



ARBOLES BINARIOS ORDENADOS. REPRESENTACIÓN Y OPERACIONES

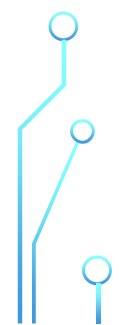
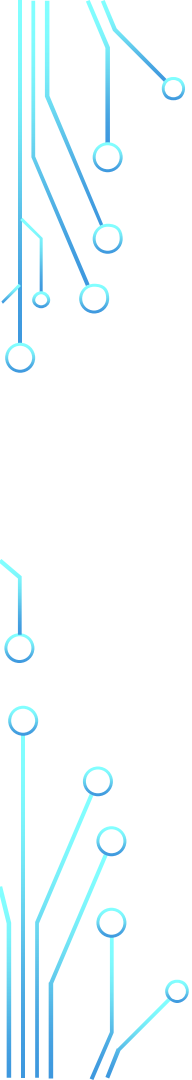
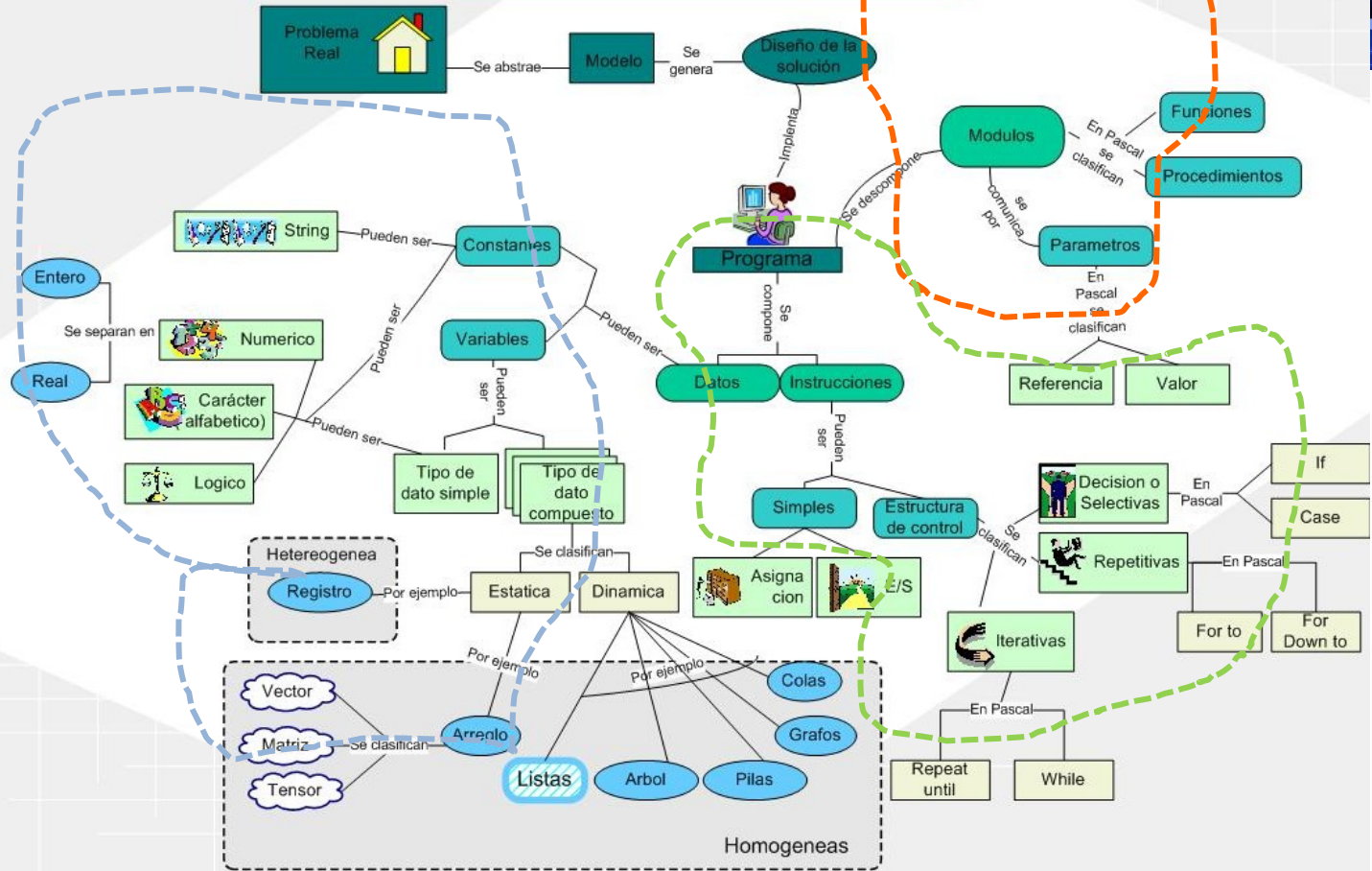
TEMAS

