Sistemas de Informação | AMF
Classificação e Pesquisa de Dados

Aula 6 - Introdução a famílias de métodos de pesquisa de dados. Pesquisa sequencial, binária e métodos adicionais

Prof. Cristiano Santos

Baseado Material:
Prof. Rhauani Fazul

Agenda

- Retomando conceitos
- Introdução à pesquisa de dados
- Pesquisa sequencial
- Pesquisa binária
- Métodos adicionais
- *Hands-on*

Conteúdo Programático

1. Métodos de Classificação de Dados

2. Pesquisa de Dados

1. Famílias de métodos de pesquisa de dados
2. Pesquisa sequencial (pesquisa linear)
3. Pesquisa binária
4. Pesquisa digital
5. Árvores de busca
    1. Árvores binárias de pesquisa sem balanceamento
    2. Árvores binárias de pesquisa com balanceamento
6. Tabelas de dispersão (*hash tables*)
    1. Funções de transformação de chave (*hashing*)
    2. Cálculo de endereços e tratamento de colisões
    3. Endereçamento aberto
    4. Listas encadeadas
    5. Hashing perfeito
7. Pesquisa de dados em memória secundária
    1. Acesso sequencial indexado
    2. Árvores de pesquisa
    3. Árvores-B.
    4. Árvores-B*

3. Introdução à Análise da Complexidade de Algoritmos

Pesquisa

- **Acesso rápido:** eficiência de localização para acessar informações específicas em grandes conjuntos de dados:
  - Permitir que os usuários de mecanismos de busca e bancos de dados encontrem o que estão procurando (UX);
  - Processamento em tempo real e resposta instantânea em sistemas sensíveis ao tempo.
- **Minimizar uso de recursos:** melhor utilização de recursos e otimização do uso da memória:
  - Muitas operações, como a atualização de um banco de dados, dependem primeiro da localização do item correto.

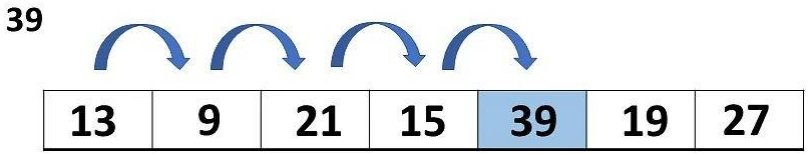
Pesquisa

- **Exemplos de uso:**
  - Mecanismos de busca na web,
  - sistemas de banco de dados,
  - sistemas de arquivos,
  - *autocomplete* em aplicativos e sites,
  - sistemas de recomendação,
  - identificação de padrões (segurança), ...
  - outros exemplos?

Na disciplina veremos métodos de pesquisa em **memória principal** e **memória secundária** e analisaremos suas **complexidades** (tempo e espaço).

Parece simples...

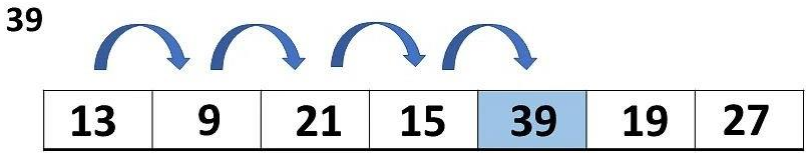
- Vamos considerar um algoritmo de busca para encontrar um inteiro i em um conjunto de N valores.



- Qual a complexidade?

Parece simples...

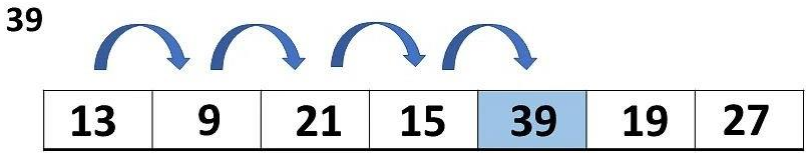
- Vamos considerar um algoritmo de busca para encontrar um inteiro i em um conjunto de N valores.



- Qual a complexidade?
- Faria diferença conhecer características dos dados do conjunto?
  - Mínimo, máximo, média, mediana;
  - Saber se só tem números pares, primos, múltiplos de 3, ...

Parece simples...

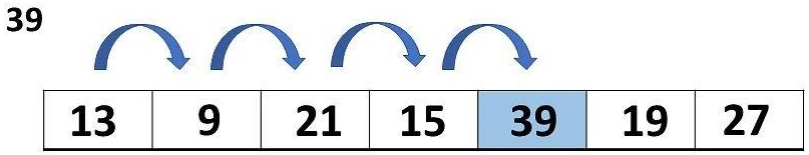
- Vamos considerar um algoritmo de busca para encontrar um inteiro i em um conjunto de N valores.



- Qual a complexidade?
- Faria diferença conhecer características dos dados do conjunto?
  - Mínimo, máximo, média, mediana;
  - Saber se só tem números pares, primos, múltiplos de 3, ...
- Faria diferença o conjunto estar semi-ordenado? e ordenado?

Parece simples...

- Vamos considerar um algoritmo de busca para encontrar um inteiro i em um conjunto de N valores.



- Qual a complexidade?
- Faria diferença conhecer características dos dados do conjunto?
  - Mínimo, máximo, média, mediana;
  - Saber se só tem números pares, primos, múltiplos de 3, ...
- Faria diferença o conjunto estar semi-ordenado? e ordenado?
- Faria diferença buscar o elemento pela 2ª vez? e se a frequência de busca do elemento aumentasse muito?

