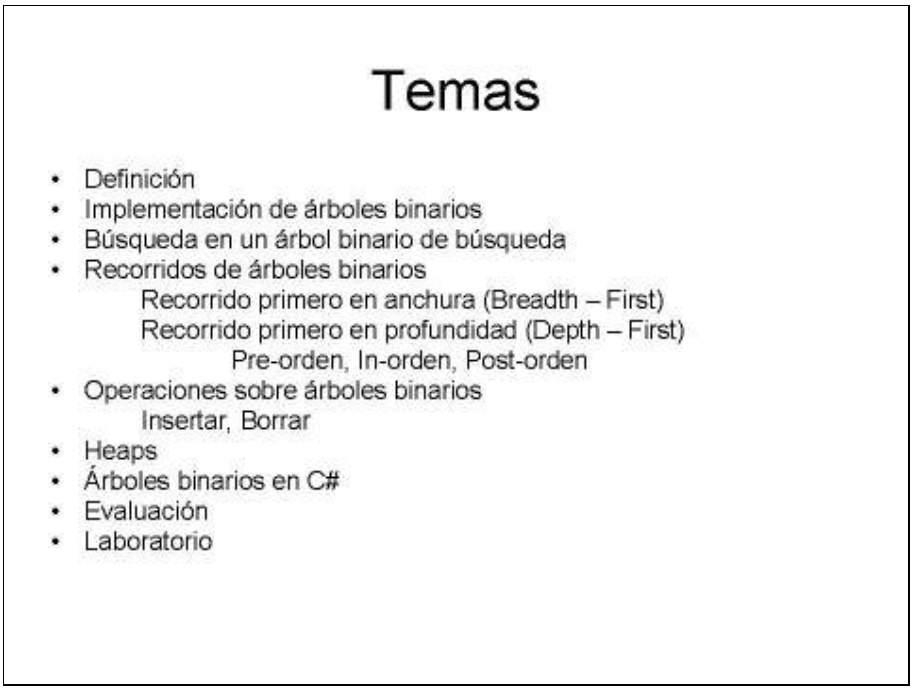
Introducción a Árboles Binarios

Toda la información que se maneja dentro de una computadora se encuentra almacenada en la memoria, que en términos simples es una secuencia de caracteres (bytes) en donde se encuentran las instrucciones y datos a los que se accede directamente a través del procesador de la computadora.

Los sistemas o métodos de organización de datos que permiten un almacenamiento eficiente de la información en la memoria del computador son conocidos como estructuras de datos. Estos métodos de organización constituyen las piezas básicas para la construcción de algoritmos complejos, y permiten implementarlos de manera eficiente.

Gracias por la consulta.

Junio 2007



Las listas ligadas tienen una mayor flexibilidad que los arreglos pero son estructuras lineales y esto dificulta la representación jerárquica de objetos, aunque las colas y pilas pueden con tener algún tipo de jerarquía esta se ve limitada a una sola dimensión.

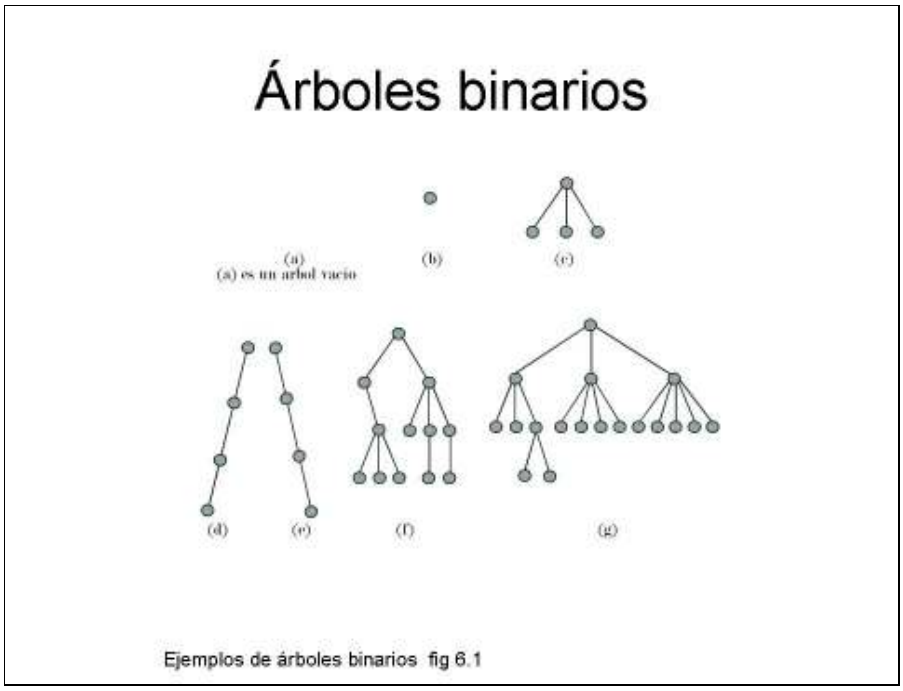
Para evitar esta limitación se han creado otro tipo de dato llamado árbol.

Árboles binarios

Definición: Un árbol es un tipo de datos que consiste de nodos y arcos entre los nodos. Estos árboles crecen hacia abajo, teniendo su raíz en la parte superior y en la parte inferior se encuentran la **hojas** que son nodos terminales.

