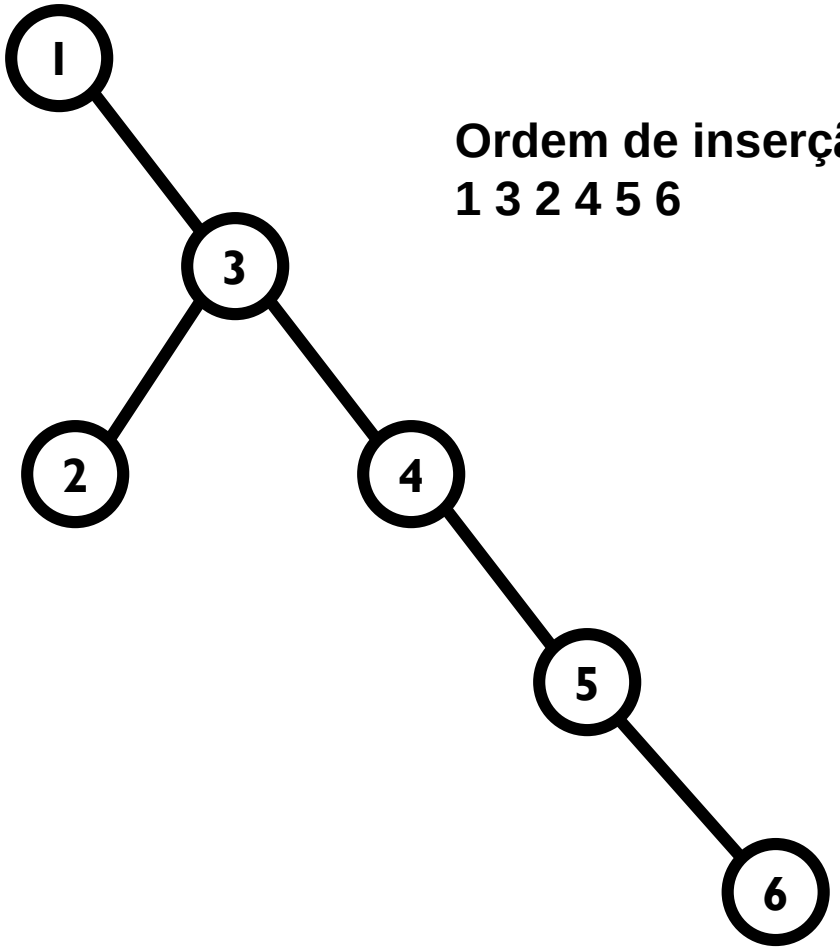
Árvores binárias de pesquisa com balanceamento

Algoritmos e Estruturas de Dados II

Árvores binárias de pesquisa

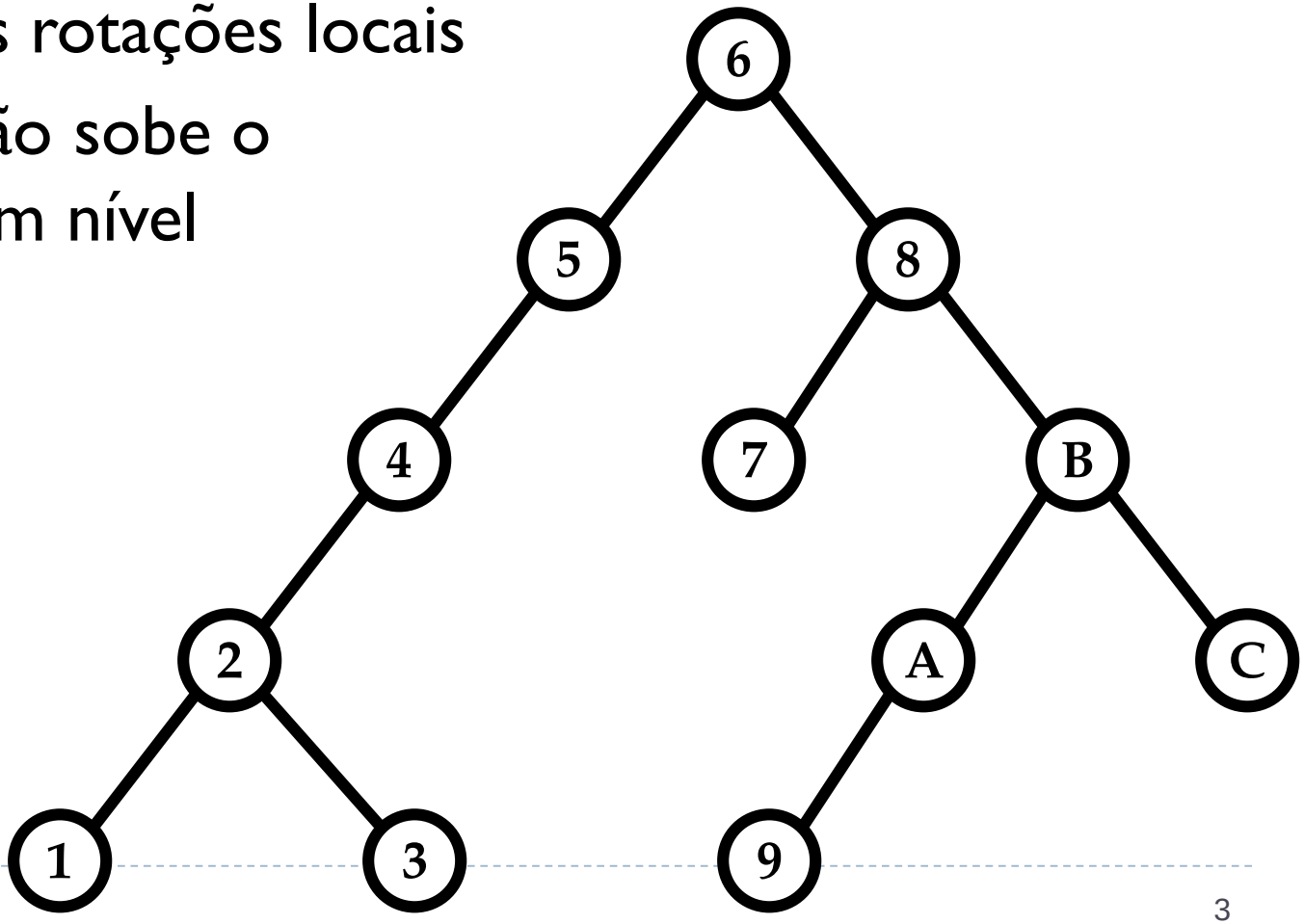
- Pior caso para uma busca é O(n)



