06 – Árboles Binarios

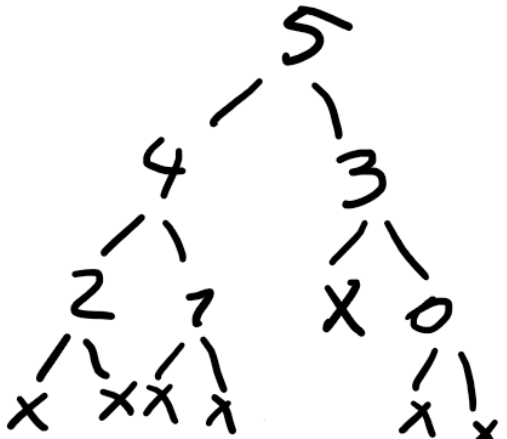
DOCENTE – FEDERICO VILENSKY

Qué son los árboles?

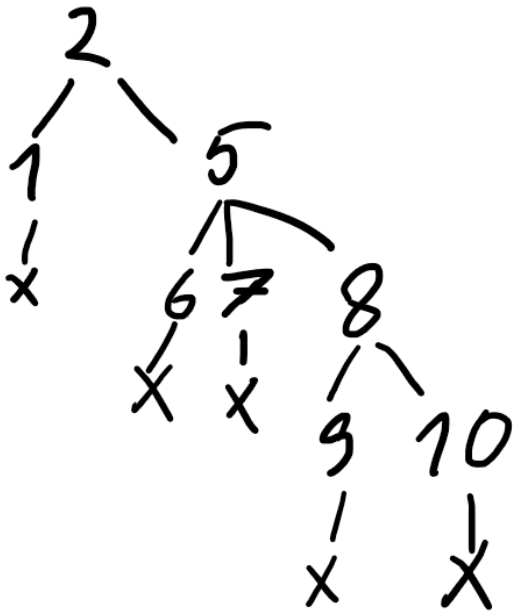
- Hasta ahora utilizamos la recursión para representar tipos relativamente simples, como naturales y listas
- Pero podemos utilizarla para generar estructuras mas complejas, las cuales vamos a llamar árboles
  - Un árbol (general o finitario/n-ario) con tipo base T es:
    - O bien la estructura vacía
    - O bien un elemento de tipo T, junto con un numero finito de estructuras de tipo árbol, con tipo base T, disjuntas, llamadas subárboles

Ejemplos de arboles de tipo base $\mathbb{N}$

árbol vacío

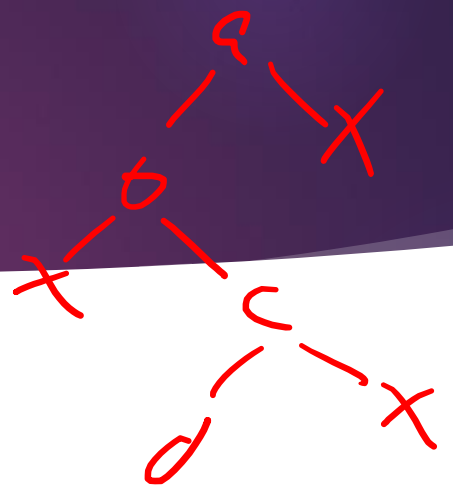


árbol binario



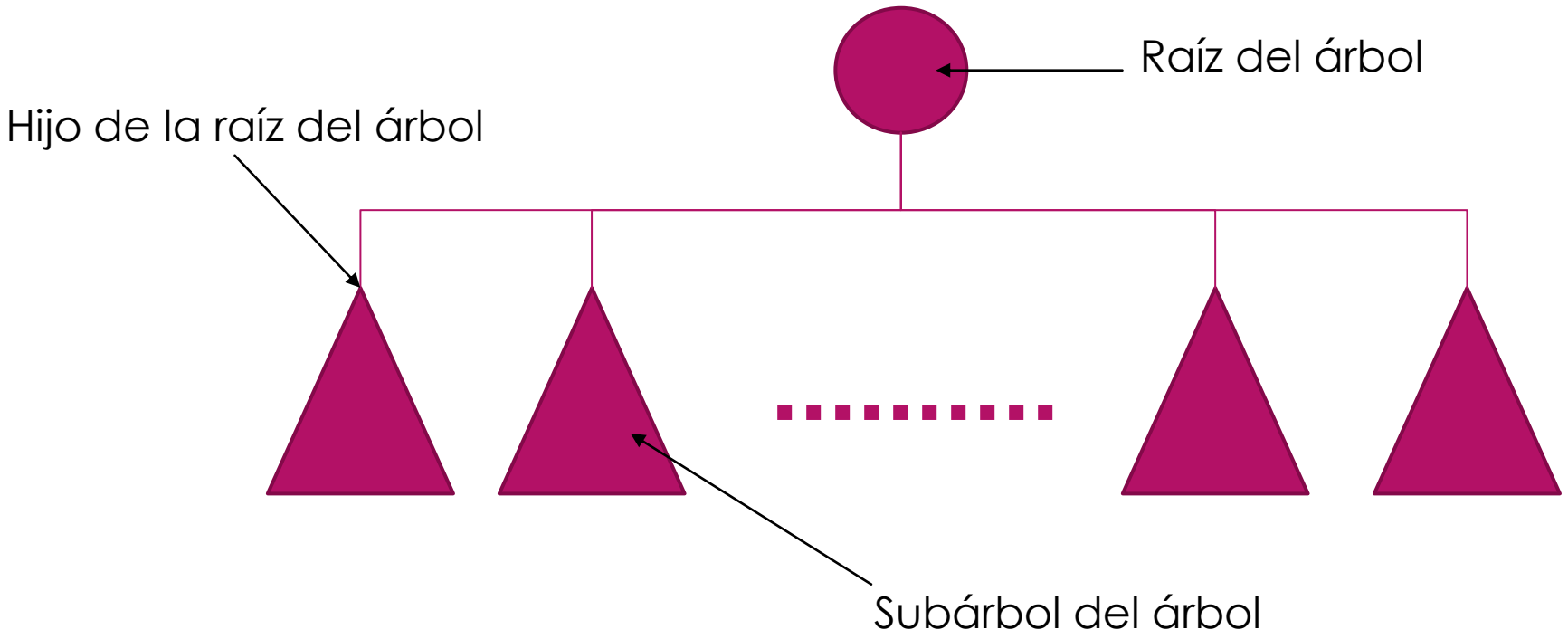
árbol general

La lista es un árbol?



