Árvore AVL

Origem: Wikipédia, a enciclopédia livre.

Árvore AVL é uma árvore binária de busca balanceada[1].

As operações de busca, inserção e remoção de elementos possuem complexidade $O(\log n)$ (no qual n é o número de elementos da árvore).

O nome AVL vem de seus criadores soviéticos Adelson Velsky e Landis, e sua primeira referência encontra-se no documento "*Algoritmos para organização da informação*" de 1962 [2].

Nessa estrutura de dados cada elemento é chamado de nó. Cada nó armazena uma chave e dois ponteiros, uma para a subárvore esquerda e outro para a subárvore direita.

No presente artigo serão apresentados: os conceitos básicos, incluindo uma proposta de estrutura; apresentação das operações busca, inserção e remoção, todas com complexidade $O(\log n)$.

Índice

História

Esta estrutura foi criada em 1962 pelos soviéticos Adelson Velsky e Landis que a criaram para que fosse possível inserir e buscar um elemento em tempo c.log (n) operações, onde n é o número de elementos contido na árvore. Tal estrutura foi a primeira árvore binária balanceada criada[3].

Conceitos básicos

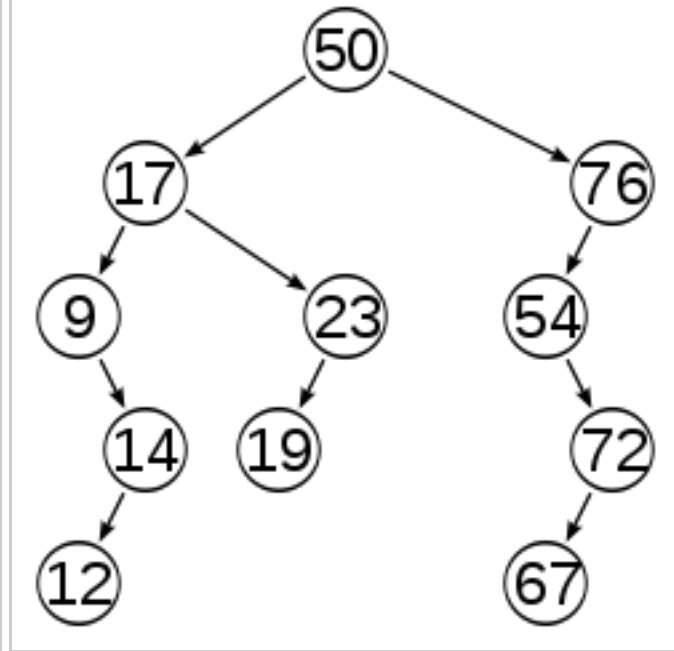
Definição

Uma árvore binária T é denominada AVL quando, para qualquer nó de T, as alturas de suas duas subárvores, esquerda e direita, diferem em módulo de até uma unidade.[4]

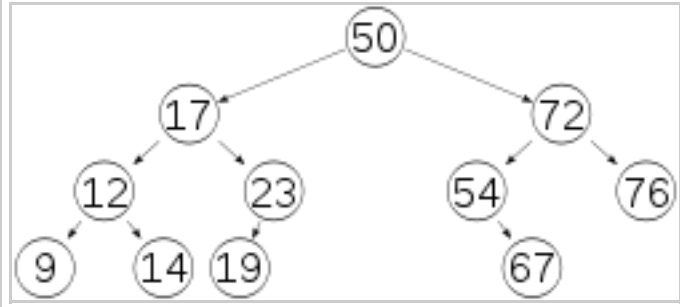
Pela definição fica estabelecido que todos os nós de uma árvore AVL devem respeitar a seguinte propriedade:

$|h_d(u) - h_e(u)| \leq 1$, onde $h_d(u)$ é a altura da subárvore direita do nó u e $h_e(u)$ é a altura da subárvore esquerda do nó u.

O valor $h_d(u) - h_e(u)$ é denominado **fator de balanço** do nó. Quando um nó possui fator de balanço com valor -1, 0 ou 1 então o mesmo é um **nó regulado**. Todos os nós de uma árvore AVL são regulados, caso contrário a árvore não é AVL.



Uma árvore não AVL



Uma árvore AVL

Estrutura

Proposta de estrutura dos nós de uma árvore AVL básica, com chave do tipo inteiro:

```
tipo No_AVL = registro
    chave: inteiro;
    fb: inteiro; // Fator de Balanço
    esq: ^No_AVL;  // aponta subárvore esquerda
    dir: ^No_AVL;  // aponta subárvore direita
fim;
```

O campo chave armazena o valor da chave. Os campos esq e dir são ponteiros para as subárvores esquerda e direita, respectivamente. O campo fb armazena o fator de balanço.

Definição da estrutura da árvore:

```
tipo Arvore_AVL = registro
    raiz: ^No_AVL;
    // definição de outros campos de interesse
fim;
```

Balanceamento

Toda árvore AVL é balanceada, isto é, sua altura é O(log n).[4]

A vantagem do balanceamento é possibilitar que a busca seja de complexidade O(log n). Entretanto, as operações de inserção e remoção devem possuir custo similar. No caso da árvore AVL, a inserção e remoção têm custo O(log n).

Por definição, todos os nós da AVL devem ter fb = -1, 0 ou 1.

Para garantir essa propriedade, a cada inserção ou remoção o fator de balanço deve ser atualizado a partir do pai do nó inserido até a raiz da árvore. Na inserção basta encontrar o primeiro nó desregulado (fb= -2 ou fb= 2), aplicar o operação de rotação necessária, não havendo necessidade de verificar os demais nós. Na remoção a verificação deverá prosseguir até a raiz, podendo requerer mais de uma rotação.

Uma árvore AVL sempre terá um tamanho menor que[5]:

$$\log_{\varphi}(\sqrt{5}(n+2)) - 2 = \frac{\log_2(\sqrt{5}(n+2))}{\log_2(\varphi)} - 2 = \log_{\varphi}(2) \cdot \log_2(\sqrt{5}(n+2)) - 2 \approx 1.44 \log_2(n+2) - 0.328$$

Onde n é o número de elementos da árvore e φ é a proporção áurea.

Complexidade

A **árvore AVL** tem complexidade O(log n) para todas operações e ocupa espaço n, onde n é o numero de nós da árvore.

	Média	Pior Caso
Espaço	O(n)	O(n)
Busca	O(log n)	O(log n)
Inserção	O(log n)	O(log n)
Deleção	O(log n)	O(log n)

Complexidade da árvore AVL em notação O.

Operações

Busca

A busca é a mesma utilizada em árvore binária de busca.

A busca pela chave de valor K inicia sempre pelo nó raiz da árvore.

Seja pt_u um ponteiro para o nó u sendo verificado. Caso o pt_u seja nulo então a busca não foi bem sucedida (K não está na árvore ou árvore vazia). Verificar se a chave K igual pt_u->chave (valor chave armazenado no nó u), então a busca foi bem sucedida. Caso contrário, se K < pt_u->chave então a busca segue pela subárvore esquerda; caso contrário, a busca segue pela subárvore direita.

Algoritmo de busca

```
busca_AVL(@pt_u:^no_AVL, K:inteiro):logico;
inicio
se  pt_u é NULO então retornar Falso;
se K = pt_u->chave então retornar Verdadeiro;
senão se K < pt_u->chave então
         retornar busca_AVL(K, u->esq);
      senão retornar busca_AVL(K, u->dir);
fim.
```

Inserção

Para inserir um novo nó de valor **K** em uma **árvore AVL** é necessária uma busca por **K** nesta mesma árvore. Após a busca o local correto para a inserção do nó **K** será em uma subárvore vazia de uma folha da árvore. Depois de inserido o nó, a altura do nó pai e de todos os nós acima deve ser atualizada. Em seguida o algoritmo de rotação simples ou dupla deve ser acionado para o primeiro nó pai desregulado.

