UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

MATA 49 – PROGRAMAÇÃO DE SOFTWARE BÁSICO

PROFESSOR: LEANDRO ANDRADE

TRABALHO PRÁTICO 20191.1
(06/05/2019)

Informações Gerais

- O trabalho deve ser feito em dupla
- A data de entrega será no dia **05 de junho de 2019** em sala de aula (16:40) e cada dupla terá cerca de 5 minutos para explicar sua solução e entregar o código fonte
  - **É indispensável que cada aluno apresente conhecimento sobre o código desenvolvido, pois isso também será considerado como critério de avaliação**
- A identificação de plágio resultará na anulação da nota da avaliação de todos os envolvidos
- **As entradas e saídas devem seguir rigorosamente a especificação.** Então leia atentamente as informações abaixo
- O programa será avaliado através de 10 testes, onde cada um valerá um ponto.

Descrição do Geral

A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. Ele foi desenvolvido em 1952 por David A. Huffman que era, na época, estudante de doutorado no MIT, e foi publicado no artigo "A Method for the Construction of Minimum-Redundancy Codes".

Uma árvore binária completa, chamada de árvore de Huffman é construída recursivamente a partir da junção dos dois símbolos de menor probabilidade, que são então somados em símbolos auxiliares e estes símbolos auxiliares recolocados no conjunto de símbolos. O processo termina quando todos os símbolos forem unidos em símbolos auxiliares, formando uma árvore binária. A árvore é então percorrida, atribuindo-se valores binários de 1 ou 0 para cada aresta, e os códigos são gerados a partir desse percurso.

O resultado do algoritmo de Huffman pode ser visto como uma tabela de códigos de tamanho variável para codificar um símbolo da fonte. Assim como em outros métodos de codificação, os símbolos mais comuns são geralmente representados usando-se menos dígitos que os símbolos que aparecem com menos frequência.

Fonte: https://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Huffman#Algoritmo

Algoritmo

Codificação:

1. Gerar uma lista de caracteres e suas respectivas frequências de ocorrência no texto
2. Ordenar a lista de caracteres de modo crescente
3. Gerar a árvore de Huffman

    I. Retire da lista de caracteres os dois primeiros nós e inclua como filhos direito e esquerdo de um nó de agregação, que não corresponde a nenhum caractere, mas cuja freqüência de ocorrência seja igual à soma das freqüências de seus filhos
    II. Inclua na lista de prioridades o nó de agregação (mantendo a ordem crescente)
    III. Repita os passos I e II até que a lista de prioridades contenha um só nó, que é a raiz da árvore de Huffman
4. Gerar a tabela de código de cada caractere
    I. Atribua rótulo 0 à aresta que conduz ao filho esquerdo e 1 para a aresta que conduz ao filho direito
    II. O código de cada caractere é obtido pela trajetória da raiz até a folha da árvore
5. A partir do código de huffman de cada caractere gerar o texto compactado
6. Gerar arquivo compactado com tabela de códigos e caracteres compactados

Decodificação:

1. Utilizando a tabela de codificações monte a árvore de Huffman
2. Escolha o primeiro bit do texto comprimido como o bit corrente
3. Desça um nível da árvore de Huffman, as partir da raiz, de acordo com o valor do bit corrente (0 filho esquerdo ou 1 filho direito) e faça o bit corrente igual ao próximo bit a decodificar.
4. Quando for atingida uma folha da árvore de Huffman, substituir os bits correspondentes à trajetória, pelo caractere contido nessa folha.
5. Repetir os passos 3 e 4 até que que o string de bits termine

Para melhor entendimento do funcionamento da compressão de Huffman e a geração da sua árvore vibe os links abaix:

- Codificação de Huffman e compressão de dados
    - https://bitismyth.wordpress.com/2016/01/22/codificacao-de-huffman-e-compressao-de-dados/
- Implementação em C (não revisada)
    - https://gist.github.com/soaresfabricio/721a19d1c2d390c6e147
    - https://rosettacode.org/wiki/Huffman_coding#C
- Compressão de dados
    - http://www.ic.unicamp.br/~thelma/gradu/MC326/2010/Slides/Aula03b-compressao.pdf
- A codificação de Huffman
    - https://inf.ufes.br/~pdcosta/ensino/2018-2-estruturas-de-dados/material/CodificacaoHuffman.pdf

Tarefa

Sua tarefa é escrever um programa que compacte um arquivo texto utilizando o algoritmo de Huffman. Além disso, o programa deve ser capaz de descompactar um arquivo previamente compactado também pelo algoritmo de Huffman. O programa inicialmente deve receber como entrada um caractere que representa função a ser executada.

As funções são:

- Compactação:

Para a função de compactação o programa deve receber a letra ‘c’ (seguido de ENTER). Após isso, o usuário deve indicar o caminho relativo do arquivo texto a ser compactado. Os arquivos devem ser em formato txt.

A saída do programa deve gerar um arquivo compactado com o mesmo nome do arquivo de entrada porém

com a extensão ‘huf’.

Exemplos:

Entrada	**Saída**
c teste/texto01.txt	Irá gerar um arquivo chamado texto01.huf
c teste/texto02.txt	Irá gerar um arquivo chamado texto02.huf

- Descompactação:

Nesta funcionalidade o programa deve realizar o processo inverso que é a partir da entrada de um arquivo compactado o programa deve gerar um arquivo texto com a formatação do texto original. Para executar essa função o programa deve receber como entrada a letra ‘d’ (seguido de ENTER). Após isso, o usuário deve indicar o caminho relativo do arquivo texto a ser descompactado. Os arquivos devem ser em formato huf.

A saída do programa deve gerar o arquivo descompactado com o formato txt e seu conteúdo original.

Exemplos:

Entrada	**Saída**
d teste/texto01.huf	Irá gerar um arquivo chamado texto01.txt
c teste/texto02.huf	Irá gerar um arquivo chamado texto02.txt

Para suporte ao desenvolvimento do programa considere os códigos e informações abaixo:

- Implementação de árvore binária com funções, de inserção, busca e impressão (pré-ordem) implementadas com o nasm:
  - https://github.com/leandrojsa/assembly/tree/master/x86/binary_search_tree
- Leitura e escriata em arquivo com nasm:
  - https://github.com/leandrojsa/assembly/tree/master/x86/read_and_write_files