- La lista de tipo base T, esta formada por
  - O bien la lista vacía
  - O bien un elemento de tipo T junto con una lista
- Efectivamente la lista es un árbol, las listas también son conocidas como árboles degenerados
  - Formalmente se le llama árbol degenerado a cualquier árbol donde cada subárbol tiene a lo sumo 1 subárbol

Nomenclatura



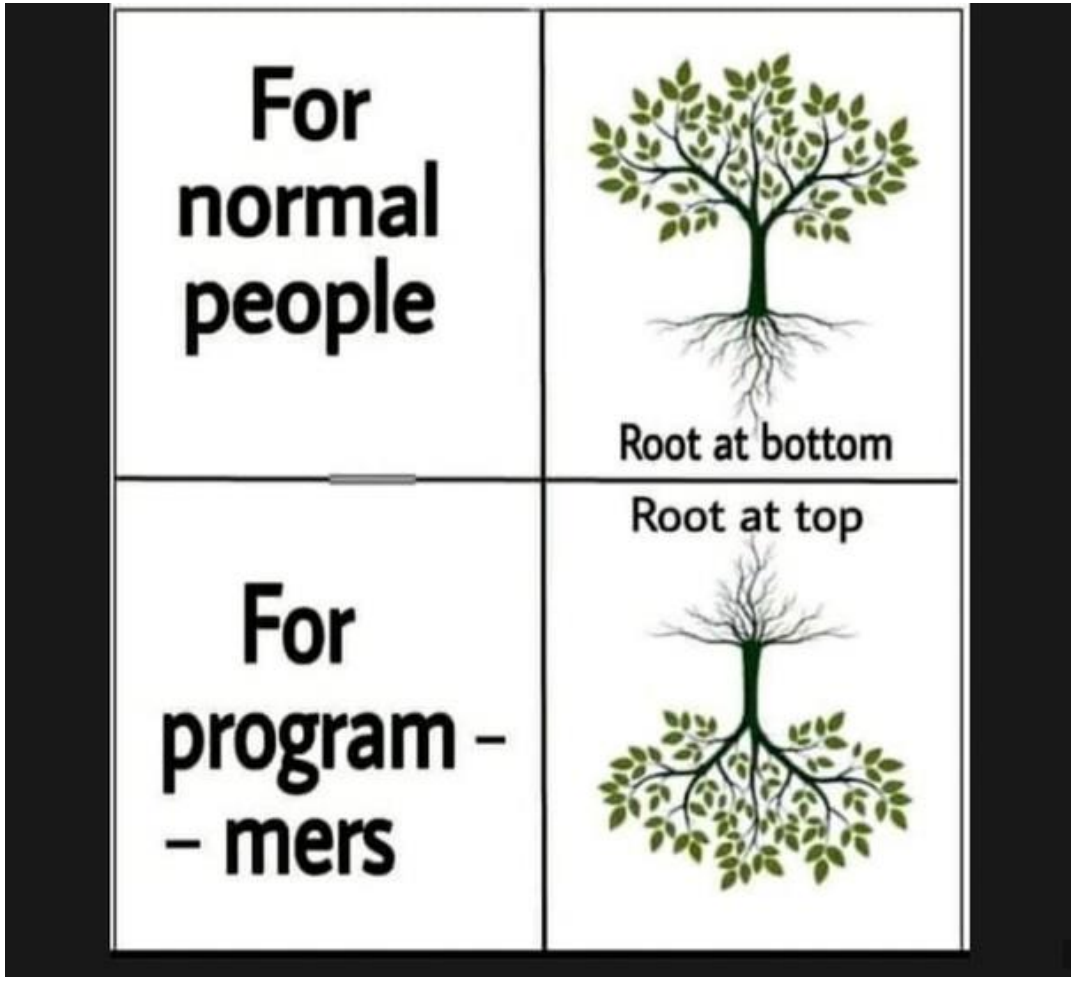
Los elementos se encuentran en nodos del árbol

Árboles n-arios

- La descripción que dimos de arboles, es casi una definición inductiva
- Nos falto mencionar nomas que queremos decir con un numero finito de subárboles
- Una posible manera es decir que tiene una cantidad fija, como por ejemplo
  - "un elemento de tipo T, junto con exactamente dos estructuras de tipo árbol, con tipo base T, disjuntas, llamadas subárboles"
  - Esto es un caso particular de un árbol n-ario, al que vamos a llamar árbol binario
- Estos árboles n-arios a su vez son casos particulares de árboles generales
- Un árbol general es cuando la cantidad de subárboles no es fija

Árboles binarios

- Vamos a estar trabajando principalmente con arboles binarios
- Esto es por la simpleza de trabajar con ellos
- Mas adelante vamos a ver que podemos representar árboles generales con árboles binarios
  - Utilizando cierta semantica


For normal people
Root at bottom
For program - - mers
Root at top

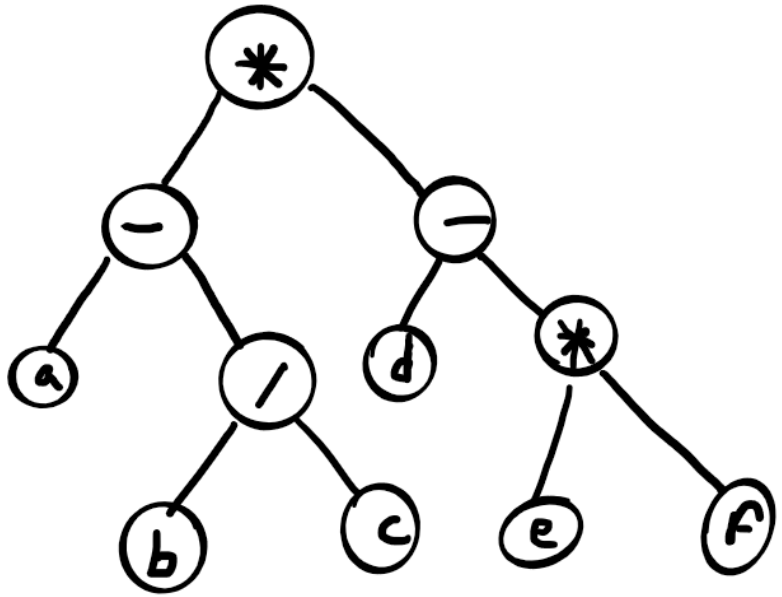
Dudas?

Ejemplo de uso
Sintaxis concreta vs. Sintaxis abstracta

- Si nosotros queremos representar $(a - b/c) * (d - e * f)$ necesitamos un montón de reglas
- Tenemos que saber que lo primero que hacemos es lo que esta dentro de los paréntesis
- Luego las divisiones y multiplicaciones
- Por ultimo las sumas y restas
- Necesitamos saber muchas cosas para que la representación no sea ambigua

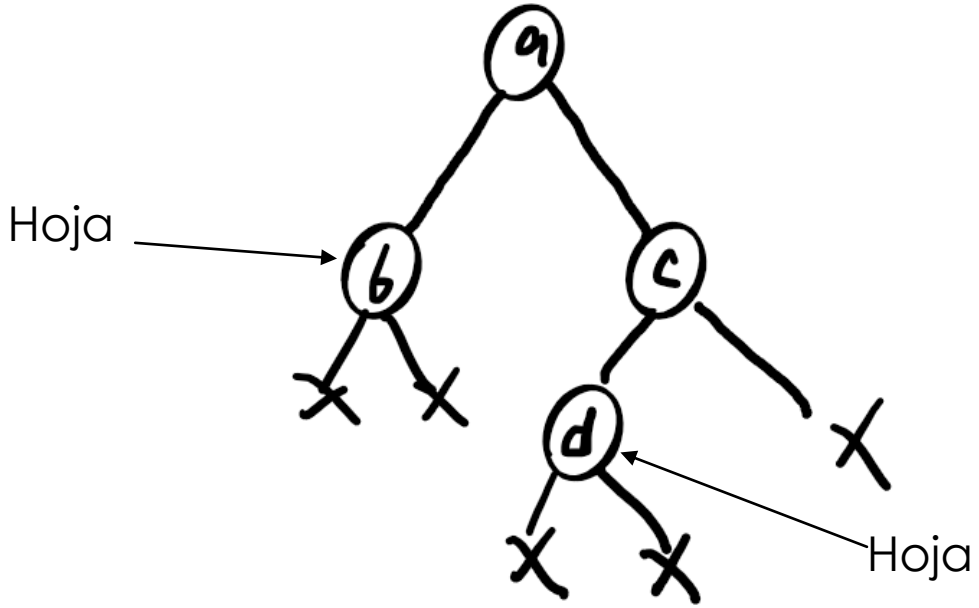
Ejemplo de uso
Sintaxis concreta vs. Sintaxis abstracta

