

Algoritmos de Pesquisa

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

- Busca Binária
- Busca por Interpolação
- Árvores de Busca
 - Árvores Binárias de Busca

Busca Binária

- Os dados devem estar ordenados em um arranjo
- A busca começa no meio da tabela
 - Se igual, busca bem-sucedida
 - Se menor, busca-se na metade inferior do arranjo
 - Se maior, busca-se na metade superior do arranjo
- Em cada passo, o tamanho do arranjo em que se busca é dividido por 2

- Exemplo: busca pelo elemento 25

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

- Exemplo: busca pelo elemento 25

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

- Exemplo: busca pelo elemento 25

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

- Exemplo: busca pelo elemento 25

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

- Implementação recursiva da pesquisa binária

```
static int busca_bin(int x, int v[], int ini, int fim){
    int meio;

    if ((ini == fim) && (x == v[ini]))
        return ini;
    else if (ini < fim){
        meio = (ini + fim) / 2;

        if (x == v[meio])
            return meio;
        else if (x < v[meio])
            return busca_bin(x, v, ini, meio - 1);
        else
            return busca_bin(x, v, meio + 1, fim);
    }else
        return -1;
}
```

- Implementação recursiva da pesquisa binária

```
int busca_binaria(int x, int v[], int n){  
    return busca_bin(v, 0, n - 1);  
}
```

- Complexidade: $O(\log n)$
 - Em cada comparação, o número de candidatos é dividido pela metade
- Vantagens
 - Eficiência da busca
 - Fácil implementação
- Desvantagens
 - Nem todo arranjo está ordenado
 - Exige o uso de um arranjo para armazenar os dados
 - Inserção e remoção de elementos são ineficientes

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

- Exercício: com base no código anterior, implementar a versão iterativa do método de busca binária.

Busca por Interpolação

Busca por Interpolação

- Derivação da busca binária
- Se as chaves estiverem uniformemente distribuídas, a busca por interpolação pode ainda mais eficiente que a busca binária
- Com chaves uniformemente distribuídas, pode-se esperar que x esteja aproximadamente na posição
 - $meio = ini + (fim - ini) * \left(\frac{x - v[ini]}{v[fim] - v[ini]} \right)$
- ini e fim são redefinidos da mesma forma em relação à busca binária

Busca por Interpolação

- Exemplo: busca pelo elemento 25

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

Busca por Interpolação

- Exemplo: busca pelo elemento 25

| | | | | | | | |
|---------|------|----|----|----|----|----|---------|
| Inf = 1 | meio | | | | | | Sup = 8 |
| 12 | 25 | 33 | 37 | 48 | 57 | 86 | 92 |

$$\text{meio} = 1 + (8 - 1) * ((25 - 12)/(92-12)) = 2$$

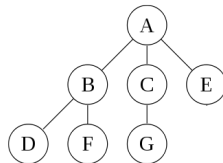
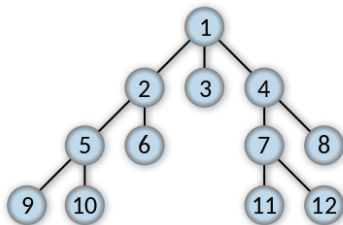
- Complexidade
 - Para chaves uniformemente distribuídas, raramente precisará mais que $\log(\log n)$ comparações
 - Logo, a complexidade será de $O(\log(\log n))$
 - Se as chaves não estiverem uniformemente distribuídas, a busca por interpolação pode ser tão ruim quanto uma busca sequencial, ou seja, ter o custo de $O(n)$
- 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"

Árvores de Busca

- **Nota: árvore é tema da disciplina "Algoritmos e Estrutura de Dados 2 (AE23CP)"**
- Árvore é uma estrutura de dados muito eficiente para armazenamento de informação (incluindo as que devem ser organizadas de forma hierárquica)
- Estrutura de dados não linear
- Aplicações
 - Sistemas de arquivos, processamento de língua natural, inteligência artificial, etc

