AED2 - Aula 05
Árvores binárias de busca balanceadas e rotações

Árvores binárias de busca

Finalizando as operações suportadas por essas árvores:

- inserção - com eficiência O(altura)
    - comece na raiz
    - repita o seguinte processo até chegar num apontador vazio
        - se k <= chave do nó atual desça para o filho esquerdo
        - se k > chave do nó atual desça para o filho direito
    - substitua o apontador vazio pelo novo objeto, atribua seu apontador pai para o objeto que o precedeu no caminho da busca e atribua NULL aos apontadores dos filhos.

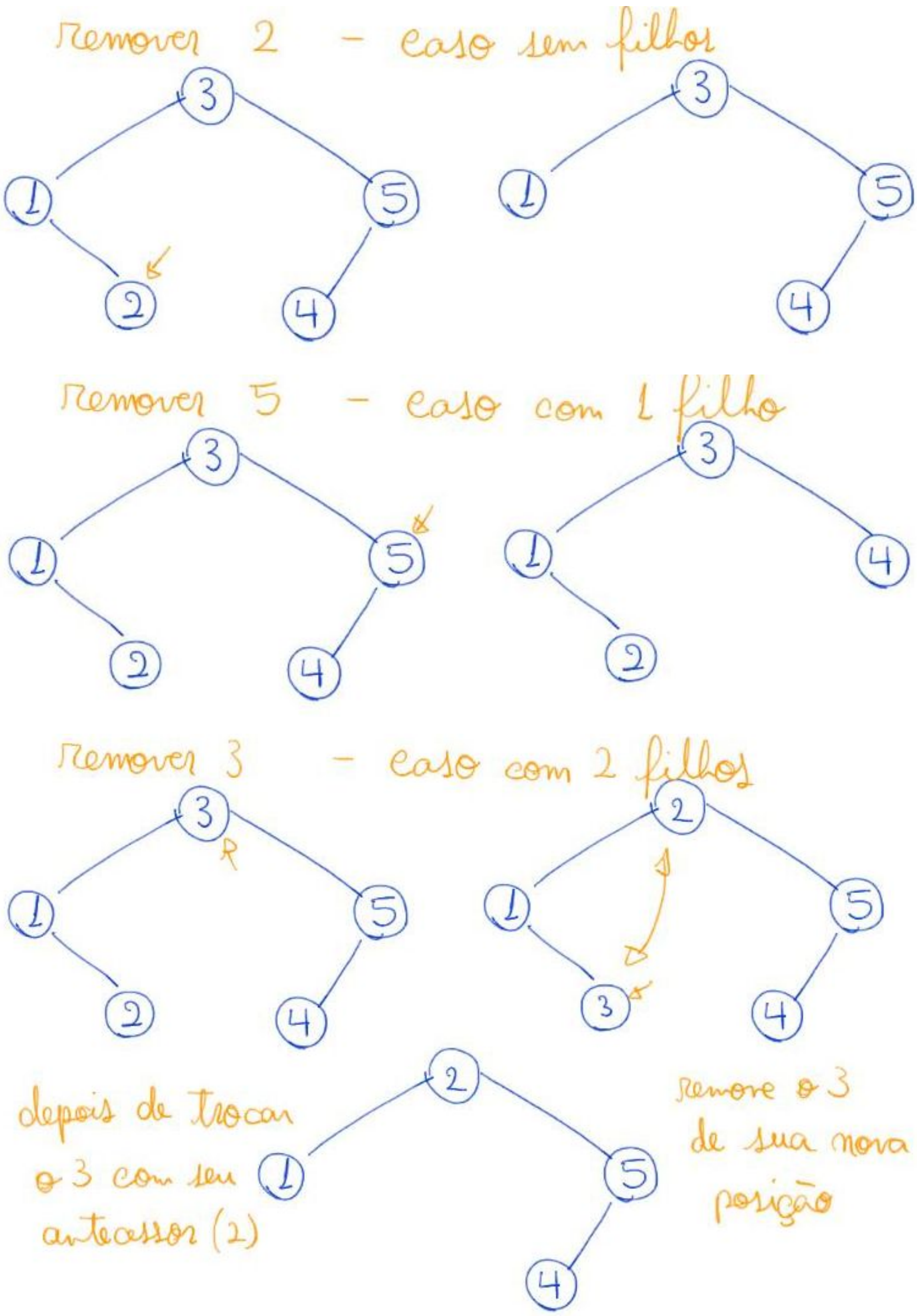
```
Noh *novoNoh(Chave chave, Item conteudo)
{
   Noh *novo;
   novo = (Noh *)malloc(sizeof(Noh));
   novo->chave = chave;
   novo->conteudo = conteudo;
   novo->esq = NULL;
   novo->dir = NULL;
   //    novo->pai = ??
   return novo;
}

Arvore insereI(Arvore r, Noh *novo)
{
   Noh *corr, *ant = NULL;
   if (r == NULL)
   {
       novo->pai = NULL;
       return novo;
   }
   corr = r;
   while (corr != NULL)
   {
       ant = corr;
       if (novo->chave <= corr->chave)
           corr = corr->esq;
       else
           corr = corr->dir;
   }
   novo->pai = ant;
   if (novo->chave <= ant->chave)
       ant->esq = novo;
```

```c
    else
        ant->dir = novo;
    return r;
}

Arvore insereR(Arvore r, Noh *novo)
{
    if (r == NULL)
    {
        novo->pai = NULL;
        return novo;
    }
    if (novo->chave <= r->chave)
    {
        r->esq = insereR(r->esq, novo);
        r->esq->pai = r;
    }
    else
    {
        r->dir = insereR(r->dir, novo);
        r->dir->pai = r;
    }
    return r;
}

Arvore inserir(Arvore r, Chave chave, Item conteudo)
{
    Noh *novo = novoNoh(chave, conteudo);
    return insereI(r, novo);
}
```

