Árbol AVL

Este artículo o sección necesita referencias que aparezcan en una publicación acreditada.
Este aviso fue puesto el 8 de noviembre de 2010.

Un **árbol AVL** es un tipo especial de árbol binario ideado por los matemáticos rusos **Adelson-Velski** y **Landis**. Fue el primer árbol de búsqueda binario auto-balanceable que se dio.

Animación que muestra la construcción de un árbol AVL a partir de todos los números.

Índice [oculta]

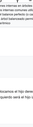
1 Descripción [editar]
2 Definición formal [editar]
2.1 Definición de la altura de un árbol
2.2 Definición de árbol AVL
3 Factor de equilibrio
4 Operaciones [editar]
4.1 Rotaciones
4.2 Inserción
4.3 Extracción
4.4 Búsqueda
5 Véase también
6 Enlaces externos

Descripción [editar]

El árbol AVL toma su nombre de las iniciales de los apellidos de sus inventores, Georgii Adelson-Velski y Yevgeniy Landis. Lo diseñaron y construyeron en la publicación de un artículo en 1962, dando un algoritmo for-ma organizativo of information. Los algoritmos para la organización de la información.

Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrar el balance, la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de alturas de los subárboles.

El ejemplo de árbol binario no equilibrado (no es AVL)



El ejemplo de árbol binario equilibrado (es AVL)

Una propiedad interesante de árboles binarios es que, mediante una simple operación llamada rotación, se puede mantener el balance perfecto (o casi perfecto) de árbol. Las rotaciones preservan también operaciones en tiempo logarítmico.

Definición formal [editar]

Definición de la altura de un árbol [editar]

Sea T un árbol binario de búsqueda y sean T_L y T_R sus subárboles, su altura $H(T)$ es:

- 0 si el árbol T contiene sólo la raíz
- $1 + \max(H(T_L), H(T_R))$ si contiene más nodos

Definición de árbol AVL [editar]

- Un árbol vacío es un árbol AVL
- Si T no es árbol vacío y T_L y T_R sus subárboles, entonces T es AVL si y solo si:
  - T_L es AVL
  - T_R es AVL
  - $|H(T_L) - H(T_R)| <= 1$

Factor de equilibrio [editar]

Cada nodo, además de la información que se pretende almacenar, debe tener los dos punteros a los árboles derecha e izquierdo, igual que que los árboles binario de búsqueda (ABB), y además el dato que contiene el factor de equilibrio:

El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo:

FE = altura subárbol derecho - altura subárbol izquierdo

Por definición, para un árbol AVL, este valor debe ser: -1, 0 ó 1.

Si el factor de equilibrio de un nodo es:

- 0 → el nodo está equilibrado y sus subárboles tienen exactamente la misma altura.
- 1 → el nodo está equilibrado y su subárbol derecho es un nivel más alto.
- −1 → el nodo está equilibrado y su subárbol izquierdo es un nivel más alto.

Si el factor de equilibrio $|FE| >= 2$ es necesario reequilibrar.

Operaciones [editar]

Las operaciones básicas de un árbol AVL implican generalmente el realizar los mismos algoritmos que varían realizadas en un árbol binario de búsqueda (ABB), y además el dato que contiene el factor de equilibrio:

Rotaciones [editar]

Las rotaciones se producen de abajo hacia arriba sobre los nodos en los que se produce el desequilibrio. Pueden darse dos casos: rotación simple o rotación doble, y a su vez estos casos pueden ser hacia la derecha o hacia la izquierda:

Las rotaciones internas en árboles binarios son operaciones internas (unicas) utilizadas para mantener el balance perfecto (o casi perfecto) del árbol. Las rotaciones preservan también operaciones en tiempo logarítmico.

Rotación simple a la derecha

Un árbol de nodo raíz (A) de hijo izquierdo (B) y derecho (C), la que hacemos será formar un nuevo árbol cuya raíz sea la raíz del hijo izquierdo, como hijo derecho colocaremos el hijo derecho de B (nuestro A) y como hijo izquierdo colocaremos el hijo izquierdo de B (nuestro C) y como hijo derecho constituimos un nuevo árbol que tendrá como raíz, la raíz del árbol A, el hijo izquierdo del C (el anterior hijo izquierdo) y el hijo derecho será el hijo derecho del árbol (A).

```
up rotRer: AVL(C) -> AVL(C) -
my rotRerIzq=BinaRer(C, arbolBin(B, T2, D2), D1) =>
arbolBin(B, T2, arbolBin(A, T2, D2))
```

Rotación simple a la izquierda

Un árbol de nodo raíz (A) de hijo derecho (C) y izquierdo (B), consiste en formar un nuevo árbol cuya raíz sea la raíz del hijo derecho, como hijo derecho colocaremos el hijo derecho de C (nuestro A) y como hijo izquierdo constituimos un nuevo árbol que tendrá como raíz la raíz del árbol (C), el hijo izquierdo de A será el hijo derecho C (la raíz) y el hijo izquierdo será el hijo izquierdo del árbol (B).

