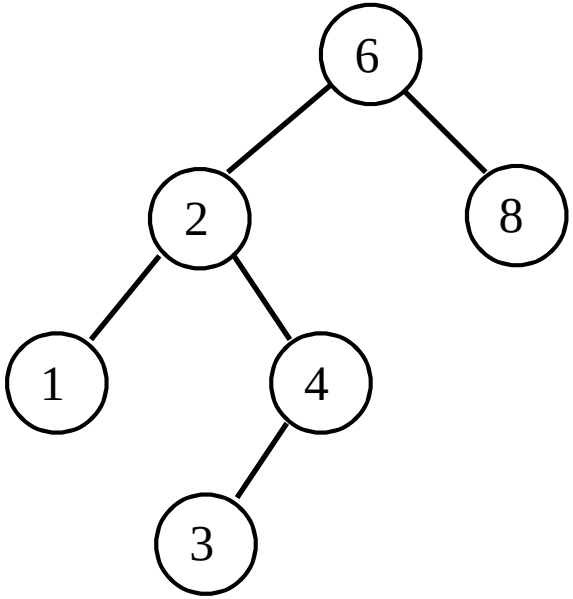
ÁRVORES ABB (ÁRVORES BINÁRIAS DE BUSCAS)

- Sérgio Carlos Portari Júnior

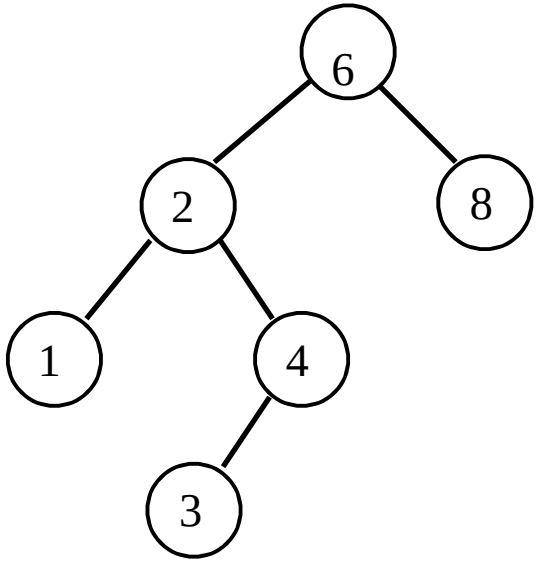
Árvore Binária de Busca (ABB)

- 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



Pesquisa em Árvore Binária 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



Tipo Árvore Binária de Busca

- árvore é representada pelo ponteiro para o nó raiz

```
struct noArv {
   int info;
   struct noArv* esq;
   struct noArv* dir;
};

typedef struct noArv NoArv;
```

ABB: Criação

- árvore vazia representada por NULL:

```
NoArv* abb_cria (void)
{
   return NULL;
}
```

ABB: Impressão

- imprime os valores da árvore em ordem crescente, percorrendo os nós em ordem simétrica

```
void abb_imprime (NoArv* a)
{
   if (a != NULL) {
      abb_imprime(a->esq);
      printf("%d\n",a->info);
      abb_imprime(a->dir);
   }
}
```

ABB: Busca

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

```
NoArv* abb_busca (NoArv* r, int v)
{
   if (r == NULL)
         return NULL;
   else if (r->info > v)
         return abb_busca (r->esq, v);
   else if (r->info < v)
         return abb_busca (r->dir, v);
   else return r;
}
```

