Listas

- Definición de lista
- Implementación
  - **- Estática**
  - **- Dinámica**

Listas

- **<u>Definición</u>:** Una **lista** es una colección de elementos homogéneos (del mismo tipo), con una relación lineal entre

- Los elementos pueden o no estar ordenados con respecto a algún valor y se puede acceder a cualquier elemento de la

- En realidad, las pilas y colas vistas en secciones anteriores son listas, con algunas restricciones.

Listas

Operaciones sobre listas:

Crear

¿Esta vacía?

¿Está llena?

Insertar un elemento
- De forma ordenada
- De forma no ordenada

Eliminar un elemento

Imprimir

Destruir

Listas

```
Módulo Mlistas
Definición
    Tipos
        TipoElemento
        TipoLista
    Algoritmo Crear(VAR lista:TipoLista)
    Algoritmo ListaVacia(lista:TipoLista):B
    Algoritmo ListaLlena(lista:TipoLista):B
    Algoritmo Insertar(VAR lista:TipoLista; elem:TipoElemento)
    Algoritmo Eliminar(VAR Lista:TipoLista; elem:TipoElemento)
    Algoritmo Imprimir(lista:TipoLista)
    Algoritmo Destruir(VAR lista: TipoLista)
Implementación
Fin
```

Listas

- Implementación:
  - Representación secuencial
  - Representación enlazada
    - Dinámica
    - Estática

Listas

Dinámicas

- Utilizaremos ***punteros*** para crear la l

- Las definiciones de tipos y la implen
operaciones se han visto.

Listas

Estáticas

- Array de registros:
  - Elemento.
  - Enlace al siguiente elemento.

- Los registros pueden almacenarse en cualquier orden físico y se enlazarán a través del campo enlace.

Ejemplo

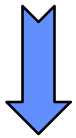
	Elemento	Enlace
1	A	5
2		
3	D	7
4		
5	B	8
6		
7	E	0
8	C	3
9		
10		

Comienzo Lista = 1

Listas

- Inconveniente ⟹ Variable estática ⟹ Elección de Tamaño Máximo

- Debemos escribir nuestros propios algoritmos de manejo de memoria



`Asignar` y **`Liberar`**.

- En el array de registros coexistirán dos listas:
  - Nuestra lista enlazada de elementos.
  - Una lista enlazada con el espacio libre disponible.

Listas

Ejemplo

	Elemento	Enlace
1	A	5
2		6
3	D	7
4		9
5	B	8
6		4
7	E	0
8	C	3
9		10
10		0

Lista = 1

Libre = 2

- La variable Lista contiene el índice del donde comienza la lista enlazada de elementos.
- La variable Libre contiene el índice del donde comienza la lista de espacio libre en el .

Listas

Consideraciones

- El cero (o cualquier otro valor distinto de [1..10]) hará las **NIL** en las listas enlazadas con punteros.
- Cuando no existe ningún elemento en la lista (lista vacía) sólo existirá la lista libre, que enlaza todas las posiciones del .
- Si la lista se llena, no tendremos lista libre.
- Para simular el manejo de memoria real mediante asignación dinámica de memoria, consideramos el y **Libre** como globales al módulo de implementación de la lista.

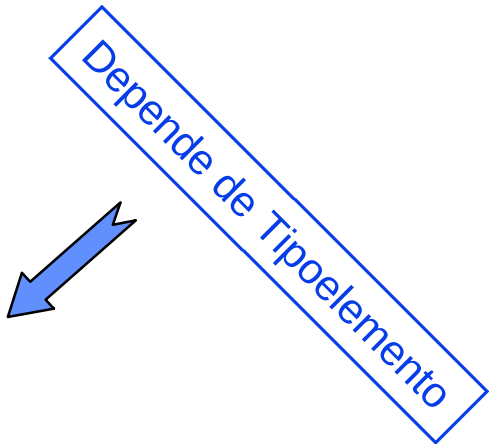
```
Módulo Mlistas
Definición
    Constantes
        Nulo = 0
        Max = 100
    Tipos
        TipoPuntero = [0..Max]
        TipoElemento = (* cualquier tipo de datos *)
        TipoNodo = REGISTRO DE
                     elemento : TipoElemento
                     enlace : TipoPuntero
                   FINREGISTRO
        Memoria = ARRAY [1..Max] DE TipoNodo
        TipoLista = TipoPuntero
    Algoritmo Crear (VAR lista:TipoLista)
    Algoritmo ListaVacia (lista:TipoLista):B
    Algoritmo ListaLlena (lista: TipoLista):B
    Algoritmo Imprimir (lista: TipoLista)
    Algoritmo Insertar (VAR lista:TipoLista; elem:TipoElemento)
    Algoritmo Eliminar (VAR lista:TipoLista; elem:TipoElemento)
    Algoritmo Destruir (VAR lista:TipoLista)
```

```
Implementación
   Variables
       Nodos : Memoria
       Libre : TipoLista

   Algoritmo Crear(VAR lista:TipoLista)
   Inicio
       lista := Nulo
   Fin

   Algoritmo ListaVacia (lista:TipoLista):B
   Inicio
       DEVOLVER (Lista = Nulo)
   Fin

   Algoritmo ListaLlena (lista:TipoLista):B
   Inicio
       DEVOLVER (Libre = Nulo)
   Fin
```

Para los procedimientos **Insertar** y **Eliminar**, necesitamos antes implementar las operaciones equivalentes a Asignar y Liberar que usamos al trabajar con memoria dinámica.

```
Algoritmo ObtenerNodo(VAR p : TipoPuntero)
Inicio
    p := Libre
    SI (Libre <> Nulo) ENTONCES
          Libre := Nodos[Libre].enlace
    FINSI
Fin

Algoritmo LiberarNodo (VAR p :TipoPuntero)
Inicio
    Nodos[p].enlace := Libre
    Libre := p
    p := Nulo
Fin
```

```
Algoritmo Imprimir (lista : TipoLista)
Variables
        ptr : TipoPuntero
Inicio
        ptr := lista
        MIENTRAS (ptr <> Nulo) HACER
                Escribir(Nodos[ptr].elemento)
                ptr := Nodos[ptr].enlace
        FINMIENTRAS
Fin
```




```
Algoritmo Insertar (VAR lista: TipoLista; elem: TipoElemento)
Variables
        ptr : TipoPuntero
Inicio
        ObtenerNodo(ptr)
        SI (ptr<> Nulo) ENTONCES
                Nodos[ptr].elemento := elem
                Nodos[ptr].enlace := lista
                lista := ptr
        FINSI
Fin
```

```
Algoritmo Eliminar (VAR lista: TipoLista; elem :TipoElemento)
Variables    pav, pret : TipoPuntero
Inicio
       pav := lista
       pret := Nulo
       MIENTRAS (Nodos[pav].elemento <> elem) HACER
              pret := pav
              pav := Nodos[pav].enlace
       FINMIENTRAS
       SI (pret = Nulo) ENTONCES
              lista := Nodos[lista].enlace
       EN OTRO CASO
              Nodos[pret].enlace := Nodos[pav].enlace
       FINSI
       LiberarNodo(pav)
Fin
```

