SCC-502 Algoritmos e Estruturas de Dados 1 **M. Cristina/Jorge**

Trabalho Rec – Codificação de Huffman

O trabalho deverá ser feito individualmente e submetido para o sistema SSP (Sistema de Submissão de Programas), do ICMC.

Data de entrega: 02 de fevereiro de 2014.

Para entrar no sistema e submeter o trabalho:
1. Entre no site <https://ssp.icmc.usp.br/>;
2. Faça login/cadastro no site;
3. Matricule-se na disciplina SCC0502 - Algoritmos e Estruturas de dados (M. Cristina) (2013/2-Turma A)
4. Clique em Submeter exercicio -> "Trabalho Rec. Codificação de Huffman";
5. Submeta seu exercício;
6. O sistema fará a correção automática do seu programa, seguindo alguns casos de teste(padrões de entrada e saída) e apresentará o número de acertos e erros. Você pode submeter o programa quantas vezes desejar (até o prazo de entrega).

Critério de Avaliação

Nota Trabalho Rec = 0,2 * Código (Identação, Modularização e Comentários) + 0,8 * Nº de casos corretos (SSP)

Descrição do Trabalho

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 foram 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.

Para fazer a codificação, utilize o seguinte algoritmo:

```
Leia uma frase
Calcule a probabilidade de cada caractere na frase e a armazene em uma lista
ordenada por código ascii de [caracteres + probabilidades]

//Construcao da arvore
enquanto tamanho(alfabeto) > 1:
  S0 := retira_menor_probabilidade(alfabeto)
  S1 := retira_menor_probabilidade(alfabeto)
  X  := novo_nó
  X.filho0 := S0
  X.filho1 := S1
  X.probabilidade := S0.probabilidade + S1.probabilidade
  insereNoComeco(alfabeto, X)
fim enquanto

X = retira_menor_símbolo(alfabeto) # nesse ponto só existe um símbolo.

para cada folha em folhas(X):
  código[folha] := percorre_da_raiz_até_a_folha(folha)
  A cada aresta da árvore é associado um dos dígitos binários (0- esquerdo ou 1-
direito)
fim para

Represente cada caractere da frase com o código da tabela X.
```

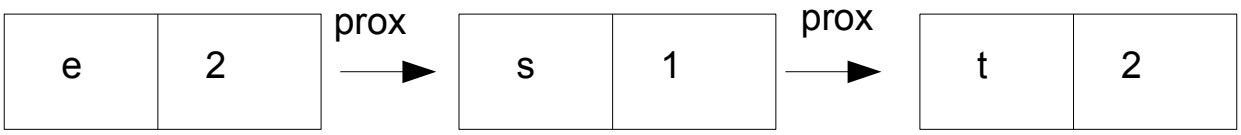
Seu programa deve:

a) Construir a árvore de codificação de huffman;
b) Apresentar a tabela de símbolos;
c) Codificar frases;
d) Cumprir as especificações de **Entrada** e **Saída**, descritas abaixo.