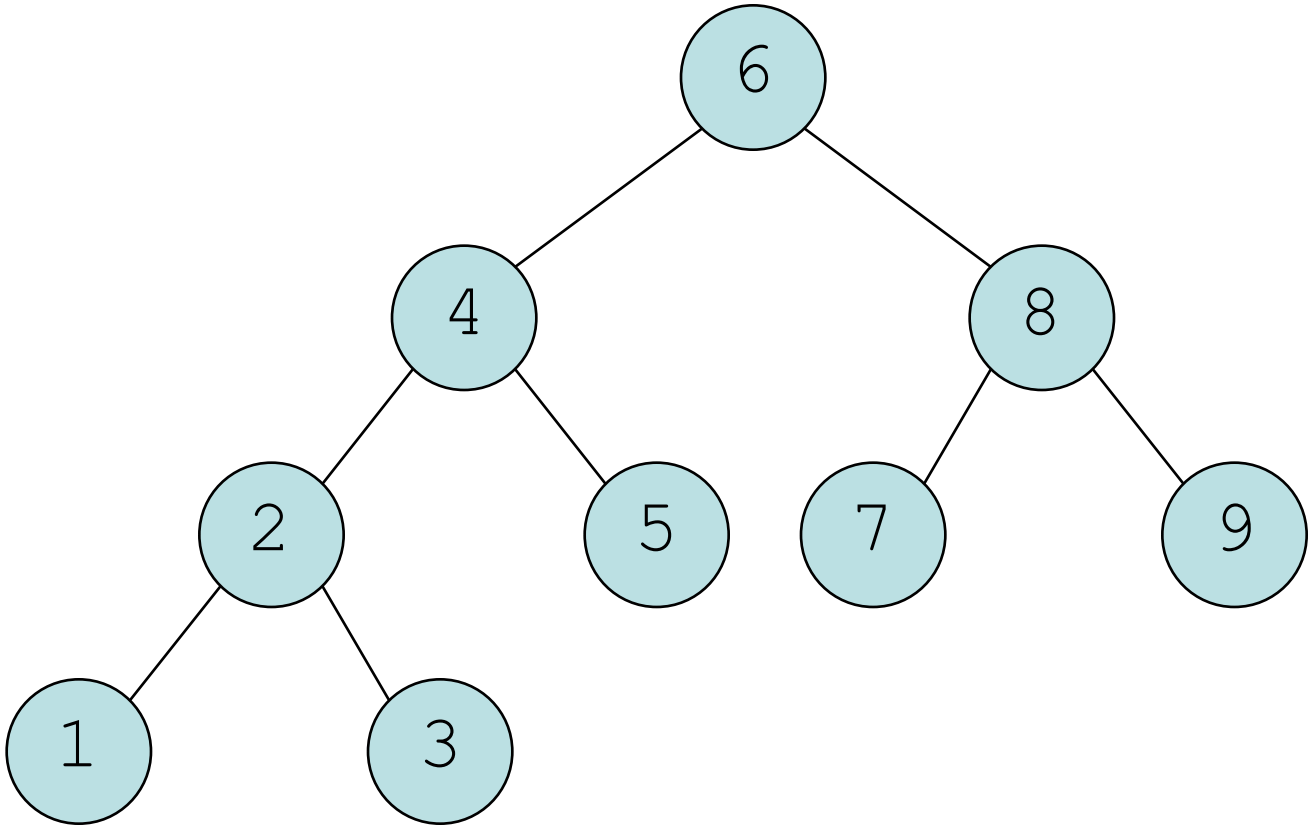
ABB: Inserção

- 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

```
NoArv* abb_insere (NoArv* a, int v)
{
   if (a==NULL) {
      a = (NoArv*)malloc(sizeof(NoArv));
      a->info = v;
      a->esq = a->dir = NULL;
   }

   else if (v < a->info)
      a->esq = abb_insere(a->esq,v);

   else  /* v >= a->info */
      a->dir = abb_insere(a->dir,v);

   return a;
}
```

é necessário atualizar os ponteiros para as sub-árvores à esquerda ou à direita quando da chamada recursiva da função, pois a função de inserção pode alterar o valor do ponteiro para a raiz da (sub-)árvore.

```
cria
insere 6
insere 4
insere 8
insere 2
insere 5
insere 1
insere 3
insere 7
insere 9
```



