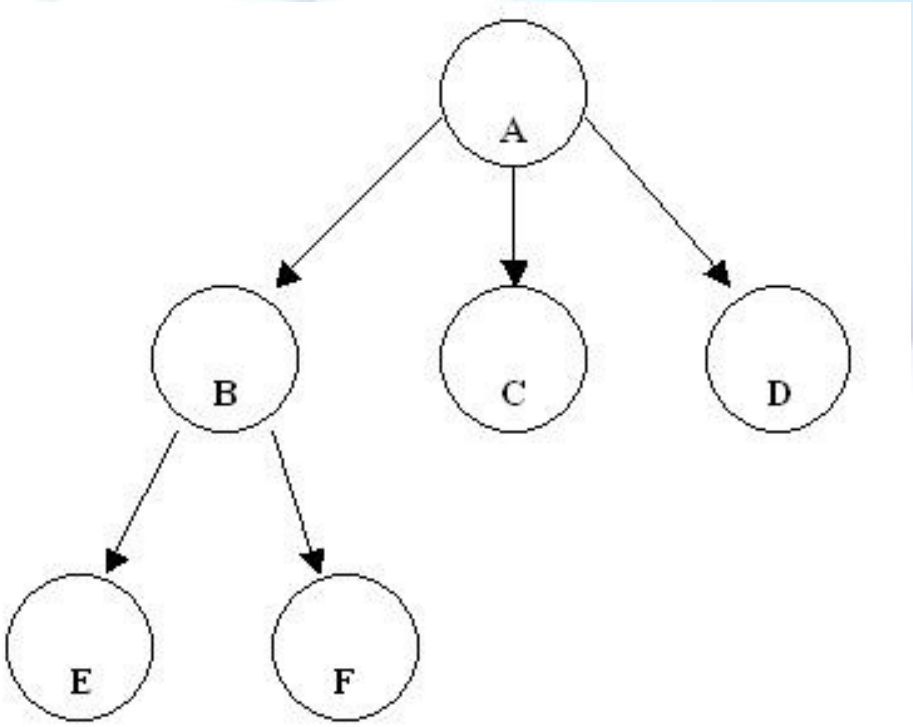
INSTITUTO SAN ISIDRO

Curso de Java a distancia

Clase 18: Estructuras dinámicas: Conceptos de árboles

Igual que la lista, el árbol es una estructura de datos. Son muy eficientes para la búsqueda de información. Los árboles soportan estructuras no lineales.



Algunos conceptos de la estructura de datos tipo árbol:

Nodo hoja: Es un nodo sin descendientes (Nodo terminal)
Ej. Nodos E ? F ? C y D.

Nodo interior: Es un nodo que no es hoja.
Ej. Nodos A y B.

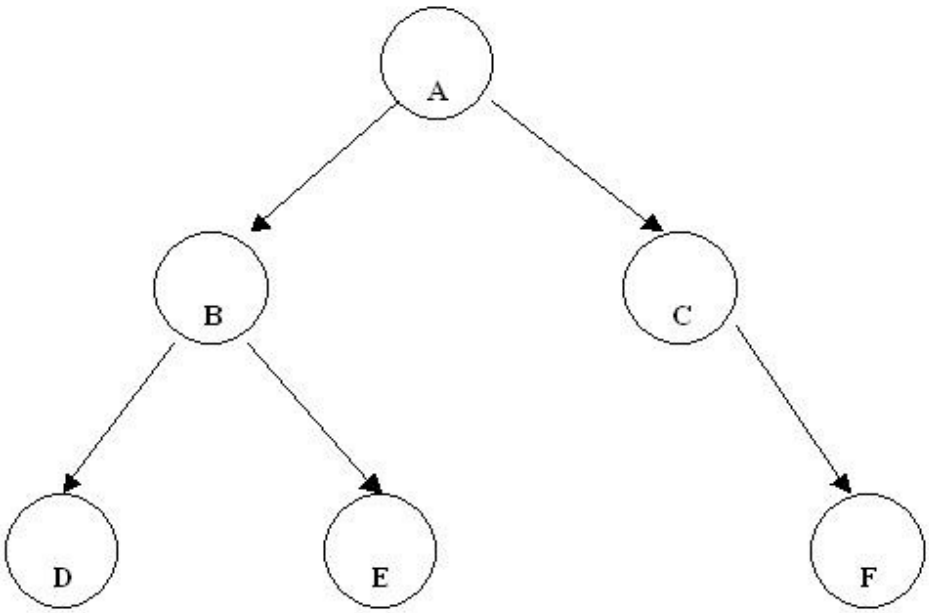
Nivel de un árbol: El nodo A está en el nivel 1 sus descendientes directos están en el nivel 2 y así sucesivamente.
El nivel del árbol está dado por el nodo de máximo nivel.
Ej. Este árbol es de nivel 3.

Grado de un nodo: es el número de nodos hijos que tiene dicho nodo (solo se tiene en cuenta los nodos interiores)
Ej. El nodo A tiene grado 3.
El nodo B tiene grado 2.
Los otros nodos no tienen grado porque no tienen descendientes.