Agenda

- Retomando conceitos
- Introdução à pesquisa de dados
- Pesquisa sequencial
- Pesquisa binária
- Métodos adicionais
- *Hands-on*

Pesquisa de Dados

- Estudo de como recuperar informação a partir de uma grande massa de informação previamente armazenada:
  - Processo de localizar um item específico em um conjunto de dados;
- Também conhecido como "busca de dados" (*searching*);
- Elementos-chave:
  - **Conjunto de dados:** onde a pesquisa é realizada.
  - **Chave de busca (item de pesquisa):** o que você está procurando.
  - **Algoritmo de pesquisa:** como você procura (método).

Pesquisa de Dados

- A **informação é dividida em registros** (entradas associadas);
- **Cada registro possui uma chave** para ser usada na pesquisa;
- **Objetivo** da pesquisa: **encontrar uma ou mais ocorrências** de registros com chaves iguais à chave de pesquisa;
- Em muitas LPs vamos ter uma variedade de métodos:
  - **.find**(), **.findAll**(), .includes()
  - **indexOf**(), **lastIndexOf**()
  - some(), every()
  - index(), count(), any(), all()

Pesquisa de Dados

- **Aplicações específicas de algoritmos de busca incluem:**
  - Problemas de **otimização combinatória**, como:
    - roteamento de veículos (problema do caminho mais curto);
    - problema da mochila (*knapsack problem*);
    - problema de agendamento em intervalos.
  - Problemas de satisfação de restrições, como:
    - problema de coloração do mapa;
    - **preenchimento do *sudoku*** ou palavras cruzadas.

Pesquisa de Dados

- **Aplicações específicas de algoritmos de busca incluem:**
  - Na teoria dos jogos, escolher o melhor movimento a ser feito a seguir (e.g., algoritmo minimax);
  - Encontrar uma combinação ou senha dentro do conjunto de possibilidades;
  - Recuperar um registro de um banco de dados;
  - Encontrar o valor máximo ou mínimo em uma lista ou array;
  - Verificar se um determinado valor está presente em um conjunto de valores.

Pesquisa de Dados

- **Operações comuns:**
  - Encontrar um item;
  - Verificar a existência de um item;
  - Contar ocorrências de um item;
  - Encontrar todos os itens que satisfazem uma condição.

- **Aplicações práticas:**
  - Bancos de Dados;
  - Sistemas de Recomendação;
  - Mecanismos de Busca.

Importância da Pesquisa de Dados

- **Eficiência:**
  - **Tempo é dinheiro:** algoritmos eficientes economizam tempo e recursos.
- **Escalabilidade:**
  - A pesquisa eficiente é crucial à medida que os conjuntos de dados crescem (e.g., cenários de *big data*).
- **Casos de uso:**
  - E-commerce: encontrar produtos rapidamente;
  - Serviços de Saúde: acesso rápido a registros médicos;
  - Segurança: detecção rápida de atividades suspeitas;
  - ...
- **Impacto na experiência do usuário (UX):**
  - Velocidade e precisão na busca melhoram a satisfação do usuário;
  - Exemplo: Mecanismos de busca como Google.

Pesquisa de Dados

- Por que busca sequencial não é ideial e precisamos de algoritmos eficientes?
  - Custo para percorrer todo conjunto pode ser tornar significativo;
  - Otimizar complexidade e tempo de execução ou consumo de recursos é essencial em muitos cenários.

Complexidade de espaço

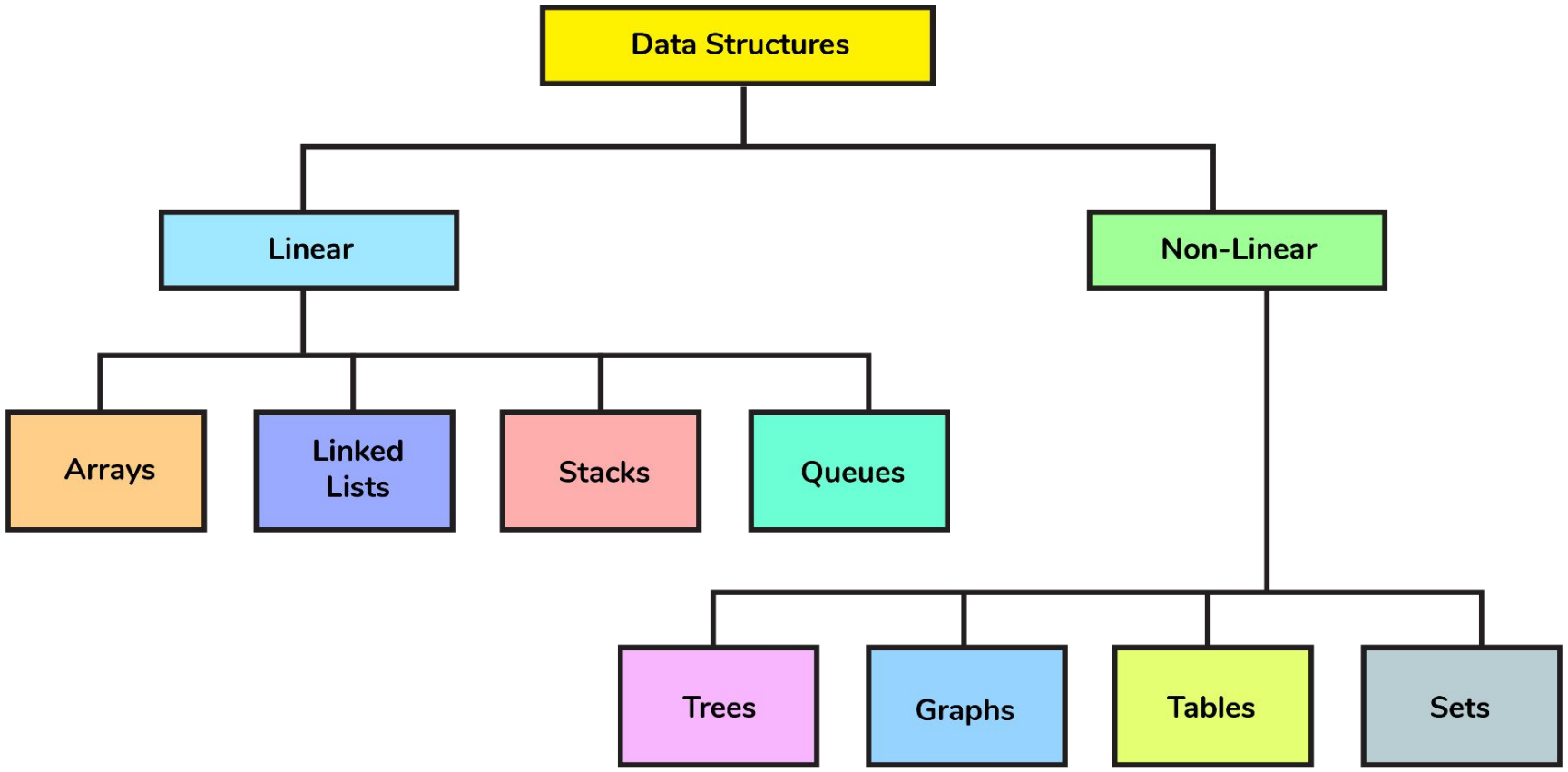
- Quando estávamos falando de **ordenação de dados**, tínhamos, comumente, como complexidades de espaço (espaço adicional) no pior caso:
  - O(1) → ordenação *in-place*;
  - O(n) → ordenação *out-of-place*;
  - O(n + k) → para alguns métodos de ordenação linear.
- Já para a **pesquisa de dados**, veremos métodos com:
  - O(1) → sequencial, binária;
  - O(n) → *hashing*, árvores.