- como modificar inserção para que ela atualize correta e eficientemente o número de objetos (tam) de cada subárvore?

- remoção
    - use a busca para localizar um objeto x a ser removido.
        - se tal objeto não existe não há o que fazer.
    - se x não possui filhos basta removê-lo e fazer o apontador de seu pai para ele igual a NULL.
        - se x fosse a raiz, a nova árvore é vazia.
    - se x possui um filho conecte diretamente o pai de x com o filho de x, atualizando seus apontadores.
        - se x fosse a raiz, seu filho se torna a nova raiz.
    - se x possui dois filhos troque x pelo objeto y que antecede x, ou seja,
        - pelo maior elemento da subárvore esquerda de x.
        - note que temporariamente a propriedade de busca é violada por x em sua nova posição.
        - então remova x de sua nova posição

        - note que essa remoção cairá num dos casos mais simples, já que na nova posição x não tem filho direito
            - caso contrário y não seria o maior elemento da subárvore esquerda.
    - a seguir exemplos dos vários casos da remoção:




```
Arvore removeRaiz(Arvore alvo)
{
   Noh *aux, *p;
   if (alvo->esq == NULL && alvo->dir == NULL)
   {
```

```
        free(alvo);
        return NULL;
    }
    if (alvo->esq == NULL || alvo->dir == NULL)
    {
        if (alvo->esq == NULL)
            aux = alvo->dir;
        if (alvo->dir == NULL)
            aux = alvo->esq;
        aux->pai = alvo->pai;
        free(alvo);
        return aux;
    }
    aux = max(alvo->esq);
    alvo->chave = aux->chave;
    alvo->conteudo = aux->conteudo;
    p = aux->pai;
    if (p == alvo)
        p->esq = removeRaiz(aux);
    else // aux->pai != alvo
        p->dir = removeRaiz(aux);
    return alvo;
}

Arvore removeI(Arvore r, Chave chave)
{
    Noh *alvo, *p, *aux;
    alvo = buscaI(r, chave);
    if (alvo == NULL)
        return r;
    p = alvo->pai;
    aux = removeRaiz(alvo);
    if (p == NULL)
        return aux;
    if (p->esq == alvo)
        p->esq = aux;
    if (p->dir == alvo)
        p->dir = aux;
    return r;
}
```

- como modificar remoção para que ela atualize correta e eficientemente o número de objetos (tam) de cada subárvore?