Precisación: Tiene que tener hijo derecho no vacío.

```
up rotIzq: AVL(C) -> AVL(C) -
my rotIzqIzqBin=BinaIzq(C, arbolBin(A, T2, D2)) =>
arbolBin(B, arbolBin(A, T1, D1), D2) -
```

Si la inserción nos produce en el hijo derecho del hijo izquierdo del nodo desequilibrado (o viceversa) hay que realizar una doble rotación.

Rotación doble a la derecha

La Rotación doble a la Derecha son dos rotaciones simples, primero rotación simple izquierda y luego rotación simple derecha.

Rotación doble a la izquierda

La Rotación doble a la Izquierda son dos rotaciones simples, primero rotación simple derecha y luego rotación simple izquierda.

Inserción [editar]

La inserción en un árbol de AVL puede hacer que el árbol quede desbalanceado y se vea obligado a realizar una rotación para que el árbol de búsqueda binario desequilibrado y después retrocederá hacia la raíz, volverá sobre cualquier nodo que pudiera haber desequilibrado durante la inserción.

Proceso de inserción:

1 Buscar hasta encontrar la posición de inserción y modificación (proceso idéntico a inserción en árbol binario de búsqueda)
2 Insertar el nuevo nodo con factor de equilibrio "equilibrado"
3 Desarrollar el camino de búsqueda, verificando el equilibrio de los nodos, y re-equilibrando si es necesario

Debido a que las rotaciones son una operación que tiene complejidad constante y a que la altura está limitada a O (log(n)), el tiempo de ejecución para la inserción en del orden O (log(n)).

```
mp inserter: RBT(X AVL(X) -> AVL(X)) -
mp insertarIzq=BinaIzq(A, arbolBin(B, T1, T2), C) =>
cmp insertarDer(A, arbolBin(B, T1, T2), C) =>
if AVL=AC2 then
insertarIzq(A, T1, B)
elseif AVL=AC1 then
if (altura(insertarIzq(A,T1)) - altura(B)) -> 2 then
if (altura(insertarIzq(A,T1), T1, B) -
else =>rRep que rempolizar
if (B1 -> rpl(T1) then
rotarDerIzq(arbolBin(A, insertarIzq, T1), B)
else
rotarDerIzq(arbolBin(A, rotarIzq(insertarIzq, T1), D1)
fi -
fi -
else -
if (altura(insertarIzq(A,T1) - altura(B) -> 2) then
arbolBin(A, insertarIzq, T1, B)
else =>> hay que rempolizar
if (B1 -> rpl(T1) then
rotarIzq(arbolBin(A, I, insertarIzq(T1, B)))
else
rotarIzq(arbolBin(A, I, rotarDer(insertarIzq(T1, B))))
fi -
fi -
fi -
fi -
```

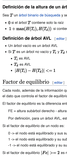
Extracción [editar]

El procedimiento de extracción es el mismo que en el caso de árbol binario de búsqueda. La diferencia se encuentra en el proceso de reequilibrado posterior. El problema de la extracción puede afectar a todos los nodos del camino de la raíz a la hoja. Es necesario llevar una determinación de la altura de la rama donde se extrajo y tendrá como efecto un cambio en el factor de equilibrio del nodo padre de la rama en cuestión, pudiendo necesitarse una rotación.

Esta disminución de la altura y la corrección de los factores de equilibrio con sus posibles rotaciones asociadas pueden propagarse hasta la raíz.

Borrar A, y la nueva raíz será M.

Borrado A, la nueva raíz es M. Aplicamos la rotación a la izquierda.



El árbol resultante ha perdido altura.

En el borrado puedan ser necesarias varias operaciones de restauración del equilibrio, y hay que seguir comprobando hasta llegar a la raíz.

```
mp eliminar: RBT(X AVL(X) -> AVL(X) -
mp eliminar(A, (raiz) -> crear -
cmp eliminarIzq(A, arbolBin(B(X), T2, D) =>
if (B(X) -> raiz(X) then
if esHoja(B) then
raiz(A)
elseif esHoja(B) then
raiz(A)
else
if (B(X) ((>>) ((<<)
if (altura(B) - altura(A)) -> 2) then
arbolBin(arbolBin(A, I, eliminarIzq(B, D)) -
elseif (altura(A) - altura(B) -> 2) then
elseif (altura(A)(altura(B) -> eliminarIzq(B(X)) then
rotarDerIzq(arbolBin(A, I, eliminarIzq(B,D))
fi -
rotRerIzq(arbolBin(A, rotIzq(I, eliminarIzq(B,D))
```

Búsqueda [editar]

Ejemplo en Java

```
public class Node {
int[] keys;
Node[] kids;
public Node(int nsz0)
{
keys = new int[nsz0];
kids = new Node[nsz0+1];
}
public void rp_expand(){
if(keys != null)
{keys = rp_expand(keys);
kids = rp_expand(kids);}
else keys = new int[1];
if(kids == null)
kids = new Node[1];
}
public int[] rp_expand(int[] nuevaRP)
{
int[] temp = new int[keys.length];
for(int i=0; i<keys.length; i++)
temp[i] = keys[i];
return nuevaRP;
}
public void insertar(Node rx, int nx){
if (rx == null)
System.out.println("Error");
else if(nx == keys[0])
insertar(rx.kids[0], nx);
else if(nx < keys[0])
insertar(rx.kids[0], nx);
else
System.out.println("Número Duplicado");
}
public void visualizar(){
System.out.print("[");
rp_visualizar();
}
}
public class BinaryTree {
public static void main(String[] largs){
AVL Arbol = new AVL();
Arbol.insertar(5);
Arbol.insertar(2);
Arbol.insertar(7);
Arbol.visualizar();
}
}
```

Ejemplo de TAD AVL en C

```
#include <stdio.h>

typedef struct AVL{
int altura; /* FB es la altura del subarbol izquierdo menos la altura del subarbol derecho
AVL *izq, *der;
bool borrado;
} AVL;

void rotarIzq(AVL-&A){ //precond: el árbol necesita una rotacion LL
AVL *aux = A->der;
A->der = aux->izq;
A->izq->FB = 0;
aux->izq = A;
A->izq = aux;
A->FB = 0;
A = aux;
}

void rotarDer(AVL-&A){ //precond: el árbol necesita una rotacion RR
AVL* aux = A->der->izq;
A->der->izq = A;
A->der->der->FB = 0;
AVL* auxI = A->der;
A->der = aux;
A = auxI;
A = auxI;
}

void rotarDerIzq(AVL-&A){ //precond: el árbol necesita una rotacion LR
rotarDer(A->izq);
rotarIzq(A);
}

void rotarRerIzq(AVL-&A){ //precond: el árbol necesita una rotacion RL
rotarIzq(A->der);
rotarDer(A);
}

AVL* Crear(){
return NULL;
}

void Insertar(int e, bool &ausento, AVL-&A){
if (A == NULL)
A = new AVL;
A->der = NULL;
A->FB = 0;
A->izq = NULL;
A->borrado = false;
ausento = true;
else if (e > A->data){
Insertar(e, ausento, A->der);
if (ausento){
switch (A->FB){
case -1:
A->FB = 0;
ausento = false;
break;
case 0:
A->FB = 1;
ausento = true;
break;
case 1:
if (A->der->FB == 1){ // Si es 1 necesito una rotacion LL si es -1 necesito una rotacion LR
rotarIzq(A);
}else{
rotarDerIzq(A);
}
ausento = false;
break;
}
}
}else{
if (e < A->data){
Insertar(e, ausento, A->der);
if (ausento){
switch (A->FB){
case 1:
A->FB = 0;
ausento = false;
break;
case 0:
A->FB = -1;
ausento = true;
break;
case -1:
if (A->der->FB == -1){ // Si es 1 necesito una rotacion RR si es -1 necesito una rotacion RL
rotarDer(A);
}else{
rotarRerIzq(A);
}
ausento = false;
break;
}
}
}
}
}

void Insertar(AVL-&A, int e){
bool ausento;
Insertr(e, ausento, A);
}

bool Esbalanceado(AVL-&A){
return A == NULL;
}

bool Pertenece(AVL-&, A, int e){
if (A == NULL)
return false;
else{
if (A->data == e){
if (A->borrado){
return false;
}else{
return true;
}
}else if (e > A->data){
return Pertenece(A->izq, e);
}else{
return Pertenece(A->der, e);
}
}
}
}

void Borrar(AVL-&A, int e){
if (A->data == e){
A->borrado = true;
}else if (e > A->data){
Borrar(A->izq, e);
}else{
Borrar(A->der, e);
}
}
```

Véase también [editar]

- Arbol binario de búsqueda
- Arbol (programación)
- Arbol B
- Arbol rojo-negro
- Arbol splay
- Arbol multiarbol

Enlaces externos [editar]

- Más información sobre árboles AVL.@
- Applet que muestra el comportamiento de diferentes estructuras de datos arbórea@

Control de autoridades: **Proyecto Wikimedia:** **BD Data:** Q900180 **Wikimedia:** AVL-tree

Categoría: Árboles (estructuras)

Esta página se editó por última vez el 16 abr 2020 a las 23:14.
Contenido bajo licencia de la Licencia Creative Commons Atribución Compartir Igual 3.0; pueden aplicarse cláusulas adicionales. Al usar este sitio, usted acepta nuestros términos de uso y la nuestra política de privacidad.
WikiTeam®es una marca registrada de la Fundación Wikimedia, Inc., una organización sin ánimo de lucro.
Política de privacidad Acerca de Wikipedia Limitación de responsabilidad Versión para móviles Desarrolladores Estadísticas Declaración de cookies
Fundación Wikimedia Powered by MediaWiki