Árvores Binárias de Busca

Definições

- Uma Árvore Binária de Busca possui as mesmas propriedades de uma AB, acrescida das seguintes propriedades:
  - Os nós pertencentes à sub-árvore esquerda possuem valores menores do que o valor associado ao nó-raiz X
  - Os nós pertencentes à sub-árvore direita possuem valores maiores do que o valor associado ao nó-raiz X
  - Não há elementos duplicados
  - Um percurso pré-ordem nessa árvore resulta na seqüência de valores em ordem crescente

Definições

- Se invertessemos as propriedades descritas na definição anterior,
  - de maneira que a sub-árvore esquerda de um nó contivesse valores maiores e a sub-árvore direita valores menores, o percurso in-ordem resultaria nos valores em ordem decrescente

- Uma árvore de busca criada a partir de um conjunto de valores não é única: o resultado depende da seqüência de inserção dos dados

Definições

- A grande utilidade da árvore binária de busca é armazenar dados contra os quais outros dados são freqüentemente verificados (busca!)

- Uma árvore de binária de busca é dinâmica e pode sofrer alterações (inserções e remoções de nós) após ter sido criada

Operações em ABB's

- **TAD ABB:**
- Definir (igual a AB)
- Remover
- Inserir
- Imprimir elementos em seqüência (igual a AB)
- Busca

Encadeada dinâmica

```
typedef  int tipo_elem;

struct nodetype {
   tipo_elem info;
   struct nodetype *esq;
   struct nodetype *dir;
}
typedef struct nodetype *NODEPTR;
```

Inserção (operações em ABB's)

- Passos do algoritmo de inserção
  - Procure um “local” para inserir o novo nó, começando a procura a partir do nó-raiz;
  - Para cada nó-raiz de uma sub-árvore, compare; se o novo nó possui um valor menor do que o valor no nó-raiz (vai para sub-árvore esquerda), ou se o valor é maior que o valor no nó-raiz (vai para sub-árvore direita);
  - Se um ponteiro (filho esquerdo/direito de um nó-raiz) nulo é atingido, coloque o novo nó como sendo filho do nó-raiz.

Inserção

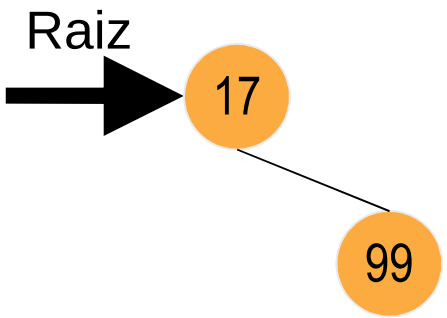
- Para entender o algoritmo considere a inserção do conjunto de números, na seqüência

  {17,99,13,1,3,100,400}

- No início a ABB está vazia!

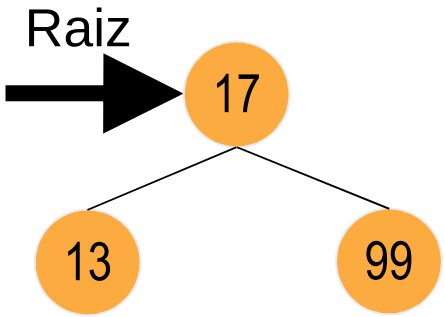
Inserção

- O número 17 será inserido tornando-se o nó raiz
- A inserção do 99 inicia-se na raiz. Compara-se 99 c/ 17.
- Como 99 > 17, 99 deve ser colocado na sub-árvore direita do nó contendo 17 (subárvore direita, inicialmente, nula)