Algoritmo recursivo

```
inserir_AVL(@p:^no_AVL, K:inteiro, @mudou_h:logico):logico;
var aux: logico; // variável local auxiliar;
inicio
se  p = NULO então
    inicio // não achou chave, inserir neste local
    p := novo(no_AVL);  // aloca novo nó dinamicamente, diretmente na subárvore do nó pai
    p^.chave := K;
    p^.esq := NULO;
    p^.dir := NULO;
    mudou_h := Verdadeiro; // sinalizar que a altura da subárvore mudou 1 unidade
    retornar Verdadeiro;
    fim;
se K < p^.chave então // inserir recursivamente na subárvore esquerda
    se inserir_AVL(p^.esq, K, mudou_h) então // ocorreu insersão na subárvore esquerda
        inicio
        se mudou_h então
            inicio  // mudou altura da subárvore esquerda de p
            p^.fb := p^.fb - 1; // fator de balanço decrementa 1 unidade
            caso p^.fb
                -2: p := rotacao_direita(p); mudou_h := Falso;
                 0: mudou_h := Falso; // não mudou a altura da subárvore de raiz p
            //   -1: mudou_h := Verdadeiro; // desnecessário pois mudou_h já é Verdadeiro
            fim
        retornar Verdadeiro;
        fim
senão
    se K > p^.chave então  // ocorreu insersão na subárvore direita
        se inserir_AVL(p^.dir, K, mudou_h) então // ocorreu insersão
        inicio
        se mudou_h então
            inicio  // mudou altura da subárvore esquerda de p
            p^.fb := p^.fb + 1; // fator de balanço incrementa 1 unidade
            caso p^.fb
                 2: p := rotacao_esquerda(p); mudou_h := Falso;
                 0: mudou_h := Falso; // não mudou a altura da subárvore de raiz p
            //   1: mudou_h := Verdadeiro; // desnecessário pois mudou_h já é Verdadeiro
            fim
        retornar Verdadeiro;
        fim
retornar Falso;
fim.
```

Os parâmetros **p** e **mudou_h** são passados por referência. O ponteiro **p** aponta para o nó atual. O parâmetro **mudou_h** é do tipo lógico e informa ao chamador se a subárvore apontada por p mudou sua altura.

Como identificar mudança de altura ?

Considerar que o nó p é raiz da subárvore T_p e houve inserção em uma de suas subárvores.

Caso a subárvore T_p tenha mudado de altura, decrementar fb (inserção na subárvore esquerda) ou incrementar fb (inserção na subárvore direita).

Caso 1: Ao inserir um nó folha, a subárvore T_p passa de altura 0 para altura 1, então T_p mudou de altura.

Caso 2: fb=0 antes da inserção foi alterado para 1 ou -1, então a subárvore T_p mudou de altura.

Caso 3: fb=1 ou -1 antes da inserção, passou a ter valor 0, então a subárvore T_p não mudou de altura.

Caso 4: O fb passou a ter valor -2 ou 2 após a inserção, então há necessidade de aplicação de alguma operação de rotação. Após a rotação, a subárvore T_p terá a mesma altura anterior à inserção.

Remoção

O primeiro passo para remover uma chave K consiste em realizar uma busca binária a partir do nó raiz. Caso a busca encerre em uma subárvore vazia, então a chave não está na árvore e a remoção não pode ser realizada. Caso a busca encerre em um nó **u** o nó que contenha a chave então a remoção poderá ser realizada da seguinte forma:

Caso 1: O nó **u** é uma folha da árvore, apenas exclui-lo.

Caso 2: O nó **u** tem apenas uma subárvore, necessariamente composta de um nó folha, basta apontar o nó pai de **u** para a única subárvore e excluir o nó **u**.

Caso 3: O nó **u** tem duas subárvores: localizar o nó **v** predecessor ou sucessor de **K**, que sempre será um nó folha ou possuirá apenas uma subárvore; copiar a chave de v para o nó u; excluir o nó v a partir da respectiva subárvore de u.

O último passo consiste em verificar a desregulagem de todos nós a partir do pai do nó excluído até o nó raiz da árvore. Aplicar rotação simples ou dupla em cada nó desregulado.

Algoritmo recursivo

```
remover_AVL(@p:^no_AVL, K:inteiro, @mudou_h:logico):logico;
var q:^No_AVL;  // ponteiro auxiliar para nó
inicio
se  p = NULO então
    retornar Falso; // não achou a chave K a ser removida
se K < p^.chave então // remover recursivamente na subárvore esquerda
    se remover_AVL(p^.esq, K, mudou_h) então // ocorreu remoção na subárvore esquerda
      inicio
      se mudou_h então
          inicio  // mudou altura da subárvore esquerda de p
          p^.fb := p^.fb + 1; // fator de balanço incrementa 1 unidade
          caso p^.fb
              2:  p := rotacao_esquerda(p);
                  se (p->fb=1) então mudou_h := Falso;
              1: mudou_h := Falso; // não mudou a altura da subárvore de raiz p
             // 0: mudou_h := Verdadeiro; // desnecessário pois mudou_h já é Verdadeiro
          fim
      retornar Verdadeiro;
      fim
senão
   se K > p^.chave então
      se remover_AVL(p^.dir, K, mudou_h) então // ocorreu remoção na s.a. direita
          inicio
          se mudou_h então
              inicio  // mudou altura da subárvore direita de p
              p^.fb := p^.fb - 1; // fator de balanço decrementa 1 unidade
              caso p^.fb
                  -2: p := rotacao_direita(p);
                      se (p->fb = -1) então mudou_h := Falso;
                  -1: mudou_h := Falso; // não mudou a altura da subárvore de raiz p
              //   0: mudou_h := Verdadeiro; // desnecessário pois mudou_h já é Verdadeiro
              fim
          retornar Verdadeiro;
          fim
   senão inicio
          se K = p^.chave então // achou a chave K
              inicio
              se p^.esq=Nulo e p^.dir=Nulo então
                  inicio // nó folha
                  delete p;
                  p := Nulo; // Aterra a subárvore respectiva do nó pai
                  mudou_h := Verdadeiro;
                  fim
              senão se p^.esq<>Nulo e p^.dir<>Nulo então
                      inicio  // nó tem duas subárvores
                      q := Predecessor(p);  // retorna nó com chave predecessora
                      p^.chave := q^.chave;
```

```
                        remover(p^.esq,p^.chave,mudou_h);
                        fim
                    senão inicio // tem apenas uma subárvore
                          se p^.esq<>Nulo então
                              inicio
                              p^.chave := p^.esq^.chave;
                              delete p^.esq;
                              p^.esq := Nulo;
                              fim
                          senão   inicio
                                  p^.chave := p^.dir^.chave;
                                  delete(p^.dir);
                                  p^.dir := Nulo;
                                  fim;
                          mudou_h = Verdadeiro;
                          fim;
                retornar Verdadeiro;
                fim
            fim
retornar Falso;
fim;

Predecessor(u:^No_AVL):^No_AVL // retorna nó contendo chave predecessora
inicio
u = u^.esq; // aponta para a raiz da subárvore esquerda
enquanto(u^.dir<>Nulo) faça  // procura a maior chave da subárvore esquerda
    u := u^.dir;
retornar u;  // retorna o predecessor
fim;
```

Como identificar mudança de altura na remoção ?

Considerar que o nó p é raiz da subárvore T_p e houve remoção em uma de suas subárvores.

Caso a subárvore T_p tenha mudado de altura, incrementar fb (remoção na subárvore esquerda) ou decrementar fb (remoção na subárvore direita).

Caso 1: Ao remover um nó folha, a subárvore T_p passa de altura 1 para altura 0, então T_p mudou de altura.

Caso 2: fb=0 antes da remoção foi alterado para 1 (remoção à esquerda) ou -1 (remoção à direita), então a subárvore T_p não mudou de altura.

Caso 3: fb=1 ou -1 antes da remoção à direita e à esquerda, respectivamente, passando a ter valor 0, então a subárvore T_p mudou de altura.

Caso 4: fb passou a ter valor -2 ou 2 após a remoção, então houve necessidade de aplicação de rotação. Após a rotação dupla T_p muda de altura, exceto quando u=0 (antes da remoção- caso não relatado na literatura).

Rotação

Após uma inserção ou remoção na **árvore AVL** pode ocorrer seu desbalanceamento. Caso seja identificado algum nó desregulado (fb = 2 ou -2), haverá necessidade de aplicar uma rotação simples ou dupla, conforme os casos apresentados a seguir.

Seja p o nó raiz da subárvore em análise e o mesmo está desregulado. Para restabelecer a regulagem do nó p poderá ser utilizada uma das quatro transformações descritas a seguir, denominadas , respectivamente, rotação simples à direita, rotação dupla à direita, rotação simples à esquerda, rotação dupla à direita. As transformações preservam a natureza da árvore como sendo binária de busca.

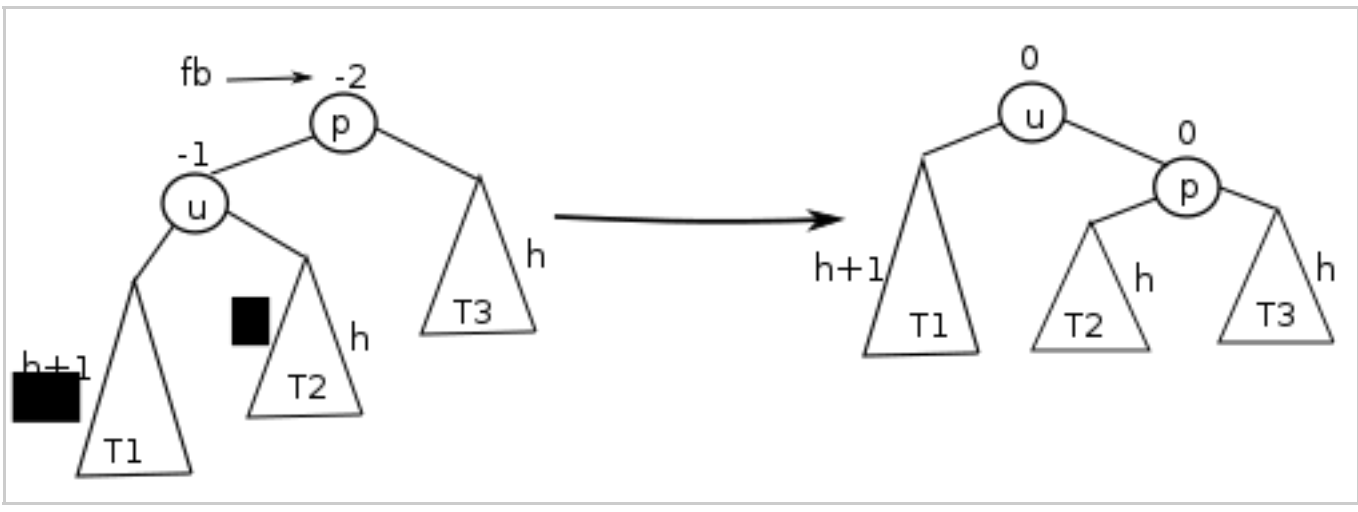
Rotação à direita

Caso o fator de balanço do nó p tenha valor -2, então haverá necessidade de aplicar rotação simples ou dupla à direita.

Para identificar se qual rotação aplicar, bastará analisar o fator de balanço do nó u, raiz da subárvore esquerda do nó p.

