MC202 - Estrutura de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Árvore Binária de Busca de Altura Balanceada

- A busca por uma chave em uma árvore binária de busca com n nós tem complexidade $O(n)$, no pior caso, e no melhor caso, quando a árvore está perfeitamente balanceada (cheia/semi-cheia), tem complexidade $O(\log n)$.

Árvore Binária de Busca de Altura Balanceada

- A busca por uma chave em uma árvore binária de busca com n nós tem complexidade $O(n)$, no pior caso, e no melhor caso, quando a árvore está perfeitamente balanceada (cheia/semi-cheia), tem complexidade $O(\log n)$.

- Para manter uma árvore binária de busca perfeitamente balanceada, bastaria remover seus elementos com um percurso em ordem simétrica, inserindo-os em um vetor auxiliar (os elementos ficariam em ordem crescente). Depois reconstruir a árvore reinserindo o elemento central do vetor, e de forma recursiva reinserindo os elementos centrais de cada metade.

Árvore Binária de Busca de Altura Balanceada

- A busca por uma chave em uma árvore binária de busca com n nós tem complexidade $O(n)$, no pior caso, e no melhor caso, quando a árvore está perfeitamente balanceada (cheia/semi-cheia), tem complexidade $O(\log n)$.

- Para manter uma árvore binária de busca perfeitamente balanceada, bastaria remover seus elementos com um percurso em ordem simétrica, inserindo-os em um vetor auxiliar (os elementos ficariam em ordem crescente). Depois reconstruir a árvore reinserindo o elemento central do vetor, e de forma recursiva reinserindo os elementos centrais de cada metade.

- A complexidade desta operação, porém, é $O(n)$, então como manter (inserção e remoção) uma árvore binária de busca em $O(\log n)$?

- Dois exemplos de resposta afirmativa são, árvores rubro-negra (Robert Sedgewicke, 2008) e AVL (Georgy Adelson-Velsky e Evgenii Landis, em 1962).

Árvore Binária de Busca de Altura Balanceada

- Dois exemplos de resposta afirmativa são, árvores rubro-negra (Robert Sedgewicke, 2008) e AVL (Georgy Adelson-Velsky e Evgenii Landis, em 1962).
- Ambas exploram o conceito de árvore binária balanceada, em vez de perfeitamente balanceada — i.e., a diferença entre as alturas das subárvores esquerda e direita é no máximo 1 para qualquer nó.

Árvore Binária de Busca de Altura Balanceada

- Dois exemplos de resposta afirmativa são, árvores rubro-negra (Robert Sedgewicke, 2008) e AVL (Georgy Adelson-Velsky e Evgenii Landis, em 1962).
- Ambas exploram o conceito de árvore binária balanceada, em vez de perfeitamente balanceada — i.e., a diferença entre as alturas das subárvores esquerda e direita é no máximo 1 para qualquer nó.
- Nas próximas duas aulas vamos ver o caso de Árvore AVL, cuja altura não ultrapassa 45% da altura de uma árvore perfeitamente balanceada.

Agenda

Árvore AVL:

- Estratégia de balanceamento.
- Inserção de um nó.
- Remoção de um nó.

Árvore AVL

- A estratégia da árvore AVL é manter um fator de balanceamento para cada nó, que indica quando suas subárvores possuem mesma altura (0), a subárvore esquerda é mais alta (-1), e a subárvore direita é mais alta (+1).

Árvore AVL

- A estratégia da árvore AVL é manter um fator de balanceamento para cada nó, que indica quando suas subárvores possuem mesma altura (0), a subárvore esquerda é mais alta (-1), e a subárvore direita é mais alta (+1).

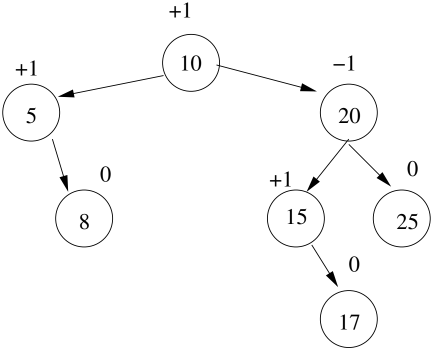
- A diferença em altura entre as subárvores esquerda e direita não pode ser em módulo maior que 1. Este evento é corrigido com rotações à esquerda e à direita, simples e dupla, dependendo do caso.

