ÁRVORES BALANCEADAS (AVL)

PAULO JOSÉ DA SILVA E SILVA

1. Árvores balanceadas

Como já vimos anteriormente, o uso de árvores binárias de busca sem a preocupação com seu balanceamento pode levar a um aumento de sua altura quando comparada a altura de uma árvore balanceada com o mesmo número de nós. Mas afinal o que é um árvore balanceada? Dizemos que uma árvore é *perfeitamente balanceada* se ela tiver apenas a altura mínima necessária para conter os seus nós.

Lembrando que o número de elementos em uma árvore de altura h é no máximo $2^{h+1}-1$, concluímos que para conter n nós uma árvore binária teve ter altura maior ou igual a $\lfloor \log_2 n \rfloor$.[1]

Desta forma, para que possamos garantir que a busca a um nó de uma árvore binária de busca será realmente rápida, demorando aproximadamente $\log_2 n$ operações, precisamos alterar as rotinas de inserção e remoção apresentadas de maneira a garantir o balanceamento das árvores binárias geradas. Infelizmente, é muito difícil manter árvores perfeitamente balanceadas.

2. Árvores AVL

Para contornar a dificuldade de manutenção de árvores perfeitamente balanceadas, uma das estratégias possíveis é relaxar o conceito de balanceamento de maneira a garantir que as árvores ainda serão “baixas”, mas agora “fáceis” de atualizar.

Uma das primeiras definições de árvores “quase balanceadas” que foi muito bem sucedida foi dada por Adelson-Velskii e Landis e estas árvores ficaram conhecidas como árvores AVL:

Definition 1. Uma árvore é dita AVL-balanceada (ou simplesmente AVL) se para todo nó a altura entre suas duas sub-árvores diferir em no máximo uma unidade.

Note que toda árvore perfeitamente balanceada é AVL, mas nem toda árvore AVL é balanceada, como mostra a Figura 1. Porém Adelson-Velskii e Landis provaram que árvores AVL têm a altura no máximo 44% maior do que a respectiva árvore balanceada, ainda sendo possíveis de ser atualizadas com operações proporcionais a $\log_2 n$.

Estudaremos a seguir as rotinas de inserção e remoção em árvores AVL.

Date: 26 de maio de 2001.

[1] $\lfloor x \rfloor$ simboliza o menor inteiro maior igual a x, conhecido como chão de x.

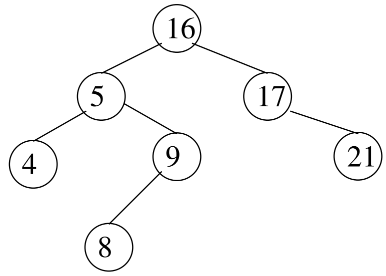


FIGURA 1. Árvore AVL, mas não balanceada

2.1. **Inserção.** O algoritmo de inserção em uma árvore AVL é recursivo e, em até certo ponto semelhante ao algoritmo de inserção em árvores binárias simples. Ele inicia com a busca do local correto para inserção do novo nó. Ao encontrar o local (que fica sempre em uma folha), insere-se o novo elemento e volta-se atrás pelo caminho percorrido na busca através da pilha de recursão. Ao voltar atrás o algoritmo deve informar ao novo nó se a sub-arvore de onde ele vem aumentou ou não de altura e, eventualmente, tomar as medidas necessárias para manter o balanceamento. Para que isto seja possível é necessário manter-se em cada nó a informação de qual o grau de balanceamento entre suas sub-árvores esquerda e direita.

Caso a sub-árvore onde foi inserido o novo elemento não tiver mudado de altura nada será necessário fazer, uma vez que a árvore original já era AVL. Já se a sub-árvore cresceu devemos verificar a nova condição de balanceamento. Por exemplo, suponhamos que a sub-árvore à esquerda de um nó X cresceu, existem três possibilidades quanto a nova condição de balanceamento de X, como mostra a Figura 2.[2]

(1) A árvore da esquerda era mais baixa que a sub-árvore da direita antes da inserção. Desta forma, a inserção à esquerda "melhorou" o balanceamento, deixando as duas árvores com a mesma altura.
(2) Ambas sub-árvores tinham mesma altura antes da inserção e assim a inserção deixou a sub-árvore da esquerda um nível mais alta, o que ainda é permitido pela definição de árvores AVL.
(3) O crescimento da sub-árvore da esquerda tenha levou-a a ficar dois níveis mais alta que a sub-árvore à direita de X, nesta situação deve-se fazer um re-balanceamento.

Nota-se aqui a necessidade de armazenar em cada nó a informação de como está seu balancemento. Para isto, vamos acrescentar um novo campo em nossa estrutura de nós, chamado `bal` que conterá a diferença das alturas das sub-árvores direita e esquerda do nó. Ou seja, este campo conterá -1 se a sub-árvore da esquerda for uma unidade mais alta, 0 se o nó tiver duas sub-árvores de mesma altura e 1 caso seja a sub-árvore da direita a mais alta.

Para fazer o re-balanceamento precisamos considerar apenas dois casos. Cabe ao leitor a observação atenta destes casos para que ele se conveça que estes são os únicos casos possíveis.

---

[2]Os casos nos quais a inserção é feita à direita são análogos.

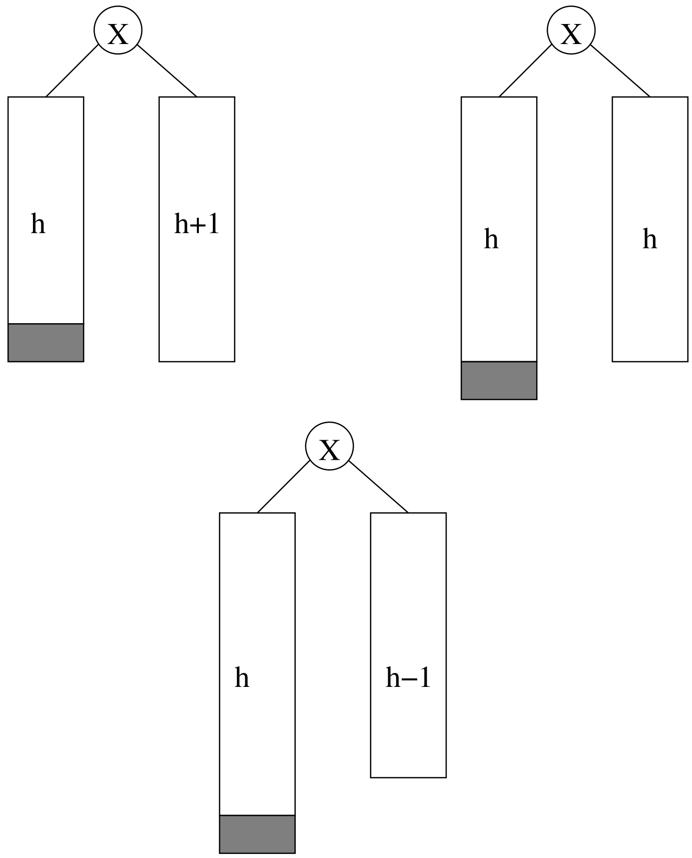


Figura 2. Situação de balanceamento após crescimento da sub-árvore à esquerda.

No primeiro o crescimento ocorreu à esquerda do filho da esquerda de X, como mostra a Figura 3. Neste caso as sub-árvores à direita de X e à direita do filho esquerdo de X devem ter mesma altura, caso contrário a condição de ser uma árvores AVL deveria ter sido violada antes da inserção ou em um passo anterior da recursão. Portando a correção deste caso pode ser feita com uma simples rotação à direita.

