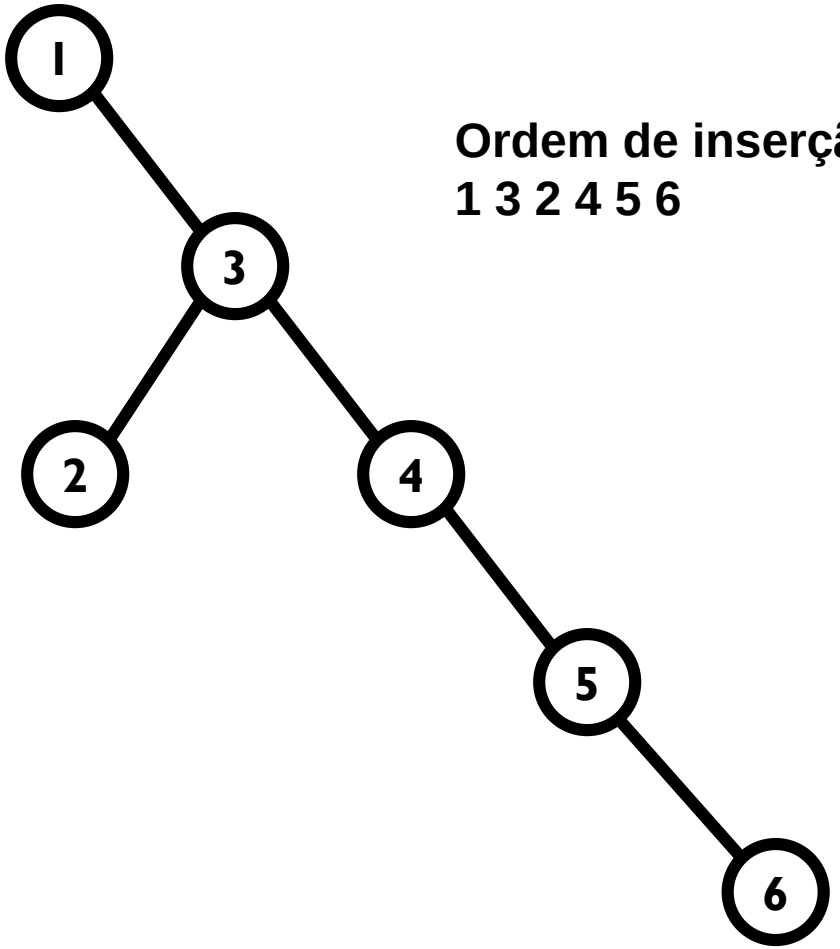
Árvores binárias de pesquisa com balanceamento

Algoritmos e Estruturas de Dados II

Árvores binárias de pesquisa

- Pior caso para uma busca é O(n)



