Estructuras y Tipos de Datos

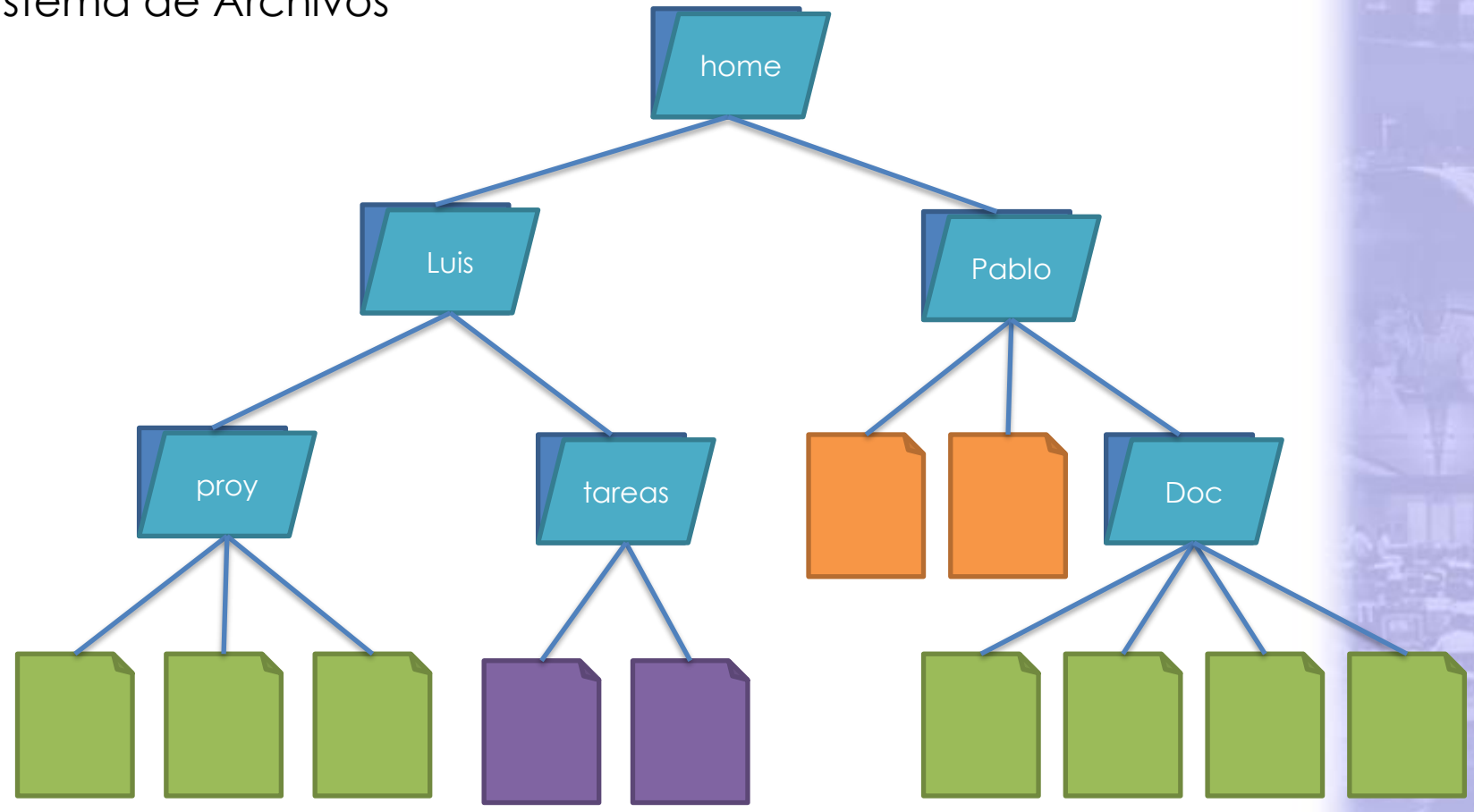
Tipo Abstracto de Datos Arboles

Tipo de Dato Árbol

- ✓ Representaciones estáticas y representaciones dinámicas.
- ✓ El tipo de dato “árbol”.
- ✓ Las variantes: árboles binarios, árboles binarios de búsqueda, árboles AVL.

Motivación

Sistema de Archivos



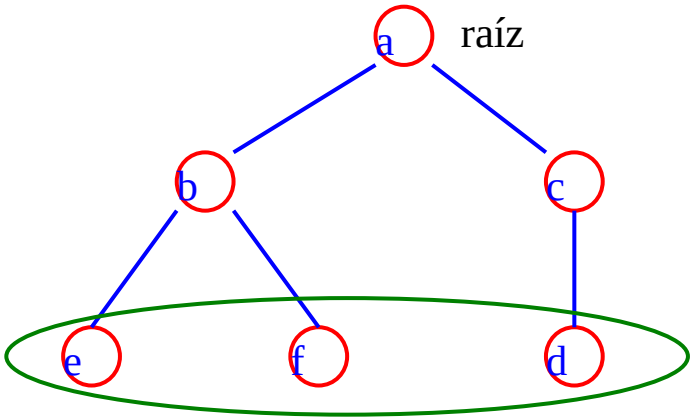
Árboles

Def 1: Un árbol nos define una organización jerárquica entre los elementos de un conjunto.

Def 2:Un grafo G se dice que es conexo si para cada par de vértices pertenecientes a V (conjunto de vértices) existe un camino en G. Por lo tanto, podemos decir que un árbol es un grafo conexo y sin ciclos.

Def 3: Una colección de elementos llamados nodos, en donde uno de ellos se conoce como raíz, y una relación de padre-hijo que origina una estructura jerárquica entre los nodos.

Ejemplo



b es padre (ancestro) de e
e es hijo de b
a es padre de c y b

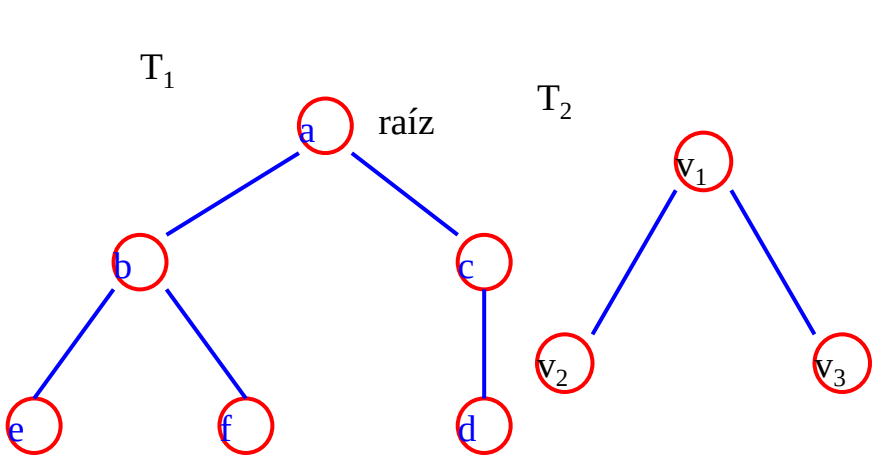
Hojas = nodos sin hijos o descendientes

Arbol, definición recursiva

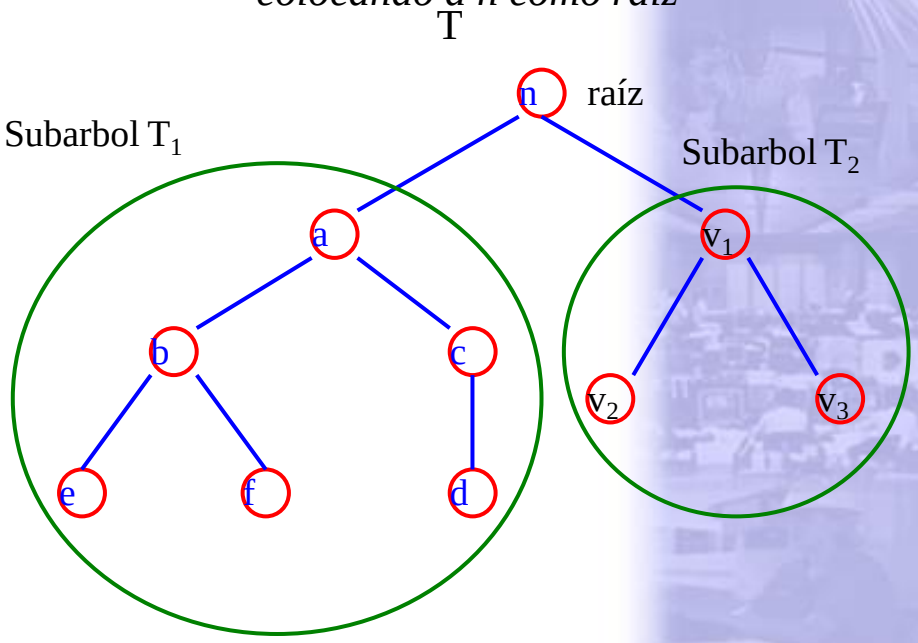
1. *Un nodo es un árbol.*
2. *Sea n y T_1, T_2, T_3, ... , T_k árboles cuyas raíces son n_1, n_2, n_3, ... , n_k. Podemos crear un nuevo árbol colocando a n como padre de los nodos n_1, n_2, n_3, ... , n_k. En este árbol, n es la raíz, T_1, T_2, T_3, ... , T_k son subárboles de la raíz y n_1, n_2, n_3, ... , n_k son hijos de n.*

Ejemplo

Sean T_1 y T_2 dos árboles



Se forma árbol T con T_1 y T_2, colocando a n como raíz



Grado de un vértice

Es el número de descendientes de un vértice (grado de salida del vértice).