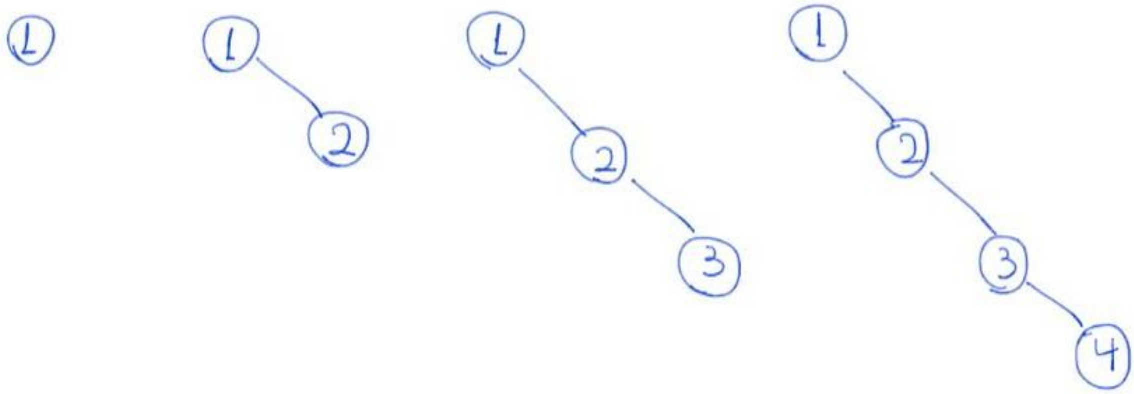
Árvores binárias de busca balanceadas

Balanceamento é crítico para eficiência,

- pois quase toda operação leva tempo proporcional à altura da árvore.

- a exceção é o percurso ordenado.
- e a altura pode variar de lg n até n-1.

Note que é fácil uma árvore ficar muito desbalanceada
- basta inserir os elementos em ordem, por exemplo.

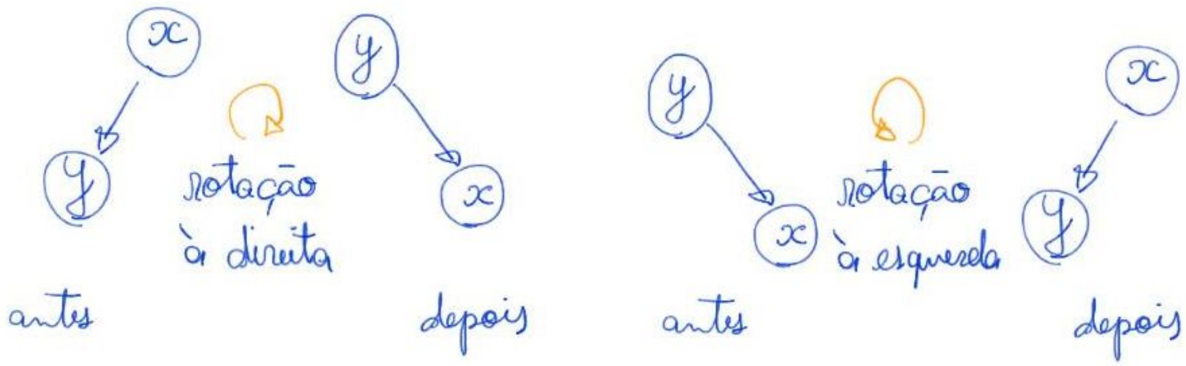


Existem diversas estratégias para resolver o problema do balanceamento
- e estas dão origem a diferentes árvores.
- Ex: árvores AVL, árvores rubro-negras, splay-trees, árvores B, árvores 2-3.