- La raíz es un nodo que no tiene padre y que solamente puede tener nodos hijo.
- Las hojas no tienen hijos, sus hijos son nulos.
- Cada nodo debe ser alcanzable desde la raíz a través de una única secuencia de arcos llamada "trayectoria"
- El número de arcos en una trayectoria se llama "longitud" de la trayectoria
- El nivel de un nodo es la longitud de la trayectoria desde la raíz al nodo más "1", que es el número de nodos en la trayectoria
- La altura de un árbol no vacío es el máximo nivel de un nodo en el árbol

La definición de un árbol no impone ninguna condición sobre el número de hijos que pueda tener un nodo. Este número puede variar desde cero hasta n número de hijos. Este característica puede ser de mucha ayuda por ejemplo para representar jerárquicamente todas las dependencias de una Universidad, imagina una Universidad con dos campus, pero cada uno de estos campus puede tener una cantidad diferente de departamentos, y a su vez cada departamento puede tener también un distinto numero de dependencias, un árbol es la mejor forma jerárquica de representar a esta Universidad.



Un árbol puede estar formado por:

ningún nodo → árbol vació
un nodo → nodo raíz
mas de un nodo → árbol con raíz y hojas

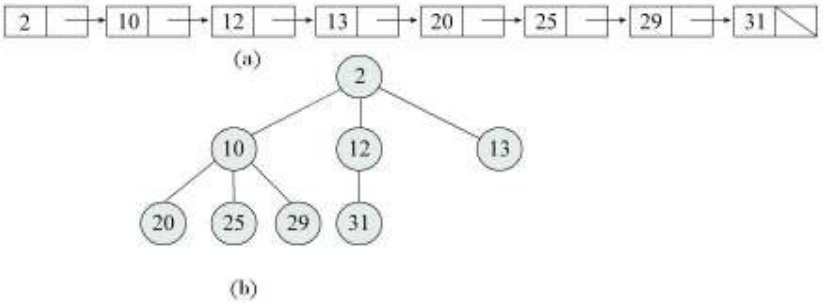
Un tipo especial de árbol es el árbol binario

Un árbol binario es una estructura de datos de tipo árbol en donde cada uno de los nodos del árbol puede tener 0, 1, ó 2 subárboles llamados de acuerdo a su caso como:

- Si el nodo raíz tiene 0 relaciones se llama hoja.
- Si el nodo raíz tiene 1 relación a la izquierda, el segundo elemento de la relación es el subárbol izquierdo.
- Si el nodo raíz tiene 1 relación a la derecha, el segundo elemento de la relación es el subárbol derecho.

Árboles binarios

Un árbol ordenado es aquel donde todos los elementos son almacenados de acuerdo a algún criterio predeterminado de ordenación, como se muestra en el siguiente ejemplo que transforma una lista ligada en un árbol



(a)

(b)

Transformando (a) una lista ligada en (b) un árbol

Pero la representación jerárquica no es la única razón por la cual es conveniente el uso de los árboles.

Considerando una lista ligada de n elementos. Para localizar un elemento, la búsqueda tiene que empezar, por el principio de la lista y considerando que este elemento puede estar hasta el final de la lista, o no estar en la lista, para saberlo tendríamos que revisar toda la lista.

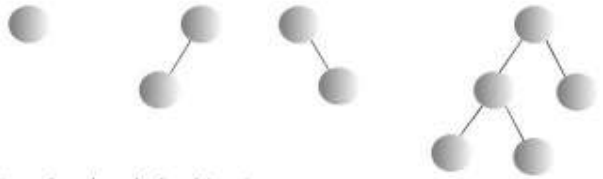
Suponiendo una lista de 10,000 nodos, para obtener la información del último nodo tendríamos que atravesar 9,999 nodos antes de poder llegar hacia esa información, esto es un gran inconveniente para usar una lista ligada.

Sin embargo usando un árbol ordenado correctamente, el numero de nodos necesarios para llegar hasta la información de los últimos nodos es sustancialmente menor, que usando una lista ligada.

Un ejemplo de esto se muestra en la figura. Anterior.

Árboles binarios

Un árbol binario es un árbol cuyos nodos tienen dos hijos (excepto las hojas) y cada hijo es designado como hijo izquierdo e hijo derecho, como se muestra en la siguiente figura



Ejemplos de arboles binarios.

Un árbol binario completo todos los nodos no terminales tienen sus dos hijos y todas las hojas están en el mismo nivel

Un árbol de decisión es un árbol binario en el cual todos los nodos tienen ya sea cero o dos hijos no vacios

Cada uno de las diferentes “circunferencias” representa un elemento o nodo del árbol. El primer nodo suele ser llamado nodo raíz o nodo de nivel 0. Es importante percatarse de que sólo puede haber un nodo raíz en cada árbol.

Los nodos hijos del nodo raíz son llamados nodos de nivel 1, por estar todos ellos al mismo “nivel” al ser todos “hijos” del nodo raíz. Este razonamiento se puede ir aplicándose a los sucesivos niveles, hablándose de nodos nietos (o hijos si nos referimos al nivel inmediatamente superior).

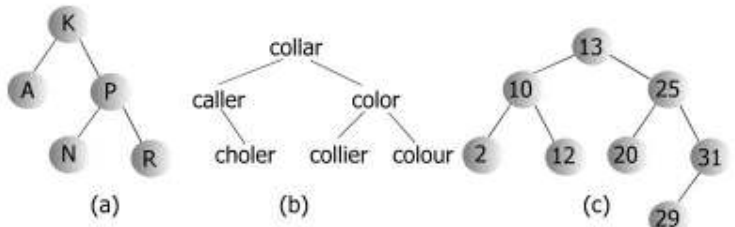
Los nodos que no tiene hijos son nodos hoja y los nodos que sí tienen hijos, salvo el raíz, son llamados nodos rama o nodos internos.

Árboles binarios

Un árbol binario de búsqueda también llamado árbol binario ordenado tienen las siguientes propiedades:

- Para cada nodo "n" del árbol, todos los valores almacenados en el subárbol izquierdo son menores que el valor "v" almacenado en "n"
- Para cada nodo "n" del árbol, todos los valores almacenados en el subárbol derecho son mayores que el valor "v" almacenado en "n"

La siguiente figura muestra ejemplos de árboles binarios de búsqueda



Ejemplos de arboles binarios de busqueda.