Profundidad de un nodo

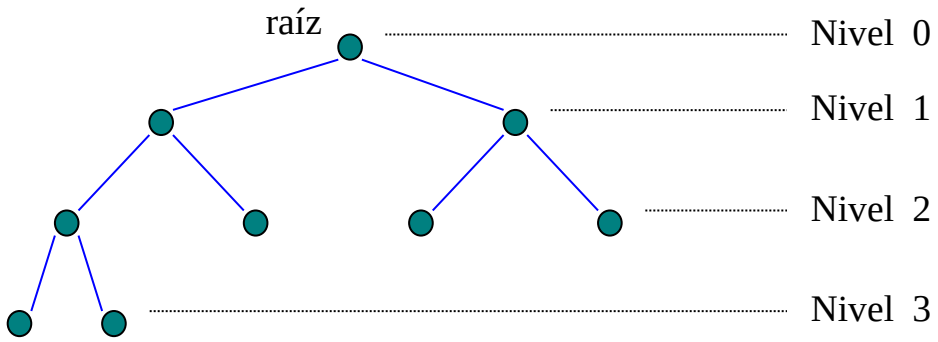
Es la longitud del camino desde la raíz hasta el nodo.

Nivel en un árbol

Definición recursiva:

1. *La raíz tiene nivel 0.*
2. *Los descendientes de un vértice a nivel i tendrán nivel i+1.*

Ejemplo:



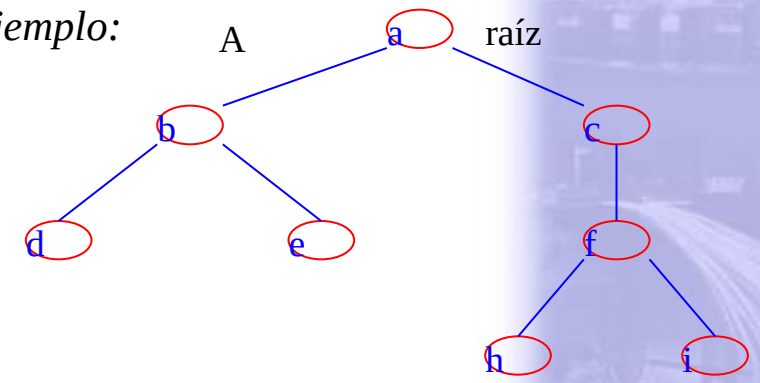
Arbol binario

Es aquel en cada nodo del árbol tiene a lo sumo 2 hijos

Longitud de un camino

Es el número de nodos que contiene el camino -1.

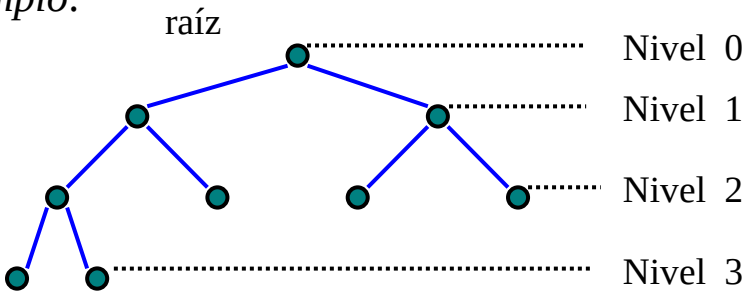
Ejemplo:



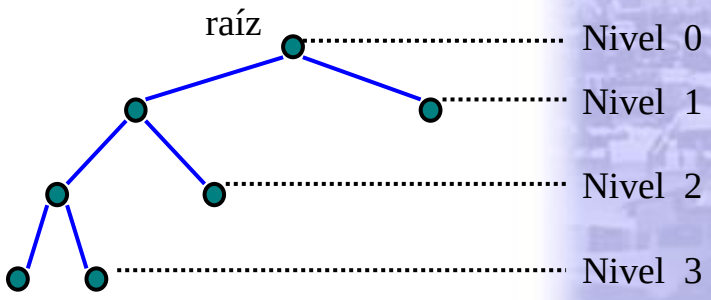
Arbol binario completo

Un árbol está completo, si el árbol con altura k tiene desde el nivel 0 hasta el nivel k-1 completos, es decir, el nivel j $\in$ [0, k-1] tiene 2^j nodos. Los hijos en el nivel k están lo más a la izquierda posible.

Ejemplo:



Arbol A: Arbol Binario con Niveles Completos

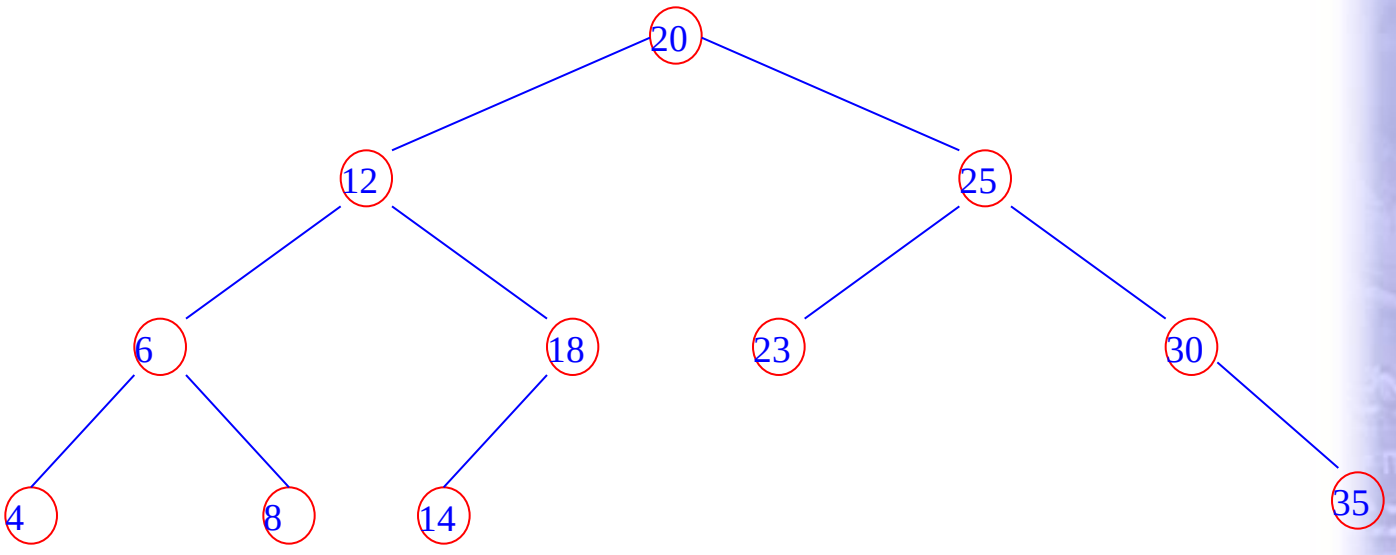


Arbol B: Arbol Binario pero los niveles no están completos

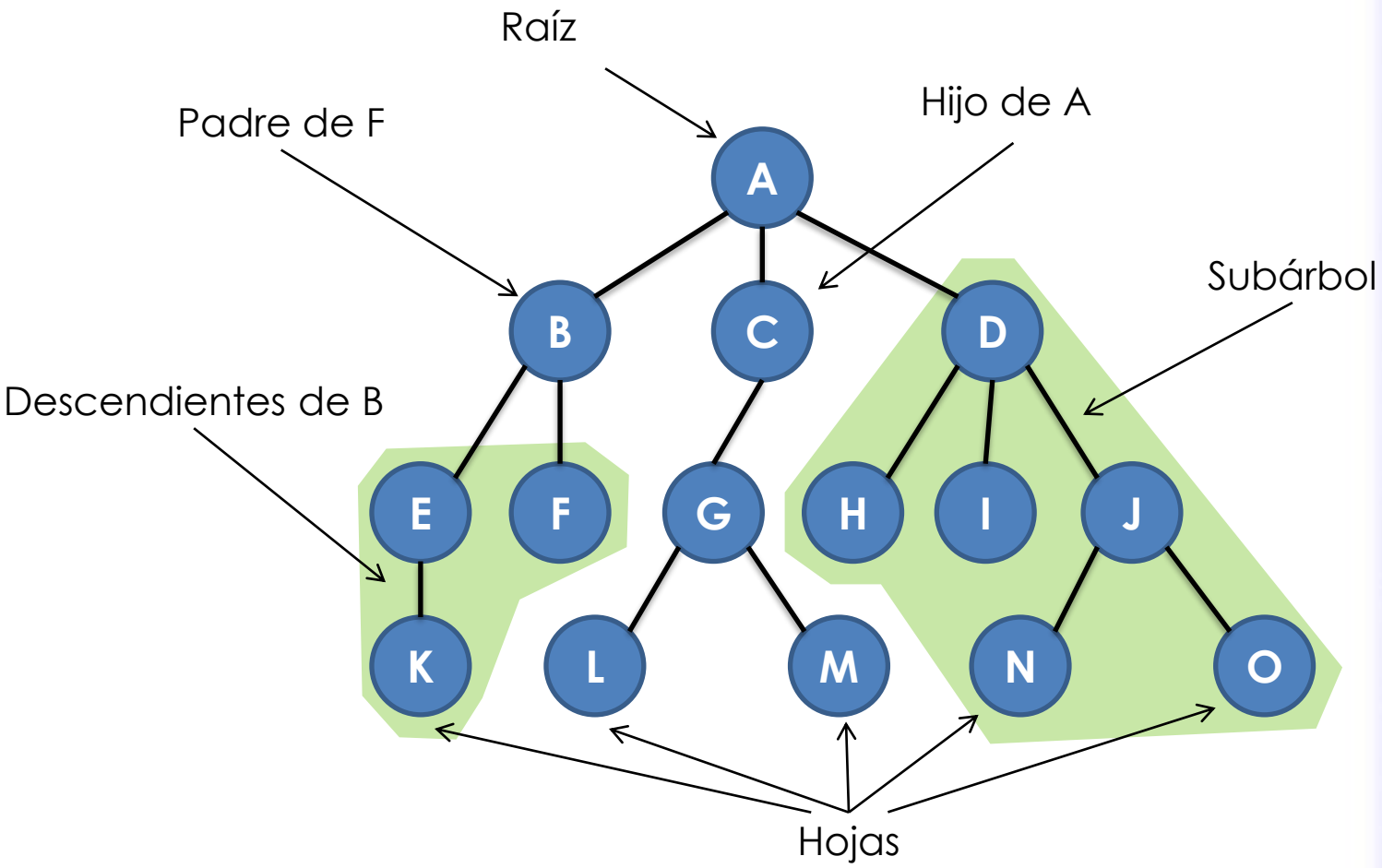
Árbol de Búsqueda Binaria

Es un árbol binario en donde el subarbol izquierdo de un nodo es menor que el padre y el subarbol derecho es mayor igual al padre.

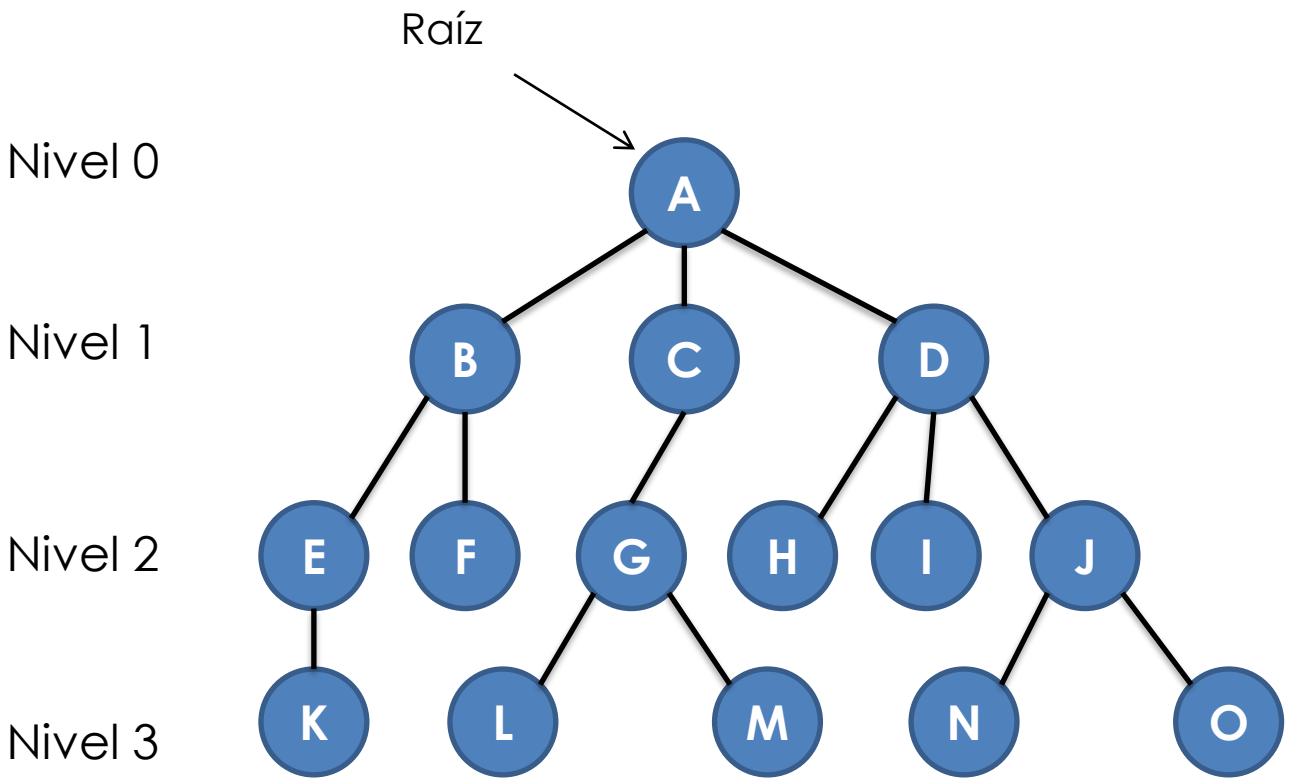
Ejemplo:



Árboles


Raíz
Padre de F
Hijo de A
Subárbol
Descendientes de B
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
Hojas

Árboles



Árbol con 15 nodos y de altura 3

Árboles

Orden de recorrido

Una operación comúnmente realizada sobre árboles consiste en recorrer todo sus nodos. Esto requiere que se imponga una ordenación sobre los nodos del árbol y nos llevas a árboles ordenados.

En un árbol ordenado los hijos de un nodo tiene un orden lineal especifico, se puede utilizar convención que están ordenados de izquierda a derecha y esta ordenación puede extenderse a los subárboles. Si un subárbol T_1esta a izquierda de un subárbol T_2, entonces todo nodo de T_1 esta a la izquierda de todo nodo de T_2.

La operación que se realiza en cada nodo durante un recorrido se conoce como visita.

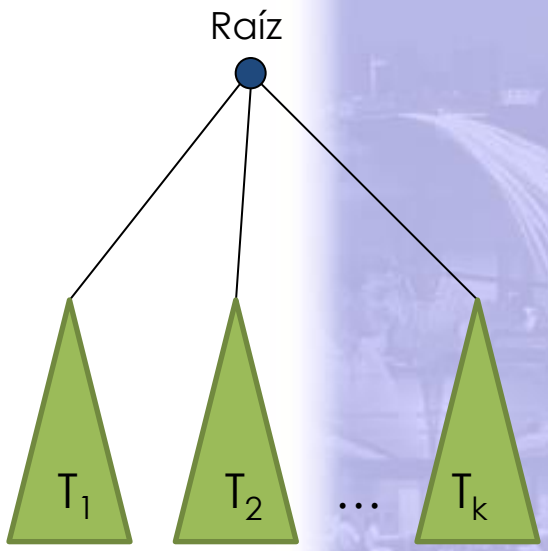
Árboles

Recorridos

Preorden: Se visita primero el nodo raíz, seguido por el recorrido en preorden de los subarboles de la raíz en el orden, es decir de izquierda a derecha (T_1,T_2, … , T_k).

Inorden: Se visita primero los nodos del primer subárbol (T_1), usando un recorrido Inorden, seguido por la visita a la raíz y después el recorrido en Inorden del resto de los subarboles (T_2,T_3, … , T_k).

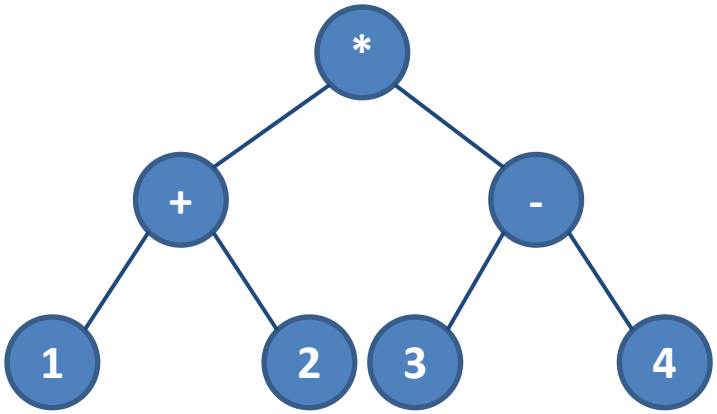
Postorden: Se realiza el recorrido en postorden de los subarboles de la raíz en orden (T_1,T_2, … , T_k) seguido por la visita a la raíz.



Árboles

Recorridos

Un ejemplo muy común es la representación de expresiones.



