DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Análisis de Algoritmos y Estructura de Datos

Implementación: TDA ABB

Prof. Violeta Chang C.

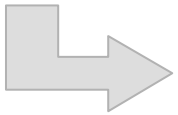
Semestre 2 – 2023

TDA ABB

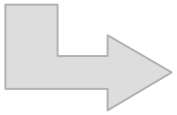
- **Contenidos**:
  - Estructura de datos TDA ABB
  - Operaciones de TDA ABB
  - Recorrido de árboles
- **Objetivos**:
  - Implementar estructura de datos y operaciones básicas de TDA ABB

Ruta de trabajo

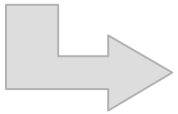
Especificación de TDA árbol binario de búsqueda (ABB)



Implementación de TDA ABB en C



Actividades de implementación

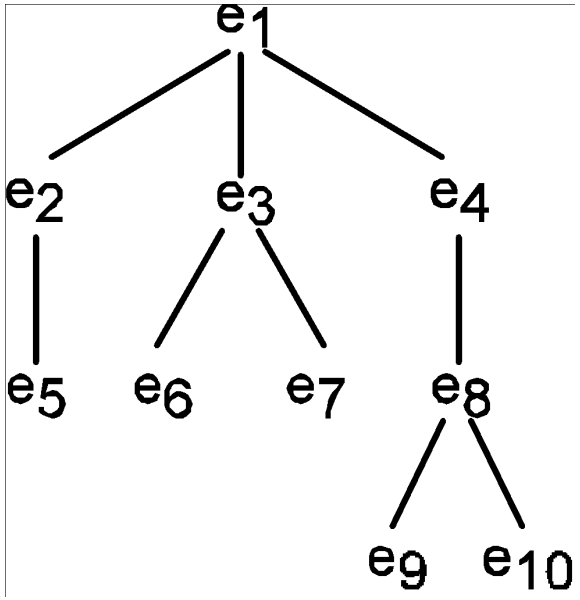


Fechas próximas

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Especificación de
TDA árbol binario de búsqueda

Especificación de TDA árbol



raíz: e_1
nodos internos: e_2, e_3, e_4, e_8
hojas: e_5, e_6, e_7, e_9, e_{10}

e_1 es el **padre** de e_2, e_3, e_4
e_2, e_3, e_4 son los **hijos** de e_1

El **grado** de e_1 es 3. El grado de e_3 es 2 y el grado de e_9 es 0.

La **altura del árbol** es 4 → el camino más largo desde la raíz a una hoja tiene 4 nodos