Já o segundo caso tem solução um pouco mais complicada. Nele o crescimento ocorreu à direita do filho esquerdo de X. Mais uma vez, para que a condição de AVL tenha sido respeitada anteriormente, podemos garantir que:

(1) As duas sub-árvores filhas do filho à direita do filho à esquerda de X têm a mesma altura.
(2) A sub-árvore à esquerda do filho da esquerda de X e a sub-árvore direita de X têm a mesma altura que é igual a altura das árvores descritas no ítem anterior mais 1.

Assim, podemos garantir que o balanceamento pode ser obtido com duas rotações, a primeira à esquerda deve ser entre o filho à esquerda de X e seu filho à direita. A segunda deve ocorrer com o novo filho à esquerda de X e X. Veja a Figura 4.

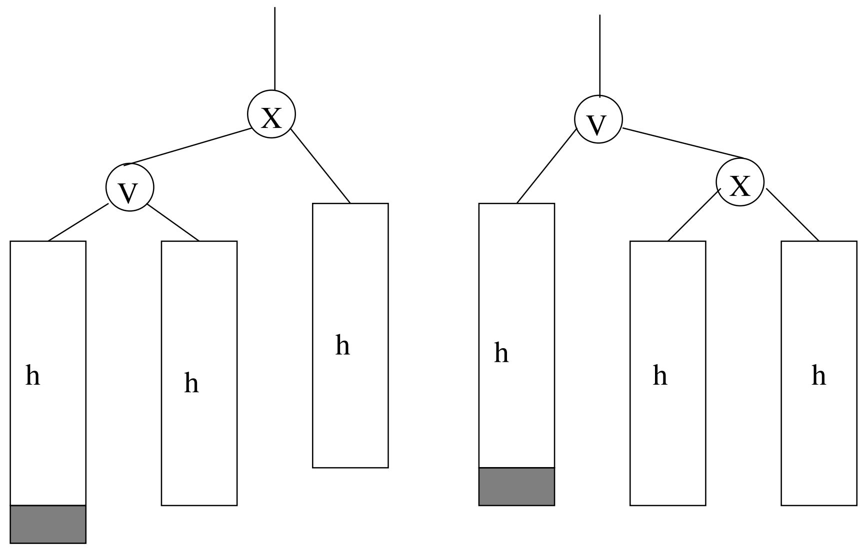


FIGURA 3. O crescimento ocorreu à esquerda do filho à esquerda de X. Solução: rotação à direita.

O código para a inserção será apresentado a seguir.

```
/* Estruturas para uma arvore de inteiros. */
typedef struct _no {
  int chave;         /* Chave para busca                          */
  int bal;           /* Indicacao sobre balanceamento:            */
                     /* -1 se esq. maior, 0 se balanceado,        */
                     /* 1 se dir maior                            */

  /* Outros campos de dados aqui */

  struct _no *esq, *dir; /* Ponteiros para sub-arvores esq e dir   */
} no;

int insereAVL(int x, no **p) {

  int cresceu;

  /* Se a arvore esta vazia insere. */
  if (*p == NULL) {
    *p = (no *) malloc(sizeof(no));
    (*p)->chave = x;
    /* Caso houvesse outros dados eles deveriam ser copiados aqui. */
    (*p)->dir = (*p)->esq = NULL;
    /* Balanceamento de uma folha e sempre perfeito */
    (*p)->bal = 0;
    /* Esta sub arvore cresceu */
    cresceu = 1;
```

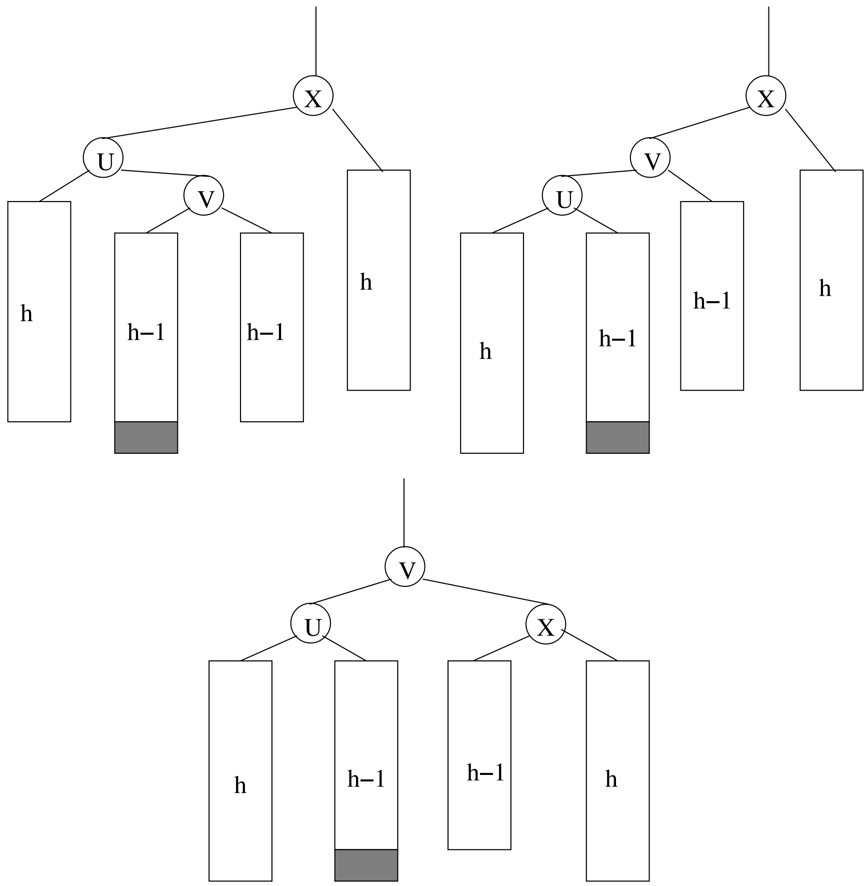


FIGURA 4. O crescimento ocorreu à direita do filho à esquerda de X. Solução: rotação dupla, primeiro à esquerda e depois à direita.

```
}
/* Senao verifica se tem que inserir a esquerda */
else if ((*p)->chave > x) {
  /* Tenta inserir a esquerda e ve se a sub-arvore cresceu */
  cresceu = insereAVL(x, &(*p)->esq);
  if (cresceu) {
    /* Verifica o estado atual de balanceamento */
    switch((*p)->bal) {
    /* Se a arvore dir. era maior nao ha crescimento               */
    case 1:
      (*p)->bal = 0;
      cresceu = 0;
      break;
    /* Se a arvore dir. tinha tamanho igual, houve crescimento     */
    case 0:
      (*p)->bal = -1;
```

```
        cresceu = 1;
        break;
      /* Se a sub-arvore da esq. ja era maior, deve-se re-balancear. */
      case -1:
        /* Verifica qual o caso de re-balanceamento.                  */
        /* Se a arvore da esquerda do filho da esquerda esta mais     */
        /* alta basta uma rotacao para direita.                       */
        if ((*p)->esq->bal == -1) {
          /* Rotacao para direita.     */
          rot_dir(p);
          /* Acerta os balanceamentos. */
          (*p)->bal = (*p)->dir->bal = 0;
        }
        else {
          /* Rotacao dupla.         */
          /* Primeiro a esquerda.   */
          rot_esq(&(*p)->esq);
          /* Depois a direita.      */
          rot_dir(p);
          /* Acerta balanceamentos. */
          if ((*p)->bal == -1) {
            (*p)->esq->bal = 0;
            (*p)->dir->bal = 1;
          }
          else {
            (*p)->dir->bal = 0;
            (*p)->esq->bal = -(*p)->bal;
          }
          (*p)->bal = 0;
        }
        cresceu = 0;
      }
    }
  }
  /* Verifica se tem que inserir a direita. */
  else if ((*p)->chave < x) {
    /* Tenta inserir a direita e ve se a sub-arvore cresceu */
    cresceu = insereAVL(x, &(*p)->dir);
    if (cresceu) {
      /* Verifica o estado atual de balanceamento */
      switch((*p)->bal) {
      /* Se a arvore esq. era maior nao ha crescimento              */
      case -1:
        (*p)->bal = 0;
        cresceu = 0;
        break;
      /* Se a arvore esq. tinha tamanho igual, houve crescimento    */
      case 0:
```

```
        (*p)->bal = 1;
        cresceu = 1;
        break;
      /* Se a arvore da dir. ja era maior, deve-se re-balancear.     */
      case 1:
        /* Verifica qual o caso de re-balanceamento.                 */
        /* Se a arvore da direita do filho da direita esta mais      */
        /* alta basta uma rotacao para esquerda.                     */
        if ((*p)->dir->bal == 1) {
          /* Rotacao para esquerda.    */
          rot_esq(p);
          /* Acerta os balanceamentos. */
          (*p)->bal = (*p)->esq->bal = 0;
        }
        else {
          /* Rotacao dupla.         */
          /* Primeiro a direita.    */
          rot_dir(&(*p)->dir);
          /* Depois a esquerda.     */
          rot_esq(p);
          /* Acerta balanceamentos. */
          if ((*p)->bal == -1) {
            (*p)->esq->bal = 0;
            (*p)->dir->bal = 1;
          }
          else {
            (*p)->dir->bal = 0;
            (*p)->esq->bal = -(*p)->bal;
          }
          (*p)->bal = 0;
        }
        cresceu = 0;
      }
    }
  }
  else cresceu = 0;

  return cresceu;
}
```

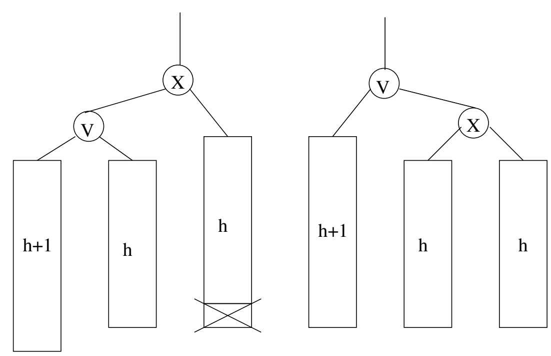
2.2. **Remoção.** A remoção em árvore AVL segue a mesma linha que a inserção. Desta forma apresentaremos apenas os esquemas de casos e deixamos o código como exercício (é um exercício difícil, mas que vale à pena ser feito), ou, para os mais curioso, há uma versão implementada no livro do Wirth [3].

Inicialmente o algoritmo deve buscar recursivamente o nó que deseja-se deletar. Uma vez encontrado, deve-se usar o algoritmo de remoção que não está baseado em rotações (que foi o tema do exercício da aula três sobe árvores). Feita a remoção, deve-se mais uma vez analisar o balanceamento.

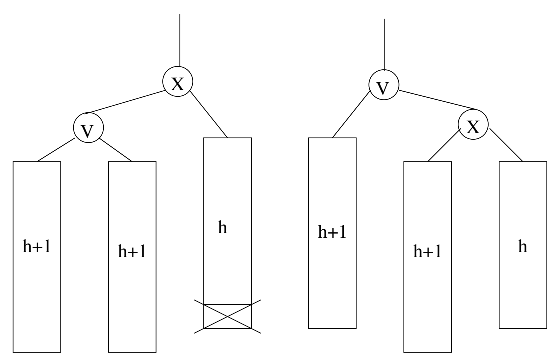
Novamente, iniciamos as possíveis mudança de balanceamento quando a remoção ocorre do lado direito de X resultado na diminuição do tamanho desta sub-árvore. Existem basicamente quatro casos que serão tratados a seguir.