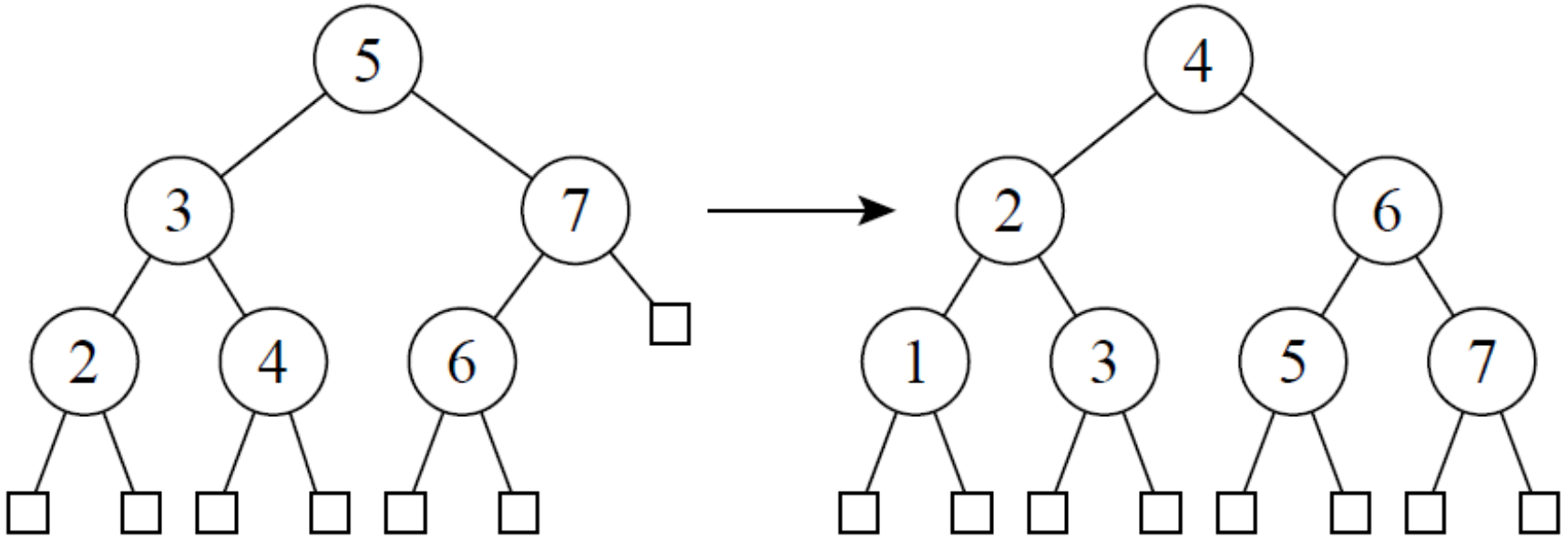
Ordem de inserção:
1 3 2 4 5 6

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

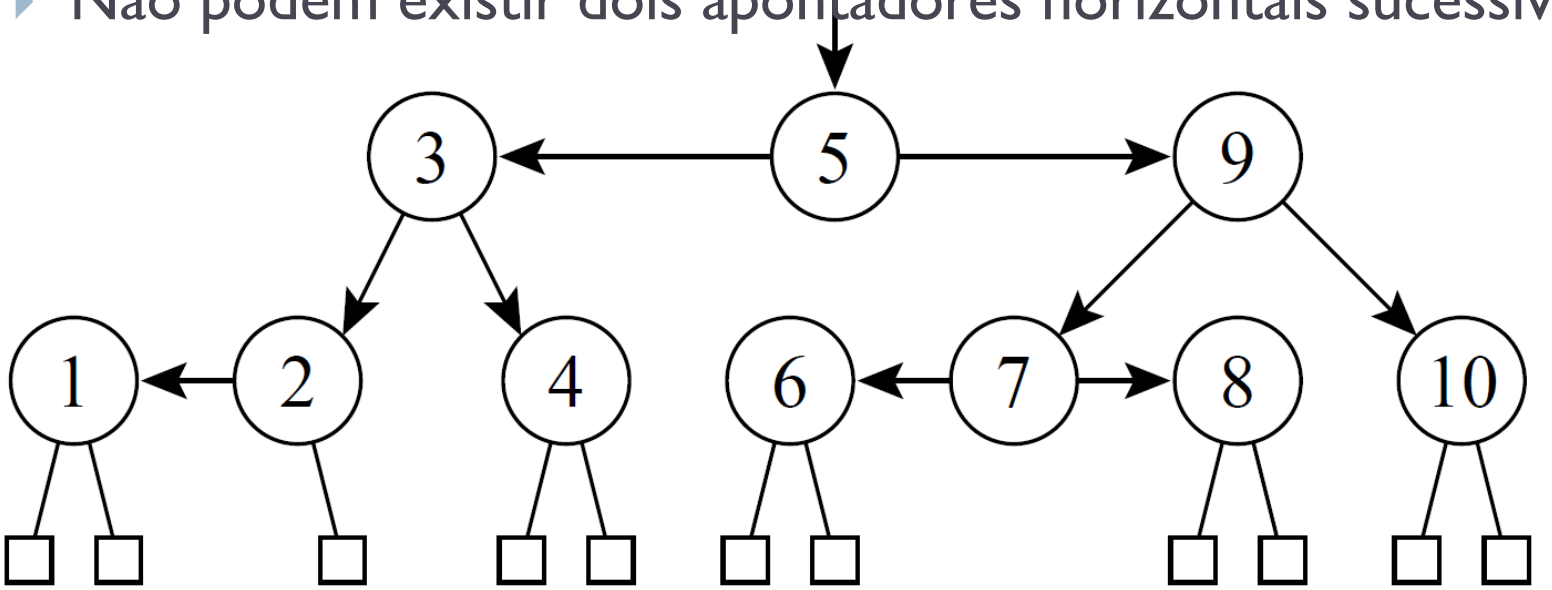
Árvore completamente balanceada

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



Árvores SBB

- Uma árvore SBB (symmetric binary B-tree) 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?