((1+2)*(3-4))

Preorden:

* + 1 2 - 3 4

Inorden:

1 + 2 * 3 - 4

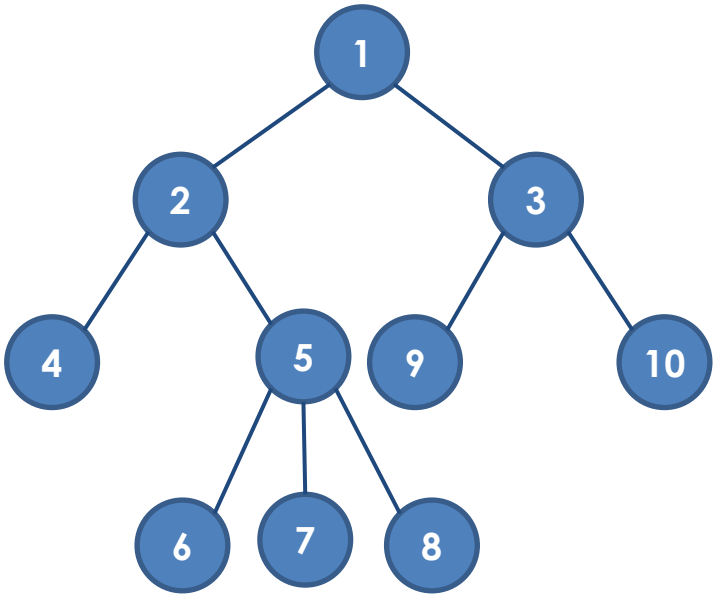
Postorden:

1 2 + 3 4 - *

Árboles

Representaciones estáticas

Matriz de Adyacencia

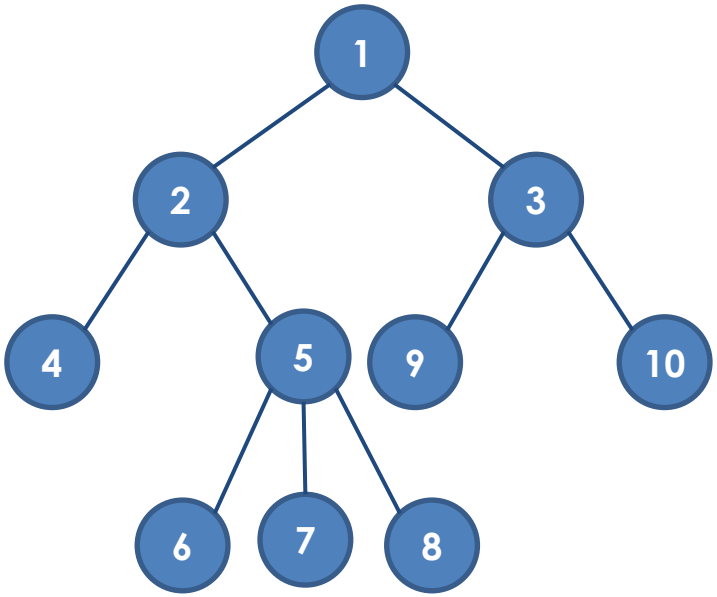


	1	2	3	4	5	6	7	8	9	10
1		X	X							
2				X	X					
3									X	X
4										
5						X	X	X		
6										
7										
8										
9										
10										

