com as menores freqüências. Para implementar este passo do algoritmo o seu programa deve utilizar uma fila de prioridades (heap). Cada célula da fila de prioridades deve conter a freqüência de uma das árvores da floresta junto com uma referência (apontador) para a raiz da árvore correspondente.

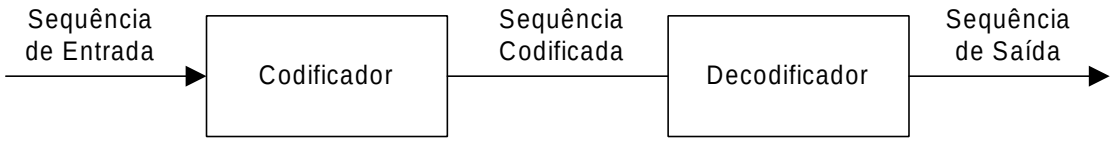
Decodificação: O programa deve descompactar o arquivo mantendo sua integridade física anterior. Para isto, é preciso reconstruir a árvore Huffman correspondente ao arquivo a ser descompactado. A decodificação depende da maneira como você armazenar a árvore e os códigos de huffman. Lembrando ambas maneiras são válidas.

7. Lempel-Ziv-Welch

O algoritmo LZW é uma extensão ao algoritmo LZ proposto por Lempel e Ziv[1], em 1977. A diferença em relação ao LZ original é que o dicionário não está vazio no início, contendo todos os caracteres individuais possíveis.

Abaixo são mostrados os algoritmos usados para a codificação e decodificação (LZW). As convenções adotadas são:

raiz	caracter individual
string	uma sequência de um ou mais caracteres
palavra código	valor associado a uma *string*
dicionário	tabela que relaciona *palavras código* e *strings*
P	string que representa um **prefixo**
C	caracter
cW	palavra código
pW	palavra código que representa um **prefixo**
X <= Y	string X assume o valor da string Y
X+Y	concatenação das string X e Y
string(w)	string correspondente à palavra código w



8. Codificação

1. No início o dicionário contém todas as raízes possíveis e P é vazio;
2. C <= próximo caracter da sequência de entrada;
3. A string P+C existe no dicionário ?
   a. se *sim*,
      i. P <= P+C;
   b. se *não*,
      i. coloque a palavra código correspondente a P na sequência codificada;
      ii. adicione a string P+C ao dicionário;
      iii. P <= C;
4. Existem mais caracteres na sequência de entrada ?
   a. se *sim*,
      i. volte ao passo 2;
   b. se *não*,
      ii. coloque a palavra código correspondente a P na sequência codificada;
      iii. FIM.

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", *IEEE Transaction on Information Theory*, vol. IT-23, pp. 337-343, 1977

9. Decodificação

1. No início o dicionário contém todas as raízes possíveis;
2. cW <= primeira palavra código na sequência codificada (sempre é uma raiz);
3. Coloque a string(cW) na sequência de saída;
4. pW <= cW;
5. cW <= próxima palavra código da sequência codificada;
6. A string(cW) existe no dicionário ?
   a. se *sim*,
      i. coloque a string(cW) na sequência de saída;
      ii. P <= string(pW);
      iii. C <= primeiro caracter da string(cW);
      iv. adicione a string P+C ao dicionário;
   b. se *não*,
      i. P <= string(pW);
      ii. C <= primeiro caracter da string(pW);
      iii. coloque a string P+C na sequência de saída e adicione-a ao dicionário;
7. Existem mais palavras código na sequência codificada ?
   a. se *sim*,
      i. volte ao passo 4;
   b. se *não*,
      i. FIM.

10. Construção do dicionário: Um Exemplo

Codificação

Deseja-se codificar a sequência abaixo:

Posição	1	2	3	4	5	6	7	8	9
Caracter	a	b	b	a	b	a	b	a	c

Seguindo-se o ***algoritmo de codificação*** e chamando SC a sequência codificada, temos:

1. SC = ∅
2. Inicialização do dicionário com as raízes:

Palavra Código	**String**
1	a
2	b
3	c

3. P = ∅
4. C = ‘a’ (posição 1)
5. P+C = ‘a’ existe no dicionário
6. P = P+C = ‘a’
7. C = ‘b’ (próximo caracter da sequência de entrada – posição 2)
8. P+C = ‘ab’ não existe no dicionário
9. SC = 1 (corresponde a P = ‘a’)

10.