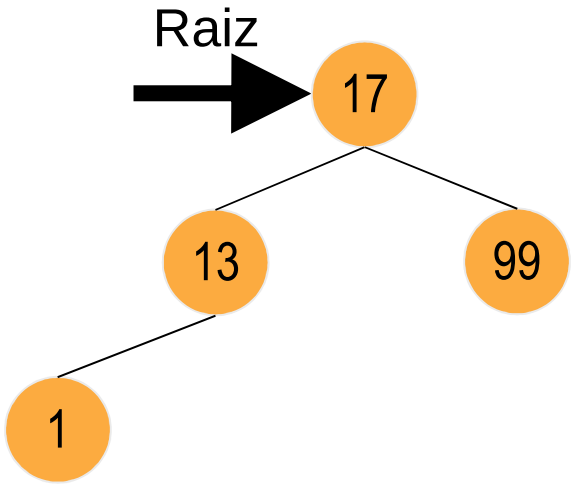
Inserção

- A inserção do 13 inicia-se na raiz
- Compara-se 13 c/ 17. Como 13 < 17, 13 deve ser colocado na sub-árvore esquerda do nó contendo 17
- Já que o nó 17 não possui descendente esquerdo, 13 é inserido na árvore nessa posição



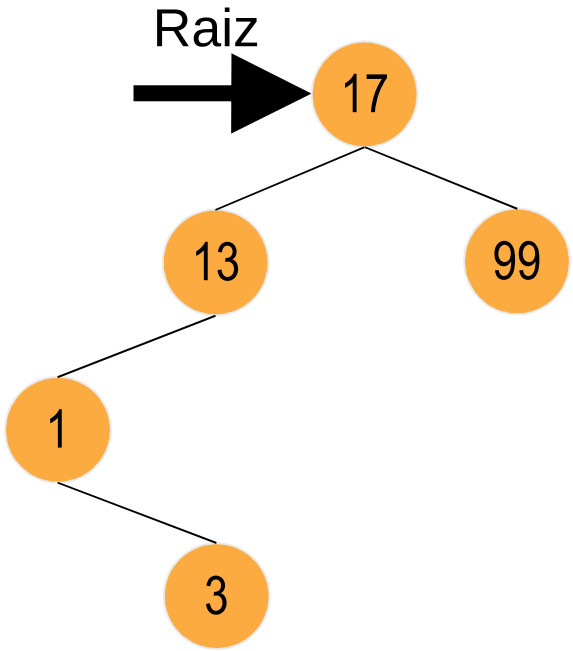
Inserção

- Repete-se o procedimento para inserir o valor 1
- 1<17, então será inserido na sub-árvore esquerda
- Chegando nela, encontra-se o nó 13, 1<13 então ele será inserido na sub-árvore esquerda de 13



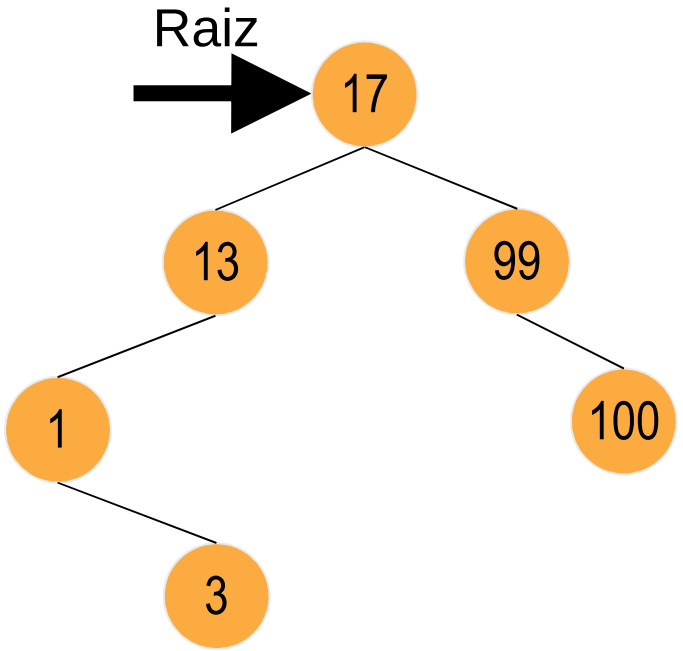
Inserção

- Repete-se o procedimento para inserir o elemento 3:
  - 3 < 17;
  - 3 < 13
  - 3 > 1