Suponemos que el elemento a borrar está en la lista

```
    Algoritmo Destruir (VAR lista: TipoLista)
    Variables
        aux: TipoLista
    Inicio
        MIENTRAS (lista <> Nulo) HACER
                aux := lista
                lista:= Nodos[lista].enlace
                LiberarNodo(aux)
        FIN
    Fin

Inicio  (*  Mlista  *)

        Libre := 1
        PARA ptr := 1 HASTA (Max-1) HACER
                Nodos[ptr].enlace := ptr + 1
        FINPARA
        Nodos[Max].enlace := Nulo

Fin  (*  Mlista  *)
```

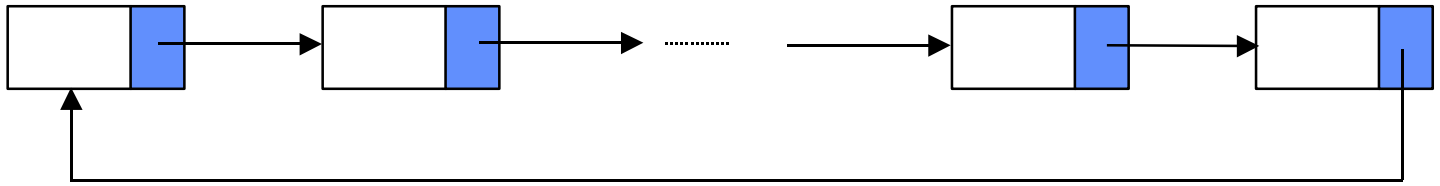
Cuerpo del módulo

Listas enlazadas circulares

- Implementación estática o dinámica.

- El campo de enlace del último nodo apunta al primer nodo de la lista, en lugar de tener el valor **NIL.**

- No existe ni primer ni último nodo. Tenemos un anillo de elementos enlazados unos con otros.



Listas enlazadas circulares

- Es conveniente, aunque no necesario, tener un enlace (puntero o índice) al último nodo lógico de la lista. Así a ambos extremos de la

- Una lista circular vacía vendrá representada por un valor **NIL** o **Nulo** en **p**.

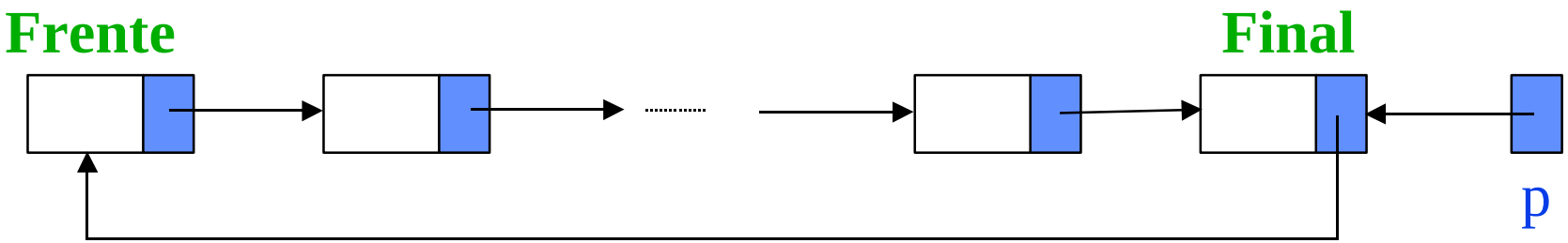
Listas enlazadas circulares

Ejemplo:

```
Algoritmo Imprimir(lista:TipoLista)
Variables
   ptr :TipoPuntero
Inicio
   ptr := lista
   SI (ptr <> Nulo) ENTONCES
      REPETIR
           Escribir(Nodos[ptr].elemento)
           ptr := Nodos[ptr].enlace
      HASTA QUE  ptr = lista
   FINSI
Fin
```

Listas enlazadas circulares

- Con una lista enlazada circular es muy fácil implementar una Cola, sin tener que disponer de un registro con dos campos para el frente y para el final.



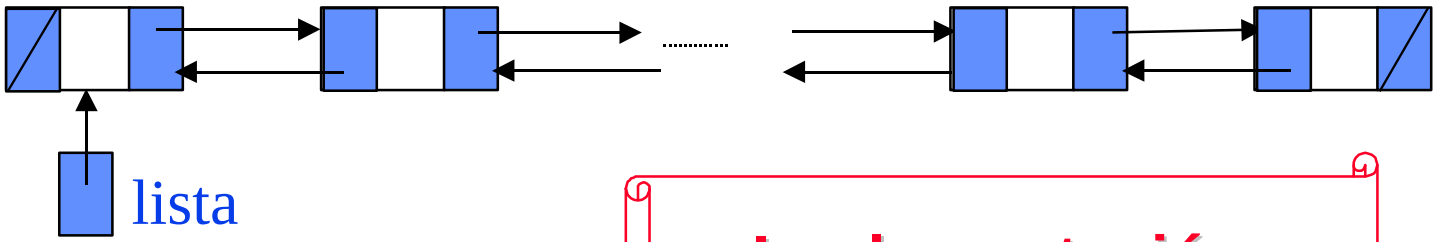
Implementación

Listas doblemente enlazadas

- Es una lista enlazada en la que cada nodo tiene al menos

  - Elemento. El dato de la lista.
  - Enlace al nodo anterior.
  - Enlace al nodo siguiente.

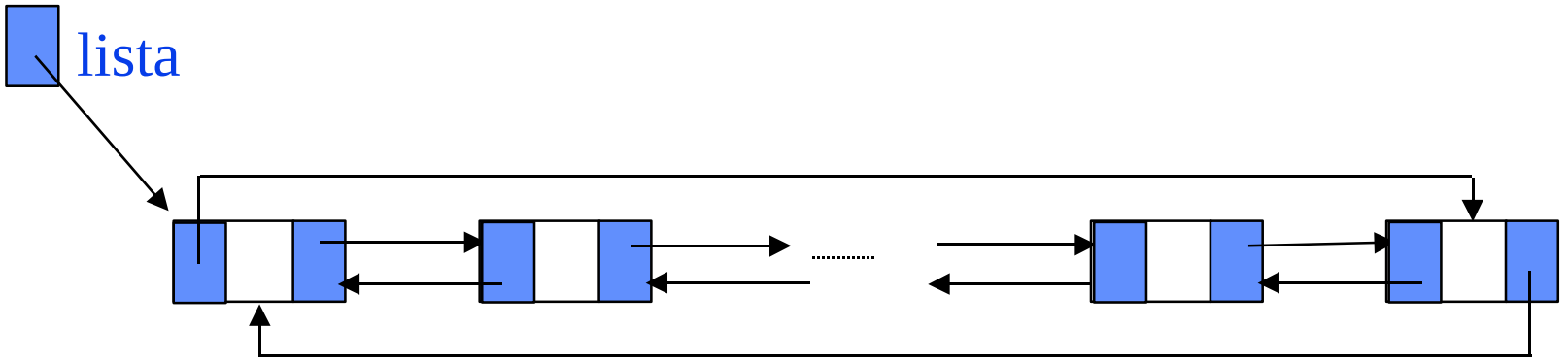
- Los algoritmos para las operaciones sobre listas doblemente enlazadas son normalmente más complicados.
- Pueden ser recorridas fácilmente en ambos sentidos.



