MAT A40 - Estrutura de Dados e Algoritmos I

Dr. George Lima
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade Federal da Bahia

Módulo 6

Árvore de binária de busca auto-balanceadas

Desempenho dos algoritmos em Árvores Binárias

Observações:

- Os algoritmos de busca, inclusão e remoção têm custo (no pior caso) proporcional à altura da árvore.
- Manter a árvore balanceada é importante.
- Remoção e inclusão de nós podem deixar a árvore desbalanceada
- A ordem de inserção de elementos interfere no balanceamento em árvores binárias de busca.
- Geralmente, insere-se cada elemento uma vez para buscá-lo várias vezes.
- Aumentar o custo da inserção/remoção de nós numa árvore pode ser efetivo para as buscas.

Balanceamento em Árvores Binárias de Busca

Balanceamento perfeito em altura

Uma árvore binária é perfeitamente balanceada se para cada um de seus nós, as sub-árvores esquerda e direita tem a mesma altura.

Balanceamento aproximado em altura – AVL

Uma árvore binária AVL é aquela para a qual para cada um de seus nós, as alturas entre as sub-árvores esquerda e direita está entre -1 e 1.

Observação: a sigla AVL vem dos nomes dos autores soviéticos, Georgy Adelson-Velsky e Evgenii Landis (artigo publicado em 1962).

Balanceamento aproximado em altura – Árvores Vermelho-Preto

Numa árvore vermelho-preto, o tamanho do caminho entre a raíz e a folha mais distante não é maior que o dobro do tamanho do caminho entre a raíz e a folha mais próxima.

Árvore AVL

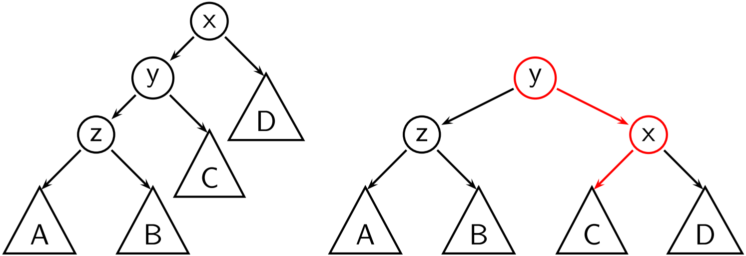
Definição

Ávores AVL são árvores binárias de busca que mantém, para cada um dos seus nós, a diferença entre as alturas das sub-árvores direita e esquerda contida no intervalo entre -1 e 1.

- É conveniente manter, para cada nó da árvore, um campo representando a altura do mesmo.
- Durante inserções e remoções pode haver a violação da restrição do fator de balanceamento para algum nó da árvore.
- Após cada inserção ou remoção de nós, checa-se o balanceamento dos nós. Caso necessário, executa-se procedimentos de balanceamento da árvore.
- Duas operações fundamentais, rotação à direita e rotação à esquerda, são responsáveis pelo balanceamento. Até duas operações de rotação podem ser necessárias para efetuar o balanceamento.

Balanceamento AVL – Caso 1

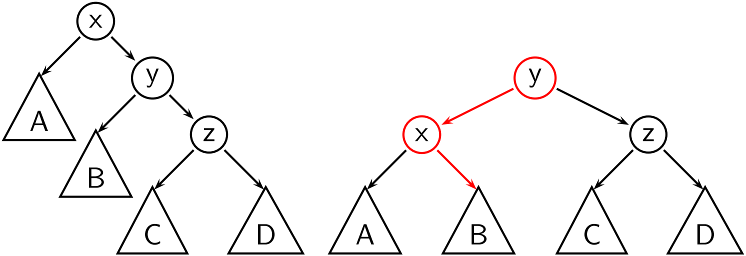
Desbalanceamento alinhado e à esquerda ⇒ Rotação à direita.




```
node_t *avl_rRight(node_t *x) {
  node_t *y = x -> left;
  x -> left  = y -> right;
  y -> right = x;
  x -> height = 1 + MAX(avl_height(x -> left),
    avl_height(x -> right));
  y -> height = 1 + MAX(avl_height(y -> left),
    avl_height(y -> right));
  return y;
}
```

Balanceamento AVL – Caso 2

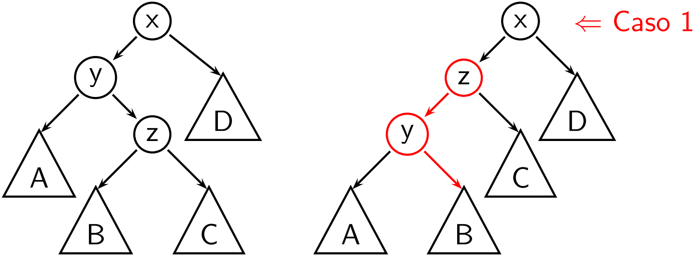
Desbalanceamento alinhado e à direita ⇒ Rotação à esquerda.




```c
node_t *avl_rLeft(node_t *x) {
  node_t *y = x -> right;
  x -> right = y -> left;
  y -> left  = x;
  x -> height = 1 + MAX(avl_height(x -> left),
     avl_height(x -> right));
  y -> height = 1 + MAX(avl_height(y -> left),
     avl_height(y -> right));
  return y;
}
```

Balanceamento AVL – Caso 3

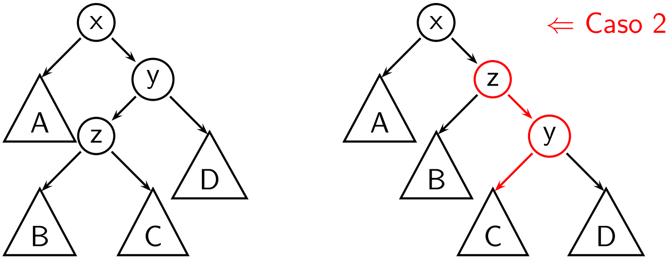
Desbalanceamento desalinhado e à esquerda $\Rightarrow$ Duas rotações.



Aplica-se rotação à esquerda entre os nós y e z e em seguida rotação à direita entre os nós x e z.

Balanceamento AVL – Caso 4

Desbalanceamento desalinhado e à direita $\Rightarrow$ Duas rotações.



Aplica-se rotação à esquerda entre os nós y e z e em seguida rotação à direita entre os nós x e z.

Inserção numa Árvore AVL

```
1  node_t *avl_insert(int e, node_t *r) {
2    /* Insere o elemento na arvore de busca */
3    if (r == NULL) {
4      r = newNode();
5      r -> t_element = e;
6      return r; // novo nao precisa de balanceamento
7    }
8    else if (r -> t_element > e)
9       r -> left = insert(e, r -> left);
10   else if (r -> t_element < e)
11      r -> right = insert(e, r -> right);
12
13   /* Checa balanceamento e ajusta arvore */
14   return rebalance(r);
15 }
```

- Notar que a linha 13 é executada para todos os nós ao longo do caminho percorrido até a inserção do novo elemento.

Remoção numa Árvore AVL

```
1  node_t *avl_delete(int e, node_t *r) {
2    node_t *p;
3    if (r == NULL) return r; // nao encontrado
4    if (r -> element < e) // procura pela direita
5      r -> right = avl_delete(e, r -> right);
6    else if (r -> element > e) // procura pela esquerda
7      r -> left = avl_delete(e, r -> left);
8    else {  // r contem elemento e
9      if (r -> left == NULL) {// r sem filho esq.
10       p = r;  r = r -> right;
11       free(p);
12     } else if (r -> right == NULL) {// r sem filho dir.
13       p = r; r = r -> left;
14       free(p);
15     } else { // r com ambos os filhos
16       p = get_max(r -> left);
17       r -> element = p -> element;
18       r -> left = avl_delete(p -> element, r -> left);
19     }
20   }
21   if (r) rebalance(r);
22   return r;
23 }
```

- Notar as linhas 21-22. A função `rebalance` só é chamada quando `r != NULL`. Quando `r` pode ser `NULL` no final da função?

Realizando o rebalanceamento numa árvore AVL

```
node_t *rebalance(node_t *r) {

  /* Recalculando a altura de r */
  int lh = avl_height(r -> left);
  int rh = avl_height(r -> right);
  int fb = lh - rh;

  r -> height = 1 + MAX(lh,rh);

  /* balanceamento da arvore  se necessario */
  if (fb == 2) { // casos 1 ou 3
    if (avl_height(r -> left -> right) > avl_height(r -> left -> left) )
      r -> left = avl_rLeft(r -> left);
    r = avl_rRight(r);
  } else if (fb == -2) { // casos 2 ou 4
    if (avl_height(r -> right -> left) > avl_height(r -> right -> right))
      r -> right = avl_rRight(r -> right);
    r = avl_rLeft(r);
  }
  return r;
}
```

Exercícios

1. Modifique as funções de inserção e remoção em árvores AVL para usar suas formas iterativas.
2. Qual o número máximo de operações de rotação a serem realizadas ao se inserir um elemento numa árvore AVL? E para remoção?
3. Suponha uma árvore binária de busca com n nós e assuma que que estes foram inseridos em ordem aleatória. Conduza experimentos para contar o número de acessos para pesquisar cada um dos elementos da árvore considerando que: (a) trata-se de uma árvore é AVL e (b) trata-se de uma árvore binária de busca (sem balaceamentos).
4. Refaça os experimentos do item anterior considerando que os n elementos foram inseridos na árvore em ordem crescente.

Árvore Vermelho-Preto

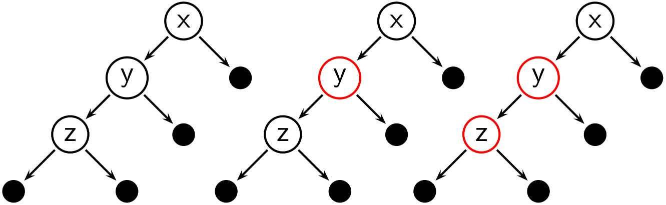
Definição

Uma árvore vermelho-preto é uma árvore binária de busca cujos nós possuem um bit adicional que sinaliza sua cor, que pode ser vermelha ou preta. Tal árvore binária de busca é uma árvore vermelho-preto se as seguintes restrições são satisfeitas:

1. Todo nó é vermelho ou preto.
2. A raíz é preta.
3. Toda folha é preta (nós nulos são considerados folhas).
4. Se um nó é vermelho, então seus filhos são pretos.
5. Para cada nó, todos os caminhos desde deste este até as folhas descendentes contém o mesmo número de nós pretos.

Ilustração

Exemplos de árvores que não são vermelho-preto (violação das regras 3 ou 4):



- Como não é possível colorir esta configuração de árvore, uma rotação deve ser realizada, similarmente ao Caso 1 das árvores AVL.
- Numa árvore vermelho-preto, inicialmente tenta-se a re-coloração dos nós de forma a satisfazer as regras que as define. Quando isso não é possível, executa-se operações de rotação.

Mais informações sobre árvore vermelho-preto

Observação:

- Detalhes sobre os algoritmos de inserção e remoção de nós em árvores vermelho-preto podem ser obtidos de outras fontes. Como a terceira avaliação avaliação da turma para o semestre 2018.2 envolverá o estudo e implementação destes algoritmos, detalhes dos mesmos são omitidos aqui. Uma fonte recomendável é:
  - Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. Introduction to Algorithms, 3rd Edition. MIT Press, 2009.
    - Há versões em português deste livro disponíveis na biblioteca da UFBA.