Complexidade de tempo

- Quando estávamos falando de **ordenação de dados**, tínhamos como limites inferiores para complexidades de tempo:
  - O(n log n) → ordenação por comparação no caso médio. O(n) no melhor caso para alguns métodos;
  - O(n) → ordenação linear (com restrições sobre a entrada).
- Já para a **pesquisa de dados**, complexidade linear **não** é sinônimo de bom algoritmo, muito pelo contrário...
  - Melhor caso esperado é O(1)
  - Buscamos um caso médio < O(n), tal como O(log n)
  - Lembrando que:

    $1 \ll \log n \ll n \ll n \log n \ll n^2 \ll n^3 \ll n!$

Pesquisa de Dados



Agenda

- Retomando conceitos
- Introdução à pesquisa de dados
- Pesquisa sequencial
- Pesquisa binária
- Métodos adicionais
- *Hands-on*

Pesquisa sequencial

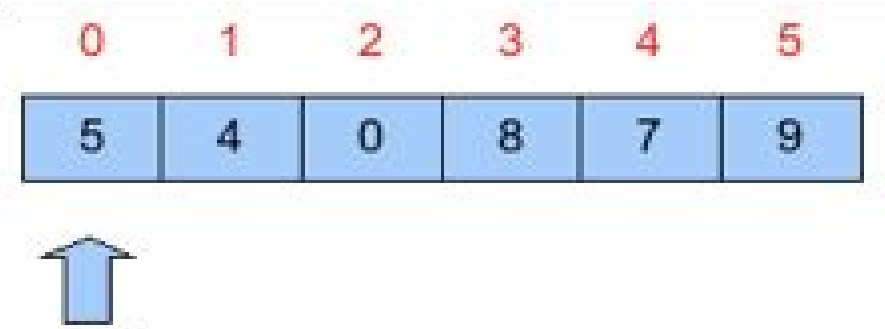
- Também chamado de pesquisa linear;
- Método de pesquisa mais simples: a partir do primeiro registro, pesquise seqüencialmente até encontrar a chave procurada; então pare;
- A implementação clássica não suporta mais de um registro com uma mesma chave, pois retorna o primeiro encontrado.

- Se o conjunto de entrada estiver ordenado, podemos otimizar o algoritmo:
  - Se elemento a ser buscado < vet[i], interrompemos a busca e retornamos (elemento não encontrado).
  - Se elemento < vet[0] ou > vet[n - 1], elemento $\notin$ vet.

Indicado para problemas de pesquisa onde n é pequeno (e.g., n < 25).