Implementación

Listas doblemente enlazadas

- Una lista doblemente enlazada puede modificarse para obtener una estructura circular de la misma



Implementación

Listas

NIVEL DE UTILIZACIÓN

- Estructura muy utilizada.

- **Ejemplo:**

Manejo de tablas hash con el método de encadenamiento para el tratamiento de sinónimos.

  - Utilizaríamos un array de listas como tabla hash.
  - Las listas tienen como elementos cadenas de caracteres.

```
Modulo  Mhash
Definición
   Tipos
      TipoElemento = ARRAY [1..20] DE $
      Rango = [0..99]
      TablaHash = ARRAY Rango DE TipoLista
Implementación
   Algoritmo Almacenamiento(VAR tabla:TablaHash;dato:TipoElemento)
   Inicio
      Insertar(tabla[Hash(dato)],dato)
   Fin
   Algoritmo Búsqueda(tabla:TablaHash; dato:TipoElemento):B
   Inicio
      DEVOLVER Buscar(tabla[Hash(dato)],dato)
   Fin
   Algoritmo Eliminación(VAR tabla:TablaHash;dato:TipoElemento)
   Inicio
      Eliminar(tabla[Hash(dato)],dato)
   Fin
Fin  (*  Mhash  *)
```

Árboles binarios

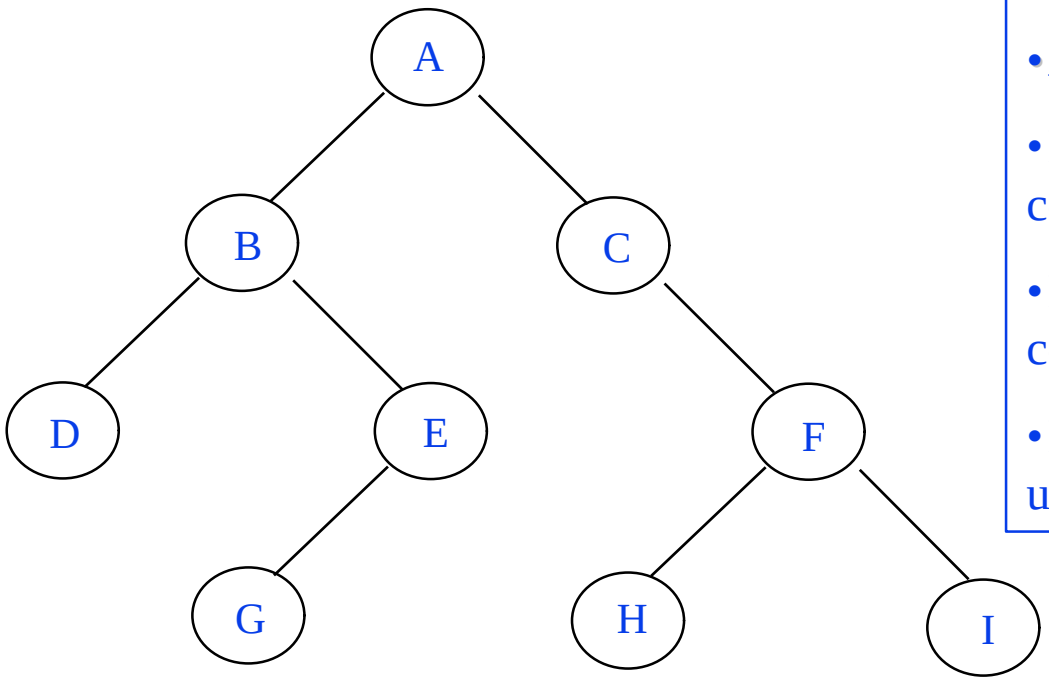
- **Definición:** es un conjunto finito de elementos que está vacío o está partido en tres subconjuntos disjuntos.
  - contiene un único elemento llamado la **raíz** del árbol binario.
  - Los otros dos subconjuntos son a su vez árboles binarios, **subárboles izquierdo** y **derecho**.

- El subárbol izquierdo o derecho puede estar vacío.

- Cada elemento de un árbol binario se denomina **nodo**.

Árboles binarios

- Un método convencional para representar gráficamente un árbol binario es:



- Consta de 9 nodos.
- **A** es el nodo raiz.
- El subárbol izquierdo tiene como nodo raiz **B**.
- El subárbol derecho tiene **C** como raiz.
- La ausencia de ramas indica un árbol vacío.

Árboles binarios

- Si **A** es la raíz de un árbol binario y **B** es la raíz de su subárbol izquierdo o derecho, se dice que **A** es el **padre** de **B** y **B** es el **hijo izquierdo** o **derecho** de **A**.

- Un nodo que no tiene hijos se denomina nodo **hoja**.

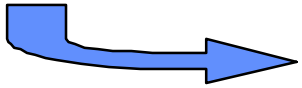
- Un nodo **n1** es un **antecesor** de un nodo **n2** (y **n2** es un **descendiente** de **n1**) si **n1** es el padre de **n2** o el padre de algún antecesor de **n2**.

- Un nodo **n2** es un **descendiente izquierdo** de un nodo **n1** si **n2** es el hijo izquierdo de **n1** o un descendiente del hijo izquierdo de **n1**. Un **descendiente derecho** se puede definir de forma similar.

- **hermanos** si son los hijos izquierdo y derecho del mismo

Árboles binarios

- **Árbol estrictamente binario**: árbol binario en que cada nodo izquierdo y derecho no vacíos.
- **Nivel de un nodo en un árbol binario**: La raíz tiene nivel 0, y el nivel de cualquier otro nodo en el árbol es uno más que el nivel de su padre.
- **Profundidad de un árbol binario**: máximo nivel de cualquier hoja del árbol.



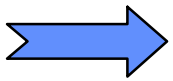
la longitud del camino más largo

- **Árbol binario completo de profundidad d:** árbol estrictamente binario con todas sus hojas con nivel d.