Rotação simples à direita

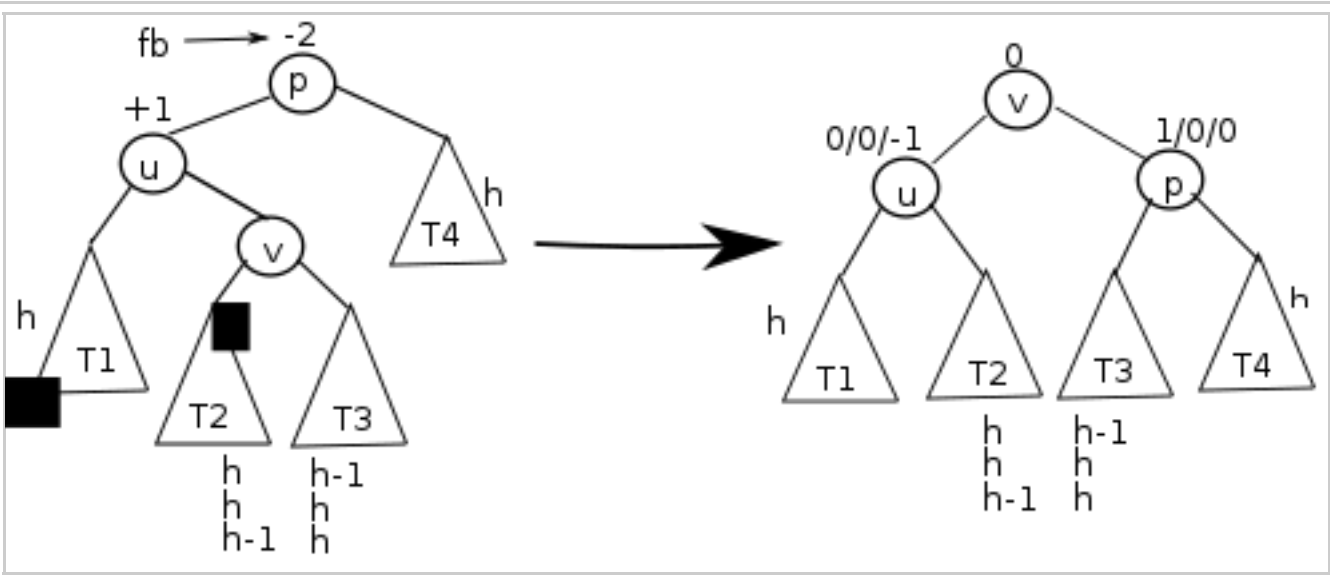
Caso o fator de balanço do nó u seja -1, então deverá ser aplicada uma rotação simples à direita. A figura a seguir mostra a configuração da árvore antes e depois da rotação.



Caso 1.1 - Rotação Simples à Direita

Rotação dupla à direita

Caso o fator de balanço do nó u seja +1, então deverá ser aplicada uma rotação dupla à direita. A figura a seguir mostra a configuração da árvore antes e depois da rotação.



Caso 1.2 - Rotação dupla à direita

Deve ser observado que as 3 possíveis combinações de alturas da subárvores T_2 e T_3 constam da figura (h, h-1; h, h; e h-1, h), implicando em nos respectivos balanços do nó u (0/0/-1) e do nó p (1/0/0).

Algoritmo de rotação à direita

```
rotaçaoDireita(p:NoAVL):@NoAVL;
inicio
var u,v:@NoAVL;
u := p^.esq;
se u^.fb > 0 então
    inicio // rotação dupla, conforme figura Caso 1.2
     v := u^.dir;
     u^.dir := v^.esq;
     p^.esq := c^.dir;
     v^.esq := u;
```

```
     v^.dir := p;
     caso v->fb
        -1: u^.fb := 0; p^.fb := 1;
        0: u^.fb := 0; p^.fb := 0;
        1: u^.fb := -1; p^.fb := 0;
     v^.fb := 0;
     retornar v;
     fim;
// rotaçao simples, conforme figura Caso 1.1
p^.esq := u^.dir;
u^.dir := p;
se u^.fb < 0 então
    inicio
    u^.fb := 0;
    p^.fb := 0;
    fim;
senão inicio  // ocorre apenas na remoção - não relatado na literatura
      u^.fb := 1;
      p^.fb := -1;
      fim;
retornar u;
fim;
```

Cabe ressaltar, apesar de não estar relatado na literatura, que na remoção poderá ocorre a situação na qual u^.fb=0. Neste caso deverá ser aplicada rotação simples e o fator de balanço dos nós u e p serão respectivamente 1 e -1.

