MATA54 - Estruturas de Dados e Algoritmos II

Arquivos Sequenciais

Flávio Assis
Versão gerada a partir de slides do Prof. George Lima

IC - Instituto de Computação

Salvador, agosto de 2021

Organização de Arquivos Sequenciais

Arquivo:

Sequência de **registros**

Registro:

Item de dado contendo informação de um elemento da aplicação

- Um registro contém um conjunto de **campos**: campo 1, campo 2,. . ., campo m

Registros estão dispostos em uma sequência de endereços físicos. Acesso aos mesmos pode ser **sequencial** ou **direto**.

Endereços:	1	2	3	. . .	n
Registros:	r_1	r_2	r_3	. . .	r_n

Chave Primária e Secundária

Uma **chave primária** é um campo (ou conjunto de campos) cujo valor identifica um registro de forma única.

Qualquer combinação de campos que não identifique um registro de forma única é chamado de uma **chave secundária**

Exemplos de chave primária:

- CPF em registros sobre pessoa física
- suponha que cada departamento em uma banco de dados de uma empresa possua um identificador único e que cada funcionário de um departamento possua um identificador único **apenas em seu departamento**. O identificador do departamento juntamente com o identificador do funcionário é uma chave primária

Chave Primária e Secundária

Qual a importância da definição de chave primária e secundária?

A busca por uma chave primária resulta em no máximo um registro como resposta. A busca por chave secundária pode retornar potencialmente vários registros como resposta.

Um arquivo pode ser organizado de forma a otimizar o tipo de resposta.

Busca por Chave Primária

A **busca por chave primária** consiste em verificar se existe ou não registro no arquivo cujo valor de chave primária é igual a um determinado valor desejado v.

Se houver, a busca deve retornar uma indicação de qual é este registro.

Caso contrário, deve informar que tal registro não existe.

Quando não houver ambigüidade, usaremos apenas **chave** para denotar chave primária.

Notação: denotaremos por $r_i.k$ o valor da chave primária de um registro de índice i

Busca por Chave Primária: Busca Sequencial

A partir do primeiro registro, compara registro após registro, na sequência do arquivo, se a chave primária do registro é igual ao valor desejado v. A busca termina, caso se encontre um registro com valor de chave primária igual ao valor desejado ou se chega ao final do arquivo.

Algorithm 1: Busca sequencial

entrada: Arquivo com n registros $r_1, r_2, ..., r_n$; valor v a ser procurado
saida : Índice do registro no arquivo cuja chave é v (caso exista) ou -1 (caso contrário)

1 $i \leftarrow 1$;
2 **while** $(i \leq n) \wedge (r_i.k \neq v)$ **do** i $\leftarrow$ i + 1;
3 **if** $i \leq n$ **then return** i;
4 **return** -1;

Complexidade

Algoritmo	**Busca com Sucesso**		**Busca sem Sucesso**
	Melhor Caso	**Pior Caso**	
Busca Sequencial (não ordenado)	**O(1)**	**O(n)**	**O(n)**

Busca Sequencial - Arquivo Ordenado

Arquivo ordenado (ordem crescente) pela chave primária.

Algorithm 2: Busca sequencial - arquivo ordenado

entrada: Arquivo **ordenado** (ordem crescente) pela chave com n registros $r_1, r_2, ..., r_n$; valor v a ser procurado

saida : Índice do registro no arquivo cuja chave é v (caso exista) ou -1 (caso contrário)

1 $i \leftarrow 1$;
2 **while** $(i \leq n) \wedge (r_i.k < v)$ **do** i $\leftarrow$ i + 1;
3 **if** $(i \leq n) \wedge (r_i.k = v)$ **then return** i;
4 **return** -1;

Complexidade

Algoritmo	**Busca com Sucesso**		**Busca sem Sucesso**	
	Melhor Caso	**Pior Caso**	**Melhor Caso**	**Pior Caso**
Busca Sequencial (não ordenado)	**O(1)**	**O(n)**	**O(n)**	
Busca Sequencial (ordenado)	**O(1)**	**O(n)**	**O(1)**	**O(n)**

Busca Binária

Algorithm 3: Busca Binária

entrada: Arquivo **ordenado** por chave primária com n registros; valor v a ser buscado

saida : Índice do registro no arquivo cuja chave é v (caso exista) ou -1 (caso contrário)

```
1 i ← 1; f ← n;
2 while i ≤ f do
3     m ← nextPos(v, i, f)
4     if v = r_m.k then return m;
5     else if v > r_m.k then i ← m + 1;
6     else f ← m − 1;
7 return −1;
```

- Qual seria uma possibilidade para a função **nextPos**?

$$\textbf{nextPos}(v, i, f) \leftarrow \left\lfloor \frac{i+f}{2} \right\rfloor$$

Complexidade

Algoritmo	**Busca com Sucesso**		**Busca sem Sucesso**	
	Melhor Caso	**Pior Caso**	**Melhor Caso**	**Pior Caso**
Busca Sequencial (não ordenado)	**O(1)**	**O(n)**	**O(n)**	
Busca Sequencial (ordenado)	**O(1)**	**O(n)**	**O(1)**	**O(n)**
Busca Binária	**O(1)**	**O(log *n*)**	**O(log *n*) - obs.1**	

Obs. 1: Pode-se facilmente alterar o algoritmo para o melhor caso da busca binária sem sucesso ser $O(1)$

Busca Binária

O que aconteceria se alterássemos a função **nextPos(v, i, f)** para que ela retornasse uma posição qualquer no intervalo $[i, f]$? Ou seja:

$$\textbf{nextPos}(v, i, f) \leftarrow \text{random}([\text{i, f}])$$

O algoritmo continuaria correto?