Árboles binarios

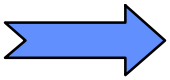
Un árbol binario contiene *m* nodos en el nivel *L*.

Contiene como máximo **2m** nodos en el nivel **L+1**.



Puede contener como máximo 2^L nodos en el nivel **L**

Un árbol binario completo de profunfidad **d** contiene exactamente 2^L nodos en cada nivel **L**, entre **0** y **d**



El número total de nodos en un árbol binario completo de profundidad **d** es:

$$t_n = 2^0 + 2^1 + 2^2 + \ldots + 2^d = 2^{d+1} - 1$$

Árboles binarios

Árbol ternario: conjunto finito de elementos que está vacío o está partido en cuatro subconjuntos disjuntos.

- El primer subconjunto contiene un único elemento llamado la **raíz** del árbol.
- Los otros tres subconjuntos son a su vez árboles.

Árbol n-ario: conjunto finito de elementos que está vacío o está **n+1** subconjuntos disjuntos.

- El primer subconjunto contiene un único elemento llamado la **raíz** del árbol.
- Los otros **n** subconjuntos son a su vez árboles.

Árboles binarios

Operaciones sobre árboles:

Crear árbol vacío

Crear árbol con sólo la raíz

¿Esta vacío?

¿Está lleno?

Crear subárbol derecho

Crear subárbol izquierdo

Devolver el contenido del nodo raíz

Módulo ModArbol

Definición

Implementación dinámica

Tipos

```
TipoElemento = (* Cualquier tipo de datos *)
TipoABinario = PUNTERO A ElementoABinario
ElementoABinario = REGISTRO DE
                       dato : TipoElemento
                       izq,der : TipoABinario
                   FINREGISTRO
```

Algoritmo CrearVacio(**VAR** arbol:TipoABinario)
Algoritmo ArbolVacio(arbol:TipoABinario):**B**
Algoritmo Info(arbol:TipoABinario):TipoElemento
Algoritmo CrearRaiz(x:TipoElemento):TipoABinario
Algoritmo CrearIzq(**VAR** arbol:TipoABinario;x:TipoElemento)
Algoritmo CrearDer(**VAR** arbol:TipoABinario;x:TipoElemento)
Algoritmo Izq(arbol:TipoABinario):TipoABinario
Algoritmo Der(arbol:TipoABinario):TipoABinario
Algoritmo Destruir(**VAR** arbol:TipoABinario)

Implementación

```
Algoritmo CrearVacio(VAR arbol:TipoABinario)
Inicio
    arbol := NIL
Fin


Algoritmo ArbolVacio(arbol:TipoABinario):B
Inicio
    DEVOLVER (arbol = NIL)
Fin


Algoritmo Info(arbol:TipoABinario):TipoElemento
Inicio
    DEVOLVER arbol^.dato
Fin
```

```
Algoritmo Izq(arbol:TipoABinario):TipoABinario
Inicio
   DEVOLVER arbol^.izq
Fin

Algoritmo Der(arbol:TipoABinario):TipoABinario
Inicio
   DEVOLVER arbol^.der
Fin

Algoritmo CrearRaiz(x:TipoElemento):TipoABinario
Variables
   arbol : TipoABinario
Inicio
   Asignar(arbol, Tamaño(ElementoABinadio))
   arbol^.dato := x
   CrearVacio(arbol^.izq)
   CrearVacio(arbol^.der)
   DEVOLVER arbol
Fin
```

```
Algoritmo CrearIzq(VAR arbol:TipoABinario;x:TipoElemento)
Inicio
   Asignar(arbol^.izq, Tamaño(ElementoABinadio))
   arbol^.izq^.dato := x
   CrearVacio(arbol^.izq^.izq)
   CrearVacio(arbol^.izq^.der)
Fin

Algoritmo CrearDer(VAR arbol:TipoABinario;x:TipoElemento)
Inicio
   Asignar(arbol^.der, Tamaño(ElementoABinadio))
   arbol^.der^.dato := x
   CrearVacio(arbol^.der^.izq)
   CrearVacio(arbol^.der^.der)
Fin
```

```
   Algoritmo Destruir(VAR arbol:TipoABinario)
   Inicio
      SI (arbol <> NIL) ENTONCES
            Destruir(arbol^.der)
            Destruir(arbol^.izq)
            Liberar(arbol, Tamaño(ElementoABinadio))
      FINSI
   Fin


Fin  (*  ModArbol  *)
```

Módulo ModArbol

Implementación estática

Definición

Constantes

Nulo = 0

Max = 100

Tipos

```
TipoABinario = [0..Max]
TipoElemento = (* Cualquier tipo de datos *)
ElementoABinario = REGISTRO DE
                       dato : TipoElemento
                       izq,der : TipoABinario
                   FINREGISTRO
Memoria = ARRAY [1..Max] DE ElementoABinario
```

Algoritmo CrearVacio(**VAR** arbol:TipoABinario)

Algoritmo ArbolVacio(arbol:TipoABinario):**B**

Algoritmo Info(arbol:TipoABinario):TipoElemento

Algoritmo CrearRaiz(x:TipoElemento):TipoABinario

Algoritmo CrearIzq(**VAR** arbol:TipoABinario;x:TipoElemento)

Algoritmo CrearDer(**VAR** arbol:TipoABinario;x:TipoElemento)

```
Algoritmo Izq(arbol:TipoABinario):TipoABinario
Algoritmo Der(arbol:TipoABinario):TipoABinario
```

Implementación

Variables

```
    Nodos : Memoria
    Libre : TipoLista
```

TipoLista es una lista estática cuyos nodos son de ElementoABinario

```
Algoritmo CrearVacio(VAR arbol:TipoABinario)
Inicio
    arbol := Nulo
Fin
Algoritmo ArbolVacio(arbol:TipoABinario):B
Inicio
    DEVOLVER (arbol = Nulo)
Fin
Algoritmo Info(arbol:TipoABinario):TipoElemento
Inicio
    DEVOLVER Nodos[arbol].dato
Fin
```

```
Algoritmo Izq(arbol:TipoABinario):TipoABinario
Inicio
   DEVOLVER Nodos[arbol].izq
Fin

Algoritmo Der(arbol:TipoABinario):TipoABinario
Inicio
   DEVOLVER Nodos[arbol].der
Fin

Algoritmo CrearRaiz(x:TipoElemento):TipoABinario
Variables
   arbol : TipoABinario
Inicio
   Asignar(arbol)
   Nodos[arbol].dato := x
   CrearVacio(Nodos[arbol].izq)
   CrearVacio(Nodos[arbol].der)
   DEVOLVER arbol
Fin
```

```
   Algoritmo CrearIzq(VAR arbol:TipoABinario;x:TipoElemento)
   Inicio
      Asignar(Nodos[arbol].izq)
      Nodos[ Nodos[arbol].izq ].dato := x
      CrearVacio(Nodos[ Nodos[arbol].izq ].izq)
      CrearVacio(Nodos[ Nodos[arbol].izq ].der)
   Fin

   Algoritmo CrearDer(VAR arbol:TipoABinario;x:TipoElemento)
   Inicio
      Asignar(Nodos[arbol].der)
      Nodos[ Nodos[arbol].der ].dato := x
      CrearVacio(Nodos[ Nodos[arbol].der ].izq)
      CrearVacio(Nodos[ Nodos[arbol].der ].der)
   Fin

Fin  (*  ModArbol  *)
```