- En cambio con arboles podemos representar lo que se llama la sintaxis abstracta, la cual no tiene ninguna ambigüedad



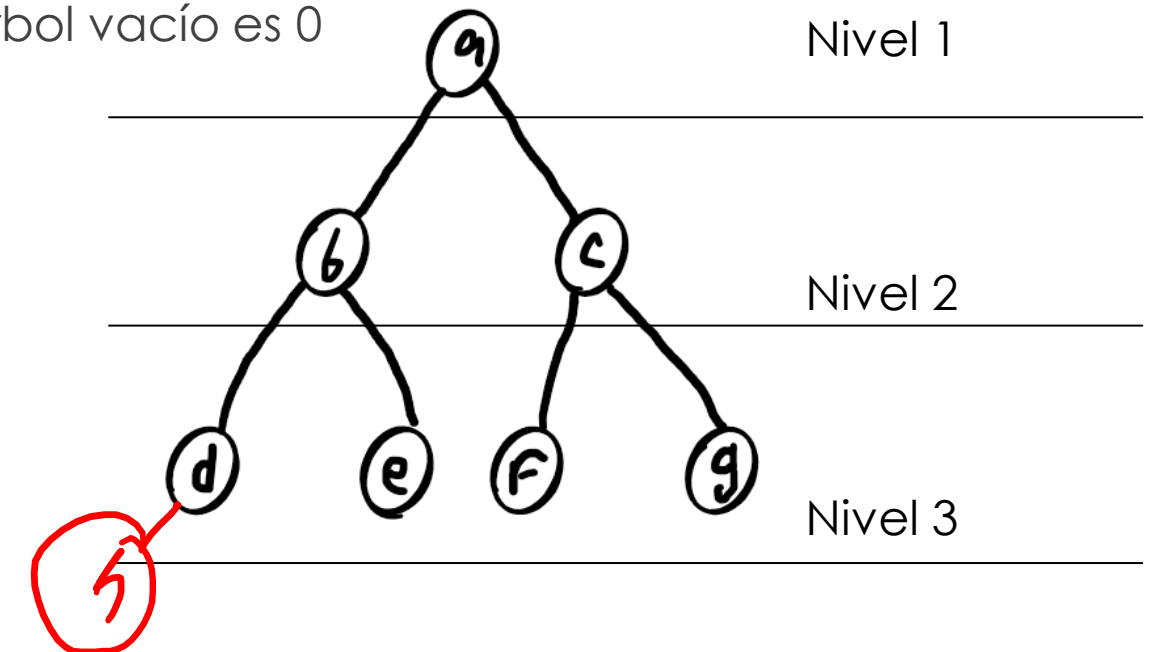
Hojas

- Hojas son los nodos en que TODOS sus subárboles son vacíos



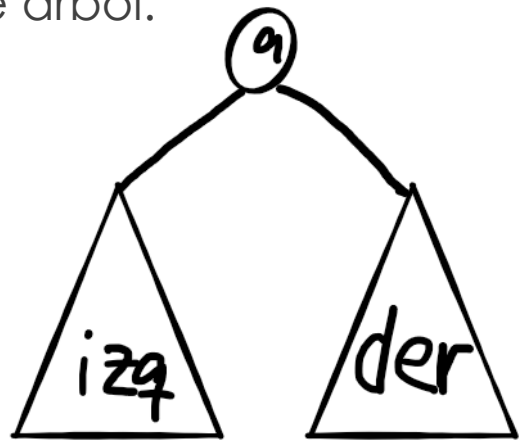
Niveles y altura

- La altura de un árbol es la cantidad de niveles que tiene
- Otra manera de pensarlo es que la altura es la cantidad de nodos que tiene en el camino más largo de la raíz a una hoja
- La altura de un árbol vacío es 0



Altura

- La altura del siguiente árbol:



- Es $1 + \max(h_{izq}, h_{der})$ donde h_{izq} es la altura del subárbol izquierdo y h_{der} es la altura del subárbol derecho

Mas formalmente

- Definición inductiva de árboles binarios
- $$\frac{}{(\,):ArbBin} \qquad \frac{izq{:}ArbBin \quad t{:}T \quad der{:}ArbBin}{(izq, t, der){:}ArbBin}$$

- $altura\big((\,)\big) = 0$
  $altura\big((izq, a, der)\big) = 1 + \max\big(altura(izq), altura(der)\big)$

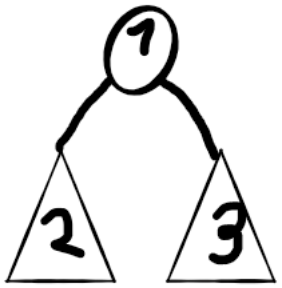
Recorridas de árboles binarios

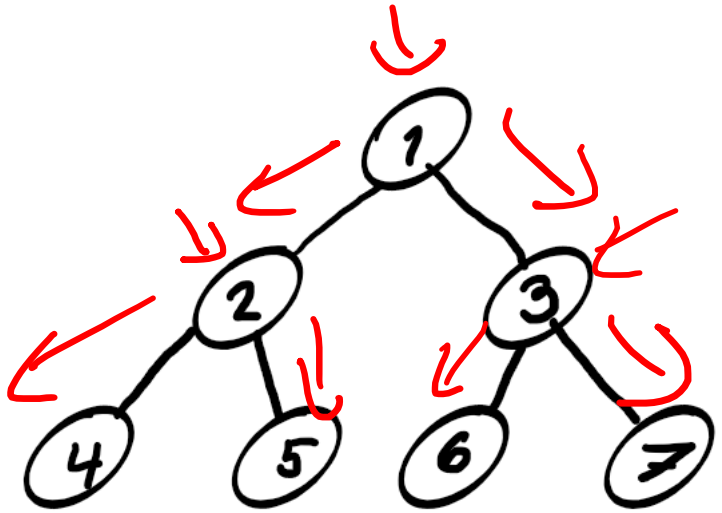
- Una recorrida es un procedimiento que visita todos los nodos de un árbol binario
  - Va efectuando cierta acción sobre cada uno de ellos
- La forma en que recorremos el árbol puede afectar al procedimiento
- Vamos a ver entonces como podemos recorrer los arboles binarios
- Si el árbol es vacío, es trivial, veamos que pasa cuando el árbol NO es vacío

Recorridas de árboles binarios

- Vamos a restringirnos a recorrer los arboles de izquierda a derecha, es decir solo podemos recorrer el subárbol derecho una vez que hayamos recorrido al izquierdo
- Vamos a tener entonces 3 formas de recorrer un árbol, y las vamos a llamar dependiendo cuándo vemos a la raíz
- PreOrder: Antes que los subárboles
  - Raiz -> Subárbol izquierdo -> Subárbol derecho
- InOrder: Entre medio de los subárboles. Muchas veces se lo llama **en orden**
  - Subárbol izquierdo -> Raíz -> Subárbol derecho
- PostOrder: Después de los subárboles
  - Subárbol izquierdo -> Subárbol derecho -> Raíz