Un árbol binario de búsqueda debe de tener características especiales debido a que para poder realizar operaciones sobre el hay que diseñarlos con estas características.

- Si A es la raíz de un árbol y B es la raíz de su subárbol izquierdo (o derecho), se dice que A es el padre de B y se dice que B es el hijo izquierdo (o derecho) de A.
- Un nodo que no tiene hijos se denomina hoja
- El nodo a es antecesor del nodo b (y recíprocamente el nodo b es descendiente del nodo a), si a es el padre de b o el padre de algún ancestro de b.
- Un nodo b es un descendiente izquierdo del nodo a, si b es el hijo izquierdo de a o un descendiente del hijo izquierdo de a. Un descendiente derecho se define de la misma forma.
- Dos nodos son hermanos si son hijos izquierdo y derecho del mismo padre.

Otros términos relacionados con árboles, tienen que ver con su funcionamiento y topología:

- Si cada nodo que NO es una hoja tiene un subárbol izquierdo y un subárbol derecho, entonces se trata de un árbol binario completo.
- El nivel de un nodo es el número de aristas que se deben recorrer para llegar desde ese nodo al nodo raíz. De manera que el nivel del nodo raíz es 0, y el nivel de cualquier otro nodo es el nivel del padre más uno.
- La profundidad de un nodo es el máximo nivel de cualquier hoja en el árbol.

Árboles binarios

Implementación de árboles binarios. Los árboles binarios pueden implementarse por lo menos de dos formas:

- Como arreglos
- Como estructuras ligadas

Para implementarlo como arreglo un nodo se declara como un objeto con un campo de información y dos campos de "referencia", como se muestra en la siguiente figura

Index	Info	Left	Right
0	13	4	2
1	31	6	-1
2	25	7	1
3	12	-1	-1
4	10	5	3
5	2	-1	-1
6	29	-1	-1
7	20	-1	-1

Los árboles binarios pueden ser implementados de dos formas: como arreglos y como estructuras ligadas, para implementar un árbol como un arreglo , un nodo es declarado como un objeto con un campo de información y dos campos de referencias. Estas referencias contienen los índices de las celdas de memoria, donde están guardados los hijos de este nodo.

Por ejemplo en la figura podemos ver que a la raíz se le asigna el índice cero y para representar una referencia a un valor nulo utilizamos el –1

Entonces:

Los dos hijos del nodo 13 están localizados en las posiciones 4 y 2
Los dos hijos del nodo 31 están localizados uno en la posición 6 y otro es nulo
Los dos hijos del nodo 29 son nulos, no tiene hijos es una hoja del arbol

Árboles binarios

El problema de la implementación como arreglo es la flexibilidad, por consiguiente es mejor implementarlo como una estructura ligada donde un nodo es una instancia de una clase compuesta de un campo de información y dos campos de referencia. El árbol es una instancia que tiene un nodo raíz y puede agregar objetos tipo nodo según sea necesario. Esta idea en C# se muestra a continuación.

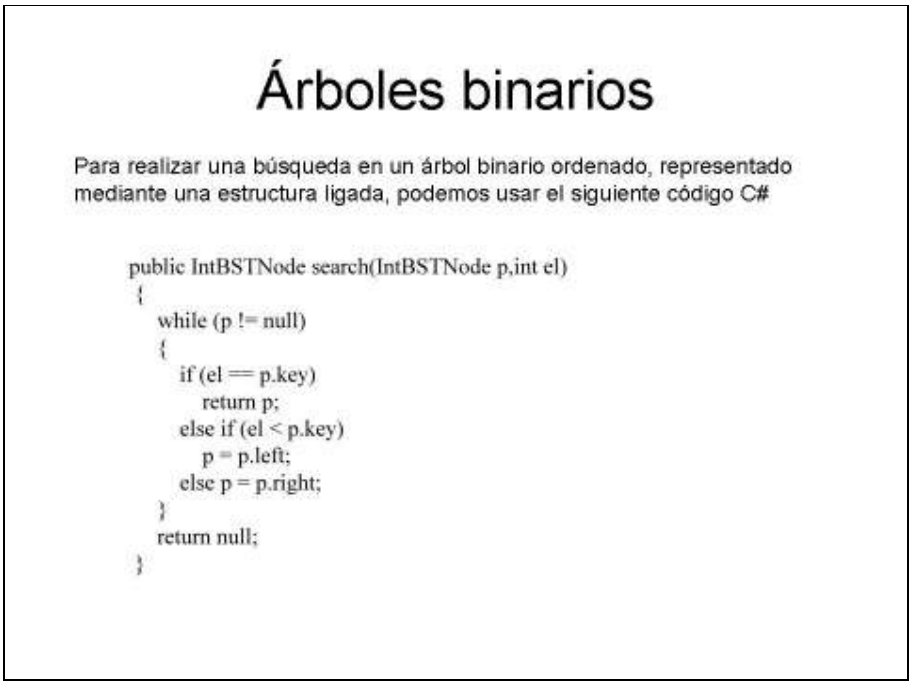
```
public class IntBSTNode{
public int key;
public IntBSTNode left, right;
public IntBSTNode(){
  left = right = null;
}
public IntBSTNode(int el){
  key = el;
  left = right = null;
}
public IntBSTNode(int el, IntBSTNode lt, IntBSTNode rt){
  key = el;
  left = lt;
  right = rt;
}
}
public class IntBST{
  public IntBSTNode root;
  public IntBST(){
    root = null;
  }
  public void visit(IntBSTNode p){
    Console.Write(p.key + " ");
  }
```

Pero la implementación de los árboles como arreglos puede llegar a ser muy inconveniente, por ejemplo las localizaciones de los hijos pueden estarse generando de forma secuencial, entonces al tratar de borrar un nodo del árbol una parte del árbol tiene que volver a rescribirse, el uso de la memoria es muy ineficiente, ya que puede haber muchos espacios de memoria pueden quedar sin usarse.