Árvore AVL

- A estratégia da árvore AVL é manter um fator de balanceamento para cada nó, que indica quando suas subárvores possuem mesma altura (0), a subárvore esquerda é mais alta (-1), e a subárvore direita é mais alta (+1).
- A diferença em altura entre as subárvores esquerda e direita não pode ser em módulo maior que 1. Este evento é corrigido com rotações à esquerda e à direita, simples e dupla, dependendo do caso.
- O balanceamento dos nós é ajustado no caminho de volta da inserção (remoção), parando quando o ajuste de um nó não altera a altura da árvore enraizada nele.

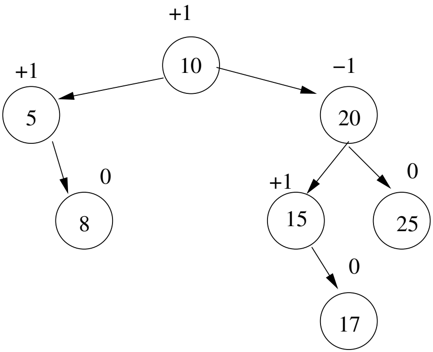
Árvore AVL

A árvore abaixo está balanceada, mas a inserção de um nó de valor 16, por exemplo, ou a remoção do nó de valor 8, fará com que perca o balanceamento.



Árvore AVL

A árvore abaixo está balanceada, mas a inserção de um nó de valor 16, por exemplo, ou a remoção do nó de valor 8, fará com que perca o balanceamento.



Como aplicar as rotações e corrigir o balanceamento?

Inserção em Árvore AVL

- A ideia é inserir nós à esquerda ou à direira de um nó folha, como fizemos para árvore binária de busca, indicando com uma *flag* $h = 1$ que a altura da subárvore da folha aumentou.

Inserção em Árvore AVL

- A ideia é inserir nós à esquerda ou à direira de um nó folha, como fizemos para árvore binária de busca, indicando com uma *flag* $h = 1$ que a altura da subárvore da folha aumentou.

- E enquanto $h = 1$, tratar aumento de altura na árvore correspondente na volta da recursão (balanceamento da raiz).

Inserção em Árvore AVL

```
void InsereValor(AVL **ab, int valor, char *h)
{
  if ((*ab) == NULL) {
    (*ab) = CriaNovoNo(valor); *h = 1;
  } else {
    if ((*ab)→info ≤ valor){
      InsereValor(&((*ab)→dir),valor,h);
      if ((*h)==1) TrataAumentoArvoreDireita(ab,h);
    }else{
      InsereValor(&((*ab)→esq),valor,h);
      if ((*h)==1) TrataAumentoArvoreEsquerda(ab,h);
    }
  }
}
```

Inserção em Árvore AVL

Vamos ver como tratar o aumento na árvore esquerda, deixando o aumento na árvore direita (simétrico) como exercício. A inserção de um nó na árvore esquerda pode ser dividida em quatro casos.

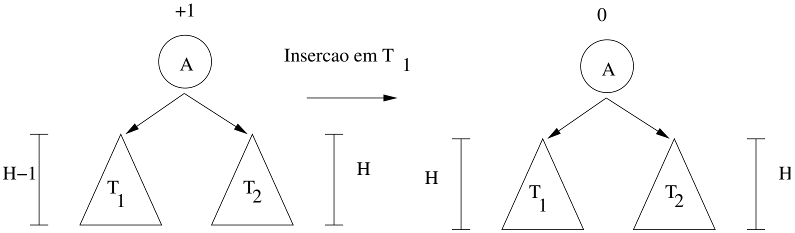
Inserção em Árvore AVL

Vamos ver como tratar o aumento na árvore esquerda, deixando o aumento na árvore direita (simétrico) como exercício. A inserção de um nó na árvore esquerda pode ser dividida em quatro casos.