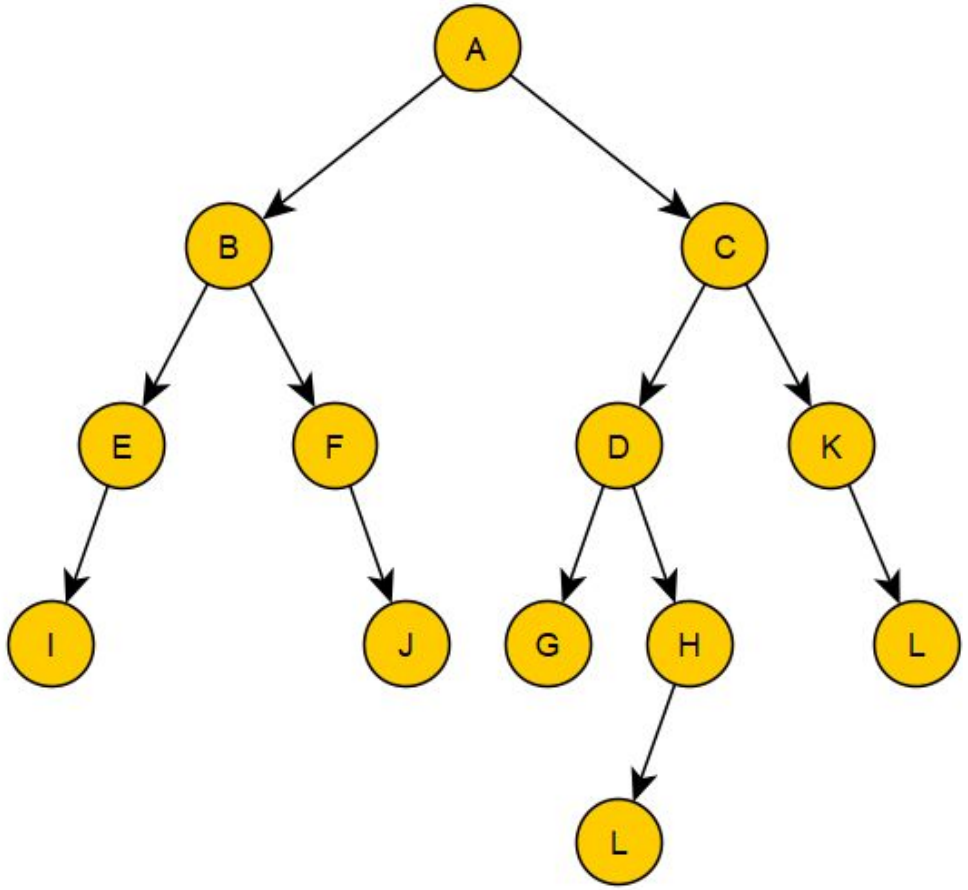
La **altura del nodo** e_4 es 3 → el camino más largo desde una hoja hasta e_4 tiene 3 nodos
La **profundidad (o nivel) del nodo** e_4 es 2 → el camino más largo desde la raíz hasta e_4 tiene 2 nodos. Lo mismo sucede con e_2, e_3

Especificación de TDA árbol

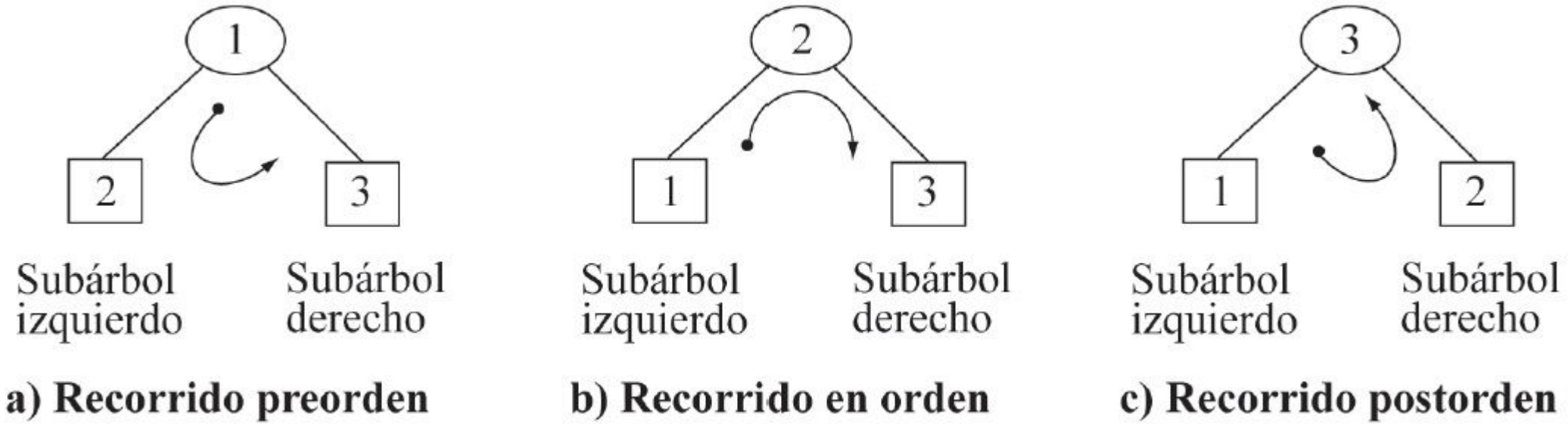
- **Operaciones:**
  - **esArbolVacío(T)**: determina si árbol T está vacío o no
  - **raíz(T)**: retorna la raíz del árbol T
  - **padre(T,nodo)**: retorna el padre de nodo. Si T es la raíz retorna nulo
  - **esHoja(T,nodo)**: indica si nodo es o no una hoja
  - **insertarNodo(T,nodo)**: inserta nodo en árbol T
  - **eliminarNodo(T,nodo)**: elimina nodo de árbol T
  - **buscarDato(T,dato)**: busca nodo con dato en árbol T
  - **recorrerArbol(T)**: muestra contenido de cada nodo de árbol T

