# Pesquisa e Classificação de Dados - Trabalho da disciplina

Prof. Ricardo Oliveira - 2020/1

### Enunciado

O trabalho consiste na implementação, na linguagem C, de um compactador e um descompactador de arquivos de texto usando a codificação de Huffman.

## Compactador

O compactador deve receber dois parâmetros por linha de comando, sendo o primeiro o arquivo de entrada (a ser compactado), e o segundo o arquivo de saída (a ser gerado).

Você pode assumir que o arquivo de entrada será sempre um arquivo de texto contedo apenas letras mínusculas (de 'a' a 'z'), espaços (' ') e fim-de-linhas ('\n').

O programa deve imprimir, na saida padrão (para o usuário), a tabela de frequências do arquivo de entrada, além do código de Huffman obtido.

Como exemplo, considere a seguinte execução. Ela compacta o arquivo faroeste.txt gerando o arquivo compactado faroeste.hzip, e imprime a tabela de frequências e o código em si ao usuário:

#### \$ compacta faroeste.txt faroeste.hzip

a:	741	u: 265	k: 01101100101
b:	65	v: 91	1: 111111
c:	214	w: 3	m: 10100
d:	233	x: 5	n: 11101
e:	663	y: 3	o: 000
f:	55	z: 35	p: 111000
g:	42	: 1067	q: 1110010
h:	75	\n: 169	r: 0111
i:	353	EOF: 1	s: 0100
j:	29		t: 10110
k:	2	a: 100	u: 11110
1:	154	b: 1110011	v: 011010
m:	207	c: 10101	w: 0110110000
n:	265	d: 10111	x: 0110110011
0:	583	e: 001	y: 0110110001
p:	120	f: 0110111	z: 11111010
q:	62	g: 11111011	: 110
r:	377	h: 1111100	\n: 01100
s:	315	i: 0101	EOF: 01101100100
t:	221	j: 01101101	

Note que, embora a tabela de frequência é exatamente a dada para o arquivo faroeste.txt, o código obtido pode ser diferente do dado acima, dependendo da ordem da rotulação da árvore e de critérios de desempate. Ainda, um simbolo especial "EOF" deve ser utilizado para marcar o final do arquivo de entrada, e a frequência deste símbolo deve sempre ser igual a 1 (um). Há, ao todo, 29 símbolos no alfabeto.

### Descompactador

O descompactador deve receber dois parâmetros por linha de comando, sendo o primeiro o arquivo de entrada (a ser descompactado), e o segundo o arquivo de saída (a ser gerado).

O programa deve imprimir, na saida padrão (para o usuário), o código de Huffman utilizado para a descompactação (que faz parte do arquivo compactado, conforme a seção seguinte). Como exemplo, considere a seguinte execução. Ela descompacta o arquivo faroeste.hzip gerando o arquivo descompactado faroeste.orig.txt, e imprime o código em si ao usuário:

#### \$ descompacta faroeste.hzip faroeste.orig.txt

a:	100	k:	01101100101	u:	11110
b:	1110011	1:	111111	v:	011010
c:	10101	m:	10100	w:	0110110000
d:	10111	n:	11101	x:	0110110011
e:	001	0:	000	у:	0110110001
f:	0110111	p:	111000	z:	11111010
g:	11111011	q:	1110010	:	110
h:	1111100	r:	0111	\n	: 01100
i:	0101	s:	0100	EO	F: 01101100100
j:	01101101	t:	10110		

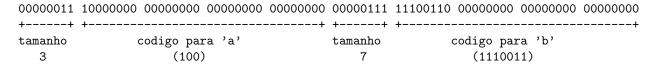
Note que o arquivo gerado pelo descompactador (no exemplo, faroeste.orig.txt) deve ser exatamente igual ao arquivo dado inicialmente ao compactador (no exemplo, faroeste.txt).

### Formato do arquivo compactado

O arquivo compactado deve ser um arquivo binário ( $n\tilde{a}o$  texto). O arquivo inicia com um cabeçalho contendo o código utilizado pela codificação, seguido pelo arquivo compactado em si.