Recorrida: PreOrder

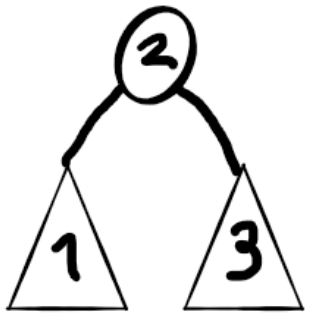


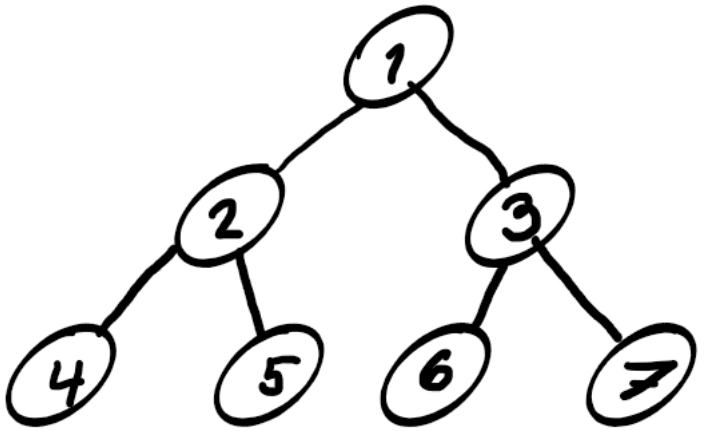


1, 2, 4, 5, 3, 6, 7

PreOrder: 1,2,4,5,3,6,7

Recorrida: InOrder

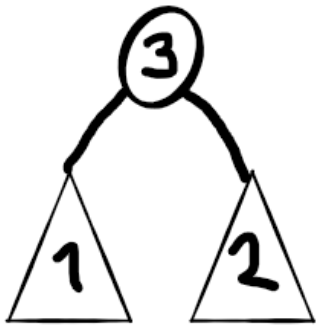


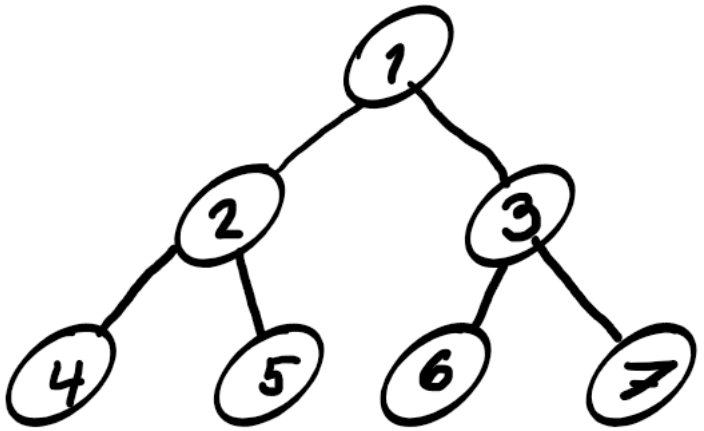


4, 2, 5, 1, 6, 3, 7

InOrder: 4,2,5,1,6,3,7

Recorrida: PostOrder





4, 5, 2, 6, 7, 3, 1

PostOrder:4,5,2,6,7,3,1

Ejemplo

- Generar una lista de elementos de un árbol binario que se obtiene al recorrerlo en orden

- $enOrden(()) = [\,]$
  $enOrden((izq, a, der)) = enOrden(izq) \; ++ \; [a] \; ++ \; enOrdern(der)$

pre((izq, a, der)) = [a] ++ pre(izq) ++ pre(der)

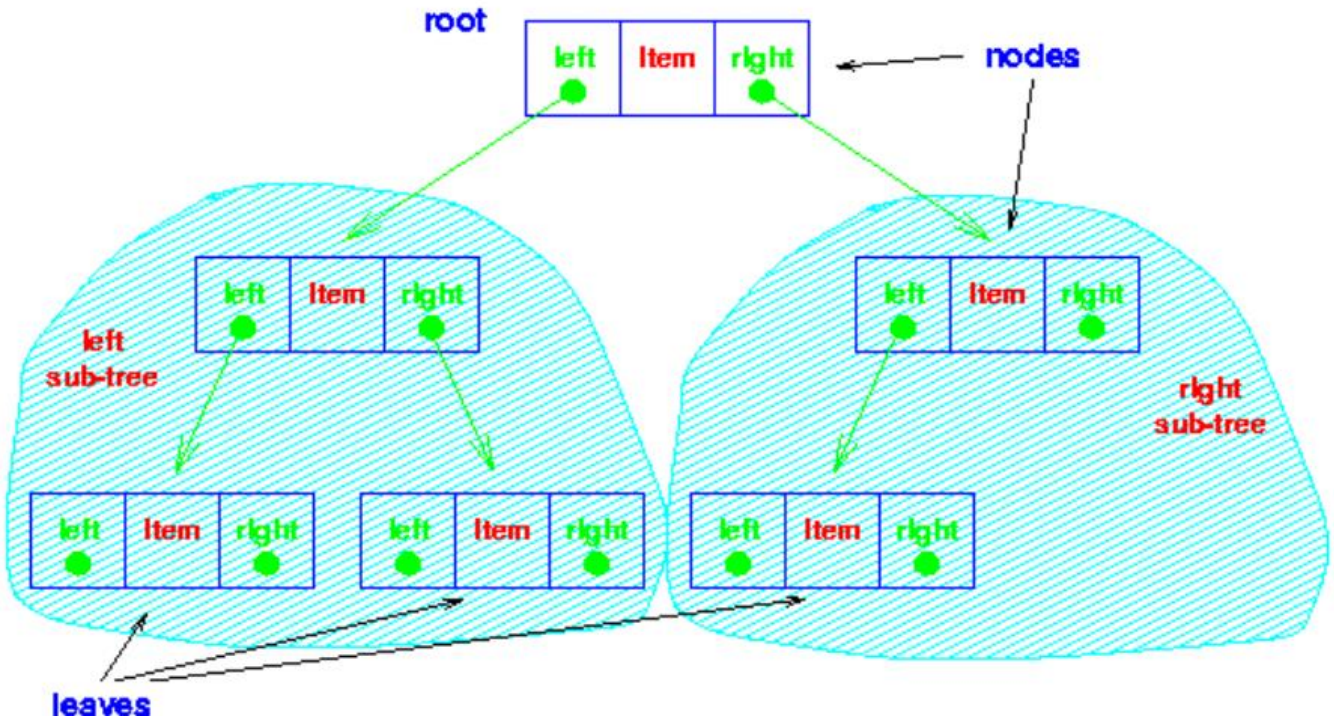
post((izq, a, der)) = post(izq) ++ post(der) ++ [a]

Implementación en C++

- Vamos a ver una posible forma de implementar el tipo de dato árbol binario en C++
- Como ya venimos haciendo, los elementos del árbol son de tipo T
- `typedef NodoAB* AB;`

```
struct NodoAB
{
    T item;
    AB left, right;
};
```

Visualización



Código de las recorridas

- Vamos a formular el código de los 3 métodos de recorrida de árbol que vimos, ahora en C++
- Vamos a usar un parámetro `t` que va a ser el árbol que estamos recorriendo
- Además vamos a asumir que conocemos la función `p` que representa la operación que queremos hacer sobre los elementos del árbol

