Estruturas de Dados

Árvores AVL

Cesar Tadeu Pozzer

pozzer3@gmail.com, pozzer@inf.ufsm.br

Curso de Ciência da Computação
UFSM

(12/12/2007)

Árvore AVL

- Uma árvore binária é denominada AVL quando a diferença de altura entre as subárvores da esquerda e direita (sae e sad) de um nó qualquer N não é superior a 1.
- Uma árvore é considerada desbalanceada quanto esta propriedade não for verificada.
- O fato da árvore estar balanceada resulta em ganho de desempenho para operações realizadas sobre árvores binárias, como na inserção, remoção e consulta de valores.
- Se a árvore estiver balanceada, todas estas operações gastam, no pior caso, um tempo O(logn), que neste caso representa a altura da árvore binária balanceada.
- Se a árvore não estiver balanceada, o tempo de busca no pior caso pode ser O(n), ou seja, equivalente ao tempo de busca de valores em uma lista encadeada (no pior caso, todos os elementos tem que ser consultados).

Árvore AVL

- Uma árvore balanceada pode se tornar desbalanceada quando se aplica uma operação de inserção ou remoção de elementos.
- Neste caso, procedimentos de rebalanceamento devem ser aplicados a árvore.
- Em 1962 os matemáticos russos G.M. **A**delson-**V**elskki e E.M.**L**andis sugeriram uma definição para "near balance" e descreveram procedimentos para inserção e eliminação de nós nessas árvores.
- Estes algoritmos de balanceamento de árvore são chamados algoritmos AVL.

3

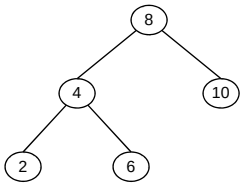
Árvore AVL

- Quando um novo nó é inserido em uma árvore balanceada podem ocorrer três situações, em função da configuração atual da árvore:
  - se h(sae) = h(sad), após a inserção as alturas vão ser diferentes, porém a árvore continua balanceada
  - se h(sae) > h(sad) e o nó for inserido na sad, as alturas ficam iguais e o balanceamento foi melhorado.
  - se h(sae) > h(sad) e o nó for inserido na sae, o balanceamento fica violado.

4

Árvore AVL

- Os nós 9 e 11 podem ser inseridos sem desbalancear a árvore.
- Os nós 1, 3, 5 e 7 geram desbalanceamento.



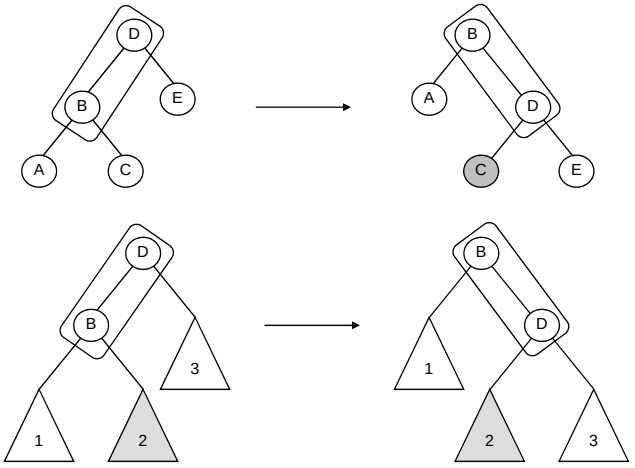
5

Rotações

- Existem 4 tipos de rotações para rebalancear uma árvore AVL
  - Rotação à Esquerda
  - Rotação à Direita
  - Dupla rotação a Esquerda (Esquerda + direita)
  - Dupla rotação a Direita (Direita + Esquerda)
- Com apenas ***uma*** rotação, após a inserção de um novo item, a árvore volta a ser balanceada.

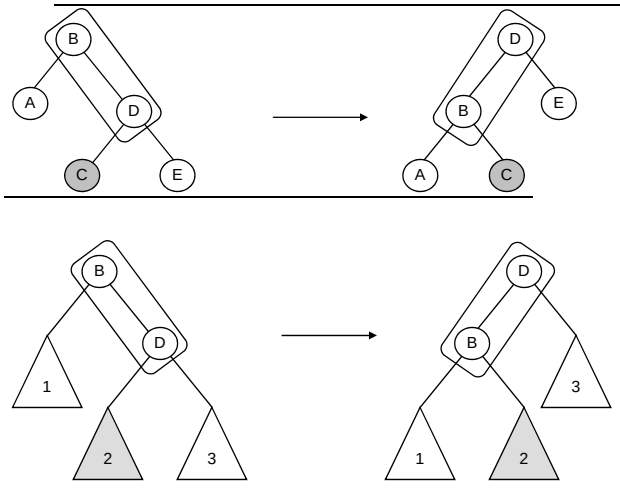
6

1 - Rotação Direita



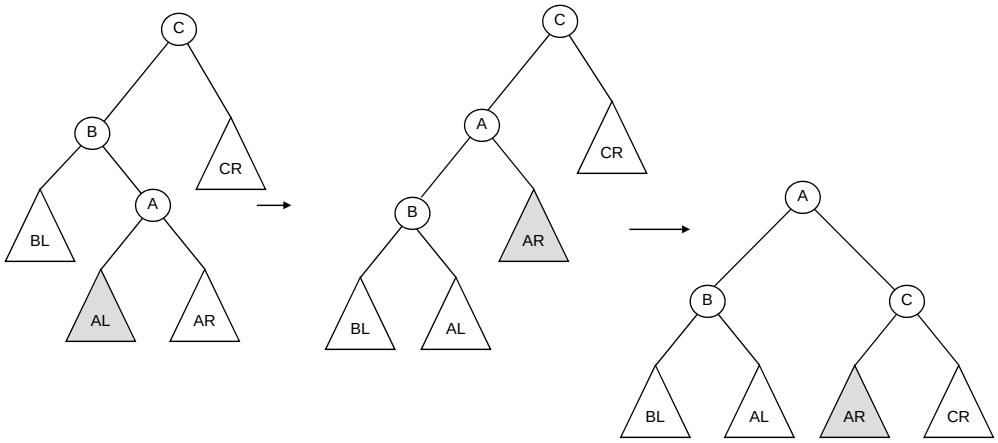
7

2 - Rotação Esquerda



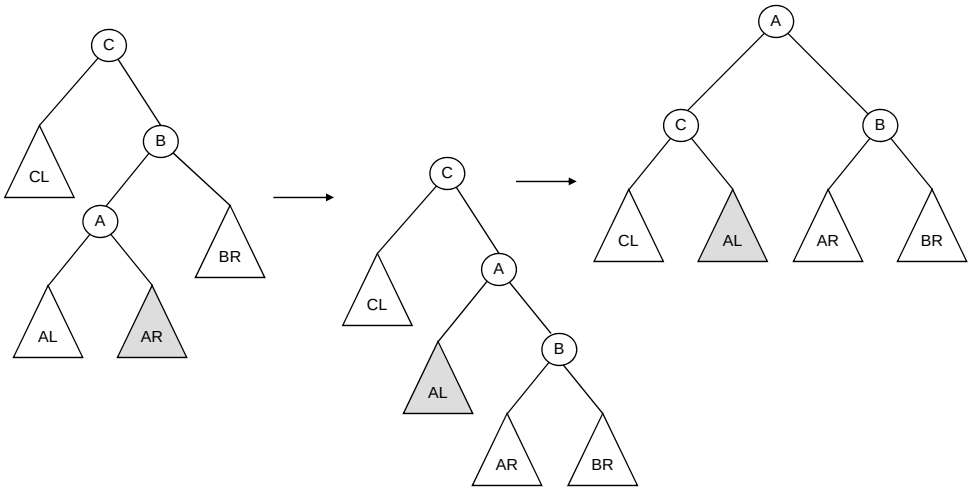
8

3 - Rotação Esquerda-Direita



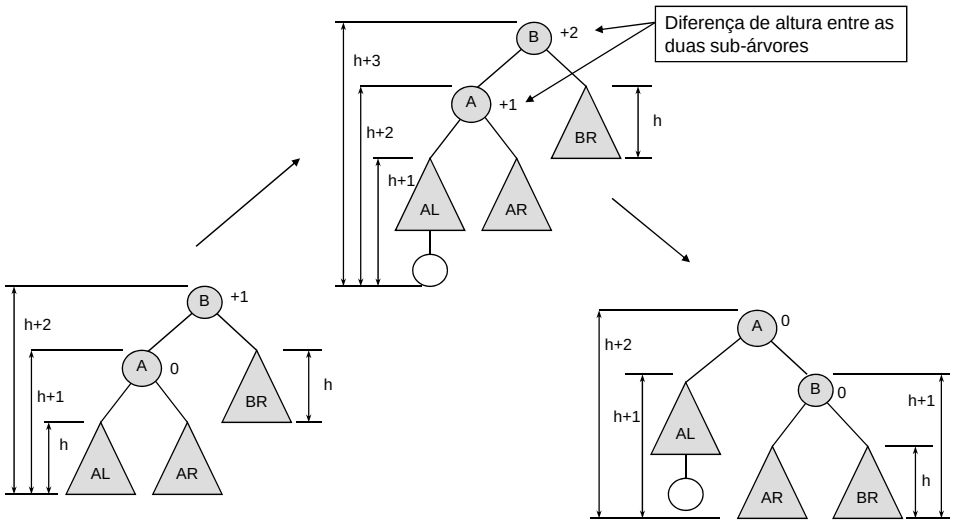
9

4 - Rotação Direita-Esquerda



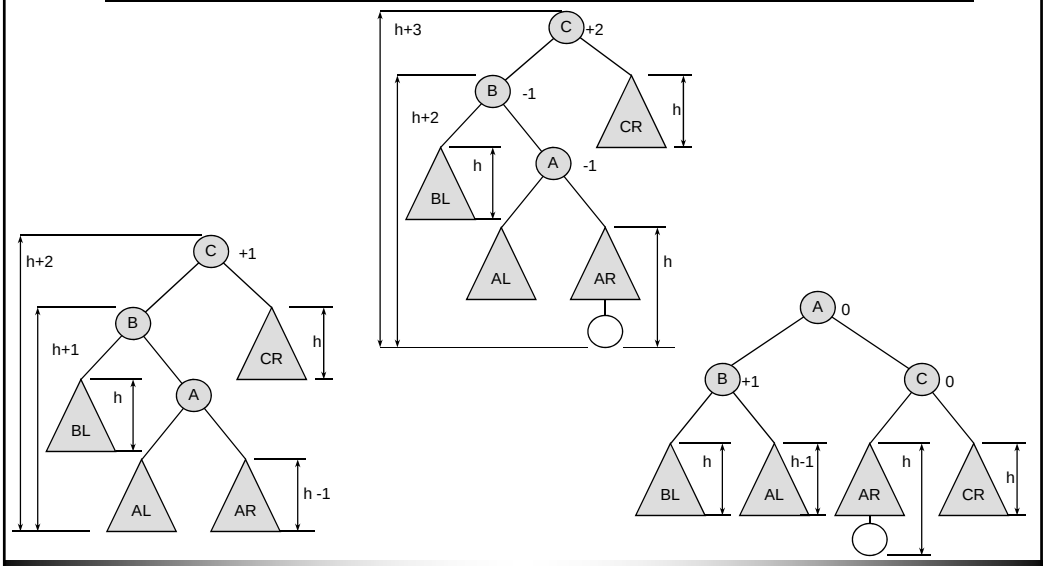
10

Adição de um nó e rotação Direita



11

Adição de um nó e rotação Dupla



12

Implementação

- Cada nó da árvore guarda a sua altura em relação à raiz;
- Usa cálculo de diferença de altura entre cada sub-árvore;
- Se a diferença de altura entre cada sub-árvore for igual a 2, aplica-se uma rotação: `no->alt = no->esq->alt – no->dir->alt`;
- Dado um nó N, a escolha da rotação é dada em função:
  - Do ramo que o nó foi inserido
  - Do valor inserido em relação ao valor armazenado no nó

```
typedef struct avl
{
   int          info;     //informação do nó
   int          altura;   //distância até a folha
   struct avl  *esq;
   struct avl  *dir;
}ArvAvl;
```

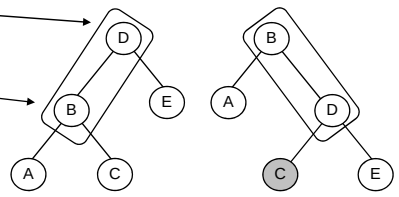

Rotação à Direita

```
ArvAvl * rotacaoDireita( ArvAvl * K2 )
{
   ArvAvl * K1;

   K1 = K2->esq;
   K2->esq = K1->dir;
   K1->dir = K2;

   K2->altura = maior( altura( K2->esq ), altura( K2->dir ) ) + 1;
   K1->altura = maior( altura( K1->esq ), K2->altura ) + 1;

   return K1;  //nova raiz
}
```



A nova altura deve ser a maior entre os ramos C,E (que não modificaram) + o nó

Rotação à Esquerda

```
ArvAvl * rotacaoEsquerda( ArvAvl * K1 )
{
   ArvAvl * K2;

   K2 = K1->dir;
   K1->dir = K2->esq;
   K2->esq = K1;

   K1->altura = maior( altura( K1->esq ), altura( K1->dir ) ) + 1;
   K2->altura = maior( altura( K2->dir ), K1->altura ) + 1;

   return K2;  //nova raiz
}
```

15

Rotação à EsquerdaDireita

```
ArvAvl * rotacaoEsquerdaDireita( ArvAvl * K3 )
{
   K3->esq = rotacaoEsquerda( K3->esq );
   return rotacaoDireita( K3 );
}
```

16

Rotação à DireitaEsquerda

```
ArvAvl * rotacaoDireitaEsquerda( ArvAvl * K1 )
{
   K1->dir = rotacaoDireita( K1->dir );
   return rotacaoEsquerda( K1 );
}
```

Acesse: http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

Aplet Java que ilustra graficamente as operações sobre árvores AVL.

17

Função de Inserção

```
ArvAvl * insere( int info, ArvAvl * arv )
{
   if( arv == NULL )
   {
      arv = aloca(info);
   }
   else  if( info < arv->info )
   {
      arv->esq = insere( info, arv->esq );
      if( altura( arv->esq ) - altura( arv->dir ) == 2 )
      {
         if( info < arv->esq->info )
            arv = rotacaoDireita( arv );
         else
            arv = rotacaoEsquerdaDireita( arv );
      }
   }
   else if( info > arv->info )
   {
      arv->dir = insere( info, arv->dir );
      if( altura( arv->dir ) - altura( arv->esq ) == 2 )
      {
         if( info > arv->dir->info )
            arv = rotacaoEsquerda( arv );
         else
            arv = rotacaoDireitaEsquerda( arv );
      }
   }
   arv->altura = maior( altura( arv->esq ), altura( arv->dir ) ) + 1;
   return arv;
}
```

```
ArvAvl * aloca(int info)
{
   ArvAvl *arv
   arv = (ArvAvl*)malloc(sizeof(ArvAvl));
   arv->info   = info;
   arv->altura = 0;
   arv->esq = arv->dir = NULL;
   return arv;
}
```

```
int altura (ArvAvl *a)
{
   if( a == NULL)
      return -1;
   return a->altura;
}
```

OBS: função muito semelhante com função de inserção de nós em árvore binária.

18

Exercícios

1. Reimplemente as funções de manipulação de árvores AVL considerando que cada nó da árvore, em vez de guardar a altura, armazene apenas a diferença de altura entre cada sub-árvore (Fator de balanceamento - FB). A diferença de altura pode ser positiva ou negativa.

```
if( arv->FB > 1) //tem maior altura na esquerda
{
   if(arv->esq->FB > 0) //tem maior altura na esquerda/esquerda
      rotacaoDireita();
   else                 //tem maior altura na esquerda/direita
      rotacaoEsquerdaDireita();
}
if( arv->FB < -1) //tem maior altura na direita
{
   if(arv->dir->FB < 0)
      rotacaoEsquerda();
   else
      rotacaoDireitaEsquerda ();
}
```

19

Exercícios

2. Gere desenhos das árvores AVL, passo a passo, com diferenças de altura para cada nó, para os seguintes conjuntos de dados. Justifique, por meio dos algoritmos apresentados, o por quê das rotações aplicadas.
   1. 5, 6, 4, 3, 2, 1
   2. 5, 1, 7, 2, 3, 4, 0
   3. 1, 2, 3, 4, 5, 6
   4. 6, 5, 4, 3, 2, 1
   5. 8, 4, 6, 1, 9, 10, 11
   6. 8, 14, 11, 5, 6, 3

20

Referências

//codigo fonte em C
http://www.cs.fiu.edu/~weiss/dsaa_c2e/files.html

http://www.cmcrossroads.com/bradapp/ftp/src/libs/C++/AvlTrees.html

http://en.wikipedia.org/wiki/AVL_tree

http://www.engcomp.ufrn.br/~diego/ed2_arvores_avl.pdf

http://www.icmc.usp.br/~sce182/arvbinrb.html

Preiss, B. R. Estruturas de Dados e Algoritmos – Padrões de projetos orientados a objetos com Java. Ed. Campus, 2001.