Estrutura de Dados II

Busca em Memória Primária

Prof. Dr. Marcelo Otone Aguiar

Universidade Federal do Espírito Santo - UFES

30 de Março de 2023

Conteúdo

- Busca sequencial
- Busca binária
- Busca binária recursiva

Conceitos Gerais

- O objetivo da pesquisa é recuperar informação a partir de uma grande massa de informação previamente armazenada.
- A informação é dividida em **registros**.
- Cada registro possui uma **chave** para ser usada na pesquisa.
- O objetivo é encontrar uma ou mais ocorrências de registros com chaves iguais à **chave de pesquisa**.

Conceitos Gerais

- Neste caso a pesquisa teve **sucesso**.
- Um conjunto de registros é chamado de **tabela** ou **arquivo**.
- Existe uma variedade enorme de métodos de pesquisa. A escolha do método de pesquisa mais adequado a determinada aplicação depende principalmente:
  - da quantidade de dados envolvidos;
  - de o arquivo estar sujeito a inserções e retiradas frequentes, ou de o conteúdo do arquivo ser praticamente estável.

Busca Sequencial

- Este é o método mais simples de pesquisa.
- Funciona da seguinte forma: **A partir do primeiro registro, pesquise sequencialmente até encontrar a chave procurada, então pare.**
- A busca sequencial é útil em arquivos de dados desordenados.
- Em dados ordenados outros métodos podem ser aplicados com maior eficiência.
- A busca sequencial pode ser implementada de duas formas:
  - Retornar o próprio elemento encontrado;
  - Retornar o índice do elemento.

Busca Sequencial

- Complexidade: $O(n)$

Caso	Melhor Caso	Pior Caso	Caso Médio
Existe	1	n	$\frac{n}{2}$
Não existe	n	n	n

Pseudo-código

```
1  In: A variavel que contem o vetor (V) e chave (K)
2  Out: elemento encontrado ou -1
3  begin
4    for i<-0 to n-1 do
5      if elemento = K then
6        return elemento
7      end if
8    end for
9    return -1
10 end
```

Sentinela

- Uma forma de tornar o algoritmo mais eficiente na busca é utilizar uma **sentinela**;
- **Sentinela**: consiste em adicionar um elemento de valor igual à chave no final da tabela;
- O objetivo é garantir que o elemento igual à chave, sempre será encontrado, o que permite eliminar uma instrução de comparação, melhorando a performance do algoritmo

Pseudo-código

```
1  In: A variavel que contem o vetor (V) e chave (K)
2  Out: elemento encontrado ou -1
3  begin
4    v[n] <- K
5    for i<-0 to v[i] <> K do
6    end for
7
8    if i < n then
9      return v[i]
10   else
11     return -1
12   end if
13 end
```

Entradas mais solicitadas

- É comum o fato de algumas entradas serem mais solicitadas do que outras
- Desta forma, se a tabela contiver nas suas primeiras posições as entradas mais solicitadas, o número médio de comparações será menor que se a lista estivesse distribuída aleatoriamente

Entrada	Frequência
A	0,5
B	0,3
C	0,15
D	0,05
E	0,03

Entradas mais solicitadas

- O número médio de comparações para a localização de uma entrada, considerando que elas aparecem na sequência dada, será:
  - $N_c = 1 * 0,5 + 2 * 0,3 + 3 * 0,15 + 4 * 0,05 + 5 * 0,03 = 1,9$ comparações
- Caso as entradas estivessem distribuídas aleatoriamente, o número médio de comparações seria dado por:
  - $N_c = \frac{n+1}{2} = \frac{5+1}{2} = 2$

Entradas mais solicitadas

- Como não é possível conhecer antecipadamente a distribuição das entradas e suas frequências de acesso, durante o processo de pesquisa é possível mover as entradas mais solicitadas para o início da tabela
- Duas estratégias podem ser adotadas:
  - **Método mover-para-frente:** mover sempre para o início
  - **Método da transposição:** trocar pelo imediatamente anterior

Entradas mais solicitadas: mover-para-frente

- Desvantagens do método **mover-para-frente**
  - Uma única recuperação não implica que o registro será frequentemente consultado
  - Perda de eficiência para outros registros
  - Custo alto para mover em vetores
- Vantagens do método **mover-para-frente**
  - Possui resultados melhores para quantidades pequena e média de buscas

Pseudo-código

```
1  In: A variavel que contem o vetor (V) e chave (K)
2  Out: elemento encontrado ou -1
3  begin
4    for i<-0 to n-1 do
5      if V[i] = K then
6        pushElemToK(V)
7        V[0] <- K
8        return V[0]
9      end if
10   end for
11   return -1
12 end
```

Busca Binária

- Se o dado a ser encontrado se apresentar de forma ordenada, pode ser utilizado um método muito superior para encontrar o elemento procurado
- Esse método é a busca binária que utiliza a abordagem dividir e conquistar
- Ele primeiro verifica o elemento central, se esse elemento é maior que a chave, ele testa o elemento central da primeira metade; caso contrário, ele testa o elemento central da segunda metade
- Esse procedimento é repetido até que o elemento seja encontrado ou que não haja mais elementos a testar

