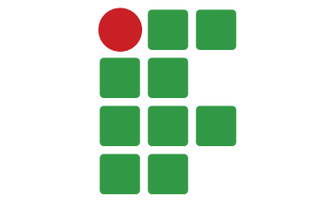
06-Busca

BCC - Estruturas de Dados

Prof. Dr. Paulo César Rodacki Gomes
paulo.gomes@ifc.edu.br



Blumenau, 2023

Tópicos

- Busca em vetor
  - Busca linear
  - Busca binária
  - Algoritmo genérico
- Árvore binária de busca
  - Apresentação
  - Operações em árvores binárias de busca
  - Árvores balanceadas

Busca em Vetor

- Busca em vetor:
  - Entrada: array **vet** com n elementos e elemento **elem** a ser encontrado
  - Saída:
    i se o elemento elem ocorrem em vet[i]
    -1 se o elemento não se encontra no vetor

Busca Linear

Percorre o array **vet**, elemento a elemento verificando se **elem** é igual a um dos elementos de **vet**

Algoritmo: int busca(int vet[], int elem)

```
int n ← v.length;
// procura elem
para i ← 0 até n-1 faça
    se elem == vet[i] então
        retorna i;
// caso não achou elem
retorna -1;
```

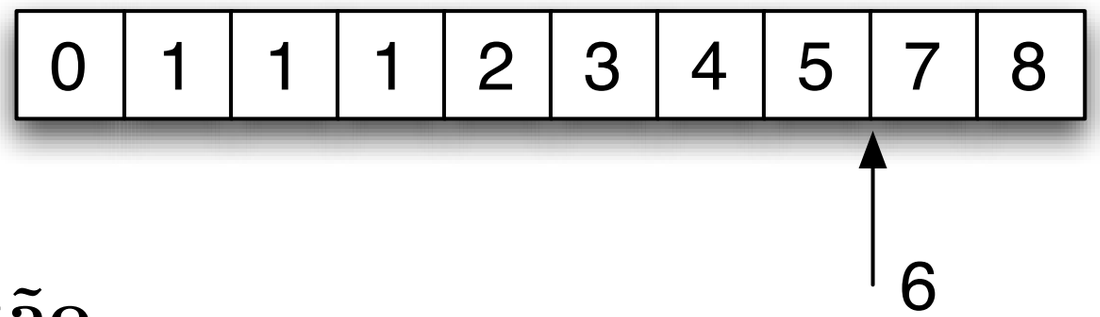
Busca Linear

- Análise da Busca Linear:
  - pior caso:
    - n comparações, onde n representa o número de elementos do vetor
      - desempenho computacional varia linearmente em relação ao tamanho do problema (algoritmo de busca *linear*)
    - complexidade: O(n)
  - caso médio:
    - n/2 comparações
      - desempenho computacional continua variando linearmente em relação ao tamanho do problema
    - complexidade: O(n)

Busca Linear - vetor em ordem crescente

Algoritmo: int buscaLinearOrdenada(int vet[], int elem)

```
int n ← v.length;
// procura elem
para i ← 0 até n-1 faça
    se elem == vet[i] então
        retorna i;
    senão
        // interrompe a busca
        se elem < vet[i] então
            retorna -1;
// caso não achou elem
retorna -1;
```



Busca Linear

- Análise de busca linear (vetor ordenado):
  - caso o elemento procurado não pertença ao vetor, a busca linear com vetor ordenado apresenta um desempenho ligeiramente superior à busca linear
  - pior caso:
    - algoritmo continua sendo linear
    - complexidade: O(n)

Busca Binária

- entrada: vetor vet com n elementos, ordenado
  elemento elem
- saída: n se o elemento elem ocorre em vet[n]
  -1 se o elemento não se encontra no vetor
- procedimento:
  - compare elem com o elemento do meio de vet
  - se elem for menor, pesquise na primeira metade do vetor
  - se elem for maior, pesquise na segunda parte do vetor
  - se for igual, retorne a posição
  - continue o procedimento, subdividindo a parte de interesse,
    até encontrar o elemento ou chegar a uma parte do vetor com tamanho 0

