Árboles

E. Rivas

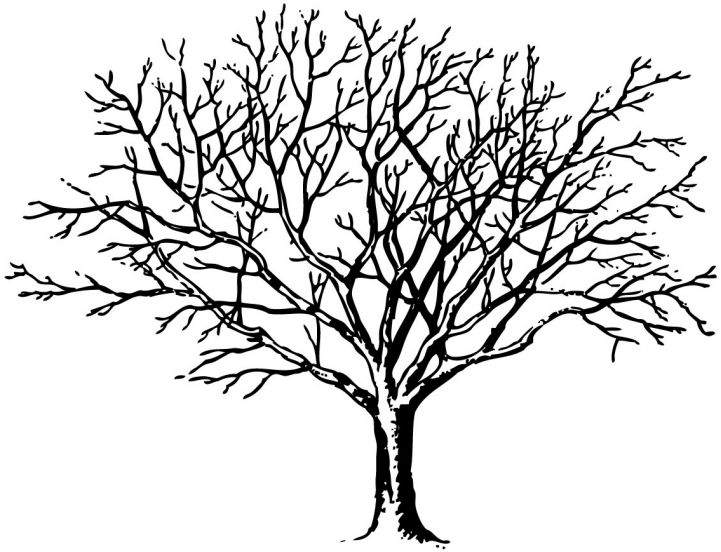
Abril de 2014

Resumen

Árboles

Árboles “enlazados”

Árboles como arreglos



Árboles - definición

árbol. un tipo de dato que implementa una colección de valores, organizados en una forma jerárquica simil a un "árbol".

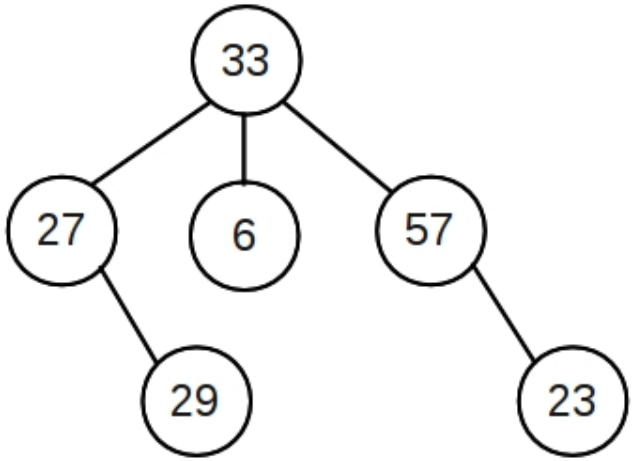
Al igual que en el caso de las listas, hablamos de los elementos de los árboles.

Árboles - definición

árbol. un tipo de dato que implementa una colección de valores, organizados en una forma jerárquica simil a un "árbol".

Al igual que en el caso de las listas, hablamos de los elementos de los árboles.

Ejemplo:



es un árbol de enteros.

Árboles - formalmente

Un árbol se puede definir formalmente utilizando la teoría de grafos.

Árboles - formalmente

Un árbol se puede definir formalmente utilizando la teoría de grafos. Pero tendrán que esperar hasta Complementos matemáticos I para esto.

Árboles - formalmente

Un árbol se puede definir formalmente utilizando la teoría de grafos. Pero tendrán que esperar hasta Complementos matemáticos I para esto.
Sin embargo, podemos notar algunas propiedades que tienen:

- Altura.
- Orden jerárquico (no lineal).

Árboles - formalmente

Un árbol se puede definir formalmente utilizando la teoría de grafos. Pero tendrán que esperar hasta Complementos matemáticos I para esto.
Sin embargo, podemos notar algunas propiedades que tienen:

- Altura.
- Orden jerárquico (no lineal).

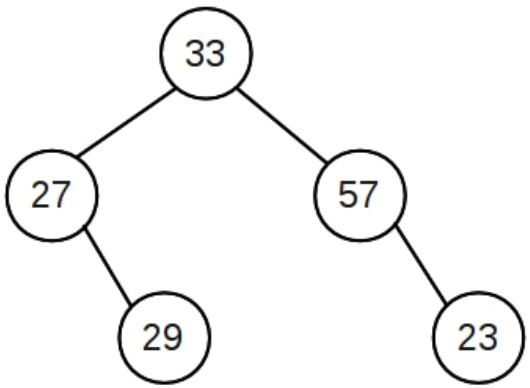
Además, existen operaciones sobre estos árboles:

- Calcular altura.
- Recorrer elementos.
- Agregar elementos.
- Acceder a sus elementos.