PreOrder C++

```
void preOrder(AB t)
{
    if(t!=NULL)
    {
        p(t->item);
        preOrder(t->left);
        preOrder(t->right);
    }
}
```

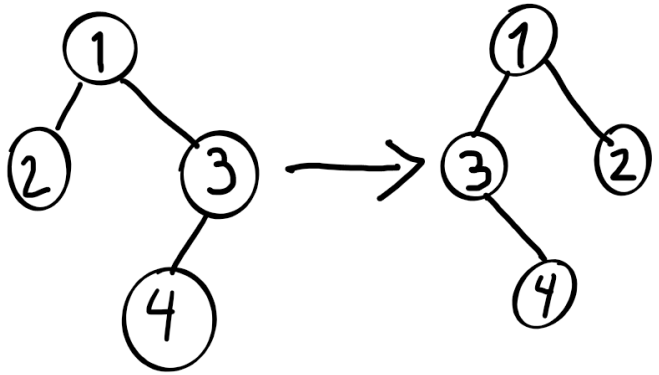
InOrder C++

```
void inOrder(AB t)
{
    if(t!=NULL)
    {
        inOrder(t->left);
        p(t->item);
        inOrder(t->right);
    }
}
```

PostOrder C++

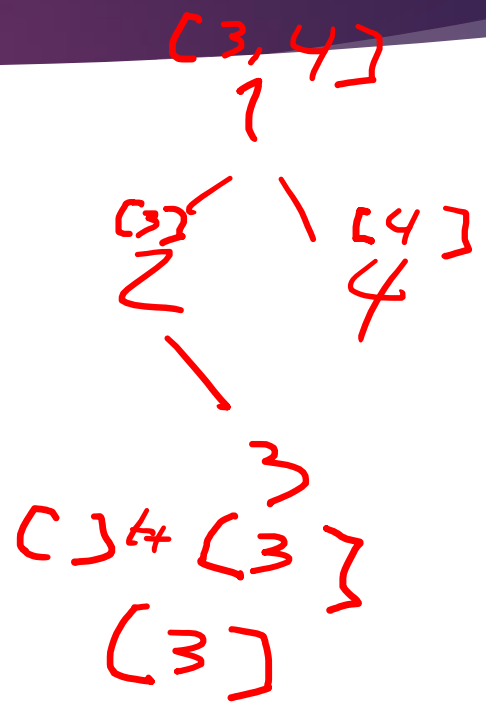
```
void postOrder(AB t)
{
    if(t!=NULL)
    {
        postOrder(t->left);
        postOrder(t->right);
        p(t->item);
    }
}
```

Cantidad de nodos de un árbol

- $cantNodos(( )) = 0$
  $cantNodos((izq, a, der)) = 1 + cantNodos(izq) + cantNodos(der)$
- ```
  int cantNodos(AB t)
  {
      if(t==NULL) return 0;
      else
          return 1 + cantNodos(t->left) + cantNodos(t->right)
  }
  ```
- Qué diferencia tiene esto con la altura?
  ```

Espejo

- $espejo(()) = ()$
  $espejo((izq, a, der)) = (espejo(der), a, espejo(izq))$
- ```
  AB espejo(AB t)
  {
      if(t == NULL) return NULL;
      else
      {
          AB espejado = new NodoAB;
          espejado->item = t->item;
          espejado->left = espejo(t->right);
          espejado->right = espejo(t->left);
          return espejado;
      }
  }
  ```
- Si quisiéramos implementar una función que retorna una copia idéntica del árbol que recibe, sin compartir memoria, cómo podríamos hacer?


  ```

Hojas de un árbol

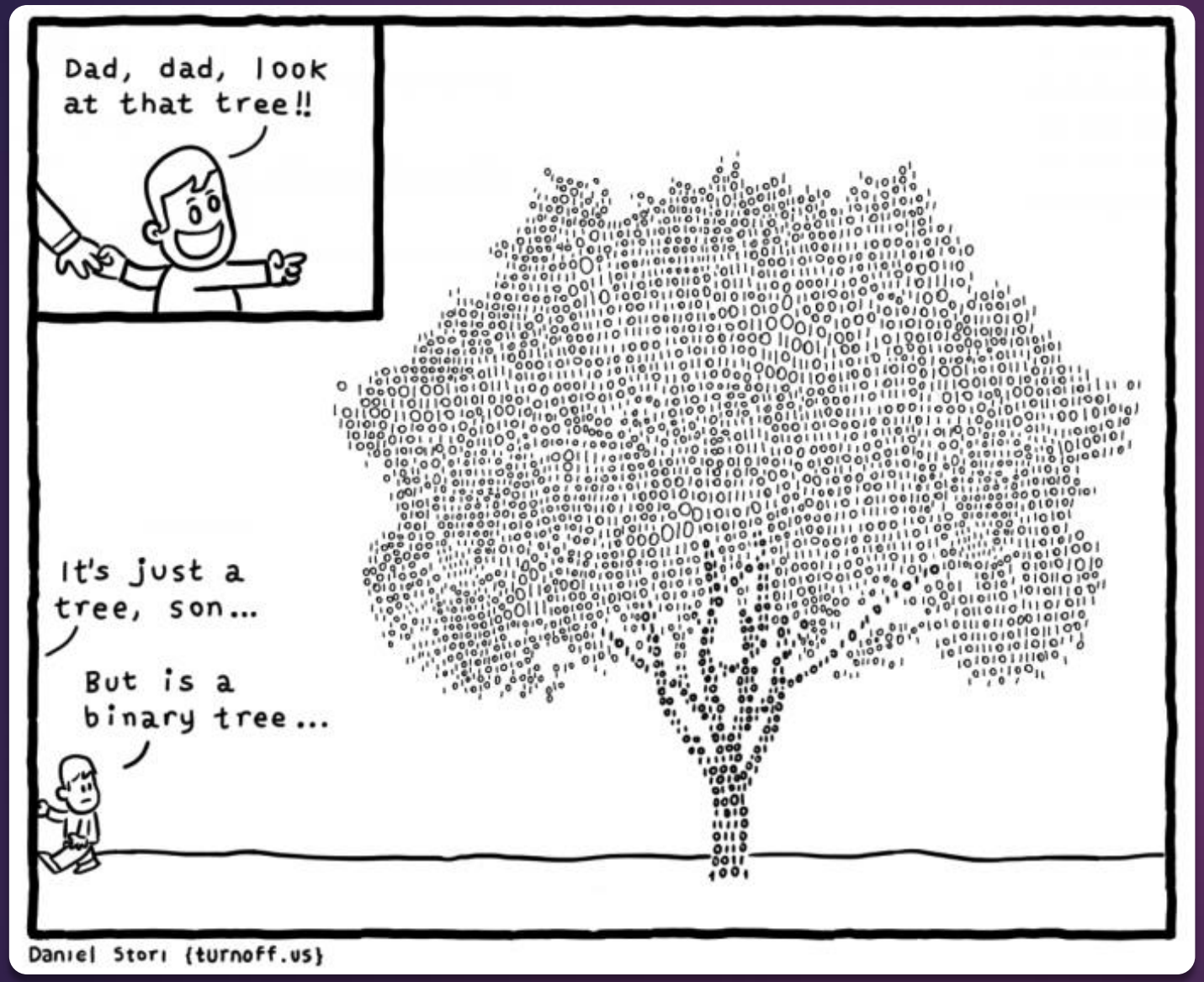
- $hojas(( )) = [\,]$
  $hojas((( ), a, ( )) = [a]$
  $hojas((izq, a, der)) = hojas(izq) + +hojas(der)$