- Caso 1: A raiz A estava com fator de balanceamento $+1$. Seu balanceamento fica 0, retornando $h = 0$ (a altura da árvore de A não aumentou).

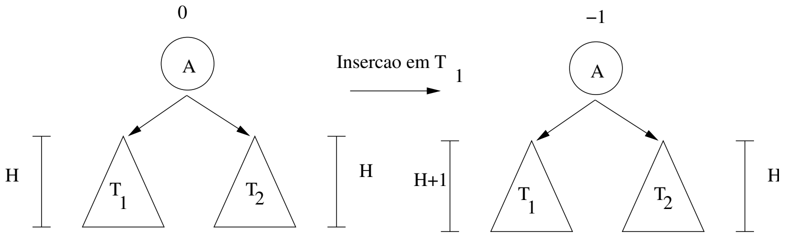
Vamos ver como tratar o aumento na árvore esquerda, deixando o aumento na árvore direita (simétrico) como exercício. A inserção de um nó na árvore esquerda pode ser dividida em quatro casos.

- Caso 1: A raiz A estava com fator de balanceamento $+1$. Seu balanceamento fica 0, retornando $h = 0$ (a altura da árvore de A não aumentou).

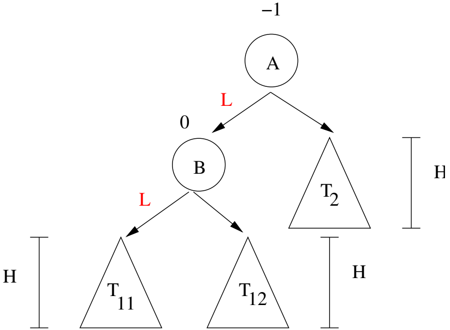


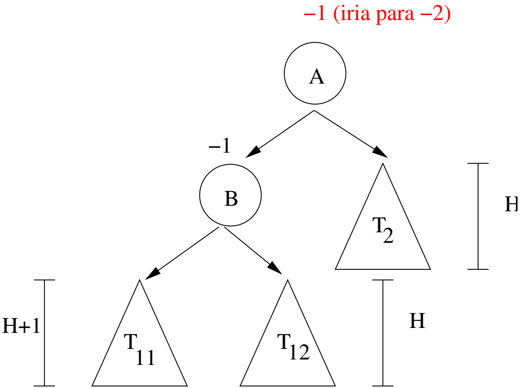
Inserção em Árvore AVL

- Caso 2: A raiz A estava com fator de balanceamento 0. Seu balanceamento fica -1 e h continua 1, pois a altura da árvore de A aumentou.

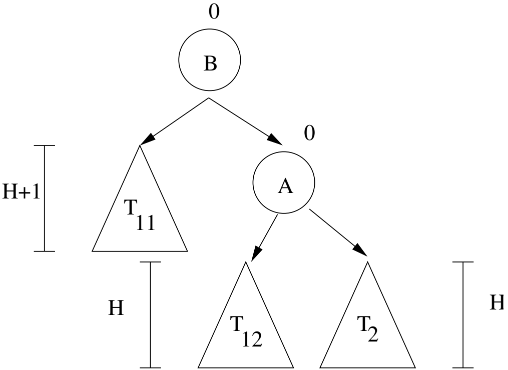


- Caso 3: Inserção na árvore esquerda de B, o filho à esquerda de A, mudando seu balanceamento de 0 para -1, quando A já estava com balanceamento -1.



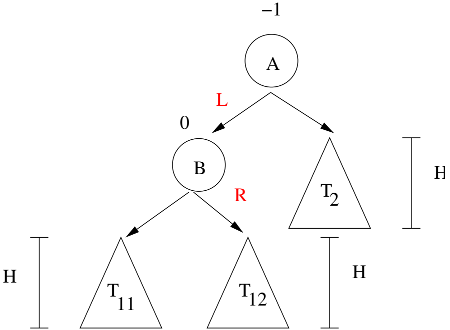


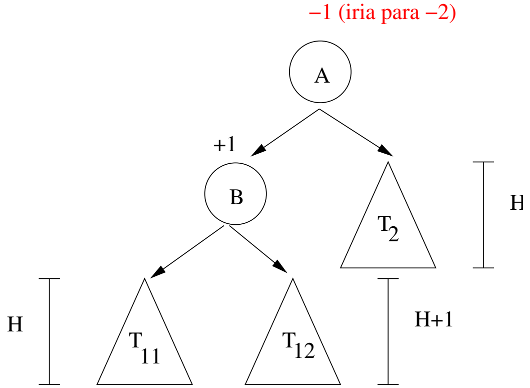
A Rotação Simples à Direita (horário), também conhecida por LL, ajusta o balanceamento de A e B para 0, retornando $h = 0$.