Árboles binarios

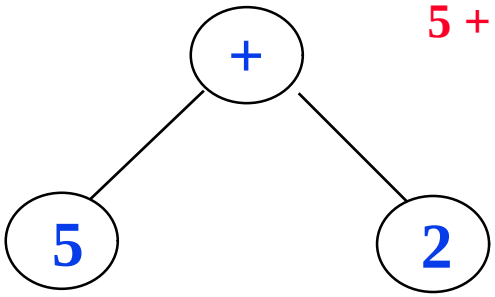
NIVEL DE UTILIZACIÓN

- Estructura de datos muy útil cuando se deben tomar decisiones de "dos

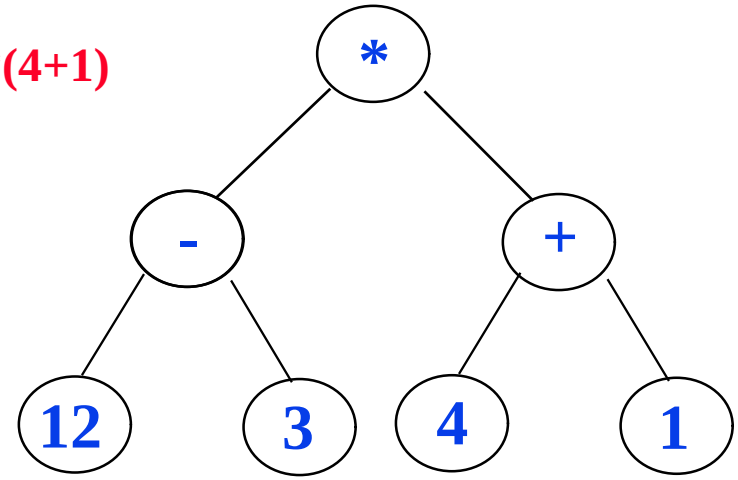
- Muy utilizado en aplicaciones relacionadas con expresiones

Ejemplos:

5 + 2



(12-3)*(4+1)



Árboles binarios

Ejemplo: **Diseñemos un algoritmo para evaluar una expresión aritmética que está almacenada en un árbol binario.**

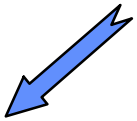
```
Tipos
   TipoDecision = (Operador, Operando)
   TipoElemento = REGISTRO
                      CASO contenido:TipoDecision SEA
                            Operador:   oper: $
                            Operando:   val: R
                      FINCASO
                FINREGISTRO
```

```
Algoritmo Eval(arbol:TipoABinario):R
Variables
   dato:TipoElemento;
   result: R
Inicio
   dato := Info(arbol)
   SI (dato.contenido = Operando) ENTONCES
       result:= dato.val
   EN OTRO CASO
       CASO dato.oper SEA
             '+': result:= Eval(Izq(arbol))+Eval(Der(arbol))
             '-': result:= Eval(Izq(arbol))-Eval(Der(arbol))
             '*': result:= Eval(Izq(arbol))*Eval(Der(arbol))
             '/': result:= Eval(Izq(arbol))/Eval(Der(arbol))
       FINCASO
   FINSI
   DEVOLVER result
Fin
```

Árboles binarios de búsqueda

La estructura lista enlazada es lineal ⟹ ¿Búsqueda?

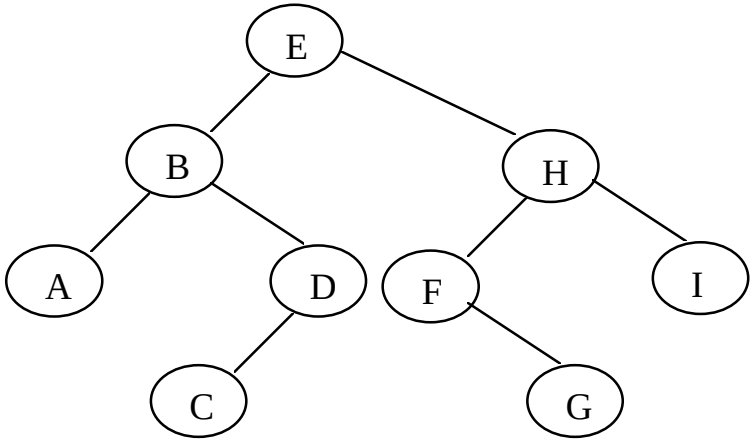
La estructura árbol ⟹ ¿Búsqueda más eficiente?



Árboles binarios de búsqueda

- Definición: árbol binario en el que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el derecho (si no está vacío) contiene valores

Ejemplo:



Árboles binarios de búsqueda

Operaciones:

Crear
Buscar
Insertar
Suprimir
Imprimir

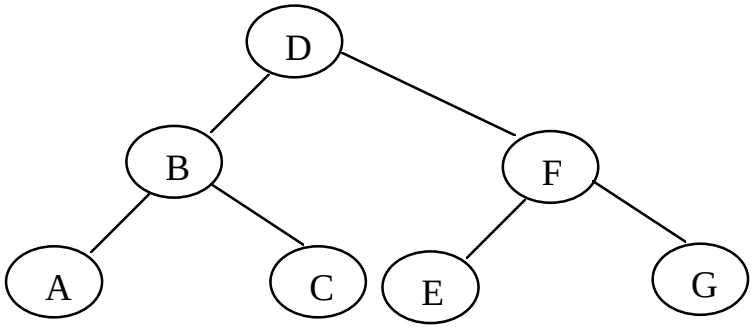
\+ Las definidas para un árbol binario general

```
Módulo MArbolBinarioBusqueda

Definición
   Tipos
      TipoRecorrido = (InOrden,PreOrden,PostOrden)
      TipoClave = Z
      TipoElemento = REGISTRO
                        clave:TipoClave
                        .........
                     FINREGISTRO
      TipoABBúsqueda = PUNTERO A NodoArbol
      NodoArbol = REGISTRO
                        izq,der:TipoABBúsqueda
                        elem:TipoElemento
                  FINREGISTRO
   Algoritmo Crear(VAR arbol:TipoABBúsqueda)
   Algoritmo Buscar(arbol:TipoABBúsqueda;c:TipoClave;
                        VAR dato:TipoElemento;VAR EnArbol:B)
   Algoritmo Insertar(VAR arbol:TipoABBúsqueda;
                                 dato:TipoElemento)
   Algoritmo Suprimir(VAR arbol:TipoABBúsqueda;c:TipoClave)
   Algoritmo Imprimir(arbol:TipoABBúsqueda;rec:TipoRecorrido)
```