TAD Árbol Binario

- **Estructura de datos**
  - Similar a TAD árbol, con la restricción que cada nodo puede tener a lo más 2 hijos
- **Operaciones**
  - TAD árbol
  - **insertar(T,x)**: O(n)
  - **eliminar(T,x)**: O(n)
  - **buscar(T,d)**: O(n)



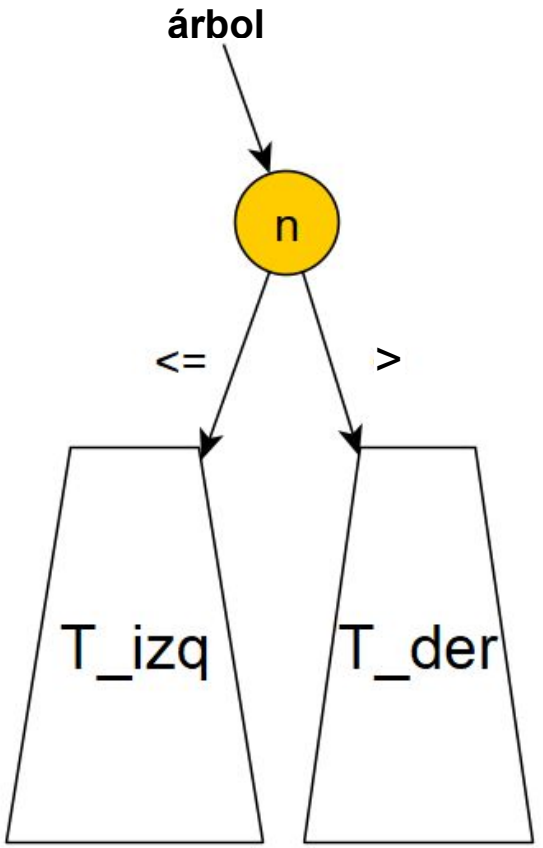
Recorrido de árbol binario



¿Cuándo se visita la raíz?

TDA árbol binario de búsqueda

- **Estructura de datos**
  - **Puntero externo**: apunta a la raíz del árbol
  - Cada nodo contiene:
    - dato
    - puntero a hijo izquierdo
    - puntero a hijo derecho
  - Un árbol es ABB ssi:
    - $n_i \leq n, \forall n_i \in T_{izq}\ y\ T_{izq}\ es\ ABB$
    - $n < n_i, \forall n_i \in T_{der}\ y\ T_{der}\ es\ ABB$



- **Operaciones**
  - TAD árbol
  - **insertar(T,x)**: O(*log* n) mejor caso
  - **eliminar(T,x)**: O(*log* n) mejor caso
  - **buscar(T,d)**: O(*log* n) mejor caso

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

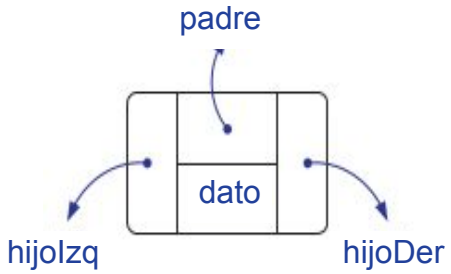
Implementación de TDA árbol binario de búsqueda