Ordem de inserção:
1 3 2 4 5 6

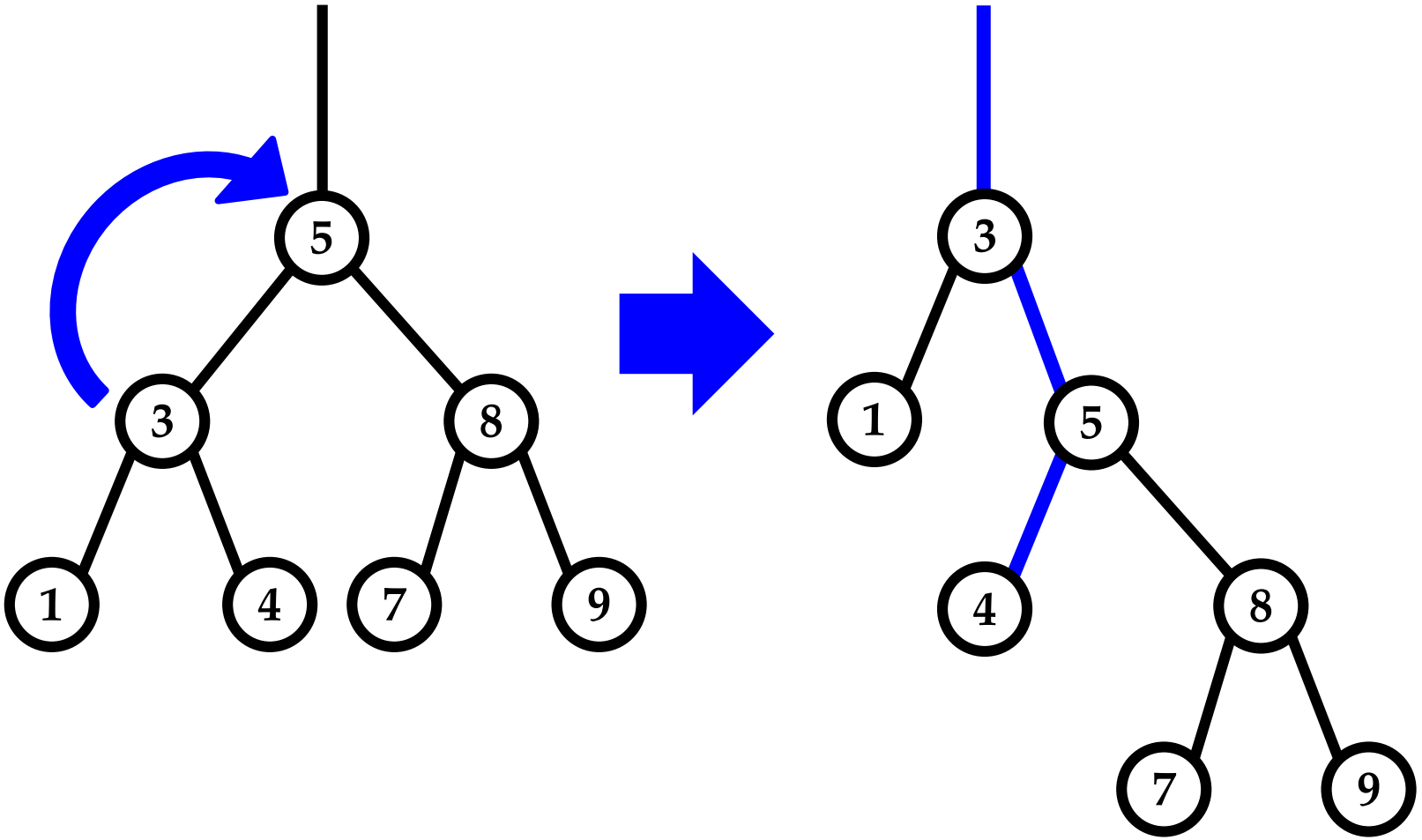
Inserção na raiz

- Adicionamos elemento como folha
- Subimos o novo elemento pra raiz
- Executamos rotações locais
- Cada rotação sobe o elemento um nível