No primeiro caso o desbalanceamento é determinado pela sub-árvore à esquerda do filho à esquerda de X. Aqui a recuperação do balanceamento pode ser obtida com uma simples rotação à direita entre X e seu filho à esquerda. Note que no primeiro sub-caso a altura da sub-árvore com raiz em X diminui, o que pode acarretar em novos re-balanceamentos em níveis superiores. Veja Figura 5.

No segundo caso quem determina o desbalanceamento é a sub-árvore direita do filho esquerdo de X. Aqui precisamos de uma dupla rotação, uma primeira à esquerda e em seguida à direita. Vejas as Figuras 6 e 7. Note que neste caso a altura da sub-árvore de raiz X sempre diminui, implicando numa eventual necessidade de re-balanceamento de ancestrais de X.



(a) A altura diminui



(b) A altura permanece constante

FIGURA 5. Remoção à direita de X e desbalanceamento causado pela árvore à esquerda do filho à esquerda de X.

Referências

[1] Notas de aula da Nami.
[2] Sedgewick, Robert. *Algorithms in C++: Parts 1-4*, 3 edição, Addison Wesley, 1998.

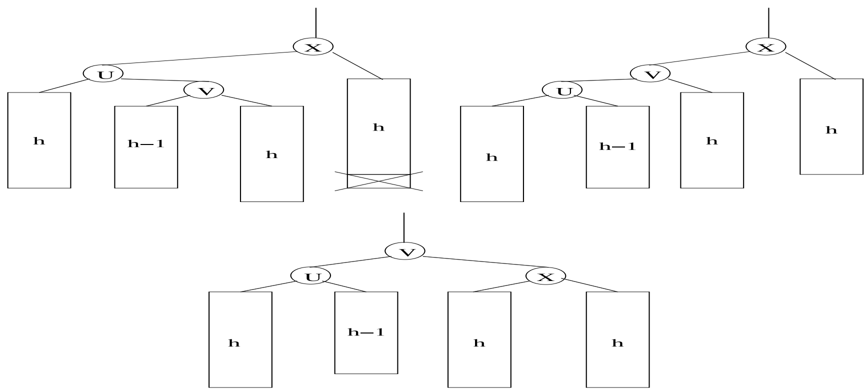


FIGURA 6. Uma das sub-árvores do filho à direita do filho à esquerda de X tem altura menor. Note que não importa qual delas o resultado é o mesmo.

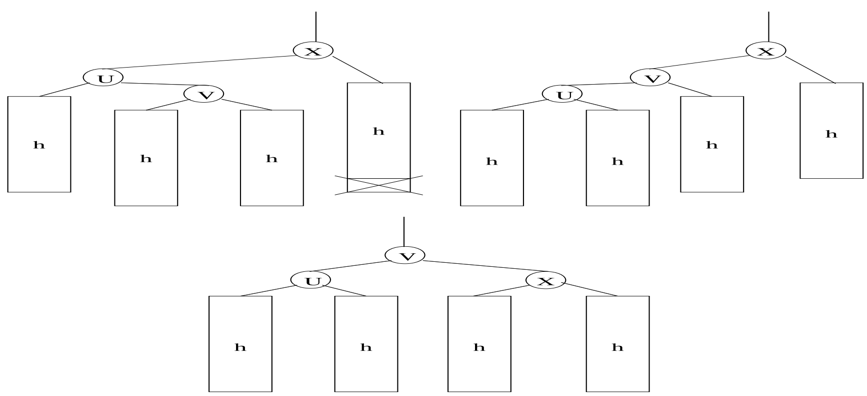


FIGURA 7. As sub-árvores do filho à direita do filho à esquerda de X têm mesma altura.

[3] Wirth, Niklaus. *Algoritmos e Estrturas de Dados*, Prentice Hall do Brasil, 1986.