LP – Linguagem de Programação I

Pesquisa Sequencial e Binária

Pesquisa Sequencial

Uma das operações mais frequentes de manipulação de dados é a pesquisa. Atualmente, e graças aos computadores, é possível fazer pesquisas rápidas de informação, pesquisas essas que outrora eram complexas ou demoradas de fazer.

A pesquisa sequencial (ou linear) é muitas vezes o único método de pesquisa disponível quando os dados a procurar não se encontram ordenados. Pesquisar sequencialmente um conjunto de dados corresponde a comparar um termo de pesquisa com cada um dos elementos do conjunto de dados, até se encontrar uma correspondência entre o termo de pesquisa e um dos dados do conjunto.

A principal desvantagem da pesquisa sequencial é que esta obriga a percorrer todo o conjunto de dados para se encontrar todas as ocorrências do termo de pesquisa (por exemplo, todas as pessoas cujo primeiro nome é Carlos). Estes dados podem estar armazenados em tabelas ou em arquivos. Numa pesquisa sequencial é necessário percorrer todo o conjunto de dados para verificar que o elemento a pesquisar não existe.

Vejamos um caso concreto: imaginemos que queremos procurar um registro numa tabela desordenada de registros (*array* de *structs*), que tenha o campo nome igual a um determinado nome introduzido pelo usuário:

```
typedef struct {
// definição da estrutura de cada ficha na tabela
  int NumBI;
  char Nome[150];
// pode ter outros campos, não relevante para o exemplo
} TBI;

TBI Tab[10000]; // tabela tem 10000 fichas do tipo tBI

// Função que devolve a posição, na tabela, do registro que contém o nome
igual ao nome na string de pesquisa NomePesquisar. Procura desde //a posição
Inicio da tabela até à Posição fim. Devolve -1 se não //encontrar nenhum
registro.

int ProcuraNome(struct TBI *T, int Inicio, int Fim, char *NomePesquisar)
{
   // Um ciclo desde Inicio até Fim
   for (int i=Inicio ; i<= Fim; i++) {
      // se encontrou a ficha, então
      if ( strcmpi(NomePesquisar, T[i].Nome) == 0)
         return i; // devolve indíce!
    }
   // não encontrou nenhum registro com o mesmo nome, devolve -1!
   return -1;
} // fim da função

//  strcmpi(S1,S2); compara S1 com S2 ignorando se as letras estão em
//maiúsculas ou minúsculas.
```

Como foi referido anteriormente a desvantagem da pesquisa sequencial reside no fato de que em última análise é necessário percorrer toda a tabela para nos certificarmos de que o valor que andamos à procura não existe.

A pesquisa do exemplo anterior podia ser feita em alternativa através do número do BI ou por qualquer outro dado que pertença à estrutura de dados. O exemplo seguinte ilustra uma pesquisa sequencial pelo número do BI:

```
// Função que devolve a posição da ficha com número de BI igual a NumBI
// ou -1 caso não exista uma ficha com esse número de BI

int ProcuraBI(TBI *T, int Inicio, int Fim, int NumBI)
{
   // Um ciclo desde Inicio até Fim
   for (int i=Inicio ; i<= Fim; i++) {
       if ( NumBI == T[i].NumBI ) // se encontrou o registro, então
          return i; // devolve índice
    }
    // não encontrou nenhuma ficha com o número NumBI, devolve -1
    return -1;
} // fim da função
```

Como estimar o tempo de pesquisa sequencial a uma tabela de dados?

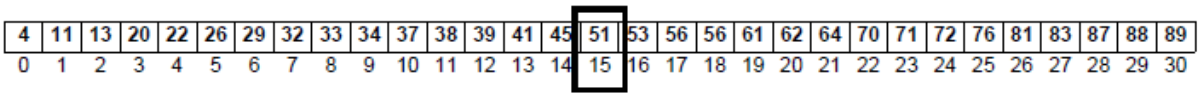
Supondo que se quer pesquisar um registro (por exemplo) com N fichas e que demoramos um tempo de T segundo para "ler e verificar" cada registro, então será de esperar que o tempo médio para achar um elemento na tabela (supondo que ele existe) seja igual a T*N/2 segundos (supondo que o registro a procurar pode estar numa qualquer posição do arquivo/tabela). Diz-se então que a pesquisa sequencial ou linear tem um peso computacional proporcional ao número de dados a pesquisar, logo tem complexidade de ordem N.

