Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Ciência da Computação / Engenharia de Computação
Redes Multimídia (TEC II / TEC IV) – Prof. Guilherme Corrêa

Exercício Prático 1: Codificação de Huffman

Instruções

- O exercício pode ser realizado em **duplas**;
- Implementar em C, C++ ou Java;
- A submissão do exercício no AVA está prevista para o dia **17/05/2017**;
- A execução deste trabalho corresponde a **25%** na componente *Avaliação Contínua*;
- A nota inclui avaliação do **andamento** do exercício (e não apenas o resultado final);
- Os alunos podem consultar outros materiais, mas cópia/plágio acarreta em nota **zero**.

Objetivos

- O objetivo do trabalho é prover compressão de um arquivo de entrada com código de comprimento variável;
- O aluno deverá criar pelo menos dois códigos-fonte separados: (1) comprimir, (2) descomprimir;
- Alternativamente, poderá criar códigos-fonte separadamente para: (1) calcular probabilidade de ocorrência dos caracteres, (2) gerar árvores de Huffman, (3) comprimir, (4) descomprimir.

Passos para Compressão

1. Receber como entrada um texto codificado em ASCII (considere apenas caracteres de A até Z maiúsculos e espaços);
2. Calcular a probabilidade de ocorrência dos caracteres;
3. Gerar uma árvore binária de acordo com o algoritmo de Huffman;
4. Gerar uma tabela com a probabilidade de ocorrência de cada caractere;
5. Salvar em um arquivo a probabilidade de ocorrência de cada caractere;
6. Codificar a entrada com os códigos de comprimento variável gerados na árvore de Huffman;
7. Armazenar o resultado (texto comprimido) em um arquivo (binário).
8. Para fins comparação, armazenem também o texto original em um arquivo (binário).

Descompressão

1. Receber uma entrada comprimida e o arquivo com a probabilidade de ocorrência de cada caractere;
2. Gerar uma árvore de Huffman de acordo com as probabilidades;
3. Decodificar a entrada com os códigos de comprimento variável gerados na árvore de Huffman;
4. Armazenar o resultado decodificado em um arquivo ASCII.

Algoritmos

Algoritmo para gerar a árvore:

```
Entrada: Fila de prioridades P com caractere e sua probabilidade
         (pode ser um vetor ordenado)
Saída: Árvore binária com códigos nas folhas, raiz em No

while (|P| > 1) do  //enquanto houver mais que um elemento em P
begin
   Sdir := retira caractere com menor probabilidade de P
   Sesq := retira caractere com menor probabilidade de P
   No := novo nodo
   No.direita := Sdir
   No.esquerda := Sesq
   No.probabilidade := P(Sdir) + P(Sesq) // P(x) retorna probabilidade de
                                         // ocorrência do caractere
   insere (P, No) //cuidado, manter a fila de prioridades P ordenada
end
No := retira(P) //raiz da árvore
```

Algoritmo para gerar tabela de códigos dos caracteres:

```
Entrada: árvore com codificação com raiz em Raiz,
         Comp e Cod são dois vetores com uma posição para cada caractere,
         Pbits é pilha de bits (inicialmente vazia)
Saida: vetor Comp com comprimento em bits de cada código, vetor Cod com os
bits propriamente ditos

Code (Raiz, Comp, Cod)
begin
  if (Raiz is not nill) then
  begin
   push(Pbits,0)
   Code(Raiz.esquerda, Comp, Cod, Pbits)
   pop(Pbits)
   push(Pbits,1)
   Code(Raiz.direita, Comp, Cod, Pbits)
   pop(Pbits)
  end
  else //chegou na folha
  begin
    pilha_codigo := Pbits //copia a pilha inteira
```

```
    codigo := 0
    comprimento := 0
    while (not empty(pilha_codigo)) do
    begin
      codigo := codigo OR top(pilha_codigo) //ou bit-a-bit
      pop(pilha_codigo)
      comprimento := comprimento + 1
    end
    Comp[Raiz.caractere] := comprimento
    Cod[Raiz.caractere] := codigo
  end
end
```