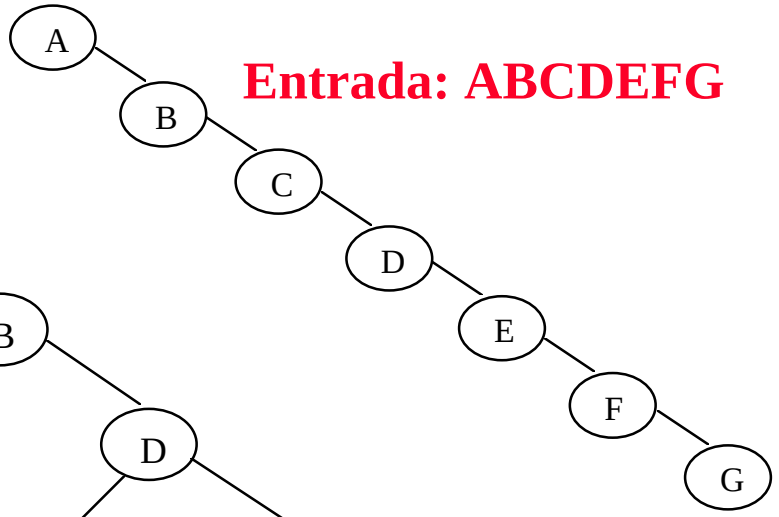
```
Implementación
        Algoritmo Crear(VAR arbol:TipoABBúsqueda)
        Inicio
                arbol := NIL
        Fin
        Algoritmo_Buscar(arbol:TipoABBúsqueda;c:TipoClave;
                                VAR dato:TipoElemento;VAR EnArbol:B)
        Inicio
                EnArbol := FALSE
                MIENTRAS ((arbol<>NIL) AND (NOT EnArbol)) HACER
                                SI (arbol^.elem.clave = c) ENTONCES
                                        EnArbol := TRUE
                                EN OTRO CASO
                                        SI (arbol^.elem.clave > c) ENTONCES
                                                arbol := arbol^.izq
                                        EN OTRO CASO
                                                arbol := arbol^.der
                                        FINSI
                                FINSI
                FINMIENTRAS
                SI EnArbol ENTONCES dato := arbol^.elem
                FINSI
        Fin
```

Árboles binarios de búsqueda

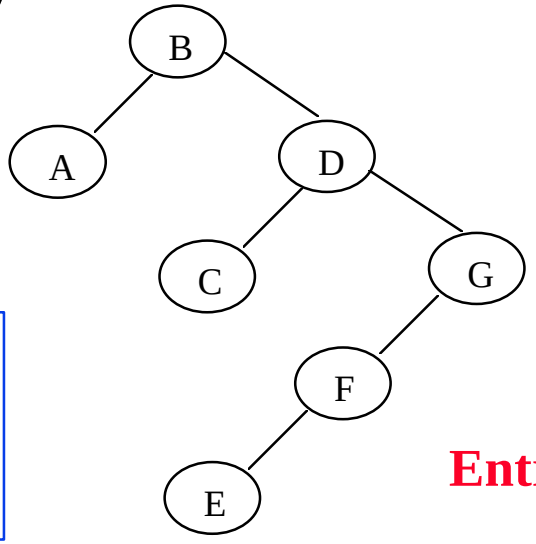
Consideraciones acerca de la operación de inserción.



Entrada: DBFACEG



Entrada: ABCDEFG



Entrada: BADCGFE

Los mismos datos, insertados en orden diferente, producirán árboles con formas o distribuciones de elementos

```
Algoritmo Insertar(VAR arbol:TipoABBúsqueda;
                                    dato: TipoElemento)
Variables      nuevonodo,pav,pret:TipoABBúsqueda
   clavenueva:TipoClave
Inicio
  Asignar(nuevonodo, Tamaño(NodoArbol))
  nuevonodo^.izq := NIL
  nuevonodo^.der := NIL
  nuevonodo^.elem := dato
  clavenueva := dato.clave
  pav := arbol
  pret := NIL
  MIENTRAS (pav <> NIL) HACER
    pret := pav
    SI (pav^.elem.clave>clavenueva) ENTONCES
        pav:= pav^.izq
    EN OTRO CASO
        pav := pav^.der
    FINSI
  FINMIENTRAS
    SI (pret = NIL) ENTONCES
              arbol := nuevonodo
    EN OTRO CASO

       SI (pret^.elem.clave > clavenueva) ENTONCES
              pret^.izq:=  nuevonodo
       EN OTRO CASO
              pret^.der := nuevonodo
       FINSI
    FINSI
Fin
```

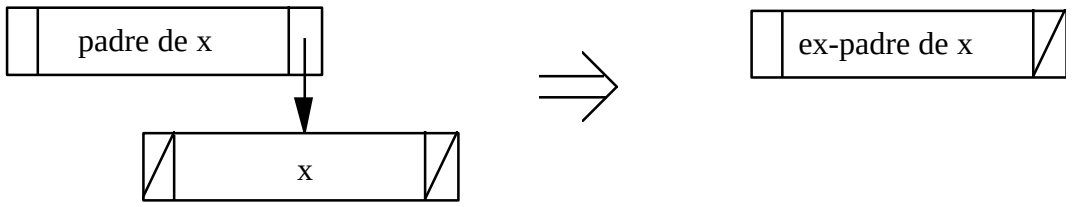
Árboles binarios de búsqueda

Consideraciones acerca de la operación de suprimir.