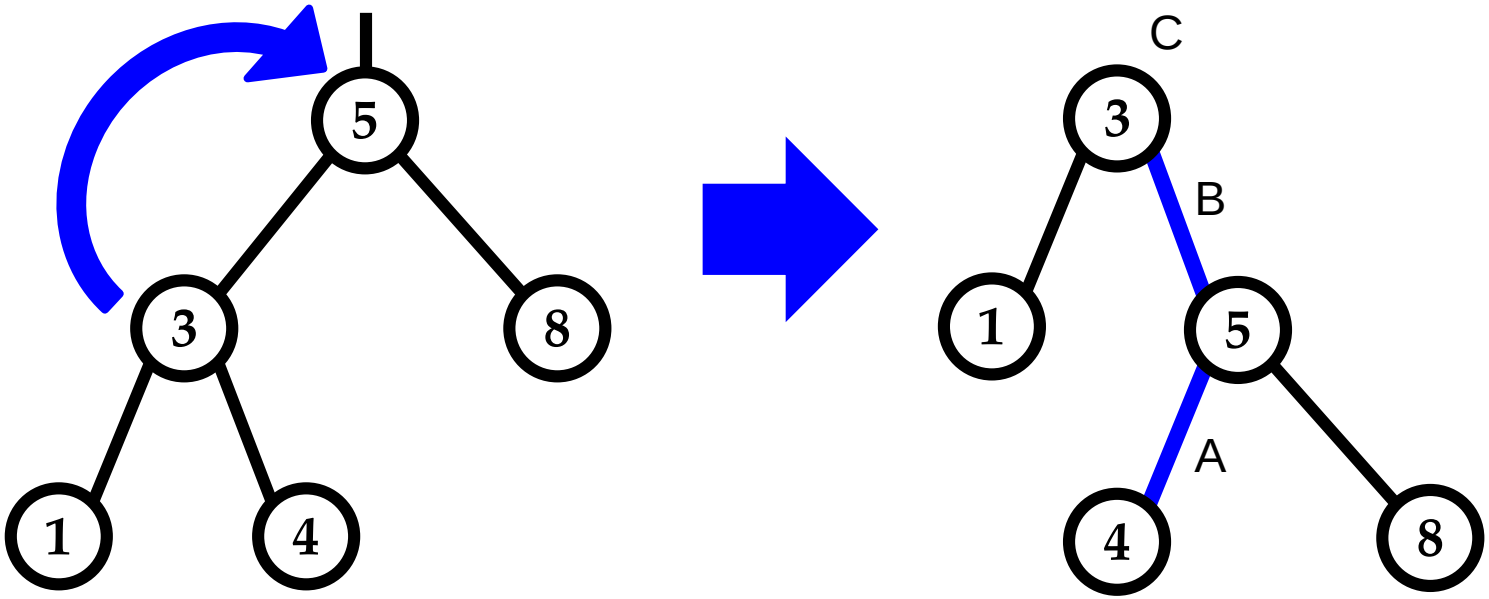
Rotação para direita

- Troca a raiz com o filho à esquerda



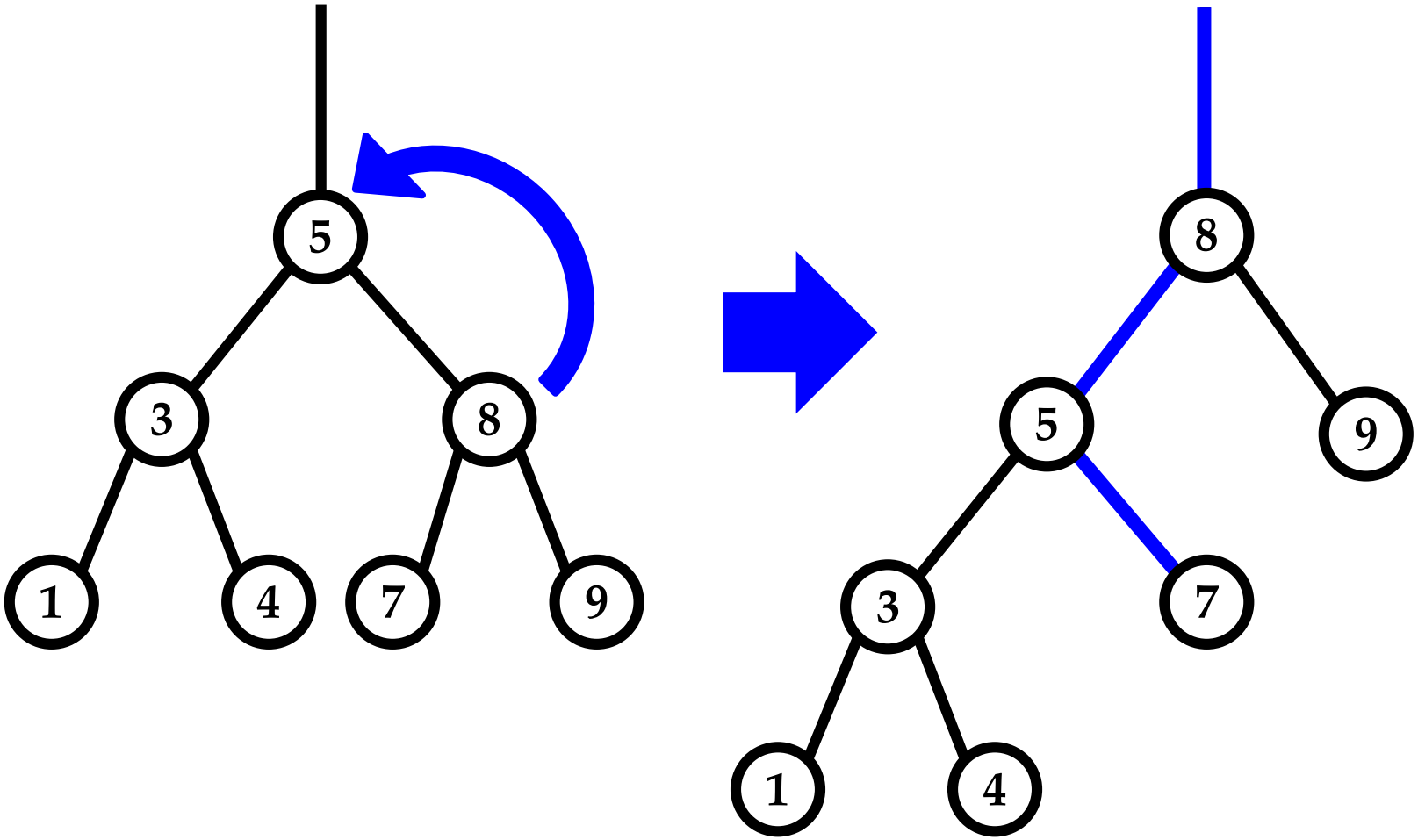
Rotação para direita

```
void rotD(struct arvore **ptr) {
    struct arvore *t = *ptr;          /* t = 5 */
    struct arvore *esq = t->esq;      /* esq = 3 */
    t->esq = esq->dir;                /* A */
    esq->dir = t;                     /* B */
    *ptr = esq;                       /* C */
}
```



