Estruturas de Dados II

Árvore AVL

Prof. Bruno Azevedo

Instituto Federal de São Paulo

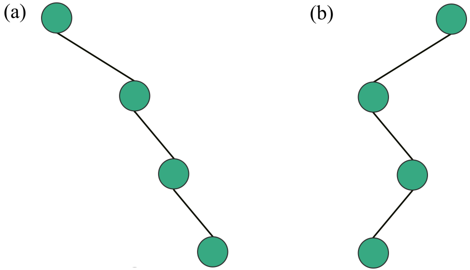
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Temas

- Cada tema só pode ser escolhido por uma única equipe.
- Temas para o Seminário:
  1. Árvores B.
  2. Árvores Fenwick.
  3. Estruturas de Dados em Blockchain.
  4. Skip Lists.
  5. Árvores Red-Black.
  6. Bucket Sort e Radix Sort.
- Quatro (4) equipes de cinco (5) alunos, uma equipe de seis (6) alunos.
- Apresentação entre 5 a 6 minutos POR ALUNO.
- Lembrando, o seminário ocorrerá dia 28/11. Serão 4 aulas com a turma unificada.

Árvore Binária de Busca

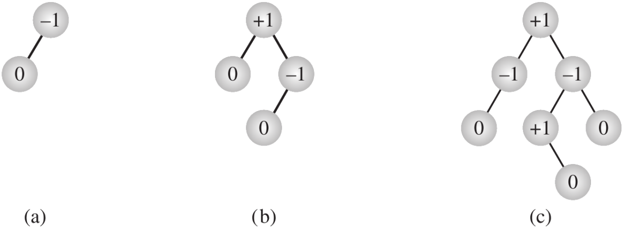
- Existe uma questão de **eficiência** que precisamos discutir relativa à Árvore Binária de Busca.
- Afinal, apesar de árvores binárias poderem possuir uma altura $O(\log_2 n)$, Árvores Binárias de Busca pode se tornar **degeneradas**.
- Na figura abaixo temos dois exemplos de Árvores Binárias de Busca degeneradas.



- Portanto, a busca, inserção e deleção em uma Árvore Binária de Busca pode possuir complexidade **linear**.
- Para solucionar essa questão, precisamos de uma árvore capaz de se auto-balancear.

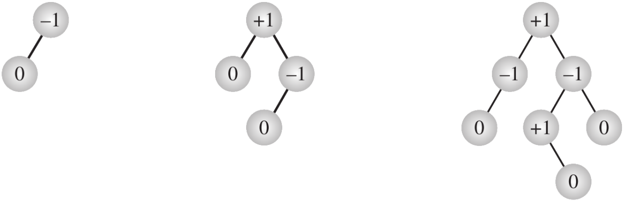
Árvores AVL

- A **Árvore AVL** é uma Árvore Binária de Busca que é capaz de se auto-balancear (nomeada a partir de seus criadores, Adelson-Velsky e Landis).
- **Uma Árvore AVL é uma árvore em que a altura das subárvores esquerda e direita de todo nó difere em, no máximo, um**.
- Todas as árvores na figura abaixo são árvores AVL.



Árvores AVL

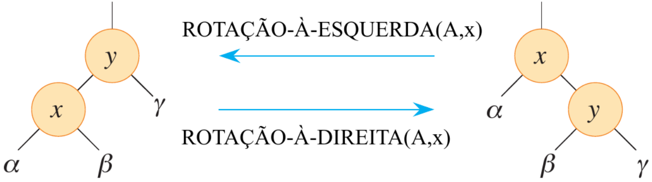
- Os números nos nós indicam os fatores de balanceamento, que são as diferenças entre as alturas das subárvores esquerda e direita.
- Ou seja, um fator de balanceamento é calculado **subtraindo a altura da subárvore esquerda da altura da subárvore direita**.
- Para uma árvore AVL, todos os fatores de balanceamento devem ser $+1$, 0 ou -1.