- Pasos:

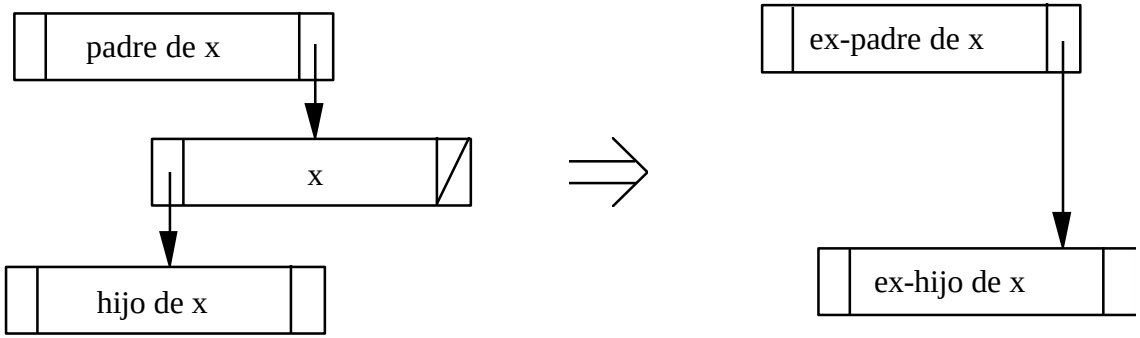
  1) Encontrar el nodo a suprimir. ⇨ Equivalente a Buscar

  2) Eliminarlo del árbol. ⇨

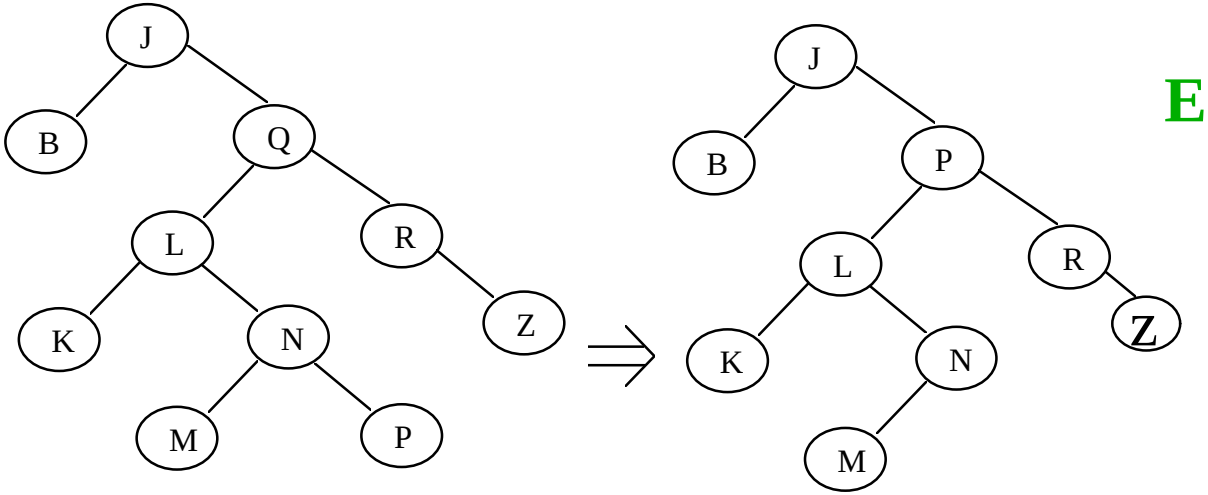


Caso 1

Árboles binarios de búsqueda



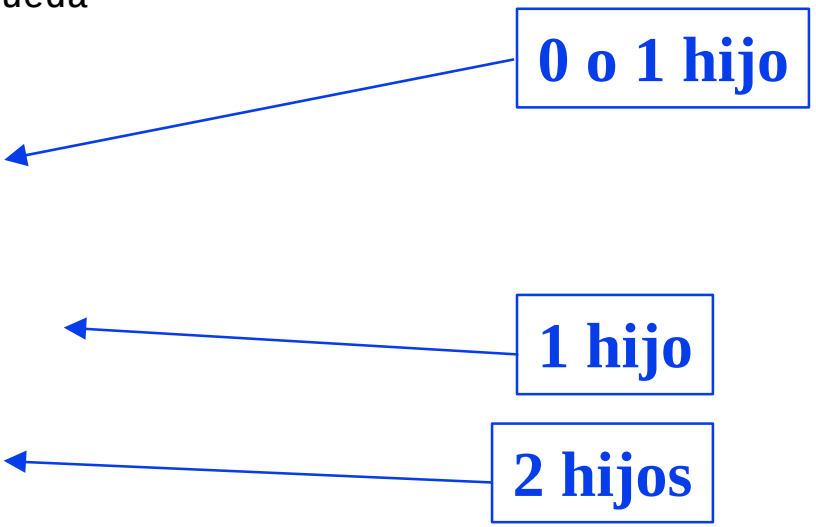
Caso 2



Eliminar Q

Caso 3

```
Algoritmo SuprimirNodo(VAR arbol:TipoABBúsqueda)
Variables                temp,ant:TipoABBúsqueda
Inicio
    temp := arbol
    SI (arbol^.der = NIL) ENTONCES
        arbol := arbol^.izq
    EN OTRO CASO
        SI (arbol^.izq = NIL) ENTONCES
                arbol := arbol^.der
        EN OTRO CASO
                temp := arbol^.izq
                ant := arbol
                MIENTRAS (temp^.der <> NIL) HACER
                        anterior := temp
                        temp := temp^.der
                FINMIENTRAS
                arbol^.elem := temp^.elem
                SI (anterior = arbol) ENTONCES
                        anterior^.izq:= temp^.izq
                EN OTRO CASO
                        anterior^.der := temp^.izq
                FINSI
        FINSI
    FINSI
    Liberar(temp, Tamaño(NodoArbol))
Fin
```



```
Algoritmo Suprimir(VAR arbol:TipoABBúsqueda;c:TipoClave)
Variables
    pav,pret:TipoABBúsqueda
Inicio
    pav := arbol
    pret := NIL
    MIENTRAS (pav^.elem.clave <> c) HACER
            pret := pav
            SI (pav^.elem.clave > c) ENTONCES
                    pav := pav^.izq
            EN OTRO CASO
                    pav := pav^.der
            FINSI
    FINMIENTRAS
    SI (pav = arbol) ENTONCES
            SuprimirNodo(arbol)
    EN OTRO CASO
            SI (pret^.izq = pav) ENTONCES
                    SuprimirNodo(pret^.izq)
            EN OTRO CASO
                    SuprimirNodo(pret^.der)
            FINSI
    FINSI
Fin
```

Árboles binarios de búsqueda

Consideraciones acerca de la operación de imprimir.

- Recorrer un árbol es "visitar" todos sus nodos para llevar a cabo algún proceso como por ejemplo imprimir los elementos que contiene.

- ¿Cómo imprimir los elementos de un árbol? ¿en qué

Árboles binarios de búsqueda

- Para recorrer un árbol binario en general (de búsqueda o no), podemos hacerlo de tres formas distintas:

a) Recorrido **InOrden**.

Pasos:
1) Recorrer el subárbol izquierdo en InOrden
2) "Visitar" el valor del nodo

PreOrden.

1) "Visitar" el valor del nodo

PostOrden.