Hojas de un árbol

```
Lista hojas (AB t)
{
  Lista p;
  if(t == NULL) return NULL;
  else if(t->left == NULL && t->right == NULL)
  {
    p = new nodoLista;
    p->info = t->item;
    p->sig = NULL;
    return p;
  }
  else return concat(hojas(t->left), hojas(t->right));
}
```

- Habría que implementar `Lista concat(Lista l1, Lista l2)`


Dad, dad, look at that tree!!
It's just a tree, son...
But is a binary tree...
Daniel Stori {turnoff.us}

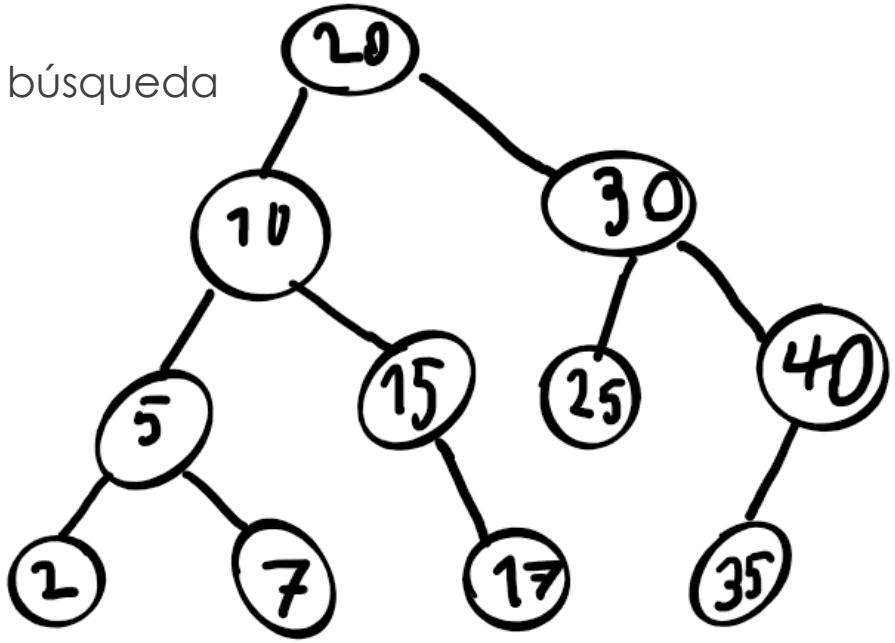
Dudas?

Árboles binarios de búsqueda (ABB/BST)

- Los arboles binarios se suelen utilizar para representar conjuntos de datos en los que los elementos pueden ser recuperables por medio de una clave única
  - O sea que podemos almacenar elementos sin repeticiones y recuperarlos
- Si un árbol está organizado de tal forma que para cada nodo n_i todas las claves en el subárbol izquierdo son menores que la clave de n_i, y todas las claves en el subárbol derecho son mayores que la clave de n_i
  - Entonces es un ABB

Ejemplo de ABB

- Como podemos (arrancando de la raíz, buscar el elemento identificado con el 7?
  - Es muy parecido a la búsqueda por biparticion

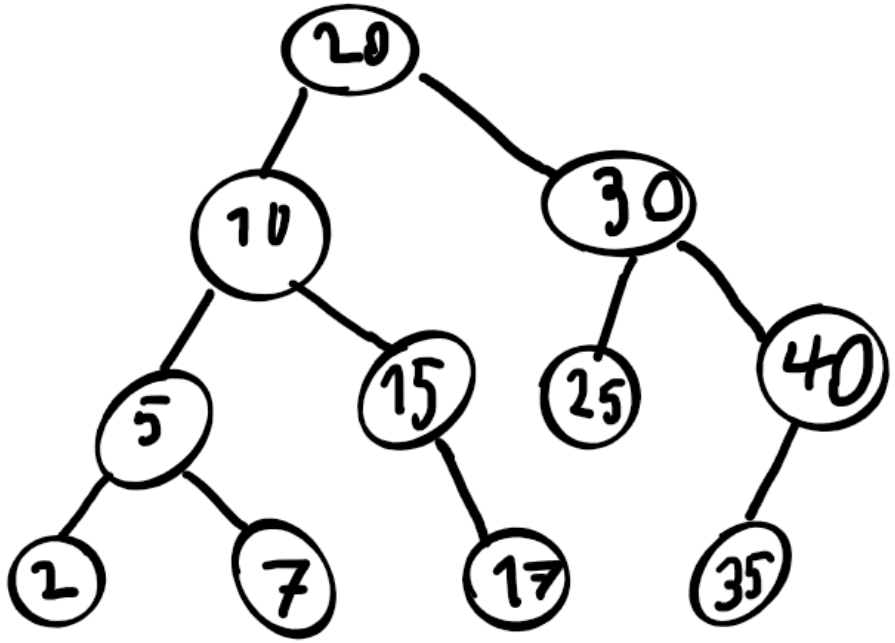


Ejemplo de ABB

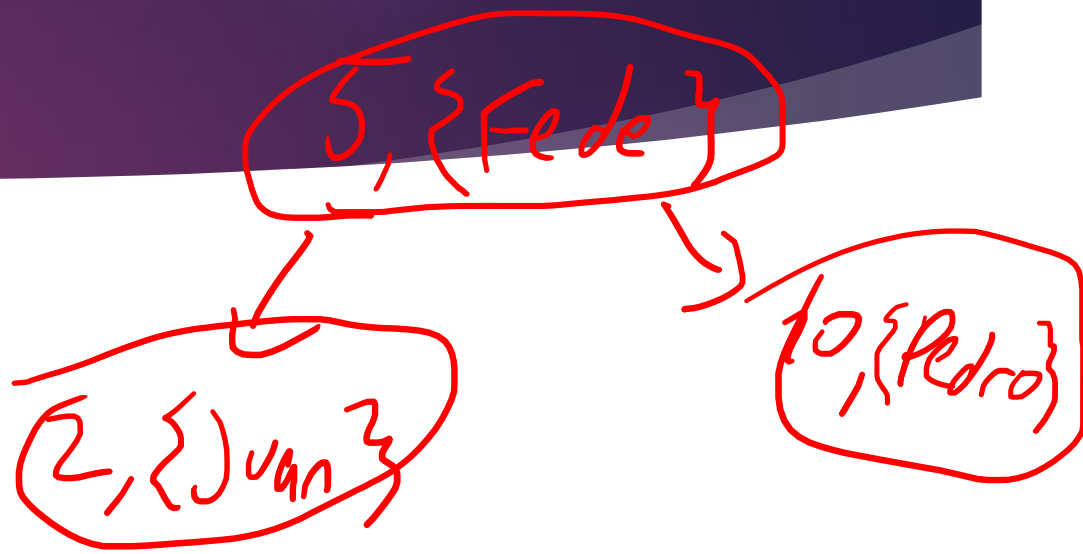
40, 35, 30, 25...