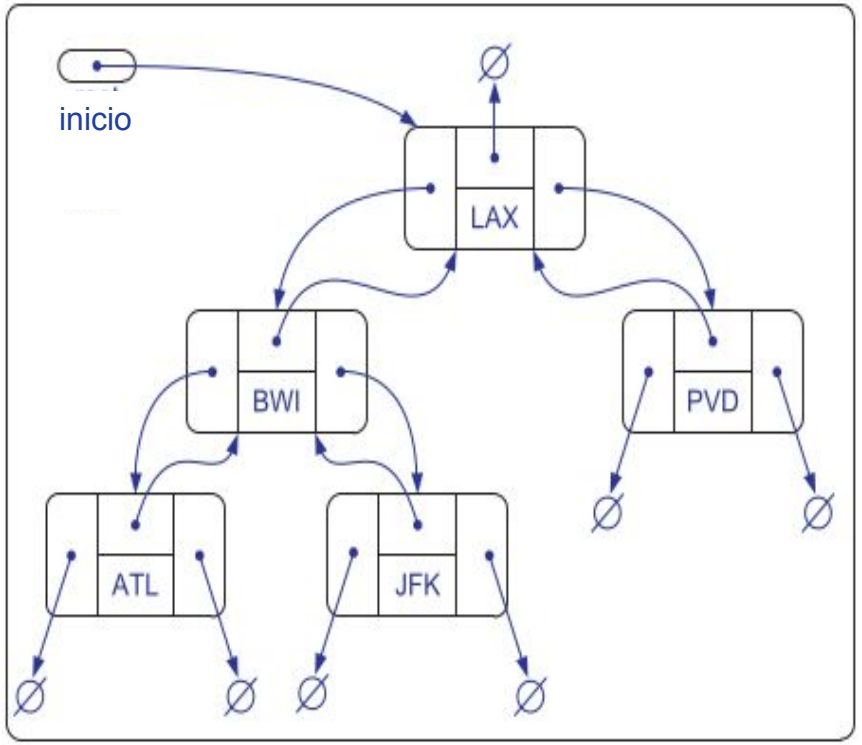
Implementación de un ABB usando punteros

- La estructura de datos que representa un árbol binario de búsqueda (ABB) usando punteros es la siguiente:

```
typedef struct nodoABB
{
   int dato;
   struct nodoABB* padre;
   struct nodoABB* hijoIzquierdo;
   struct nodoABB* hijoDerecho;
}nodoABB;

typedef nodoABB* TDAabb;
```





Implementación de un ABB usando punteros

```c
/*--------------- operaciones de creación -----------------*/

TDAabb crearABBVacio()
{
  TDAabb abb=(TDAabb)malloc(sizeof(TDAabb));
  abb=NULL;
  return abb;
}

int esABBvacio(TDAabb abb)
{
  if (abb == NULL)
    return 1;
  else
    return 0;
}
```

```c
/*--------------- operaciones de posicion -----------------*/

//devuelve NULO si el árbol está vacío
nodoABB* raizABB(TDAabb arbol)
{
  if (arbol!=NULL)
    return arbol;
  return NULL;
}

nodoABB* padreNodoABB(nodoABB* nodo)
{
  if (nodo!=NULL)
    return nodo->padre;
}

int esHojaABB(nodoABB* nodo)
{
  if ((nodo->hijoIzquierdo==NULL)&&(nodo->hijoDerecho==NULL))
    return 1;
  return 0;
}
```

Implementación de un ABB usando punteros

```
/*--------------- operaciones auxiliares -----------------*/

nodoABB* buscarMenorABB(TDAabb arbol, nodoABB* nodo)
{
  nodoABB* aux;
  if (!esABBvacio(arbol))
  {
    aux=nodo;
    while (aux!=NULL)
    {
      if (aux->hijoIzquierdo!=NULL)
      {
        aux=aux->hijoIzquierdo;
      }
      else //aux es nodo que contiene dato
      {
        return aux;
      }
    }
  }
  return NULL;
}
```