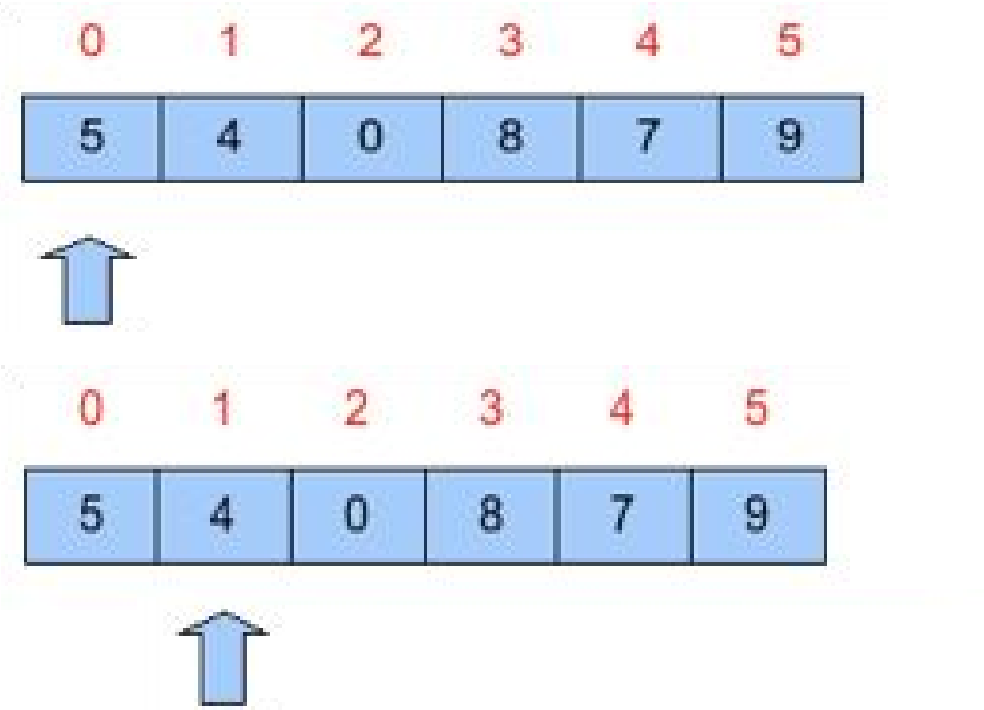
Pesquisa sequencial

Busca pela chave 0 (que está na posição de índice 2 do vetor)



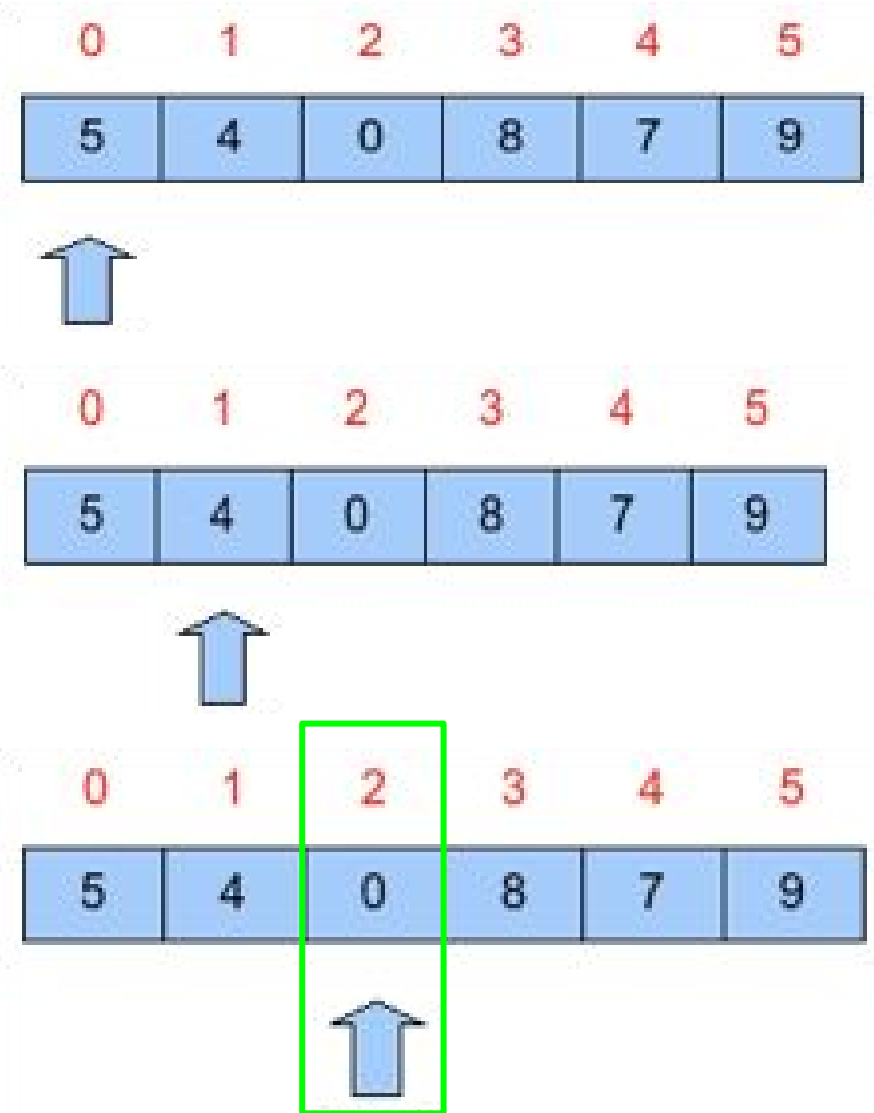
Pesquisa sequencial

Busca pela chave 0 (que está na posição de índice 2 do vetor)



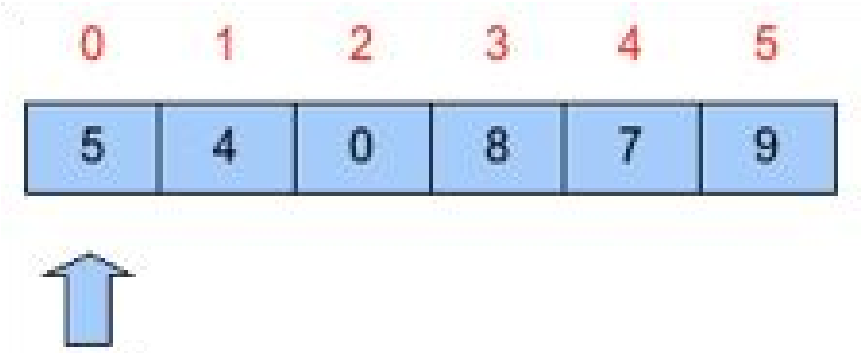
Pesquisa sequencial

Busca pela chave 0 (que está na posição de índice 2 do vetor)



