CI1057: Algoritmos e Estruturas de Dados III
Árvores Binárias

Profa. Carmem Hara

Departamento de Informática/UFPR

17 de março de 2024

Árvore Binária de Busca - Interface

- ArvBusca insereArv(ItemArv, ArvBusca)
- ArvBusca insereArvIterativo(ItemArv, ArvBusca)
  Insere um novo Item na árvore como folha - função iterativa
- ArvBusca buscaArv(ItemArv, ArvBusca)
- ArvBusca buscaElementoK(int, ArvBusca)
- void visitaOrdenado(ArvBusca, void (*funcao)(ItemArv))
- ArvBusca insereNaRaiz(ItemArv, ArvBusca)
- ArvBusca removeArv(ItemArv, ArvBusca)
- ArvBusca juntaArv(ArvBusca, ArvBusca)

Busca na árvore binária de busca

```
1 ArvBusca buscaArv( ItemArv v, ArvBusca p ){
2   if( p == NULL )
3     return NULL;
4   if( eq( v, p->item))
5       return p;
6   if( lt(v, p->item ))
7     return buscaArv( v, p->esq );
8   return buscaArv( v, p->dir );
9 }
```

Visita Ordenada

```
1 void visitaOrdenado( ArvBusca p, void (*funcao)(ItemArv) ){
2   if( p == NULL )
3     return;
4   visitaOrdenado( p->esq, funcao );
5   funcao( p->item );
6   visitaOrdenado( p->dir, funcao );
7 }
```

Aplicação: Ordenação por árvore

Composta por 2 passos:

1. pré-processamento
   geração da árvore binária de busca
2. percurso da árvore em-ordem

Custo:

- pior caso: n^2
  quando os elementos são lidos em ordem ascendente ou descendente
- caso medio: $nlg(n)$
  a altura de uma árvore balanceada é $\lceil lg(n+1) \rceil$; Assim, para inserir cada um dos n elementos na árvore são necessárias no máximo $lg(n)$ comparações.

Para obter uma arvore balanceada:
após a entrada de uma chave k, metade dos elementos tem chave menor que k e metade tem chaves maiores que k.

Custo da busca

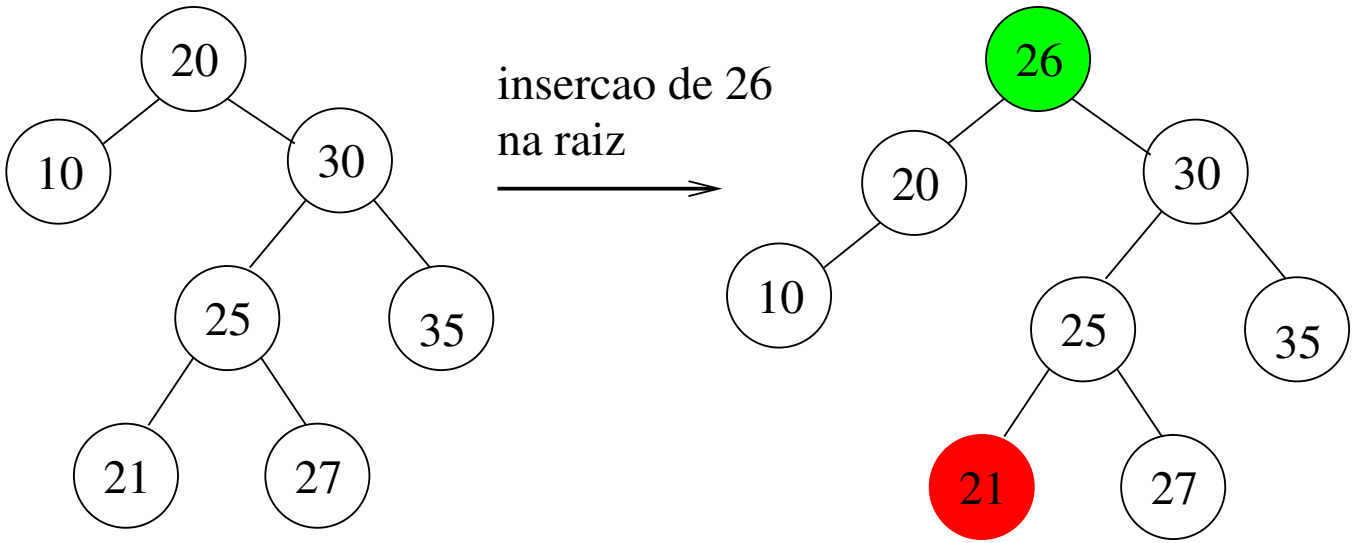
Custo médio de busca em uma árvore binária com n nós:
$(s + n)/n$, onde:

- s e' o COMPRIMENTO DO CAMINHO INTERNO, ou seja, a soma do comprimentos dos caminhos entre a raiz e cada um dos nós da árvore.
  Para acessar um determinado nó de nivel l são necessárias $(l + 1)$ comparações.
- considera que todos os nós tem igual probabilidade de serem acessados

Pode ser mostrado que o número esperado de comparações em uma árvore de pesquisa randômica e' $1.39lg(n)$, ou seja, somente 39% pior que a árvore completamente balanceada.

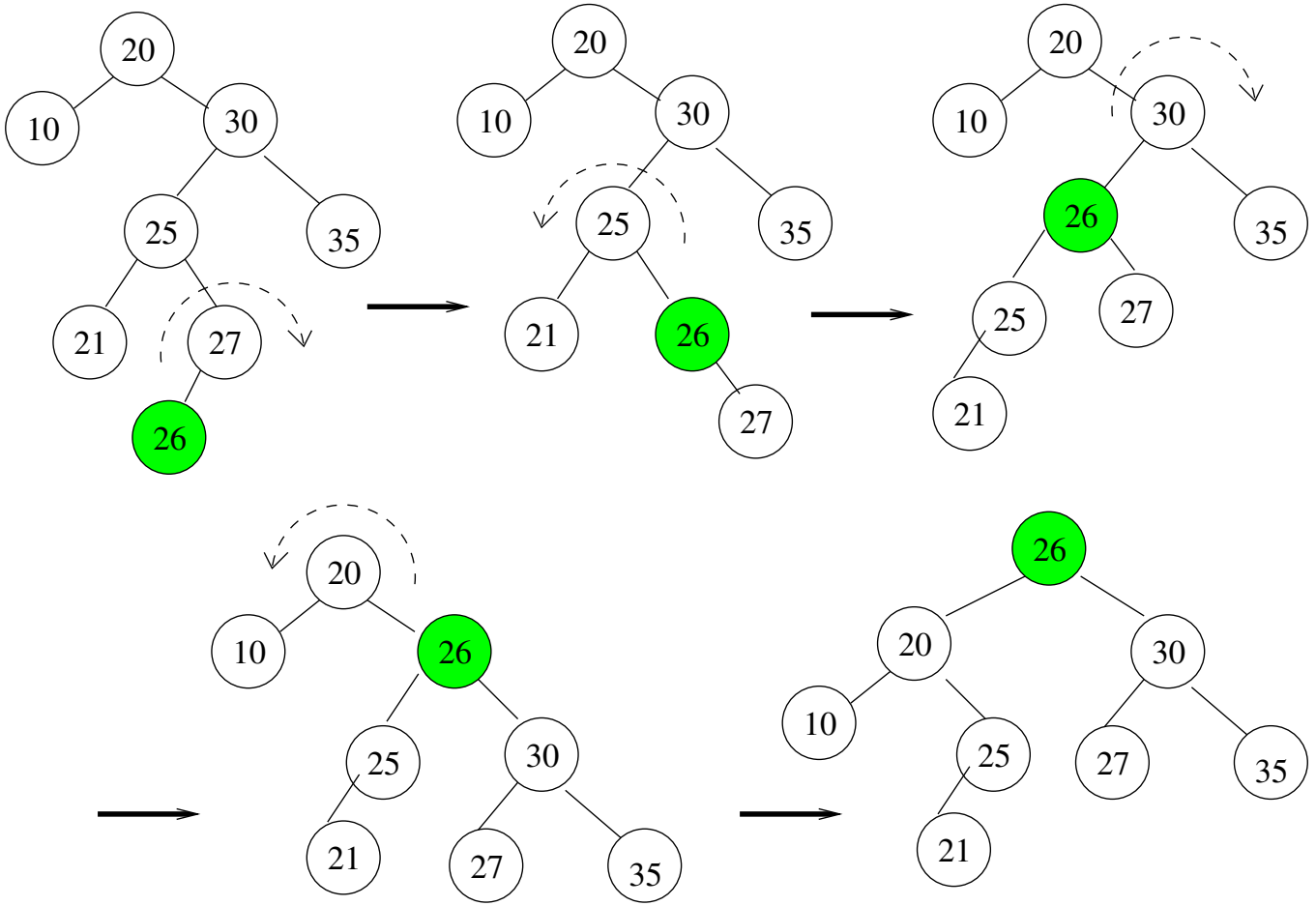
Suposição: a permutação dos n elementos é igualmente provável

