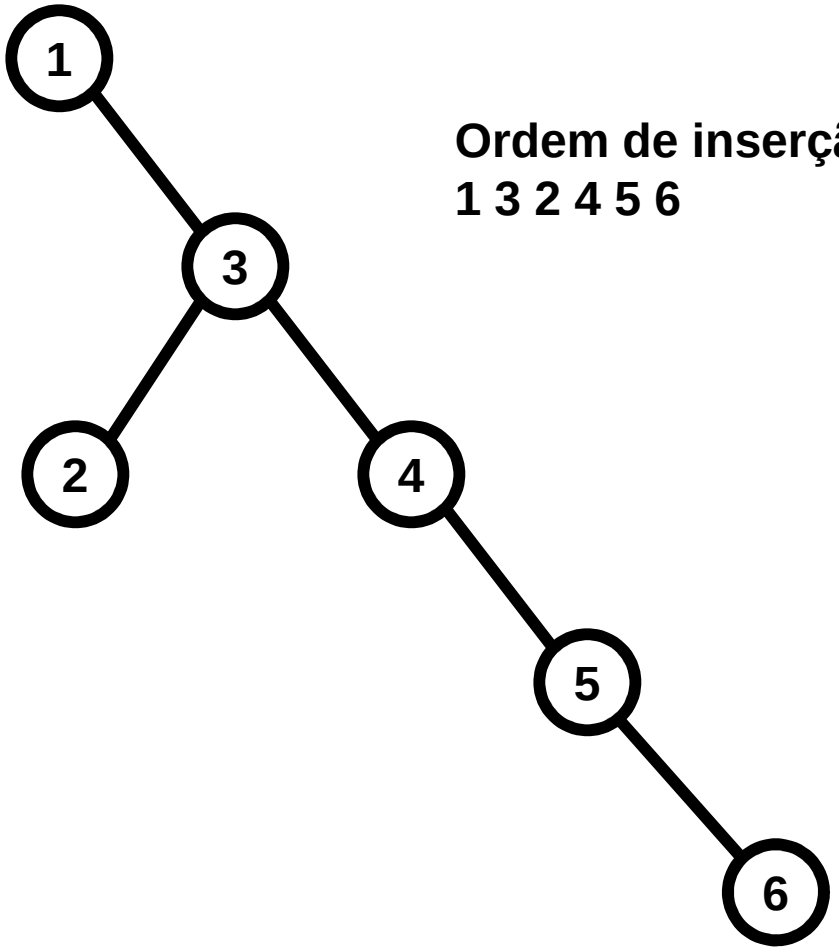
Árvores binárias de pesquisa com balanceamento

Algoritmos e Estruturas de Dados II

Árvores binárias de pesquisa

- Pior caso para uma busca é O(n)



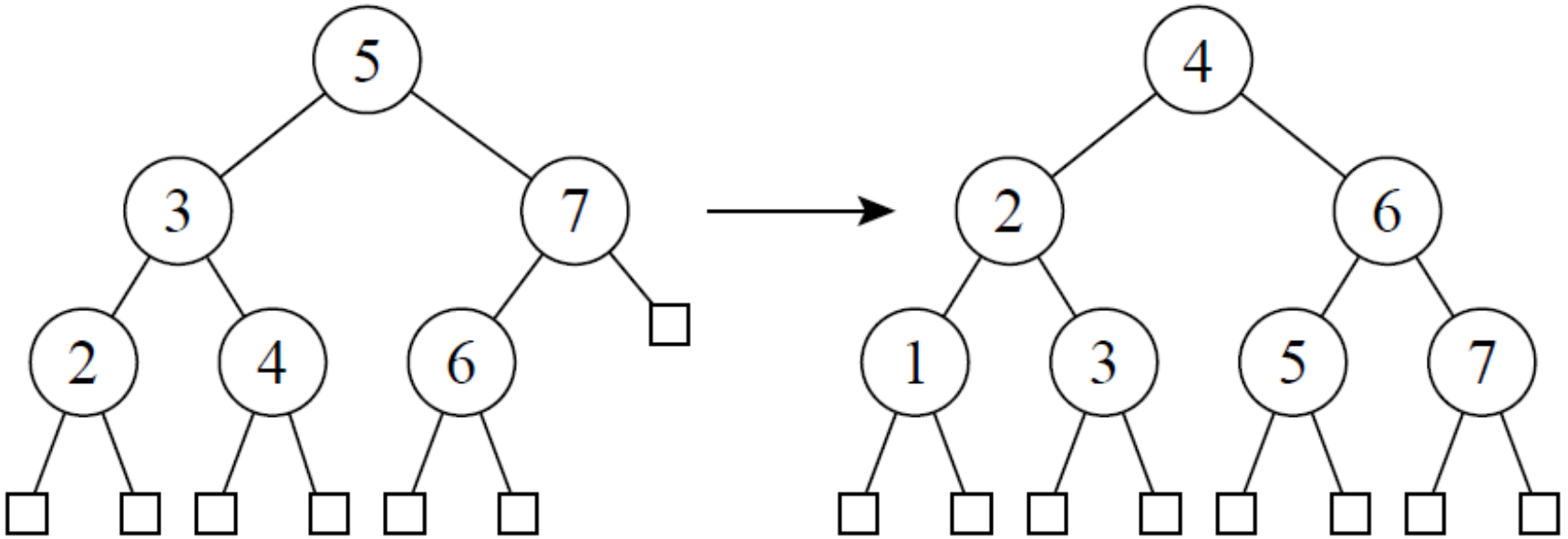
Ordem de inserção:
1 3 2 4 5 6

Árvore completamente balanceada

- Nós folha (externos) aparecem em no máximo dois níveis diferentes
- Minimiza o tempo médio de pesquisa
  - Assumindo distribuição uniforme das chaves
- Problema: manter árvore completamente balanceada após cada inserção é muito caro