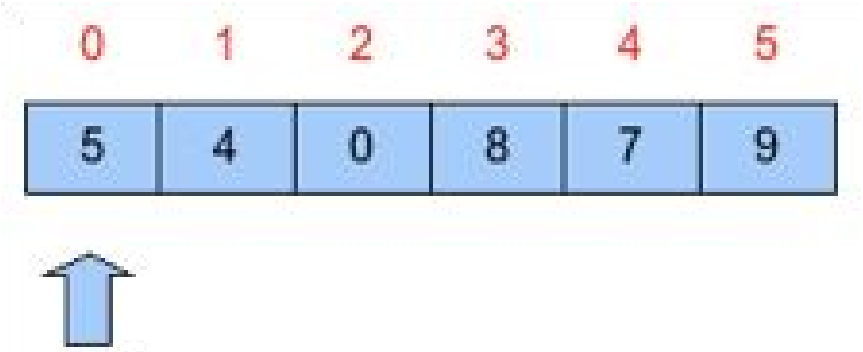
Pesquisa sequencial

Agora procurando pelo valor 3 **(não existe no vetor)**

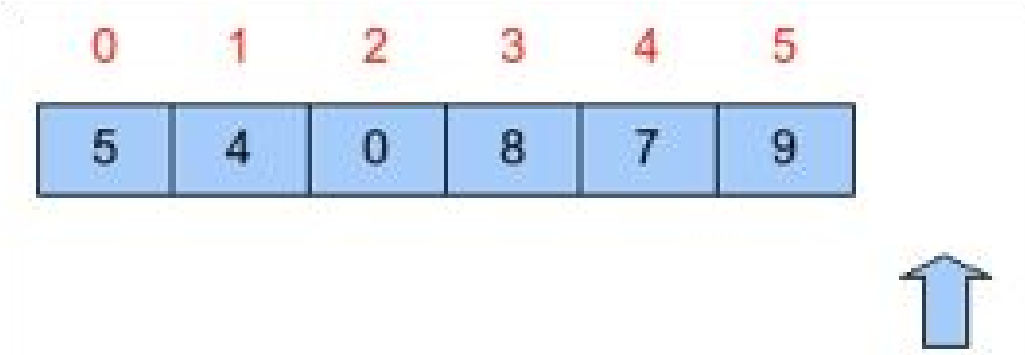


Pesquisa sequencial

Agora procurando pelo valor 3 **(não existe no vetor)**



...



Pesquisa sequencial

Considerando o número de comparações C(n)

- Pesquisa com sucesso:
  - melhor caso: $C(n) = 1 \rightarrow O(1)$
  - pior caso: $C(n) = n \rightarrow O(n)$
  - caso médio: $C(n) = (n + 1) / 2 \rightarrow O(n)$
- Pesquisa sem sucesso:
  - $C(n) = n + 1 \rightarrow O(n)$

Em termos de ordem de complexidade de tempo (pior e caso médio), continuamos a ter uma variação linear, ou seja, O(n), pois O(k *n), onde k é uma constante relativamente pequena, é igual a O(n).

Já a complexidade de espaço é O(1)

Agenda

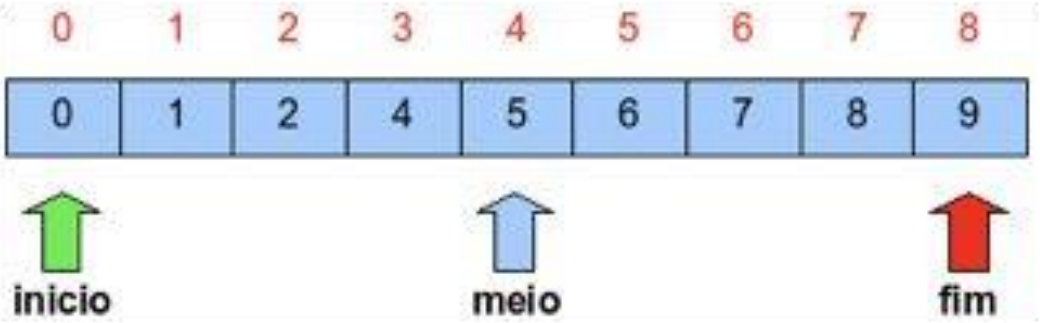
- Retomando conceitos
- Introdução à pesquisa de dados
- Pesquisa sequencial
- Pesquisa binária
- Métodos adicionais
- *Hands-on*

Pesquisa binária

- Para aplicar a busca binária, precisamos que os registros estejam mantidos em ordem em uma tabela (estrutura tabular qualquer, incluindo *arrays*);
- Para saber se uma chave está presente no vetor:
  - Compare a chave buscada com o registro que está na posição do meio do vetor, ou seja, na posição (esq + dir) / 2;
  - Se a chave de busca é **menor** que o registro do meio, então o item de pesquisa está na primeira metade (esquerda) do vetor;
  - Se a chave de busca é **maior** que o registro do meio, então o item de pesquisa está na segunda metade (direita) do vetor;
  - Se a chave de busca é **igual** ao registro do meio, retorne o registro do meio;
  - Repita até que a chave seja encontrada ou que se constate que a chave não existe no vetor (caso base: parte do vetor com tamanho 0).