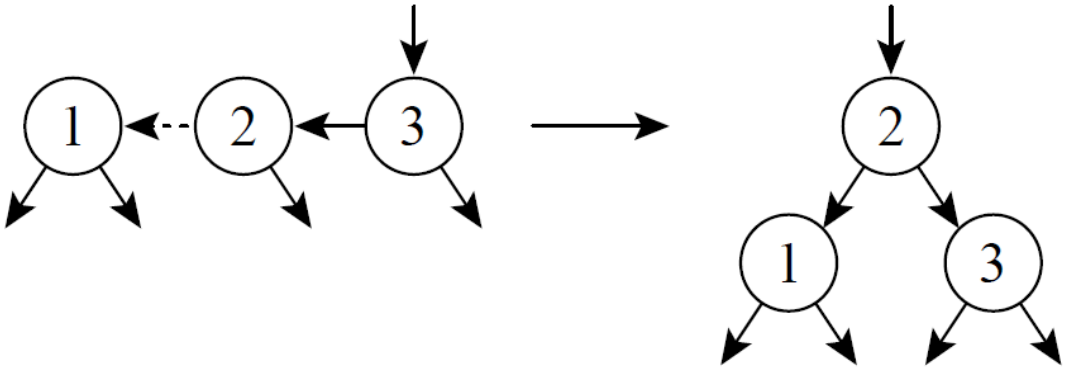
- Transformação local para manter as propriedades da árvore SBB

Criação de um nó

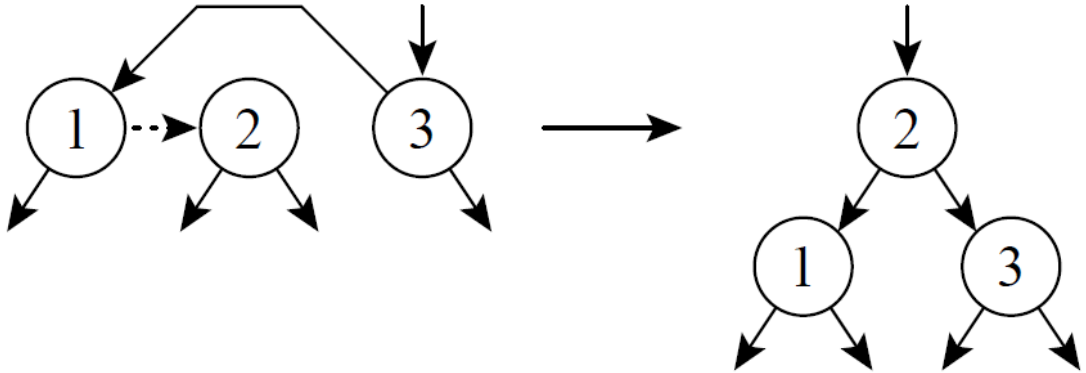
```
struct sbb *cria_no(struct registro reg) {

    struct sbb *no = malloc(sizeof(struct sbb));
    no->reg = reg;
    no->esq = NULL;
    no->dir = NULL;
    no->esqtipo = SBB_VERTICAL;
    no->dirtipo = SBB_VERTICAL;
    return no;
}
```

