```
procedimento quicksort (var v: vetor
   [1..100] de inteiro; ind_pri_ele: inteiro;
   ind_ult_ele: inteiro)
var pont_part: inteiro
inicio
   se (ind_ult_ele-ind_pri_ele>0) entao
      pont_part <- particionar(v, ind_pri_ele,
   ind_ult_ele)
      quicksort (v, ind_pri_ele, pont_part-1)
      quicksort (v, pont_part+1, ind_ult_ele)
   fimse
fimprocedimento
```

Classificação por Particionamento - quicksort

A tarefa de escolher o pivô pode ser executada de forma mais eficiente.

A chave ideal para o pivô seria a mediana das chaves, com a qual se teria uma divisão a mais balanceada possível.

Só que para determinar a mediana é preciso ordenar o vetor!

Como a distribuição das chaves é, em princípio, aleatória, qualquer uma tem a mesma chance de ser mediana.

Mas, ao se tomar a primeira, como foi feito anteriormente, corre-se o risco de ser esta a menor (ou a maior) de todas, tornando praticamente inócuo o trabalho na primeira fase de particionamento (pois se teria uma partição sem nenhum elemento e outra com *n-1* elementos).

Uma escolha melhor é a mediana entre a primeira chave, a última e a do meio do vetor.

Exercício 58:

Reescreva os algoritmos anteriores, considerando que o pivô não é mais o primeiro elemento mas, sim a mediana entre o primeiro elemento, o último e o elemento do meio do vetor.

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

$$\mathbf{m}: \mathbf{S} \rightarrow \mathbf{T}$$

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. Onde 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 sequencial e a pesquisa binária.

UNIVASF

Pesquisa Sequencial

A pesquisa sequencial 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 0 (zero), convencionalmente, caso a chave buscada não seja localizada, tendo-se comparado todas as chaves, até o fim da tabela.

O algoritmo (módulo) a seguir nos mostra uma função que efetua uma busca sequencial em uma tabela.

UNIVASF

```
funcao pesq (T: tabela; C: chave): inteiro
var i: inteiro
inicio
   para i de 1 ate T.N faca
      se (T.TAB[i].CH = C) entao
            retorne (i)
      fimse
   fimpara
   retorne (0)
fimfuncao
```

Pesquisa Sequencial

O desempenho da pesquisa sequencial pode melhorar um pouco se a tabela estiver *ordenada* em função da chave: pode-se interromper a pesquisa assim que se alcançar uma entrada com chave maior do que a pesquisada (no caso de uma ordenação crescente), significando ser desnecessário prosseguir até o fim da tabela.

O pior caso (busca da última chave) continua gerando uma varredura total da tabela.

Exercício: Construa um módulo que efetue uma busca sequencial em uma tabela ordenada de forma crescente em função da chave.

UNIVASF

```
funcao pesq_seq (T: tabela; C: chave): inteiro
var i: inteiro
inicio
   para i de 1 ate T.N faca
      se (T.TAB [i].CH >= C) entao
         se (T.TAB [i].CH = C) entao
            retorne (i)
         senao
            retorne (0)
         fimse
      fimse
   fimpara
   retorne (0)
fimfuncao
```

Pesquisa Binária

Sobre a tabela ordenada é possível se obter um algoritmo bem mais eficiente.

A ideia 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.

Exercício: No algoritmo que elaboraremos 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.

```
funcao pesq_bin (T: tabela; C: chave): inteiro
var meio, PRIM, ULT: inteiro
      achou: logico
inicio
   PRIM <- 1
   ULT <- T.N
   achou <- falso
   enquanto (PRIM<=ULT e nao achou) faca
      meio <- (PRIM+ULT)\2
      se (C= T.TAB[meio].CH) entao
         achou <- verdadeiro
      senao
         se (C>T.TAB[meio].CH) entao
            PRIM <- meio+1          //busca na parte final
         senao
            ULT <- meio-1           //busca na parte inicial
         fimse
      fimse
   fimenquanto
   se (achou) entao
      retorne (meio)
   senao
      retorne (0)
   fimse
fimfuncao
```

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).

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 a reformulação recursiva 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_bin_r (T, C, 1, T.N).

UNIVASF

Pesquisa Binária

Exercício:

Como exercício, construa um módulo recursivo que implemente a pesquisa binária.

UNIVASF

```
funcao pesq_bin_r (T:tabela; C: chave; PRIM: inteiro; ULT:
    inteiro): inteiro
var meio: inteiro
inicio
   meio <- (PRIM+ULT)\2
   se (C=T.TAB [meio].CH) entao
         retorne (meio)
   senao
      se (PRIM = ULT) entao
         retorne (0)
      senao
         se (C > T.TAB[meio].CH) entao
            retorne pesq_bin_r(T, C, meio+1, ULT)
         senao
            retorne pesq_bin_r(T, C, PRIM, meio-1)
         fimse
      fimse
   fimse
fimfuncao
```

Pesquisa Binária

Exercício:

Com base no que foi estudado, defina um novo tipo de dado denominado tabela e construa um algoritmo que manipule adequadamente uma variável do tipo tabela. Dentre as possibilidades de manipulação deve ser possível efetuar uma busca binária sobre a tabela à procura de uma determinada entrada.

Dica: Utilizar como base o exercício 43.

UNIVASF