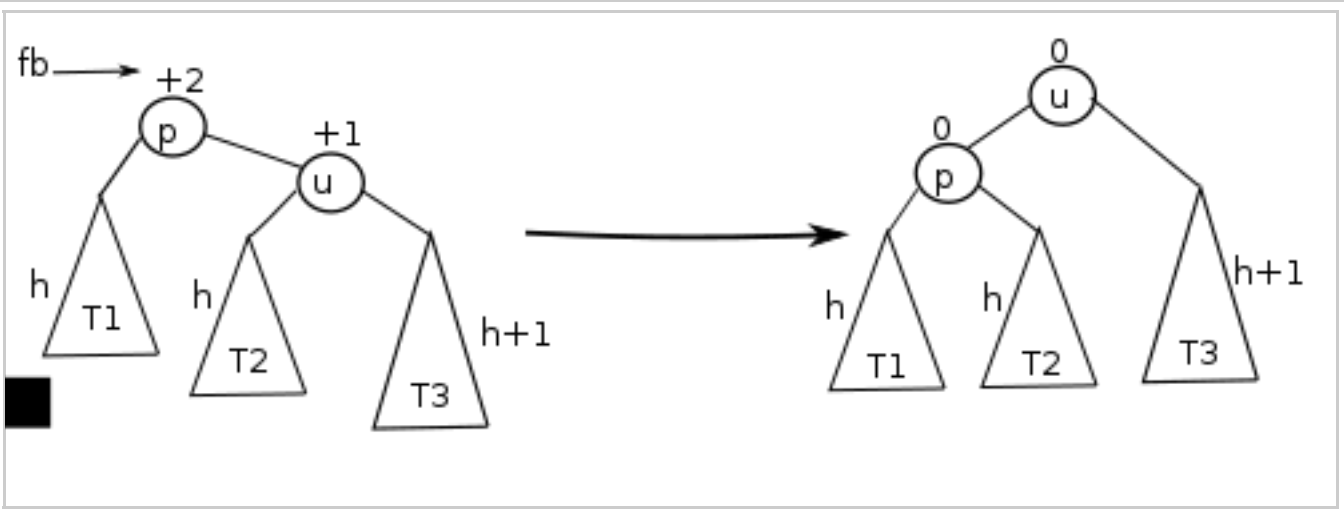
Rotação à esquerda

Caso o fator de balanço do nó p tenha valor +2, então haverá necessidade de aplicar rotação simples ou dupla à esquerda.

Para identificar se qual rotação aplicar, bastará analisar o fator de balanço do nó u, raiz da subárvore direita do nó p.

Rotação simples à esquerda

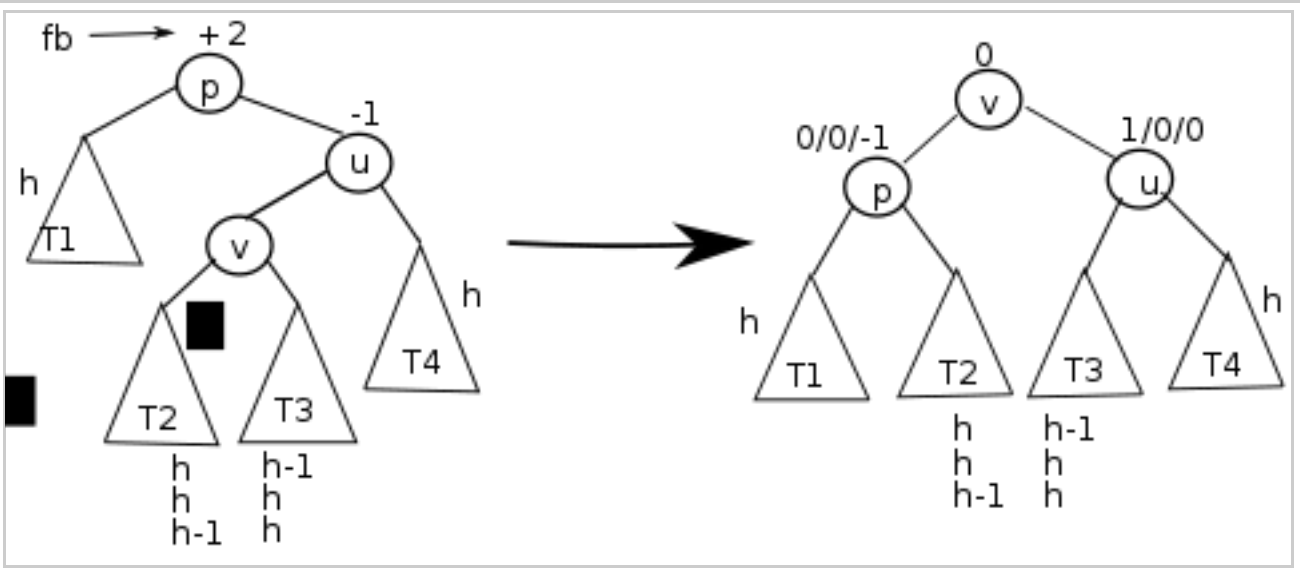
Caso o fator de balanço do nó u seja +1, então deverá ser aplicada uma rotação simples à esquerda. A figura a seguir mostra a configuração da árvore antes e depois da rotação.