Árboles

Representaciones estáticas

Apuntadores al padre

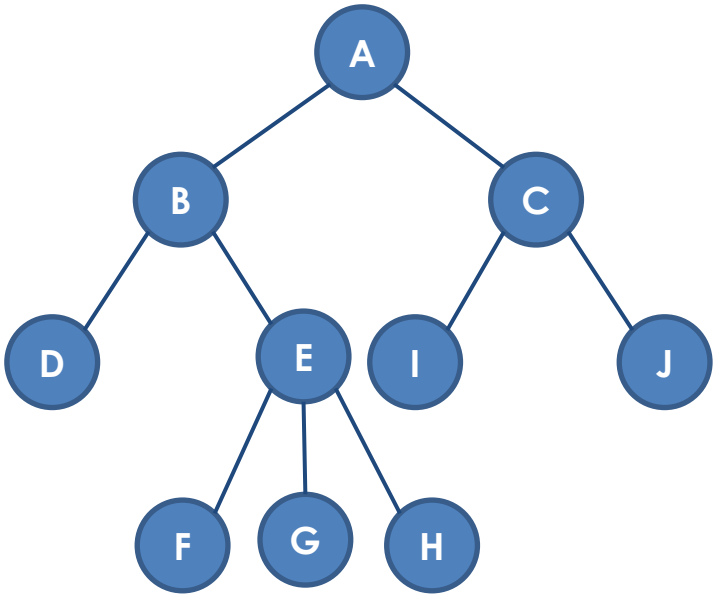


1	2	3	4	5	6	7	8	9	10
0	1	1	2	2	5	5	5	3	3

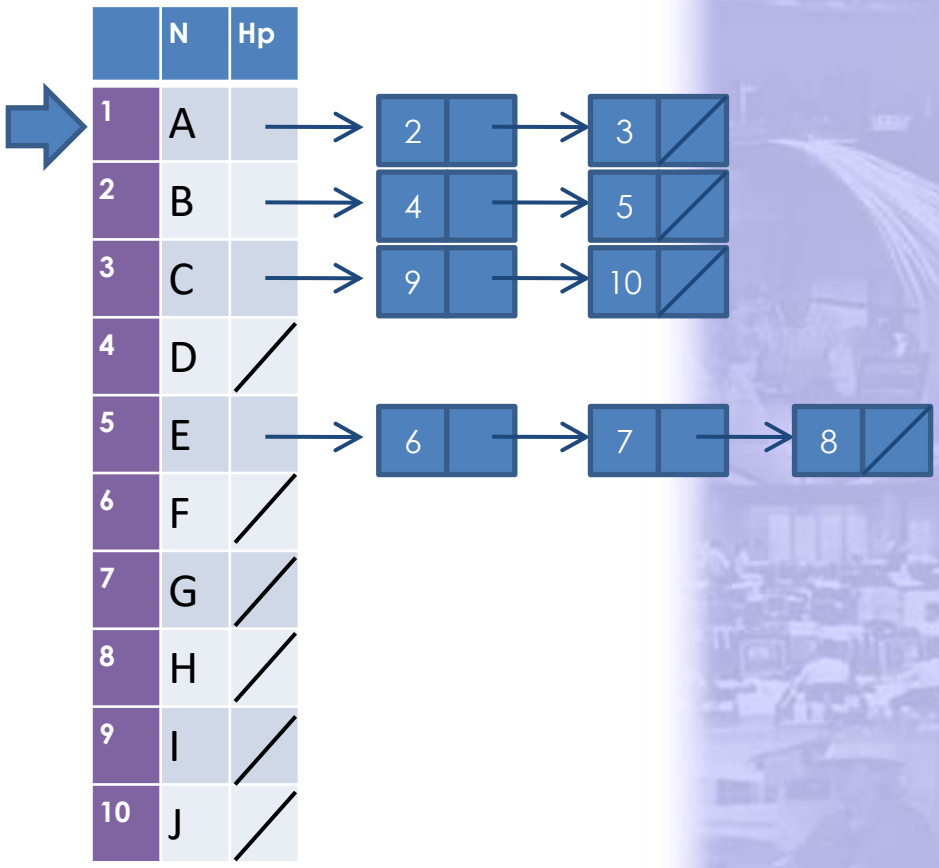
Árboles

Representaciones dinámicas

Listas de Hijos



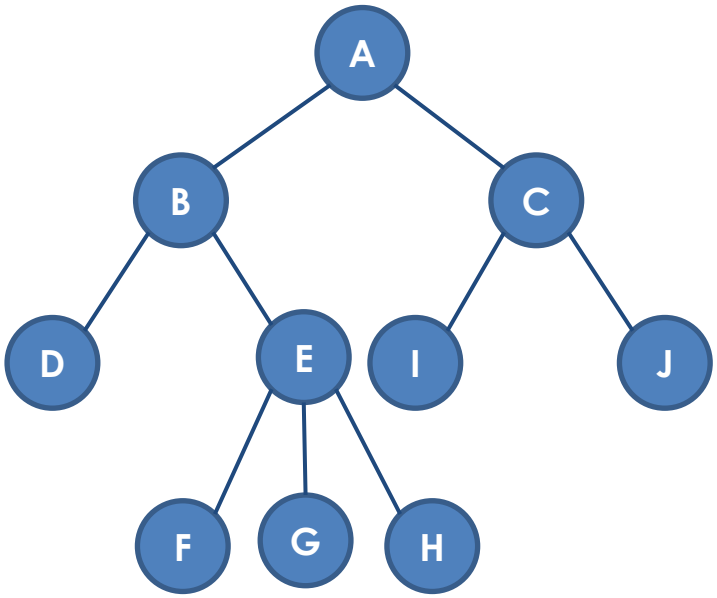
Encabezado

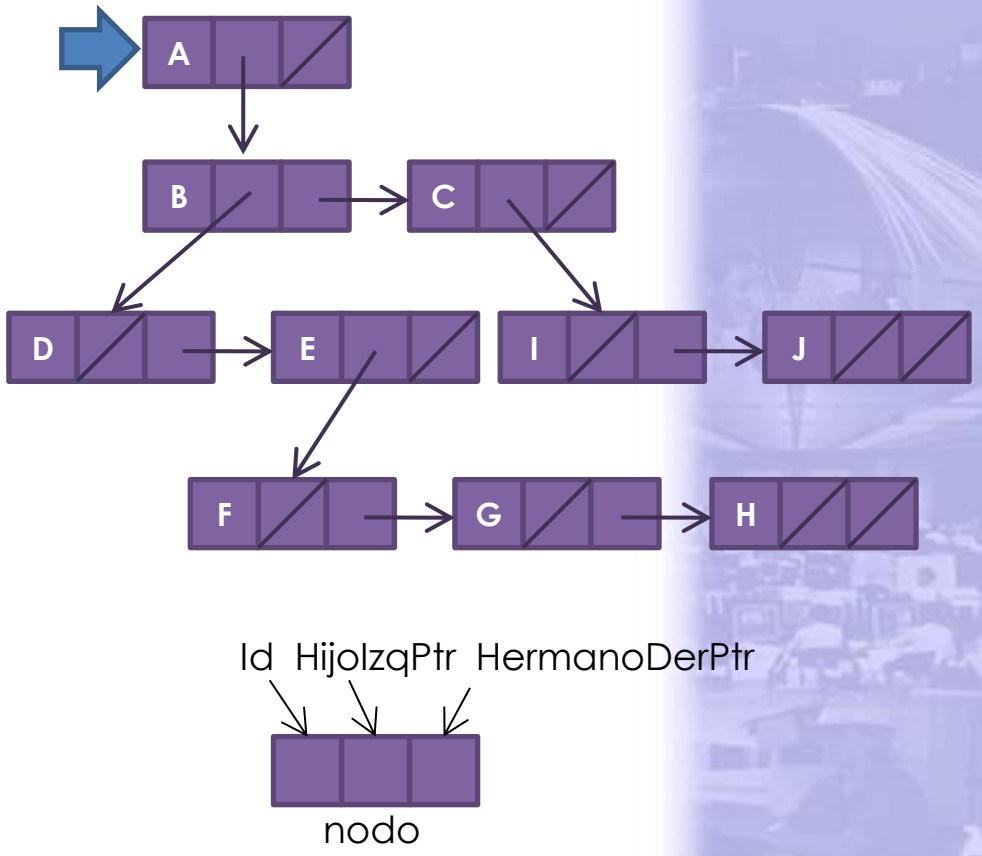


Árboles

Representaciones dinámicas

Estructuras Multienlazadas



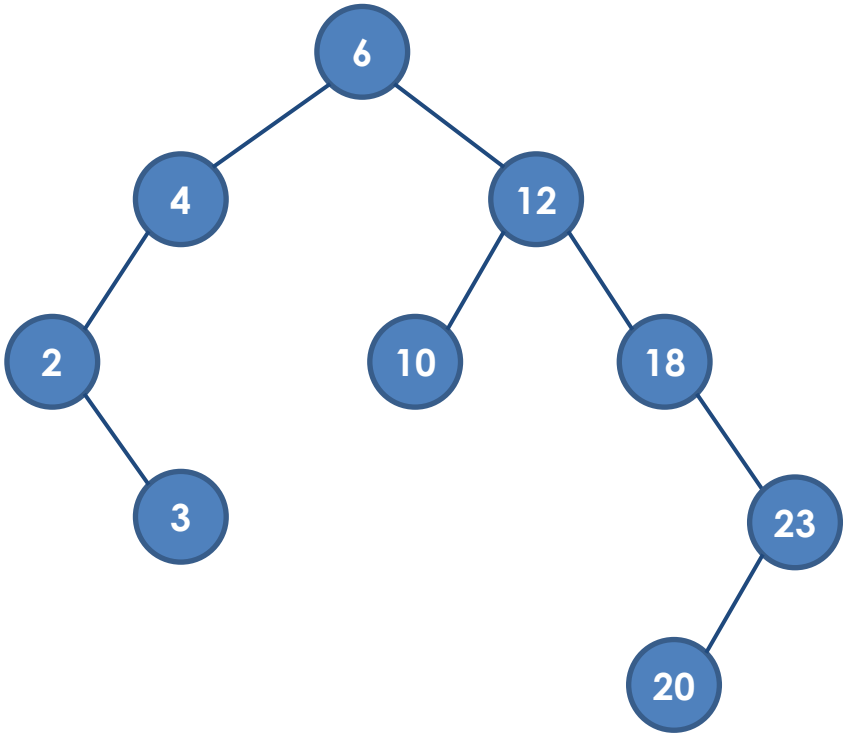


Árboles Binarios

Un árbol binario, es un tipo especial de árbol en el cual cada nodo tiene dos hijos, un hijo o ningún hijo. Es decir todo nodo tiene un máximo de dos hijos: el izquierdo y el derecho.

Árbol Binario de Búsqueda, es una estructura cuyos elementos están organizados en un árbol binario, con nodos con datos enteros o que contenga una clave que pueda ser comparada, y los nodos están organizados de tal forma que para cualquier nodo su valor siempre es mayor que los valores de los nodos de su descendencia del subárbol izquierdo y menor que los valores de los nodos de su descendencia del subárbol derecho.

Árboles Binarios de Búsqueda



Árboles Binarios de Búsqueda

Operaciones:

- Insertar
- Buscar
- Mínimo
- Máximo
- Antecesor
- Predecesor
- Eliminar
- Recorrer (Inorden, Preorden, Postorden)
- Imprimir
- Altura …

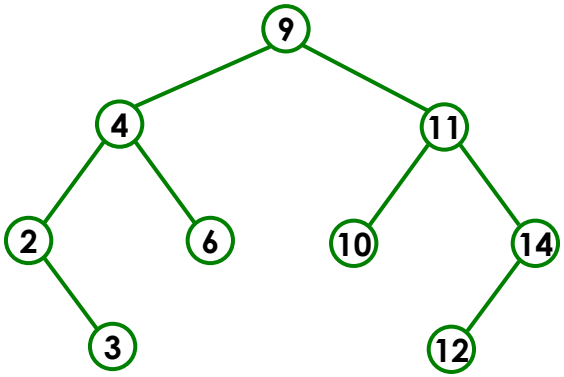
Árboles Binarios de Búsqueda

Buscar un elemento:

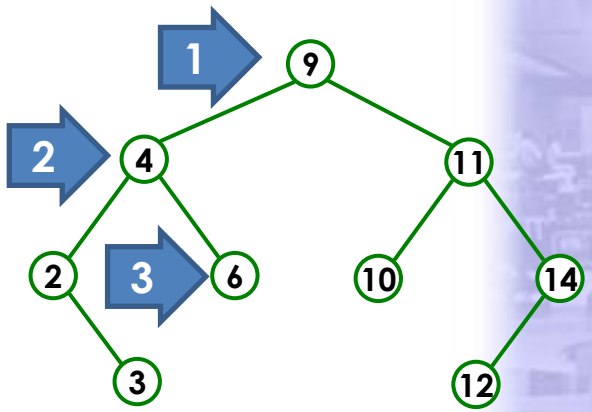
Si el árbol está vacío el elemento no está.

Si el elemento coincide con el de la raíz listo.

Si el elemento es menor que el de la raíz se busca en el subárbol izquierdo, caso contrario se busca en el subárbol derecho.



Buscar 6



Operaciones sobre Árboles Binarios de Búsqueda

Operaciones de Consulta

Se comprueba si la raíz contiene la clave, si la clave es menor que la raíz se busca en el subárbol izquierdo y en caso contrario se busca en el subárbol derecho.

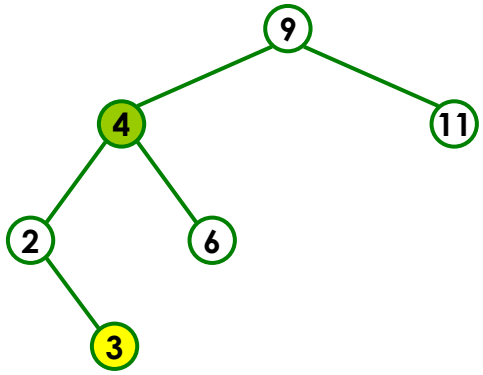
```
Buscar(ABB abb, CLAVE clave)
        v ← Raiz(abb)
        Si v = NULO o Clave(v) = clave Entonces
                Devolver v
        Si Clave < Clave(v) Entonces
                Devolver Buscar(Izquierdo(v),clave)
        Sino
                Devolver Buscar(Derecho(v),clave)
```

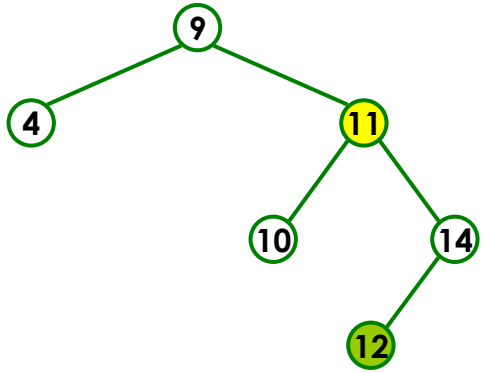
Operaciones sobre Árboles Binarios de Búsqueda

Operaciones de Máximo y Mínimo

Clave mínima en el extremo izquierdo del árbol

Clave máxima en el extremo derecho del árbol



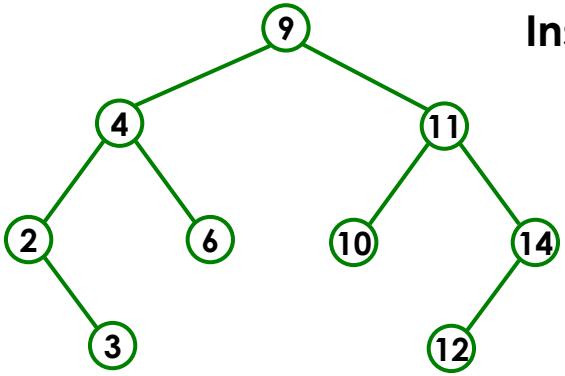


Árboles Binarios de Búsqueda

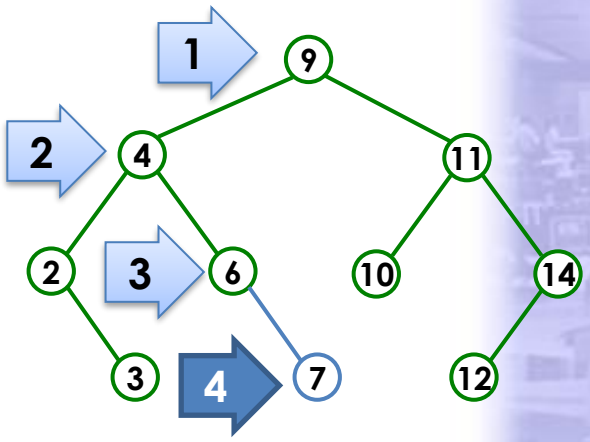
Agregar un elemento:

Si el elemento está en el árbol no se agrega.