Inserção na Raiz

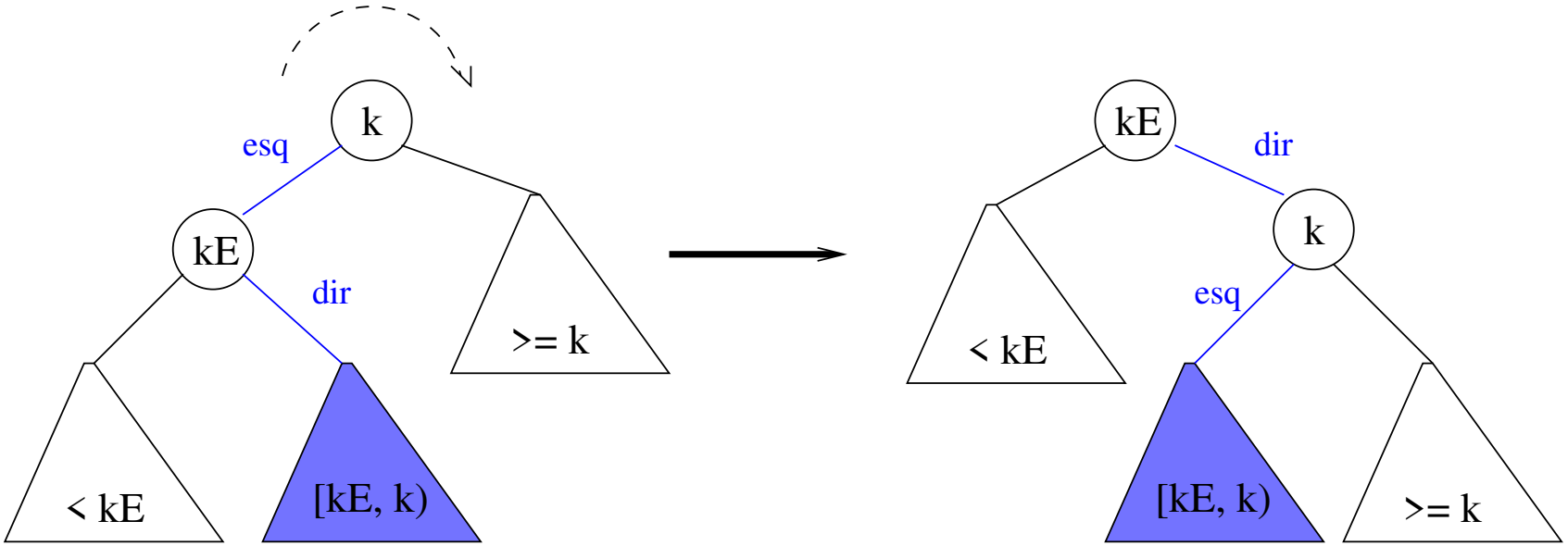


- Viola a definição da árvore binária de busca
- **Solução:** série de rotações após a inserção de 26 como uma folha