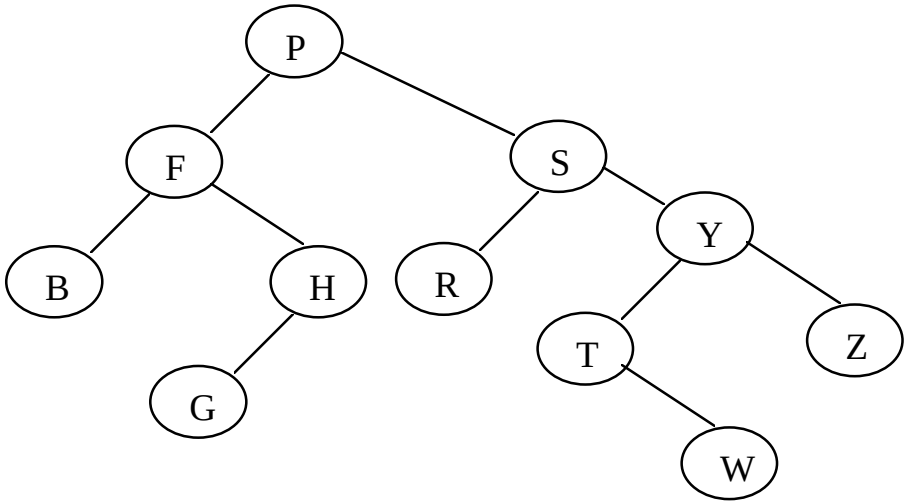
Pasos:
1) Recorrer el subárbol izquierdo en PostOrden
2) Recorrer el subárbol derecho en PostOrden
3) "Visitar" el valor del nodo

```
Algoritmo Imp_InOrden(arbol:TipoABBúsqueda)
Inicio
      SI (arbol <> NIL) ENTONCES
            Imp_InOrden(arbol^.izq)
            ImpNodo(arbol^.elem)
            Imp_InOrden(arbol^.der)
      FINSI
Fin

Algoritmo Imp_PreOrden(arbol:TipoABBúsqueda)
Inicio
      SI (arbol <> NIL) ENTONCES
            ImpNodo(arbol^.elem)
            Imp_PreOrden(arbol^.izq)
            Imp_PreOrden(arbol^.der)
      FINSI
Fin
```

```
   Algoritmo Imp_PostOrden(arbol:TipoABBúsqueda)
   Inicio
      SI (arbol <> NIL) ENTONCES
            Imp_PostOrden(arbol^.izq)
            Imp_PostOrden(arbol^.der)
            ImpNodo(arbol^.elem)
      FINSI
   Fin

   Algoritmo Imprimir(arbol:TipoABBúsqueda; rec:TipoRecorrido)
   Inicio
      CASE rec SEA
            inorden: Imp_InOrden(arbol)
            preorden: Imp_PreOrden(arbol)
            postorden: Imp_PostOrden(arbol)
      FINCASO
   Fin
Fin  (*  MArbolBinarioBusqueda  *)
```

Árboles binarios de búsqueda

Ejemplo:



- El recorrido InOrden mostraría: **BFGHPRSTWYZ**
- El recorrido PreOrden: **PFBHGSRYTWZ**
- El recorrido PostOrden: **BGHFRWTZYSP**

Árboles binarios de búsqueda

Nivel de utilización

- Aplicaciones en las que estén implicas operaciones de búsqueda de elementos
- **Ejemplo**: procesamiento de índices de palabras de

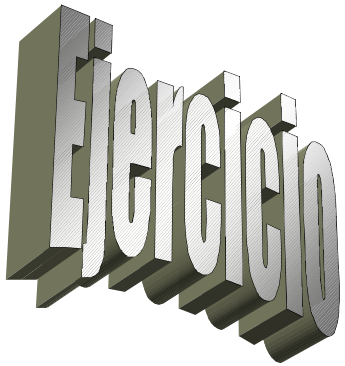
se puede usar un árbol binario de búsqueda para construir y manipular un índice de este tipo de una manera fácil y eficiente.

Árboles binarios de búsqueda

- Supongamos que tenemos construido un árbol binario de búsqueda que contiene un índice de palabras de un determinado libro.
- Queremos realizar actualizaciones sobre él a partir de datos dados por
- Estos datos serán:

- Código operación.
  - **InsertarP**. Insertar una nueva palabra con sus páginas.
  - **SuprimirP**. Suprimir una palabra del índice.
  - **AñadirP**. Añadir más páginas a una palabra.
  - **FinalizarP**. Finalizar la actualización.

⟸ Caso de operación **InsertarP** o **AñadirP**



Tipos

```
TipoCódigo = (Insertar, Suprimir, Añadir, Eliminar)
TipoClave = ARRAY [1..30] DE $
TipoPaginas = REGISTRO
                        total:N
                        num: ARRAY [1..10] DE N
              FINREGISTRO
TipoElemento = REGISTRO
                        clave:TipoClave
                        pag: TipoPaginas
               FINREGISTRO
```

```
Algoritmo ActualizarIndice(VAR indice:TipoABBúsqueda)
Variables
   codigo : TipoCódigo
   elem:TipoElemento
   pag:N
   palabra:TipoClave
   encontrada:B
Inicio
   leerCódigo(codigo)
   MIENTRAS (codigo <> FinalizarP) HACER
       CASO CAP(codigo) SEA
         InsertarP: leer(elem.clave)
                    elem.pag.total := 0
                    leer(pag)
                    MIENTRAS (pag <> 0) HACER
                           elem.pag.total := elem.pag.total + 1
                           elem.pag.num[elem.pag.total] := pag
                           leer(pag)
                    FINMIENTRAS
                    Insertar(indice,elem)
```

```
        SuprimirP: leer(palabra)
                   Suprimir(indice,palabra)

        AñadirP:   leer(palabra)
                   Buscar(indice,palabra,elem,encontrada)
                   SI (NOT encontrada) ENTONCES
                          escribir("palabra no está en el indice")
                   EN OTRO CASO
                          leer(pag)
                          MIENTRAS (pag <> 0) HACER
                                 elem.pag.total :=elem.pag.total + 1
                                 elem.pag.num[elem.pag.total] := pag
                                 leer(pag)
                          FINMIENTRAS
                          Suprimir(indice,palabra)
                          Insertar(indice,elem)
                   FINSI
      FINCASO
      leerCódigo(codigo)
   FINMIENTRAS
Fin
```

Bibliografía

- ***Walls and mirrors. Modula-2 edition.*** Hellman, Veroff.. Ed. Benjamin/Cummings Series. 1988.
- ***Pascal y Estructuras de Datos.*** Dale, N. y Lilly, S. Ed. MacGraw Hill 1989.
- ***Estructuras de Datos en Pascal***. Tanenbaum, A. y Augenstein, M. Prentice Hall. 1983
- ***Fundamentos de Programación. Algoritmos y Estructuras de Datos.*** Joyanes, L. McGraw Hill. 2ª Ed. 1996.