Si el elemento no está, éste se inserta a continuación del último nodo visitado en la búsqueda, respetando la regla de construcción del árbol, a la izquierda o derecha del nodo dependiendo si el valor a agregar es menor o mayor respectivamente que el del nodo.



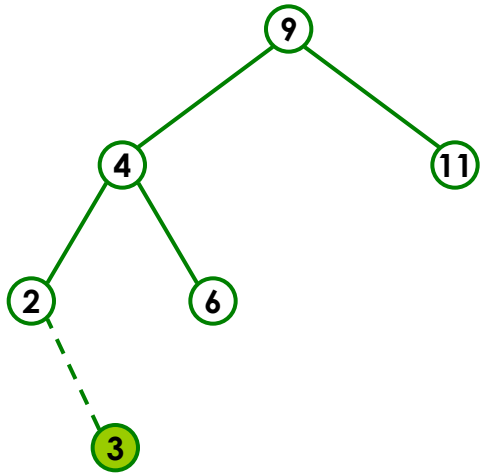
Insertar 7



Operaciones sobre Árboles Binarios de Búsqueda

Insertar

Si la clave no esta (prerrequisito para la inserción) la operación de buscar encontrará un puntero nulo en una de las hojas del árbol. Esta es la posición donde debe ser insertado en nuevo nodo.



Operaciones sobre Árboles Binarios de Búsqueda

Eliminar

Eliminar un nodo sin hijos, se establece el apuntador adecuado a Null en el padre y se libera el nodo.

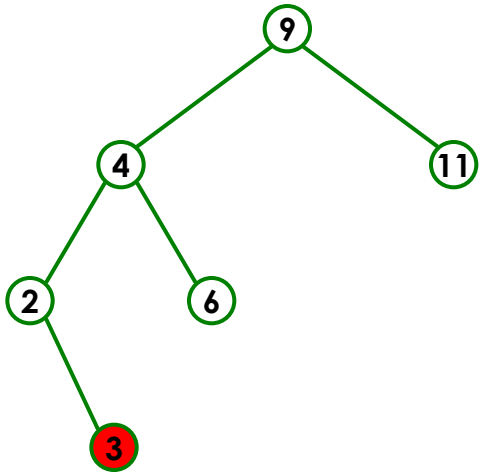
Eliminar un nodo con un hijo, se establece el apuntador adecuado en el padre al hijo del nodo que se esta eliminando y se libera el nodo

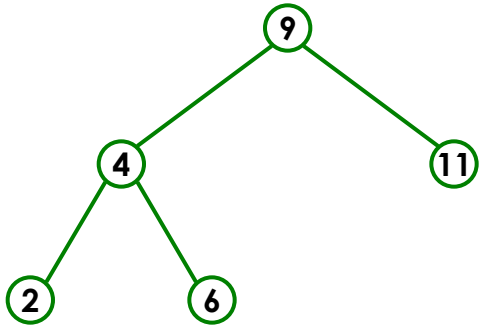
Eliminar un nodo con dos hijos, se reemplaza por su predecesor (en el recorrido inorden) y se elimina el nodo que dejo libre el predecesor, que se reduce a uno de los dos casos anteriores.