Inserção na Raiz0 com Rotações

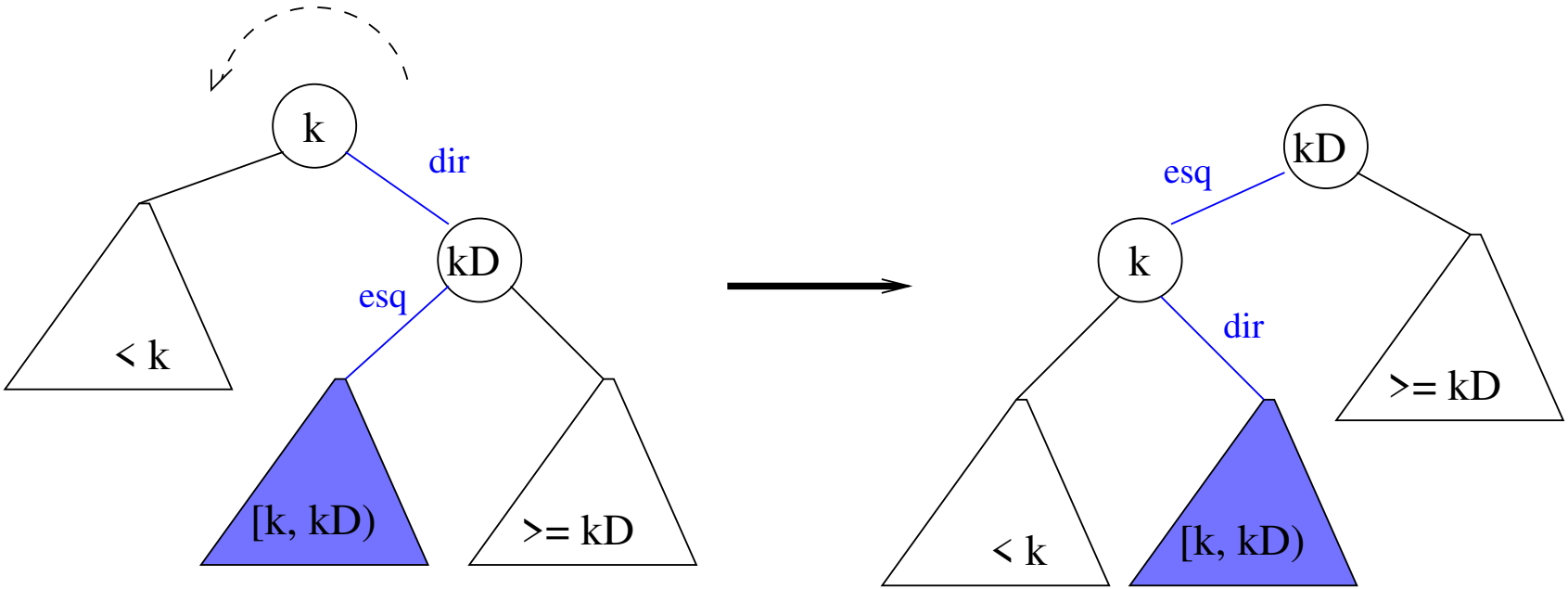


Rotação à Direita




```
1 ArvBusca rotacaoDir( ArvBusca p ){
2   ArvBusca esqP;
3
4   esqP = p->esq;  p->esq = esqP->dir; esqP->dir = p;
5   return esqP;
6 }
```

Rotação à Esquerda

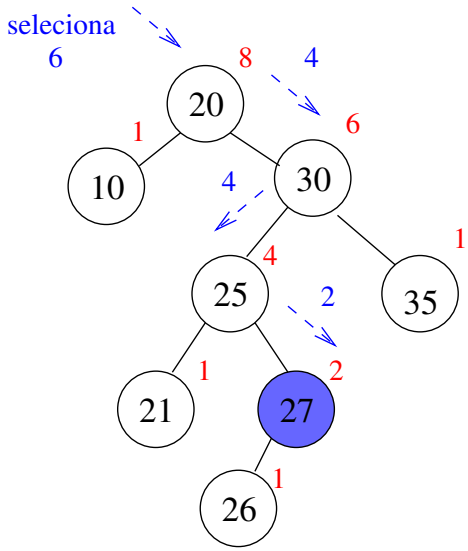



```
1 ArvBusca rotacaoEsq( ArvBusca p ){
2   ArvBusca dirP;
3
4   dirP = p->dir;  p->dir = dirP->esq; dirP->esq = p;
5   return dirP;
6 }
```

Pergunta

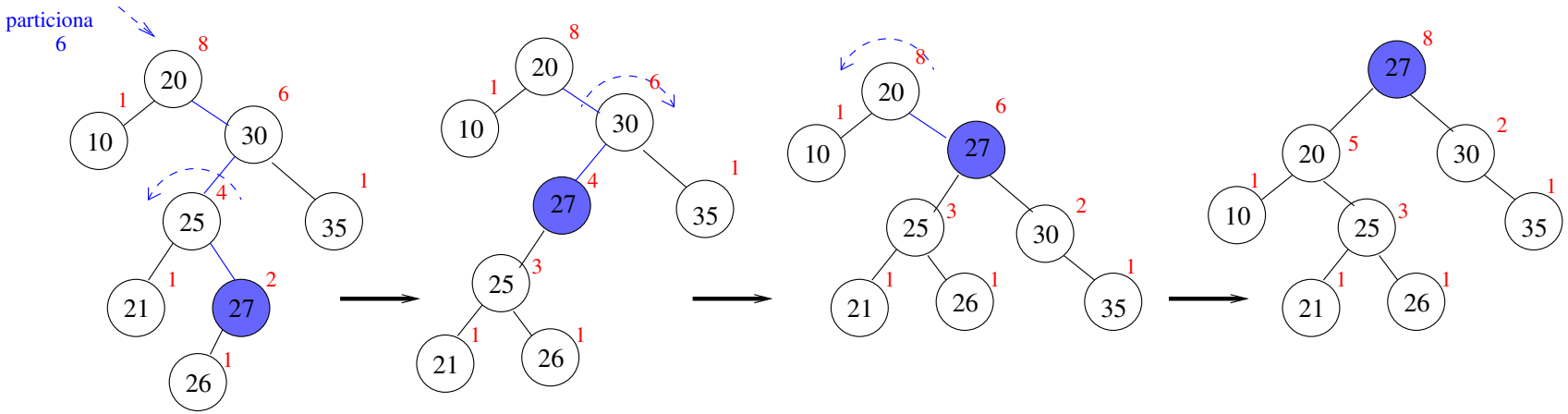
Como atualizar os contadores de nodos (n)?

Seleção do n-ésimo elemento da árvore




```
1  ArvBusca buscaElementoK( int k, ArvBusca p ){
2    int contEsq;
3
4    if( p == NULL )
5      return NULL;
6    contEsq= ( p->esq == NULL )?0: p->esq->n;
7    if( k == contEsq+1 )
8      return p;
9    else if( k <= contEsq )
10     return buscaElementoK( k, p->esq );
11   else
12     return buscaElementoK( k-contEsq-1, p->dir );
13 }
```

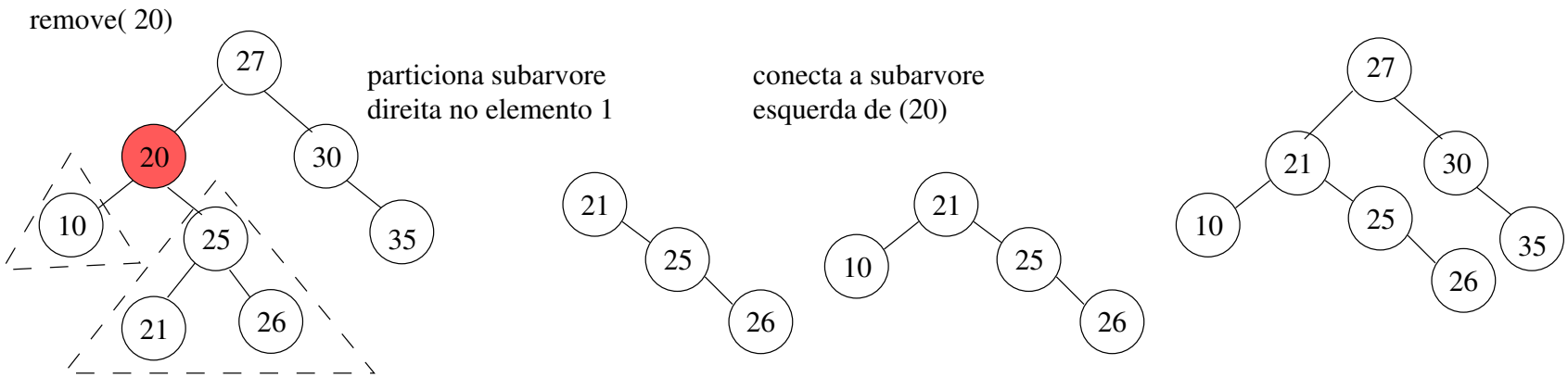
Partição no n-ésimo elemento da árvore



Partição no n-ésimo elemento - Implementação

```
1  ArvBusca particaoR( int k, ArvBusca p ){
2    int contEsq;
3
4    contEsq= (p->esq==NULL)?0: p->esq->n;
5    if( k == contEsq+1 )
6      return p;
7    else if( k <= contEsq ){
8      p->esq= particaoR( k, p->esq );
9      return rotDireita( p );
10   }
11   else{
12     p->dir= particaoR( k-contEsq-1, p->dir );
13     return rotEsquerda( p );
14   }
15 }
16
17 ArvBusca particao( int k, ArvBusca raiz ){
18   if( raiz->n < k )
19     return raiz;
20   else
21     return particaoR( k, raiz );
22 }
```

Remoção de elemento da árvore



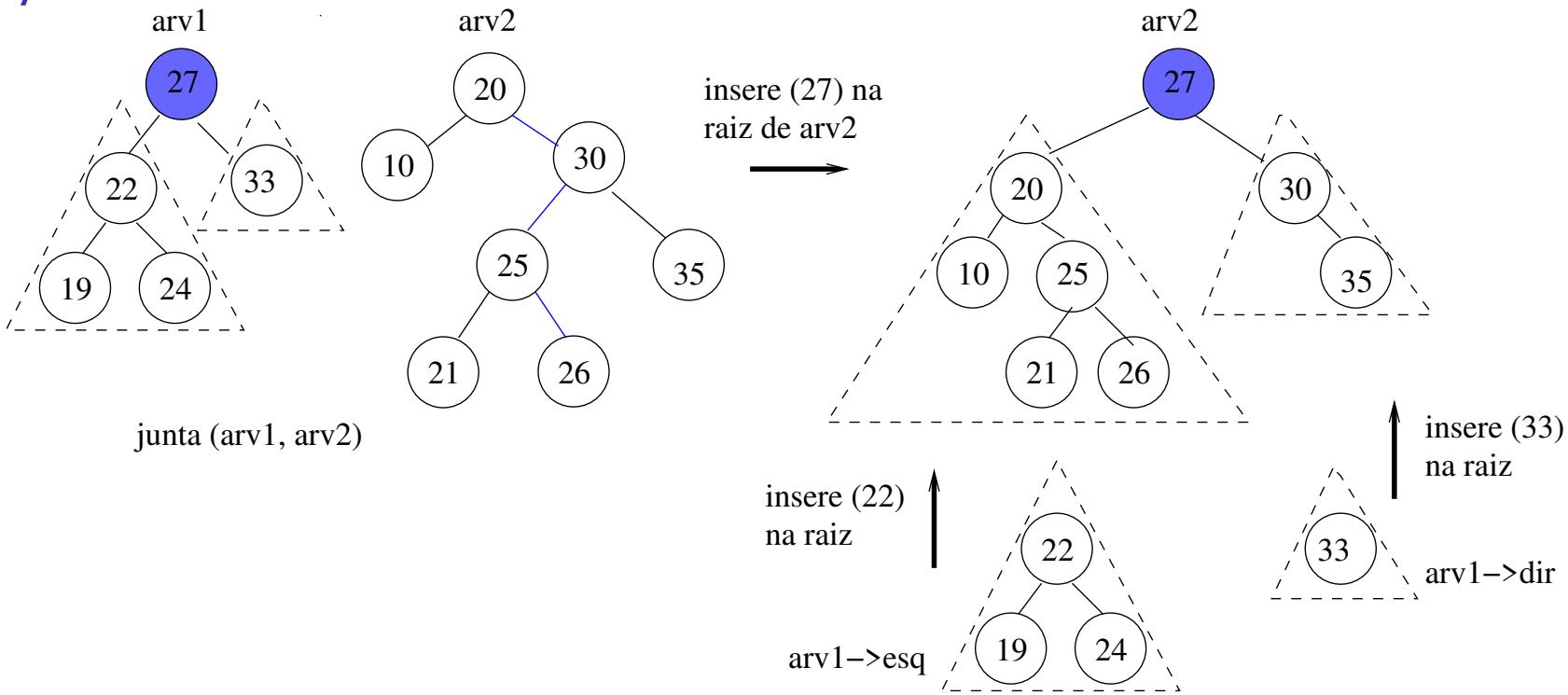
Ideia geral:

1. ao encontrar o elemento a ser removido, criar duas subárvores com os seus filhos direito e esquerdo.
2. junta as árvores colocando como raiz o menor elemento da subárvore à direita ou o maior elemento da subárvore à esquerda.

Remoção de elemento da árvore - Implementação

```
1  ArvBin juntaEsqDir( ArvBin e, ArvBin d ){
2    if( d == nodoNull )
3      return e;
4    d = particao( d, 1 );
5    d->esq = e;
6    return d;
7  }
8
9  ArvBin removeArv( ItemArv v, ArvBin p ){
10   ArvBin nodoV;
11
12   if( p == NULL) return p;
13   if( lt( v, p->item) )
14     p->esq = removeArv( v, p->esq );
15   else if( gt( v, p->item) )
16     p->dir = removeArv( v, p->dir );
17   else {
18     nodoV = p;
19     p = juntaEsqDir( p->esq, p->dir );
20     free( nodoV );
21   }
22   return p;
23 }
```

Junção de duas árvores binárias de busca



1. insere a raiz (r1) de arv1 na raiz de arv2: com isso, as subárvores esquerda e direita de arv1 e arv2 conterão valores menores e maiores que r1, respectivamente.
2. inserir recursivamente $arv1->dir$ na raiz da subárvore direita de arv2 e $arv1->esq$ na raiz da subárvore esquerda de arv2.

Junção de árvores - Implementação

```
1 ArvBin juntaArv( ArvBin r1, ArvBin r2 ){
2   if( r2 == NULL) return r1;
3   if( r1 == NULL) return r2;
4   r2 = insereNaRaiz( r1->chave, r2 );
5   r2->esq = juncao( r1->esq, r2->esq );
6   r2->dir = juncao( r1->dir, r2->dir );
7   free( r1 );
8   return r2;
9 }
```

Referências

- Secoes 12.5, 12.9 (Sedgewick)