(a) (b) (c)

Árvores AVL

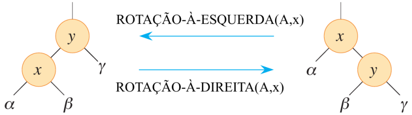
- Se o fator de balanceamento de qualquer nó em uma árvore AVL se tornar menor que -1 ou maior que 1, **a árvore precisa ser balanceada**.
- O balanceamento é feito através de **rotações**.
- Uma rotação é uma operação que preserva a propriedade da Árvore Binária de Busca.
- A figura abaixo mostra dois tipos de rotações: rotações à esquerda e rotações à direita.



Árvores AVL

- O pseudocódigo para rotação à esquerda é exibido à seguir. O código para a rotação à direita é simétrico ao da rotação à esquerda.

```
ROTACAO-A-ESQUERDA(A, x)
   se x->dir == NULL
      retorne; // Não é possível realizar a rotação se não há nó à direita
   y = x->dir
   x->dir = y->esq            // transforma a subárvore esquerda de y
                              // na subárvore direita de x
   if y->esq ≠ NULL       // se a subárvore esquerda de y não estiver vazia
      y->esq->p = x           // x se torna o pai da raiz da subárvore
   y->p = x->p                // o pai de x se torna o pai de y
   if x->p == NULL            // se x era a raiz
      A->raiz = y             // y se torna a raiz
   else if x == x->p->esq     // caso contrário, se x era um filho esquerdo
      x->p->esq = y           // y se torna o filho esquerdo
   else
      x->p->dir = y           // ou x era um filho direito e agora y é
   y->esq = x                 // x se torna o filho esquerdo de y
   x->p = y
```



Inserção em Árvores AVL

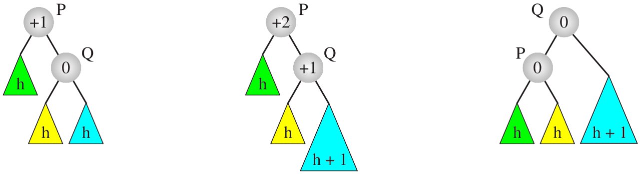
- A inserção de um novo nó em uma Árvore AVL é idêntica à inserção em uma Árvore Binária de Busca.
- Busca-se a posição correta para que o novo nó seja inserido como uma nova folha da árvore, obedecendo as propriedades de uma Árvore Binária de Busca.
- Entretanto, essa inserção pode causar o **desbalanceamento** da árvore.
- Caso isso ocorra, será necessário realizar rotações para rebalancear a Árvore AVL.

Inserção em Árvores AVL

- O código para a rotação à direita é simétrico ao da rotação à esquerda.
- Se o fator de balanceamento de qualquer nó em uma árvore AVL se tornar menor que -1 ou maior que 1, a árvore precisa ser balanceada.
- Uma árvore AVL pode ficar desbalanceada em quatro situações, mas apenas duas delas precisam ser analisadas; as outras duas são simétricas.
- Nas figuras a seguir, as alturas das subárvores participantes serão indicadas dentro das subárvores.
- Os dois casos que veremos referem-se a um desequilíbrio à direita. Os restantes são referentes a um desequilíbrio à esquerda.

Inserção em Árvores AVL

- O primeiro caso, resultante da **inserção de um nó na subárvore direita do filho direito**, é ilustrado na figura abaixo.
- Na árvore AVL da figura (a), um nó é inserido em algum lugar na subárvore direita de Q (figura (b)), o que perturba o balanceamento da árvore em P.

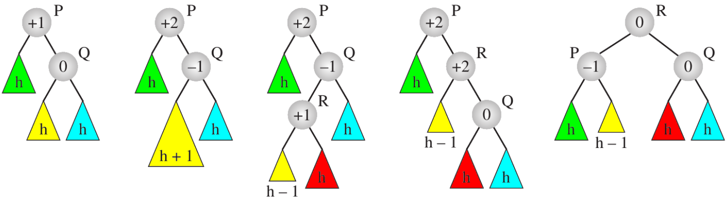


(a) (b) (c)

- Nesse caso, o problema pode ser corrigido ao rotacionar o nó Q em relação ao seu pai P (figura (c)), de forma que o fator de balanceamento tanto de P quanto de Q se torne zero, ou seja, até melhor do que no início.