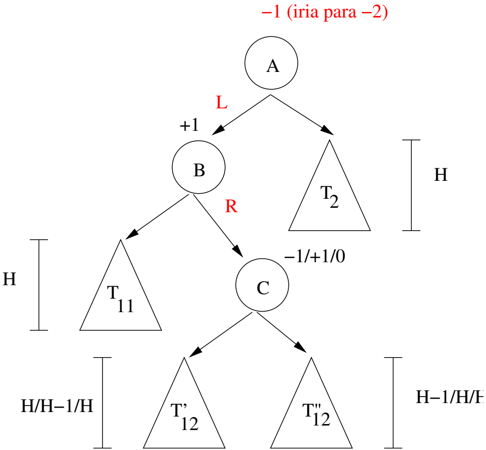
A Rotação Simples à Direita (horário), também conhecida por LL, ajusta o balanceamento de A e B para 0, retornando $h = 0$.

- Caso 4: Inserção na árvore direita do filho B à esquerda da raiz A, quando A já estava com fator de balanceamento -1.

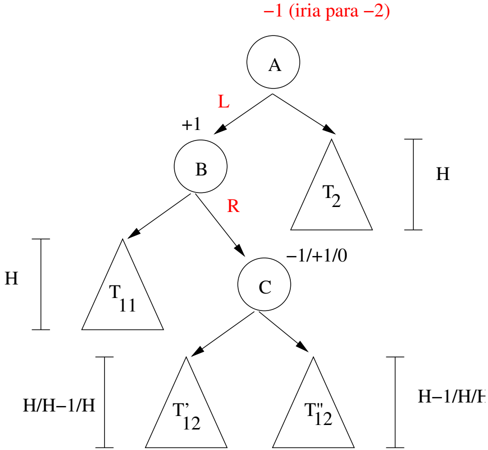




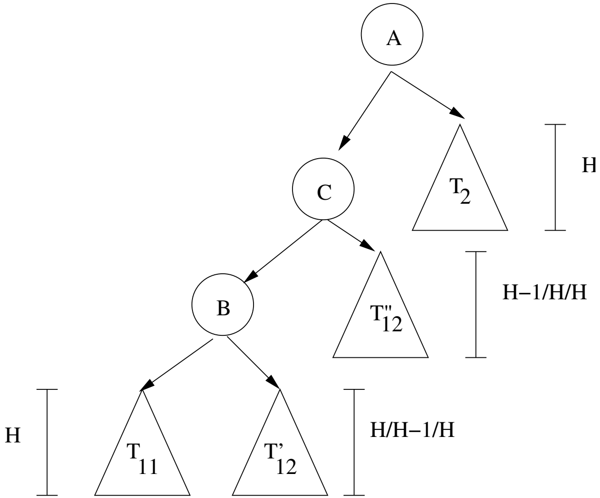
A Rotação Dupla à Direita, também conhecida por LR, equivale à rotação simples à esquerda (anti-horário) em B seguida de rotação simples à direita (horário) em A. O balanceamento de A e B vai para 0 e h retorna 0.



A inserção pode ter sido em T'_{12} ou $T"_{12}$, mudando o balanceamento de C, filho à direita de B, para -1, $+1$, ou 0.