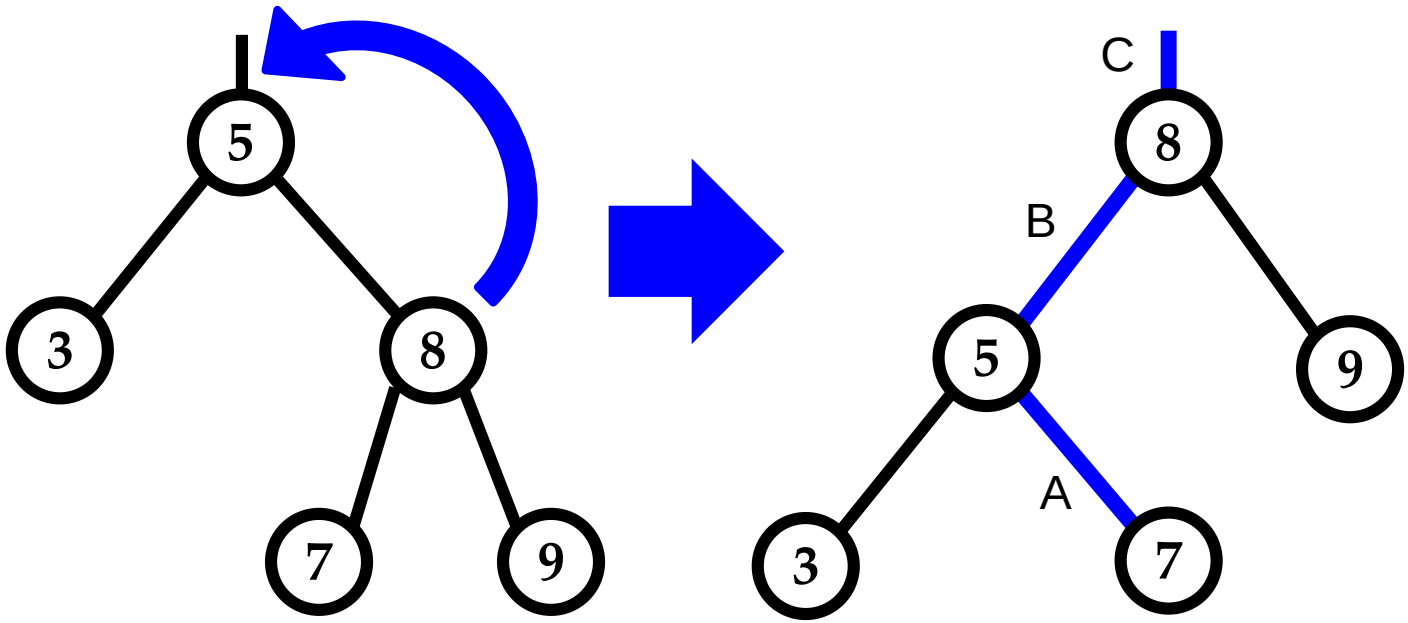
Rotação para esquerda

- Troca a raiz com o filho à direita



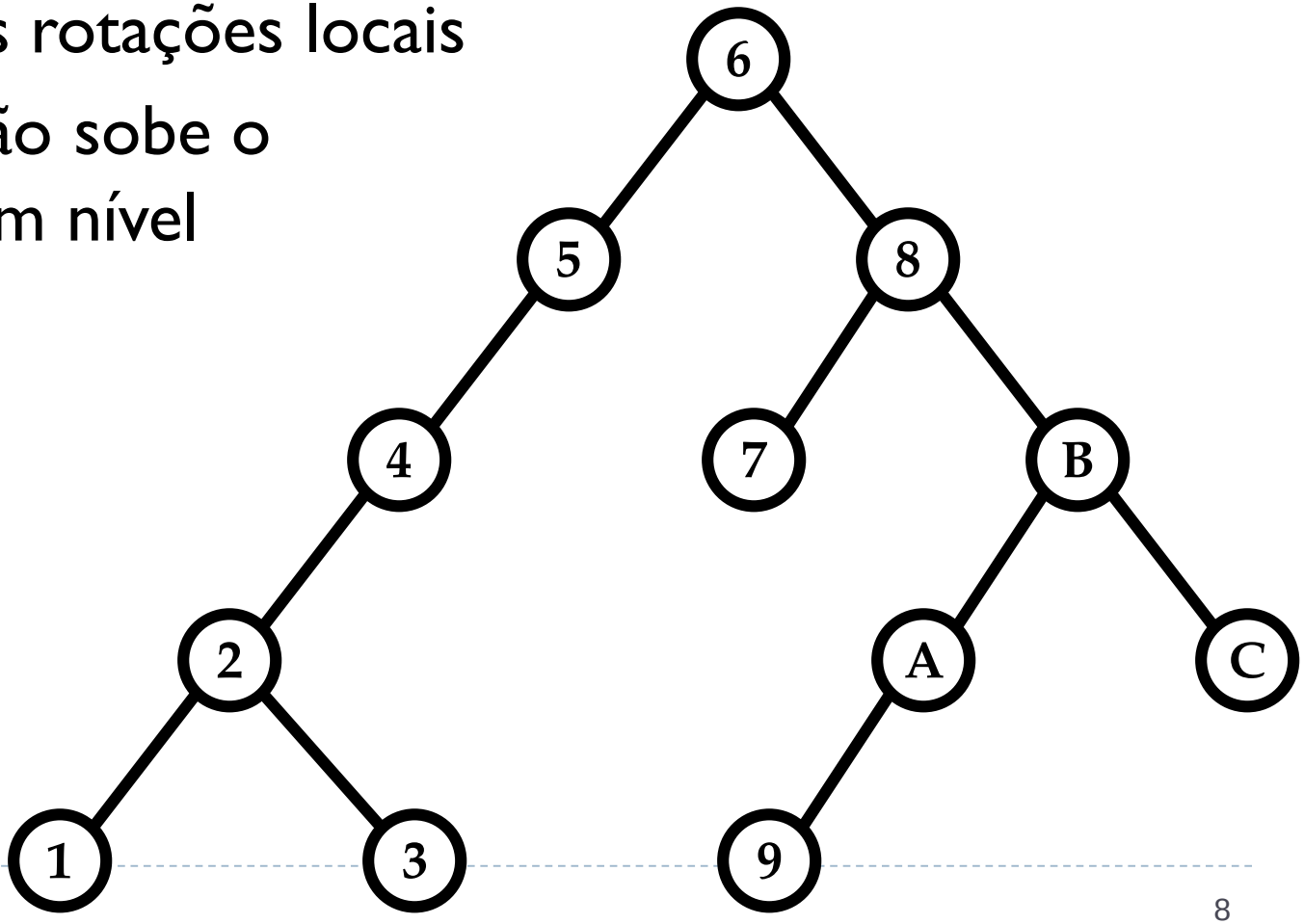
Rotação para esquerda

```
void rotE(struct arvore **ptr) {
    struct arvore *t = *ptr;          /* t = 5 */
    struct arvore *dir = t->dir;      /* dir = 8 */
    t->dir = dir->esq;                /* A */
    dir->esq = t;                     /* B */
    *ptr = dir;                       /* C */
}
```