Transformações para manter propriedadas da árvore SBB

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

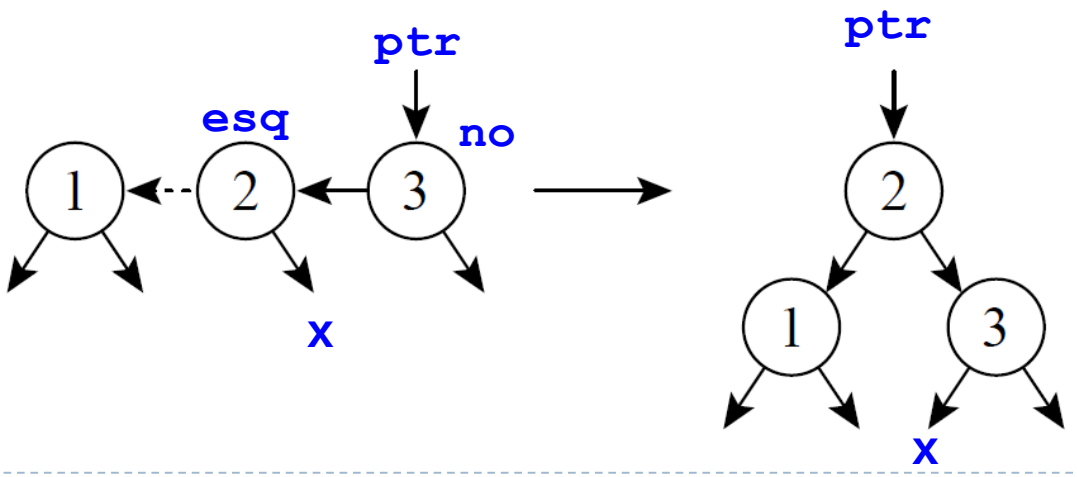


  - Esquerda-direita (e direita-esquerda)



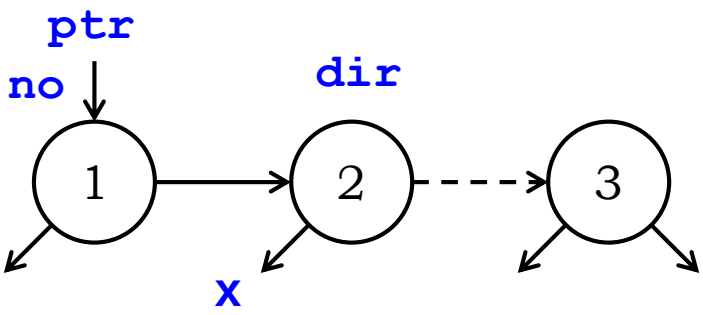
Transformações para manter propriedadas da árvore SBB - código

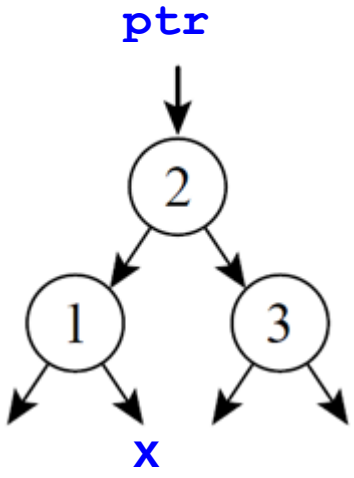
```
void ee(struct sbb **ptr) {
struct sbb *no = *ptr;
struct sbb *esq = no->esq;

    no->esq = esq->dir;    // rotD(ptr)
    esq->dir = no;
    esq->esqtipo = SBB_VERTICAL;
    no->esqtipo = SBB_VERTICAL;
    *ptr = esq;
}
```