- Recorrido de un árbol
- Máximo y mínimo
- Búsqueda de un elemento
- Borrado de un nodo



MAPA CONCEPTUAL

MATERIA: PROGRAMACIÓN DE COMPUTADORAS



ARBOLES - CONCEPTOS



Características

Cada elemento del árbol se relaciona con cero o más elementos a quienes llama **hijos**.

Si el árbol no está vacío, hay un único elemento al cual se llama **raíz** y que no tiene **padre** (predecesor), es decir, no es hijo de ningún otro.

Todo otro elemento del árbol posee un único padre y es un descendiente (hijo del hijo del hijo, etc.) de la raíz.

ARBOLES - CONCEPTOS



Características

Es una estructura de datos homogénea.

Es una estructura de datos dinámica.

Es una estructura no lineal, ya que cada nodo puede tener 0,1 o más sucesores.

Es una estructura de datos jerárquica.

ABO- OPERACIONES

Operaciones continuación:

- Recorridos posibles
- Mínimo y máximo
- Borrar un nodo



Árboles Binarios Ordenados - Operaciones

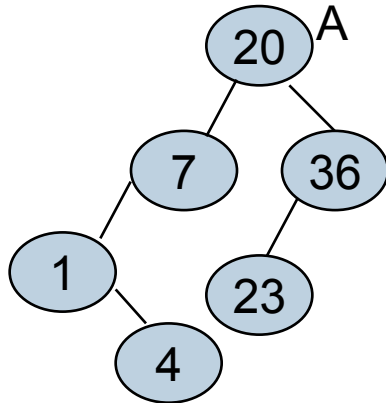


- Recorrido de un árbol: permite desplazarse a través de un árbol en forma tal que, cada nodo sea visitado una y solo una vez.

Existen varios métodos

- **Recorrido En – Orden**
- **Recorrido Pre – Orden**
- **Recorrido Post – Orden**

Árboles Binarios Ordenados - Operaciones



```
Procedure enOrden
```

```
Begin
```

```
If a <> nil then
```

```
  if ( a^.izq <> nil ) then enOrden (a^.izq),
```

```
  write (a^.dato);
```

```
  if ( a^.der <> nil ) then enOrden (a^.der);
```

```
end;
```

¿Qué pasa con la eficiencia en cantidad de operaciones con esta solución?, comparar con el algoritmo visto la clase pasada

Imprime los elementos de menor a mayor

1 4 7 20 23 36

Árboles Binarios Ordenados - Operaciones

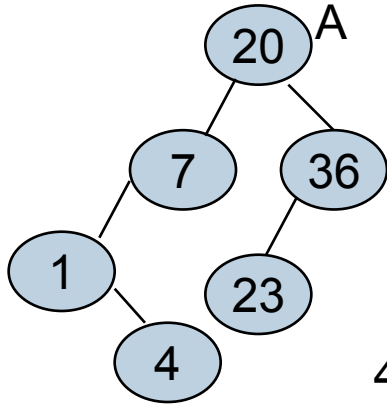
Una opción mas eficiente para el recorrido en orden, porque?



```
Procedure enOrden ( a : arbol );  
Begin  
  If a <> nil then  
    enOrden (a^.izq);  
    write (a^.dato);  
    enOrden (a^.der);  
  end;
```

Se recorre el árbol en orden. Este tipo de recorrido se usa para recorrer todos los nodos de menor a mayor o de mayor a menor (invirtiendo los llamados). Una posible aplicación sería equilibrar las ramas de un árbol; la forma de hacer este procedimiento es: recorremos el árbol en inorden y vamos guardando los elementos en un array; como este array estará ordenado, hacemos un recorrido binario del mismo y vamos reinsertando los elementos en un nuevo árbol que finalmente estará equilibrado. Para usar el array en Pascal debemos suponer una cierta dimensión posible