Várias bibliotecas possuem implementações de árvores balanceadas. Como exemplos temos:
- a classe map em C++,
- a classe TreeMap e java.

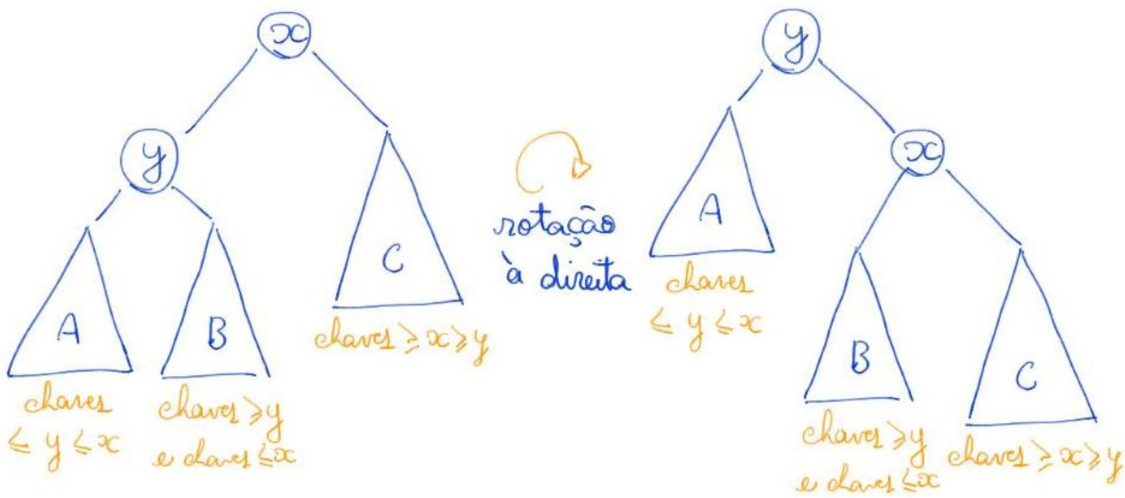
Vamos estudar a estratégia de rotações
- e veremos duas árvores balanceadas baseadas nessa estratégia:
    - árvores AVL,
    - árvores rubro-negras.

Rotações:
- uma rotação pega um par pai-filho e inverte sua relação.
    - temos rotações à esquerda e à direita.

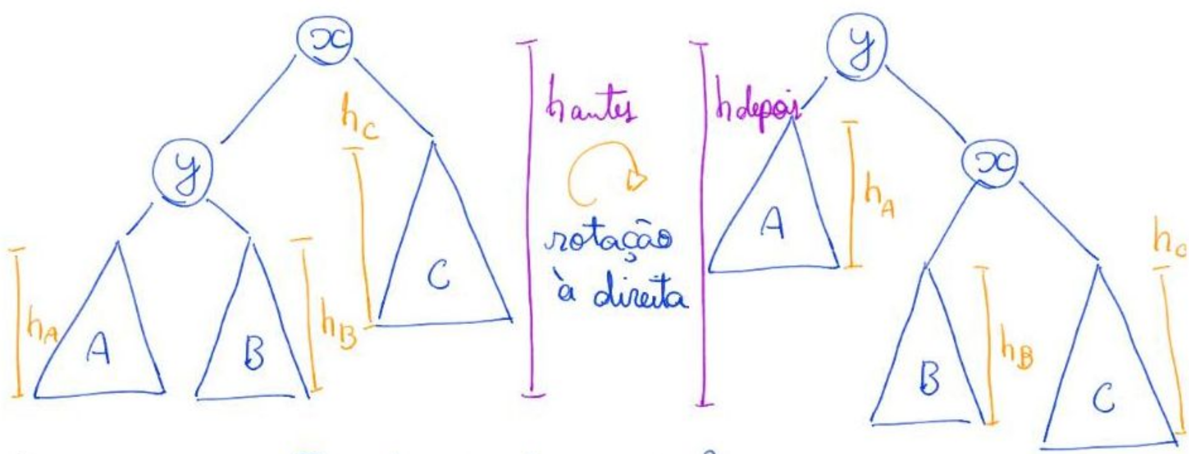


```
Arvore rotacaoDir(Arvore r)
{
    Noh *aux;
    aux = r->esq;
    r->esq = aux->dir;
    if (aux->dir != NULL)
        aux->dir->pai = r;
    aux->dir = r;
    r->pai = aux;
    return aux;
}
```

