Algoritmos de pesquisa

emanoelim@utfpr.edu.br

Algoritmos de pesquisa

- Têm como objetivo buscar um determinado elemento dentro de um conjunto de dados (array, lista, árvore, etc).

Pesquisa sequencial

- É o método de pesquisa mais simples que existe: percorre um arranjo sequencialmente a partir do primeiro registro até encontrar a chave buscada ou até chegar ao final do array.

Pesquisa sequencial

- O algoritmo abaixo é um exemplo de busca sequencial, retornando o índice onde a chave foi encontrada ou -1 se a chave não foi encontrada:

```
int busca_sequencial(int v[], int n, int chave) {
  int i;
  for(i = 0; i < n; i++)
    if(v[i] == chave)
      return i;
  return -1;
}
```

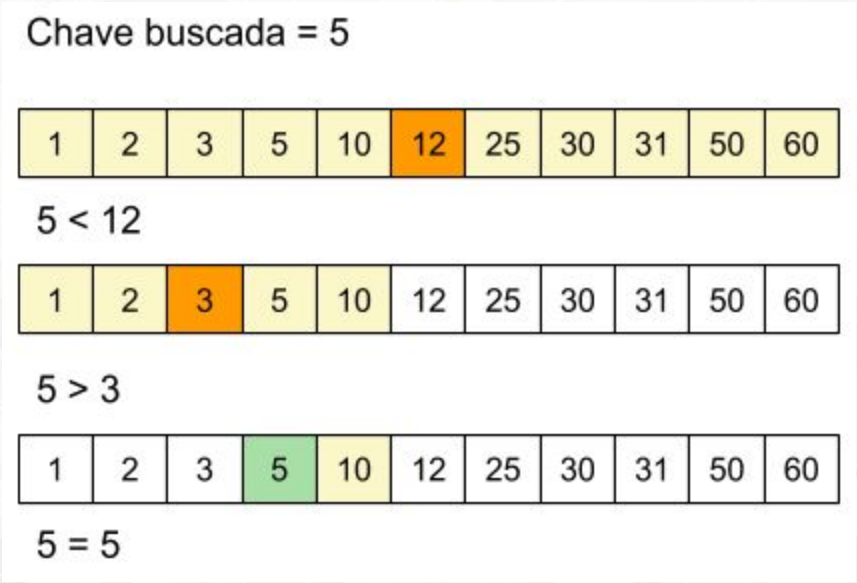
Pesquisa sequencial

- **Melhor caso:** a chave buscada está no primeiro índice do array. Nesse caso o custo seria O(1).
- **Pior caso:** a chave buscada está no último índice do array ou então a chave não existe, pois o laço precisaria ser executado todas as vezes, levando a um custo O(n).

Pesquisa binária

- Aplica-se quando o array já está ordenado.
  - A chave é comparada com o item que está no meio do array.
  - Se a chave for igual ao item, a busca termina.
  - Se a chave for menor que o item, repete a busca com a metade esquerda do array.
  - Se a chave for maior que o item, repete a busca com a metade direita do array.

Pesquisa binária



Pesquisa binária

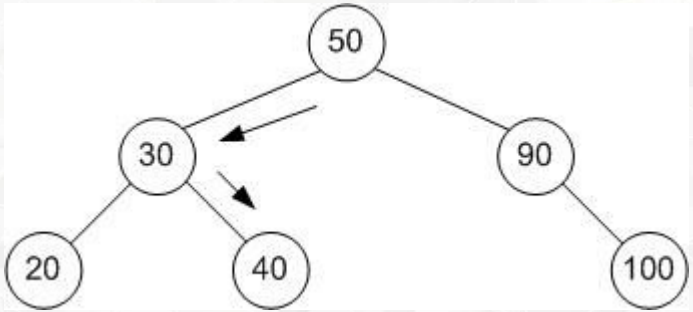
```
int busca_binaria(int v[], int n, int chave) {
  int p = 0, r = n - 1;
  int q;
  while(p <= r) {
    q = (p + r) / 2;
    if(v[q] == chave)
      return q;
    else {
      if(v[q] > chave)
        r = q - 1;
      else
        p = q + 1;
    }
  }
  return -1;
}
```

Pesquisa binária

- **Melhor caso:** a chave buscada está no índice do meio, levando a complexidade O(1).
- **Pior caso:** a chave buscada não existe. A complexidade será O(log n).

Pesquisa em árvore binária

- Os dados ficam organizados no formato de uma árvore binária, onde todos os nós da subárvore esquerda possuem valor menor do que o nó raiz, e todos os nós da subárvore direita possuem valor maior do que o nó raiz:



Nesse tipo de estrutura é possível aplicar uma busca binária. Para buscar o 40, por ex.:

- 40 < 50, então caminhamos para o lado esquerdo da árvore, encontrando o 30;
- 40 > 30, então caminhamos para o lado direito da árvore, encontrando o 40.

Pesquisa em árvore binária

- Este tipo de estrutura será estudados em algoritmos 2:
  - Criação da árvore;
  - Inserção;
  - Remoção;
  - Busca;
  - Etc.

Hashing

- Todos os métodos anteriores são baseados na comparação de uma chave com os itens do array.
- O método de hashing associa chaves a valores.
- É como se cada chave fosse o “índice” de um item do array, então a partir de uma chave seria possível acessar diretamente o item associado a ela.

Hashing

- Algumas linguagens já trazem isso implementado nativamente, como a linguagem Python e seus dicionários:

```
>>> quadrados = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81} # criação do dicionário
>>> quadrados[5] # acesso direto a um item do dicionário
25
>>> idades = {"maria": 25, "josé": 20, "marta": 30, "suzana": 45}
>>> idades["suzana"]
45
```

Hashing

- Tabelas de Hash serão estudadas após a segunda avaliação.

Atividades

- É possível perceber que o problema da busca binária tem estrutura recursiva, pois o array vai sendo sempre dividido por 2 até encontrar o item buscado ou até sobrar um subvetor de apenas 1 item. Escreva uma versão recursiva para o algoritmo de pesquisa binária.

Referências

- Ziviani, N., “Projeto de algoritmos - com implementações em Pascal e C”. 3ª ed. São Paulo, Cengage Learning, 2010.
- Schildt, H., “C completo e total”. 3ª ed. São Paulo, Makron Books, 1996.
- Cormen, T. H., “Desmistificando algoritmos”. 1ª ed. Rio de Janeiro, Elsevier, 2014.
- Visualização dos algoritmos de busca: https://www.cs.usfca.edu/~galles/visualization/Search.html
- Visualização da árvore binária: https://www.cs.usfca.edu/~galles/visualization/BST.html