Algoritmo: int buscaBinaria(int vet[], int elem)

```
// no início consideramos todo o vetor
int ini ← 0;
int fim ← n − 1;
int meio;
// enquanto a parte restante for maior que zero
enquanto ini ≤ fim faça
    meio ← (ini + fim)/2;
    se elem < vet[meio] então
        // ajusta posição final
        fim ← meio − 1;
    senão
        se elem > vet[meio] então
            // ajusta posição inicial
            ini ← meio + 1;
        senão
            // elemento encontrado
            retorna meio;
// caso não achou elem
retorna -1;
```

Busca Binária

- Análise de busca binária
  - pior caso: O(log n)
    - elemento não ocorre no vetor
    - 2 comparações são realizadas a cada ciclo
    - a cada repetição, a parte considerada na busca é dividida à metade
    - logo, no pior caso, são necessárias log n repetições

Repetição	Tamanho do problema
1	*n*
2	*n/2*
3	*n/4*
...	...
log n	*1*

Busca Binária

- Busca binária – implementação recursiva:
  - dois casos a tratar:
    - busca deve continuar na primeira metade do vetor:
      - chamada recursiva com parâmetros:
        - » o número de elementos da primeira parte restante
        - » o mesmo ponteiro para o primeiro elemento (pois a primeira parte tem o mesmo primeiro elemento do que o vetor como um todo)
    - busca deve continuar apenas na segunda parte do vetor:
      - chamada recursiva com parâmetros:
        - » número de elementos restantes
        - » ponteiro para o primeiro elemento dessa segunda parte
      - valor retornado deve ser corrigido

Algoritmo: int buscaBinariaRecursiva(int vet[], int ini, int fim, int elem)

```
se ini < fim então
    int meio ← ini + (fim − ini)/2;
    se elem < vet[meio] então
        // busca no subvetor inferior
        retorna buscaBinariaRecursiva(vet, ini, meio, elem);
    senão
        se elem > vet[meio] então
            // busca no subvetor superior
            retorna buscaBinariaRecursiva(vet, meio+1, fim, elem);
        senão
            // elemento encontrado
            retorna meio;
// caso não achou elem
retorna -1;
```

Árvore Binária de Busca

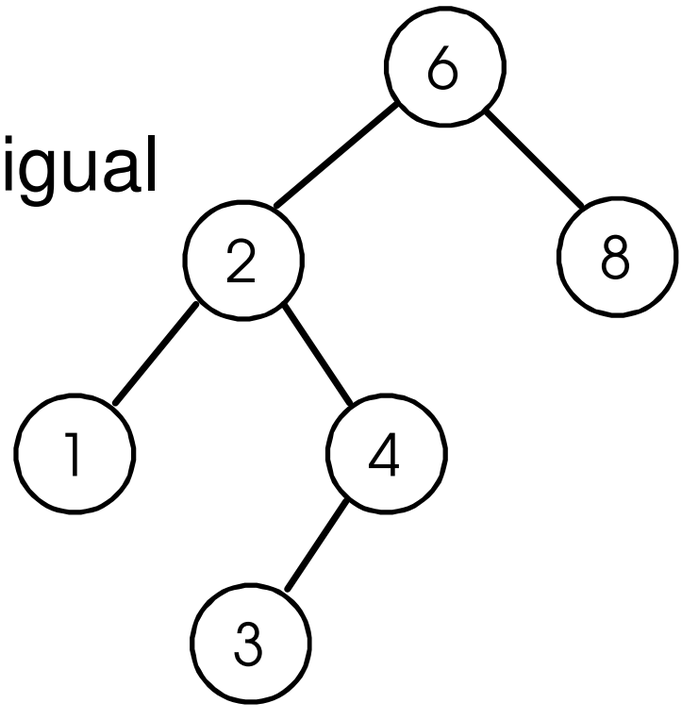
- Busca binária em vetor:
  - dados armazenados em vetor, de forma ordenada
  - bom desempenho computacional para pesquisa
  - inadequado quando inserções e remoções são freqüentes
    - exige re-arrumar o vetor para abrir espaço uma inserção
    - exige re-arrumar o vetor após uma remoção