- vamos analisar como uma rotação pode ser realizada
  - utilizando um número pequeno de operações
  - e preservando a propriedade de busca.



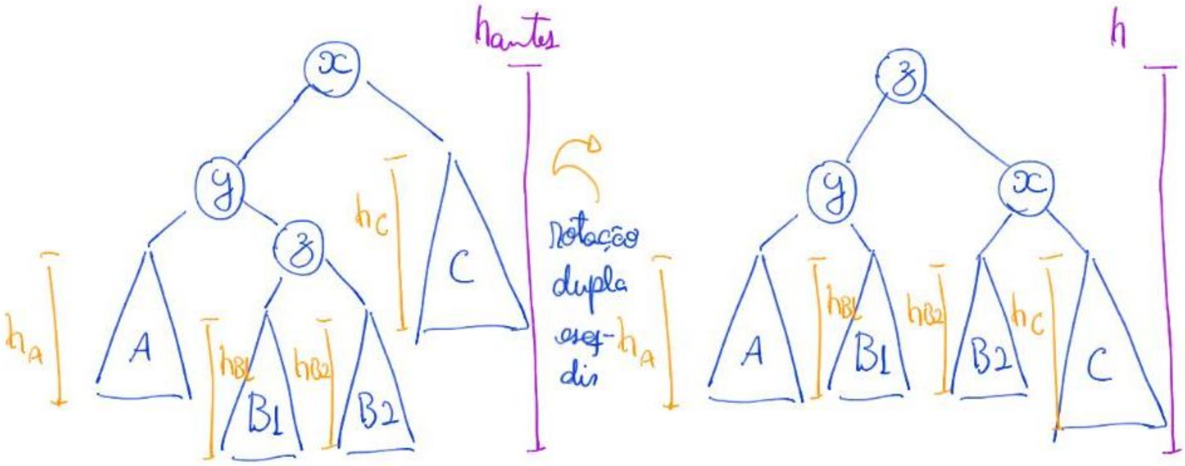
- então vamos analisar o impacto de uma rotação na altura das subárvores envolvidas.



$$h_{antes} = \max\{2 + h_A,\ 2 + h_B,\ 1 + h_C\}$$

$$h_{depois} = \max\{1 + h_A,\ 2 + h_B,\ 2 + h_C\}$$

- note que a rotação parece interessante se h_a > h_c,
    - pois diminui o impacto de h_a na altura final,
    - mas aumenta o impacto de h_c.
- observe que o impacto de h_b na altura não é alterado pela rotação.
    - para tanto precisaremos fazer uma rotação dupla



$$h_{antes} = \max\{2+h_A, 3+h_{B1}, 3+h_{B2}, 1+h_C\}$$

$$h_{depois} = \max\{2+h_A, 2+h_{B1}, 2+h_{B2}, 2+h_C\}$$

- observe que a rotação dupla corresponde a:
    - uma rotação simples que inverte a relação entre y e z,
    - outra rotação simples entre x e z.
- verifique que a propriedade de busca é preservada
- e que o impacto e h_b1 e h_b2 na altura final é reduzido
- enquanto o impacto de h_c aumenta.