Inserção

- Repete-se o procedimento para inserir o elemento 100:
  - 100 > 17
  - 100 > 99



Inserção

- Repete-se o procedimento para inserir o elemento 400:
  - 400 > 17
  - 400 > 99
  - 400 > 100

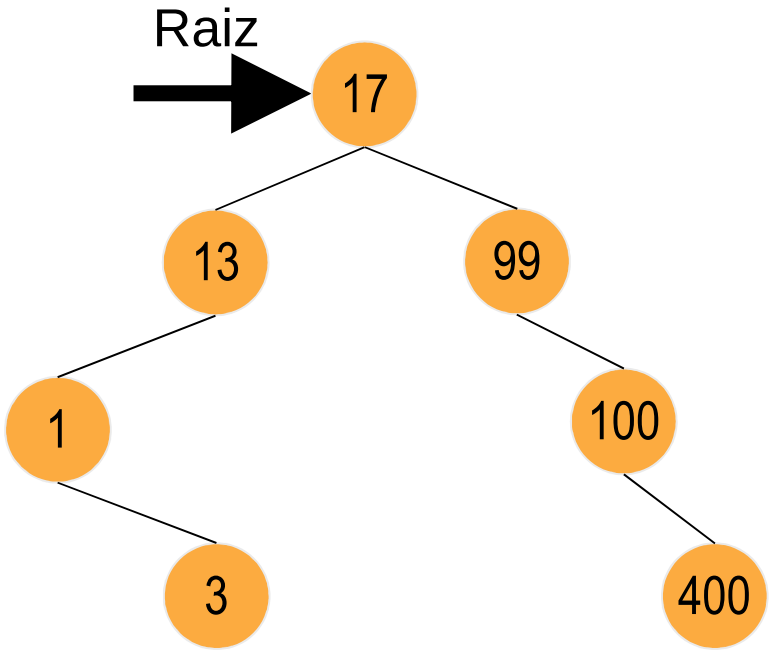
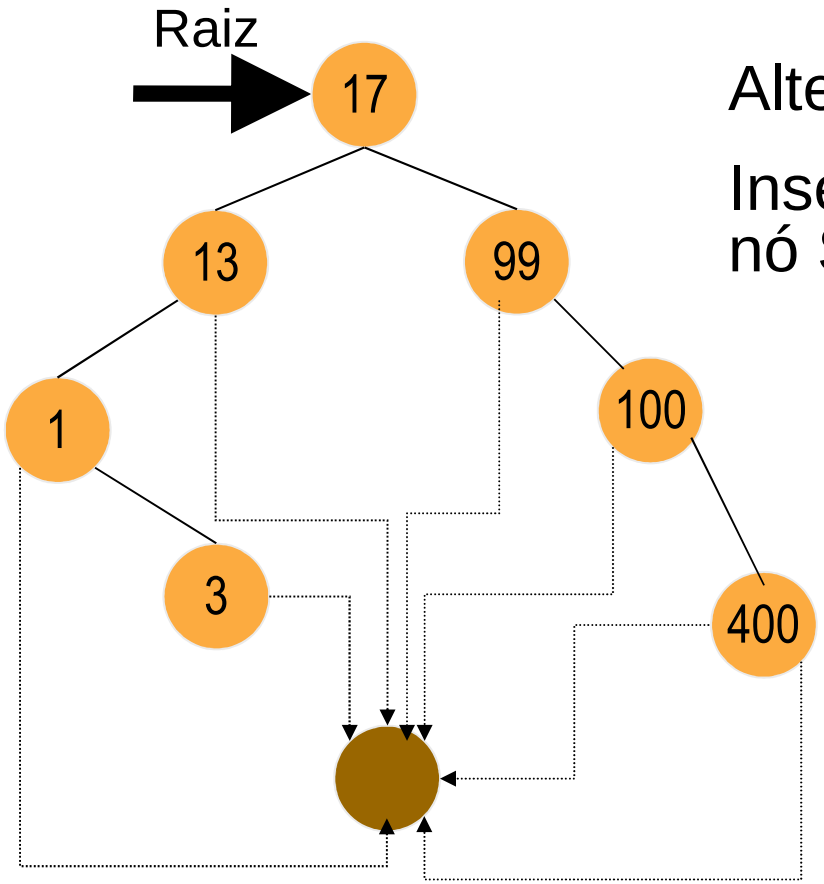


ABB com sentinela



Altera a operação definir

Insere valor procurado no nó Sentinela. Vantagem??

Busca (operações em ABB's)

- Passos do algoritmo de busca
  - Comece a busca a partir do nó-raiz;
  - Para cada nó-raiz de uma sub-árvore compare: se o valor procurado é menor que o valor no nó-raiz (continua pela sub-árvore esquerda), ou se o valor é maior que o valor no nó-raiz (sub-árvore direita);
  - Caso o nó contendo o valor pesquisado seja encontrado, retorne um ponteiro para o nó; caso contrário, retorne um ponteiro nulo.

Busca

```
pno * busca (Arv raiz, tipo_elem valor){
pno q;
 busca= NULL; q = raiz;
   while (q != NULL) {
      if (valor < q->info) {
              q = q->esq;
      }
      else if (valor > q ->info) {
             q = q ->dir;
              }
              else {
               busca = q; q= NULL;
              }
   }
}
```

Algoritmo de busca na ABB c/ sentinela

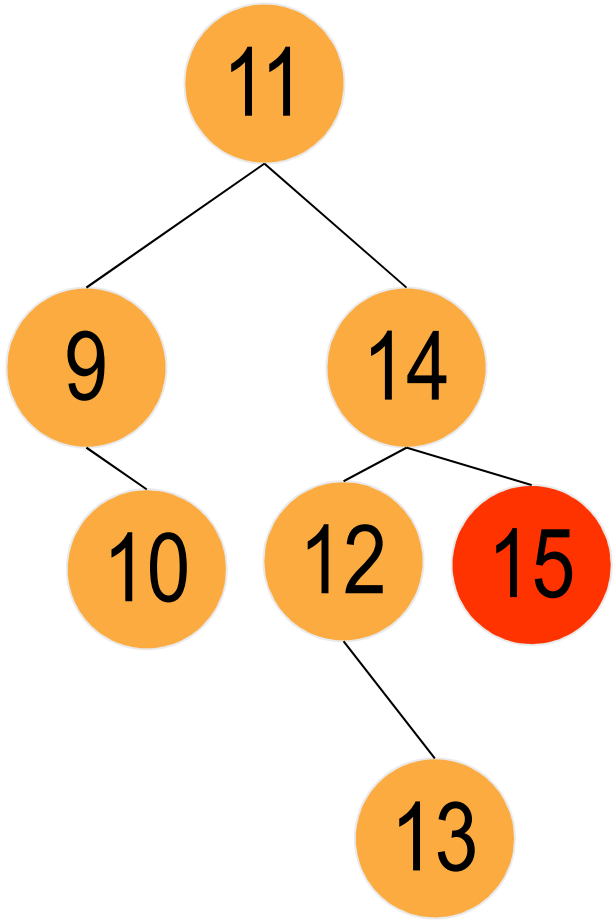
```
pno*Busca (tipo_elemen valor, pno *raiz, pno *sentinela){
pno  *p;
     if raiz != NULL{
       p=raiz;  sentinela->info=valor;
         while (p->info != valor) {
               if (p->info < valor) {
                  p = p->dir;
              }
                 else {
                    p = p->esq;
              }
          if (p == sentinela) {Busca=NULL;}
         else Busca= p;
   else Busca= NULL;
}
```

Remoção (operações em ABB's)

- Casos a serem considerados no algoritmo de remoção de nós de uma ABB:
  - Caso 1: o nó é folha
    - O nó pode ser retirado sem problema;
  - Caso 2: o nó possui uma sub-árvore (esq./dir.)
    - O nó-raiz da sub-árvore (esq./dir.) “ocupa” o lugar do nó retirado;
  - Caso 3: o nó possui duas sub-árvores
    - O nó contendo o menor valor da sub-árvore direita pode “ocupar” o lugar; ou o maior valor da sub-árvore esquerda pode “ocupar”

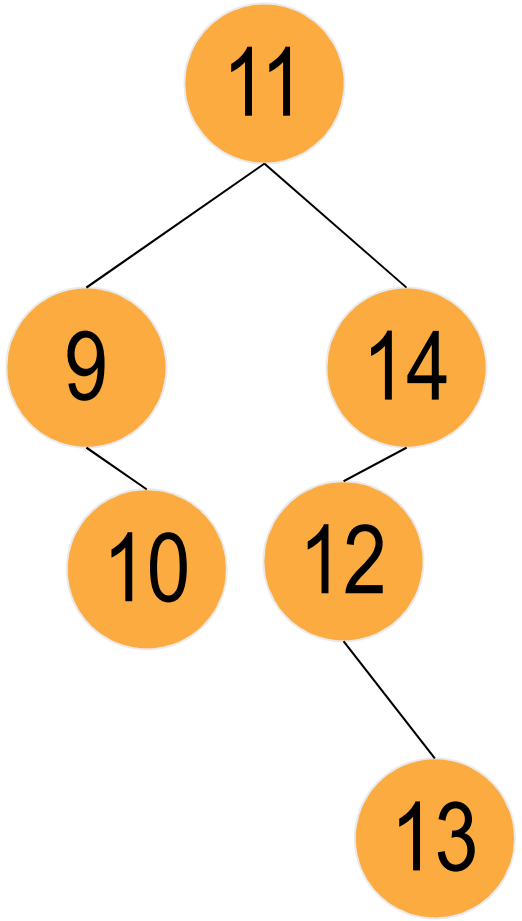
Remoção – Caso 1

- Caso o valor a ser removido seja o 15
- pode ser removido sem problema, não requer ajustes posteriores



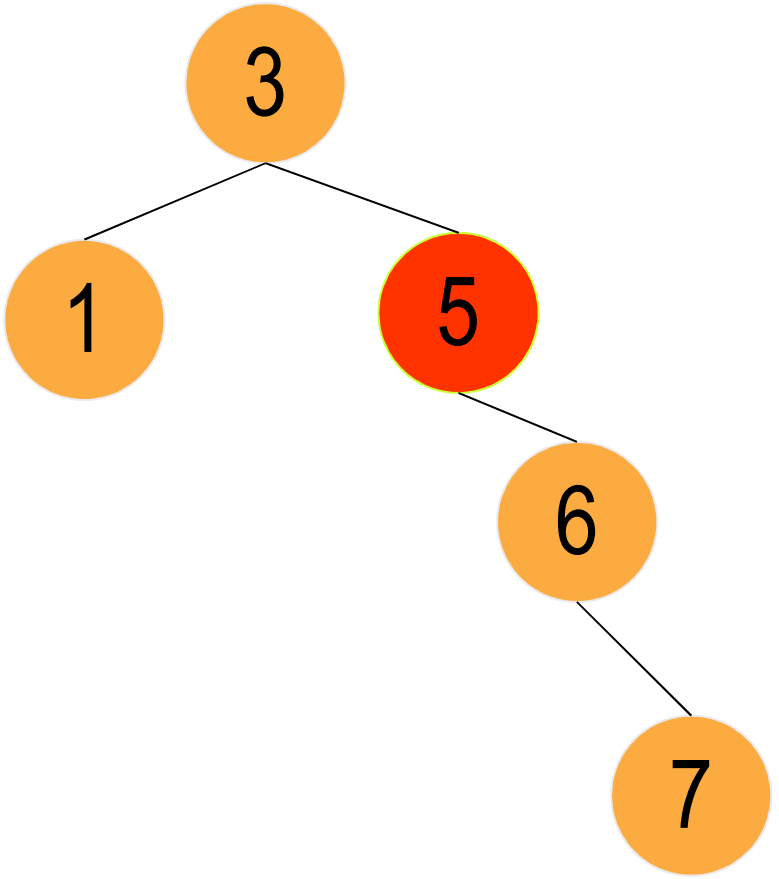
Remoção – Caso 1

- Os nós com os valores 10 e 13 também podem ser removidos!