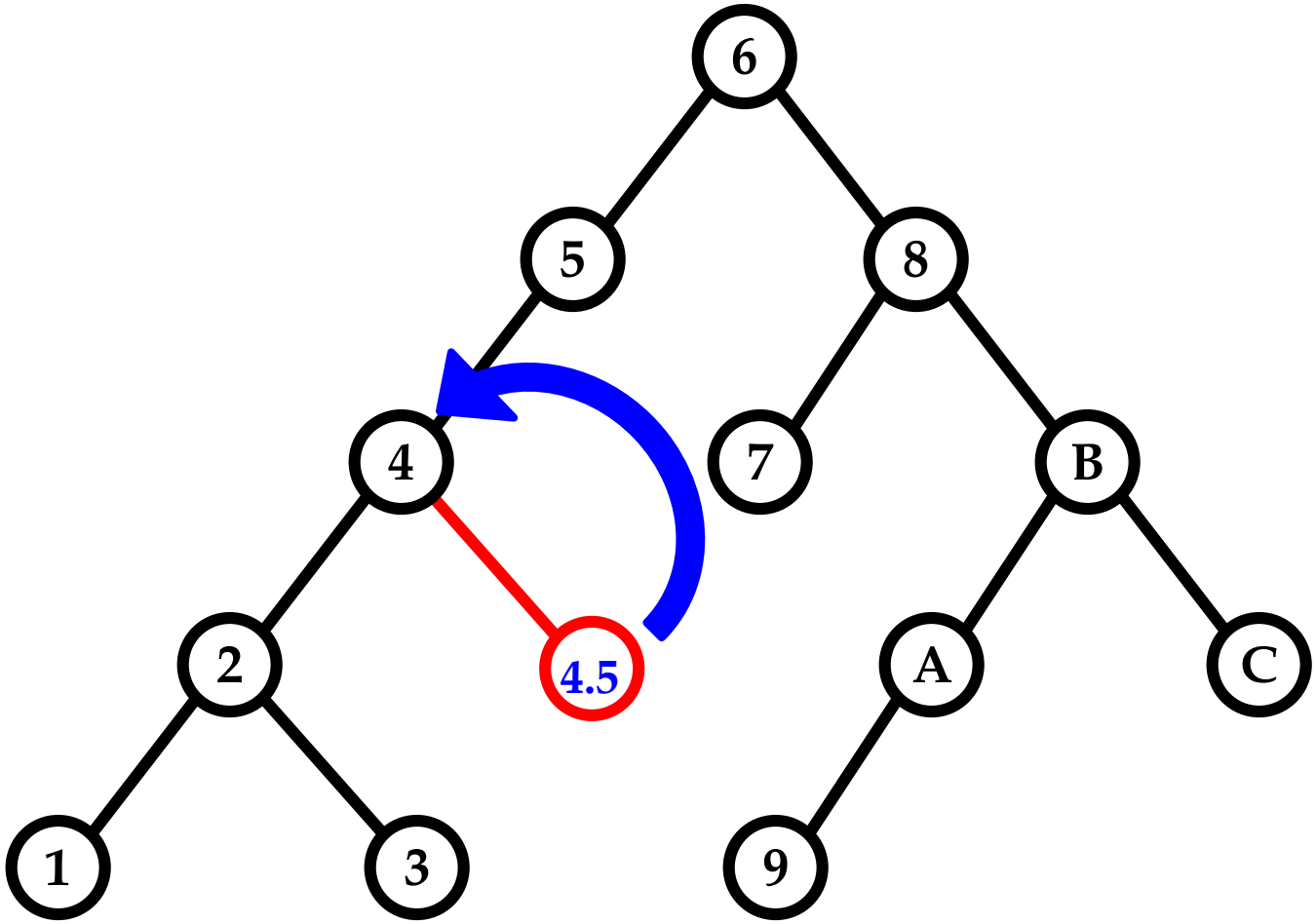
Inserção na raiz

- Adicionamos elemento como folha
- Subimos o novo elemento pra raiz
- Executamos rotações locais
- Cada rotação sobe o elemento um nível