Transformações para manter propriedadas da árvore SBB - código

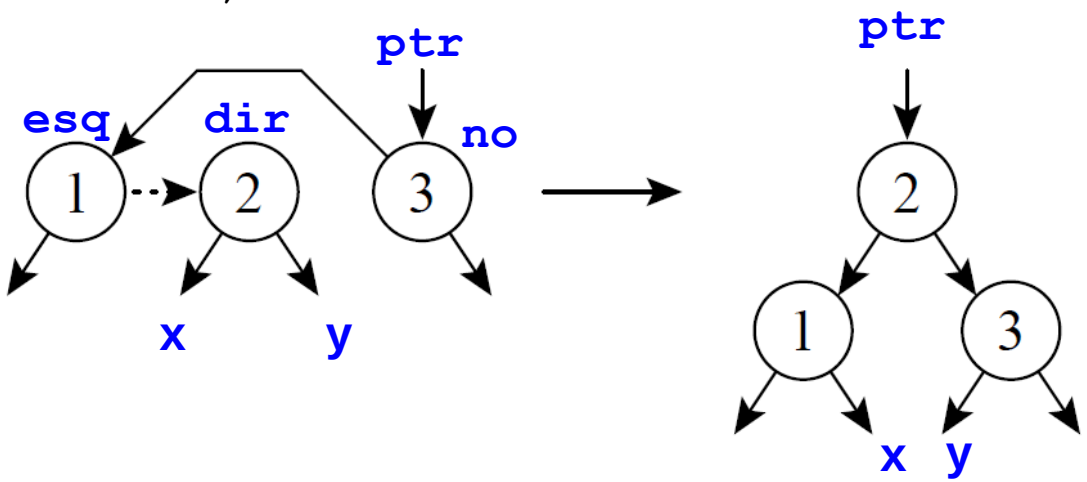
```
void dd(struct sbb **ptr) {
struct sbb *no = *ptr;
struct sbb *dir = no->dir;

    no->dir = dir->esq;      // rotE(ptr)
    dir->esq = no;
    dir->dirtipo = SBB_VERTICAL;
    no->dirtipo = SBB_VERTICAL;
    *ptr = dir;
}
```





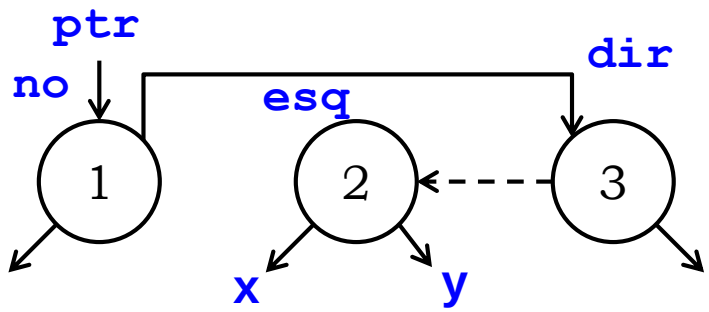
Transformações para manter propriedadas da árvore SBB - código

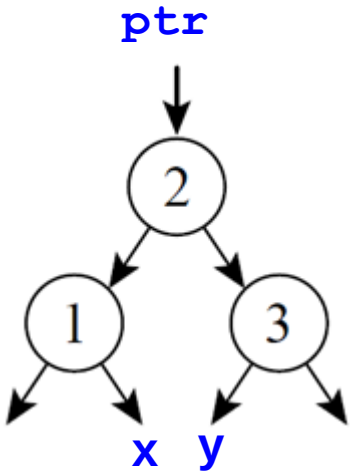
```
void ed(struct sbb **ptr) {
struct sbb *no = *ptr;
struct sbb *esq = no->esq;
struct sbb *dir = esq->dir;

    esq->dir = dir->esq;        // rotE(&(no->esq))
    dir->esq = esq;
    no->esq = dir->dir;         // rotD(ptr)
    dir->dir = no;
    esq->dirtipo = SBB_VERTICAL;
    no->esqtipo = SBB_VERTICAL;
    *ptr = dir;
}
```



Transformações para manter propriedadas da árvore SBB - código

```
void de(struct sbb **ptr) {
struct sbb *no = *ptr;
struct sbb *dir = no->dir;
struct sbb *esq = dir->esq;

     dir->esq = esq->dir;    // rotD(&(no->dir))
     esq->dir = dir;
     no->dir = esq->esq;     // rotE(ptr)
    esq->esq = no;
    dir->esqtipo = SBB_VERTICAL;
    no->dirtipo = SBB_VERTICAL;
    *ptr = esq;
}
```