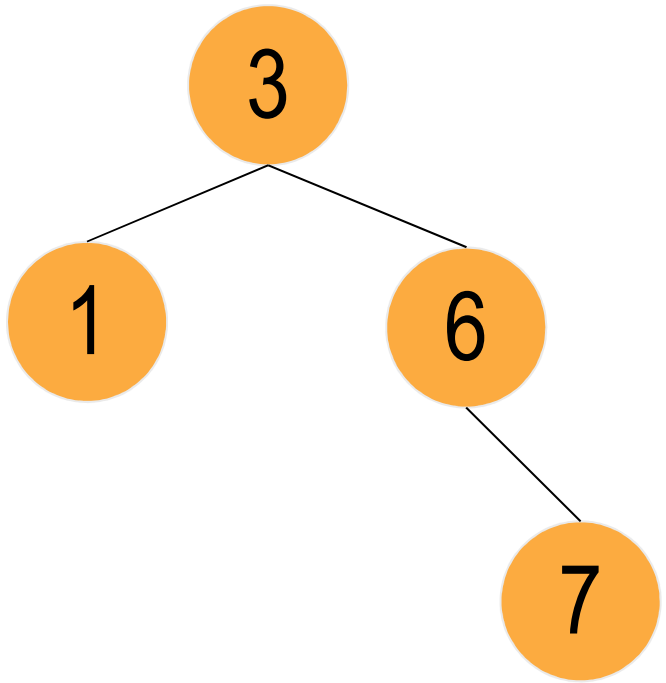
Remoção – Caso 2

- Removendo-se o nó com o valor 5
- Como ele possui uma sub-árvore direita, o nó contendo o valor 6 pode “ocupar” o lugar do nó removido



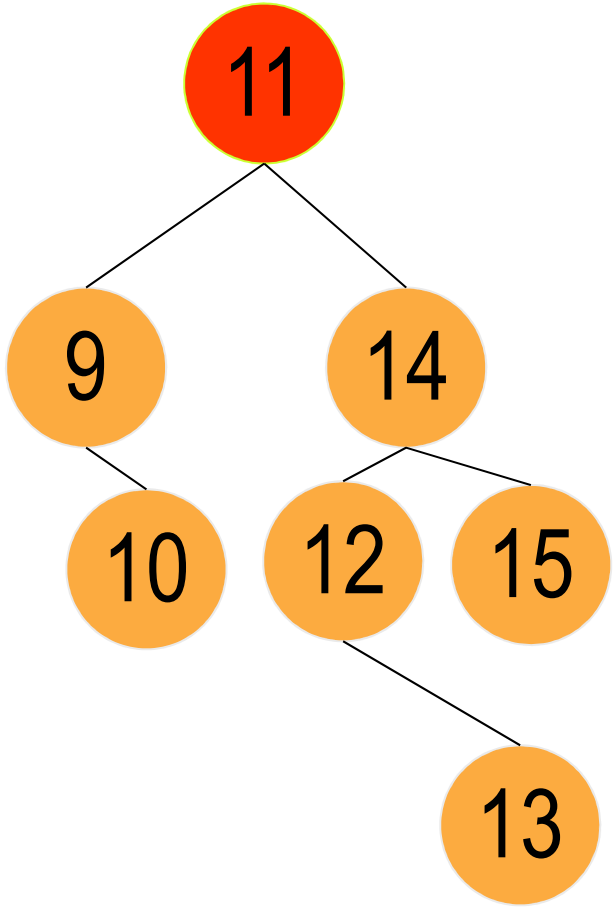
Remoção – Caso 2

- Esse segundo caso é análogo, caso existisse um nó com somente uma sub-árvore esquerda