Operaciones sobre Árboles Binarios de Búsqueda

Eliminar

Eliminar un nodo sin hijos

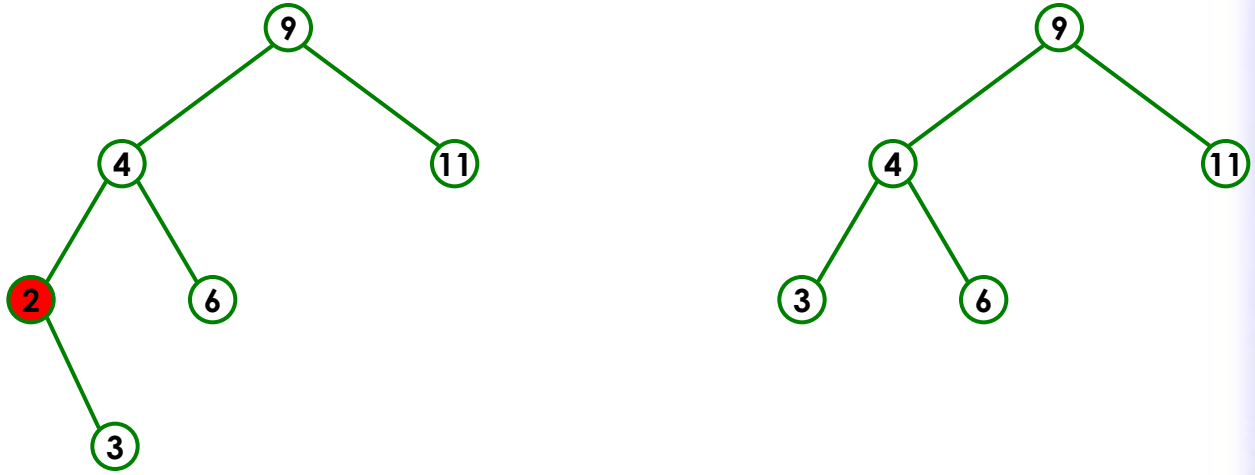




Operaciones sobre Árboles Binarios de Búsqueda

Eliminar

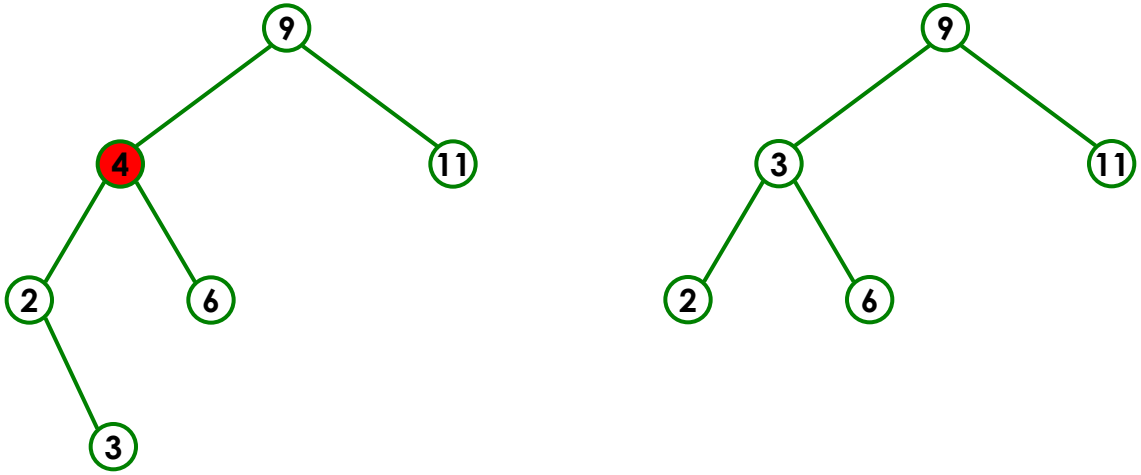
Eliminar un nodo con un hijo



Operaciones sobre Árboles Binarios de Búsqueda

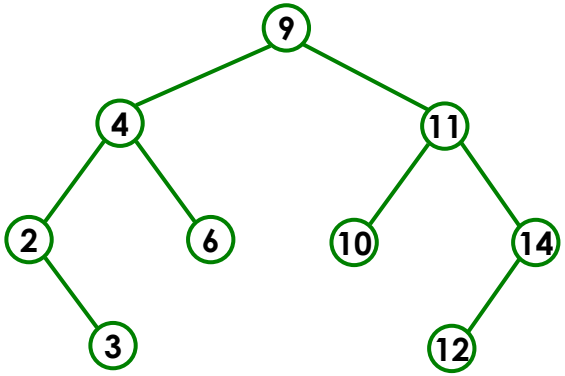
Eliminar

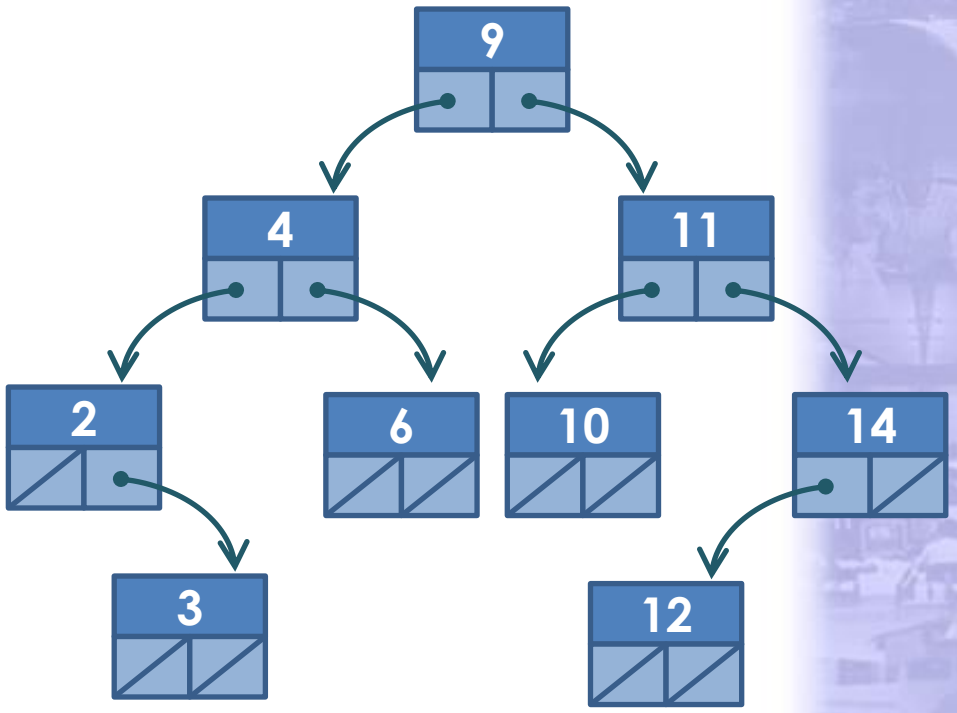
Eliminar un nodo con dos hijos



Implementación de Árboles Binarios de Búsqueda

Representaciones dinámicas





Arboles

Data structures: Introduction to Trees

https://www.youtube.com/watch?v=qH6yxkw0u78

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Estructura para la implemetacion de arbol binario
 */

struct nodo{
     int elemento;
     struct nodo *izq,*der;
} nodo_t;
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Inserta un elemento en un arbol binario
 */

int inserta(nodo_t **arbolbp,int elemento){
     if(*arbolbp==NULL){
          *arbolbp=(nodo_t *)malloc(sizeof(nodo_t));
          (*arbolbp)->elemento=elemento;
          (*arbolbp)->izq=(*arbolbp)->der=NULL;
          return 1;
     }
     if(elemento==(*arbolbp)->elemento)
          return 0;
     if(elemento<(*arbolbp)->elemento)
          return inserta(&((*arbolbp)->izq),elemento);
     else
          return inserta(&((*arbolbp)->der),elemento);
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Busca un elemento en un arbol binario
 */

int buscar(nodo_t *arbolbp, int elemento){
     if(arbolbp==NULL)
          return 0;
     if(elemento==arbolbp->elemento)
          return 1;
     if(elemento < arbolbp->elemento)
          return buscar(arbolbp->izq,elemento);
     else
          return buscar(arbolbp->der,elemento);
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *    Busca el menor elemento en un arbol binario
 */

int menor(nodo_t *arbolbp, int *elemento){
      if(arbolbp==NULL)
            return 0;
      while(arbolbp->izq != NULL)
            arbolbp=arbolbp->izq;
      *elemento = arbolbp->elemento;
      return 1;
}

/*
 *    Busca el mayor elemento en un arbol binario
 */

int mayor(nodo_t *arbolbp, int *elemento){
      if(arbolbp==NULL)
            return 0;
      while(arbolbp->der != NULL)
            arbolbp=arbolbp->der;
      *elemento = arbolbp->elemento;
      return 1;
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Cuenta el numero de nodos en un arbol binario
 */

int nro_nodos(nodo_t *arbolbp){
     if(arbolbp==NULL)
          return 0;
     return nro_nodos(arbolbp->izq)+nro_nodos(arbolbp->der)+1;
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *  Elimina todos los nodos de un arbol binario
 */

void borrar_arbol(nodo_t *arbolbp){
    if(arbolbp==NULL)
        return;
    borrar_arbol(arbolbp->izq);
    borrar_arbol(arbolbp->der);
    free(arbolbp);
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 * Borra un nodo de un arbol binario,
 * nota primera llamada raiz=borrar(raiz,clave);
 */

nodo_t *borrar(nodo_t *arbolbp, int elemento){
    nodo_t *ap;

    if(arbolbp == NULL)
        return NULL;
    if(elemento==arbolbp->elemento){      // se encontro el elemento a borrar
        if(arbolbp->izq == arbolbp->der){ // el nodo tiene ambos subarboles
            free(arbolbp);                // vacios, retorna un arbol vacio
            return NULL;
        }
        if(arbolbp->izq==NULL){     // el nodo tiene el subarbol izquierdo
            ap=arbolbp->der;        // vacio, retorna el subarbol derecho
             free(arbolbp);
            return ap;
        }

// Continua Siguiente Lamina
```

Implementación de Árboles Binarios de Búsqueda

```
// Viene de la lamina Anterior

        if(arbolbp->der==NULL){     // el nodo tiene el subarbol derecho
            ap=arbolbp->izq;        // vacio, retorna el subarbol izquierdo
             free(arbolbp);
            return ap;
        }
        ap=arbolbp->izq;                    // El nodo tiene ambos subarboles,
        while(ap->der != NULL)              // reeemplaza el elemento por su
            ap=ap->der;                     // predecesor y borra el nodo que
        arbolbp->elemento = ap->elemento; // contenia al predecesor
        arbolbp->izq=borrar(arbolbp->izq,ap->elemento);
        return(arbolbp);
    }
    if(elemento < arbolbp->elemento)
        arbolbp->izq=borrar(arbolbp->izq,elemento);
    else
        arbolbp->der=borrar(arbolbp->der,elemento);
    return arbolbp;
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *       Calcula la altura de un arbol binario
 */

int h_arbol(nodo_t *arbolbp){
        int hi=0,hd=0;

        if(arbolbp==NULL)
                return 0;
        if(arbolbp->izq != NULL)
                hi=1+h_arbol(arbolbp->izq);
        if(arbolbp->der != NULL)
                hd=1+h_arbol(arbolbp->der);
        if(hd>hi)
                return hd;
        return hi;
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Recorrido inorden de un arbol binario
 */

void inorden(nodo_t *arbolbp){
     if(arbolbp==NULL)
          return;
     inorden(arbolbp->izq);
     printf("%d ",arbolbp->elemento);
     inorden(arbolbp->der);
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Recorrido preorden de un arbol binario
 */

void preorden(nodo_t *arbolbp){
     if(arbolbp==NULL)
          return;
     printf("%d ",arbolbp->elemento);
     preorden(arbolbp->izq);
     preorden(arbolbp->der);
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Recorrido postorden de un arbol binario
 */

void postorden(nodo_t *arbolbp){
     if(arbolbp==NULL)
          return;
     postorden(arbolbp->izq);
     postorden(arbolbp->der);
     printf("%d ",arbolbp->elemento);
}
```

Implementación de Árboles Binarios de Búsqueda

```
/*
 *   Imprime un arbol binario
 */

void imprime(nodo_t *arbolbp,int nivel){
     int i;

     if(arbolbp==NULL)
          return;
     imprime(arbolbp->der,nivel+1);
     for(i=0;i<nivel;i++)
          printf("   ");
     printf("%d\n",arbolbp->elemento);
     imprime(arbolbp->izq,nivel+1);
}
```

Implementación de Árboles Binarios de Búsqueda

```
int main(void){
	nodo_t *arbol=NULL;
	int e;

	inserta(&arbol,6);
	inserta(&arbol,4);
	inserta(&arbol,12);
	inserta(&arbol,2);
	inserta(&arbol,10);
	inserta(&arbol,18);
	inserta(&arbol,3);
	inserta(&arbol,23);
	inserta(&arbol,20);
	imprime(arbol,1);
	printf("\nNro de elementos:%d",nro_nodos(arbol));
	printf("\nAltura:%d",h_arbol(arbol));
	printf("\nBuscar Elemento 3:%d",buscar(arbol,3));
	printf("\nBuscar Elemento 9:%d",buscar(arbol,9));
	if(menor(arbol,&e))
		printf("\nmenor:%d",e);
	if(mayor(arbol,&e))
		printf("\nmayor:%d",e);
	printf("\nEliminar Elemento: 6\n");
	arbol=borrar(arbol,6);
	imprime(arbol,1);
	printf("\nInorden:");
	inorden(arbol);
	printf("\nPreorden:");
	preorden(arbol);
	printf("\nPostorden:");
	postorden(arbol);
	borrar_arbol(arbol);
	return 0;
}
```