Árboles binarios

```
public IntBSTNode search(IntBSTNode p){
      return search(p, root.key);
}
public IntBSTNode search(IntBSTNode p, int el){...}
public void breadthFirst(){...}
public void preorder(){
   preorder(root);
}
public void breadthFirst(){...}
public void preorder(){
   preorder(root);
}

protected void preorder(IntBSTNode p){...}
public void inorder(){...}
protected void inorder(IntBSTNode p){...}
public void postorder(){...}
protected void postorder(IntBSTNode p)
public void iterativePreorder(){...}
public void iterativeInorder(){...}
public void iterativePostorder(){...}
public void Morrisorder(){...}
public void insert(int el){...}
public void deleteByMerging(int el){...}
public void deleteByCopying(int el){...}
public void balance(int []data, int first,int last){...}
}
```



Un algoritmo para localizar un elemento en un árbol es el mostrado en la figura:

Para cada nodo, compara el valor a ser localizado (key), con el valor del nodo que esta siendo referenciado, si la key es menor que el valor, se busca en el subárbol izquierdo. Si la key es mayor que el valor, se busca en el subárbol derecho(es por esto que solo se permite que un nodo tenga dos hijos), si la key coincide con el valor entonces se ha encontrado el nodo deseado y se sale de la búsqueda. La búsqueda es terminada si no tiene camino hacia donde seguir, indicando que el valor buscado no esta en el árbol.

Árboles binarios

Recorrido de árboles. Es el proceso de visitar cada nodo en el árbol exactamente una vez.

Recorrido primero en anchura. Consiste en visitar cada nodo empezando por el nivel mas bajo y moviéndose hacia abajo nivel por nivel, visitando los nodos de cada nivel de izquierda a derecha. Este algoritmo en C# se muestra enseguida (observe el uso de una cola)

```
public void breadthFirst()
{
  IntBSTNode p = root;
  Queue queue = new Queue();
  if (p != null)
  {
    queue.enqueue(p);
    while (!queue.isEmpty())
    {
      p = (IntBSTNode)queue.dequeue();
      visit(p);
      if(p.left != null)
        queue.enqueue(p.left);
      if (p.right != null)
        queue.enqueue(p.right);
    }
  }
}
```

Consiste en ir visitando el árbol por niveles. Primero se visitan los nodos de nivel 1 (como mucho hay uno, la raíz), después los nodos de nivel 2, así hasta que ya no queden más.

En este caso el recorrido no se realizará de forma recursiva sino iterativa, utilizando una cola como estructura de datos auxiliar. El procedimiento consiste en encolar (si no están vacíos) los subárboles izquierdo y derecho del nodo extraído de la cola, y seguir desencolando y encolando hasta que la cola esté vacía.

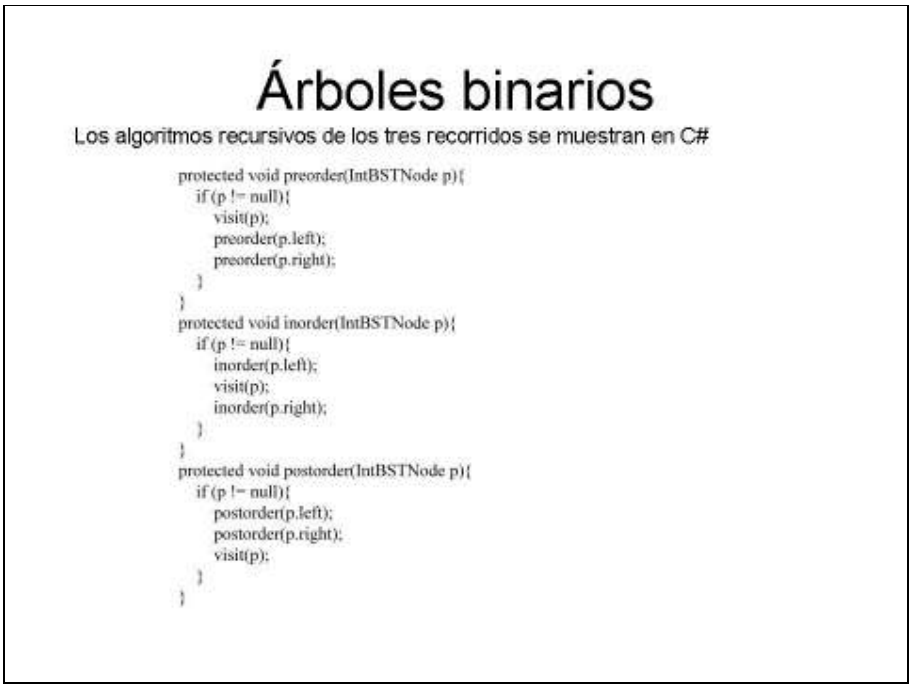
Árboles binarios

Recorrido primero en profundidad. Consiste en proceder tan lejos como sea posible a la izquierda, entonces regresar hasta la primera bifurcación, dar un paso a la derecha y nuevamente proceder tan lejos como sea posible a la izquierda. Este tipo de recorrido tiene variantes, siendo necesarios los siguientes conceptos:

- V = Visitar un nodo
- L = Recorrer el subárbol izquierdo
- R = Recorrer el subárbol derecho

Lo anterior da origen a tres algoritmos (recursivos) básicos de recorridos primero en profundidad dependiendo del orden de las acciones "V", "L" y "R"

- Recorrido en preorden (VLR): primero se visita el nodo, después se recorre el subárbol izquierdo y al final el subárbol derecho.
- Recorrido en inorden (LVR): primero se recorre el subárbol izquierdo, después se visita el nodo y finalmente se recorre el subárbol derecho
- Recorrido en postorden (LRV): primero se recorre el subárbol izquierdo, después del subárbol derecho y finalmente se visita el nodo.