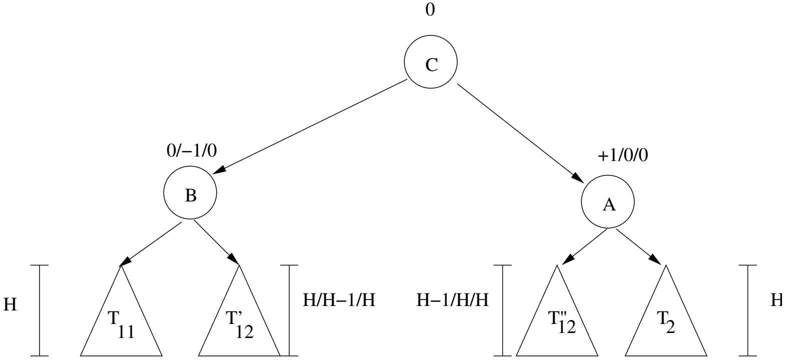


Note que C já pode ter sido resultado de rotação com subárvores de alturas $H-1$ e $H-2$, gerando T'_{12} e $T"_{12}$ com altura H.



Após rotação simples à esquerda na árvore de B.

Inserção em Árvore AVL



Após rotação simples à direita na árvore de A. Portanto, o balanceamento final de A e B vai depender de como estava o balanceamento de C após a inserção em sua árvore.

Remoção em Árvore AVL

- A ideia é aplicar remoção como fizemos para árvore binária de busca, para nós com único/nenhum filho e nós internos, indicando por $h = 1$ que a altura da subárvore correspondente reduziu.

Remoção em Árvore AVL

- A ideia é aplicar remoção como fizemos para árvore binária de busca, para nós com único/nenhum filho e nós internos, indicando por $h = 1$ que a altura da subárvore correspondente reduziu.

- E enquanto $h = 1$, tratar redução de altura na árvore da raiz correspondente na volta da recursão.

Remoção em Árvore AVL

- A ideia é aplicar remoção como fizemos para árvore binária de busca, para nós com único/nenhum filho e nós internos, indicando por $h = 1$ que a altura da subárvore correspondente reduziu.

- E enquanto $h = 1$, tratar redução de altura na árvore da raiz correspondente na volta da recursão.

- Note, porém, que a remoção de um nó interno X gera tratamento adicional da árvore que contém seu sucessor como filho à esquerda, seguido de tratamento da árvore direita de X, antes de retomar ao percurso de volta da recursão.

Remoção em Árvore AVL

```
void RemoveValor(AVL **ab, int valor, char *h)
{
  if ((*ab) != NULL){
    if ((*ab)→info != valor) {
      if ((*ab)→info < valor) {
        RemoveValor(&((*ab)→dir),valor,h);
        if ((*h)==1) TrataReducaoArvoreDireita(ab,h);
      } else {
        RemoveValor(&((*ab)→esq),valor,h);
        if ((*h)==1) TrataReducaoArvoreEsquerda(ab,h);
      }
    } else {
      RemoveDeFato(ab,h);
    }
  }
}
```

Remoção em Árvore AVL

Vamos ver como tratar a redução na árvore esquerda, deixando a redução na árvore direita (simétrico) como exercício. A remoção de um nó na árvore esquerda pode ser dividida em quatro casos.

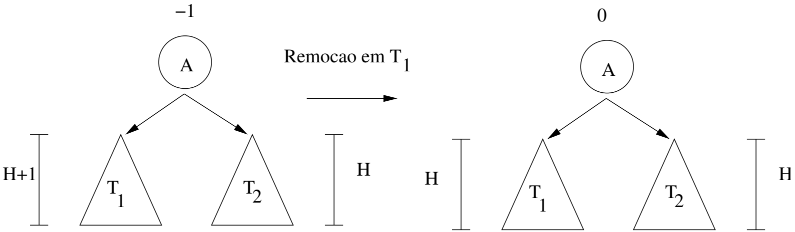
Remoção em Árvore AVL

Vamos ver como tratar a redução na árvore esquerda, deixando a redução na árvore direita (simétrico) como exercício. A remoção de um nó na árvore esquerda pode ser dividida em quatro casos.