cria

insere 6

insere 4

insere 8

insere 2

insere 5

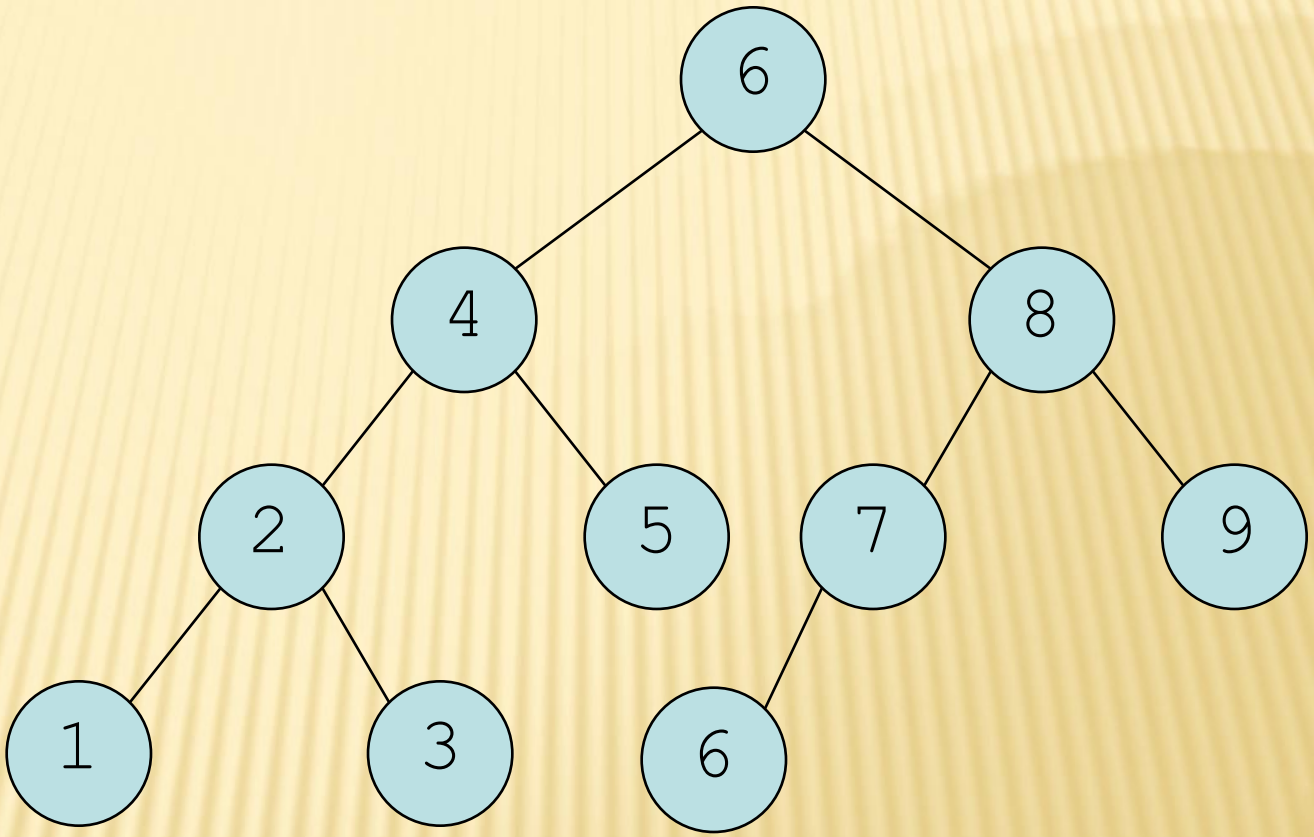
insere 1

insere 3

insere 7

insere 9

insere 6



a repetição está permitida!

```
NoArv* abb_insere (NoArv* a, int v)
{
    if (a==NULL) {
        a = (NoArv*)malloc(sizeof(NoArv));
        a->info = v;
        a->esq = a->dir = NULL;
    }

    else if (v < a->info)
        a->esq = abb_insere(a->esq,v);

    else if (v > a->info)
        a->dir = abb_insere(a->dir,v);

    return a;
}
```

ABB: Remoção

- recebe um valor v a ser retirado
- 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