Pesquisa Binária

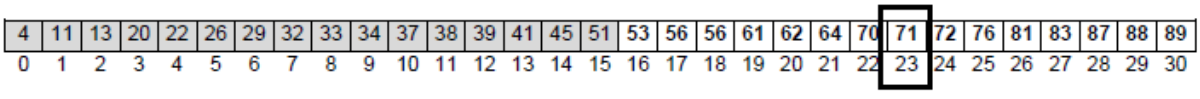
Quando o conjunto dos dados a pesquisar está **ordenado** segundo um determinado critério então podemos utilizar um método de pesquisa mais eficiente do que a pesquisa sequencial: a pesquisa binária (ou dicotómica). Esta forma de pesquisa é bastante rápida e consiste no seguinte:

1. Toma-se o conjunto de dados ordenados a pesquisar;
2. Compara-se o elemento no meio do conjunto com o termo de pesquisa. Se forem iguais terminou a pesquisa;
3. Se o termo de pesquisa for anterior ao elemento do meio então reduz-se o conjunto a pesquisar apenas aos elementos anteriores ao elemento do meio. Se o termo de pesquisa for posterior ao elemento do meio então reduz-se o conjunto de dados a pesquisar à metade posterior ao elemento do meio.
4. Enquanto o conjunto resultante não for nulo, volta-se ao passo 2, agora com metade dos elementos para procurar.

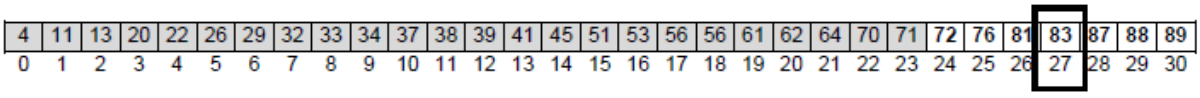
Vejamos um exemplo do funcionamento da pesquisa binária: Consideremos uma tabela de inteiros Tab com 31 valores ordenados. Queremos saber em que posição da tabela está o número 81.



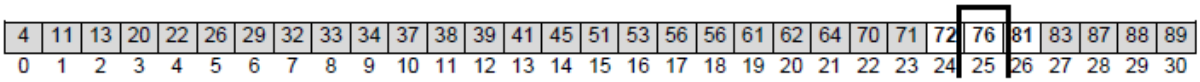
Começa-se a pesquisa pelo elemento do meio da tabela. Define-se o elemento do meio através da seguinte expressão: **(Índice Final – Índice Inicial)/2**. Neste caso concreto: Meio = (0 + 30)/2 = 15. Como **Tab[15] = 51 < 81**, então o elemento a pesquisar está entre a posição 16 e a posição 30 (ver abaixo):



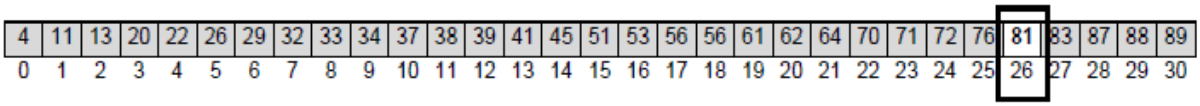
A pesquisa continua agora apenas na sub-tabela entre os índices 16 e 30. O novo elemento do meio é agora: Meio = (16+30)/2 = 23. **Tab[23] = 71 < 81**, o que implica que o elemento a procurar está entre posição 24 e 30.



A pesquisa continua agora na sub-tabela entre os índices 24 e 30. Mais uma vez determina-se o elemento do meio, Meio = (24+30)/2 = 27. Como **Tab[27] = 83 > 81** então o elemento a procurar está abaixo do elemento do meio, logo da posição 24 à posição 26.



A pesquisa continua agora entre os elementos com índices 24 e 26. O novo elemento do meio passa a ser: Meio = (24+26)/2 = 25. **Tab[25] = 76 < 81** o que implica que o elemento a procurar está acima do meio, logo apenas poderá estar no elemento com índice 26.



Finalmente analisa-se o único elemento que falta: Meio= (26+26)/2 = 26. Como Tab[26] = 81 então encontrou-se o elemento que se procurava na posição 26. Foram apenas necessárias 5 iterações até se encontrar este valor. Se tivesse sido utilizada a pesquisa sequencial neste mesmo exemplo teriam sido necessárias 27 iterações.

Resumindo, a pesquisa binária vai dividindo ao meio o conjunto a procurar até se chegar a um conjunto com apenas 1 elemento.

Como estimar o tempo de pesquisa binária de uma base de dados?

Supondo que se quer pesquisar um arquivo/tabela (por exemplo) com N fichas e que demoramos um tempo de T segundo para ler e verificar cada tabela, então podemos esperar que o tempo necessário para encontrar um elemento na tabela (caso exista ou não) seja em média T*log2(N+1) segundos (supondo que a tabela se encontra ordenada e que o registro a procurar se pode encontrar numa qualquer posição da tabela). Diz-se então que a pesquisa binária tem um peso computacional proporcional ao logaritmo dos dados a pesquisar, logo uma complexidade de ordem log(N).

Na tabela abaixo apresenta-se a correspondência entre o número de elementos de uma tabela ordenada e o número de leituras necessárias até se encontrar o elemento utilizando pesquisa binária:

Número de elementos na tabela	Número de leituras (pesquisa binária)
16...31	5
1.000	10
100.000	17
1.000.000	20
100.000.000	27

Isto quer dizer que se tivermos um arquivo com 100 milhões de dados ordenados, apenas temos que ler 27 desses dados até chegar à posição do dado procurado. Numa pesquisa sequencial teríamos que ler em média 50 milhões de fichas nessa mesma pesquisa. Podemos concluir que o fato de termos uma tabela ordenada permite reduzir em muito o tempo de pesquisa caso seja utilizada a pesquisa binária.

Implementação da pesquisa binária em C

Podemos implementar a pesquisa binária utilizando dois tipos de abordagem: uma abordagem sequencial e uma abordagem recursiva. A título de exemplo consideremos uma tabela de N inteiros ordenados por ordem crescente. Consideremos em primeiro lugar o caso sequencial. O algoritmo de pesquisa binária pode ser resumido da seguinte forma:

Algoritmo de pesquisa binária (estrutura repetitiva):

Dados: **Inicio=0**, o índice do primeiro elemento da tabela, **Fim = N-1** o índice do último elemento da tabela e **Pesq** o elemento de pesquisa;

Passo 1: Determina o elemento no meio da tabela: **Meio = (Inicio+Fim)/2**, arredondado para baixo;

Passo 2: Determina o valor da tabela no índice Meio: **Valor=Tab[Meio]**;

Passo 3: Se o valor a pesquisar **Pesq** for igual a Valor então foi encontrado o valor de pesquisa. Termina o algoritmo devolvendo o índice do Valor (Meio).

Passo 4: Se **Pesq** for maior do que Valor então procura na sub-tabela da direita, ou seja o novo **Inicio=Meio+1**. Caso contrário procura na sub-tabela da esquerda, ou seja o novo **Fim=Meio-1**;

Passo 5: Se **Inicio <= Fim** então repete de novo a partir do passo 1;

Passo 6: Como Inicio > Fim ==> Insucesso! Não foi encontrado qualquer elemento igual na tabela.

Algoritmo de pesquisa binária (recursiva):

Protótipo da função:

```
int PesquisaBinariaRec(tabela, Inicio, Fim, Valor a pesquisar)
```

1) `Se inicio maior que fim retorna -1 // não achou nada`
2) Determina o elemento no meio da tabela: **Meio = (Inicio+Fim)/2**, arredondado para baixo;
3) Verifica se valor a pesquisar for maior que elemento do meio da tabela e chama a mesma função, passando o inicio como Meio+1.
4) Se valor a pesquisar for menor que o elemento do meio, chama a mesma função passando como parâmetro Fim Meio-1.
5) Se não for maior nem menor, só pode ser igual... achou o elemento!

Bibliografia

Faculdade de Ciências e Tecnologia da Universidade de Coimbra, ESTRUTURAS DE DADOS E ALGORITMOS ALGORITMOS E ESTRUTURAS DE DADOS, 2003/2004.