Árvore Binária de Busca

- Árvores binárias:
  - árvore binária balanceada:
    - os nós internos têm todos, ou quase todos, 2 filhos
    - qualquer nó pode ser alcançado a partir da raiz em *O(log n)* passos
  - árvore binária degenerada:
    - todos os nós têm apenas *1* filho, com exceção da (única) folha
    - qualquer nó pode ser alcançado a partir da raiz em *O(n)* passos

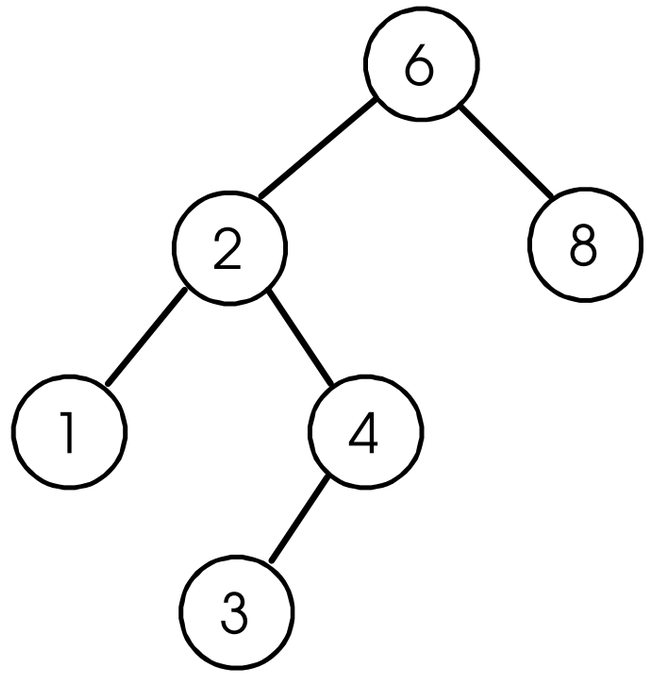
Árvore Binária de Busca

- Árvores binárias de busca:
  - o valor associado à raiz é sempre maior que o valor associado a qualquer nó da sub-árvore à esquerda (*sae*) e
  - o valor associado à raiz é sempre menor ou igual (para permitir repetições) que o valor associado a qualquer nó da sub-árvore à direita (*sad*)
  - quando a árvore é percorrida em ordem simétrica (*sae - raiz - sad*), os valores são encontrados em ordem não decrescente



Árvore Binária de Busca

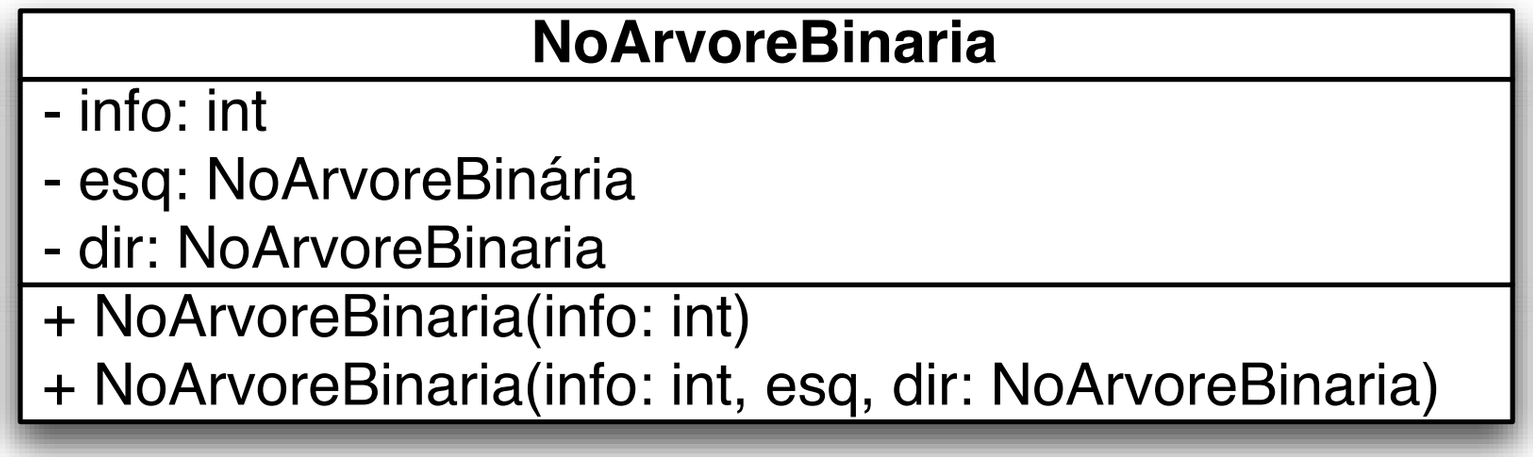
- Pesquisa em árvores binárias de busca:
  - compare o valor dado com o valor associado à raiz
  - se for igual, o valor foi encontrado
  - se for menor, a busca continua na sae
  - se for maior, a busca continua na sad



Implementação

- Representação da árvore: ponteiro para o nó raiz
- Representação de um nó da árvore:

Classe NoArvoreBinaria



Classe ÁrvoreBináriaBusca

ArvoreBinariaBusca
- raiz : NoArvoreBinaria
+ ArvoreBinariaBusca() + busca(v : int) : NoArvoreBinaria - busca(a : NoArvoreBinaria, v : int) : NoArvoreBinaria + insere(v : int) : void - insere(a : NoArvoreBinaria, v : int) : NoArvoreBinaria + retira(v : int) : void - retira(a : NoArvoreBinaria, v : int) : NoArvoreBinaria + toString() : String + toStringDecrescente() : String

Árvore Binária de Busca

- Operação de impressão: imprime os valores em ordem crescente percorrendo a árvore em ordem simétrica

Algoritmo: void imprime(NoArvoreBinaria no)

se $no \neq null$ **então**
    imprime(no.esq);
    System.out.print(no.info);
    imprime(no.dir);

Árvore Binária de Busca

- Operação de busca
  - explora a propriedade de ordenação da árvore
  - possui desempenho computacional proporcional à altura (O(log n) para o caso de árvore balanceada)

```
private NoArvoreBinaria buscaAux(NoArvoreBinaria r, int valor)
se r == null então
    retorna null;
senão
    se valor < r.info então
        retorna buscaAux(r.esq, valor);
    senão
        se valor > r.info então
            retorna buscaAux(r.esq, valor);
        senão
            retorna r;
```

Árvore Binária de Busca

- Operação de inserção
  - recebe um valor v a ser inserido
  - retorna o eventual novo nó raiz da (sub-)árvore
  - para adicionar v na posição correta, faça:
    - se a (sub-)árvore for vazia
      - crie uma árvore cuja raiz contém v
    - se a (sub-)árvore não for vazia
      - compare v com o valor na raiz
      - insira v na sae ou na sad, conforme o resultado da comparação

Árvore Binária de Busca

```
private NoArvoreBinaria insereAux(NoArvoreBinaria no, int valor)

se no == null então
    no ← new NoArvoreBinaria();
    no.info ← valor;
    no.esq ← null;
    no.dir ← null;
senão
    se valor < no.info então
        no.esq ← insereAux(no.esq, valor);
    senão
        no.dir ← insereAux(no.dir, valor);
retorna no;
```

É necessário atualizar as referências para as sub-árvores à esquerda ou à direita quando da chamada recursiva da função, pois a função de inserção pode alterar o valor da referência para a raiz da (sub-)árvore.

Árvore Binária de Busca

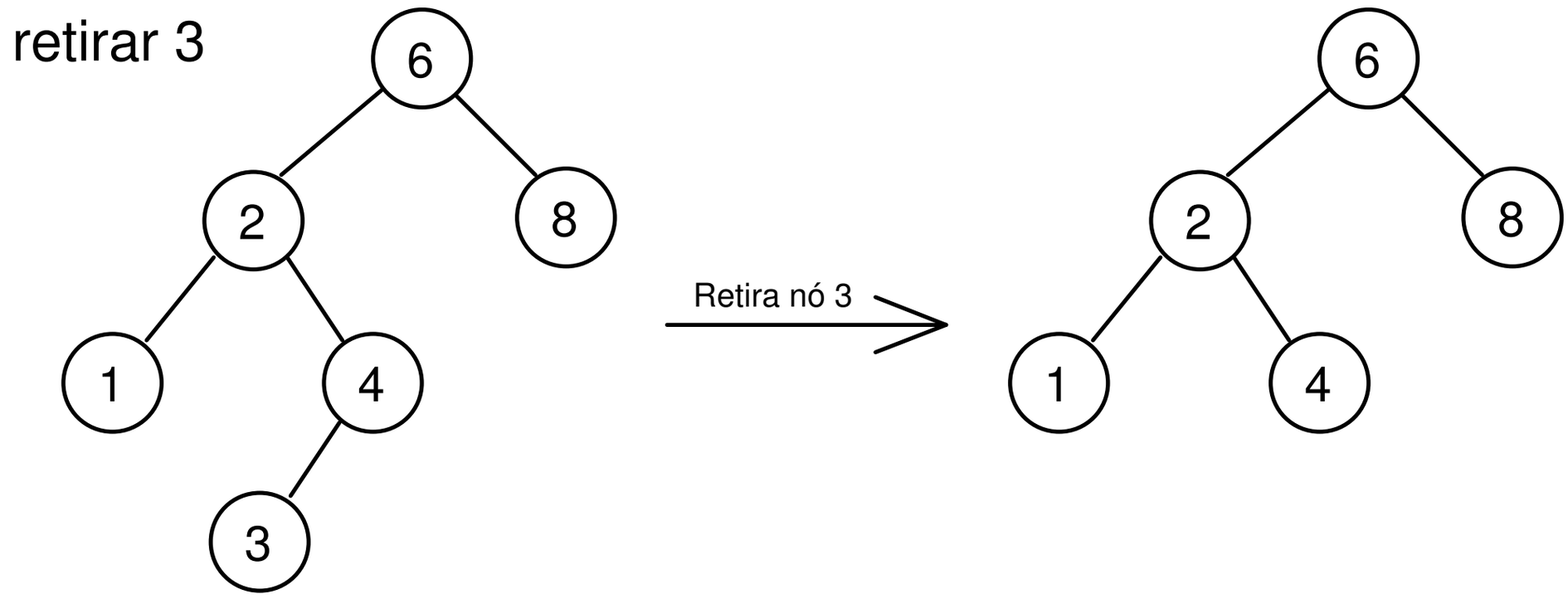
- Operação de remoção:
  - recebe um valor v a ser **removido**
  - retorna a eventual nova raiz da árvore
  - para remover v, faça:
    - se a árvore for vazia
      - nada tem que ser feito
    - se a árvore não for vazia
      - compare o valor armazenado no nó raiz com v
      - se for maior que v, retire o elemento da sub-árvore à esquerda
      - se for menor do que v, retire o elemento da sub-árvore à direita
      - se for igual a v, retire a raiz da árvore

Árvore Binária de Busca

- Operação de remoção (cont.):
  - para retirar a raiz da árvore, há 3 casos:
    - caso 1: a raiz que é folha
    - caso 2: a raiz a ser retirada possui um único filho
    - caso 3: a raiz a ser retirada tem dois filhos