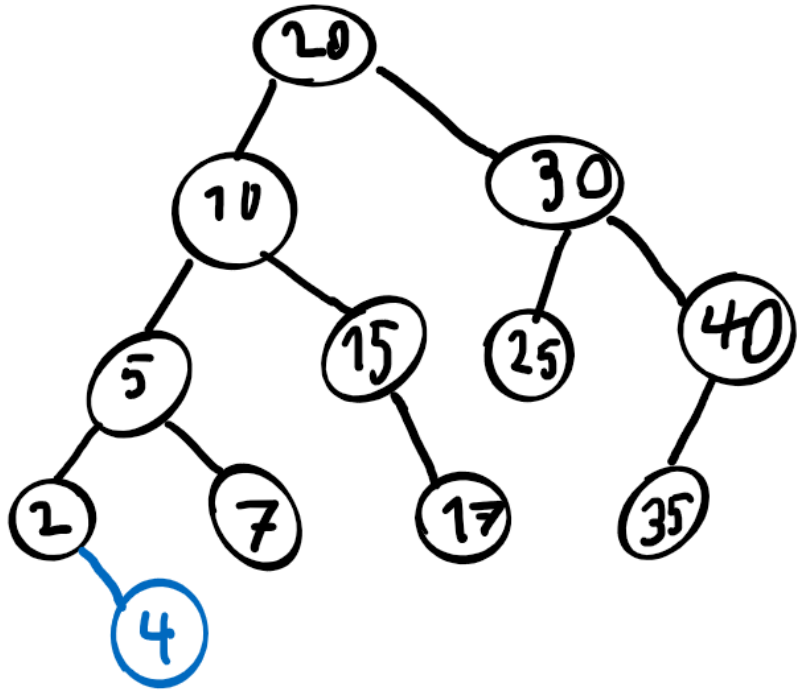
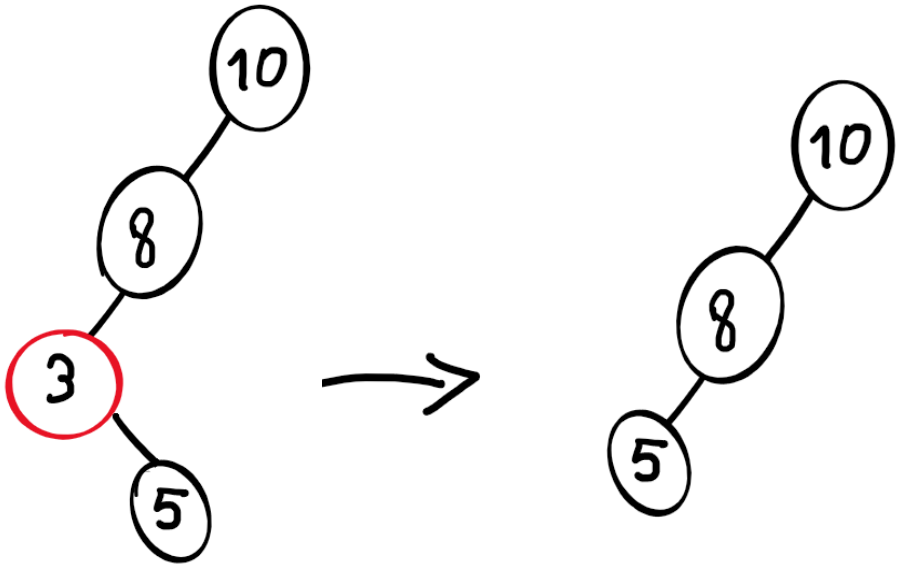
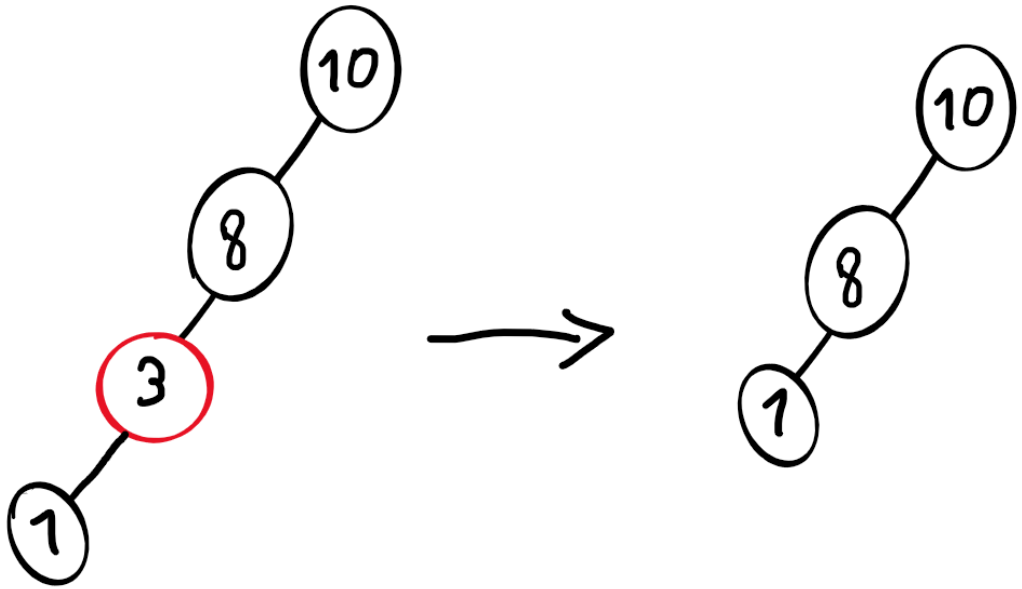
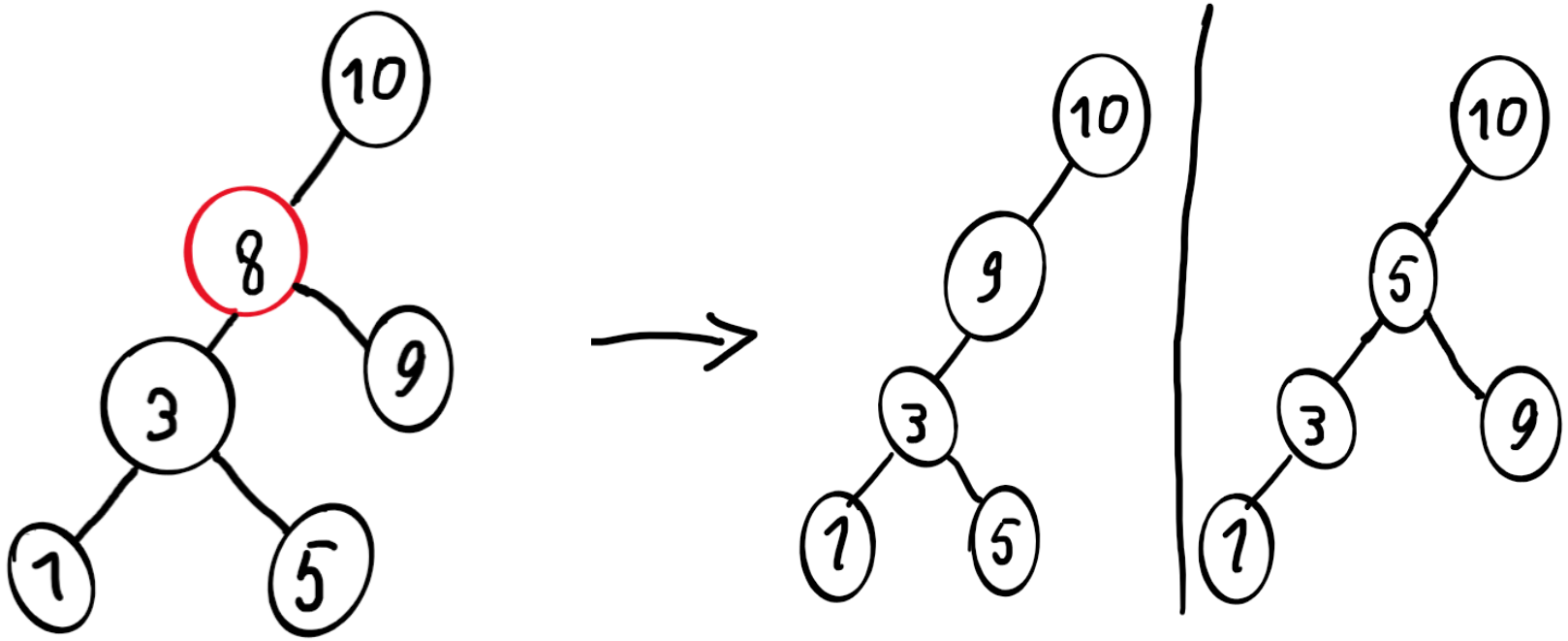
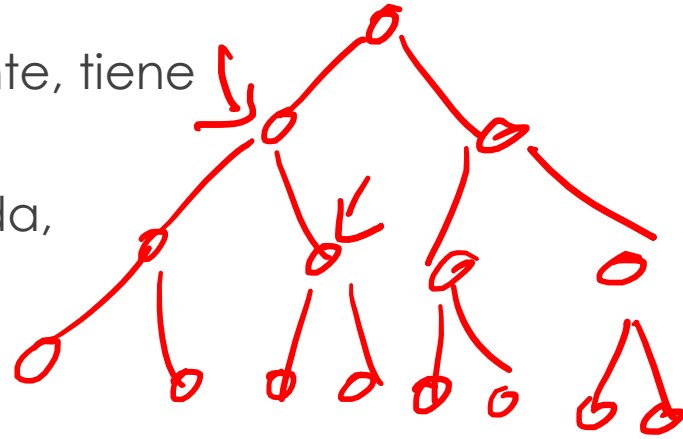
- Dónde está el mínimo y el máximo en un ABB?
- Qué pasa con el recorrido inOrder?

