Universidad Tecnológica de la Mixteca

Estructuras de datos

Tema: Árboles Balanceados

Integrantes

López Santiago Jorge Alberto

Marysol Mejía Cervantes

Grupo 201-F.

Semestre Marzo/2003- Julio/2003

Árboles balanceados

Es un árbol binario de búsqueda que tiene como objetivo realizar reacomodos o balanceos después de inserciones o eliminaciones de elementos.

Clasificación.

- Árboles AVL (o balanceados por altura). ⟶
  En este tipo de árboles las alturas de los dos subárboles asociados con cada elemento no pueden diferir en más de uno, y los dos subárboles deben ser también AVL. Por definición un árbol binario vacío es AVL.
- Árboles perfectamente balanceados.
  El numero de elementos en cada uno los subárboles asociados no pueden diferir en mas de uno, y los dos subárboles asociados deben ser también perfectamente balanceados. Por definición un árbol binario vacío es perfectamente balanceado.

Eficiencia de los árboles balanceados.

- Se mantiene el árbol ordenado.
- En los algoritmos de inserción y eliminación es posible buscar, insertar y eliminar un elemento en un árbol balanceado de n nodos en O(log n) unidades de tiempo.
- Se realizan más rotaciones en las operaciones de inserción que en las de eliminación.
- Garantizan que la operación de búsqueda de un elemento tiene complejidad $O(\log_2 n)$ a costa de algoritmos mas complicados para implementar las modificadoras, puesto que deben alterar en cada inserción y supresión la estructura del árbol.

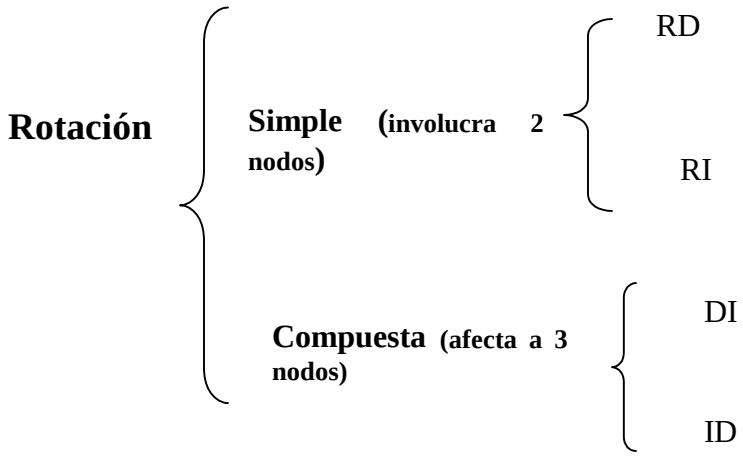
Reestructuración del árbol balanceado

Reestructurar el árbol significa rotar los nodos del mismo. Para que la rotación se efectué se requiere de un factor de equilibrio (FE o Balance); el cual se define como ¨ la diferencia entre las alturas del árbol izquierdo y el derecho ¨:
FE = altura subárbol izquierdo - altura subárbol derecho. Para un árbol AVL, estos valores deben ser -1, 0 ó 1.

- La altura de un árbol, es la longitud mas larga de un árbol que parte de la raíz +1.

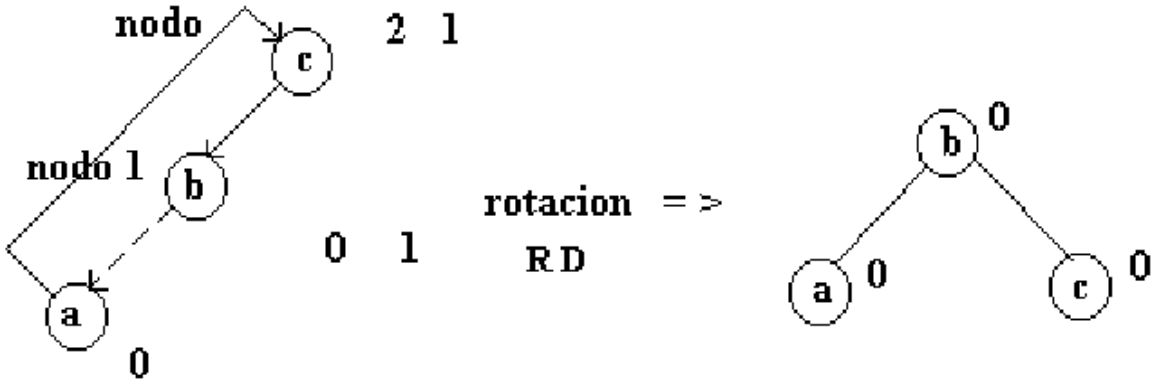
- Balance es =0 si el subárbol der e izq, tienen como altura uno.
- Balance es =-1 si el subárbol der. Excede en uno a la altura del izq.
- Balance es =1 si el subárbol izq. Excede de uno al árbol der.

Clasificación de las rotaciones



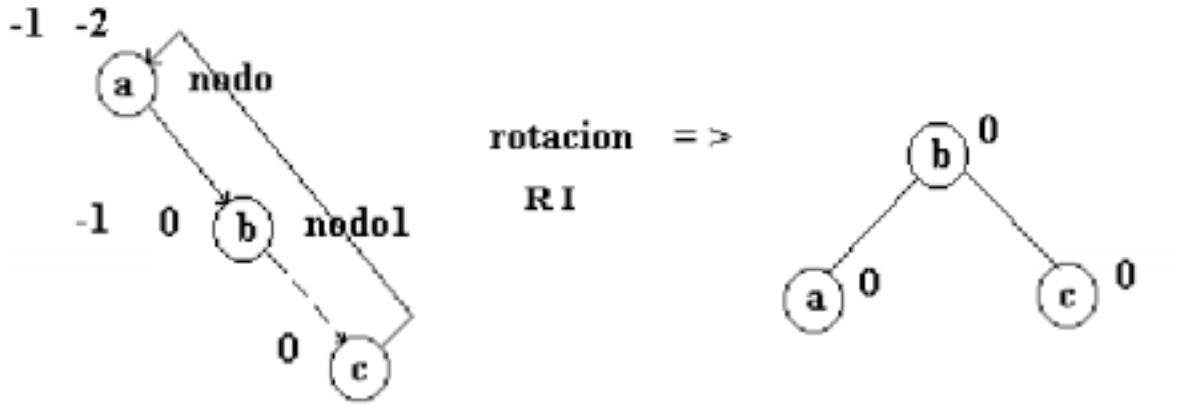
Rotación simple a la derecha (RD)

El subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho cuando su FE sea de 2. Y además, la raíz del subárbol izquierdo tenga una FE de 1, es decir, que esté cargado a la izquierda.



Rotación simple a la izquierda (RI)

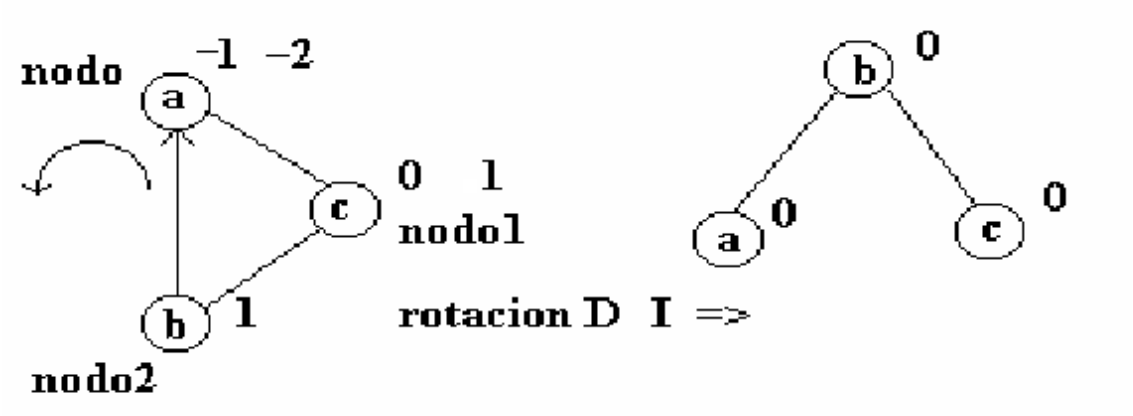
El subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su FE sea de -2. Y además, la raíz del subárbol derecho tenga una FE de -1, es decir, que esté cargado a la derecha.



Compuesta

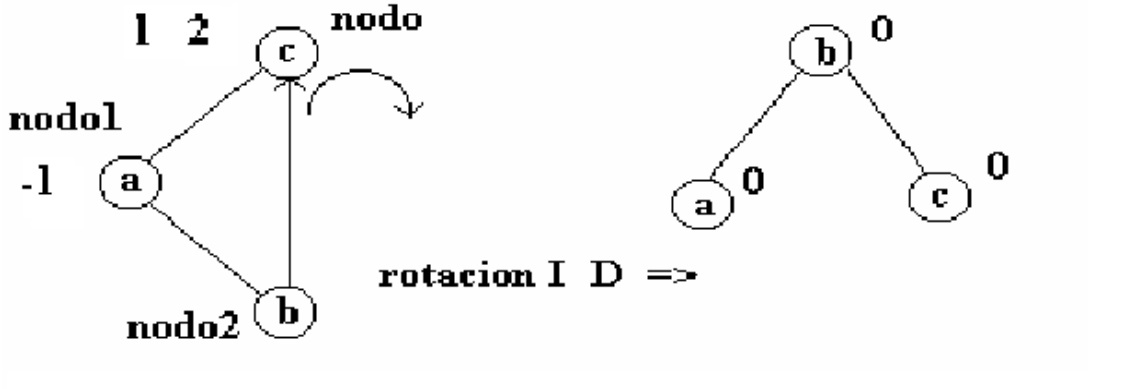
Rotación derecha Izquierda (DI)

Cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su FE sea de -2. Y además, la raíz del subárbol izquierdo tenga una FE de 1, es decir, que esté cargado a la derecha.



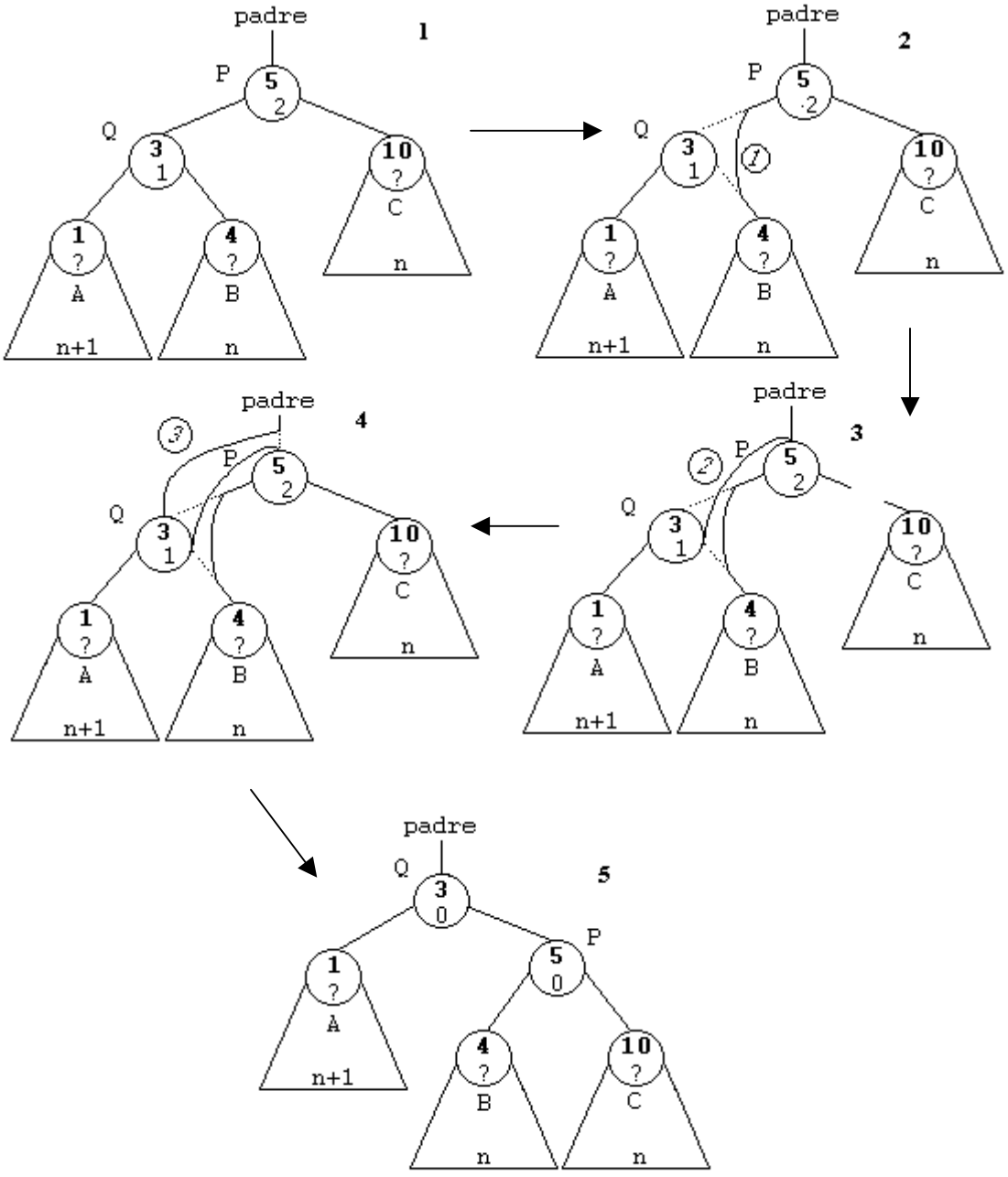
Rotación Izquierda Derecha (ID)

Cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su FE sea de 2. Y además, la raíz del subárbol derecho tenga una FE de -1, es decir, que esté cargado a la izquierda.

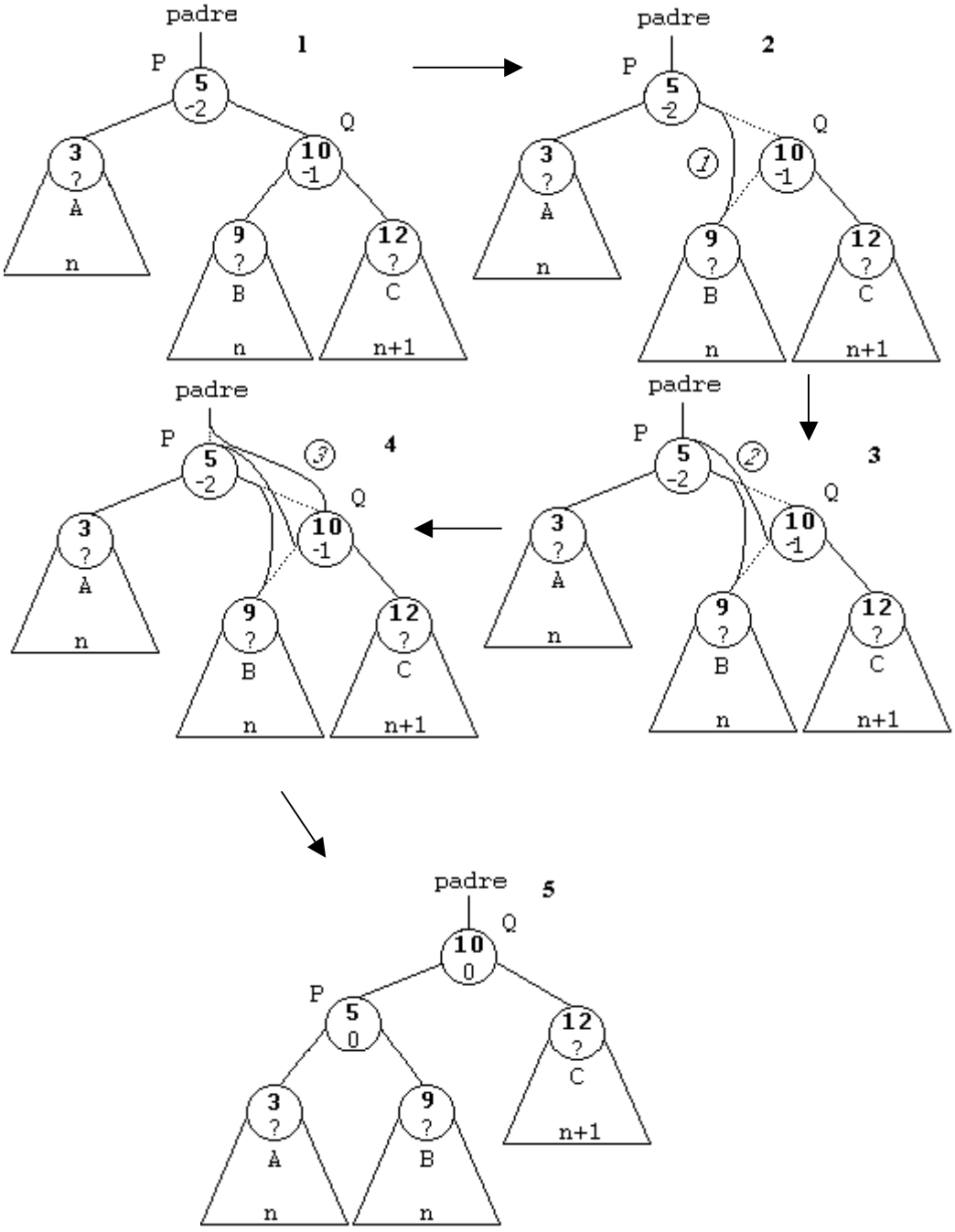


Ejemplos de Rotaciones simples de nodos

Rotación simple a la derecha (RD)

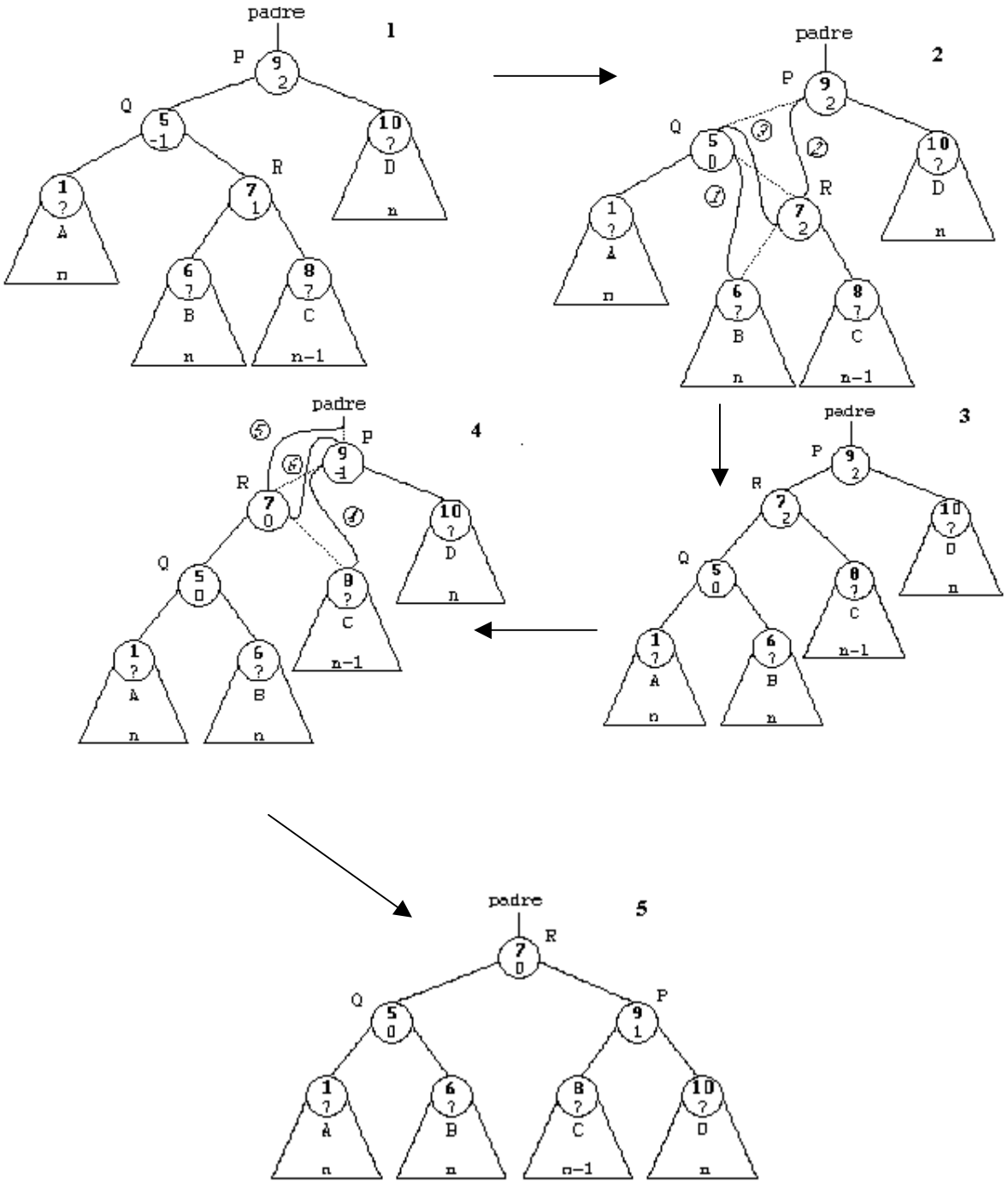


Rotación simple a la izquierda (RI)

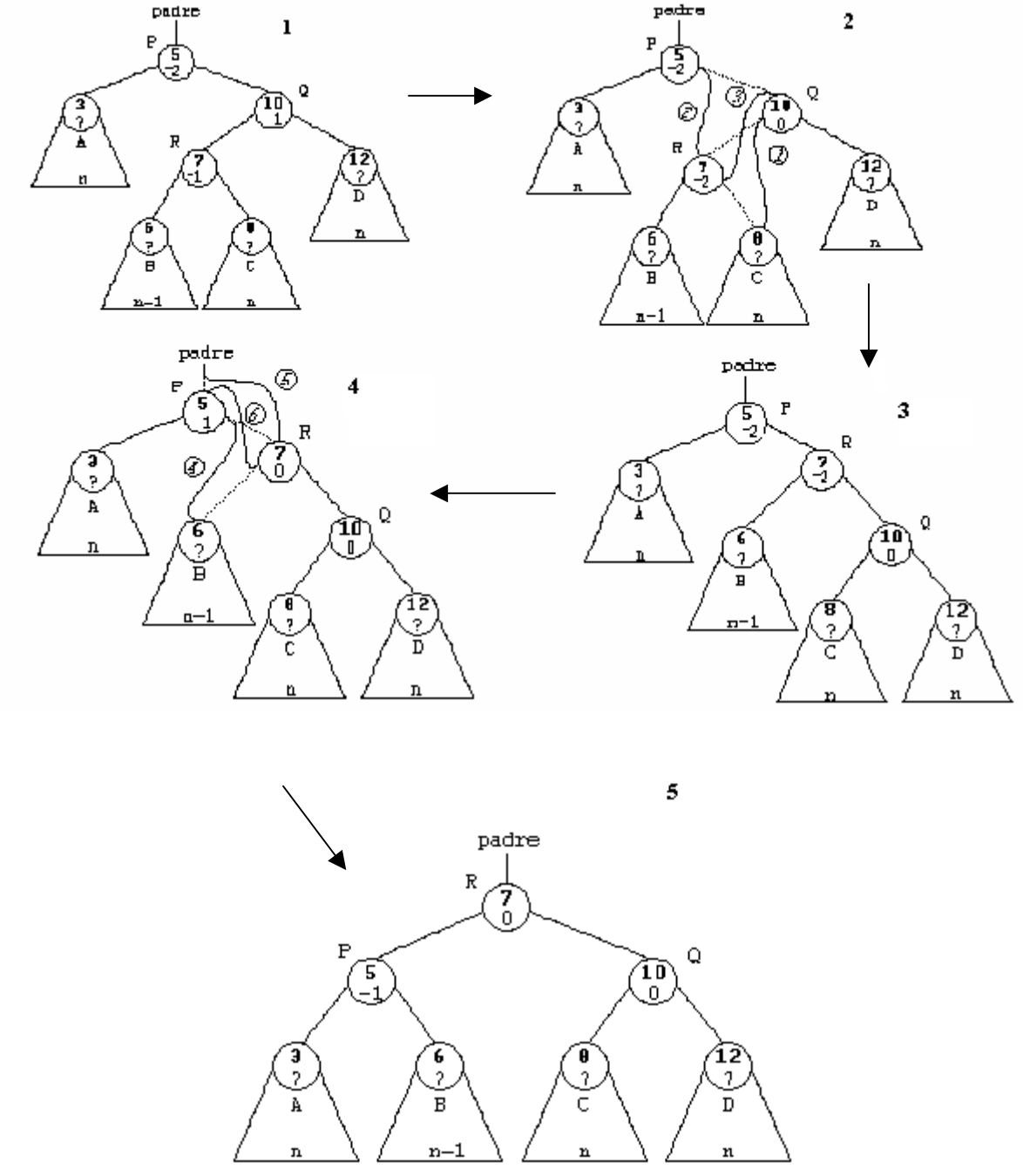


Ejemplo de Rotaciones Compuesta

Rotación derecha izquierda (DI)



Rotación izquierda derecha (ID)



Inserción en árboles balanceado

Casos para insertar un elemento en un árbol balanceado.

1. Las ramas izquierda (RI) y derecha (RD) tienen la misma altura ($H_{RI}=H_{RD}$), por lo tanto:
   - Si se inserta un elemento en RI entonces H_{RI} será mayor a $H_{RD.}$
   - Si se inserta un elemento en RD entonces H_{RD} será mayor aH_{RI}.

2. las ramas izquierda (RI) y derecha (RD) del árbol tienen altura diferente ($H_{RI}!=H_{RD}$):
   - H_{RI} menor a $H_{RD:}$

     ---Si se inserta un elemento en RI entonces $H_{RI}=H_{RD.}$

     (Las ramas tienen la misma altura, por lo que se mejora el equilibrio del árbol).

     ---Si se inserta un elemento en RD entonces se rompe el criterio de equilibrio del árbol y es necesario reestructurarlo.
   - H_{RI} mayor a $H_{RD:}$

     ---Si se inserta un elemento en RI entonces se rompe el criterio de equilibrio del árbol y es necesario reestructurarlo.

     ---Si se inserta un elemento en RD entonces $H_{RI}=H_{RD.}$

     (Las ramas tienen la misma altura, por lo que se mejora el equilibrio del árbol).

Borrado en árboles Balanceados

- Si el elemento a borrar es terminal u hoja, simplemente se suprime.
- Si el elemento a borrar tiene un solo descendiente, entonces tiene que sustituirse por ese descendiente.
- Si el elemento a borrar tiene dos descendientes, entonces tiene que sustituirse por el nodo que se encuentra más a la izquierda en el subárbol izquierdo o por el nodo que se encuentra más a la derecha en el subárbol izquierdo.

Código fuente

```
AVL RotaIzq(AVL a)  //Funcion rota izquierda
{ AVL  t=a->der;
  a->der=t->izq;
  t->izq=a;
  return t;   // retorna t como raiz
}

AVL RotaDer(AVL a) //Funcion rota derecha
{ AVL  t=a->izq;
  a->izq=t->der;
  t->der=a;
  return t;     // retorna a t como raiz
}

AVL RotaDerIzq(AVL a) //Funcion rota derecha izquierda
{ a->der=RotaDer(a->der);
  return RotaIzq(a);  //retorna la raiz
}

AVL RotaIzqDer(AVL a) //Funcion rota izquierda derecha
{ a->izq=RotaIzq(a->izq);
  return RotaDer(a);
}

int Altura(AVL a) //Funcion que calcula la altura de un arbol
{    int aizq,ader;
     if(a==NULL)
              return 0;
     if(a->izq==NULL&&a->der==NULL)
              return 1;
     aizq=Altura(a->izq);
     ader=Altura(a->der);
     if(aizq>ader)
              return 1+aizq;
     else
              return 1+ader;
}

int Balance(AVL a) //Funcion que obtiene el FE
{   return Altura(a->izq)-Altura(a->der); }
```

```
int InsertaAVL(AVL *a,Tinfo x)
{
    if(*a==NULL)
    {       *a=(AVL)malloc(sizeof(TAVL));
            if(*a==NULL)
                    return -1;//memoria insuficiente
            (*a)->info=x;
            (*a)->izq=(*a)->der=NULL;
            (*a)->bal=Balance(*a);
            return 1;//si se inserto
    }
    else
    {       if(x<ArbolRaiz(*a))
            {       InsertaAVL(&((*a)->izq),x); //llamada recursiva
                    if(Balance(*a)==2&&Balance((*a)->izq)==1)//rotacion RD mediante el FE
                            *a=RotaDer(*a);
                    else
                    if(Balance(*a)==-2&&Balance((*a)->der)==-1)//rotacio RI mediante el FE
                            *a=RotaIzq(*a);
                    else
                    if(Balance(*a)==2&&Balance((*a)->izq)==-1)//rotacion ID mediante el FE
                            *a=RotaIzqDer(*a);
                    else
                    if(Balance(*a)==-2&&Balance((*a)->der)==1)//rotacion DI mediante el FE
                            *a=RotaDerIzq(*a);
                    return 1;
            }
            else
               if(x>ArbolRaiz(*a))
                 {  InsertaAVL(&((*a)->der),x);
                    if(Balance(*a)==2&&Balance((*a)->izq)==1) //rotacion RD mediante el FE
                            *a=RotaDer(*a);
                    else
                    if(Balance(*a)==-2&&Balance((*a)->der)==-1)//rotacion RI mediante el FE
                            *a=RotaIzq(*a);
                    else
                    if(Balance(*a)==2&&Balance((*a)->izq)==-1) //rotacion ID mediante el FE
                            *a=RotaIzqDer(*a);
                    else
                    if(Balance(*a)==-2&&Balance((*a)->der)==1) //rotacion DI mediante el FE
                            *a=RotaDerIzq(*a);
                            return 1;
                    }
                    else
                            return -2;//elemento repetido
    }
}
```

```
int EliminaAVL(AVL *a,Tinfo x) //Funcion elimina
{  AVL aux,aux1,otro;
  if(*a==NULL)
    return 0;

  else
    {
     if(x<ArbolRaiz(*a))
       {EliminaAVL(&((*a)->izq),x);
        if(Balance(*a)==2&&Balance((*a)->izq)==1)
           *a=RotaDer(*a);
        else
           if(Balance(*a)==-2&&Balance((*a)->der)==-1)
              *a=RotaIzq(*a);
           else
                if(Balance(*a)==2&&Balance((*a)->izq)==-1)
                   *a=RotaIzqDer(*a);
                else
                   if(Balance(*a)==-2&&Balance((*a)->der)==1)
                      *a=RotaDerIzq(*a);
       return 1;
       }
     else
         if(x>ArbolRaiz(*a))
           {EliminaAVL(&((*a)->der),x);
            if(Balance(*a)==2&&Balance((*a)->izq)==1)
                *a=RotaDer(*a);
            else
                 if(Balance(*a)==-2&&Balance((*a)->der)==-1)
                    *a=RotaIzq(*a);
                 else
                    if(Balance(*a)==2&&Balance((*a)->izq)==-1)
                       *a=RotaIzqDer(*a);
                    else
                       if(Balance(*a)==-2&&Balance((*a)->der)==1)
                           *a=RotaDerIzq(*a);
           return 1;
           }
         else
            {otro=*a;
             if(otro->der==NULL) //cuando tiene un descendiente
                {*a=otro->izq;
                 free(otro);
                 otro=NULL;
                 return 1;
                 }
```

```
            else
                if(otro->izq==NULL) //cuando tiene un ascendiente
                  {*a=otro->der;
                   free(otro);
                   otro=NULL;
                   return 1;//Si lo elimino
                  }
                else                  //cuando tiene los ascendiente
                  {aux1=aux=(*a)->izq;
                   while(aux->der!=NULL)
                         { aux1=aux;
                           aux=aux->der;
                         }
                   (*a)->info=aux->info;
                   if(aux->der==NULL)
                     { if(aux==aux1)
                           {otro->izq=aux1->izq;
                            free(aux);
                            aux=NULL;
                           }
                        else
                           {
                            aux1->der=aux->izq;
                            free(aux);
                            aux=aux1->der;
                            aux->der->izq=NULL;
                           }
                     }
                   if(Balance(*a)==2&&Balance((*a)->izq)==1)  //reestructura el arbol
                     *a=RotaDer(*a);
                   else
                        if(Balance(*a)==-2&&Balance((*a)->der)==-1)
                          *a=RotaIzq(*a);
                        else
                          if(Balance(*a)==2&&Balance((*a)->izq)==-1)
                            *a=RotaIzqDer(*a);
                          else
                             if(Balance(*a)==-2&&Balance((*a)->der)==1)
                                 *a=RotaDerIzq(*a);

                   return 1;
                  }
            }
      }
}
```

Bibliografía

- http://c.conclase.net/edd/pagina037.html

- Estructuras de datos en C
- Tenenbaum, Langsman y Augenstein

- Estructuras de datos y algoritmos
- Aho, Hopcroft y Ullman

- Estructura de datos
- Cairó-Guardati