ABB: Remoção

- 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

ABB: Remoção de folha

- 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

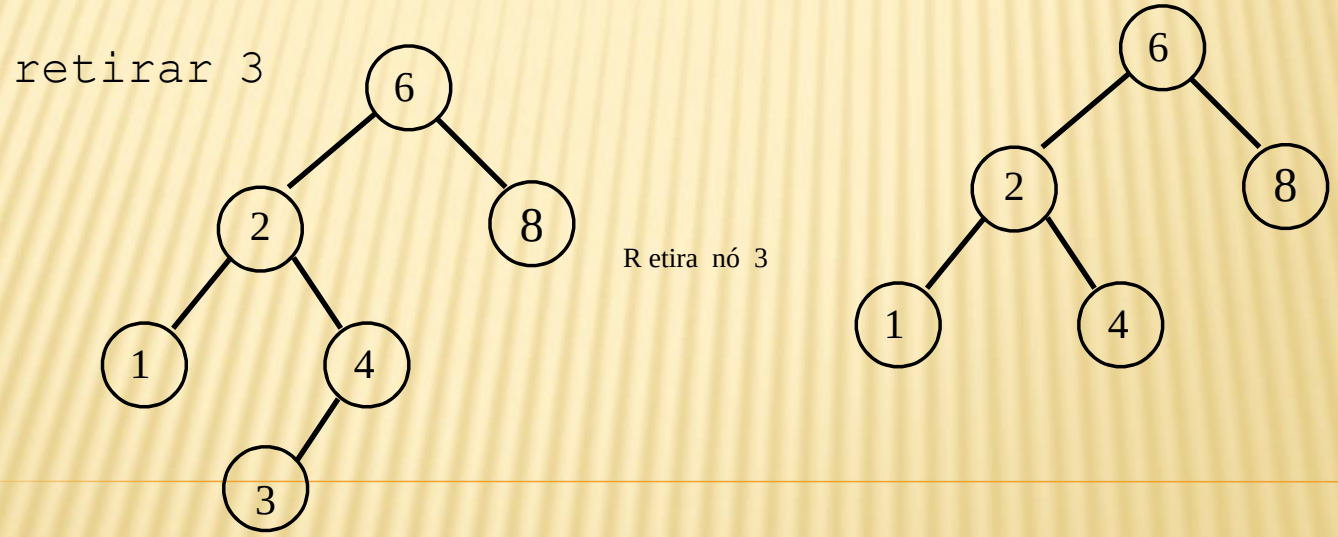