Palavra Código	String
1	a
2	b
3	c
4	ab

11. P = C = ‘b’
12. C = ‘b’ (próximo caracter da sequência de entrada – posição 3)
13. P+C = ‘bb’ não existe no dicionário
14. SC = 1 2 (corresponde a P = ‘b’)
15.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb

16. P = C = ‘b’
17. C = ‘a’ (próximo caracter da sequência de entrada – posição 4)
18. P+C = ‘ba’ não existe no dicionário
19. SC = 1 2 2 (corresponde a P = ‘b’)
20.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba

21. P = C = ‘a’
22. C = ‘b’ (próximo caracter da sequência de entrada – posição 5)
23. P+C = ‘ab’ existe no dicionário
24. P = P+C = ‘ab’
25. C = ‘a’ (próximo caracter da sequência de entrada – posição 6)
26. P+C = ‘aba’ não existe no dicionário
27. SC = 1 2 2 4 (corresponde a P = ‘ab’)

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba

28. P = C = ‘a’
29. C = ‘b’ (próximo caracter da sequência de entrada – posição 7)
30. P+C = ‘ab’ existe no dicionário
31. P = P+C = ‘ab’
32. C = ‘a’ (próximo caracter da sequência de entrada – posição 8)
33. P+C = ‘aba’ existe no dicionário
34. C = ‘c’ (próximo caracter da sequência de entrada – posição 9)

35. P+C = ‘abac’ não existe no dicionário
36. S = 1 2 2 4 7 (corresponde a P = ‘aba’)
37.

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba
8	abac

38. P = C = ‘c’
39. Não existem mais caracteres a codificar, logo, SC = 1 2 2 4 7 3 (correspondente a P = ‘c’)
40. FIM

Decodificação

Deseja-se agora decodificar a sequência codificada acima:

Posição	1	2	3	4	5	6
Palavra Código	*1*	*2*	*2*	*4*	*7*	*3*

Seguindo-se o ***algoritmo de decodificação*** e chamando SS a sequência de saída, temos:

1. SS = ∅
2. Inicialização do dicionário com as raízes:

Palavra Código	String
1	a
2	b
3	c

3. cW = *1* (primeira palavra código)
4. SS = *a* (corresponde a string(cW))
5. pW = cW = *1*
6. cW = *2* (próxima palavra código da sequência codificada – posição 2)
7. string(cW) = ‘b’ existe no dicionário
8. SS = a *b* (corresponde a string(cW))
9. P = string(pW) = ‘a’
10. C = 1º caracter da string(cW) = ‘b’
11. P+C = ‘ab’

Palavra Código	String
1	a
2	b
3	c
4	ab

12. pW = cW = *2*
13. cW = *2* (próxima palavra código da sequência codificada – posição 3)
14. string(cW) = ‘b’ existe no dicionário
15. SS = a b *b* (corresponde a string(cW))
16. P = string(pW) = ‘b’
17. C = 1º caracter da string(cW) = ‘b’
18. P+C = ‘bb’

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb

19. pW = cW = *2*
20. cW = *4* (próxima palavra código da sequência codificada – posição 4)
21. string(cW) = ‘ab’ existe no dicionário
22. SS = a b b *a b* (corresponde a string(cW))
23. P = string(pW) = ‘b’
24. C = 1º caracter da string (cW) = ‘a’
25. P+C = ‘ba’

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba

26. pW = cW = *4*
27. cW = *7* (próxima palavra código da sequência codificada – posição 5)
28. sting(cW) = ‘ab’ não existe no dicionário
29. P = string(pW) = ‘ab’
30. C = 1º caracter da string (pW) = ‘a’
31. SS = a b b a b *a b a* (correspondente a P+C = ‘aba’)
32. P+C = ‘aba’

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba

33. pW = cW = 7
34. cW = 3 (próxima palavra código da sequência codificada – posição 6)
35. sting(cW) = ‘c’ existe no dicionário
36. SS = a b b a b a b a *c* (correspondente a string(cW))
37. P = string(pW) = ‘aba’
38. C = 1º caracter da string(cW) = ‘c’

39. P+C = ‘abac’

Palavra Código	String
1	a
2	b
3	c
4	ab
5	bb
6	ba
7	aba
8	abac

40. FIM

11. Especificações da implementação

Faça um programa que permita codificar e decodificar arquivos, seguindo o algoritmo LZW proposto.

Codificação: O programa deve ler um arquivo, compactar seguindo o algoritmo LZW dado como exemplo. Novamente tendo a opção de salvar o dicionário ou mantê-lo na memória é dada.

Decodificação: O programa deve descompactar o arquivo mantendo sua integridade física anterior.