Árboles Binarios Ordenados - Operaciones



Imprime los hijos y después los padres

4 1 7 23 36 20

```
Procedure postOrden ( a : arbol );
```

```
begin
```

```
  if ( a <> nil ) then begin
```

```
    postOrden ( a^.izq)
```

```
    postOrden ( a^.der)
```

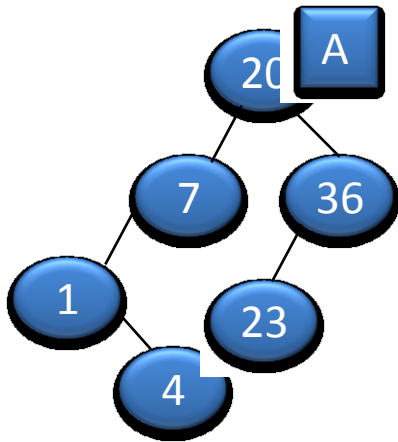
```
    write ( a^.dato)
```

```
  end;
```

```
end;
```

Primero se procesan los hijos y después el padre. Este tipo de recorrido se puede utilizar, por ejemplo, para borrar todos los nodos de un árbol. De esta forma evitamos perder nodos ya que si usásemos otro recorrido, como por ejemplo el recorrido en pre-order, estaríamos borrando primero el nodo padre, con lo que habríamos perdido la dirección de los nodos que vamos a borrar.

Árboles Binarios Ordenados - Operaciones



20 7 1 4 36 23

```
Procedure preOrden ( a : arbol );  
begin  
  if ( a <> nil ) then begin  
    write ( a^.dato )  
    preOrden ( a^.izq )  
    preOrden ( a^.der )  
  end;  
end;
```

Primero se procesan los nodos "padre" y después los nodos "hijo". Este tipo de recorrido se usa, por ejemplo, para guardar un árbol en un fichero y reconstruirlo posteriormente exactamente cómo estaba. Cuando queramos leer de fichero para reconstruir el árbol lo único que hay que hacer es ir insertando los elementos en el árbol según el orden en que están dispuestos en el fichero. Así conseguimos construir un árbol idéntico al que guardamos.

Árboles Binarios Ordenados – Operaciones



¿Cómo se localiza el nodo con valor mínimo en un árbol binario de búsqueda en forma recursiva?

¿Cómo se localiza el nodo con valor máximo en un árbol binario de búsqueda en forma recursiva?

ABO – OPERACIONES – MÍNIMO



Program uno;

Type

arbol = ^nodo;

nodo = record

elem:integer;

hi:arbol

hd:arbol

end;

{Implementar procedimientos}

Var

a,pundato: arbol; n:integer;

Begin

inicializar(a); cargar(a);

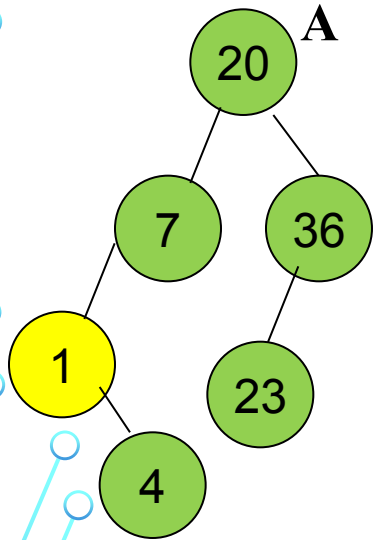
pundato:= **mínimo(a)**;

if (pundato <> nil) then

write(pundato^.elem);

End.

ABO – OPERACIONES – MÍNIMO



Se comienza el recorrido por la raíz (20).

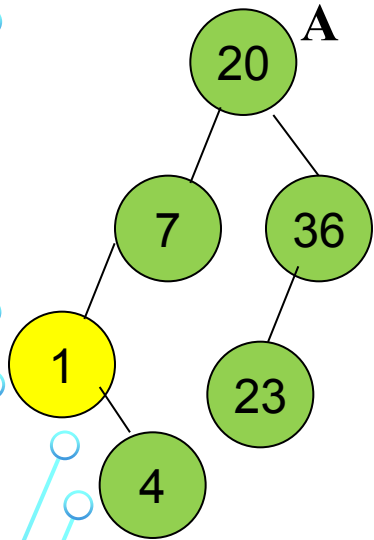
Al tener hijo izquierdo se elige el mismo para seguir el recorrido (7).