Caso 2.1 - Rotação Simples à Esquerda

Rotação dupla à esquerda

Caso o fator de balanço do nó u seja -1, então deverá ser aplicada uma rotação dupla à esquerda. A figura a seguir mostra a configuração da árvore antes e depois da rotação.



Caso 2.2 - Rotação Dupla à Direita

Aplicações

A **árvore AVL** é muito útil pois executa as operações de inserção, busca e remoção em tempo O(log n) sendo inclusive mais rápida que a árvore rubro-negra para aplicações que fazem uma quantidade excessiva de buscas, porém esta estrutura é um pouco mais lenta para inserção e remoção. Isso se deve ao fato de as **árvores AVL** serem mais rigidamente balanceadas.

Dicionários

Árvore AVL pode ser usada para formar um dicionário de uma linguagem ou de programas, como os opcodes de um assembler ou um interpretador.

Geometria Computacional

Árvore AVL pode ser usada também na geometria computacional por ser uma estrutura muito rápida. Sem uma estrutura com complexidade O(log n) alguns algoritmos da geometria computacional poderiam demorar dias para serem executados.

Conjuntos

Árvore AVL podem ser empregadas na implementação de conjuntos, principalmente aqueles cujas chave não são números inteiros.

A complexidade das principais operações de conjuntos usando árvore AVL:

- Inserir - O(log n);
- Remover - O(log n);
- Pertence - O(log n);
- União - O(n.log n);
- Interseção - O(n.log n).

Referências

1. Cormen, Thomas H. (2009). *Introduction to Algorithms,* (Massachusetts Institute of Technology: MIT Press). p. 333.
2. Adel'son-Vel'skiĭ, G. M.; Landis, E. M. (1962), "An algorithm for organization of information", Doklady Akademii Nauk SSSR, 146: 263–266.
3. Kent, Allen; Williams, James G. (1993), *Encyclopedia of Computer Science and Technology: Volume 28 - Supplement 13: AerosPate Applications of Artificial Intelligence to Tree Structures*, CRC Press, p. 373, ISBN 9780824722814.
4. Swarcfiter, Jayme Luiz (1994). *Estruturas de dados e seus algoritmos* (Rio de Janeiro, Brasil: LTC). p. 130.
5. Knuth, Donald E. (2000). *Sorting and searching* (2. ed., 6. printing, newly updated and rev. ed.). Boston [u.a.]: Addison-Wesley. p. 460. ISBN 0-201-89685-0.

Bibliografia

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Third Edition. MIT Press, 2009. p. 333. ISBN 978-0-262-53305-8.
- Adelson-Velskii, G.; E. M. Landis (1962). "An algorithm for the organization of information". *Proceedings of the* USSR Academy of Sciences ***146****: 263–266*. ***(Russian)*** *English translation by Myron J. Ricci in* Soviet Math. Doklady*, 3:1259–1263, 1962*.
- Jayme Luiz Szwarcfiter, Lilian Markenzon. Estruturas de Dados e Seus Algoritmos, 2010. p. 130-144. ISBN 85-215-1014-9.

Ver também

- Árvore binária de busca
- Árvore rubro-negra
- Árvore B
- Árvore Patricia
- Inserção, remoção e alteração em árvores AVL (http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html)
- Notação O