Inserção em Árvores AVL

- O segundo caso, resultante da **inserção de um nó na subárvore esquerda do filho direito**, é um pouco mais complexo.
- Um nó é inserido na árvore da figura (a) abaixo. A árvore resultante é mostrada na figura (b) e com mais detalhes na figura (c).
- Para restaurar o equilíbrio desta árvore, uma dupla **rotação** é realizada.
- O balanceamento da árvore em P é restaurado ao rotacionar R em relação ao nó Q – figura (d) – e, em seguida, rotacionar R novamente, desta vez em relação ao nó P – figura (e).

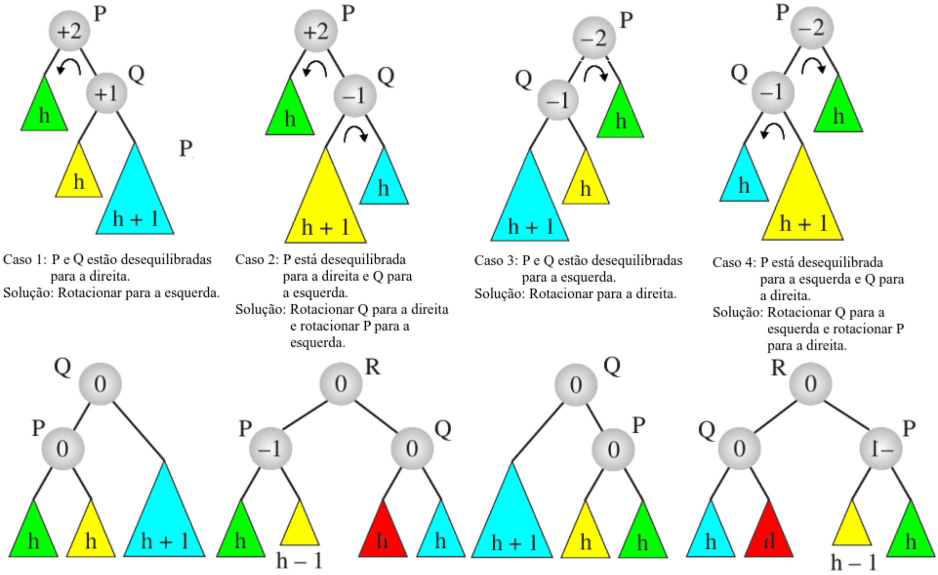


(a) (b) (c) (d) (e)

Inserção em Árvores AVL

- Os outros dois casos são simétricos.
- **Inserção de um nó na subárvore esquerda do filho esquerdo**.
- **Inserção de um nó na subárvore direita do filho esquerdo**.
- Ambos resultam em um desequilíbrio à esquerda.
- Ainda estão confusos? Vamos ver um resumo de todos os casos.

Inserção em Árvores AVL



Inserção em Árvores AVL

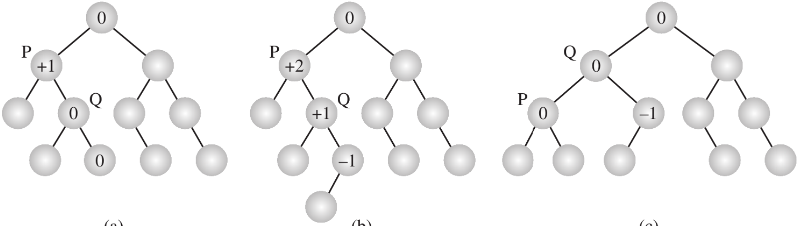
- Nesses dois casos, a árvore P é considerada uma árvore independente.
- Mas P pode fazer parte de uma árvore AVL maior, podendo ser um filho de algum outro nó na árvore.
- Ou seja, o balanceamento descrito **funciona mesmo se estivermos balanceando uma subárvore**.
- Uma dúvida comum seria: "Se um nó é inserido na árvore e o balanceamento de P é perturbado e depois restaurado, é necessário fazer algum ajuste nos predecessores de P?". A resposta é **não**.
- As mudanças feitas na subárvore P são suficientes para restaurar o balanceamento de toda a árvore AVL (pensem um pouco!).