Grado de un árbol: Es el máximo de los grados de todos los nodos de un árbol.
Ej. El grado del árbol es 3.

Longitud de camino del nodo x: Al número de arcos que deben ser recorridos para llegar a un nodo x, partiendo de la raiz.
La raiz tiene longitud de camino 1, sus descendientes directos tienen longitud de camino 2, etc.
En forma general un nodo en el nivel i tiene longitud de camino i.

Árbol binario: Un árbol es binario si cada nodo tiene como máximo 2 descendientes.

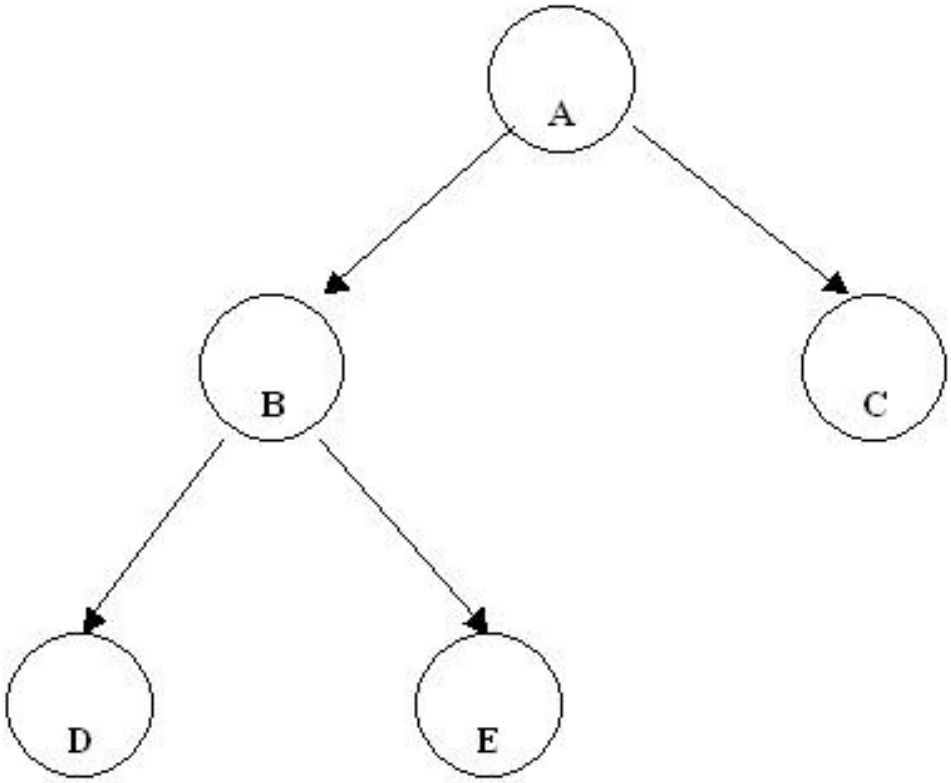


Para cada nodo está definido el subárbol izquierdo y el derecho.
Para el nodo A el subárbol izquierdo está constituido por los nodos B, D y E. Y el subárbol derecho está formado por los nodos C y F.
Lo mismo para el nodo B tiene el subárbol izquierdo con un nodo (D) y un nodo en el subárbol derecho (E).
El nodo D tiene ambos subárboles vacíos.
El nodo C tiene el subárbol izquierdo vacío y el subárbol derecho con un nodo (F).

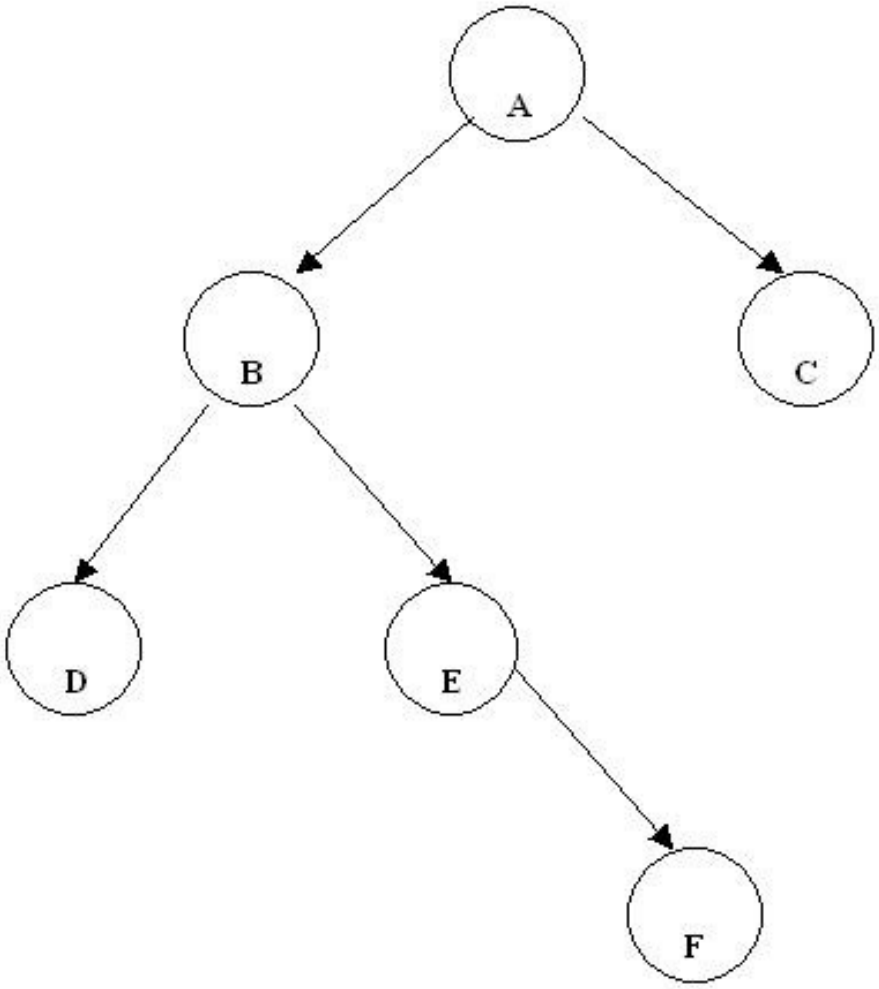
Árbol binario perfectamente equilibrado: Si para cada nodo el número de nodos en el subárbol izquierdo y el número de nodos en el subárbol derecho, difiere como mucho en una unidad.
Hay que tener en cuenta todos los nodos del árbol.

