



# Métodos de Busca

## Parte 1

---

### **Introdução à Ciência da Computação II**

Prof. Diego Raphael Amancio



# Introdução

---

- Importância em estudar busca
  - Busca é uma tarefa muito comum?
- Vários métodos e estruturas de dados podem ser empregados para se fazer busca
  - Quais estruturas de dados?
- Certos métodos de organização/ordenação de dados podem tornar o processo de busca mais eficiente



# Introdução

---

- O problema da busca (ou pesquisa)

*“Dado um conjunto de elementos, onde cada um é identificado por uma **chave**, o objetivo da busca é localizar, nesse conjunto, o elemento que corresponde a uma chave específica”*



# Termos Relacionados

---

- Tabela: termo genérico, pode ser qualquer estrutura de dados usada para armazenamento interno e organização dos dados
- Uma tabela é um conjunto de elementos, chamados registros



# Termos Relacionados

---

- Existe uma chave associada a cada registro, usada para diferenciar os registros entre si
  - Chave interna: chave está contida dentro do registro, em uma localização específica
  - Chave externa: essas chaves estão contidas em uma tabela de chaves separada que inclui ponteiros para os registros
  - Chave primária: para todo arquivo existe pelo menos um conjunto exclusivo de chaves
    - Dois registros não podem ter o mesmo valor de chave
  - Chave secundária: são as chaves não primárias
    - Chaves que não precisam ter seus valores exclusivos



# Termos Relacionados

---

- Algoritmo de busca
  - Formalmente, é o algoritmo que aceita um argumento **a** e tenta encontrar o registro cuja chave seja **a**



# Termos Relacionados

---

- Operações na tabela

- Inserção: adicionar um novo elemento à tabela
- Algoritmo de busca e inserção: se não encontra o registro, insere um novo
- Remoção: retirar um elemento da tabela
- Recuperação: procurar um elemento na tabela e, se achá-lo, torná-lo disponível



# Tipos de Busca

---

- A **tabela** pode ser:
  - Um vetor de registros
  - Uma lista encadeada
  - Uma árvore
  - Etc.
- A tabela pode ficar:
  - Totalmente na memória (**busca interna**)
  - Totalmente no armazenamento auxiliar (busca externa)
  - Dividida entre ambos





# Tipos de Busca

---

- Algumas **técnicas de busca em memória interna** são
  - Busca Seqüencial
  - Busca Binária
  - Busca por Interpolação
  - Busca em Árvores
  - Hashing
- O objetivo é encontrar um dado registro com o **menor custo**
  - Cada técnica possui vantagens e desvantagens



# Busca Seqüencial

---

- A busca seqüencial é a forma mais simples de busca
  - É aplicável a uma tabela organizada como um **vetor** ou como uma **lista encadeada**



# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

|    |    |    |    |    |    |    |    |     |  |
|----|----|----|----|----|----|----|----|-----|--|
| 1  |    |    |    |    |    |    |    | N=8 |  |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92 |     |  |



# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48

|    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|-----|
| 1  |    |    |    |    |    |    | N=8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92  |



# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48

|    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|-----|
| 1  |    |    |    |    |    |    | N=8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92  |
| ↑  |    |    |    |    |    |    |     |



# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48

|    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|-----|
| 1  |    |    |    |    |    |    | N=8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92  |
|    | ↑  |    |    |    |    |    |     |



# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48

|    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|-----|
| 1  |    |    |    |    |    |    | N=8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92  |

↑



# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48

|    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|-----|
| 1  |    |    |    |    |    |    | N=8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92  |

↑





# Busca Seqüencial

---

- Busca mais simples que há
  - Percorre-se registro por registro em busca da chave

Procure por 48

|    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|-----|
| 1  |    |    |    |    |    |    | N=8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92  |

↑



# Busca Seqüencial

---

- Implementação
  - Algoritmo de busca seqüencial em um vetor  $A$ , com  $N$  posições (0 até  $N-1$ ), sendo  $x$  a chave procurada



# Busca Seqüencial

---

- Uma maneira de tornar o algoritmo mais eficiente é usar um **sentinela**
  - Sentinela: consiste em adicionar um elemento de valor x no final da tabela
  - Qual a vantagem de se usar um nó sentinela?