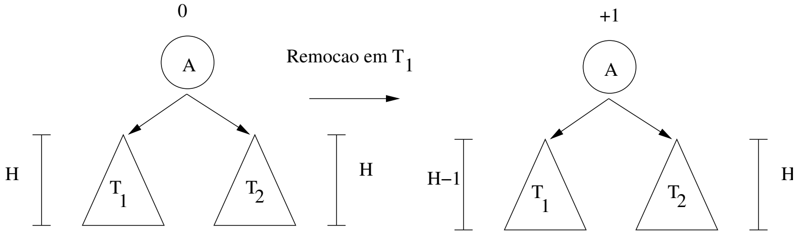
- Caso 1: A raiz A estava com fator de balanceamento -1. O balanceamento de A fica 0 e h retorna 1, pois a redução de altura de A pode afetar o balanceamento de seus ancestrais.

Vamos ver como tratar a redução na árvore esquerda, deixando a redução na árvore direita (simétrico) como exercício. A remoção de um nó na árvore esquerda pode ser dividida em quatro casos.

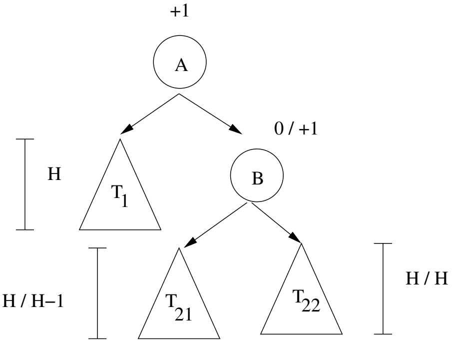
- Caso 1: A raiz A estava com fator de balanceamento -1. O balanceamento de A fica 0 e h retorna 1, pois a redução de altura de A pode afetar o balanceamento de seus ancestrais.



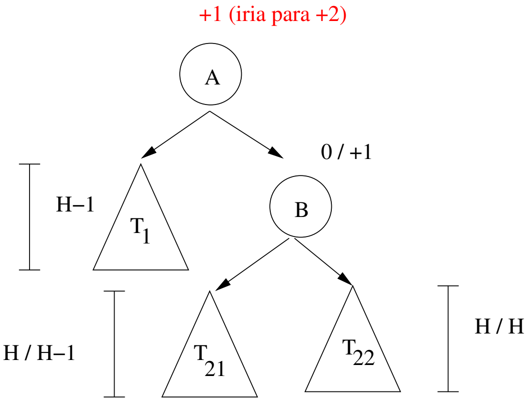
- Caso 2: A raiz A estava com fator de balanceamento 0. O balanceamento de A fica $+1$ e h retorna 0, pois a altura de A não mudou.



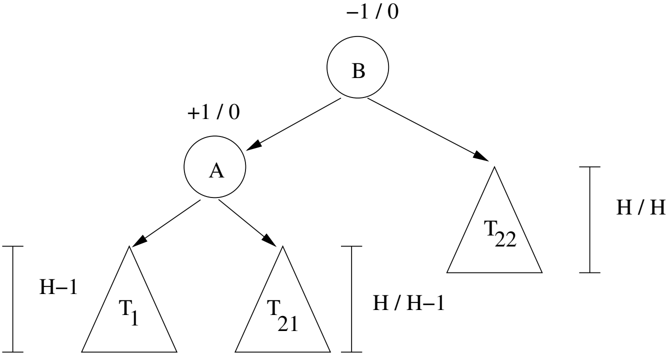
- Caso 3: A raiz A, de altura $H+2$, estava com fator de balanceamento $+1$ e seu filho B à direita com fator 0 ou $+1$.



Remoção em Árvore AVL

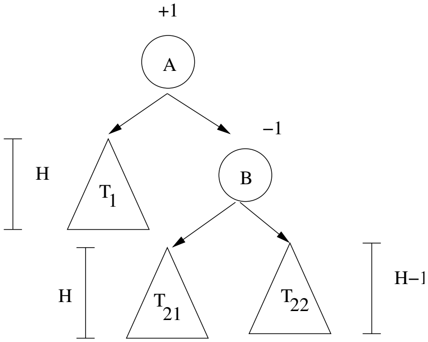


Após remoção em T_1, a Rotação Simples à Esquerda deve ajustar o balanceamento de A para $+1/0$, B para $-1/0$, e h pode retornar $0/1$, dependendo de como estava o balanceamento de B.