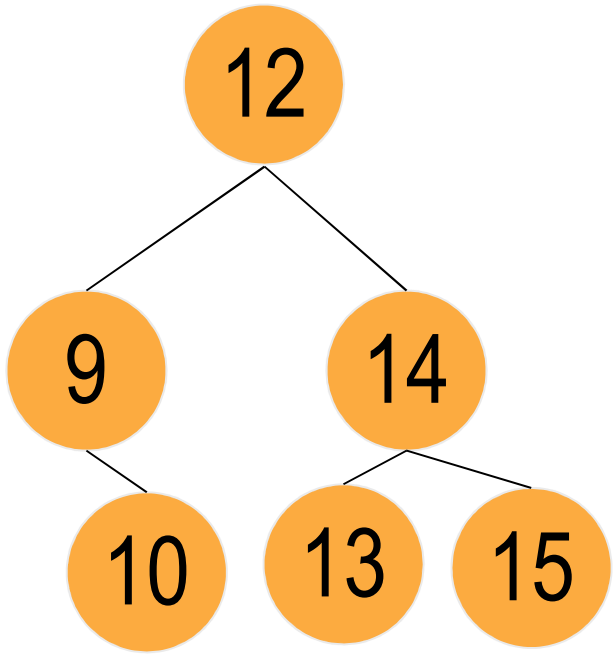
Remoção – Caso 3

- Eliminando-se o nó de chave 11
- Neste caso, existem 2 opções:
  - O nó com chave 10 pode "ocupar" o lugar do nó-raiz, ou
  - O nó com chave 12 pode "ocupar" o lugar do nó-raiz



Remoção – Caso 3

- Esse terceiro caso, também se aplica ao nó com chave 14, caso seja retirado.
  - Nessa configuração, o nó com chave 15 poderia “ocupar” o lugar.



Exercício

- Desenvolver os procedimentos:

caso1(raiz,valor)

caso2(raiz,valor, “esq”)

caso2(raiz,valor, “dir”)

case3(raiz,valor)

```
pno* remove (pno* r, int v)  {
pno* t, pai, f;
if (r == NULL) return NULL;
else if (r->info > v)
        r->esq = remove(r->esq, v);
    else if (r->info < v)
        r->dir = remove(r->dir, v);
else {                                  /* achou o elemento */
        if (r->esq == NULL && r->dir == NULL) {
    /* elemento sem filhos */
                free (r); r = NULL;}
else if (r->esq == NULL) {          /* só tem filho à direita */
        t = r;
        r = r->dir;  free (t);}
else if (r->dir == NULL) {          /* só tem filho à esquerda */
        t = r;
        r = r->esq;  free (t); }
else { /* tem os dois filhos */
        ...
```