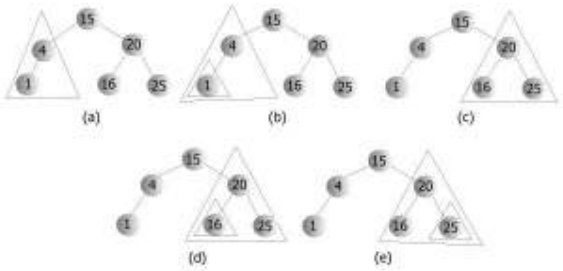
Recorrido en preorden: consiste en visitar el nodo actual (visitar puede ser simplemente mostrar la información contenida en el nodo en la pantalla), y después visitar el subárbol izquierdo y una vez visitado, visitar el subárbol derecho.

Recorrido en inorden u orden central: se visita el subárbol izquierdo, el nodo actual, y después se visita el subárbol derecho

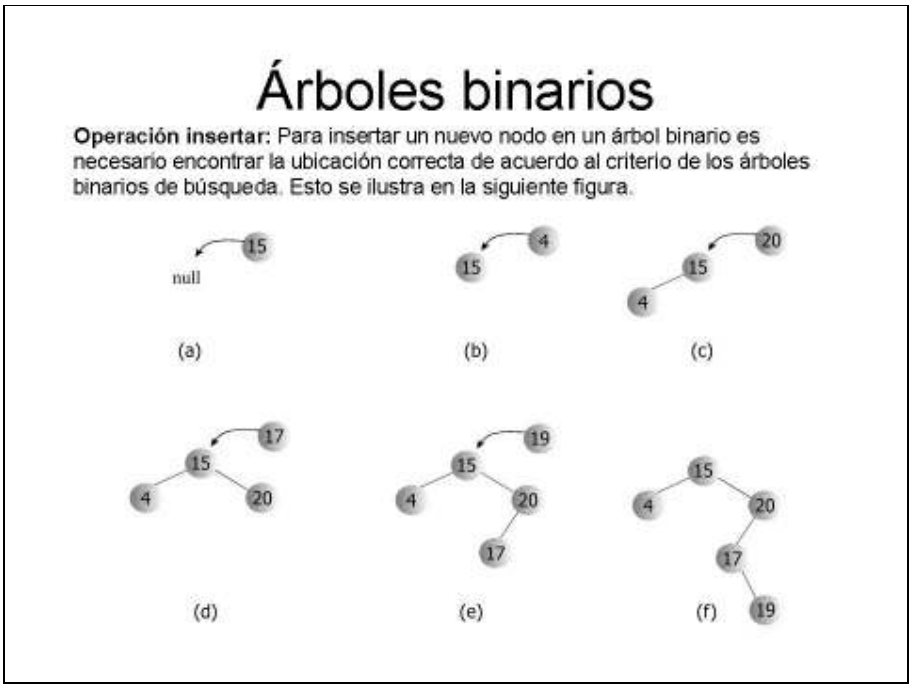
Recorrido en postorden: se visitan primero el subárbol izquierdo, después el subárbol derecho, y por último el nodo actual

Árboles binarios

Para el árbol binario de la siguiente figura se muestra el resultado de los tres recorridos.



Recorrido en preorden (VLR): 15, 4, 1, 20, 16, 25
Recorrido en inorden (LVR): 1, 4, 15, 16, 20, 25
Recorrido en postorden (LRV): 1, 4, 16, 25, 20, 15



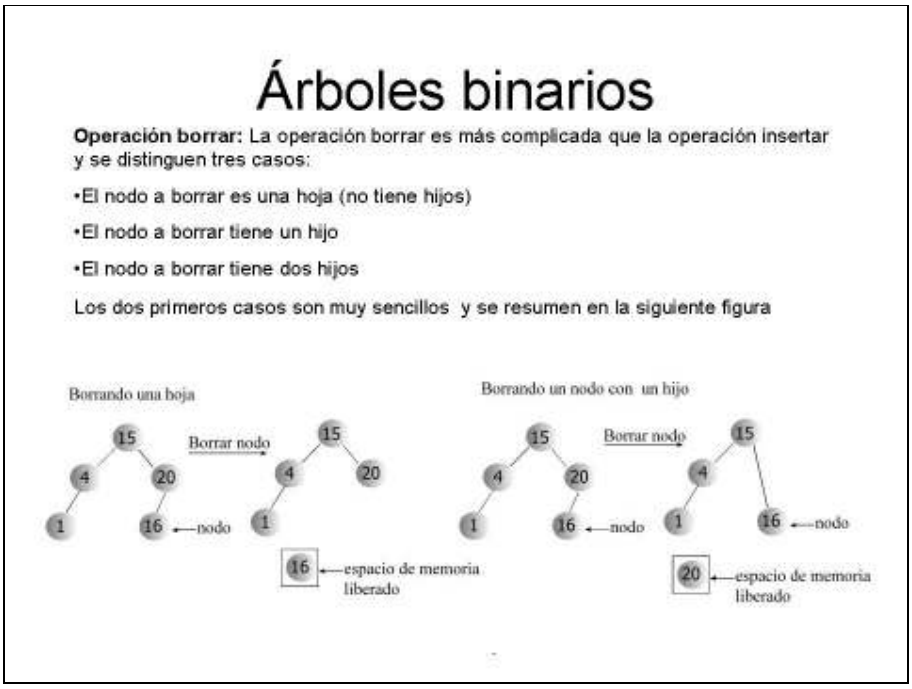
Haciendo una búsqueda en un arbolo, no lo modifica, ni tampoco el recorrido. Pero hay operaciones que por definición deben de hacer modificaciones sistemáticas en el árbol, estas operaciones son la inserción, el borrado, la modificación de elementos, la combinación de árboles y el balanceo.