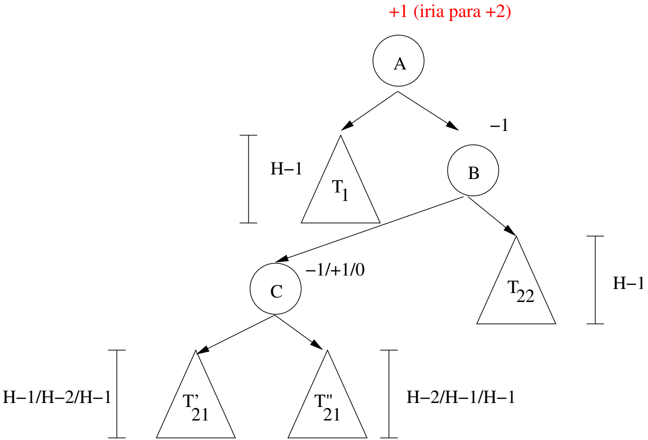


A árvore pode ficar com a mesma altura, $H+2$ (h retorna 0), ou com altura $H+1$ reduzida (h retorna 1).

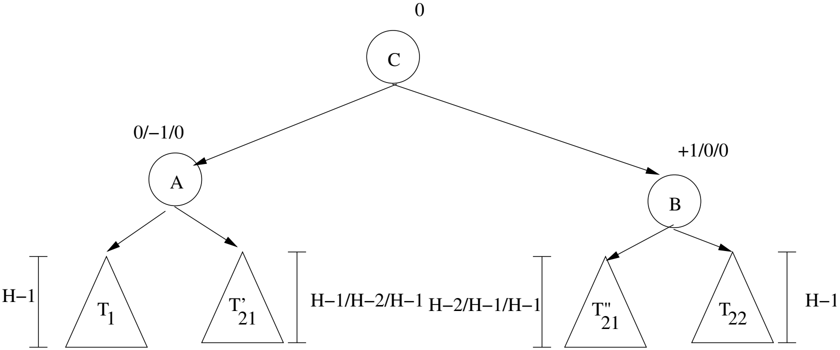
- Caso 4: A raiz A estava com fator de balanceamento $+1$ e seu filho B à direita com fator -1.



Remoção em Árvore AVL



Após remoção em T_1, a Rotação Dupla à Esquerda deve ajustar o balanceamento de A para $0/-1/0$, B para $1/0/0$, dependendo de como estava o balanceamento de C, e h retorna 1.



A rotação dupla à esquerda equivale à uma rotação simples à direita em B, seguida de outra simples à esquerda em A.

Remoção em Árvore AVL

É comum (e bastante confuso), portanto, chamar as rotações simples à direita de LL, simples à esquerda de RR, dupla à esquerda de RL, e dupla à direita de LR, onde L significa *left* (esquerda) e R significa *right* (direita).

Remoção em Árvore AVL

No caso de remoção de nó de grau 2, o balanceamento deve ser tratado na volta do percurso de busca pelo sucessor e depois na raiz da árvore direita.

No caso de remoção de nó de grau 2, o balanceamento deve ser tratado na volta do percurso de busca pelo sucessor e depois na raiz da árvore direita.

```
void RemoveDeFato(AVL **ab, char *h)
{
  if (RemoveNoGrau0ou1(ab,h)==0){
    int bal = (*ab)→bal; // salva
    SubstituiRemoveMenorSucessor(ab,&((*ab)→dir),h);
    (*ab)→bal = bal; // recupera
    if (*h)
      TrataReducaoArvoreDireita(ab,h);
  }
}
```

Remoção em Árvore AVL

```
void SubstituiRemoveMenorSucessor(AVL **ab, AVL **maisesq, char *h)
{
  if ((*maisesq)→esq == NULL){
    (*ab)→info = (*maisesq)→info;
    RemoveNoGrau0ou1(maisesq,h);
  } else {
    SubstituiRemoveMenorSucessor(ab,&((*maisesq)→esq),h);
    if (*h)
      TrataReducaoArvoreEsquerda(maisesq,h);
  }
}
```