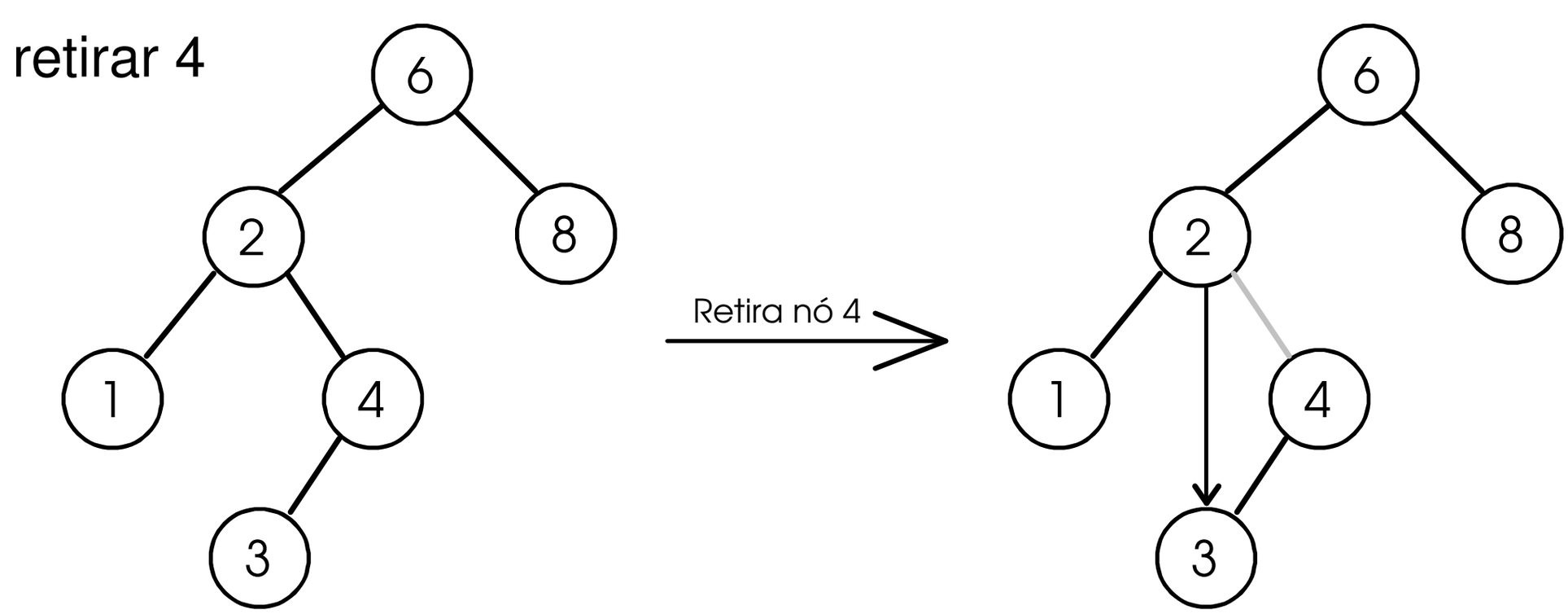
Árvore Binária de Busca

- Caso 1: a raiz da sub-árvore é folha da árvore original
  - libere a memória alocada pela raiz
  - retorne a raiz atualizada, que passa a ser NULL



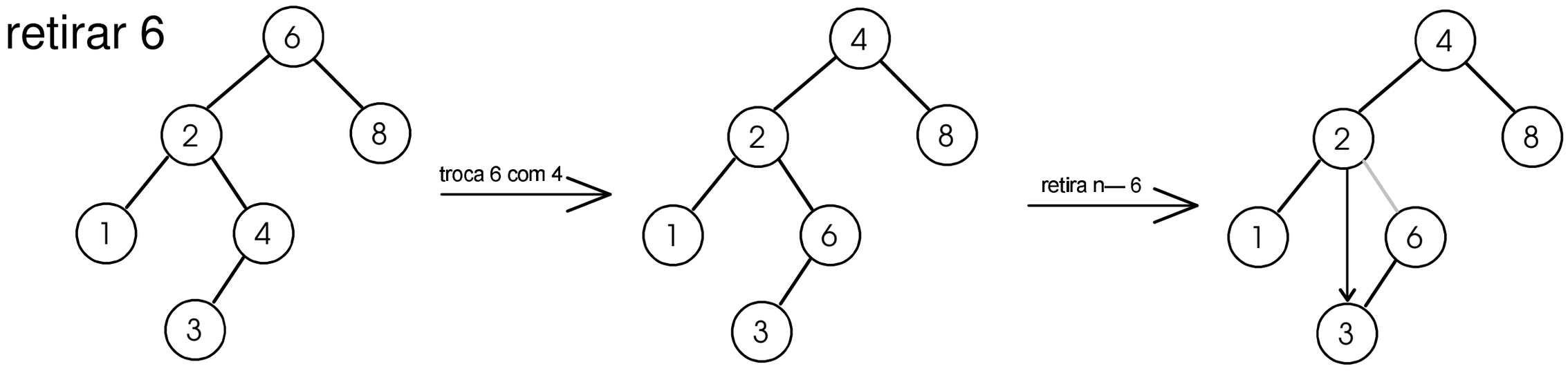
Árvore Binária de Busca

- Caso 2: a raiz a ser retirada possui um único filho
  - libere a memória alocada pela raiz
  - a raiz da árvore passa a ser o único filho da raiz