Como el 7 también posee hijo izquierdo se toma nuevamente su hijo izquierdo para seguir su recorrido.

Por último como el 1 no posee hijo izquierdo se determina que es el valor más chico del árbol.

¿Qué ocurre si el árbol es nil?

ABO – OPERACIONES – MÍNIMO



```
function minimo ( a:arbol): arbol;    {Recursiva}
begin
  if a=nil then minimo:=nil
  else if a^.izq = nil then minimo:= a
  else minimo:=minimo(a^.izq)
end;
```

¿Y el máximo?

Árboles Binarios Ordenados - Operaciones



```
Function Buscar_Max ( a:arbol): arbol;
```

{Iterativa}

```
begin
```

```
if a <> nil then
```

```
    while (a^.der<> nil) do
```

```
        a := a^.der;
```

```
    Buscar_max := a;
```

```
end;
```


Árboles Binarios Ordenados - Operaciones



Cómo utilizo los algoritmos anteriores para realizar un módulo que devuelva la suma de los elementos del árbol?

```
Program uno;  
Type  
  arbol....  
  
Var  
  a: arbol;  
  sum: integer;  
Begin  
  sum:=0;  
  a:=nil;  
  cargar (a); {generar árbol}  
  sumar (a, sum);  
End.
```

```
Procedure sumar (a: arbol; var s: integer);  
Begin  
  If (a <> nil) then begin  
    s:= s + a^.dato;  
    sumar (a^.hi, s);  
    sumar (a^.hd, s);  
  end;  
End.
```

¿Inicialización de s?
El árbol pasado por valor.

Árboles Binarios Ordenados - Operaciones



- Operación Buscar: localiza un elemento X en el árbol

Buscar (A: arbol; x:elemento) : arbol

Esta operación retorna un puntero al nodo en el árbol A que tiene valor x o Nil si no existe.

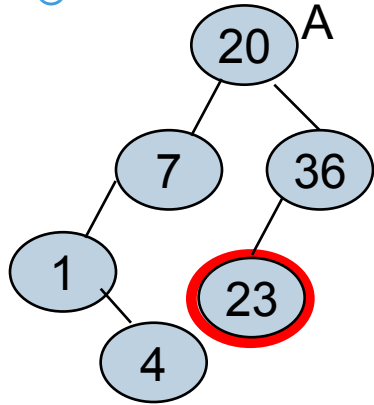
Se presentan una solución recursiva y una solución iterativa

La versión iterativa de la operación Buscar devuelve un valor booleano que indica si el dato se encuentra o no en el árbol.

Árboles Binarios Ordenados - Operaciones



Supongamos que se quiere buscar el valor 23.



¿Qué debe devolver si el árbol está vacío?

¿Para que sirve que sea un ABB?

Empiezo recorriendo el árbol por su raíz, como no es el valor buscado se compara $20 < 23$, por lo tanto la búsqueda debe continuarse por el subárbol derecho a 20.

Luego se compara 36 con 23, como no es el valor buscado y $23 < 36$ se debe buscar por su árbol izquierdo.

Al comparar $23 = 23$ se devuelve ese nodo.

¿Si el elemento que buscábamos es 24 hasta donde recorro?

Árboles Binarios Ordenados - Operaciones

Solución Recursiva

```
Function Buscar (a:arbol; x:elemento): arbol;
```

```
begin
```

```
  if (a=nil) then Buscar:=nil
```

```
    else if (x= a^.dato) then Buscar:=a
```

```
      else
```

```
        if (x < a^.dato) then
```

```
          Buscar:=Buscar(a^.izq ,x)
```

```
        else
```

Árboles Binarios Ordenados - Operaciones

Solución Iterativa

Function Buscar (A : arbol; Dato:itemType): Boolean;

{ Retorna True si Dato es un nodo del árbol, False en caso contrario }

Var auxi : arbol;