2, 5, 7, 10, 15, 17, 20, 25, 30, 35, 40



Implementación de ABB




- ```
  typedef NodoABB* ABB;

  struct NodoABB
  {
    Ord key;
    T info;
    ABB left, right;
  }
  ```
- La implementación del tipo ABB es muy parecida a la de AB
- La diferencia es que ahora diferenciamos un campo, `key`, cuyo tipo es `Ord` un ordinal
  - Lo vamos a utilizar para recuperar al dato
  - Y además lo vamos a utilizar para comparar a ver si es menor o mayor con los otros nodos
  ```

Búsqueda binaria

- En los ABB podemos encontrar un nodo con una clave arbitraria, arrancando desde la raíz del árbol
  - Si la clave que estamos buscando es igual que la clave de la raíz, encontramos al dato
  - Si la clave que estamos buscando es menor que la clave de la raíz, buscamos en el subárbol izquierdo
  - Si la clave que estamos buscando es mayor que la clave de la raíz, buscamos en el subárbol derecho
  - Si llegamos a NULL, es que la clave no se encuentra en el ABB
- Como recorremos un único camino, podemos hacer esto de forma iterativa de una forma bastante simple

Búsqueda binaria iterativa

```
ABB buscarIterativo(ABB t, Ord x)
{
  while(t != NULL && t->key != x)
  {
    if(x > t->key) t = t->right;
    else t = t->left;
  }
  if(t!= NULL) return t;
  else return NULL;
}
```

Búsqueda binaria recursiva

```
ABB busquedaRecursiva(ABB t, Ord x)
{
  if(t == NULL) return NULL;
  else if(x == t->key) return t;
  else if(x > t->key) return busquedaRecursiva(t->right, x);
  else return busquedaRecursiva(t->left, x);
}
```

Pertenencia

- Otra de las funciones claves de los arboles binarios es la pertenencia al mismo
- Es muy parecida a la búsqueda, pero en vez de retornarnos el nodo, simplemente nos retorna un booleano
- Nos retorna true si el elemento pertenece, false en caso contrario
- Este va para que lo hagan ustedes

Inserción

- Vamos a recorrer el árbol, hasta llegar al final, y agregar una nueva hoja
- En la raíz
  - Si la raíz es NULL, creamos nuestra nueva hoja
  - Si la clave del dato que queremos agregar es mayor a la clave de la raíz, lo insertamos en el subárbol derecho
  - Si la clave del dato que queremos agregar es menor a la clave de la raíz, lo insertamos en el subárbol izquierdo



Inserción

```
void insercionABB(ABB &t, Ord clave, T dato)
{
  if(T == NULL)
  {
    t = new NodoABB;
    t->key = clave;
    t->info = dato;
    t->left = NULL;
    t->right = NULL;
  }
  else if(clave > t->key)
  {
    insercionABB(t->right, clave, dato);
  }
  else if (clave < t->key)
  {
    insercionABB(t->left, clave, dato);
  }
}
```

Eliminar en un ABB

- Vamos a tener que fijarnos que pasa cuando
  - No tiene hijo a la izquierda
  - Tiene hijo a la izquierda pero no a la derecha
  - Tiene hijo a la izquierda Y a la derecha

Eliminar en un ABB

- Sin hijo a la izquierda



Eliminar en un ABB

- Sin hijo a la derecha



Eliminar en un ABB

- Con hijo a la izquierda Y a la derecha



Eliminar en un ABB

```
void eliminarABB(ABB &t, Ord x)
{
  if (t != NULL)
  {
    if(x > t->key) eliminarABB(t->right, x);
    else if (x < t->key) eliminarABB(t->left, x);
    else
    {
      if(t->left == NULL)
      {
       ABB aux = t;
       t = t->right;
       delete aux;
      }
      if(t->right== NULL)
      {
       ...
      }
      else
      {
         ABB max_izq = maximo(t->izq); // o ABB min_der = minimo(t->right)
         t->key = max_izq->key;
         t->info = max_izq->info;
         eliminarABB(t->izq, x);
      }
    }
  }
}
```

Ordenes de ejecución

- Si un algoritmo recorre un árbol haciendo cosas de orden constante, tiene $O(n)$ en peor caso y caso promedio (pasamos por los n nodos)
- Si queremos recorrer un único camino del árbol (como la búsqueda, inserción, etc. en un ABB)
  - Tiene $O(n)$ en peor caso
    - Cuando es un árbol degenerado (una lista)
  - Tiene $O(\log n)$ en caso promedio
    - En cada nivel vamos dividiendo en dos la cantidad de nodos del subárbol



Dudas?

nobody:
binary search trees:

Ejercicio: Aplanar un ABB

- Completar el siguiente código para tener una lista ordenada con los elementos de un ABB, implementando **`aplanarEnLista`**, de tal manera que aplanar tenga $O(n)$ peor caso
  - n es la cantidad de elementos en el ABB
- ```
  Lista aplanar(ABB t)
  {
    Lista l = NULL;
    aplanarEnLista(l, t);
    return l;
  }
  void aplanarEnLista(Lista &l, ABB t){...}
  ```
  ```