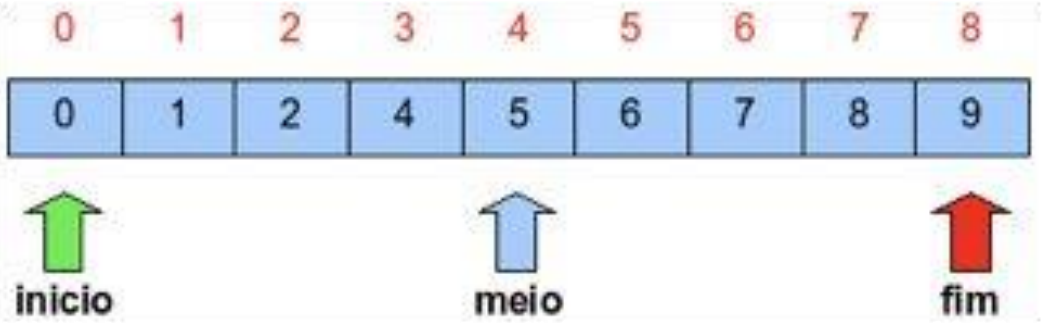
Pesquisa binária

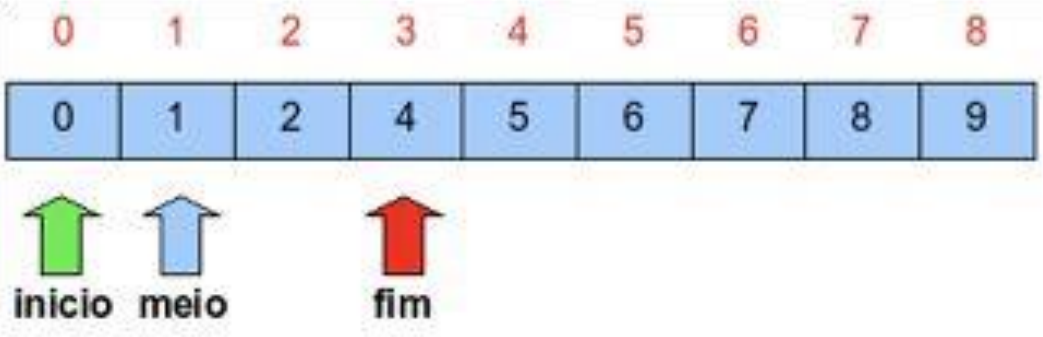
Busca pela chave 4 (que está na posição de índice 3 do vetor)



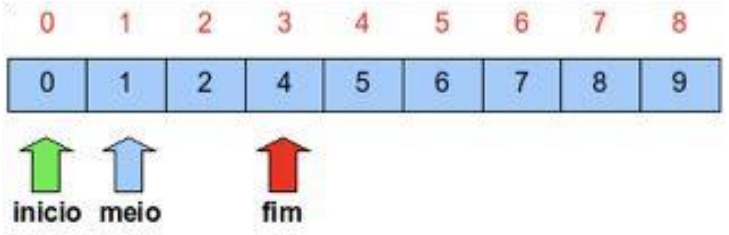
Pesquisa binária

Busca pela chave 4 (que está na posição de índice 3 do vetor)



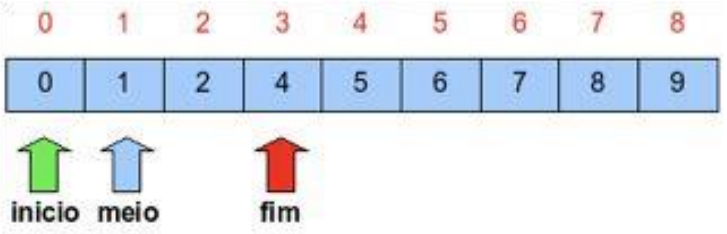


Pesquisa binária

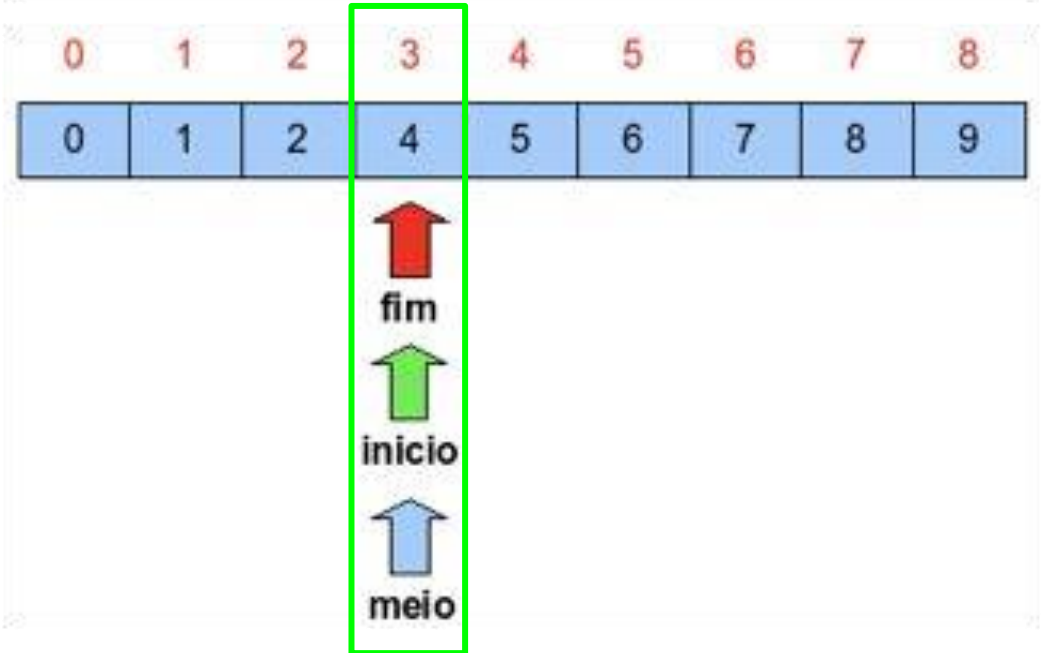


Busca pela chave 4 (que está na posição de índice 3 do vetor)