Árvore Binária de Busca

- Caso 3: a raiz a ser retirada tem dois filhos
  - encontre o nó N que precede a raiz na ordenação
    (o elemento mais à direita da sub-árvore à esquerda)
  - troque o dado da raiz com o dado de N
  - retire N da sub-árvore à esquerda
    (que agora contém o dado da raiz que se deseja retirar)
    - retirar o nó N mais à direita é trivial, pois N é um nó folha ou
      N é um nó com um único filho (no caso, o filho da direita nunca existe)



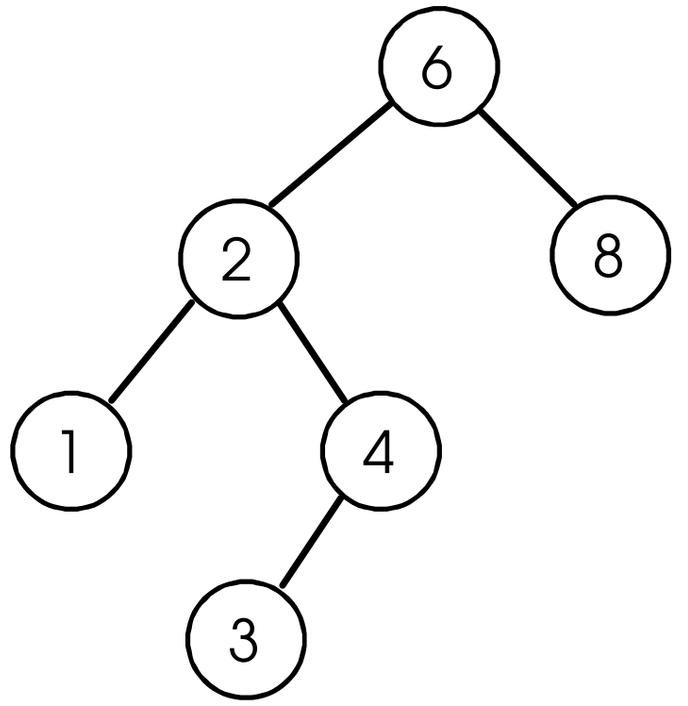
```
private NoArvoreBinaria retiraAux(NoArvoreBinaria no, int valor)
se no == null então
    retorna null;
senão
    se valor < no.info então
        no.esq ← retiraAux(no.esq, valor);
    senão
        se valor > no.info então
            no.dir ← retiraAux(no.dir, valor);
        senão
            se no.esq == null e no.dir == null então
                no ← null;
            senão
                se no.esq == null então
                    no ← no.dir;
                senão
                    se r.dir == null então
                        no ← no.esq;
                    senão
                        NoArvoreBinaria p ← no.esq;
                        enquanto p.dir ≠ null faça
                            p ← p.dir;
                        no.info ← p.info;
                        p.info ← valor;
                        no.esq ← retiraAux(no.esq, valor);
retorna no;
```

Resumo

- Busca linear em vetor:
  - percorra o vetor, elemento a elemento, verificando se o elemento de interesse é igual a um dos elementos do vetor
- Busca binária:
  - compare elem com o elemento do meio de vet
  - se elem for menor, pesquise na primeira metade do vetor
  - se elem for maior, pesquise na segunda parte do vetor
  - se for igual, retorne a posição
  - continue o procedimento, subdividindo a parte de interesse, até encontrar o elemento ou chegar a uma parte do vetor com tamanho 0

Resumo

- Árvores binárias de busca:
  - o valor associado à raiz é maior que o valor associado a qualquer nó da sub-árvore à esquerda (sae) e
  - o valor associado à raiz é menor ou igual que o valor associado a qualquer nó da sub-árvore à direita (sad)



- Operação de busca
  - explora a propriedade de ordenação da árvore
  - possui desempenho computacional proporcional à altura (O(log n) para o caso de árvore balanceada)