Inserção em Árvores AVL

- O único problema é encontrar um nó P cujo fator de balanceamento se torne inaceitável após a inserção de um nó na árvore.
- Mas esse nó pode ser detectado subindo em direção à raiz da árvore a partir da posição onde o novo nó foi inserido e atualizando os fatores de balanceamento dos nós encontrados.
- Se for encontrado um nó com fator de balanceamento ± 1, esse fator pode ser alterado para ± 2, e este primeiro nó cujo fator de balanceamento é alterado dessa forma se torna a raiz P de uma subárvore cuja balanceamento precisa ser restaurado.
- Os fatores de balanceamento não precisam ser atualizados acima desse nó (dos predecessores), pois eles permanecem os mesmos.

Inserção em Árvores AVL

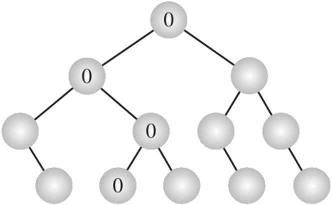
- Um exemplo de inserção de um novo nó em uma árvore AVL (figura b), que requer uma única rotação (figura c) para restaurar o balanceamento de altura.



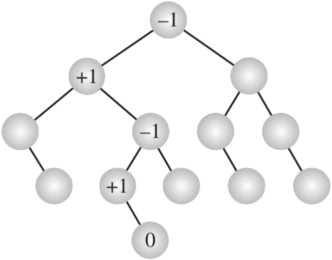
(a) (b) (c)

Inserção em Árvores AVL

- Um exemplo de inserção de um novo nó em uma árvore AVL (figura b) onde não são necessários ajustes de altura.



(a)



(b)

Inserção em Árvores AVL

- O algoritmo para atualizar os balanceamentos é dado à seguir:

```
ATUALIZACAO-BALANCEAMENTOS()
    Q = o nó recém-inserido;
    P = pai de Q;
    se Q é o filho esquerdo de P
        P->fatorDeBalanceamento--;
    senão
        P->fatorDeBalanceamento++;
    enquanto P não é a raiz e P->fatorDeBalanceamento ≠ ±2
        Q = P;
        P = pai de P;
        se Q->fatorDeBalanceamento == 0
            retorne;
        se Q é o filho esquerdo de P
            P->fatorDeBalanceamento--;
        senão
            P->fatorDeBalanceamento++;
        se P->fatorDeBalanceamento == ±2
            REBALANCEARSUBARVORE(P) // rebalanceie a subárvore com raiz em P;
```

Exercícios

1. Qual é a complexidade computacional, no pior caso, do tempo para inserir um novo nó em uma Árvore AVL? Em que situação ocorre o pior caso?
2. Qual é a complexidade computacional, no pior caso, do tempo para buscar um nó em uma Árvore AVL? Em que situação ocorre o pior caso?
3. Implemente em C/C++ os códigos para rotação à esquerda e rotação à direita. Garanta que os fatores de balanceamento sejam atualizados durante o processo de rotação.
4. Implemente em C/C++ as funções `ATUALIZACAO-BALANCEAMENTOS` e `REBALANCEARSUBARVORE`. Na função `REBALANCEARSUBARVORE`, assegure-se de tratar os quatro casos distintos de desbalanceamento que podem ocorrer em uma árvore AVL.
5. Finalmente, implemente a Árvore AVL e sua operação de inserção balanceada em C/C++. Ou seja, incluindo os códigos de rotação e atualização dos fatores de balanceamento.