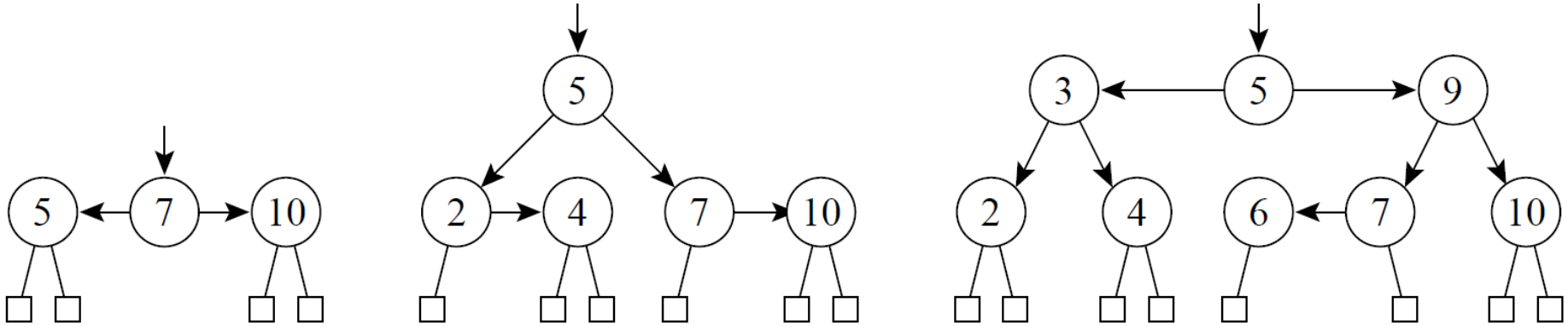


Exemplo de inserção em árvore SBB

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

Exemplo de inserção em árvore SBB

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



Inserção em árvores SBB

```
void iinsere(struct registro reg, struct sbb **ptr,
             int *incli, int *fim) {
    /* adiciona, pois encontrou uma folha */
    if(*ptr == NULL)
        iinsere_aqui(reg, ptr, incli, fim);

    /* busca na sub-árvore esquerda */
    } 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;
        }
    }
    /* continua */
```

Inserção em árvores SBB

```
    /* busca na sub-árvore direita */
    } else if (reg.chave > (*ptr)->reg.chave) {
        iinsere (reg, &(*ptr->dir), &(*ptr->dirtipo), fim);
        if (*fim) return;
        if (*ptr->dirtipo == SBB_VERTICAL) {
            *fim = TRUE;
        } else if (*ptr->dir->dirtipo == SBB_HORIZONTAL) {
            dd(ptr); *incli = SBB_HORIZONTAL;
        } else if (*ptr->dir->esqtipo == SBB_HORIZONTAL) {
            de(ptr); *incli = SBB_HORIZONTAL;
        }

    /* chave já existe */
    } else {
         printf("erro: chave já está na árvore.\n");
         *fim = TRUE;
    }
}
```

Inserção em árvores SBB

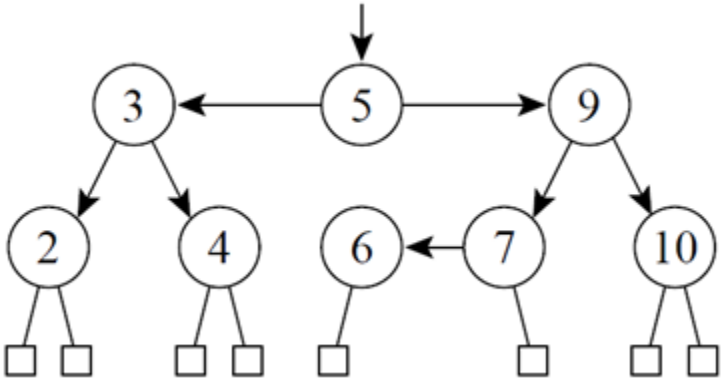
```
void iinsere_aqui(struct registro reg, struct sbb **ptr,
                  int *incli, int *fim) {

    struct sbb *no = malloc(sizeof(struct sbb));
    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;
}
```

Exemplo de inserção em árvore SBB

- Inserir a chave 5.5 na árvore a seguir



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: $C(n) = O(\lg(n))$