Árvores de Busca

- Árvore
 - Conjunto finito de nós
 - Existe um nó raiz r com zero ou mais sub-árvores
 - Os nós internos são filhos de r
 - Cada sub-árvore também possui um nó raiz, que é descendente (nó filho) de r
 - Nós folhas são os nós que não possuem filhos

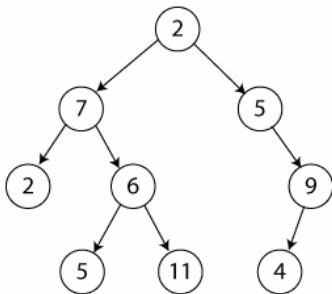


- Tipos de árvores
 - Árvore binária
 - Árvore binária de busca
 - AVL
 - Árvore trie
 - Árvore PATRICIA
 - Árvore vermelha-preta (rubro-negra)
 - Árvore B
 - Etc.

Árvores de Busca

Árvores Binárias de Busca

- Árvore binária
 - Árvore em que cada nó contém um ou dois filhos
 - As sub-árvores também contêm entre um e dois nós, exceto se são nós-folhas



Árvores de Busca

Árvores Binárias de Busca

- Árvore binária
 - Podemos representar uma árvore por meio de um registro (*struct*) de três campos:
 - Informação
 - Sub-árvore esquerda
 - Sub-árvore direita

| left | item | right |
|-----------------|-------------------|----------------|
| esquerda | informação | direita |

- Estrutura de dados para a representação de uma árvore binária:

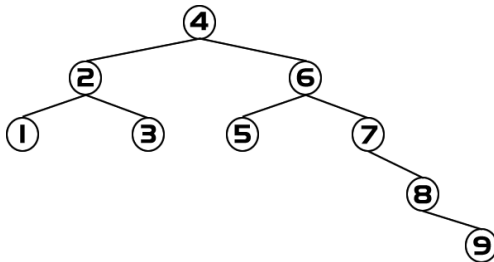
```
typedef struct Node Node;
```

```
struct Node{  
    int item;  
    Node *right;  
    Node *left;  
};
```

Árvores de Busca

Árvores Binárias de Busca

- Árvore binária de busca é uma árvore binária com propriedades de ordenação
 - Todos os nós com chaves menores que a da raiz ficam do lado esquerdo
 - Todos os nós com chaves maiores que a da raiz ficam do lado direito



Árvores de Busca

Árvores Binárias de Busca

- Principais operações em uma árvore binária de busca:
 - Pesquisa
 - Inserção
 - Remoção

Árvores de Busca

Árvores Binárias de Busca

- A busca é iniciada pela raiz da árvore
 - Caso o valor seja encontrado, o nó é retornado
 - Caso contrário:
 - É realizada uma chamada recursiva para a sub-árvore esquerda se o valor procurado é menor que o item verificado
 - É realizada uma chamada recursiva para a sub-árvore direita se o valor procurado é maior

- Implementação do método de busca para árvores binárias de busca

```
Node* search(Node* tree, int value){  
    if (tree != NULL)  
        if (tree->item == value)  
            return tree;  
        else if (tree->item > value)  
            return search(tree->left, value);  
        else  
            return search(tree->right, value);  
    else  
        return NULL;  
}
```

Árvores de Busca

Árvores Binárias de Busca

- Complexidade da operação de busca
 - Melhor caso: $O(1)$
 - Caso médio: $O(\log n)$
 - Pior caso: $O(n)$
- A complexidade das operações em árvores é diretamente relacionada com o balanceamento da árvore (será visto em "Algoritmos e Estrutura de Dados 2")



Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.



Horowitz, E., Sahni, S. Rajasekaran, S.
Computer Algorithms.
Computer Science Press, 1998.



Rosa, J. L. G.
Métodos de Busca. SCE-181 – Introdução à Ciência da
Computação II.
Slides. Ciência de Computação. ICMC/USP, 2018.



Ziviani, N.
Projeto de Algoritmos - com implementações em Java e C++.
Thomson, 2007.