ABB: Remoção de pai de filho único

- 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

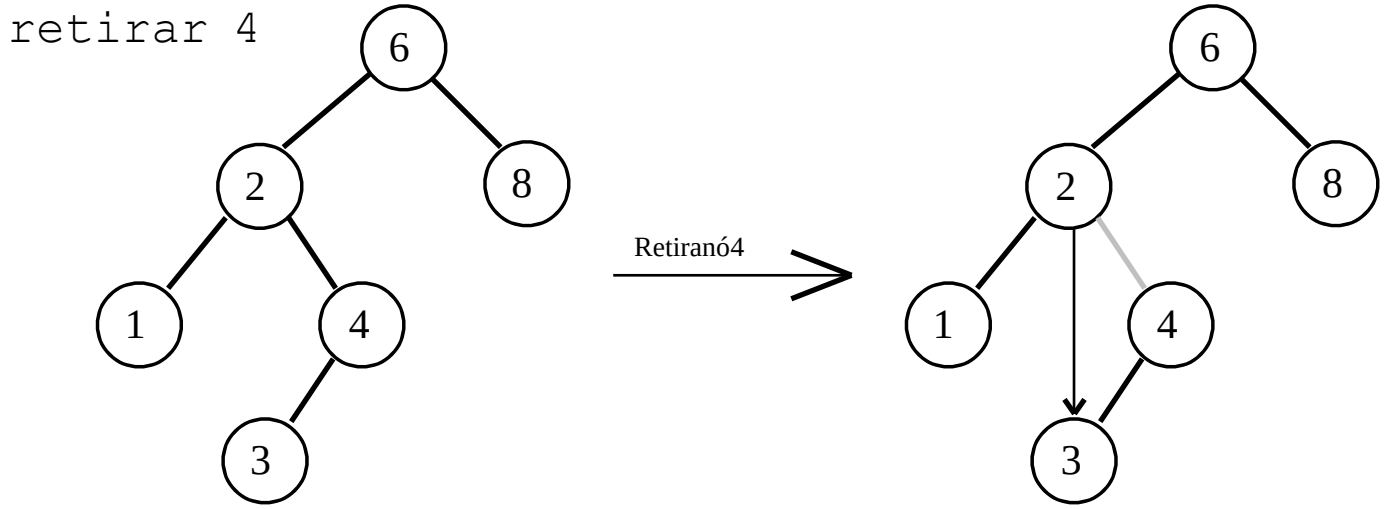
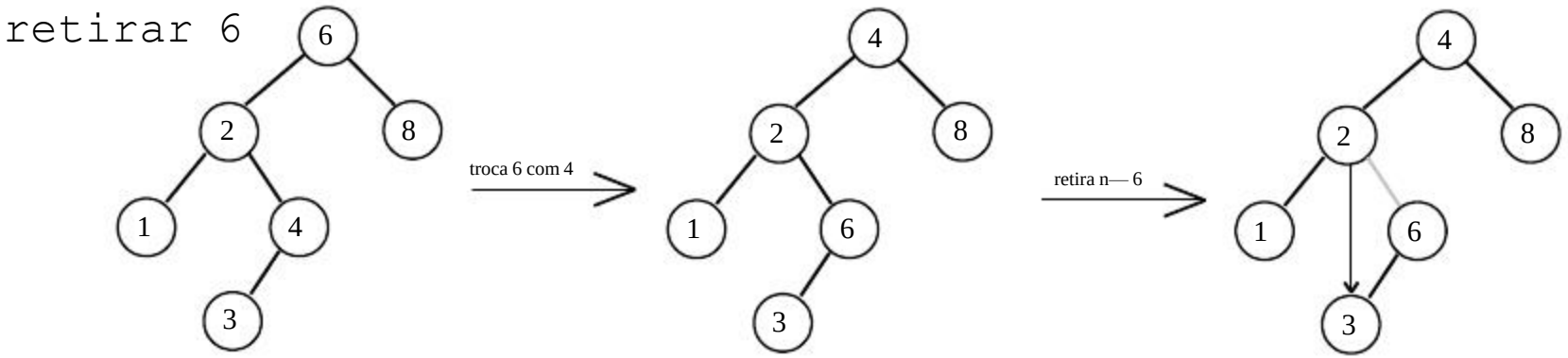