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Implementación de ABB usando punteros

```
/*-------------- operaciones de actualización -----------------*/

void insertarNodoABB(TDAabb* arbol, int dato)
{
  nodoABB* nuevo=(nodoABB*)malloc(sizeof(nodoABB));
  nuevo->dato=dato;
  nuevo->hijoIzquierdo=NULL;
  nuevo->hijoDerecho=NULL;

  if (!esABBvacio(*arbol))
  {
    if (dato <= (*arbol)->dato)  //vamos por la izquierda
    {
      if ((*arbol)->hijoIzquierdo==NULL) // no hay un hijo a la izquierda
      {
        (*arbol)->hijoIzquierdo=nuevo;
        nuevo->padre=*arbol;
      }
      else  //si ya hay un hijo a la izquierda
        insertarNodoABB(&((*arbol)->hijoIzquierdo),dato);
    }
    else //vamos por la derecha
    {
      if ((*arbol)->hijoDerecho==NULL) // no hay un hijo a la izquierda
      {
        (*arbol)->hijoDerecho=nuevo;
        nuevo->padre=*arbol;
      }
      else  //si ya hay un hijo a la izquierda
        insertarNodoABB(&((*arbol)->hijoDerecho),dato);
    }
  }
  else  // se debe insertar la raíz
  {
    nuevo->padre=NULL;
    *arbol=nuevo;
  }
}
```

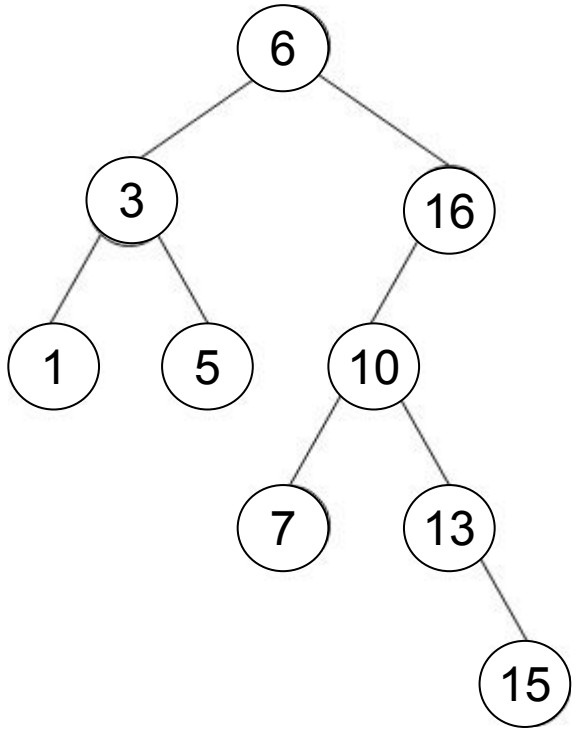
Actividades de implementación

Actividad 1

1. Compilar y ejecutar **lab11-arbolesBinariosBusqueda.c**
2. Crear el árbol mostrado con llamadas en función *main()* de **lab11-arbolesBinariosBusqueda.c**
3. Implementar en conjunto la siguiente función en **TDAabb.h**

```
void recorridoInOrdenABB(TDAabb arbol);
```

4. Mostrar el recorrido en orden después de insertar cada nodo del paso 2 con llamadas en función *main()* de **lab11-arbolesBinariosBusqueda.c**