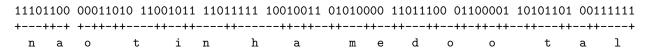
O cabeçalho tem exatamente 145 bytes, composto por 29 blocos de 5 bytes cada. O primeiro bloco indica o código de 'a'; o segundo bloco indica o código de 'b'; e assim por diante. O antepenultímo, o penúltimo e o último bloco indicam o código do espaço, fim-de-linha e do símbolo especial EOF, respectivamente.

O primeiro byte de cada bloco contém o tamanho do código de seu símbolo correspondente. Os próximos 4 bytes contém o código em si (preenchido com zeros à direita, se necessário). Como exemplo, estes são os primeiros 10 bytes do arquivo compactado gerado no exemplo, indicando os dois primeiros blocos (para 'a' e 'b'):



Em hexa, os primeiros bytes do arquivo compactado são  $03\ 80\ 00\ 00\ 00\ 07\ E6\ 00\ 00\ 00$ .

Após o cabeçalho, o arquivo deve conter a sequência de bytes que representa o arquivo compactado em si. Considerando o exemplo dado, os primeiros bytes são:



Em hexa, os primeiros bytes após o cabeçalho do arquivo compactado são EC 1A CB DF 93 50 DC 61 AD 3F.

Caso a sequência de bits não tenha tamanho múltimo de 8, o último byte deve ser preenchido com zeros à direita conforme necessário. Como exemplo, dado o código acima, mas considerando que o arquivo de texto original contém apenas a<sup>1</sup>, os bytes após o cabeçalho do arquivo compactado são:

Em hexa, temos os bytes 4D 90 neste caso.

## Orientações

- O trabalho pode ser feito por equipes de até 2 (dois) estudantes;
- Submeta, via *Moodle*, um pacote (zip ou tar.gz) contendo todo o código fonte do trabalho, além de um arquivo de texto (txt) onde conste:
  - O nome de todos os integrantes da equipe;
  - Toda informação que a equipe julgar relevante para a correção (como bugs conhecidos, detalhes de implementação, escolhas de projeto, etc.)
- Comente adequadamente seus códigos para facilitar a correção.
- Atenção: a correção será parcialmente automatizada, e a saída do programa será testada com outras entradas além das fornecidas como exemplo. Siga fielmente o formato de saída dado nos exemplos, sob pena de grande redução da nota;
- Certifique-se que seu programa funciona antes de submetê-lo;
- A nota será incrementada com 10 **pontos extras** (o trabalho pode valer 110 ao todo) caso uma *minheap* seja implementada e utilizada corretamente no compactador;
- O trabalho deve ser entregue até **23 de Outubro**, **23:59**, via *Moodle*. É suficiente que o trabalho seja submetido por apenas um estudante da equipe;
- Trabalhos copiados ou plagiados receberão todos a nota 0 (ZERO).

#### Dicas

- Você pode ver em hexa qualquer arquivo, incluindo binários, com ferramentas do tipo *HexDump*, como a disponível online em https://www.fileformat.info/tool/hexdump.htm;
- Parâmetros por linha de comando são lidos, em C, através dos argumentos argc e argv da função main, e são passados por prompts de comando do sistema operacional. Leia http://linguagemc.com.br/argumentos-em-linha-de-comando/;
- Arquivos de texto são lidos e escritos com as funções fscanf e fprintf. Arquivos binários são lidos e escritos com as funções fread e fwrite;
- Para evitar problemas com big e little-endian, prefira sempre ler e escrever exatamente um byte "por vez" ao chamar as funções fread e fwrite. O trecho abaixo cria um arquivo e escreve nele o byte ED:

```
FILE *arquivo = fopen("arquivo.hzip", "w");
unsigned char c = 0xED;
fwrite(&c, sizeof(unsigned char), 1, arquivo);
fclose(arquivo);
```

<sup>&</sup>lt;sup>1</sup>Isto é apenas uma exemplo. Este arquivo não geraria o código dado, uma vez que a tabela de frequência seria outra.