Begin

auxi := A;

while (auxi <> nil) **and** (auxi^.dato <> Dato) **do**

if Dato < auxi^.dato **Then** auxi := auxi^.izq

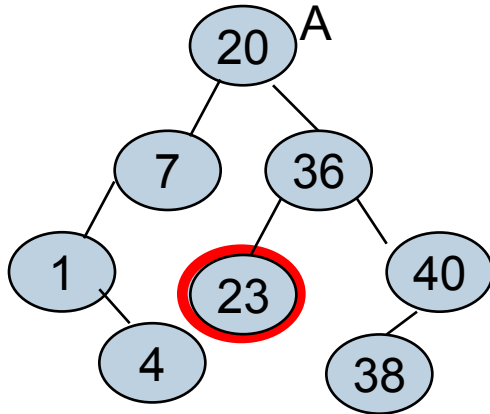
Else auxi := auxi^.der;

Buscar := (auxi <> nil);

Árboles Binarios Ordenados - Operaciones

• Borrado de un nodo del árbol

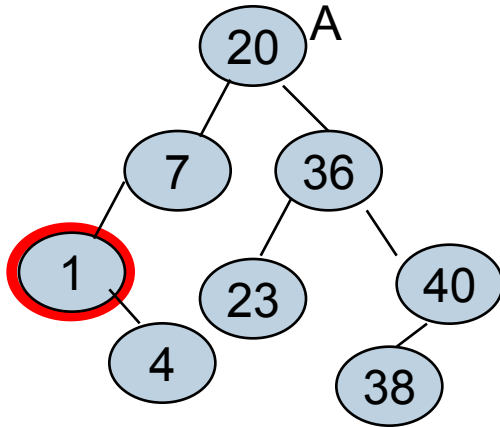
Se deben considerar diferentes situaciones:



1. Si el nodo es una hoja

✓ Se puede borrar inmediatamente (actualizando direcciones)

Árboles Binarios Ordenados - Operaciones



2. Si el nodo tiene un hijo

- ✓ Si **el nodo tiene un hijo**, el nodo puede ser borrado después que su padre actualice el puntero al hijo del nodo que se quiere borrar.

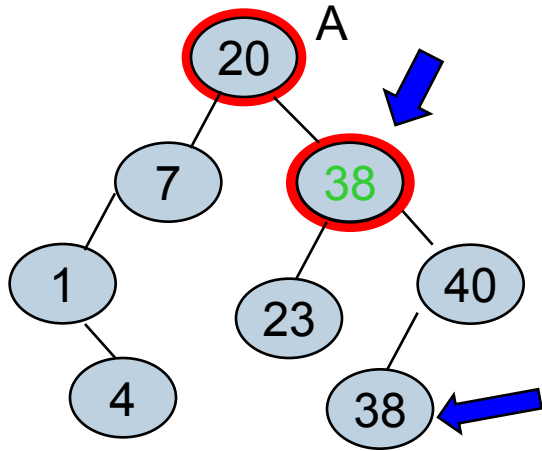
Árboles Binarios Ordenados - Operaciones



3. Si el nodo tiene dos hijos

Se asume una estrategia...

1. Se busca el valor a borrar (ej 36).
2. Se busca y selecciona el hijo mas a la izquierda del subárbol derecho del nodo a borrar (o el hijo mas a la derecha del subárbol izquierdo). ¿Por qué?
3. Se intercambia el valor del nodo encontrado por el que se quiere borrar
4. Se llama al borrar a partir del hijo derecho con el valor del nodo encontrado. ¿Qué característica tiene ese nodo encontrado?



Árboles Binarios Ordenados - Borrado



```
Procedure Borrar ( x:elemento; var a: arbol; var ok:boolean);
  Var
    aux : arbol;
  begin
    if a=nil then ok:=false
    else begin
      if (x<a^.dato) then Borrar(x,a^.izq,ok) {Busco en el subarbol izquierdo}
      else if (x>a^.dato) then Borrar (x,a^.der,ok) {Busco en el sub.derecho}
      else begin {solo hijo a derecha}
        if a^.izq =nil then begin
          aux := a;
          a := a^.der;
          dispose (aux);
        end
      end
    end
  end
```

Árboles Binarios Ordenados - Borrado

```
else {solo hijo a izquierda}
```

```
if a^.der = nil then begin
```

```
    aux := a;
```

```
    a := a^.izq;
```

```
    dispose (aux);
```

```
end
```

```
{2 hijos. Reemplazo con el más pequeño de la derecha}
```

```
else begin
```

```
    aux := buscar_Min(a^.der)
```

```
    a^.dato := aux^.dato;
```

```
    Borrar(a^.dato, a^.der, ok);
```

```
End
```