# Busca Seqüencial

---

- Uma maneira de tornar o algoritmo mais eficiente é usar um **sentinela**
  - Sentinela: consiste em adicionar um elemento de valor x no final da tabela
  - O sentinela garante que o elemento será encontrado, o que elimina um teste, melhorando a performance do algoritmo



# Busca Seqüencial

---

- Implementação
  - Busca seqüencial com sentinela



# Busca Seqüencial

---

- Limitações do vetor
  - Tamanho fixo
    - Pode desperdiçar ou faltar espaço
- Alternativa
  - Lista encadeada
    - O que muda na busca seqüencial?



# Busca Seqüencial

---

```
//busca sequencial com sentinela no fim;  
//exige que arranjo seja declarado com uma posição a mais  
int busca_sequencial_com_sentinela(int v[], int n, int x) {  
    int i;  
    v[n]=x;  
    for(i=0; x!=v[i]; i++);  
    if (i<n) return(i);  
    else return(-1);  
}
```



# Busca Seqüencial

---

- Complexidade

- Se o registro for o primeiro: 1 comparação
- Se o registro procurado for o último: N comparações
- Se for igualmente provável que o argumento apareça em qualquer posição da tabela, em média:  $(n+1)/2$  comparações
- Se a busca for mal sucedida: N comparações
- Logo, a busca seqüencial, no pior caso, é  $O(n)$





# Busca Seqüencial

---

- Arranjo não ordenado
  - Inserção no final do arranjo
  - Remoção
    - Realocação dos registros acima do registro removido



# Busca Seqüencial

---

- Para aumentar a eficiência
  - Reordenar continuamente a tabela de modo que os registros mais acessados sejam deslocados para o início
    - A) Método mover-para-frente: sempre que uma pesquisa obtiver êxito, o registro recuperado é colocado no início da lista
    - B) Método da transposição: um registro recuperado com sucesso é trocado com o registro imediatamente anterior
      - Ambos se baseiam no fenômeno da recuperação recorrente de registros



# Busca Seqüencial

---

- Desvantagens do método mover-para-frente
  - Uma única recuperação não implica que o registro será freqüentemente recuperado
    - Perda de eficiência para os outros registros
  - O método é mais “caro” em vetores do que em listas
    - Por quê?



# Busca Seqüencial

---

- Busca **seqüencial** em tabela ordenada
  - A eficiência da operação de busca melhora se as chaves dos registros estiverem ordenadas
    - No **pior caso (caso em que a chave não é encontrada)**, são necessárias  $N$  comparações quando as chaves estão desordenadas
    - No caso médio,  $N/2$  comparações se as chaves estiverem ordenadas, pois se para a busca assim que uma chave maior do que a procurada é encontrada



# Busca Seqüencial

---

- Busca seqüencial indexada
  - Existe uma tabela auxiliar, chamada tabela de índices, além do próprio arquivo ordenado
  - Cada elemento na tabela de índices contém uma **chave** (*kindex*) e um **indicador do registro** no arquivo que corresponde a *kindex*
    - Faz-se a busca a partir do ponto indicado na tabela, sendo que a busca não precisa ser feita desde o começo
  - Pode ser implementada como um vetor ou como uma lista encadeada
    - O indicador da posição na tabela pode ser um ponteiro ou uma variável inteira

# Busca Seqüencial

Tabela de índices

*index*

|     |  |
|-----|--|
| 321 |  |
| 592 |  |
| 876 |  |
| ... |  |

Chave Registro

|            |  |
|------------|--|
| ....       |  |
| <b>14</b>  |  |
| <b>38</b>  |  |
| <b>115</b> |  |
| <b>321</b> |  |
| <b>387</b> |  |
| <b>512</b> |  |
| <b>567</b> |  |
| <b>583</b> |  |
| <b>592</b> |  |
| <b>611</b> |  |
| <b>741</b> |  |
| <b>811</b> |  |



# Busca Seqüencial

---

- Busca seqüencial indexada
  - Se a tabela for muito grande, pode-se ainda usar a **tabela de índices secundária**
    - O índice secundário é um índice para o índice primário

# Busca Seqüencial

Índice secundário

Índice primário

*index*

|     |  |
|-----|--|
| 321 |  |
| 876 |  |
| ... |  |

*index*

|     |  |
|-----|--|
| 321 |  |
| 592 |  |
| 876 |  |
| ... |  |

| Chave | Registro |
|-------|----------|
| 14    |          |
| 38    |          |
| 115   |          |
| 321   |          |
| 387   |          |
| 512   |          |
| 567   |          |
| 583   |          |
| 592   |          |
| 611   |          |
| 741   |          |
| 811   |          |
| 876   |          |





# Busca Seqüencial

---

- Vantagem

- Os itens na tabela poderão ser examinados seqüencialmente sem que todos os registros precisem ser acessados
  - O tempo de busca diminui consideravelmente

- Desvantagens



# Busca Seqüencial

---

- Vantagem

- Os itens na tabela poderão ser examinados seqüencialmente sem que todos os registros precisem ser acessados
  - O tempo de busca diminui consideravelmente

- Desvantagens

- A tabela tem que estar ordenada



# Busca Seqüencial

---

- Vantagem

- Os itens na tabela poderão ser examinados seqüencialmente sem que todos os registros precisem ser acessados
  - O tempo de busca diminui consideravelmente

- Desvantagens

- A tabela tem que estar **ordenada**
- Exige **espaço adicional** para armazenar a(s) tabela(s) de índices

- **Algo mais?**



# Busca Seqüencial

---

- Vantagem

- Os itens na tabela poderão ser examinados seqüencialmente sem que todos os registros precisem ser acessados
  - O tempo de busca diminui consideravelmente

- Desvantagens

- A tabela tem que estar ordenada
- Exige espaço adicional para armazenar a(s) tabela(s) de índices

- Cuidados com inserção e remoção



# Busca Seqüencial

---

- Remoção

- Remove-se o elemento e rearranja-se a tabela inteira e o(s) índice(s)
- Marca-se a posição do elemento removido, indicando que ela pode ser ocupada por um outro elemento futuramente
  - A posição da tabela fica vazia



# Busca Seqüencial

---

## ■ Inserção

- Se houver espaço vago na tabela, rearranjam-se os elementos localmente
- Se não houver espaço vago
  - Rearranjar a tabela a partir do ponto apropriado e reconstruir o(s) índice(s)

# Busca Seqüencial

- Inserção do elemento 512 com espaço vago
  - 567 e 583 descem
  - 512 é inserido

*index*

|     |  |
|-----|--|
| 321 |  |
| 592 |  |
| 876 |  |
| ... |  |

Chave      Registro

|      |  |
|------|--|
| .... |  |
| 14   |  |
| 38   |  |
| 115  |  |
| 321  |  |
| 387  |  |
| 567  |  |
| 583  |  |
|      |  |
| 592  |  |
| 611  |  |
| 741  |  |
| 811  |  |

# Busca Seqüencial

- Inserção do elemento 512 com espaço vago
  - 567 e 583 descem
  - 512 é inserido

*index*

|     |  |
|-----|--|
| 321 |  |
| 592 |  |
| 876 |  |
| ... |  |

Chave      Registro

|      |  |
|------|--|
| .... |  |
| 14   |  |
| 38   |  |
| 115  |  |
| 321  |  |
| 387  |  |
| 512  |  |
| 567  |  |
| 583  |  |
| 592  |  |
| 611  |  |
| 741  |  |
| 811  |  |



# Busca Seqüencial

- Inserção do elemento 512 sem espaço vago

- Elementos a partir de 567 descem
- 512 é inserido
- Índice é re-construído

*index*

|     |  |
|-----|--|
| 321 |  |
| 592 |  |
| 876 |  |
| ... |  |

Chave      Registro

|      |  |
|------|--|
| .... |  |
| 14   |  |
| 38   |  |
| 115  |  |
| 321  |  |
| 387  |  |
| 567  |  |
| 583  |  |
| 585  |  |
| 592  |  |
| 611  |  |
| 741  |  |
| 811  |  |

# Busca Seqüencial

- Inserção do elemento 512 sem espaço vago

- Elementos a partir de 567 descem
- 512 é inserido
- Índice é re-construído

*index*

|     |  |
|-----|--|
| 321 |  |
| 592 |  |
| 876 |  |
| ... |  |

Chave Registro

|      |  |
|------|--|
| .... |  |
| 14   |  |
| 38   |  |
| 115  |  |
| 321  |  |
| 387  |  |
| 512  |  |
| 567  |  |
| 583  |  |
| 585  |  |
| 592  |  |
| 611  |  |
| 741  |  |



# Busca Seqüencial

---

- Como montar o índice primário
  - Se a tabela não estiver ordenada, ordene-a
  - Divide-se o número de elementos da tabela pelo tamanho do índice desejado:  $n/\text{tamanho-índice}$
  - Para montar o índice, recuperam-se da tabela os elementos 0,  $0+n/\text{tamanho-índice}$ ,  $0+2*n/\text{tamanho-índice}$ , etc.
  - Cada elemento do índice representa  $n/\text{tamanho-índice}$  elementos da tabela



# Busca Seqüencial

---

- Exemplo

- Divide-se o número de elementos da tabela pelo tamanho do índice desejado
  - Se a tabela tem 1.000 elementos e deseja-se um índice primário de 10 elementos, faz-se  $1.000/10=100$
- Para montar o índice, recuperam-se da tabela os elementos 0,  $0+n/\text{tamanho-índice}$ ,  $0+2*n/\text{tamanho-índice}$ , etc.
  - O índice primário é montado com os elementos das posições 0, 100, 200, etc. da tabela
- Cada elemento do índice representa  $n/\text{tamanho-índice}$  elementos da tabela
  - Cada elemento do índice primário aponta para o começo de um grupo de 100 elementos da tabela



# Busca Seqüencial

---

- Para montar um índice secundário, aplica-se raciocínio similar sobre o índice primário
- Em geral, não são necessários mais do que 2 índices



# Busca Seqüencial

---

- Exercício

- Escrever em C a sub-rotina para produzir o índice primário de um vetor ordenado

`void criaIndice`

*kindex* ←

- `bloco_indice indice[]`

*pos* ←

- `int tamanho_indice`

- `int v[]`

- `int n`



# Implementação

---

```
//função que cria o índice primário para
//a busca sequencial indexada usar
void cria_indice(bloco_indice indice[],
                int tam_indice, int v[], int n) {
    int pos, i=0;
    while (i<tam_indice) {
        pos=i*n/tam_indice;
        indice[i].kindex=v[pos];
        indice[i].pos=pos;
        i++;
    }
}
```



# Busca Seqüencial

---

- Exercício (nota adicional)
  - Escrever em C uma sub-rotina de busca sequencial indexada por um elemento em uma tabela com índice primário
  - `int bsi` □ retorna posicao
    - `int v[], int n`
    - `bloco_indice indice[], int tamanho_indice`
    - `int x`



```
//busca sequencial indexada: além do vetor ordenado, exige que se tenha  
o índice previamente criado (pela função acima)  
int busca_sequencial_indexada(int v[], int n, bloco_indice indice[], int  
tam_indice, int x) {  
    int i;  
  
    //busca no índice primário  
    for (i=0; i<tam_indice && indice[i].kindex<=x; i++);  
    i=indice[i-1].pos;  
  
    //busca na tabela  
    while ((i<n) && (v[i]<x))  
        i++;  
    if ((i<n) && (v[i]==x))  
        return(i);  
    else return(-1);  
}
```



# Busca Binária

---

- Se os dados estiverem **ordenados** em um arranjo, pode-se tirar vantagens dessa ordenação
  - Busca binária

$A[i] \leq A[i+1]$ , se ordem crescente

$A[i] \geq A[i+1]$ , se ordem decrescente



# Busca Binária

---

- O elemento buscado é comparado ao elemento do meio do arranjo
  - Se igual, busca bem-sucedida
  - Se menor, busca-se na metade inferior do arranjo
  - Se maior, busca-se na metade superior do arranjo



# Busca Binária

---

- Busca-se por 25

|       |    |         |    |    |    |    |    |
|-------|----|---------|----|----|----|----|----|
| inf=1 |    | sup=N=8 |    |    |    |    |    |
| 12    | 25 | 33      | 37 | 48 | 57 | 86 | 92 |



# Busca Binária

---

- Busca-se por 25

|       |    |      |    |         |    |    |    |
|-------|----|------|----|---------|----|----|----|
| inf=1 |    | meio |    | sup=N=8 |    |    |    |
| 12    | 25 | 33   | 37 | 48      | 57 | 86 | 92 |

↑  
25 < 37



# Busca Binária

---

- Busca-se por 25

|       |       |    |     |    |    |    |    |
|-------|-------|----|-----|----|----|----|----|
| inf=1 | sup=3 |    | N=8 |    |    |    |    |
| 12    | 25    | 33 | 37  | 48 | 57 | 86 | 92 |



# Busca Binária

---

- Busca-se por 25

|       |      |       |    |    |    |    |     |
|-------|------|-------|----|----|----|----|-----|
| inf=1 | meio | sup=3 |    |    |    |    | N=8 |
| 12    | 25   | 33    | 37 | 48 | 57 | 86 | 92  |

↑ =25



# Busca Binária

---

- Busca-se por 25

|       |      |       |    |    |    |    |     |
|-------|------|-------|----|----|----|----|-----|
| inf=1 | meio | sup=3 |    |    |    |    | N=8 |
| 12    | 25   | 33    | 37 | 48 | 57 | 86 | 92  |

↑ =25

Em cada passo, o tamanho do arranjo em que se busca é dividido por 2





# Busca Binária

---

- Exercício
  - Escrever em C uma sub-rotina de busca binária por um elemento em um arranjo ordenado
    - Versão recursiva
    - Versão não recursiva

```
//busca binária não recursiva
int busca_binaria(int v[], int n, int x) {
    int inf=0, sup=n-1, meio;

    while (inf<=sup) {
        meio=(inf+sup)/2;
        if (x==v[meio])
            return(1);
        else if (x<v[meio])
            sup=meio-1;
        else inf=meio+1;
    }

    return(0);
}
```

//busca binária recursiva: deve passar os limites inferior (inicialmente igual a 0) e superior (inicialmente igual a n-1)

```
int busca_binaria_rec(int v[], int inf, int sup, int x) {  
    int meio;  
  
    if (inf<=sup) {  
        meio=(inf+sup)/2;  
        if (x==v[meio])  
            return(1);  
        else if (x<v[meio])  
            return(busca_binaria_rec(v,inf,meio-1,x));  
        else return(busca_binaria_rec(v,meio+1,sup,x));  
    }  
    else return(0);  
}
```



# Busca Binária

---

- Complexidades?



# Busca Binária

---

- Complexidades?
  - $O(\log(n))$ , pois cada comparação reduz o número de possíveis candidatos por um fator de 2



# Busca Binária

---

- Vantagens

- Eficiência da busca
- Simplicidade da implementação

- Desvantagens

- Nem todo arranjo está ordenado



# Busca Binária

---

- Vantagens

- Eficiência da busca
- Simplicidade da implementação

- Desvantagens

- Nem todo arranjo está ordenado
- Exige o uso de um arranjo para armazenar os dados
  - Faz uso do fato de que os índices do vetor são inteiros consecutivos
- Inserção e remoção de elementos são ineficientes
  - Realocação de elementos



# Busca Binária

---

- A busca binária pode ser usada com a organização de tabela seqüencial indexada
  - Em vez de pesquisar o índice seqüencialmente, pode-se usar uma busca binária





# Busca por Interpolação

---

- Se as chaves estiverem uniformemente distribuídas, esse método pode ser ainda mais eficiente do que a busca binária
- Com chaves uniformemente distribuídas, pode-se esperar que  $x$  esteja aproximadamente na posição

$$\text{meio} = \text{inf} + (\text{sup} - \text{inf}) * ((x - A[\text{inf}]) / (A[\text{sup}] - A[\text{inf}]))$$

sendo que  $\text{inf}$  e  $\text{sup}$  são redefinidos iterativamente como na busca binária



# Busca por Interpolação

---

- Complexidade
  - $O(\log(\log(n)))$  se as chaves estiverem uniformemente distribuídas
    - Raramente precisará de mais comparações
  - Se as chaves **não estiverem uniformemente** distribuídas, a busca por interpolação pode ser **tão ruim quanto uma busca seqüencial**



# Busca por Interpolação

---

- Desvantagem
  - Em situações práticas, as **chaves tendem a se aglomerar** em torno de determinados valores e não são uniformemente distribuídas
    - Exemplo: há uma quantidade maior de nomes começando com "S" do que com "Q"