ABB: remoção de pai de dois filhos

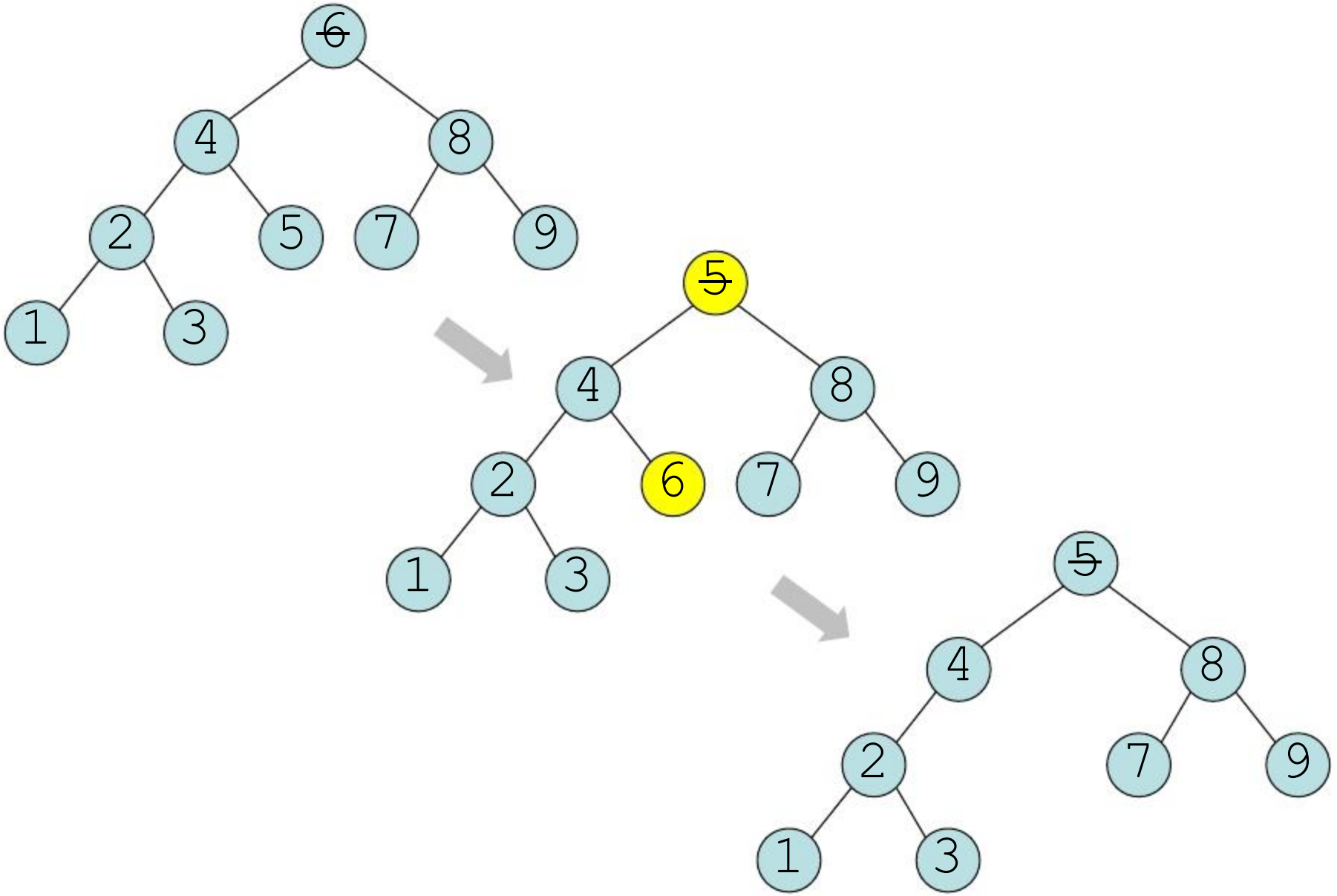
- Caso 3: a raiz a ser retirada tem dois filhos
  - encontre o nó N que precede a raiz na ordenação
    (o elemento mais à direita da sub-árvore à esquerda)
  - troque o dado da raiz com o dado de N
  - retire N da sub-árvore à esquerda
    (que agora contém o dado da raiz que se deseja retirar)
    - 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)

Destruição do nó 6



```
NoArv* abb_retira (NoArv* r, int v)
{
   if (r == NULL)
      return NULL;
   else if (r->info > v)
      r->esq = abb_retira(r->esq, v);
   else if (r->info < v)
      r->dir = abb_retira(r->dir, v);
   else {          /* achou o nó a remover */
      /* nó sem filhos */
      if (r->esq == NULL && r->dir == NULL) {
         free (r);
         r = NULL;
      }
      /* nó só tem filho à direita */
      else if (r->esq == NULL) {
         NoArv* t = r;
         r = r->dir;
         free (t);
      }
```

```
      /* só tem filho à esquerda */
      else if (r->dir == NULL) {
         NoArv* t = r;
         r = r->esq;
         free (t);
      }
      /* nó tem os dois filhos */
      else {
         NoArv* f = r->esq;
         while (f->dir != NULL) {
            f = f->dir;
         }
         r->info = f->info; /* troca as informações */
         f->info = v;
         r->esq = abb_retira(r->esq,v);
      }
   }
   return r;
}
```


6
4
8
2
5
7
9
1
3
5
4
8
2
6
7
9
1
3
5
4
8
2
7
9
1
3