Pesquisa binária



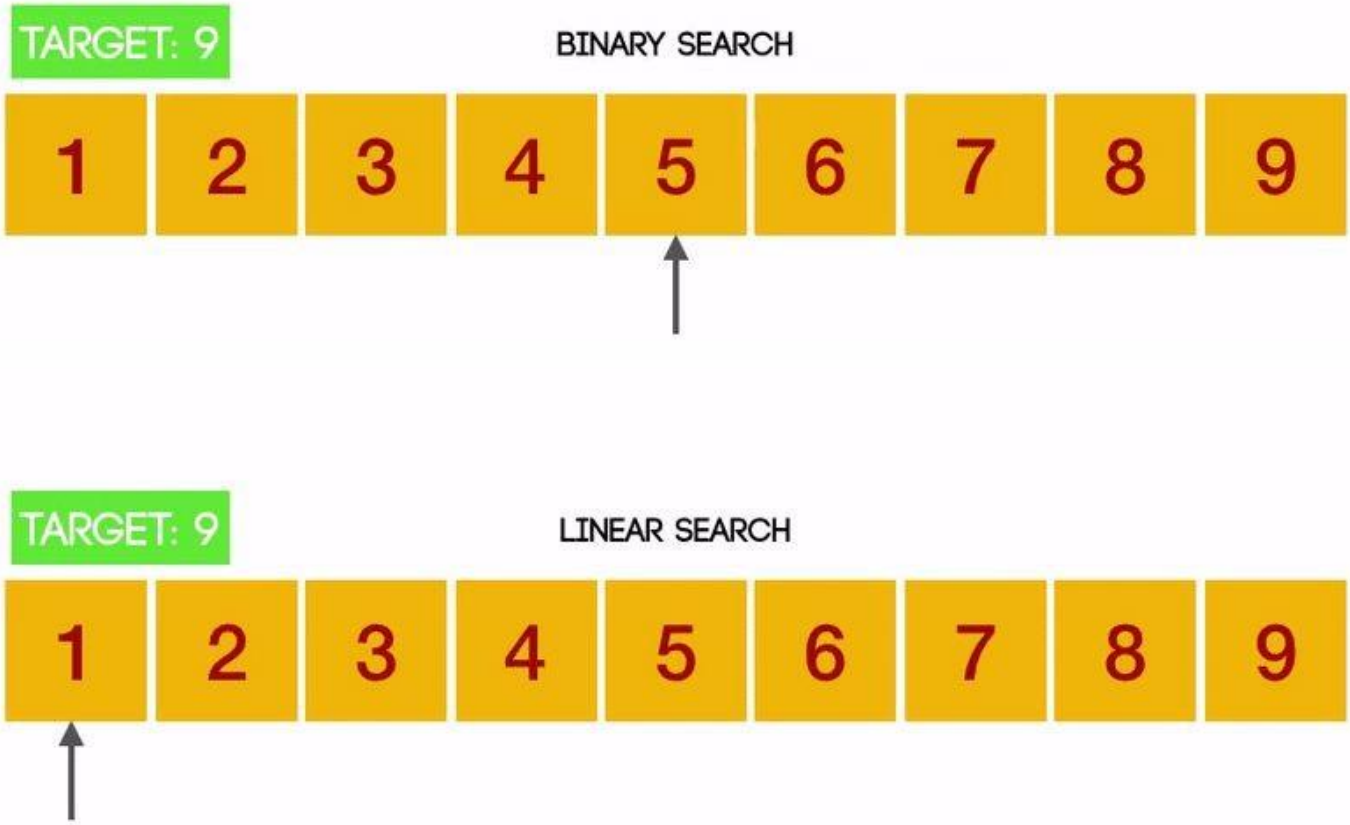
Busca pela chave 4 (que está na posição de índice 3 do vetor)



Pesquisa binária

- A cada iteração do algoritmo, o tamanho da vetor é dividido ao meio;
- Logo, o número de vezes que o tamanho do vetor é dividido ao meio é cerca de **log n**. Assim sua complexidade de tempo é O(log n)
  - Complexidade de espaço é O(1)
- **Ressalva:** o custo para manter o conjunto ordenado é alto: cada inserção na posição p implica no deslocamento dos registros a partir da posição p para as posições seguintes (*shift right*).
- Consequentemente, a pesquisa binária não deve ser usada em aplicações muito dinâmicas.

Pesquisa binária x sequencial



Agenda

- Retomando conceitos
- Introdução à pesquisa de dados
- Pesquisa sequencial
- Pesquisa binária
- Métodos adicionais
- *Hands-on*

Pesquisa sequencial com transposição

- ***Transpose sequential search*** é uma variação da pesquisa sequencial básica;
- Quando um item é encontrado, ele é trocado com o item anteriormente acessado;
- A ideia dessa técnica é que itens **frequentemente acessados** sejam movidos gradualmente para o início da lista, tornando buscas futuras mais rápidas.