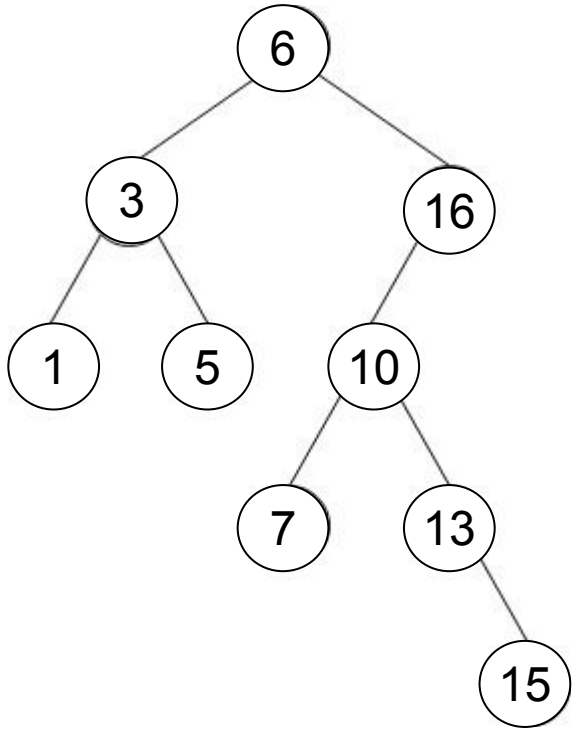
Actividad 2

1. Implementar la siguiente función en **TDAabb.h**

```
TDAabb reconstruirABB(TDAlista preOrden, TDAlista inOrden);
```

que, dadas dos listas enlazadas, cada una con el recorrido preorden e inorden, respectivamente, devuelva un árbol binario de búsqueda (en caso de poder generarse un ABB) reconstruido a partir de ambos recorridos. En caso de que el árbol resultante no fuese un ABB, se debería devolver un puntero nulo.

2. Mostrar el ABB creado luego de ingresar recorrido pre-orden e in-orden del árbol mostrado con llamada en función *main()* de **lab11-arbolesBinariosBusqueda.c**



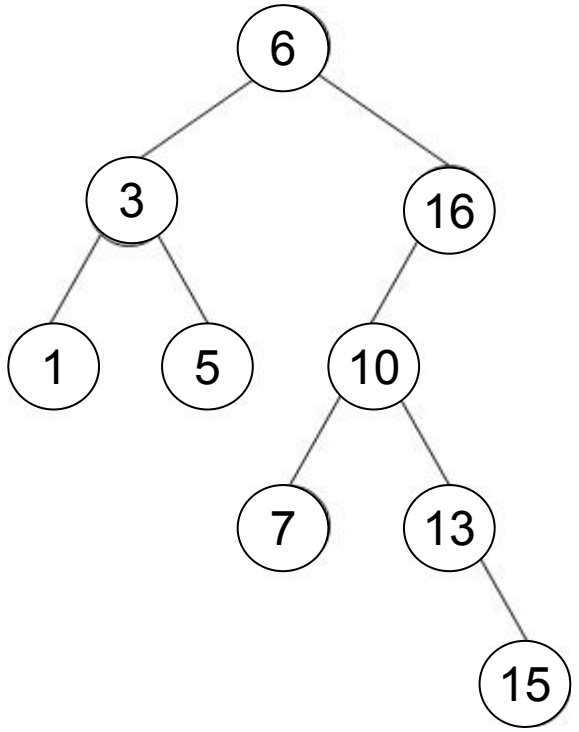
Actividad 3

1. Implementar la siguiente función en **TDAabb.h**

```
void corregirABB(TDAabb arbol);
```

para identificar (mostrar en pantalla) los dos elementos de árbol que han sido intercambiados por error.

2. Suponer que se han intercambiado los datos 5 y 13 del árbol mostrado. Con ello, evaluar la función corregirABB con llamada en función *main()* de **lab11-arbolesBinariosBusqueda.c**



Entrega de actividad de laboratorio

- Entrega *obligatoria*
- Subir actividad 2 y 3 de esta sesión en buzón de uVirtual, en único archivo **s11_apellido_nombre.zip**
- Se espera **lab11-arbolesBinariosBusqueda.c** y **TDAabb.h** (modificados para responder a actividades 2 y 3) comprimidos en archivo .zip
- Plazo: **hoy** dentro del horario de laboratorio de cada coordinación

Actividad de cierre

código: 72 84 08 8

Próximas fechas...

- Resumen de la semana:
  - TDA árbol
  - TDA árbol binario
  - TDA árbol binario de búsqueda

  ~~cátedra~~ – ~~refuerzo~~ – ~~laboratorio~~

- Próxima semana:
  - TDA árbol AVL

U4 - S11