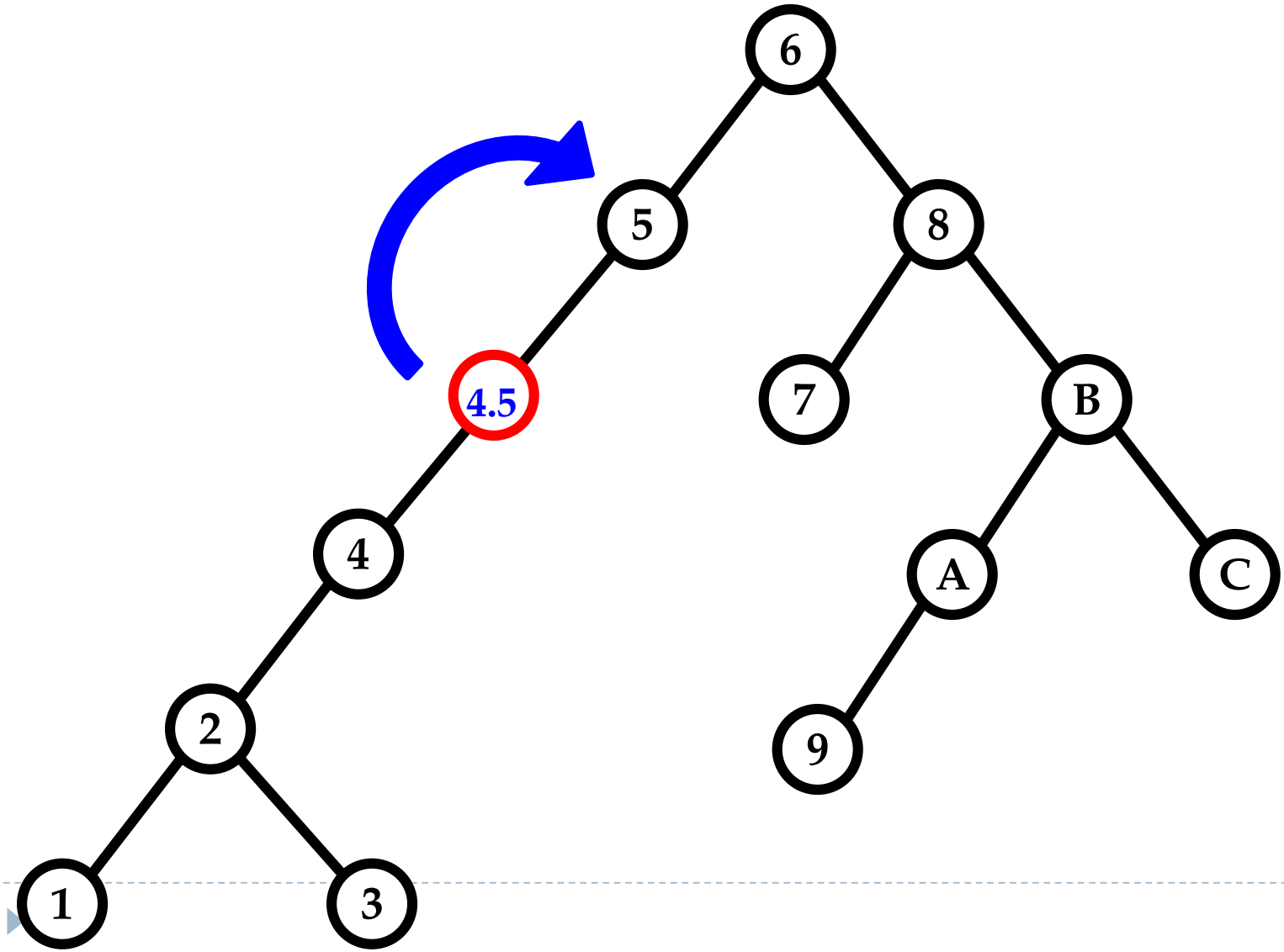


Inserção na raiz

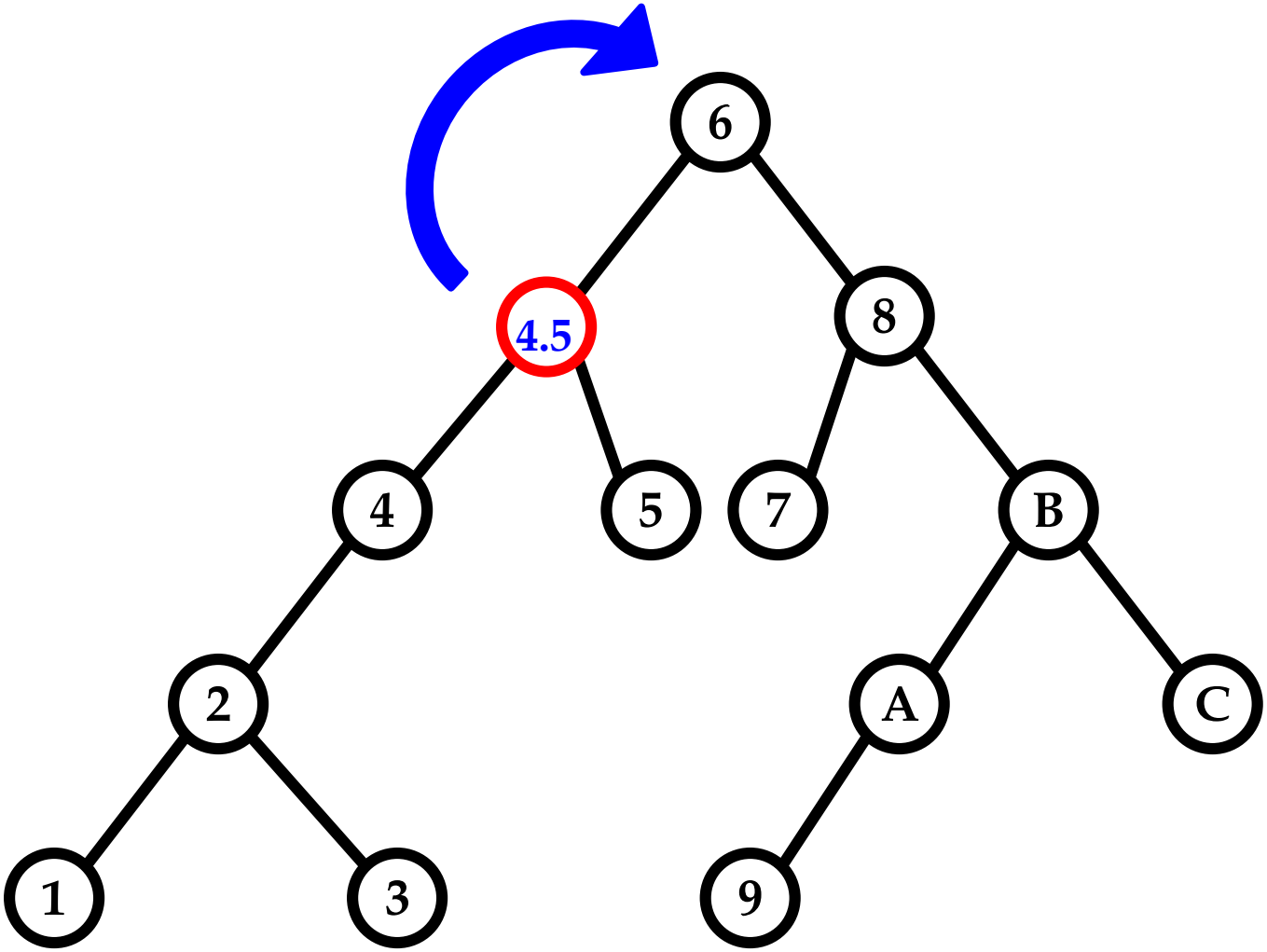
- Vamos inserir 4.5



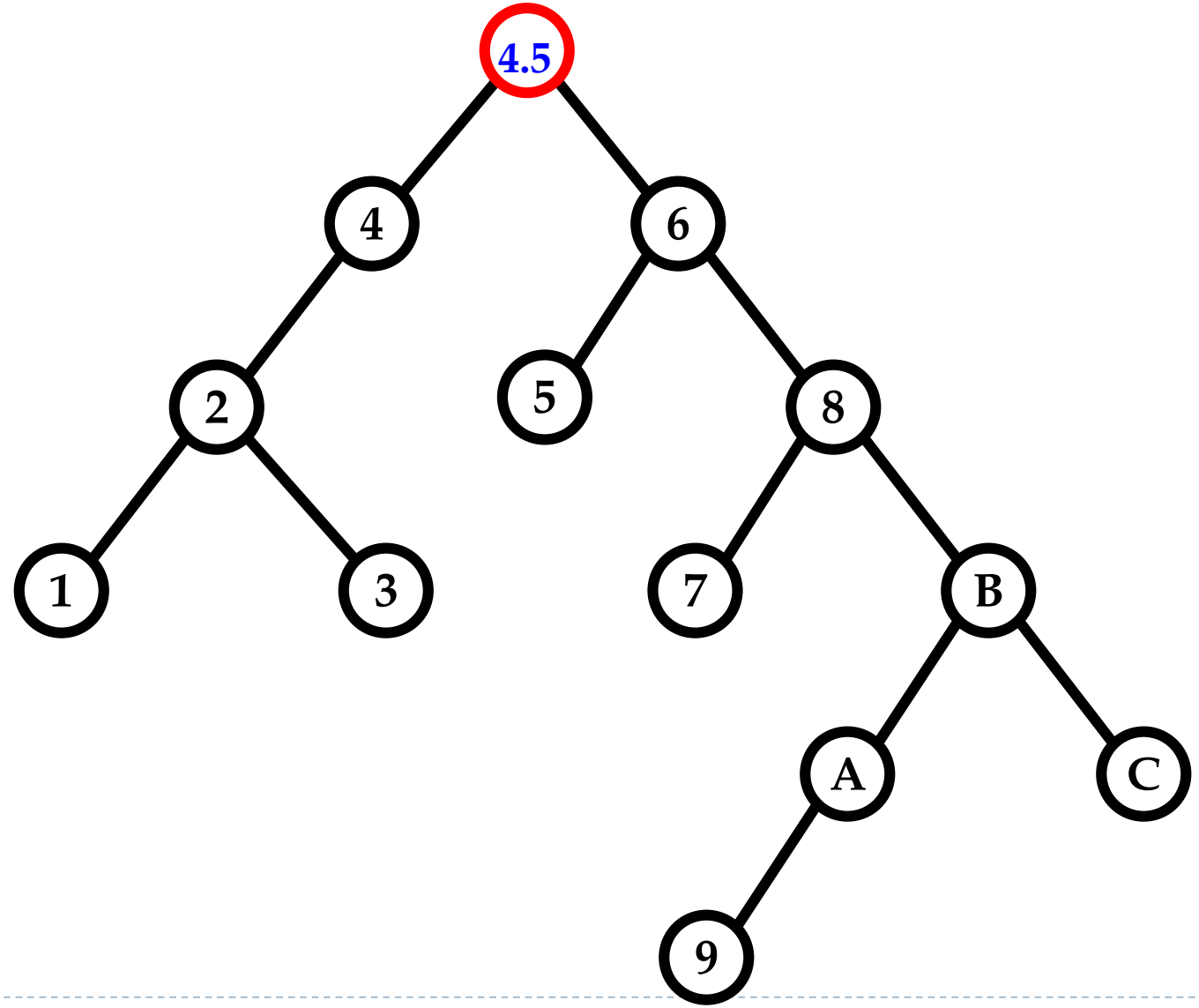
Inserção na raiz



Inserção na raiz



Inserção na raiz



Inserção na raiz

```
void insere_raiz(struct arvore **tptr, struct registro *reg)
{
    if(*tptr == NULL) {  // encontrou a folha
        struct arvore *novo = cria_arvore(reg);
        *tptr = novo;
        return;
    }
    struct arvore *t = *tptr;
    if(reg->chave < t->reg->chave) {
        insere_raiz(&(t->esq), reg);
        rotD(tptr);
    } else if(reg->chave > t->reg->chave) {
        insere_raiz(&(t->dir), reg);
        rotE(tptr);
    } else {
        printf("elemento ja existe\n");
    }
}
```

Inserção na raiz e pesquisa com reorganização

- Inserção na raiz pode ser útil quando aplicação busca por alguns elementos mais frequentemente
- Nesse caso, uma melhoria é modificar o método de inserção para colocar itens procurados na raiz
  - Busca por elementos próximos à raiz fica mais rápida
  - Caso médio e pior caso difíceis de analisar
    - Dependem do padrão de buscas da aplicação