**Programa 5.43** *Procedimento para extrair palavras de um texto*

```
program ExtraiPalavra (arqTxt, arqAlf);
const MAXALFABETO = 255;
type TipoAlfabeto = array [0..MAXALFABETO] of boolean;
var ArqTxt, ArqAlf: text;
    Alfabeto      : TipoAlfabeto;
    Palavra, Linha: string[255];
    i             : integer;
    aux           : boolean;

  procedure DefineAlfabeto (var Alfabeto: TipoAlfabeto);
  var Simbolos: string[MAXALFABETO]; i: integer;
  begin { Os simbolos devem estar juntos em um linha no arquivo }
    for i := 0 to MAXALFABETO do Alfabeto[i] := false;
    readln (ArqAlf, Simbolos);
    for i := 1 to length(Simbolos) do Alfabeto[ord(Simbolos[i])]:=true;
    Alfabeto[0] := false; { caractere de codigo zero: separador }
  end;

begin
  reset (ArqTxt); reset (ArqAlf);
  DefineAlfabeto (Alfabeto); { Le alfabeto definido em arquivo }
  aux := false;
  while not eof (ArqTxt) do
    begin
    readln (ArqTxt, Linha);
    Linha := Linha + char (0); { Coloca separador no final da linha }
    for i := 1 to length(Linha) do
      begin
      if Alfabeto[ord (Linha[i])]
      then begin Palavra := Palavra + Linha[i]; aux := true; end
      else if aux
           then begin
                writeln (Palavra); { Palavra extraida }
                Palavra := ''; aux := false;
                end;
      end;
    end;
end.
```

Observe que, apesar de o *hashing* ser mais eficiente do que árvores de pesquisa, existe uma desvantagem na sua utilização: após atualizado todo o índice remissivo, é necessário imprimir suas palavras em ordem alfabética. Isso é imediato em árvores de pesquisa, mas, quando se usa *hashing*, isso é problemático, sendo necessário ordenar a tabela *hash* que contém o índice remissivo.

Utilize o exemplo anterior para testar seu programa. Comece a pensar tão logo seja possível, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.



**20.** Considere duas listas ordenadas de números. Determine se cada elemento da lista menor está presente também na lista maior. (Pode assumir que não existem duplicações em nenhuma das duas listas.) Considere os seguintes casos:

- Uma lista contém apenas um elemento, a outra $n$;
- As duas listas contêm $n$ elementos;
- Uma lista contém $\sqrt{n}$ elementos, a outra $n$.

a) Sugira algoritmos eficientes para resolver o problema;

b) Apresente o número de comparações necessário;

c) Mostre que cada algoritmo minimiza o número de comparações.

**21.** Árvore Patricia.

Desenhe a **árvore Patricia** que resulta da inserção sucessiva das chaves Q U E S T A O F C I L, nesta ordem, em uma árvore inicialmente vazia.

**22.** Árvore Patricia.

a) Desenhe a **árvore Patricia**. que resulta da inserção sucessiva das chaves M U L T I C S, nesta ordem, em uma árvore inicialmente vazia.

b) Qual é o custo para pesquisar em uma árvore Patricia construída com o emprego de $n$ inserções randômicas? Explique.

c) Qual é o custo para construir uma árvore Patricia via $n$ inserções randômicas? Explique.

i) Sob o ponto de vista prático, quando $n$ é muito grande (digamos 100 milhões), qual é a maior dificuldade para construir a árvore Patricia?

ii) Como a dificuldade apontada no item anterior pode ser superada?

**23.** Considere o seguinte trecho do poema "Quadrilha", de **Carlos Drummond de Andrade** (Murta, 1992):

"João amava Teresa que amava Raimundo que amava
Maria que amava Joaquim que amava Lili que não
amava ninguém."

Construa uma **árvore Patricia** para indexar o texto acima. Considere a seguinte codificação para as palavras do texto:

| | | | |
|---|---|---|---|
| João | 01001011 | Maria | 01100101 |
| amava | 00011101 | Joaquim | 00101110 |
| Teresa | 11101011 | Lili | 01010011 |
| que | 10100101 | não | 10011100 |
| Raimundo | 11011010 | ninguém | 10110010 |

a) Faça uma pesquisa pelas chaves "amava", "que amava" e "Lili". Mostre o caminho percorrido para cada pesquisa e as ocorrências do termo pesquisado.