El árbol de más arriba es perfectamente equilibrado.

Ej. árbol que no es perfectamente equilibrado:
El nodo A tiene 3 nodos en el subárbol izquierdo y solo uno en el subárbol derecho, por lo que no es perfectamente equilibrado.

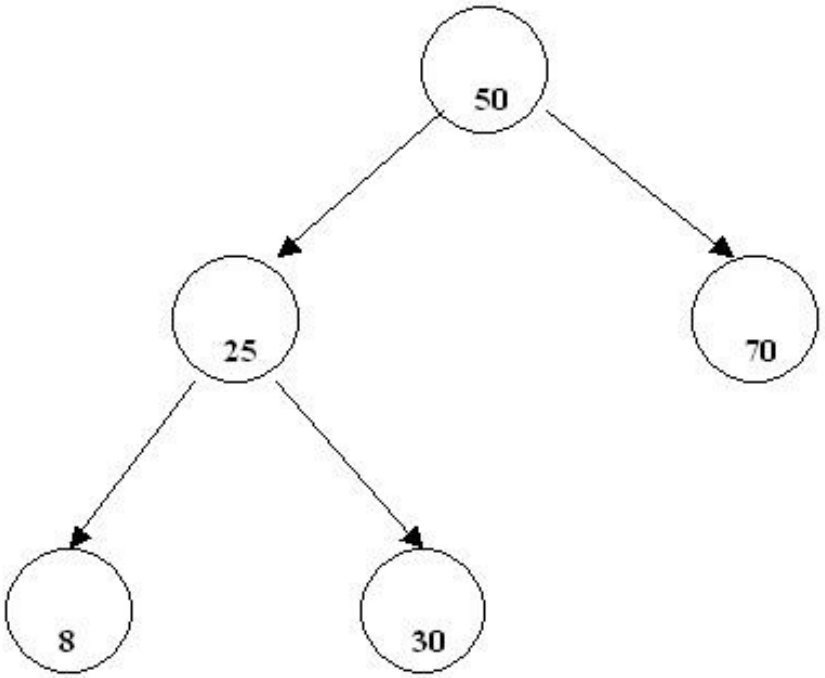


Árbol binario completo: Es un árbol binario con hojas como máximo en los niveles n-1 y n (Siendo n el nivel del árbol)
Los dos árboles graficados son completos porque son árboles de nivel 3 y hay nodos hoja en el nivel 3 en el primer caso, y hay nodos hoja en los niveles 3 y 2 en el segundo caso.
Ej. Árbol binario no completo:



Hay nodos hoja en los niveles 4, 3 y 2. No debería haber nodos hojas en el nivel 2.

Árbol binario ordenado: Si para cada nodo del árbol, los nodos ubicados a la izquierda son inferiores al que consideramos raíz para ese momento y los nodos ubicados a la derecha son mayores que la raíz.



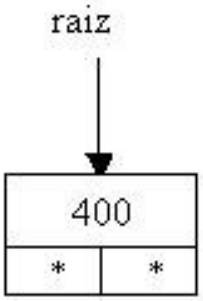
Ej. Analicemos si se trata de un árbol binario ordenado:
Para el nodo que tiene el 50:
Los nodos del subárbol izquierdo son todos menores a 50? 8, 25, 30 Si
Los nodos del subárbol derecho son todos mayores a 50? 70 Si.
Para el nodo que tiene el 25:
Los nodos del subárbol izquierdo son todos menores a 25? 8 Si
Los nodos del subárbol derecho son todos mayores a 25? 30 Si.

No hace falta analizar los nodos hoja. Si todas las respuestas son afirmativas podemos luego decir que se trata de un árbol binario ordenado.

