Métodos de Pesquisa

Objetivos e Caracterizações

Para que se possa falar em algoritmos de pesquisa, é necessário inicialmente introduzir a noção de mapeamento que é uma das mais primitivas em programação. Refere-se a uma regra de associação entre os valores de um conjunto (domínio) e os valores de outro (imagem). Escreve-se

$$m: S \rightarrow T$$

para se declarar que m é um mapeamento do conjunto S para o conjunto T. Obtém-se um valor de T aplicando-se m(i), onde $i \in S$. Os vetores e matrizes são os casos típicos. Aí os índices são normalmente objetos simples ou no máximo tuplas homogenias (pares, triplas, etc) de valores simples.

Objetivos e Caracterizações

As estruturas chamadas **tabelas** são a realização da idéia genérica de mapeamento, em que os valores do domínio podem ser quaisquer. A organização das tabelas pode se reduzir aos arranjos, mas para se falar em tabelas, usa-se uma terminologia específica:

tabela: uma coleção de *entradas*;

entrada: um conjunto de *campos*, formando um *registro*, ou *linha*, da tabela;

chave: um campo escolhido para *identificar* a entrada.

Como pode-se perceber uma operação importante é a busca de uma entrada dado o valor da chave.

Objetivos e Caracterizações

A tabela, como mapeamento, poderia ser operada, por exemplo, para uma consulta, da seguinte forma:

E = T[C];

Considerando E: entrada, T:tabela; C:chave.

Diversas estratégias são propostas para implementação da operação de pesquisa, levando em conta aspectos das operações usuárias e da representação física da tabela. Veremos agora dois métodos utilizados para implementação de pesquisa em tabelas, a pesquisa seqüencial e a pesquisa binária.

UNIVASF

Pesquisa Seqüencial

A pesquisa seqüencial é o método mais simples. Consiste na mera varredura serial, entrada por entrada, devolvendo-se o índice da entrada cuja chave for igual à chave fornecida como argumento da pesquisa. Ou devolvendo -1, convencionalmente, caso a chave buscada não seja localizada, tendo-se comparado todas as chaves, até o fim da tabela. O algoritmo a seguir nos mostra uma função que efetua uma busca seqüencial em uma tabela.

```
int pesq (tabela T, chave C)
{
   int i;
   for (i=0; i<T.N; i++)
        if (T.TAB [i].CH == C)
                return i;
   return -1;
}
```

Pesquisa Seqüencial

A complexidade do algoritmo acima é **O**(n). O desempenho da pesquisa seqüencial pode melhorar um pouco se a tabela estiver *ordenad*a em função da chave: pode-se interromper a pesquisa assim que se alcançar uma entrada com chave maior do que a pesquisada, significando ser desnecessário prosseguir até o fim da tabela. O pior caso (busca da última chave) continua **O**(n). No algoritmo a seguir visualizamos um exemplo de uma função que efetua a busca seqüencial em uma tabela ordenada em função da chave.

```
int pesq_seq (tabela T, chave C)
{
   int i;
   for (i=0; i<T.N; i++)
       if (T.TAB [i].CH >= C)
              if (T.TAB [i].CH == C)
              return i;
              else
                     return -1;
   return -1;
}
```

Pesquisa Binária

Sobre a tabela ordenada é possível obter-se um algoritmo bem mais eficiente. A idéia do mesmo pode ser assimilada pelo seguinte exemplo: ao procurar uma palavra no dicionário (que é uma tabela ordenada), começa-se em qualquer ponto do mesmo. Se a página aberta contiver a palavra, a busca terminou; senão, a palavra pode estar antes ou depois dessa página, conforme ela seja lexicograficamente menor ou maior que as palavras dessa página.

UNIVASF

Pesquisa Binária

De modo que novamente se abre o dicionário no setor adequado, anterior ou posterior, repetindo-se este processo até encontrar a palavra. No algoritmo apresentado agora, a primeira comparação de chaves é feita no meio da tabela. Se não for encontrada aí a chave procurada, pode-se abandonar metade da tabela, repetindo o processo com a divisão da outra metade pelo meio, até ser encontrada a chave ou ter-se uma metade constituída de apenas uma entrada, caracterizando-se assim a ausência da chave na tabela.

```
int pesq_bin (tabela T, chave C)  {
    int meio, PRIM, ULT, achou;
    PRIM = 0; ULT = T.N-1, achou = 0;
    while (PRIM <= ULT && !achou) {
        meio = (ULT + PRIM) / 2;
        if (C== T.TAB [meio].CH)
            achou = 1;
        else
            if (C > T.TAB[meio].CH)
                PRIM = meio + 1;    //busca na parte final
            else
                ULT = meio - 1;     //busca na parte inicial
    }
     if (achou)
        return meio;
    else
        return -1;
}
```

Pesquisa Binária

O algoritmo anterior nos mostra uma função que implementa a pesquisa binária de forma iterativa. Nota-se que, a cada comparação, o universo de chaves a comparar é reduzido à metade e que o pior caso é quando a busca prossegue até a subtabela pesquisada ter só um elemento (encontrando-se ou não a chave procurada). Para isso acontecer, o tamanho da tabela vai-ser reduzido de *n* para *n/2, (n/2)/2*, ... até *1*. Logo, a complexidade dessa solução é **O**(log n).

Pesquisa Binária - Exercício

Com base no algoritmo anterior, defina um TAD tabela e codifique um programa em C, o qual deve se utilizar da operação pesq_bin do TAD definido.

Pesquisa Binária

A solução recursiva é natural nesse caso, pois a metade da tabela é também uma tabela passível de operação estritamente similar à tabela inteira. Para o reaproveitamento recursivo da função, informa-se os índices PRIM e ULT, que se referem à primeira e à última entrada do trecho onde se efetua a busca. A invocação inicial deve ser pesq (T, C, 0, T.N-1). Como exercício, construa uma função recursiva que implemente a pesquisa binária.