Árvore completamente balanceada

- Para inserir a chave 1 na árvore à esquerda e manter a árvore completamente balanceada precisamos movimentar todos os nós



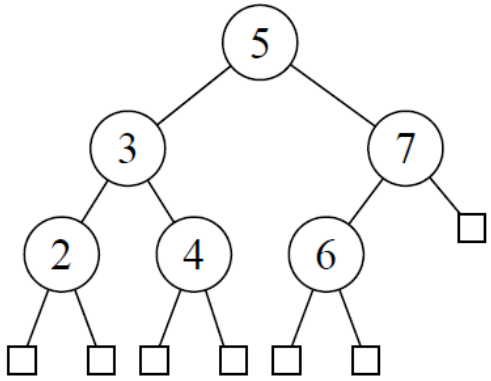
Árvores "quase balanceadas"

- Objetivo:
  - Funções para pesquisar, inserir e retirar eficientes
    - $O(\lg(n))$

- Solução intermediária que mantém a árvore "quase balanceada", em vez de tentar manter a árvore completamente balanceada

Árvores "quase balanceadas"

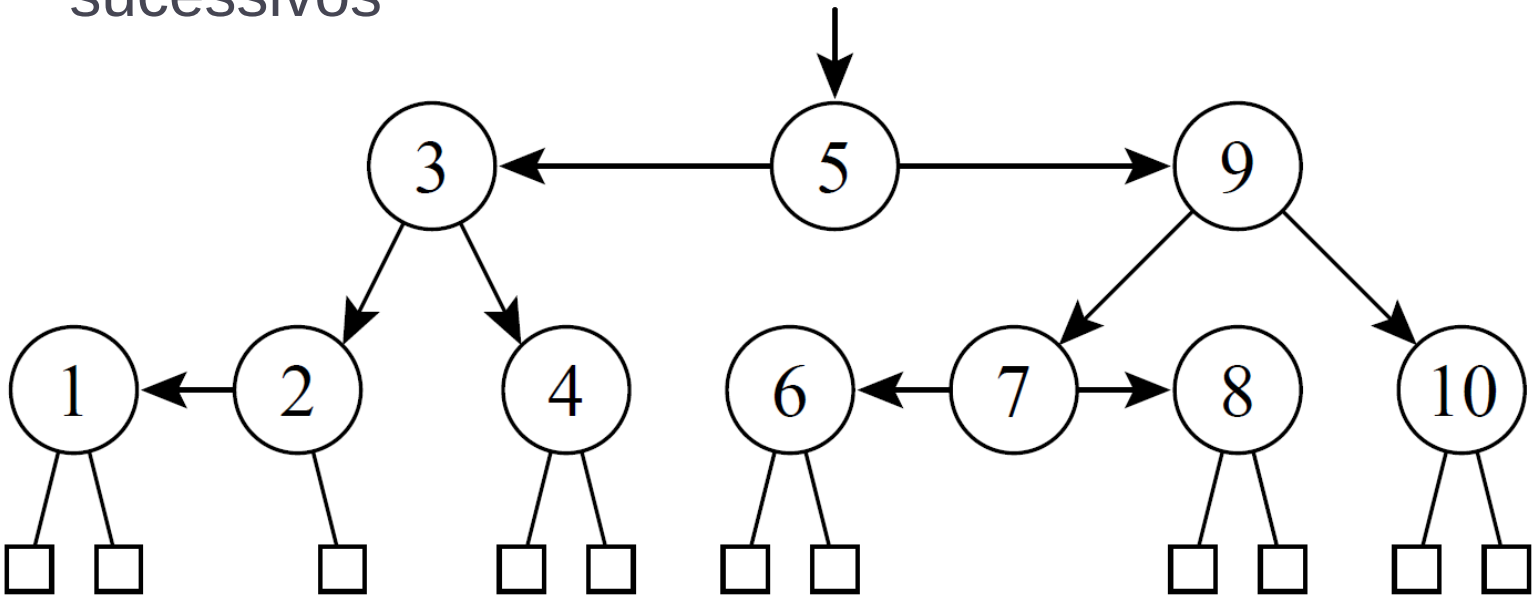
- Várias formas de definir e construir uma árvore "quase balanceada"
- Exemplos de restrições aplicadas a árvores para fazê-las "quase balanceadas"
  - Fazer que todas as folhas aparecem no mesmo nível
  - Restringir a diferença entre as alturas das subárvores de cada nó
  - Minimizar o comprimento do caminho interno da árvore



8 = 0 + 1 + 1 + 2 + 2 + 2

Árvores SBB

- Uma árvore SBB é uma árvore binária com apontadores *verticais* e *horizontais*, tal que:
  - Todos os caminhos da raiz até cada nó externo possuem o mesmo número de apontadores verticais
  - Não podem existir dois apontadores horizontais sucessivos