Busca Binária

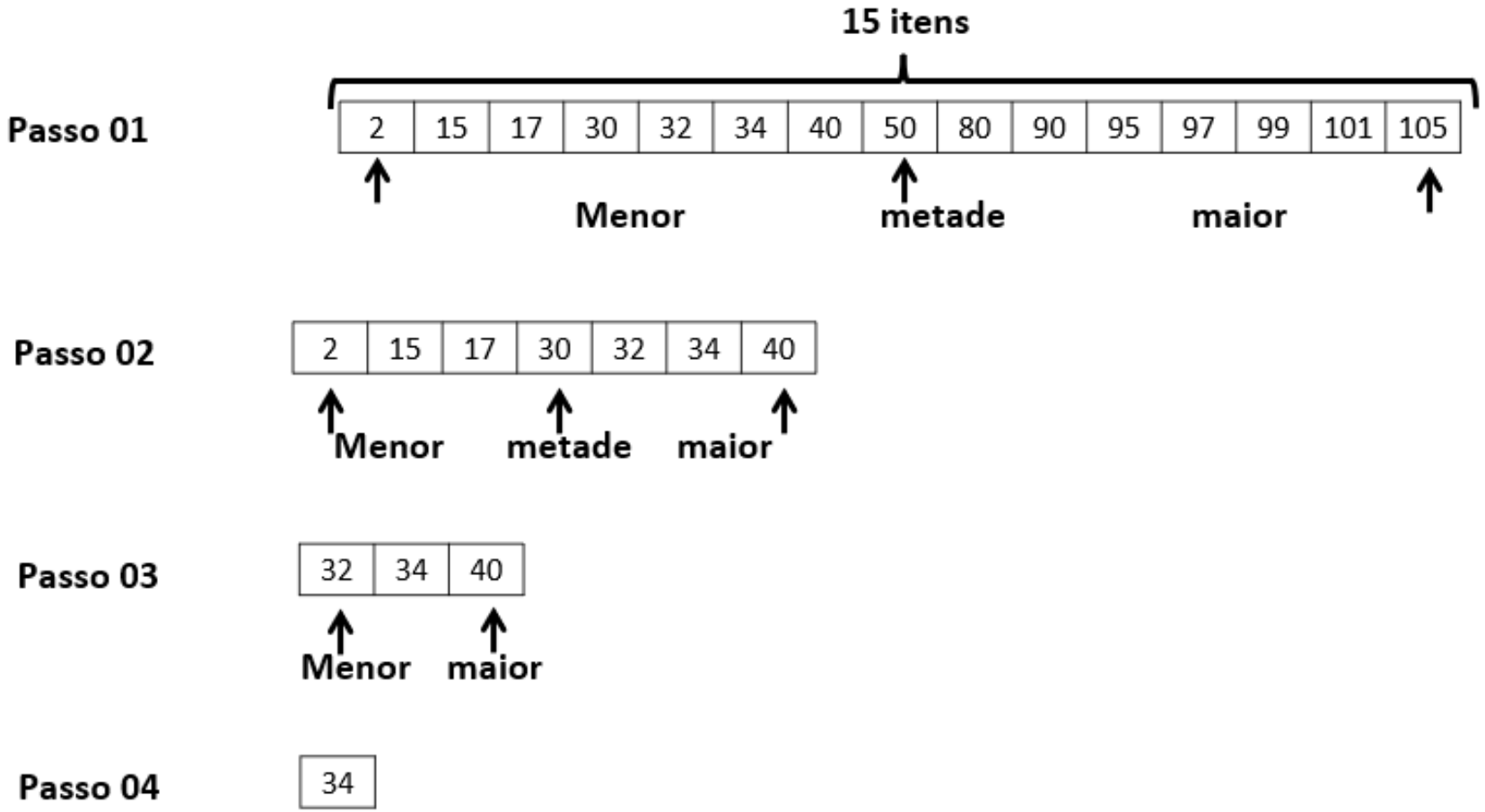
- Por exemplo, para encontrar o número **3** na matriz **1 2 3 4 5 6 7 8 9**, uma pesquisa binária primeiro testa o elemento médio, nesse caso **5**
- Visto que é maior que **3**, a pesquisa continua com a primeira metade ou **1 2 3 4**. O elemento central agora é **2**, que é menor que **3**, então a primeira metade é descartada. A pesquisa continua com **3 4**
- Nesse momento o elemento é encontrado

Busca Binária

- O desempenho deste algoritmo é dado pela expressão O(lgn) sendo o n o número de elementos da tabela
- Entretanto, o custo para manter a tabela ordenada é alto
- Cada inserção na posição **p** da tabela implica o deslocamento dos registros a partir da posição **p** para as posições seguintes
- Consequentemente, a pesquisa binária não deve ser usada em aplicações muito dinâmicas

Busca Binária

Exemplo: busca pelo elemento **34**



Pseudo-código

```
1  In: A variavel que contem o vetor (V) e chave (K)
2  Out: elemento encontrado ou -1
3  begin
4    inicio <- 0
5    fim <- N-1
6    while inicio <= fim do
7      meio <- (inicio + fim) / 2
8      if V[meio] = K then
9        return meio
10     else if V[meio] < K then
11         inicio <- meio + 1
12     else
13         fim <- meio - 1
14     end if
15   end while
16   return -1
17 end
```

Pseudo-código

```
1  In: A variavel que contem o vetor (V), chave (K) o
       inicio e o fim
2  Out: elemento encontrado ou -1
3  begin
4    meio <- (inicio + fim) / 2
5
6    if inicio > fim then
7      return -1
8    else if V[meio] = K then
9      return meio
10   else if V[meio] < K then
11       return SearchBinRec(V, K, meio + 1, fim)
12   else
13       return SearchBinRec(V, K, inicio, meio - 1)
14   end if
15 end
```