Árboles - nomenclatura

Un árbol está compuesto por *nodos*. Cada nodo tiene un único *padre* (excepto el nodo *raíz*, que no tiene ninguno).

Además, si n es el padre de m, entonces m es el *hijo* de n. Si un nodo no tiene hijos, entonces decimos que es una *hoja*.



Un árbol es n-*ario* si cada nodo tiene a lo sumo n hijos. Si un árbol n-ario tiene completos todos sus niveles (a excepción del último donde están todos a izquierda), decimos que es un árbol n-*ario completo*.

Árboles - recorridos

Una particularidad de los árboles, es que al tener una estructura jerárquica, no lineal, tenemos varias maneras de recorrerlos:

- Pre-orden: visitamos el nodo, luego los subárboles de izquierda a derecha.

Árboles - recorridos

Una particularidad de los árboles, es que al tener una estructura jerárquica, no lineal, tenemos varias maneras de recorrerlos:

- Pre-orden: visitamos el nodo, luego los subárboles de izquierda a derecha.
- Post-orden: visitamos los subárboles de izquierda a derecha, luego el nodo.

Árboles - recorridos

Una particularidad de los árboles, es que al tener una estructura jerárquica, no lineal, tenemos varias maneras de recorrerlos:

- Pre-orden: visitamos el nodo, luego los subárboles de izquierda a derecha.
- Post-orden: visitamos los subárboles de izquierda a derecha, luego el nodo.
- Primero por extensión: visitamos los nodos por niveles, de izquierda a derecha.

Árboles - recorridos

Una particularidad de los árboles, es que al tener una estructura jerárquica, no lineal, tenemos varias maneras de recorrerlos:

- Pre-orden: visitamos el nodo, luego los subárboles de izquierda a derecha.
- Post-orden: visitamos los subárboles de izquierda a derecha, luego el nodo.
- Primero por extensión: visitamos los nodos por niveles, de izquierda a derecha.
- (binario) En orden: visitamos primero el subárbol izquierdo, luego el nodo, y luego el subárbol derecho.

Árboles binarios - C

Podemos representar un árbol 2-ario de manera similar a una lista, esta vez un nodo tiene un dato y dos punteros a otros nodos: hijo izquierdo y derecho.

```
struct _BTree {
  int data;
  struct _BTree *left, *right;
};

typedef struct _BTree BTree;
```

Un árbol lo pensamos como un puntero al raíz (`BTree *`).

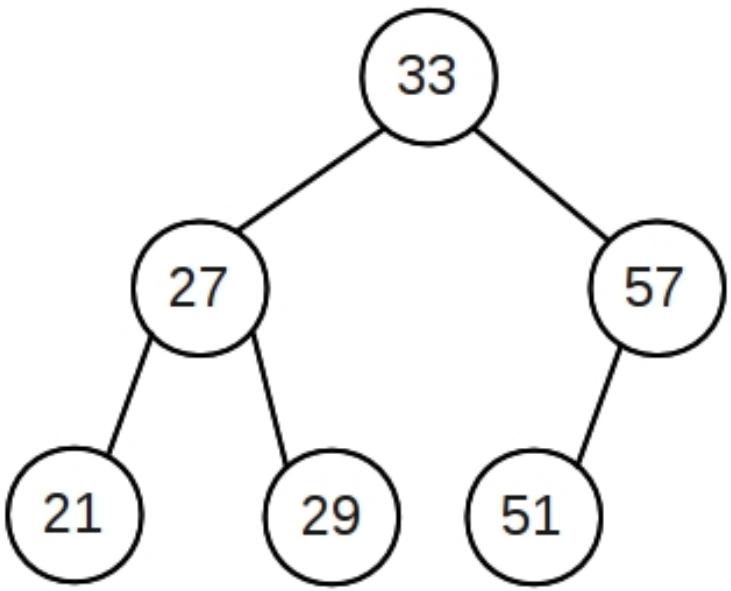
Árboles binarios - creación

Podemos crear un árbol creando los nodos correspondientes, y luego uniendolos.

```
BTree n1, n2, n3;

n1.data = 12;
n1.left = &n2;
n1.right = NULL;

n2.data = 99;
n2.left = &n3;
n2.right = NULL;

n3.data = 37;
n3.left = NULL;
n3.right = NULL;
```