A complexidade continuaria a mesma?

Busca por Interpolação

Tentativa de se escolher uma posição mais apropriada:

$$\textbf{nextPos}(v, i, f) \leftarrow \left\lceil i + \frac{(v - r_i.k)(f - i)}{r_f.k - r_i.k} \right\rceil$$

Busca por Interpolação

O algoritmo deve ser levemente alterado para acomodar o caso de busca sem sucesso (valor fora da faixa de valores entre as posições i e f).

Algorithm 4: Busca por Interpolação

entrada: Arquivo **ordenado** por chave primária com n registros; valor v a ser buscado

saida : Índice do registro no arquivo cuja chave é v (caso exista) ou -1 (caso contrário)

1 $i \leftarrow 1; f \leftarrow n;$
2 **while** $(r_f.k \geq v) \wedge (v > r_i.k)$ **do**
3 $\quad m \leftarrow \left\lceil i + \frac{(v - r_i.k)(f-i)}{r_f.k - r_i.k} \right\rceil$
4 $\quad$ **if** $v = r_m.k$ **then** $i \leftarrow m;$
5 $\quad$ **else if** $v > r_m.k$ **then** $i \leftarrow m + 1;$
6 $\quad$ **else** $f \leftarrow m - 1;$
7 **if** $r_i.k = v$ **then return** $i;$
8 **return** $-1;$

Busca por Interpolação - Exemplo

Procura-se por $v = 26$:

1	2	3	4	5	6	7
6	12	15	25	26	35	40
i						f

Passos:

1) $m \leftarrow \left\lceil 1 + \frac{(26-6)(7-1)}{(40-6)} \right\rceil = 5$

O valor da chave na posição 5 é igual a 26, ou seja, $r_5.k = v$

Achou com apenas uma comparação!

Quantas comparações seriam necessárias na busca binária?

Busca por Interpolação - Pior Caso

Procura-se por $v = 97$:

1	2	3	4	5
1	97	98	99	100
i				f

Passos:

1) $m \leftarrow \left\lceil 1 + \frac{(97-1)(5-1)}{(100-1)} \right\rceil = 5$

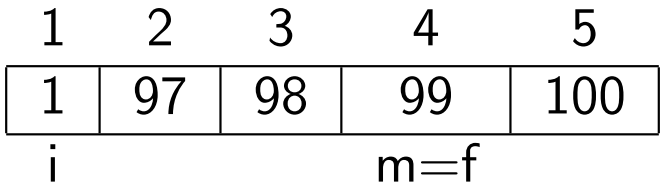
1	2	3	4	5
1	97	98	99	100
i				m=f

$r_5.k > v$: $f \leftarrow m - 1$

Busca por Interpolação - Pior Caso

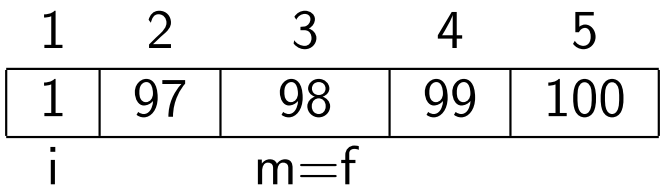
Passos:

2)



$m \leftarrow \left\lceil 1 + \frac{(97-1)(4-1)}{(99-1)} \right\rceil = 4$
$r_4.k > v$: $f \leftarrow m - 1$

3)



$m \leftarrow \left\lceil 1 + \frac{(97-1)(3-1)}{(98-1)} \right\rceil = 3$
$r_3 k > v$: $f \leftarrow m - 1$

Busca por Interpolação - Pior Caso

Passos:

4)

1	2	3	4	5
1	97	98	99	100
i	m=f			

$$m \leftarrow \left\lceil 1 + \frac{(97-1)(2-1)}{(97-1)} \right\rceil = 2$$

$r_2.k = v$: Achou! Com $n - 1$ passos! **O(n)**

Complexidade

Algoritmo	Busca com Sucesso		Busca sem Sucesso	
	Melhor Caso	Pior Caso	Melhor Caso	Pior Caso
Busca Sequencial (não ordenado)	**O(1)**	**O(n)**	**O(n)**	
Busca Sequencial (ordenado)	**O(1)**	**O(n)**	**O(1)**	**O(n)**
Busca Binária	**O(1)**	**O($\log n$)**	**O($\log n$) - obs.1**	
Interpolação	**O(1)**	**O(n)**	**O(1)**	**O(n)**

Obs. 1: Pode-se facilmente alterar o algoritmo para o melhor caso da busca binária sem sucesso ser $O(1)$

Obs. 2: A busca por interpolação possui caso médio **O($\log \log n$)**

Arquivos Sequenciais Auto-Organizados

- Manter arquivo em ordem física de chaves é custoso. Quais alternativas?
- E se registros com maior probabilidade de serem pesquisados estiverem no início:
  - **Mover para frente**: ao acessar r_i, colocar r_i na primeira posição, deslocando os demais registros à direita
  - **Transpor**: ao acessar r_i $(i > 1)$, trocar sua posição com r_{i-1}
  - **Contador de acesso**: mantém a ordem de acesso proporcional ao número de acessos a cada registro.
- Qual o custo-benefício? Outras possibilidades para auto-organização?