Para la inserción se utiliza una técnica muy parecida a la de búsqueda, si el valor a insertar (e1) es menor que el valor del nodo se revisa el valor del hijo izquierdo, si es mayor que el valor se revisa el valor del hijo derecho, si el valor de este hijo es vació, entonces se termina la búsqueda y ahí se inserta el nuevo nodo, si no es vació se continua la búsqueda hasta encontrar un nodo vació.

Árboles binarios

Algoritmo insertar: El algoritmo en C# es el siguiente.

```
public void insert(int el)
{
   IntBSTNode p = root, prev = null;
   while (p != null)
   {
      prev = p;
      if (p.key < el)
         p = p.right;
      else p = p.left;
   }
   if (root == null)
      root = new IntBSTNode(el);
   else if (prev.key < el)
      prev.right = new IntBSTNode(el);
   else prev.left = new IntBSTNode(el);
}
```

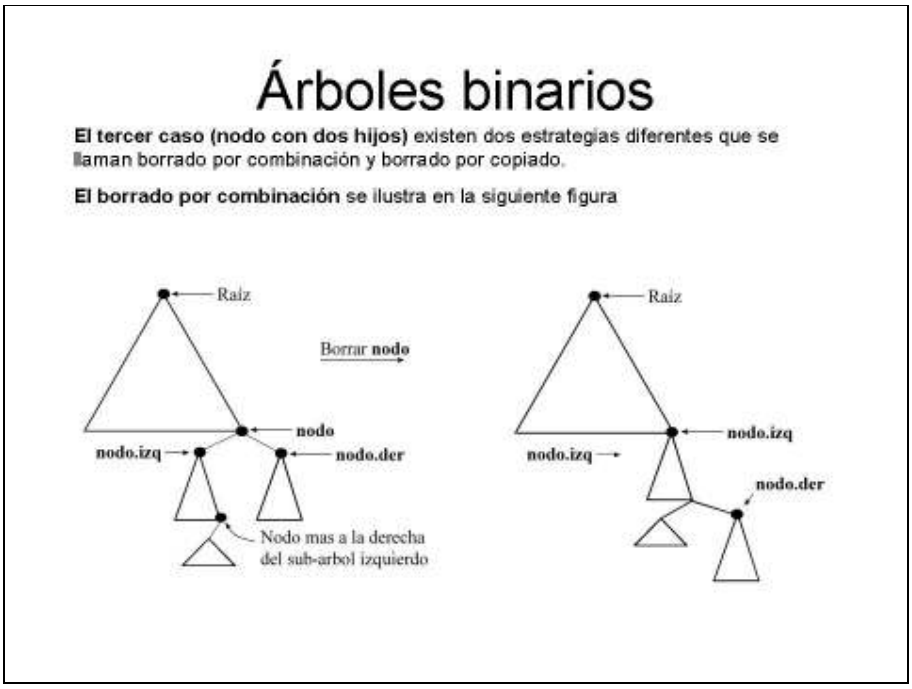


La operación de borrado si resulta ser algo más complicada. Se recuerda que el árbol debe seguir siendo de búsqueda tras el borrado. Pueden darse tres casos, una vez encontrado el nodo a borrar:

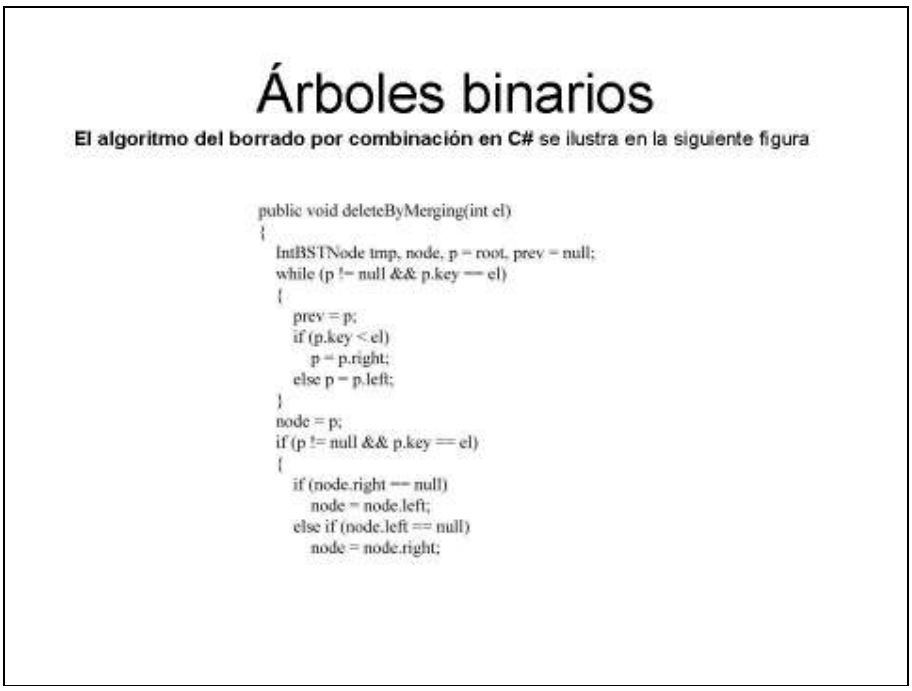
1) El nodo no tiene descendientes. Simplemente se borra.

2) El nodo tiene solo un descendiente por una sola rama. Se borra dicho nodo, y su primer descendiente se asigna como hijo del padre del nodo borrado.

Ejemplo: en el árbol de la figura se borra el nodo cuyo valor es 20. El nodo cuyo valor es 16 toma el lugar de ese nodo.



El nodo tiene al menos un descendiente por cada rama. Al borrar dicho nodo es necesario mantener la coherencia de los enlaces, además de seguir manteniendo la estructura como un árbol binario de búsqueda. La solución consiste en sustituir la información del nodo que se borra por el de una de las hojas, y borrar a continuación dicha hoja.



Borrado por combinación

Por la naturaleza del árbol binario cada valor en el subárbol derecho es mayor que cada valor en el subárbol izquierdo, por lo tanto si el nodo a borrar se puede sustituir por el nodo mas a la a derecha del subárbol izquierdo, o por el nodo mas a la izquierda del subárbol derecho, procurando que si se modifica el subárbol afectado este quede con una estructura de árbol binario de búsqueda.

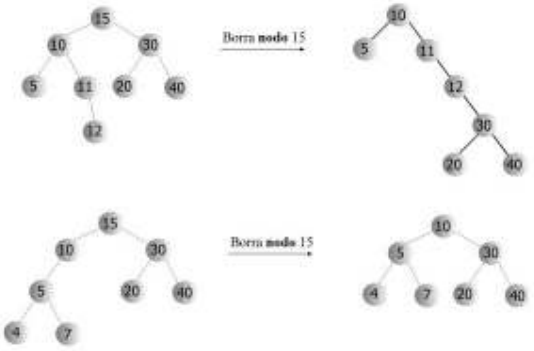
Un inconveniente de este algoritmo es que la altura del árbol puede ser incrementada

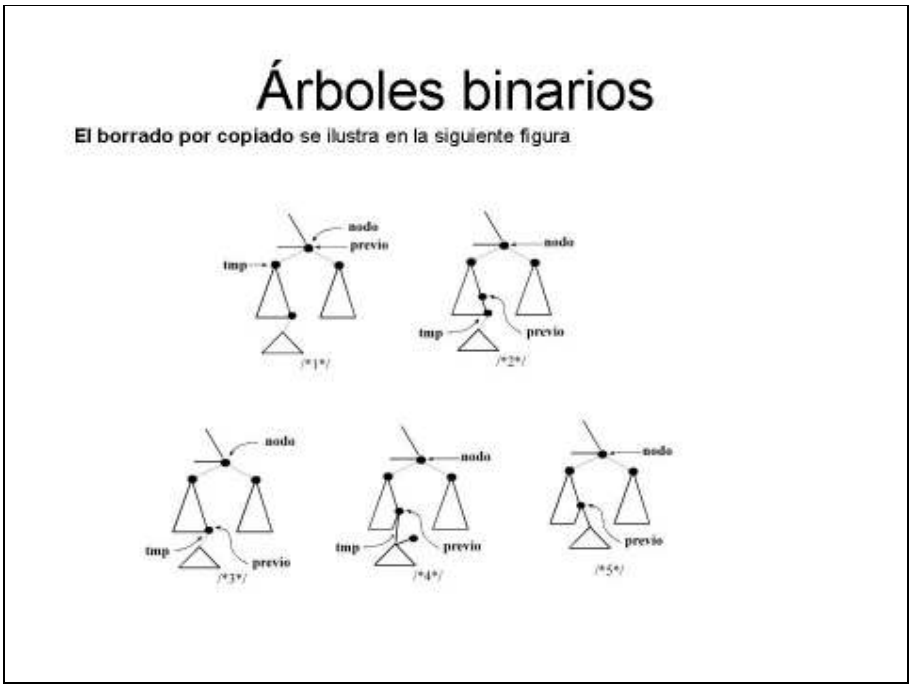
Árboles binarios

```
        else
        {
            tmp = node.left;
            while (tmp.right != null)
                tmp = tmp.right;
            tmp.right = node.right;
            node = node.left;
        }
        if (p == root)
            root = node;
        else if (prev.left == p)
            prev.left = node;
        else prev.right = node;
    }
    else if (root != null)
        Console.Write("el número " + el + "no está en el árbol");
    else Console.Write("el árbol está vacío");
}
```

Árboles binarios

Un ejemplo del algoritmo es el siguiente





Otra solución llamada borrado por copiado, fue propuesto pos Thomas Hibbard y Donald Knuth. Si el nodo tiene dos hijos, el problema puede reducirse a uno de los dos simples casos:

El nodo es una hoja
El nodo tiene solo un hijo no vacío

Esto puede ser hecho substituyendo el valor que es suprimida por su sucesor inmediato o predecesor

Primero el predecesor tiene que ser localizado. Se hace esto, moviéndose hacia la izquierda alcanzando la raíz del nodo del subárbol derecho, dejando otra vez el subárbol secundario y después moverlo tan lejos a la derecha como sea posible

Árboles binarios

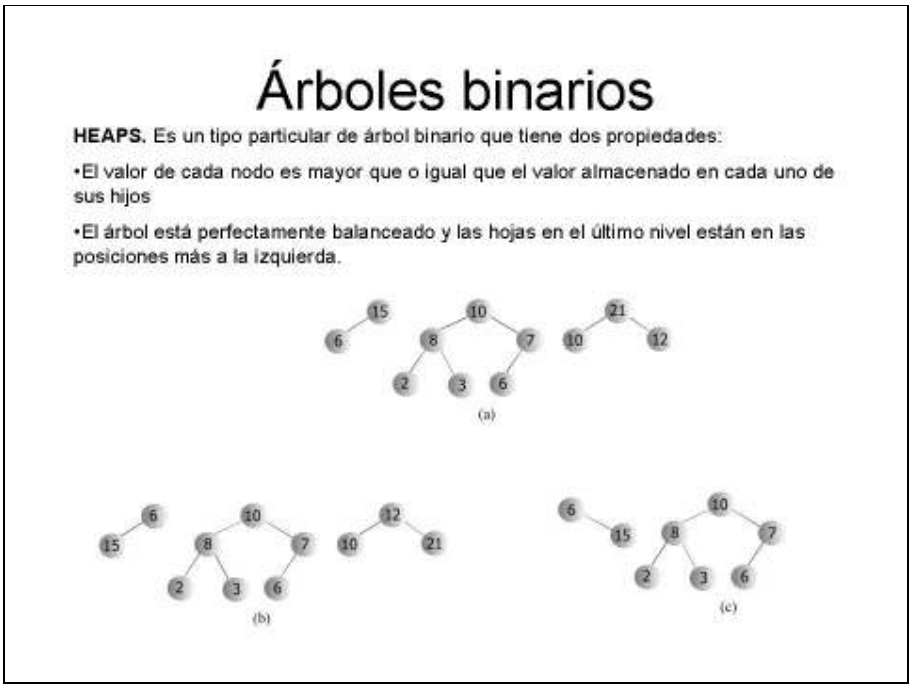
El algoritmo del borrado por copiado en C# se ilustra en la siguiente figura

```
public void deleteByCoping(int el)
    {
      IntBSTNode node, p = root, prev = null;
      while (p != null && p.key != el)
      {
        prev = p;
        if (p.key < el)
          p = p.right;
        else p = p.left;
      }
      node = p;
      if (p != null && p.key == el)
      {
        if (node.right == null)
          node = node.left;
        else if (node.left == null)
          node = node.right;
```

Árboles binarios

```
    else
    {
      IntBSTNode tmp = node.left;
      IntBSTNode previous = node;
      while (tmp.right != null)
      {
        previous = tmp;
        tmp = tmp.right;
      }
      node.key = tmp.key;

      if (previous == node)
        previous.left = tmp.left;
      else previous.right = tmp.left;
    }
    if (p == root)
      root = node;
    else if (prev.left == p)
      prev.left = node;
    else prev.right = node;
  }
  else if (root != null)
    Console.Write("el número " + el + "no está en el árbol");
  else Console.Write("el árbol está vacío");
}
```



Un heap o montículo es un árbol binario completo, y además parcialmente ordenado.

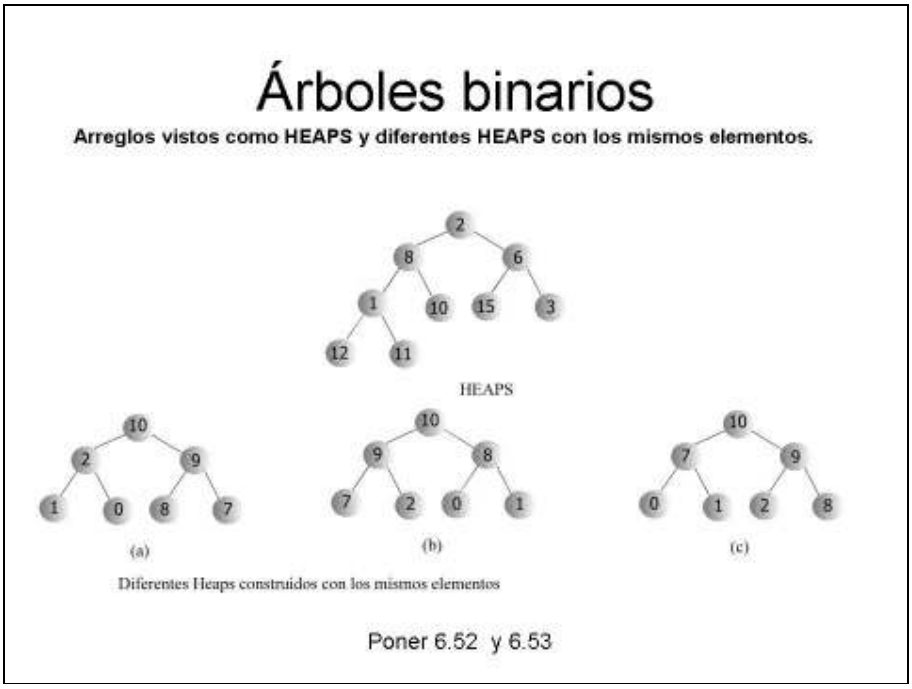
Un árbol parcialmente ordenado es aquél que tiene todas y cada una de sus ramas, consideradas como listas, totalmente ordenadas, ya sea de forma creciente o decreciente.

En relación al heap, se exige una ordenación decreciente, lo que puede dar lugar a una definición más natural de heap: un heap es un árbol binario completo, tal que el valor de su raíz es mayor o igual que las raices de sus hijos, siendo también heaps ambos hijos.

Los árboles en la figura con la (a) Son todas heaps

Los árboles en la figura con la (b) violan la primera regla

Los árboles en la figura con la (b) violan la segunda regla



Los Heaps pueden ser implementados como arreglos, y en ese sentido cada heap es un arreglo, pero desde luego no todos los arreglos pueden ser vistos como heaps.

En algunas situaciones, la más notable es la ordenación heapsort, en donde nosotros necesitamos convertir un arreglo en un heap para ordenar dicho arreglo

Evaluación

Ejercicio 1. ¿Cómo funciona un árbol binario de búsqueda?

Ejercicio 2. ¿Cuáles son y como funcionan los recorridos en preorden, inorden y postorden?

Ejercicio 3. ¿Cómo se realiza la inserción y el borrado en árboles binarios ?

Laboratorio

Desarrollo: Implementar la estructura de datos árbol binario de búsqueda en C# que incluya las funciones:

- Imprimir los elementos del árbol
- Buscar un elemento en el árbol
- Insertar un elemento en el árbol
- Borrar un elemento del árbol
- Recorrido en anchura
- Recorridos preorden, postorden y inorden