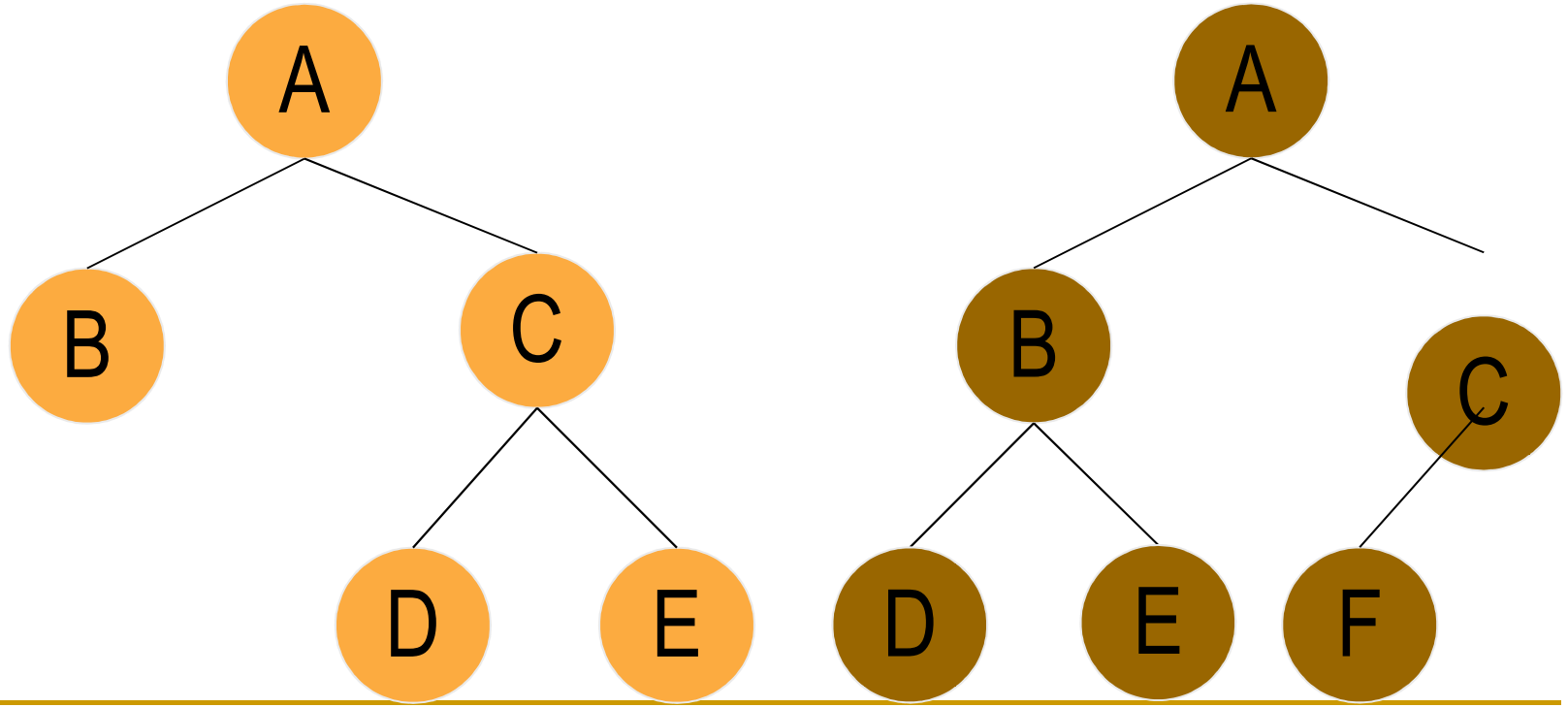
```
pno* remove (pno* r, int v)  { /* continuação */
....
else {              /* tem os dois filhos */
        pai = r;
        f = r->esq;  /* substitui pelo antecessor */
        while (f->dir != NULL) {
                pai = f;
                f = f->dir; }
        r->info = f->info; /* troca as informações */
        f->info = v;
        if (pai == r) {r->esq= f->esq};
        else          {pai->dir = f->esq};
        free(f); }
    }
    return (r);
}
```

Custo da busca em ABB

- Pior caso: número de passos é determinado pela altura da árvore
- Altura da ABB depende da seqüência de inserção das chaves...
  - Considere, p.ex., o que acontece se uma seqüência ordenada de chaves é inserida...
  - Seria possível gerar uma árvore balanceada com essa mesma seqüência, se ela fosse conhecida a priori. Como?
- Busca eficiente se árvore razoavelmente balanceada...

Árvore Binária Balanceada

- Para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1



Árvore Binária Perfeitamente Balanceada

- O número de nós de suas sub-árvores esquerda e direita difere em, no máximo, 1
- É a árvore de altura mínima para o conjunto de chaves
- Toda AB Perfeitamente Balanceada é Balanceada, sendo que o inverso não é necessariamente verdade
- Os algoritmos de inserção e remoção vistos não garantem que a árvore resultante de uma inserção/remoção seja perfeitamente balanceada ou mesmo apenas balanceada

Agradecimentos pelo material didático

Walter Aoiama Nagai

http://www.walternagai.hpg.com.br

Revisão: M. Cristina

Revisado: Roseli A F Romero, nov/2011.