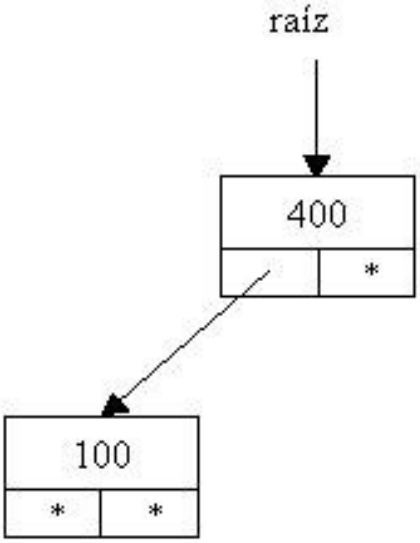
Estructuras dinámicas: Inserción de nodos y recorrido de un árbol binario

Para administrar un árbol binario ordenado debemos tener especial cuidado en la inserción. Inicialmente el árbol está vacío, es decir raíz apunta a null:

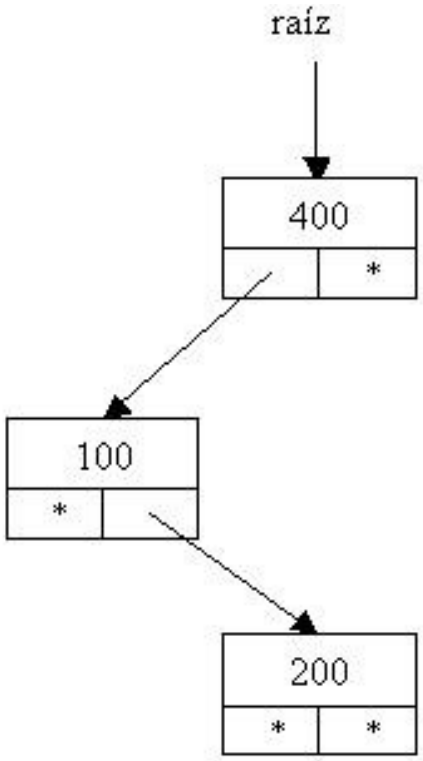
Insertamos el 400



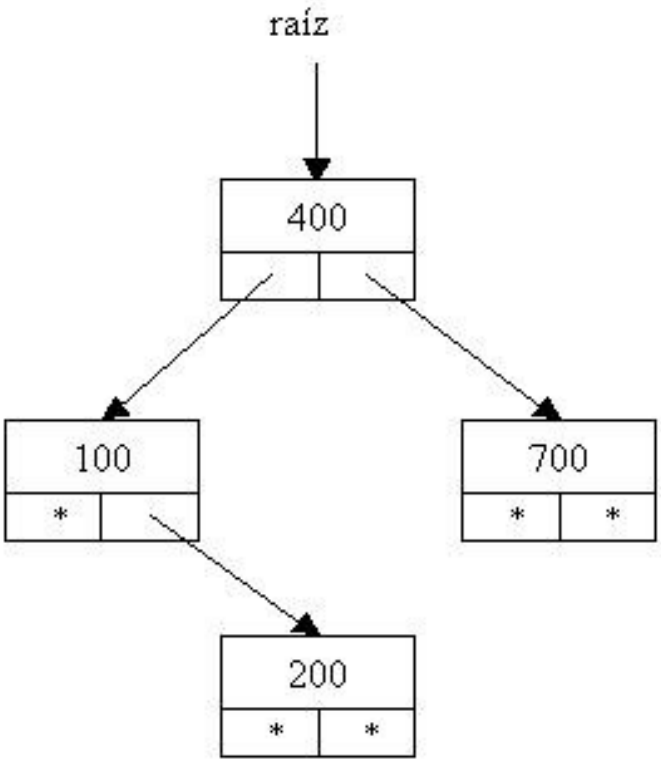
Insertamos el valor 100. Debemos analizar si raíz es distinto a null verificamos si 100 es mayor o menor a la información del nodo apuntado por raíz, en este caso es menor y como el subárbol izquierdo es null debemos insertarlo allí.



Insertamos el 200. Hay que tener en cuenta que siempre comenzamos las comparaciones a partir de raíz. El 200 es menor que 400, descendemos por el subárbol izquierdo. Luego analizamos y vemos que el 200 es mayor a 100, debemos avanzar por derecha. Como el subárbol derecho es null lo insertamos en dicha posición.



Insertamos el 700 y el árbol será:



Como podemos observar si cada vez que insertamos un nodo respetamos este algoritmo siempre estaremos en presencia de un árbol binario ordenado. Posteriormente veremos el algoritmo en java para la inserción de información en el árbol.

Búsqueda de información en un árbol binario ordenado.

Este es una de los principales usos de los árboles binarios.
Para realizar una búsqueda debemos ir comparando la información a buscar y descender por el subárbol izquierdo o derecho según corresponda.

Ej. Si en el árbol anterior necesitamos verificar si está almacenado el 700, primero verificamos si la información del nodo apuntado por raíz es 700, en caso negativo verificamos si la información a buscar (700) es mayor a la información de dicho nodo (400) en caso afirmativo descendemos por el subárbol derecho en caso contrario descendemos por el subárbol izquierdo.
Este proceso lo repetimos hasta encontrar la información buscada o encontrar un subárbol vacío.

Recorridos de árboles binarios.

Recorrer: Pasar a través del árbol enumerando cada uno de sus nodos una vez.
Visitar: Realizar algún procesamiento del nodo.

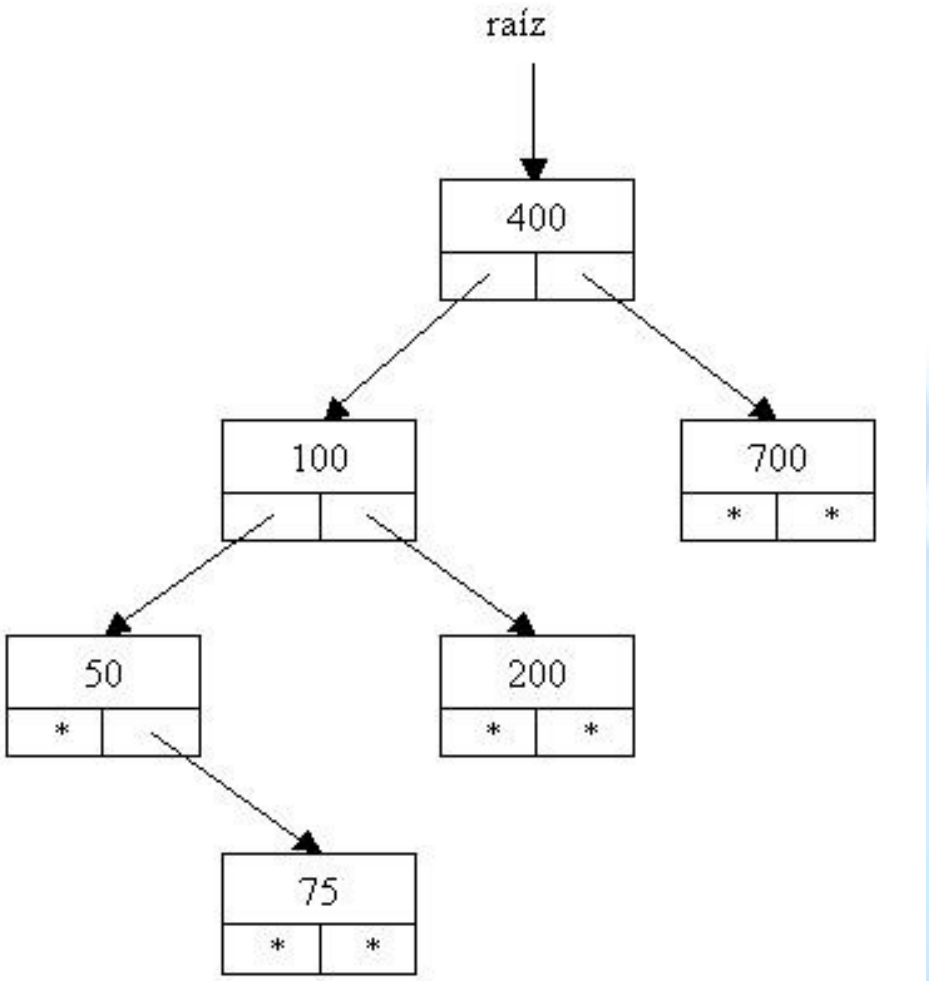
Los árboles pueden ser recorridos en varios órdenes:

Pre-orden:
- Visitar la raíz.
- Recorrer el subárbol izquierdo en pre-orden.
- Recorrer el subárbol derecho en pre-orden.

Entre-orden
- Recorrer el subárbol izquierdo en entre-orden.
- Visitar la raíz.
- Recorrer el subárbol derecho en entre-orden.

Post-orden
- Recorrer el subárbol izquierdo en post-orden.
- Recorrer el subárbol derecho en post-orden.
- Visitar la raíz.

Ejemplo:



Veamos como se imprimen las informaciones de los nodos según su recorrido:

Recorrido preorden:

Visitar la raiz: 400
Recorrer el subárbol izquierdo en preorden.
Visitar la raiz: 100
Recorrer el subárbol izquierdo en preorden.
Visitar la raiz: 50
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Visitar la raiz: 75
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Visitar la raiz: 200
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Visitar la raiz: 700
Recorrer el subárbol izquierdo en preorden.
Vacío
Recorrer el subárbol derecho en preorden.
Vacío

Es decir que el orden de impresión de la información es:

400 ? 100 ?50 ? 75 ?200 ? 700

Es importante analizar que el recorrido de árboles es recursivo. Recorrer un subárbol es semejante a recorrer un árbol.
Es buena práctica dibujar el árbol en un papel y hacer el seguimiento del recorrido y las visitas a cada nodo.

Recorrido entreorden:

Recorrer el subárbol izquierdo en entreorden.
    Recorrer el subárbol izquierdo en entreorden.
        Recorrer el subárbol izquierdo en entreorden.
            Vacío
        Visitar la raiz: *50*
        Recorrer el subárbol derecho en entreorden.
            Recorrer el subárbol izquierdo en entreorden.
                Vacío
            Visitar la raiz : *75*
            Recorrer el subárbol derecho en entreorden.
                Vacío
    Visitar la raiz: *100*
    Recorrer el subárbol derecho en entreorden.
        Recorrer el subárbol izquierdo en entreorden.
            Vacío
        Visitar la raiz: *200*
        Recorrer el subárbol derecho en entreorden.
            Vacío

Visitar la raiz: 400
Recorrer el subárbol derecho en entreorden.
    Recorrer el subárbol izquierdo en entreorden.
        Vacío
    Visitar la raiz: *700*
    Recorrer el subárbol derecho en entreorden.
        Vacío

Es decir que el orden de impresión de la información es:

50 ?75 ? 100 ?200 ? 400 ? 700

Si observamos podemos ver que la información aparece ordenada.
Este tipo de recorrido es muy útil cuando queremos procesar la información del árbol en orden.

Recorrido postorden:

Recorrer el subárbol izquierdo en postorden.
Recorrer el subárbol izquierdo en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.
Vacío
Visitar la raiz: 75
Visitar la raiz: 50
Recorrer el subárbol derecho en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.
Vacío
Visitar la raiz: 200
Visitar la raiz: 100

Recorrer el subárbol derecho en postorden.
Recorrer el subárbol izquierdo en postorden.
Vacío
Recorrer el subárbol derecho en postorden.

Visitar la raiz: 700

Visitar la raiz: 400

Es decir que el orden de impresión de la información es:

75 ? 50 ? 200 ? 100 ? 700 ? 400

Estructuras dinámicas: Implementación en Java de un árbol binario ordenado

Problema 1:

A continuación desarrollamos una clase para la administración de un árbol binario ordenado.

Programa:

```
public class ArbolBinarioOrdenado {
   class Nodo
```

```
{
  int info;
  Nodo izq, der;
}
Nodo raiz;

public ArbolBinarioOrdenado() {
   raiz=null;
}

public void insertar (int info)
{
   Nodo nuevo;
   nuevo = new Nodo ();
   nuevo.info = info;
   nuevo.izq = null;
   nuevo.der = null;
   if (raiz == null)
      raiz = nuevo;
   else
   {
      Nodo anterior = null, reco;
      reco = raiz;
      while (reco != null)
      {
         anterior = reco;
         if (info < reco.info)
            reco = reco.izq;
         else
            reco = reco.der;
      }
      if (info < anterior.info)
         anterior.izq = nuevo;
      else
         anterior.der = nuevo;
   }
}


private void imprimirPre (Nodo reco)
{
   if (reco != null)
   {
      System.out.print(reco.info + " ");
      imprimirPre (reco.izq);
      imprimirPre (reco.der);
   }
}
```

```
public void imprimirPre ()
{
   imprimirPre (raiz);
   System.out.println();
}

private void imprimirEntre (Nodo reco)
{
   if (reco != null)
   {
      imprimirEntre (reco.izq);
      System.out.print(reco.info + " ");
      imprimirEntre (reco.der);
   }
}

public void imprimirEntre ()
{
   imprimirEntre (raiz);
   System.out.println();
}


private void imprimirPost (Nodo reco)
{
   if (reco != null)
   {
      imprimirPost (reco.izq);
      imprimirPost (reco.der);
      System.out.print(reco.info + " ");
   }
}


public void imprimirPost ()
{
   imprimirPost (raiz);
   System.out.println();
}

public static void main (String [] ar)
{
   ArbolBinarioOrdenado abo = new ArbolBinarioOrdenado ();
   abo.insertar (100);
   abo.insertar (50);
   abo.insertar (25);
   abo.insertar (75);
```

```
        abo.insertar (150);
        System.out.println ("Impresion preorden: ");
        abo.imprimirPre ();
        System.out.println ("Impresion entreorden: ");
        abo.imprimirEntre ();
        System.out.println ("Impresion postorden: ");
        abo.imprimirPost ();
    }
}
```

```
public void insertar (int info)
    {
        Nodo nuevo;
        nuevo = new Nodo ();
        nuevo.info = info;
        nuevo.izq = null;
        nuevo.der = null;
        if (raiz == null)
            raiz = nuevo;
        else
        {
            Nodo anterior = null, reco;
            reco = raiz;
            while (reco != null)
            {
                anterior = reco;
                if (info < reco.info)
                    reco = reco.izq;
                else
                    reco = reco.der;
            }
            if (info < anterior.info)
                anterior.izq = nuevo;
            else
                anterior.der = nuevo;
        }
    }
```

Creamos un nodo y disponemos los punteros izq y der a null, guardamos la información que llega al método en el nodo.
Si el árbol está vacío, apuntamos raíz al nodo creado; en caso de no estar vacío, dentro de una estructura repetitiva vamos comparando info con la información del nodo, si info es mayor a la del nodo descendemos por el subárbol derecho en caso contrario descendemos por el subárbol izquierdo.
Cuando se encuentra un subárbol vacío insertar el nodo en dicho subárbol. Para esto llevamos un puntero anterior dentro del while.

```
    private void imprimirPre (Nodo reco)
```

```
{
   if (reco != null)
   {
      System.out.print(reco.info + " ");
      imprimirPre (reco.izq);
      imprimirPre (reco.der);
   }
}

public void imprimirPre ()
{
   imprimirPre (raiz);
   System.out.println();
}
```

El método imprimirPre(), es decir el no recursivo se encarga de llamar al método recursivo pasando la dirección del nodo raiz.

El método recursivo void imprimirPre (Nodo reco) lo primero que verifica con un if si reco está apuntando a un nodo (esto es verdad si reco es distinto a null), en caso afirmativo ingresa al bloque del if y realiza:

- Visitar la raiz.
- Recorrer el subárbol izquierdo en pre-orden.
- Recorrer el subárbol derecho en pre-orden.

La visita en este caso es la impresión de la información del nodo y los recorridos son las llamadas recursivas pasando las direcciones de los subárboles izquierdo y derecho.

Los algoritmos de los recorridos en entreorden y postorden son similares. La diferencia es que la visita la realizamos entre las llamadas recursivas en el recorrido en entre orden:

```
private void imprimirEntre (Nodo reco)
{
   if (reco != null)
   {
      imprimirEntre (reco.izq);
      System.out.print(reco.info + " ");
      imprimirEntre (reco.der);
   }
}
```

y por último en el recorrido en postorden la visita la realizamos luego de las dos llamadas recursivas:

```
private void imprimirPost (Nodo reco)
{
   if (reco != null)
   {
      imprimirPost (reco.izq);
      imprimirPost (reco.der);
      System.out.print(reco.info + " ");
   }
```

```
    }
```

Problema 2:

Confeccionar una clase que permita insertar un entero en un árbol binario ordenado verificando que no se encuentre previamente dicho número.
Desarrollar los siguientes métodos:
1 - Retornar la cantidad de nodos del árbol.
2 - Retornar la cantidad de nodos hoja del árbol.
3 - Imprimir en entre orden.
4 - Imprimir en entre orden junto al nivel donde se encuentra dicho nodo.
5 - Retornar la altura del árbol.
6 - Imprimir el mayor valor del árbol.
7 - Borrar el nodo menor del árbol.

```
public class ArbolBinarioOrdenado {
   class Nodo
   {
      int info;
      Nodo izq, der;
   }
   Nodo raiz;
   int cant;
   int altura;

   public ArbolBinarioOrdenado() {
      raiz=null;
   }

   public void insertar (int info) {
      if (!existe(info)) {
         Nodo nuevo;
         nuevo = new Nodo ();
         nuevo.info = info;
         nuevo.izq = null;
          nuevo.der = null;
         if (raiz == null)
            raiz = nuevo;
         else {
            Nodo anterior = null, reco;
            reco = raiz;
            while (reco != null)  {
               anterior = reco;
               if (info < reco.info)
                  reco = reco.izq;
               else
                  reco = reco.der;
            }
            if (info < anterior.info)
               anterior.izq = nuevo;
            else
```

```
            anterior.der = nuevo;
        }
    }
}

public boolean existe(int info) {
    Nodo reco=raiz;
    while (reco!=null) {
        if (info==reco.info)
            return true;
        else
            if (info>reco.info)
                reco=reco.der;
            else
                reco=reco.izq;
    }
    return false;
}

private void imprimirEntre (Nodo reco)  {
    if (reco != null)  {
        imprimirEntre (reco.izq);
        System.out.print(reco.info + " ");
        imprimirEntre (reco.der);
    }
}

public void imprimirEntre () {
    imprimirEntre (raiz);
    System.out.println();
}


private void cantidad(Nodo reco) {
    if (reco!=null) {
        cant++;
        cantidad(reco.izq);
        cantidad(reco.der);
    }
}

public int cantidad() {
    cant=0;
    cantidad(raiz);
    return cant;
}

private void cantidadNodosHoja(Nodo reco) {
```

```
    if (reco!=null) {
        if (reco.izq==null && reco.der==null)
            cant++;
        cantidadNodosHoja(reco.izq);
        cantidadNodosHoja(reco.der);
    }
}

public int cantidadNodosHoja() {
    cant=0;
    cantidadNodosHoja(raiz);
    return cant;
}

private void imprimirEntreConNivel (Nodo reco,int nivel)  {
    if (reco != null) {
        imprimirEntreConNivel (reco.izq,nivel+1);
        System.out.print(reco.info + " ("+nivel+") - ");
        imprimirEntreConNivel (reco.der,nivel+1);
    }
}

public void imprimirEntreConNivel () {
    imprimirEntreConNivel (raiz,1);
    System.out.println();
}

private void retornarAltura (Nodo reco,int nivel)    {
    if (reco != null) {
        retornarAltura (reco.izq,nivel+1);
        if (nivel>altura)
            altura=nivel;
        retornarAltura (reco.der,nivel+1);
    }
}

public  int retornarAltura () {
    altura=0;
    retornarAltura (raiz,1);
    return altura;
}

public void mayorValorl() {
    if (raiz!=null) {
        Nodo reco=raiz;
        while (reco.der!=null)
            reco=reco.der;
        System.out.println("Mayor valor del árbol:"+reco.info);
```

```
    }
  }

  public void borrarMenor() {
    if (raiz!=null) {
      if (raiz.izq==null)
        raiz=raiz.der;
      else {
        Nodo atras=raiz;
        Nodo reco=raiz.izq;
        while (reco.izq!=null) {
          atras=reco;
          reco=reco.izq;
        }
        atras.izq=reco.der;
      }
    }
  }

  public static void main (String [] ar)
  {
    ArbolBinarioOrdenado abo = new ArbolBinarioOrdenado ();
    abo.insertar (100);
    abo.insertar (50);
    abo.insertar (25);
    abo.insertar (75);
    abo.insertar (150);
    System.out.println ("Impresion entreorden: ");
    abo.imprimirEntre ();
    System.out.println ("Cantidad de nodos del árbol:"+abo.cantidad());
    System.out.println ("Cantidad de nodos hoja:"+abo.cantidadNodosHoja());
    System.out.println ("Impresion en entre orden junto al nivel del nodo.");
    abo.imprimirEntreConNivel();
    System.out.print ("Artura del arbol:");
    System.out.println(abo.retornarAltura());
    abo.mayorValorI();
    abo.borrarMenor();
    System.out.println("Luego de borrar el menor:");
    abo.imprimirEntre ();
  }
}
```

Para verificar si existe un elemento de información en el árbol disponemos un puntero reco en el nodo apuntado por raiz. Dentro de un while verificamos si la información del parámetro coincide con la información del nodo apuntado por reco, en caso afirmativo salimos del método retornando true, en caso contrario si la información a buscar es mayor a la del nodo procedemos a avanzar reco con la dirección del subárbol derecho:

```
  public boolean existe(int info) {
    Nodo reco=raiz;
```

```
    while (reco!=null) {
      if (info==reco.info)
        return true;
      else
        if (info>reco.info)
          reco=reco.der;
        else
          reco=reco.izq;
    }
    return false;
  }
```

Para retornar la cantidad de nodos del árbol procedemos a inicializar un atributo de la clase llamado cant con cero. Llamamos al método recursivo y en cada visita al nodo incrementamos el atributo cant en uno:

```
  private void cantidad(Nodo reco) {
    if (reco!=null) {
      cant++;
      cantidad(reco.izq);
      cantidad(reco.der);
    }
  }

  public int cantidad() {
    cant=0;
    cantidad(raiz);
    return cant;
  }
```

Para imprimir todos los nodos en entre orden junto al nivel donde se encuentra planteamos un método recursivo que llegue la referencia del nodo a imprimir junto al nivel de dicho nodo.
Desde el método no recursivo pasamos la referencia a raiz y un uno (ya que raiz se encuentra en el primer nivel)
Cada vez que descendemos un nivel le pasamos la referencia del subárbol respectivo junto al nivel que se encuentra dicho nodo:

```
  private void imprimirEntreConNivel (Nodo reco,int nivel)  {
    if (reco != null) {
      imprimirEntreConNivel (reco.izq,nivel+1);
      System.out.print(reco.info + " ("+nivel+") - ");
      imprimirEntreConNivel (reco.der,nivel+1);
    }
  }

  public void imprimirEntreConNivel () {
    imprimirEntreConNivel (raiz,1);
    System.out.println();
  }
```

Para obtener la altura del árbol procedemos en el método no recursivo a inicializar el atributo altura con el valor cero. Luego llamamos al método recursivo con la referencia a raiz que se encuentra en el nivel uno. Cada vez que visitamos un nodo procedemos a verificar si el parámetro nivel supera al atributo altura, en dicho caso actualizamos el atributo altura con dicho nivel.

```
private void retornarAltura (Nodo reco,int nivel)    {
   if (reco != null) {
      retornarAltura (reco.izq,nivel+1);
      if (nivel>altura)
         altura=nivel;
      retornarAltura (reco.der,nivel+1);
   }
}

public  int retornarAltura () {
   altura=0;
   retornarAltura (raiz,1);
   return altura;
}
```

Para imprimir el mayor valor del árbol debemos recorrer siempre por derecha hasta encontrar un nodo que almacene null en der:

```
public void mayorValorl() {
   if (raiz!=null) {
      Nodo reco=raiz;
      while (reco.der!=null)
         reco=reco.der;
      System.out.println("Mayor valor del árbol:"+reco.info);
   }
}
```

Para borrar el menor valor del árbol lo primero que comprobamos es si el subárbol izquierdo es nulo luego el menor del árbol es el nodo apuntado por raiz. Luego si el subárbol izquierdo no está vacío procedemos a descender siempre por la izquierda llevando un puntero en el nodo anterior. Cuando llegamos al nodo que debemos borrar procedemos a enlazar el puntero izq del nodo que se encuentra en el nivel anterior con la referencia del subárbol derecho del nodo a borrar:

```
public void borrarMenor() {
   if (raiz!=null) {
      if (raiz.izq==null)
         raiz=raiz.der;
      else {
         Nodo atras=raiz;
         Nodo reco=raiz.izq;
         while (reco.izq!=null) {
            atras=reco;
            reco=reco.izq;
         }
```

```
            atras.izq=reco.der;
        }
    }
}
```

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar