Bacharelado em Ciência da Computação

ANÁLISE DE ALGORITMOS

Compactador de Arquivo[1]

1. Motivação

Os compactadores de arquivos são softwares especializados em gerar uma representação mais eficiente de vários arquivos dentro de um único arquivo de modo que ocupem menos espaço na mídia de armazenamento ou o tempo de transferência deles sobre uma rede seja reduzido.

Os compactadores foram muito utilizados no passado quando as mídias de armazenamento tinham preços elevados e era necessário economizar espaço para armazenamento. Atualmente o uso deles é mais voltado a transferência de arquivos pela Internet para reduzir a massa de dados a ser transferida pela rede.

Os compactadores de arquivo utilizam algoritmos de compressão de dados sem perdas para gerar a representação mais eficiente combinando diversas técnicas conhecidas para um melhor desempenho. Uma das técnicas usadas por estes algoritmos é reduzir a redundância de sequências de bits recorrentes contidas nos arquivos gerando uma representação que utiliza menos bits para representar estas sequências, um exemplo desse é a **Codificação de Huffman**. O código de *Huffman* é técnica muito utilizada e bastante eficiente para compressão de dados com economia de 20% a 90%, dependendo do arquivo a ser comprimido.

2. Algoritmo de *Huffman*

O algoritmo de *Huffman* é um **algoritmo guloso** que usa uma tabela de **tabela de frequência de caracteres** para obter um código binário único (**código préfixo**) de cada caractere encontrado no arquivo de entrada. A partir da **tabela de frequência** é gerada uma **lista de frequência de caracteres** em ordem crescente das frequências dos caracteres. O algoritmo recebe como entrada a **lista de frequência** e constrói uma **árvore binária** onde cada nó da tem sempre dois ou nenhum filho. Para gerar os códigos prefixos dos caracteres basta fazer um percurso na árvore.

```
Algoritmo de Huffman
Entrada: Uma lista L de caracteres e suas frequências em ordem
         crescente de freqüência.
Saida: raiz da árvore de Huffman
Inicio
  n = |L|
  para i = 1 até n-1 faça
      CriaNo(z)
      z.esq  = ExtraiMinimo( L )
      z.dir  = ExtraiMinimo( L )
      z.freq = z.esq.freq + z.dir.freq
      InsereCrescente ( L , z )
  fim-para
  retorne ExtraiMinimo( L )
Fim
```

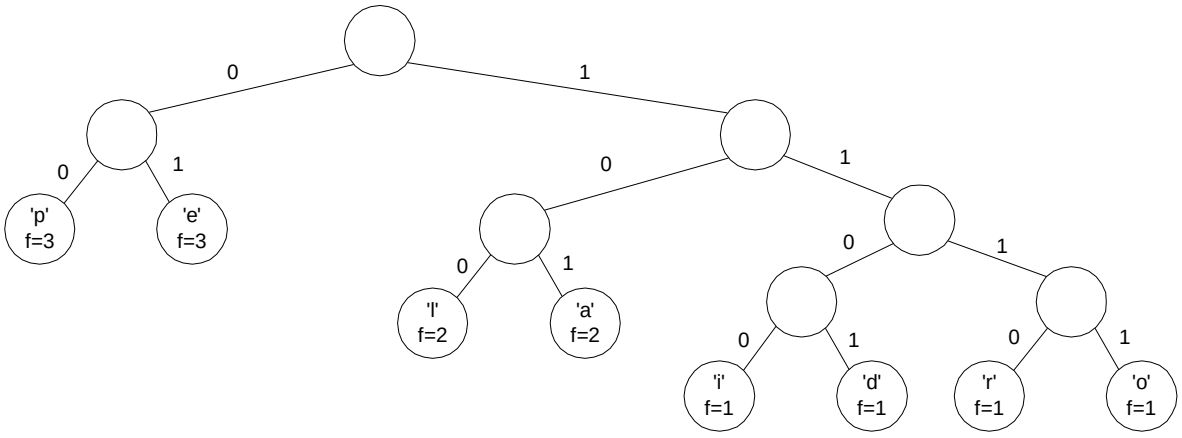
[1] **Importante**: A especificação desse trabalho pode sofrer modificações de acordo com discussões que tivermos em sala de aula.

O algoritmo de *Huffman* tem complexidade $O(n^2)$ pois cada operação de inserir na lista em ordem crescente pode consumir tempo $O(n)$, onde n é o tamanho da lista, e como o algoritmo repete de 1 até n–1 vezes e a cada iteração temos operações de inserir na lista podemos dizer que a complexidade total do algoritmo é $O(n^2)$. Se a lista fosse implementada como um Heap a inserção poderia ser feita em $O(logn)$, com isso o tempo total seria $O(nlogn)$

2. Objetivo

O objetivo deste trabalho é ter como entrada um arquivo texto com caracteres ASCII e a partir do arquivo construir uma tabela de frequência de caracteres e por fim a geração do código prefixo a partir da árvore de *Huffman*. O seu programa tem como resultado uma codificação binária do arquivo de entrada.

Suponha que o arquivo de entrada tenha a sentença `paralelepipedo` para codificação. Essa sequência contém 8 frequências para os diferentes caracteres encontrados, isso significa que o texto a ser decodificado contém 2 caracteres ‘`a`’, 1 caractere ‘`d`’, 3 caracteres ‘`e`’, 1 caractere ‘`i`’, 2 caracteres ‘`l`’, 1 caractere ‘`o`’, 3 caracteres ‘`p`’ e 1 caracter ‘`r`’. Com esse conjunto de caracteres e frequências poderemos construir a seguinte *árvore binária de prefixos*.




```
a 101
d 1101
e 01
i 1100
l 100
o 1111
p 00
r 1110
```

Após construir a *árvore Huffman* associa-se os caracteres do conjunto a sua *string binária*, isso é feito, realizando um percurso, para cada um dos caracteres, a partir da raiz da árvore até a folha que o caracter representa. Para converter a sentença do arquivo de entrada em uma seqüência de ‘0’ e ‘1’ basta substituir os seus caracteres pela *string binária* correspondente, na mesma ordem que aparece na entrada, como mostra a seguir:

Sentença entrada: `paralelepipedo`

Sentença codificada: `00101111010110001100010011000001110111111`

3. Entrada

A entrada será um arquivo texto **não compactado**, o seu programa fará a leitura do arquivo e construirá a partir do arquivo texto um tabela de frequência de caracteres e gera como saída um arquivo **compactado**. Depois faça o contrário, ou seja, que receba como entrada um arquivo **compactado** e gera a reprodução do arquivo original **não compactado** como saída. Os arquivos de entrada e saída deverão ser informados no início do programa.

4. Saída

O arquivo de saída deverá ter o seguinte formato, um inteiro indicando a quantidade de caracteres distintos no arquivo de entrada, a lista de frequências para esses caracteres, em ordem crescente de caracteres, e os caracteres do arquivo de entrada convertidos para suas *string binárias*.

```
8
2 a 1 d 3 e 1 i 2 l 1 o 3 p 1 r
00101111010110001100010011000001110111111
```

5. Desafio:

A partir do entendimento da codificação tente gerar a decodificação, ou então, tente, em vez de gravar uma seqüência de ‘0’ e ‘1’ grave a codificação em bits.

5. Restrições do projeto

O trabalho entregue **será avaliado** de acordo com os seguintes itens:

- Funcionamento do programa;
- O programa deve estar na linguagem **C**
- O quão fiel é o programa quanto à descrição do enunciado;
- Identação, comentários e legibilidade do código;
- Clareza na nomenclatura de variáveis e funções.

Grupo

A atividade pode ser feita em grupo de no máximo **dois integrantes**, para entregar basta que somente um dos integrantes submeta o trabalho no Moodle com o nome dos integrantes do grupo no arquivo fonte.

Importante

Como este trabalho pode ser feito em grupo, evidentemente você pode “discutir”" o problema dado com outros grupos, inclusive as “dicas”" para chegar às soluções, mas você deve ser responsável pela solução final e pelo desenvolvimento do seu programa. Assim não repasse para e nem copie o programa de outro grupo. Trabalhos considerados plágios receberão nota 0 (zero).