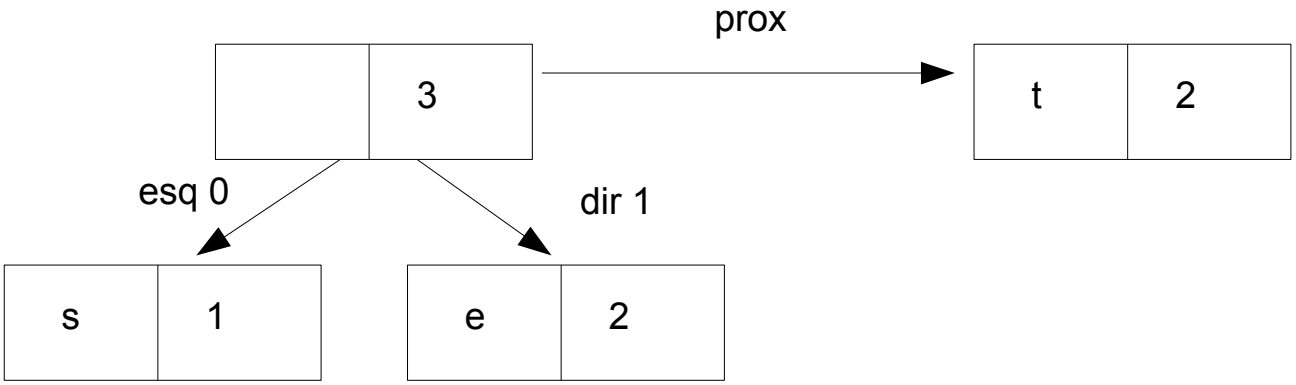
Exemplos de Codificação

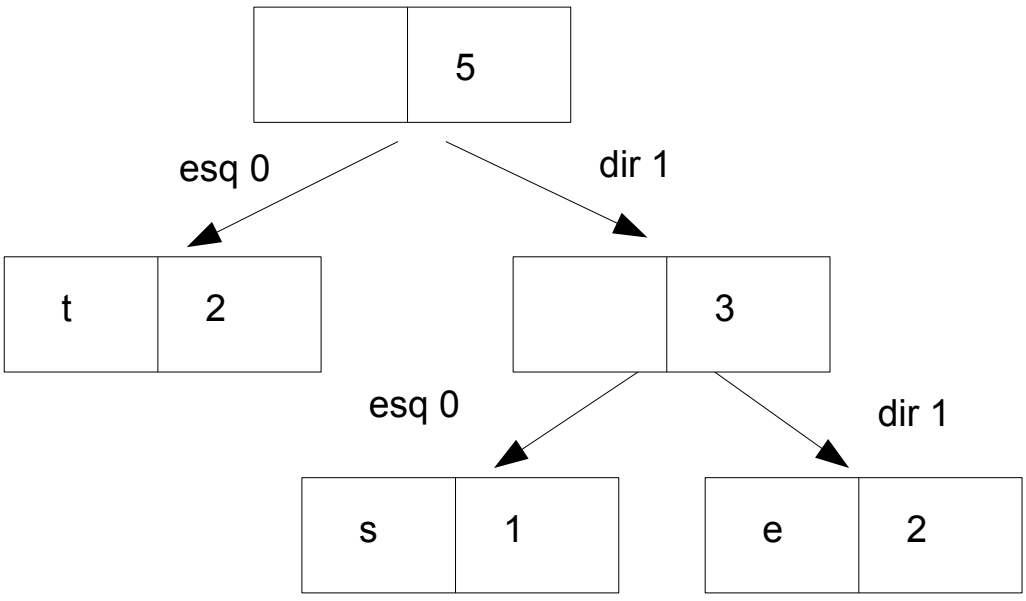
1) Considere a palavra “teste”

Letras + probabilidades (probabilidade = número de vezes que o caractere aparece)




```
enquanto tamanho(alfabeto) > 1:
  S0 := retira_menor_probabilidade(alfabeto)
  S1 := retira_menor_probabilidade(alfabeto)
  X  := novo_nó
  X.filho0 := S0
  X.filho1 := S1
  X.probabilidade := S0.probabilidade + S1.probabilidade
  insereNoComeco(alfabeto, X)
```





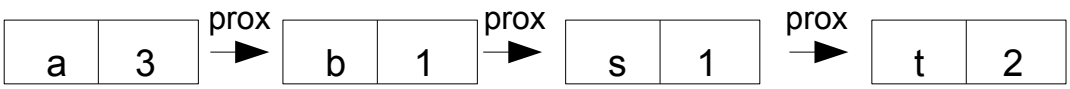
Neste exemplo, a tabela de símbolos que associa o caractere ao seu código será:

“t”	0
“s”	10
“e”	11

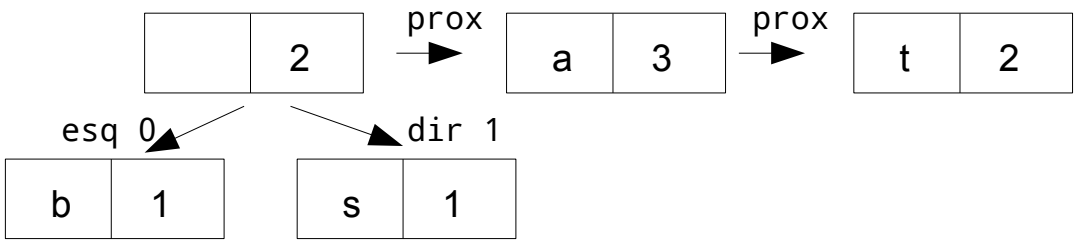
E a palavra codificada será:
01110011

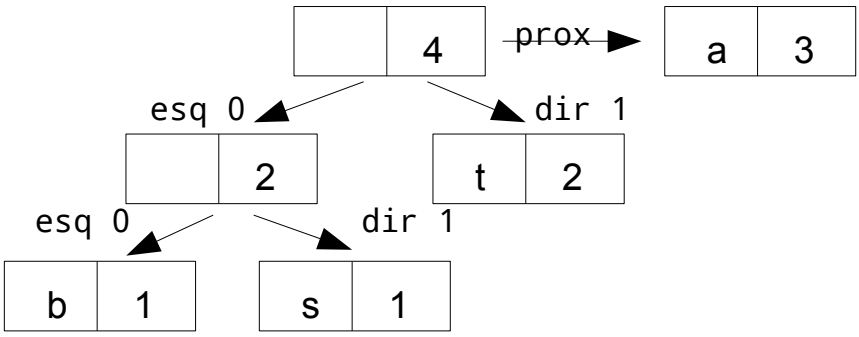
2) Considere a palavra “batatas”

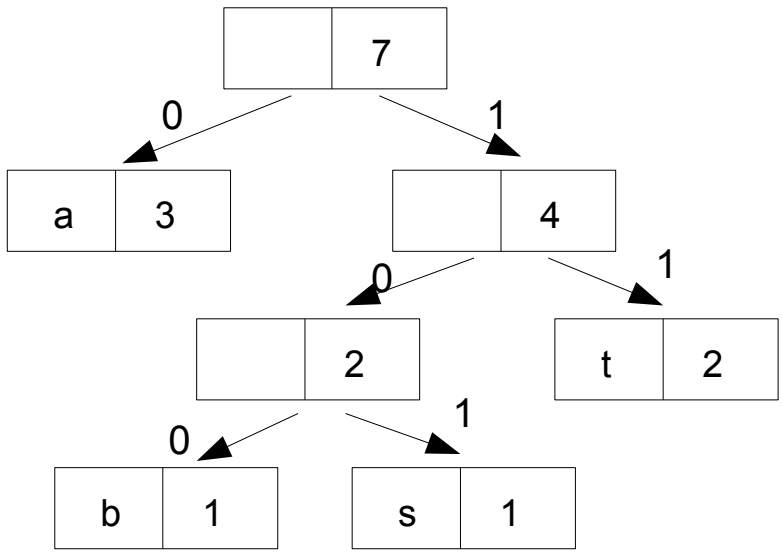
Letras + probabilidades




```
//Construcao da arvore
```







A tabela de símbolos é:

“a”	0
“b”	100
“s”	101
“t”	11

E a palavra codificada:

1000110110101

Entrada

A entrada deve ser lida da entrada padrão (stdin)

A entrada contém vários casos de teste.

Cada caso de teste contém um comando: mostrar a tabela (**t**), mostrar a codificação (**c**) e sair (**s**).

1) Mostrar a tabela (t)

```
t <frase>
```

```
<frase> - Conjunto de caracteres: “A”-”Z”, “a”-”z”, ”0”-”9”,“_”, “.”, “!”, “?”
```

exemplo:

```
t teste_de_frase_teste_teste!!!
```

Este comando deve imprimir a tabela de símbolos da frase digitada.

2) Mostrar a codificação (c)

```
c <frase>
```

```
<frase> - Conjunto de caracteres: “A”-”Z”, “a”-”z”, ”0”-”9”, “_”, “.”, “!”, “?”
```

exemplo:

```
c teste_de_frase_teste_teste!!!
```

Este comando deve imprimir a codificação binária da frase digitada.

3) Sair (s)

Este comando deve sair do programa (retornar 0).

Saída

A saída deve ser apresentada na saída padrão, **stdout**

A saída deve corresponder ao comando na entrada:

1) Mostrar a tabela (t)

Mostra a tabela, na forma:

`“<caractere>” <código><\n>`

Exemplo:

`<entrada>`

`t batatas`

<saída>

"a" 0

"b" 100

"s" 101

"t" 11

2) Mostrar a codificação (c)

Mostra o código, na forma

`<código><\n>`

Exemplo:

`<entrada>`

`t batatas`

<saída>

1000110110101

3) Sair (s)

Não imprime nada na tela e retorna 0.

Exemplos de execução

Entrada	Saída
t testes c testes s	"t" 0 "e" 10 "s" 11 0101101011
t algoritmos_e_estruturas_de_dados_1!!! c algoritmos_e_estruturas_de_dados_1!!! s	"m" 0000 "i" 00010 "l" 00011 "!" 001 "1" 01000 "g" 01001 "u" 0101 "s" 011 "_" 100 "r" 1010 "t" 1011 "e" 1100 "o" 1101 "a" 1110 "d" 1111 11100001101001110110100001010110000011010111001100100110001110110100101101101011010111001110011111110010011111111011111101011100010001001001
t teste_de_palavras_abacate_banana_abacaxi? c teste_de_palavras_abacate_banana_abacaxi? t fruta_do_conde c fruta_do_conde s	"i" 00000 "l" 00001 "?" 00010 "d" 00011 "v" 00100 "x" 00101 "p" 00110 "r" 00111 "e" 010 "s" 0110 "b" 0111 "_" 100 "t" 1010 "c" 10110 "n" 10111 "a" 11 10100100110101001010000011010100000110110000011100100000111110110100110111111011011101001010001111110111111011111110011011111101101001010000000010

	"o" 00 "_" 010 "d" 011 "e" 1000 "f" 1001 "a" 1010 "c" 1011 "t" 1100 "u" 1101 "n" 1110 "r" 1111 10011111110111001010010011000101011001110011100
t abacate t abacaxi c abacate c abacaxi s	"a" 0 "e" 100 "t" 101 "b" 110 "c" 111 "a" 0 "i" 100 "x" 101 "b" 110 "c" 111 011001110101100 011001110101100