Procedure Borrar (x:elemento; var a: arbol; var ok:boolean);

Var

aux : arbol;

begin

if a=nil then ok:=false

else begin

if (x<a^.dato) then Borrar(x,a^.izq,ok) {Busco en el subarbol izquierdo}

else if (x>a^.dato) then Borrar (x,a^.der,ok) {Busco en el sub.derecho}

else begin *{solo hijo a derecha}*

if a^.izq =nil then begin

aux := a;

a := a^.der;

dispose (aux);

end

else*{solo hijo a izquierda}*

if a^.der =nil **then begin**

aux := a;

a := a^.izq;

dispose (aux);

end

{2 hijos. Reemplazo con el más pequeño de la derecha}

else begin

aux := buscar_Min(a^.der)

a^.dato := aux^.dato;

Borrar(a^.dato,a^.der,ok);

End

End

End

AGREGAR A UN ÁRBOL CUANDO HAY ELEMENTOS REPETIDOS



- Podemos modificar el algoritmo base para poder agregar repetidos.
- Soluciones posibles:
 - a) Contar la cantidad de ocurrencias de un elemento y llevar un contador
 - b) Agregar al nodo una estructura de datos para guardar todos los datos, por ejemplo tener un puntero a una lista o árbol para guardar todos los datos

AGREGAR CONTANDO CANTIDAD DE OCURRENCIAS



- Suponer que se dispone de una lista de números enteros ya cargada desordenada y con repetidos. Se quiere generar una árbol binario ordenado donde en cada nodo se disponga el valor del número y la cantidad de repeticiones que hubo de ese número en la lista

AGREGAR CONTANDO CANTIDAD DE OCURRENCIAS



```
Program uno;  
Type  
arbol = ^nodo;  
nodo = record  
  elem:integer;  
  cant:integer;  
  izq:arbol  
  der:arbol  
end;
```

```
Type  
lista = ^nodolis;  
nodolis = record  
  num:integer;  
  sig :lista,  
end;
```

```
{Definir procedimientos}  
Var  
a: arbol; n:integer; l:lista,  
Begin  
  l:= nil;  
  cargar(l);  
  While l <> nil do begin  
    agregar(a,l^.num);  
    l:=l^.sig;  
  End;  
End.
```

AGREGAR CONTANDO CANTIDAD DE OCURRENCIAS

```
Procedure Agregar ( var A : arbol; n: integer); begin
```

```
  if A = nil Then begin { llegué al final de la rama }
```

```
    New( A );
```

```
    A^.dato := n;
```

```
    A^.cant:= 1;
```

```
    A^.izq := nil;
```

```
    A^.der := nil;
```

```
  end
```

```
  else
```

```
    If (A^.dato = n) then A^.cant := A^.cant + 1;
```

```
      else if (n < A^.dato) then Agregar(A^.izq, n)
```

```
        else Agregar(A^.der, n);
```

```
End;
```



REPASO DE ESTRUCTURAS DE DATOS CONOCIDAS HASTA EL MOMENTO



Estructuras de Datos - Características

1. Tienen un comportamiento propio

2. Soportan operaciones

→ propias

→ generales

REPASO DE ESTRUCTURAS DE DATOS CONOCIDAS HASTA EL MOMENTO



3. Representan una colección de elementos que desde el punto de vista del dispositivo donde residen pueden ser:

→ **persistentes**

archivos (no los estudiamos acá...)

→ **volátiles**

arreglos

registros

conjunto

listas enlazadas

árboles

string

REPASO DE ESTRUCTURAS DE DATOS CONOCIDAS HASTA EL MOMENTO



4. La colección de elementos pueden ser:

→ homogéneos

arreglos

conjunto

listas enlazadas

árboles

string

→ heterogéneos

registros

REPASO DE ESTRUCTURAS DE DATOS CONOCIDAS HASTA EL MOMENTO

5. Se pueden implementar de forma:

→ **estática**

→ **dinámica**

6. Se accede a sus elementos de manera:

→ **secuencial**

→ **directa**



ARBOLES - CONCEPTOS



LISTAS SIMPLES

Homogénea

Dinámica

Un sucesor para c/nodo

Un solo predecesor

Lineal

ARBOLES

Homogénea

Dinámica

Múltiples sucesores p/c/ nodo

Un solo predecesor

Estructura jerárquica