  Suponha vet = [5, 7, 9, 2, 8, 1] e busca pelo 2 → 4ª posição (vet[3])

  Com a pesquisa sequencial com transposição, trocamos o número 2 com o número 9, que é o item anteriormente acessado, assim vet = [5, 7, 2, 9, 8, 1]

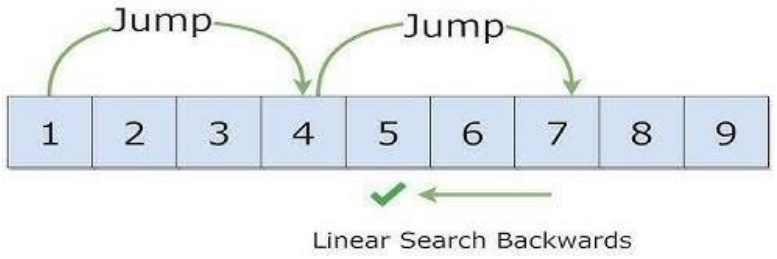
  Se o número 2 for frequentemente pesquisado, ele continuará se movendo para o início da lista, tornando a pesquisa mais rápida para esse item em consultas subsequentes.

A vantagem dessa técnica é que ela adapta o conjunto com base nos **padrões de acesso**, beneficiando itens frequentemente pesquisados.

A eficácia real dessa técnica pode variar dependendo da natureza dos dados e dos padrões de acesso (explora localidade temporal).

Pesquisa por salto

- ***Jump search*** é um algoritmo de busca para vetores ordenados. A ideia básica é verificar menos elementos (do que a pesquisa linear), avançando em etapas fixas ou pulando alguns elementos em vez de pesquisar todos os elementos sequencialmente:
  - Ao invés vez de percorrer cada elemento da lista, divide-se a lista em blocos de tamanho fixo (comumente √n) e verifica-se o último elemento de cada bloco. Uma vez que o bloco de busca é identificado, uma pesquisa linear é realizada dentro desse bloco.
- Complexidade de tempo: O(√n)
- Uso: em geral é mais eficiente que a pesquisa linear para listas grandes, mas menos eficiente que a pesquisa binária.



Pesquisa *Fibonacci*

- *Fibonacci search* é usado para pesquisa em um conjunto ordenado usando um algoritmo de divisão e conquista baseado na **sequência de *Fibonacci***;
- Ao contrário do algoritmo de busca binária que divide o vetor em duas partes de tamanhos iguais, a busca de *Fibonacci* divide o vetor em duas partes que possuem tamanhos que são números consecutivos da sequência.
- Complexidade de tempo média também é O(log n) (≃ -4% comparações)

Chave de busca = **80**
n = 7

50	58	64	74	80	85	90	99
0	1	2	3	4	5	6	7

Menor número da sequência ≥ n é 8

F(m) = 8 → F(m-1) = 5, F(m-2) = 3, offset= -1

i = min(offset + F(m-2), n - 1) = 2

50	58	64	74	80	85	90	99
0	1	2	3	4	5	6	7

v[i] = v[2] = 64, e 64 < 80

F(m) = 5 → F(m-1) = 3, F(m-2) = 2, offset = 2

i = min(offset + F(m-2), n - 1) = 4

50	58	64	74	80	85	90	99
0	1	2	3	4	5	6	7

v[i] = v[4] = 80 ← encontrou!

Pesquisa por interpolação

- ***Interpolation search*** é uma melhoria da pesquisa binária para situações em que os valores armazenados são numericamente **ordenados** e **distribuídos de forma uniforme**;
- Em vez de dividir a lista ao meio, **estima-se a posição do item** desejado com base na diferença entre o valor desejado e o primeiro valor na lista;

$$meio = ini + ((fim - ini) * (chave - v[ini])) / (v[fim] - v[ini])$$

- Complexidade de Tempo:
  - **O(log(log n))**, mas pode degradar para O(n) se os valores não estiverem uniformemente distribuídos.
- Uso: é útil para listas numericamente ordenadas com distribuição uniforme.

2	4	6	8	10	12	14	16
0 ini	1	2	3	4	5	6	7 fim

Busca pelo 4

meio = 0 + ((7-0) * (4 - 2)) / (16-2) = 1

v[meio] = v[1] = 4 ← encontrou!

Bibliografia

- **Básica**

  CORMEN, T. H. et al. **Algoritmos: Teoria e Prática**. 3ª ed. Rio de Janeiro: Elsevier, 2012.

  CELES, W.; Cerqueira, R.; Rangel J. L. **Introdução a Estrutura de Dados**. Rio de Janeiro: Elsevier. 2004.

  GOODRICH, M.; TAMASIA, R. **Projeto de Algoritmos: fundamentos, análise e exemplos da internet**. Porto Alegre: Bookman. 2004.

- **Complementar**

  LAMBERT, K. A. **Fundamentos de python: estrutura de dados**. São Paulo: Cengage Learning. 2022.

  BHARGAVA, A. Y. **Entendendo algoritmos: um guia ilustrado para programadores e outros curiosos**. São Paulo: Novatec Editora. 2017.

  SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3. ed. São Paulo: LTC, 2010.

  TOSCANI, L. V.; VELOSO, P. S. **Complexidade de Algoritmos**. 3ª ed. Porto Alegre: Bookman. 2012.

  ARAÚJO, E. C. **Algoritmos: fundamento e prática**. 3ª ed. Florianópolis: Visual Books, 2006.

Agenda

- Retomando conceitos
- Introdução à pesquisa de dados
- Pesquisa sequencial
- Pesquisa binária
- Métodos adicionais
- *Hands-on*

Atividade Individual

1. Implemente o algoritmo de busca binária:
   a. de forma iterativa;
   b. de forma recursiva.

2. Implemente os métodos adicionais vistos em aula:
   a. Pesquisa por salto;
   b. Pesquisa Fibonacci;

3. Faça um teste de desempenho considerando um conjunto de 10k números.

   Questões de Reflexão:

   a. Qual método teve o menor número de comparações em média?
   b. Em quais situações você acha que cada método seria mais apropriado?
   c. Como a ordenação da lista afeta a eficiência de cada método?