

## Continuação do Programa 5.27

```

procedure Insere (x: TipoItem; var p: TipoPesos;
                 var T: TipoDicionario);
begin
  if Pesquisa (x.Chave, p, T) = nil
  then Ins (x, T[h(x.Chave, p)])
  else writeln (' Registro ja  esta  presente')
end; { Insere }

procedure Retira (x: TipoItem; var p: TipoPesos;
                 var T: TipoDicionario);
var Ap: TipoApontador;
begin
  Ap := Pesquisa (x.Chave, p, T);
  if Ap = nil
  then writeln (' Registro nao esta  presente')
  else Ret (Ap, T[h(x.Chave, p)], x)
end; { Retira }

```

**Análise** Assumindo que qualquer item do conjunto tem igual probabilidade de ser endereçado para qualquer entrada de T, então o comprimento esperado de cada lista encadeada é  $N/M$ , em que  $N$  representa o número de registros na tabela e  $M$  o tamanho da tabela.

Logo, as operações Pesquisa, Insere e Retira custam  $O(1 + N/M)$  operações em média, sendo que a constante 1 representa o tempo para encontrar a entrada na tabela e  $N/M$  o tempo para percorrer a lista. Para valores de  $M$  próximos de  $N$ , o tempo torna-se constante, isto é, independente de  $N$ .

## 5.5.3 Endereçamento Aberto

Quando o número de registros a serem armazenados na tabela puder ser previamente estimado, então não haverá necessidade de usar apontadores para armazenar os registros. Existem vários métodos para armazenar  $N$  registros em uma tabela de tamanho  $M > N$ , os quais utilizam os lugares vazios na própria tabela para resolver as colisões. Tais métodos são chamados **endereçamento aberto** (do inglês **open addressing**; Knuth, 1973, p. 518).

Em outras palavras, todas as chaves são armazenadas na própria tabela, sem o uso de apontadores explícitos. Quando uma chave  $x$  é endereçada para uma entrada da tabela já ocupada, uma sequência de localizações alternativas  $h_1(x), h_2(x), \dots$  é escolhida dentro da tabela. Se nenhuma das  $h_1(x), h_2(x), \dots$  posições está vazia, então a tabela está cheia e não podemos inserir  $x$ .

Existem várias propostas para a escolha de localizações alternativas. A mais simples é chamada de **hashing linear**, na qual a posição  $h_j$  na tabela é dada por:

$$h_j = (h(x) + j) \bmod M, \text{ para } 1 \leq j \leq M - 1.$$

Se a  $i$ -ésima letra do alfabeto é representada pelo número  $i$  e a função de transformação  $h(\text{Chave}) = \text{Chave} \bmod M$  é utilizada para  $M = 7$ , então a Figura 5.15 mostra o resultado da inserção das chaves  $L U N E S$  na tabela, usando **hashing linear** para resolver colisões. Por exemplo,  $h(L) = h(12) = 5$ ,  $h(U) = h(21) = 0$ ,  $h(N) = h(14) = 0$ ,  $h(E) = h(5) = 5$ , e  $h(S) = h(19) = 5$ .

T	
0	U
1	N
2	S
3	
4	
5	L
6	E

Figura 5.15 Endereçamento aberto.

A estrutura de dados **endereçamento aberto** será utilizada para implementar o tipo abstrato de dados Dicionário, com as operações Inicializa, Pesquisa, Insere, Retira. A estrutura do dicionário é apresentada no Programa 5.28. A implementação das operações sobre o Dicionário são mostradas no Programa 5.29.

## Programa 5.28 Estrutura do dicionário usando endereçamento aberto

```

const VAZIO = '!!!!!!!!';
      RETIRADO = '*****';
      M = 7;
      N = 10; { Tamanho da chave }

type TipoApontador = integer;
      TipoChave = packed array [1..N] of char;
      TipoItem = record
                    Chave: TipoChave
                    { outros componentes }
                  end;

      TipoIndice = 0..M - 1;
      TipoDicionario = array [TipoIndice] of TipoItem;
{— Entra aqui TipoPesos do Programa 5.22 ou do Programa 5.24 —}

```