Árvores SBB – estrutura

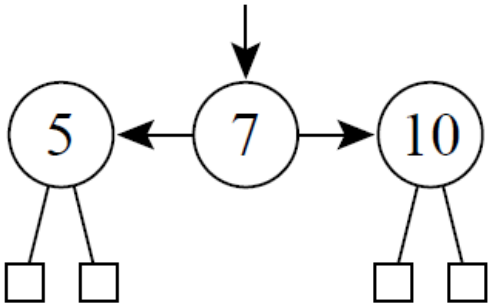
```
#define SBB_VERTICAL 0
#define SBB_HORIZONTAL 1

struct sbb {
    struct registro reg;
    struct sbb *esq;
    struct sbb *dir;
    int esqtipo;
    int dirtipo;
}
```

Pesquisa em árvore SBB

- Idêntica à pesquisa em uma árvore de busca binária não balanceada
  - Ignoramos a direção dos apontadores

Inserção numa árvore SBB

- A chave a ser inserida é sempre inserida após o apontador vertical mais baixo na árvore
- Dependendo da situação anterior à inserção podem aparecer dois apontadores horizontais

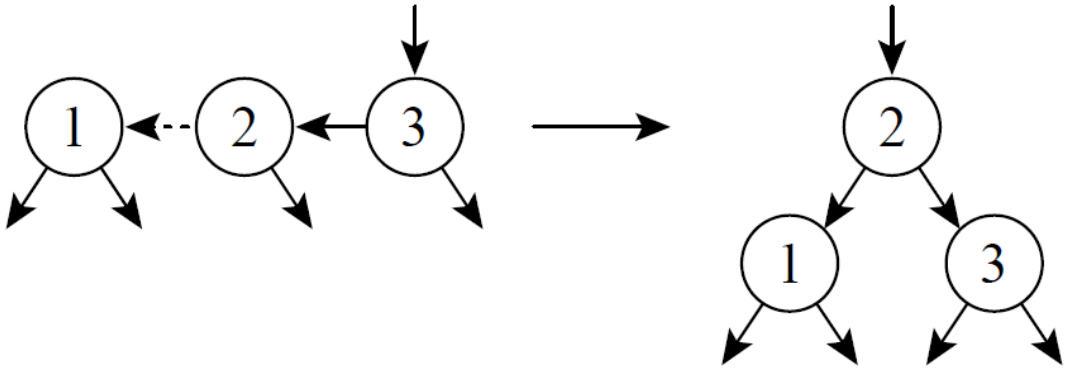


Inserção do 4, 6, 8, 11?

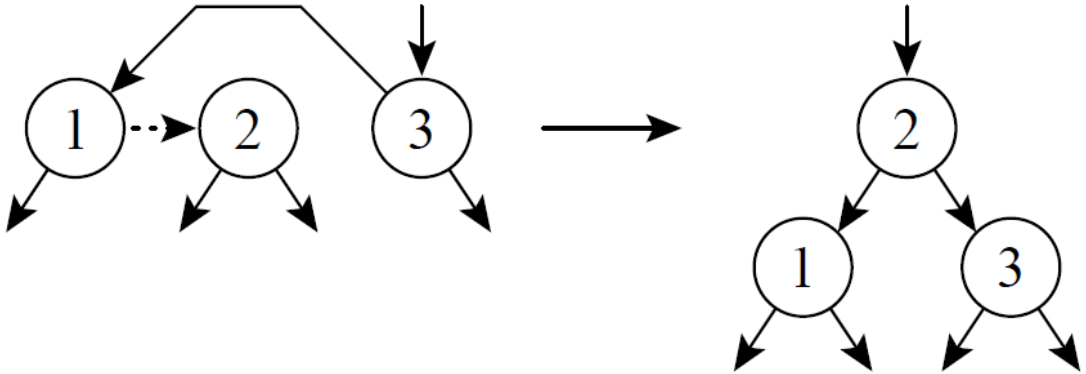
- Transformação local para manter as propriedades da árvore SBB

Transformações para manter propriedadas da árvore SBB

- Métodos para reorganizar casos onde aparecem dois ponteiros horizontais consecutivos
  - Esquerda-esquerda (e direita-direita)

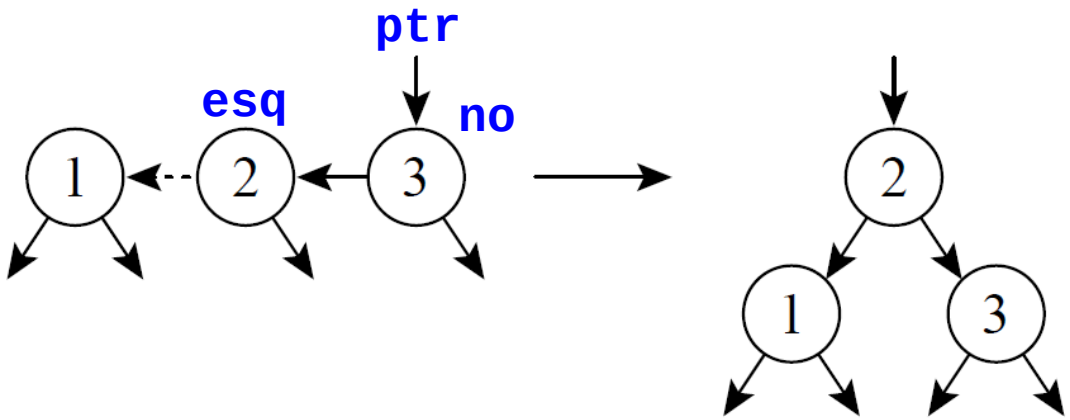


  - Esquerda-direita (e direita-esquerda)