```
        23
          20
      18
    12
      10
  6
    4
      3
    2
Nro de elementos:9
Altura:4
Buscar Elemento 3:1
Buscar Elemento 9:0
menor:2
mayor:23
Eliminar Elemento: 6
        23
          20
      18
    12
      10
  4
      3
    2
Inorden:2 3 4 10 12 18 20 23
Preorden:4 2 3 12 10 18 23 20
Postorden:3 2 10 20 23 18 12 4
```

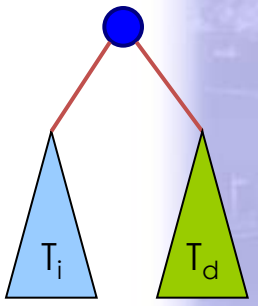
Árboles Binarios de Búsqueda Equilibrado

En el caso de árboles binarios de búsqueda, las operaciones de inserción, búsqueda y eliminar, el peor caso requeriría O(n). Esto del hecho que en el caso peor la altura del árbol de n nodos es n-1. Y esto se produce cuando los elementos se insertan en el árbol de orden creciente o decreciente.

Los árboles de búsqueda equilibrado son casos especiales de los árboles binarios de búsqueda – Tienen la estructura de un árbol binario y tienen la propiedad de un árbol binario de búsqueda - Es decir aplican directamente las estrategia de búsquedas previamente desarrolladas.

Árboles Binarios de Búsqueda Equilibrado

Entre los árboles de búsqueda equilibrados se pueden hacer referencia a: árboles AVL, los árboles rojo-negros, árboles biselados auto-ajustables.



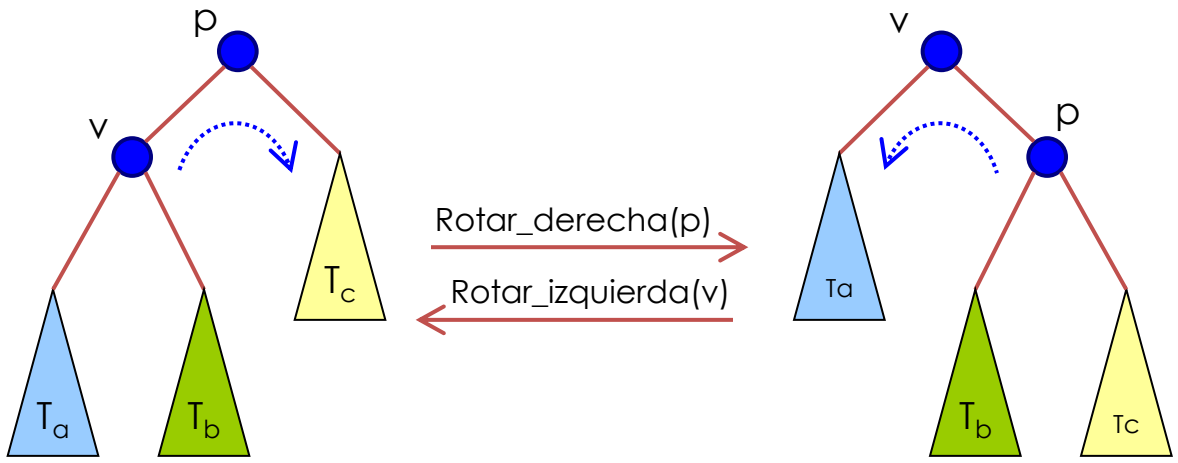
En los árboles de búsqueda equilibrados se aplican restricciones adicionales, que aunque pueden diferir de unos a otros, todas sirven al propósito general de restringir la altura de los subárboles.

Una característica que comparten los árboles de búsqueda equilibrado es que a fin de satisfacer sus restricciones reorganizan sus subárboles utilizando **rotaciones**.

Árboles Binarios de Búsqueda Equilibrado

Rotaciones

Cada uno de los patrones de rotación se construye utilizando una o más operaciones básicas de rotación mostradas a continuación.



Las dos operaciones básicas de rotación

Árboles Binarios de Búsqueda Equilibrado

Rotaciones

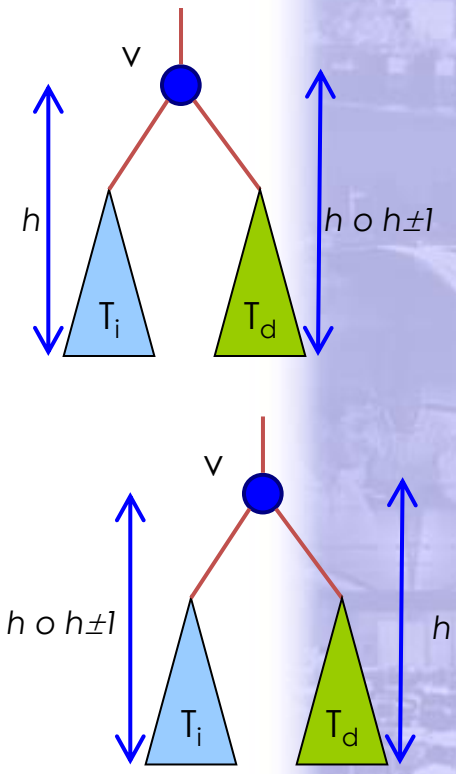
La característica más importante de estas operaciones de rotación, es que no destruyen la propiedad de los árboles binarios de búsqueda en los subárboles sobre los que se ejecutan. Se puede verificar que el orden de izquierda a derecha de los subárboles no cambian y que la mayoría de los subárboles a la izquierda y derecha de v y p no cambian después de la rotación, la excepción es Ta, en el caso de Rotar_derecha(p) y Tc, en el caso de Rotar_izquierda(v). Considérese el primer caso, antes de la rotación Ta esta a la izquierda del nodo p, pero luego no. de hecho Ta ya no es descendiente de p después de la rotación. Sin embargo antes de la rotación se verifica que clave(p)>clave(v)>clave(Ta) y es fácil comprobar que estas desigualdades no se violan después de la rotación. Un argumento similar se puede utilizar con tc y Rotar_izquierda(v).

Árboles AVL

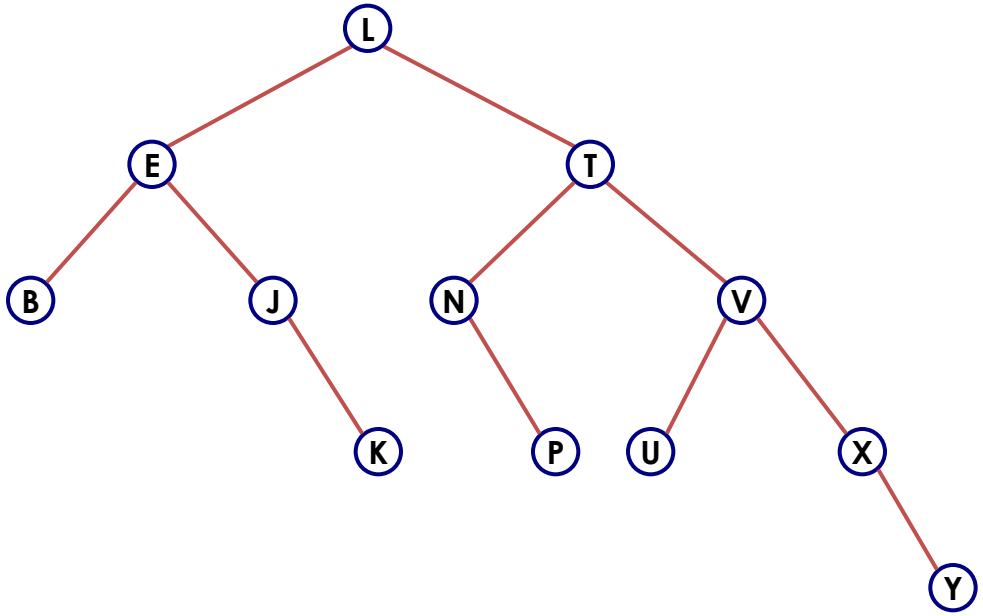
Un árbol AVL es un árbol de binario de búsqueda en el que las alturas de los subárboles izquierdo y derecho difieren a lo sumo en 1.

Dada la definición es posible probar que el tiempo de ejecución es de O(log n) para operación antes mencionadas.

El equilibrio de un nodo viene dado por la altura de su subárbol derecho menos la altura de su subárbol izquierdo. Así, el equilibrio de un nodo en un árbol AVL es -1, 0, ó +1.



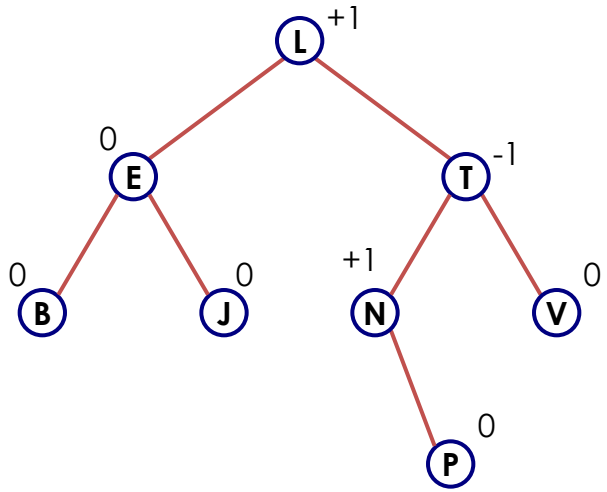
Árboles AVL



Un árbol AVL

Numero de Nodos = 12
Altura = 4

Árboles AVL



Un árbol AVL
Mostrando los equilibrios de cada uno de sus nodos

Operaciones sobre árboles AVL

Búsqueda o Consulta

Se pueden emplear los mismos algoritmos de búsqueda empleados para Árboles Binarios de Búsqueda para operaciones tales como:

- ✓ Buscar
- ✓ Máximo
- ✓ Mínimo
- ✓ Predecesor
- ✓ Sucesor