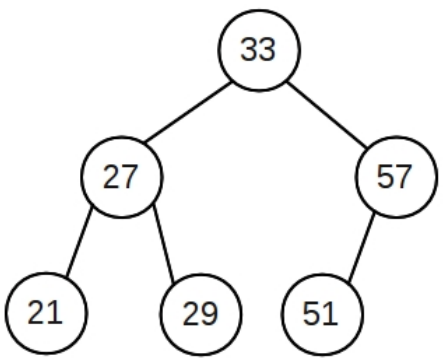
Árboles binarios ordenados - definición

árbol binario ordenado. un árbol binario ordenado tiene la propiedad de que el elemento de cada nodo es mayor que los elementos del subárbol izquierdo y menor que los elementos del subárbol derecho.



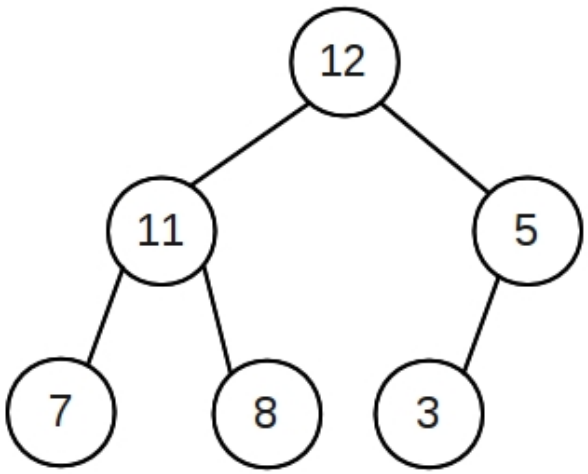
Árboles binarios ordenados - ejercicio

- Implementar una función **BTree *append(BTree *, int)** que toma una árbol binario ordenado, y un entero. La función crea un nuevo nodo cuyo dato es el entero pasado como parámetro, y agrega un nuevo nodo en el árbol, según corresponda para que el árbol siga siendo un árbol binario ordenado.
- Construya el siguiente árbol usando la función anterior.



Árboles binarios completos - idea

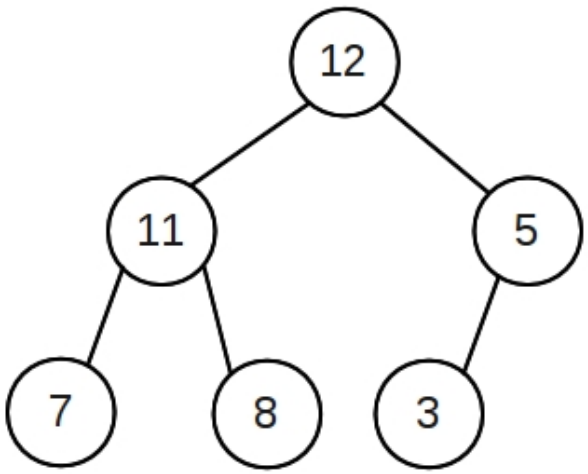
Los árboles binarios completos podemos representarlos con arreglos:

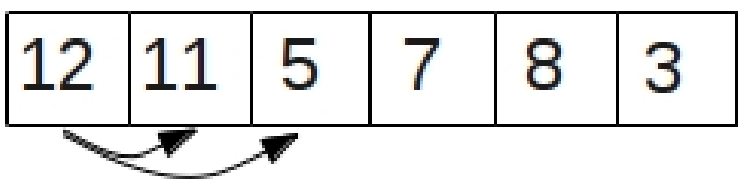


12	11	5	7	8	3

Árboles binarios completos - idea

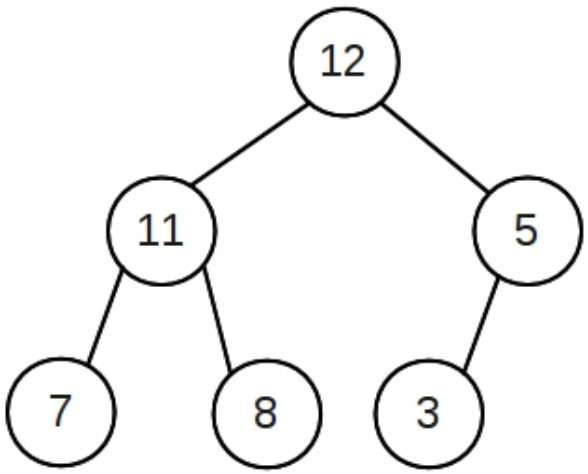
Los árboles binarios completos podemos representarlos con arreglos:

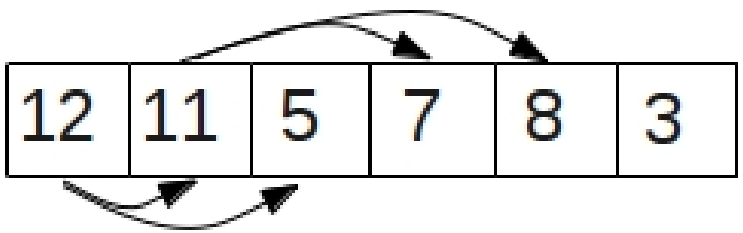




Árboles binarios completos - idea

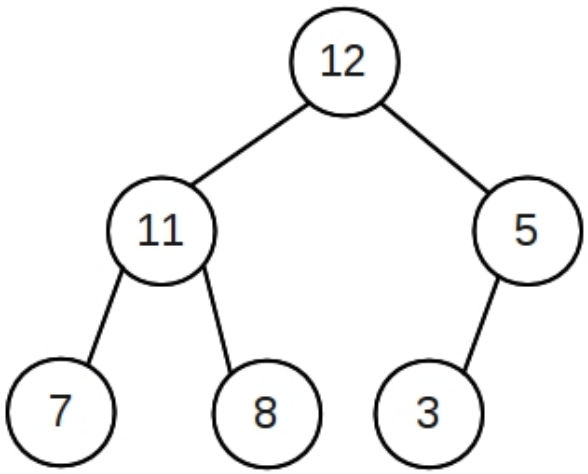
Los árboles binarios completos podemos representarlos con arreglos:

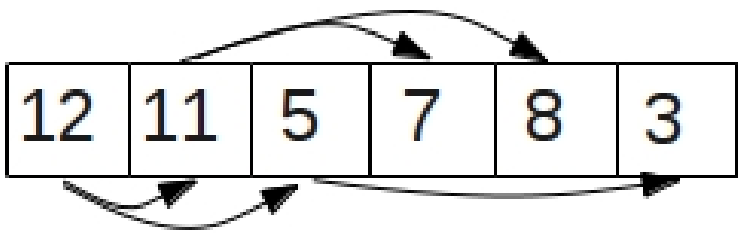




Árboles binarios completos - idea

Los árboles binarios completos podemos representarlos con arreglos:





Árboles binarios completos - C

Por ejemplo, utilizando una estructura, podemos representar un árbol de hasta 127 elementos:

```
struct _BCTree {
  int data[127];
  int nelems;
};
typedef struct _BCTree BCTree;
```

En `nelems` llevamos la cantidad de elementos utilizados.

Árboles binarios completos - hijos/padres

Si representamos a un árbol binario completo de esta manera, si un nodo está en la posición i del arreglo entonces:

- Su hijo izquierdo está en la posición $2i$.
- Su hijo derecho está en la posición $2i + 1$.
- Su padre tiene está en la posición $\frac{i}{2}$.

La raíz siempre está en la posición 1 (índice 0).

Árboles binarios completos - agregado

En un árbol binario completo podemos siempre considerar una única posición siguiente: ahí agregamos elementos.

```
BCTree *insert(BCTree *t, int n) {
  t->data[t->nelems] = n;
  t->nelems++;
  return t;
}
```

Árboles binarios completos - agregado

En un árbol binario completo podemos siempre considerar una única posición siguiente: ahí agregamos elementos.

```
BCTree *insert(BCTree *t, int n) {
  t->data[t->nelems] = n;
  t->nelems++;
  return t;
}
```

¿Qué puede fallar con esta función?

Árboles binarios completos - visita

Para esta implementación es sencillo visitar los nodos por niveles:

```
void bctree_print(BCTree *t) {
  int i;
  for (i = 0; i < t->nelems; i++) {
    printf("%d\t", t->data[i]);
  }
}
```

esta función imprime los nodos en orden por niveles.