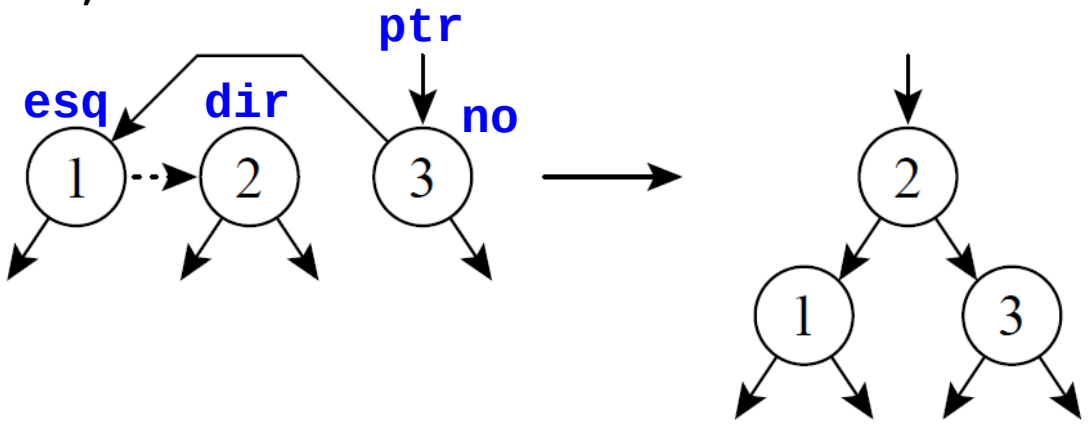
Transformações para manter propriedadas da árvore SBB - código

```
void ee(struct sbb **ptr) {
    struct sbb *no = *ptr;
    struct sbb *esq = no->esq;
    no->esq = esq->dir;
    esq->dir = no;
    esq->esqtipo = SBB_VERTICAL;
    no->esqtipo = SBB_VERTICAL;
    *ptr = esq;
}
```



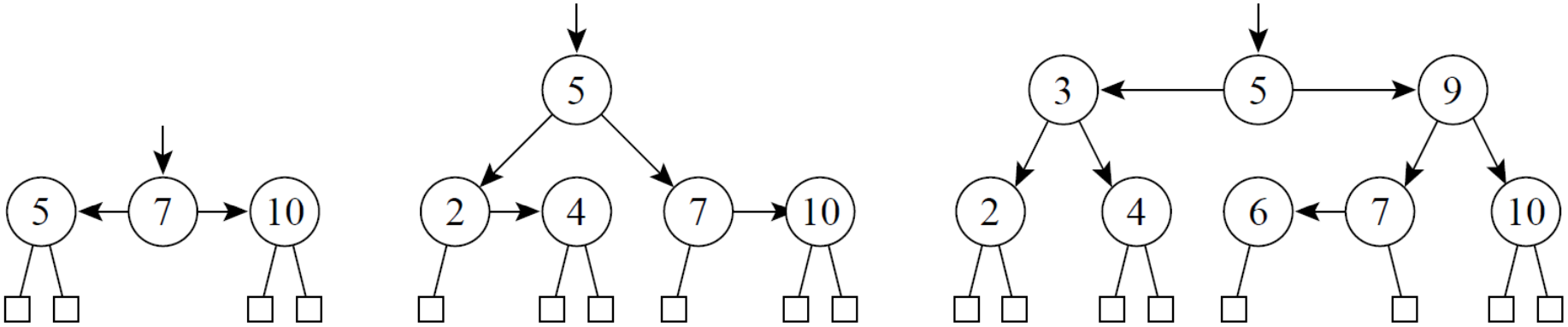
Transformações para manter propriedadas da árvore SBB - código

```
void ed(struct sbb **ptr) {
    struct sbb *no = *ptr;
    struct sbb *esq = no->esq;
    struct sbb *dir = esq->dir;
    esq->dir = dir->esq;
    no->esq = dir->dir;
    dir->esq = esq;
    dir->dir = no;
    esq->dirtipo = SBB_VERTICAL;
    no->esqtipo = SBB_VERTICAL;
    *ptr = dir;
}
```



Exemplo de inserção em árvore SBB

- Inserir chaves 7, 10, 5, 2, 4, 9, 3, 6, 5.5



Inserção em árvores SBB

```
void iinsere(struct registro reg, struct sbb **ptr,
             int *incli, int *fim) {
    if(*ptr == NULL) {
        iinsere_aqui(reg, ptr, incli, fim);
    } else if(reg.chave < *ptr->reg.chave) {
        iinsere(reg, &(*ptr->esq), &(*ptr->esqtipo), fim);
        if(*fim) return;
        if(*ptr->esqtipo == SBB_VERTICAL) {
            *fim = TRUE;
        } else if(*ptr->esq->esqtipo == SBB_HORIZONTAL) {
            ee(ptr); *incli = SBB_HORIZONTAL;
        } else if(*ptr->esq->dirtipo == SBB_HORIZONTAL) {
            ed(ptr); *incli = SBB_HORIZONTAL;
        }
    } else if(reg.chave > (*ptr)->reg.chave) {
         /* igual acima, só inverte o lado */
    } else {
         printf("erro: chave já está na árvore.\n");
         *fim = TRUE; } }
```

Inserção em árvores SBB

```
void iinsere_aqui(struct registro reg, struct sbb **ptr,
                  int *incli, int *fim) {
    struct sbb *no = malloc(sizeof(struct sbb));
    if(!no) { perror(NULL); exit(EXIT_FAILURE); }
    no->reg = reg;
    no->esq = NULL;
    no->dir = NULL;
    no->esqtipo = SBB_VERTICAL;
    no->dirtipo = SBB_VERTICAL;
    *ptr = no;
    *incli = SBB_HORIZONTAL;
    *fim = FALSE;
}
```

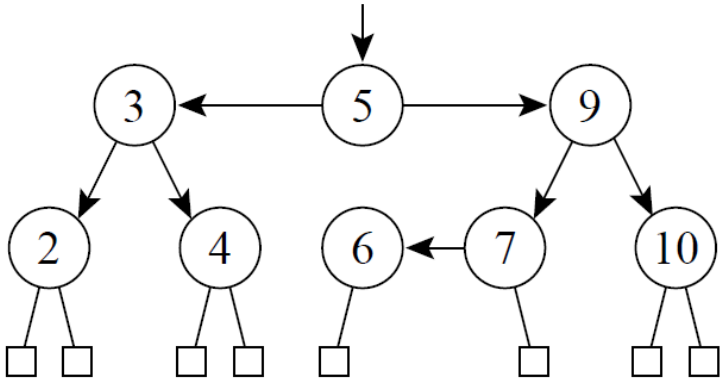
Inserção em árvores SBB

```
void insere(struct registro reg, struct sbb **raiz)
{
    int fim = FALSE;
    int inclinacao = SBB_VERTICAL;
    iinsere(reg, raiz, &inclinacao, &fim);
}

void inicializa(struct sbb **raiz)
{
    *raiz = NULL;
}
```

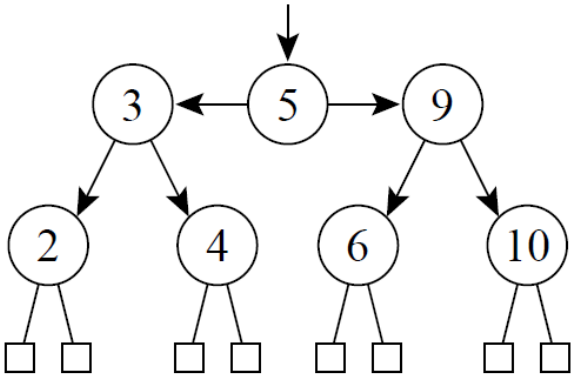
Retirada numa árvore SBB

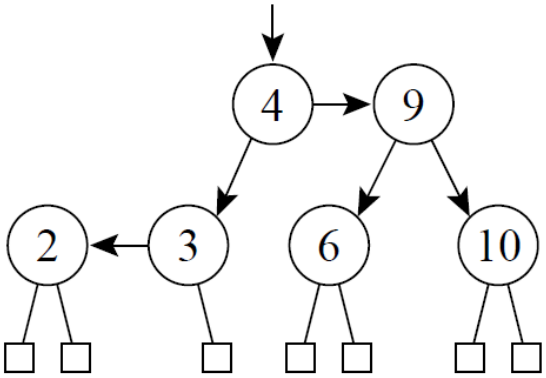
- Usaremos três procedimentos auxiliares
  - EsqCurto: chamado quando o nó é retirado à esquerda e diminui a altura da árvore
  - DirCu o à direita e diminu
  - Antec ui dois filhos

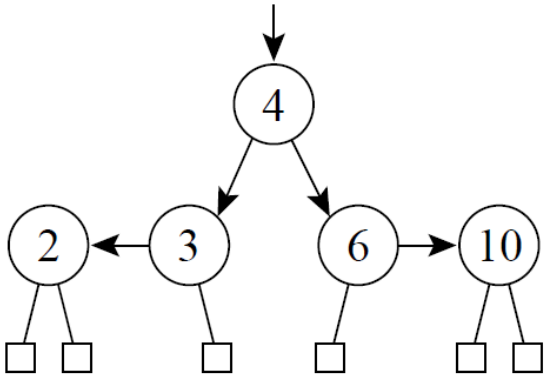
Retirada em árvore SBB – exemplos



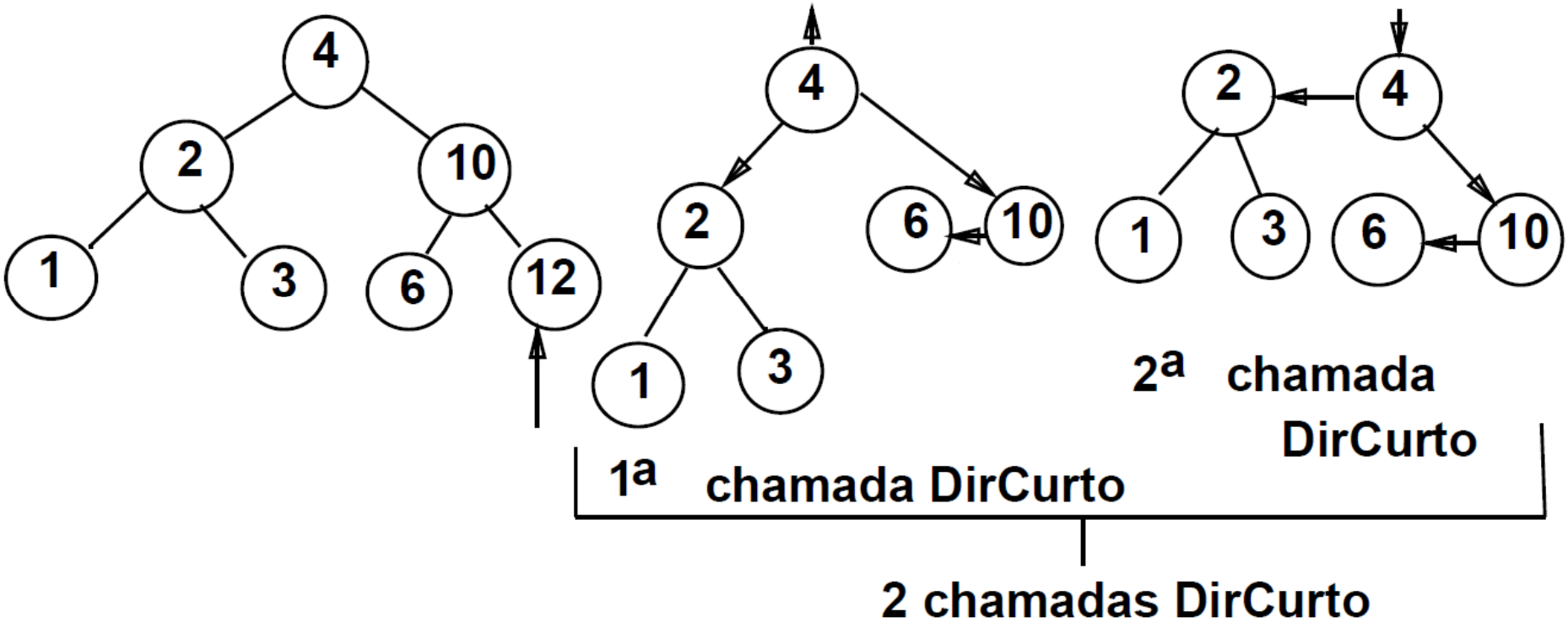
Retirando 7, 4 e 9:



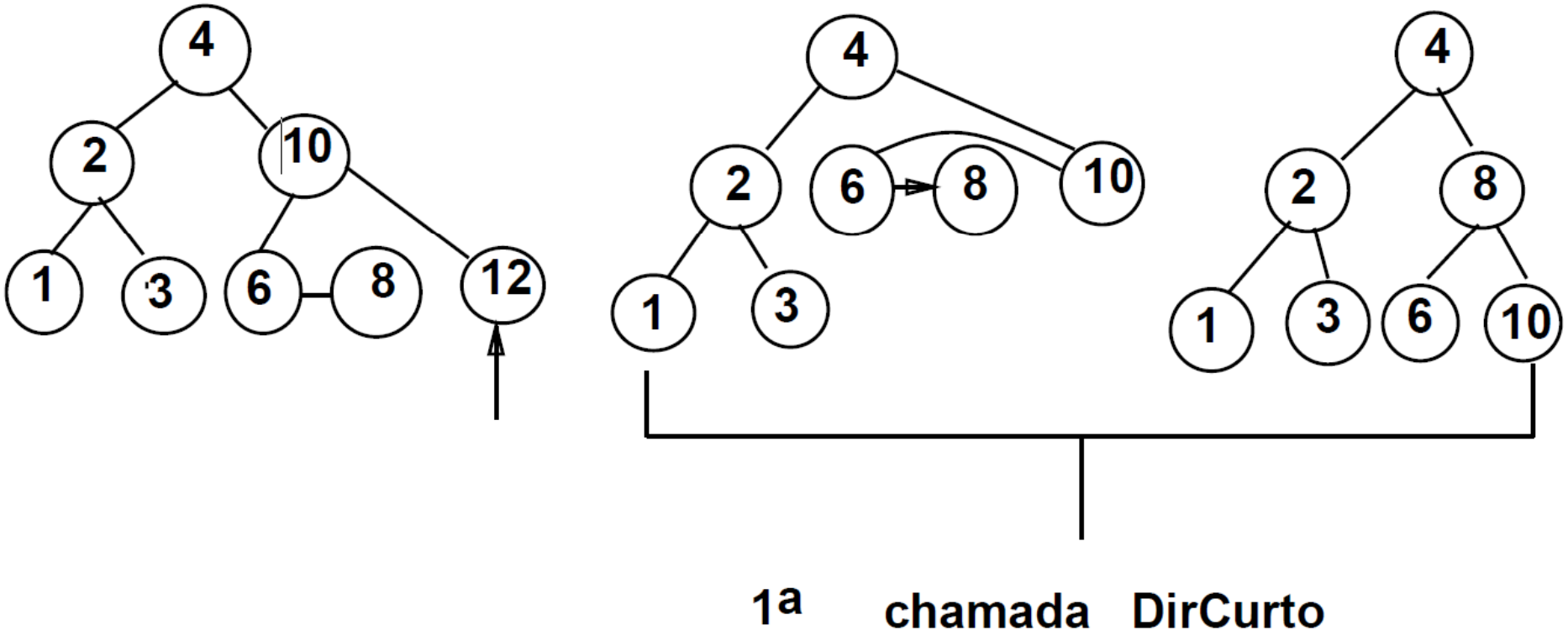




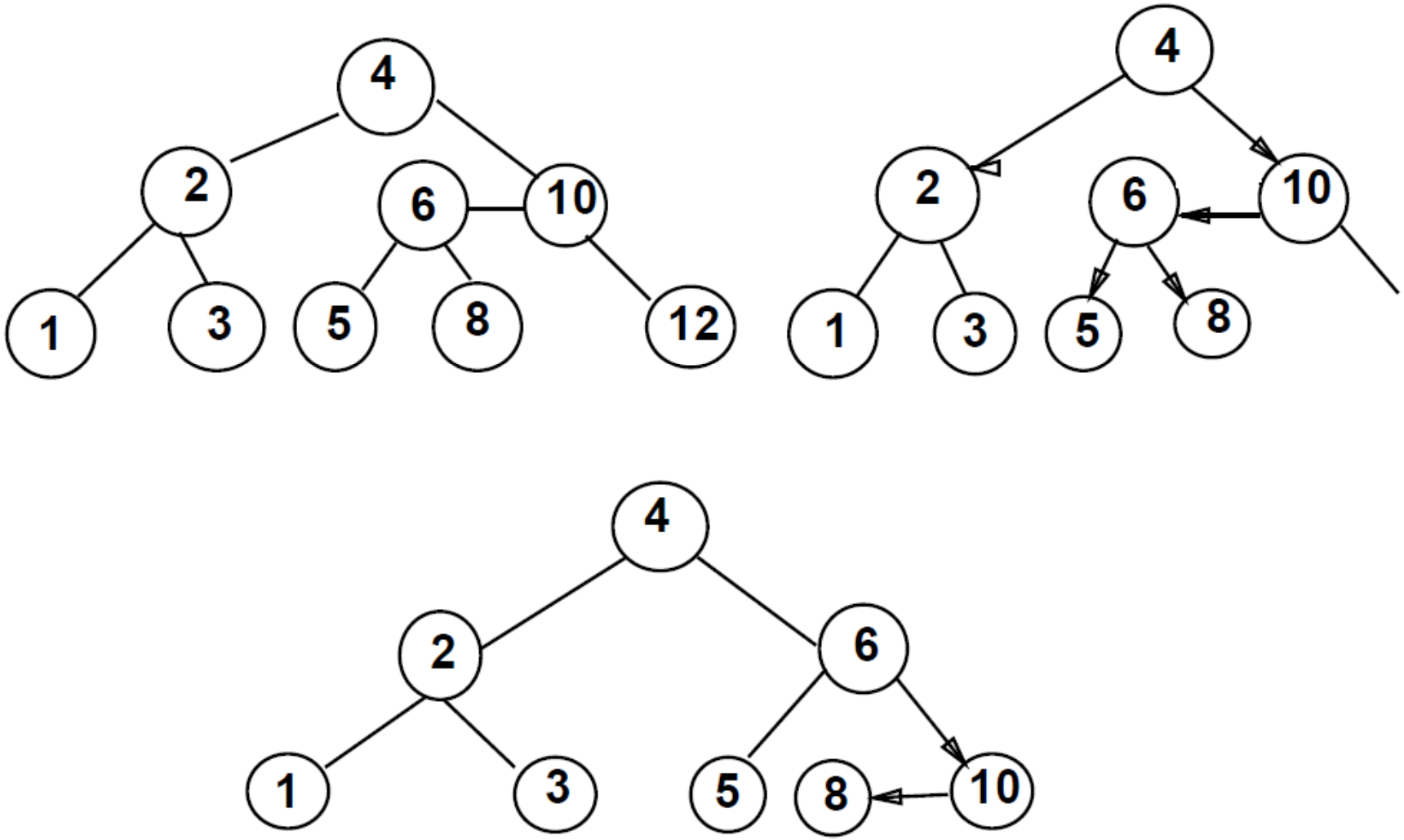
Retirada em árvore SBB – exemplos



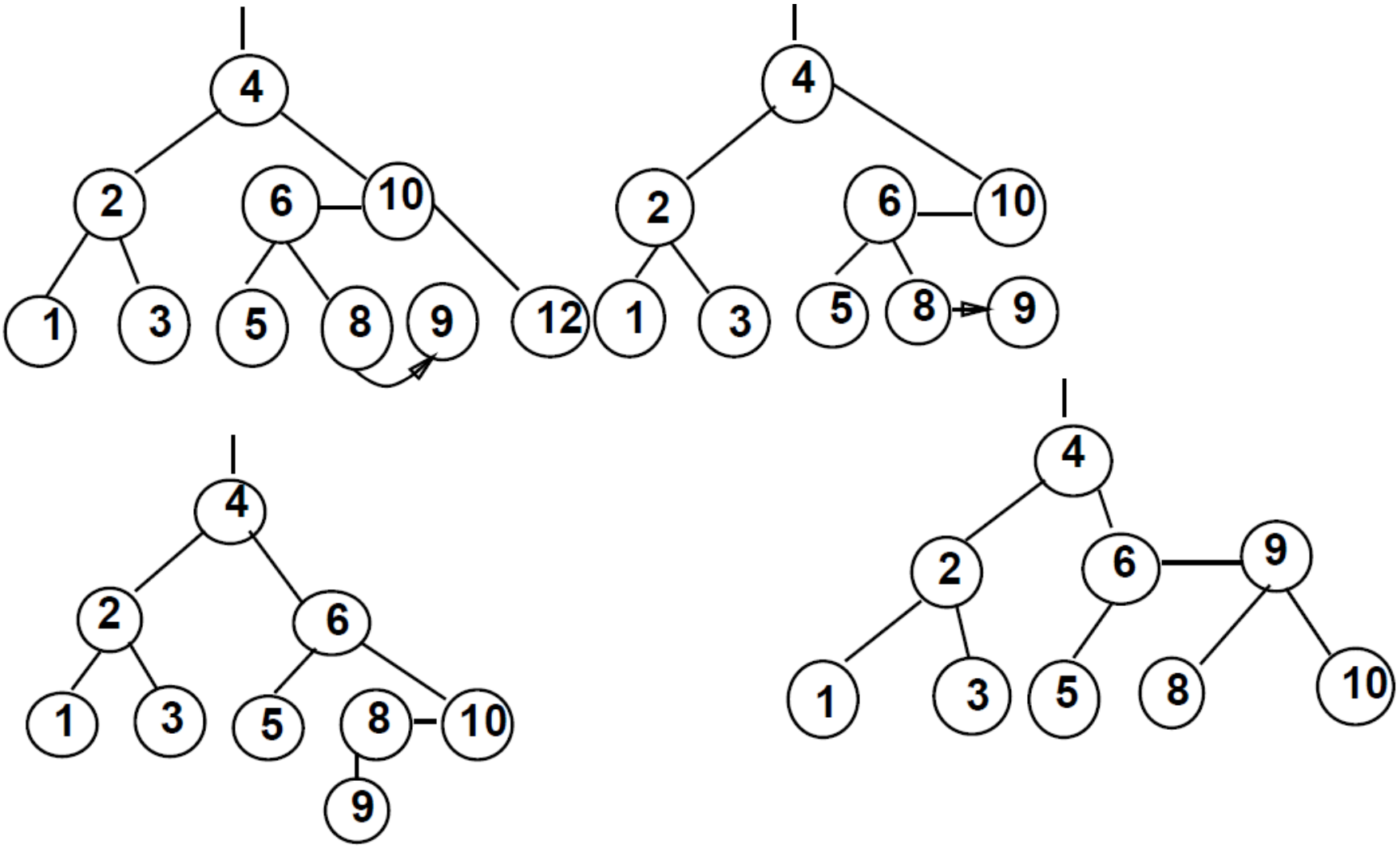
Retirada em árvore SBB – exemplos



Retirada em árvore SBB – exemplos



Retirada em árvore SBB – exemplos



Retirada de árvore SBB – código

```
void iretira(struct registro reg, struct sbb **ptr, int *fim) {
    if(*ptr == NULL) { *fim = TRUE; return; }
    if(reg.chave < *ptr->reg.chave) {
        iretira(reg, &(*ptr->esq), fim);
        if(!(*fim)) { EsqCurto(ptr, fim);
    } else if(reg.chave > *ptr->reg.chave) {
        /* igual acima, só inverte o lado. */
    } else { /* achou o registro: */
        *fim = FALSE;
        struct sbb *no = *ptr;
        if(no->dir == NULL) {
            *ptr = no->esq;
            free(no);
            if(*ptr != NULL) { *fim = TRUE; }
        } else if(no->esq == NULL) { /* igual acima. */ }
        } else {
            antecessor(ptr, &(*ptr->esq), fim);
            if(!(*fim)) EsqCurto(ptr, fim);
        }
}}
```

Retirada de árvore SBB – código

```
void EsqCurto(struct sbb **ptr, int *fim) {
    if(*ptr->esqtipo == SBB_HORIZONTAL) {
        *ptr->esqtipo = SBB_VERTICAL; *fim = TRUE;
    } else if(*ptr->dirtipo == SBB_HORIZONTAL) {
        struct sbb *dir = *ptr->dir;
        *ptr->dir = dir->esq;
        dir->esq = *ptr;
        *ptr = dir;
        if(*ptr->esq->dir->esqtipo == SBB_HORIZONTAL) {
            de(&(*ptr->esq)); *ptr->esqtipo = SBB_HORIZONTAL;
        } else if(*ptr->esq->dir->dirtipo == SBB_HORIZONTAL) {
            dd(&(*ptr->esq)); *ptr->esqtipo = SBB_HORIZONTAL;
        }
        *fim = TRUE;
    } else {
        *ptr->dirtipo = SBB_HORIZONTAL;
        if(*ptr->dir->esqtipo == SBB_HORIZONTAL) {
            de(ptr); *fim = TRUE;
        } else if(/* checa outro lado */) { ... }
    }}}
```

SBBs – análise

- Dois tipos de altura
  - Altura vertical h: conta o número de apontadores verticais da raiz até as folhas
  - Altura k: o número de ponteiros atravessados (comparações realizadas) da raiz até uma folha
- A altura k é maior que a altura h sempre que existirem apontadores horizontais na árvore
- Para qualquer árvore SBB temos $h \leq k \leq 2h$

SBBs – análise

- Bayer (1972) mostrou que

$$\lg(n+1) \leq k \leq 2\lg(n+2) - 2$$

- Custo para manter a propriedade SBB depende da altura da árvore $O(\lg(n))$
- Número de comparações em uma pesquisa com sucesso numa árvore SBB
  - Melhor caso: $C(n) = O(1)$
  - Pior caso: $C(n) = O(\lg(n))$
  - Caso médio: $C(n) = O(\lg(n))$

Exercícios

- Desenhe a árvore SBB resultante da inserção das chaves Q U E S T A O F C I L em uma árvore vazia.
- Qual o maior número de nós que pode ser armazenado numa árvore SBB com altura h?
- Dê a lista de inserções que gera a árvore SBB abaixo: