Listas Posicionales

- **Definición:** Una **lista posicional** es una colección de elementos homogéneos, con una relación lineal entre ellos, en la que se puede acceder a los elementos mediante su posición.
- Se puede insertar y suprimir en cualquier posición.
- En realidad, las pilas y colas vistas en secciones anteriores son listas, con algunas restricciones.

Listas posicionales

Operaciones sobre listas:

Crear

¿Esta vacía?

Insertar un elemento en una posición

Eliminar el elemento de una posición

Consultar el elemento de una posición

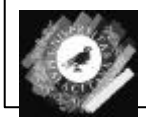
Longitud

Destruir

Listas posicionales Interfaz

```
INTERFAZ CLASE CListaPos
      TIPOS
            TipoElem  .....
      MÉTODOS
       Crear()
       Insertar(N pos;TipoElem elem)
       Eliminar(N pos)
       Consultar(N pos;S TipoElem elem)
       N Longitud()
       Destruir()
FIN CListaPos
```

El tipo se definirá más tarde

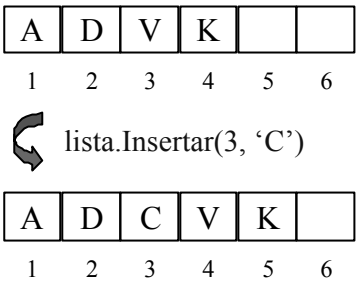


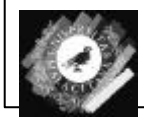
Listas posicionales Implementación

Estáticas

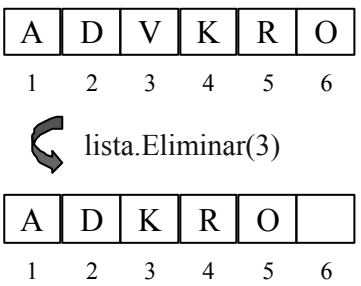
- Utilizaremos ***arrays*** para crear la lista de elementos.
- Al insertar un elemento en la posición "i", desplazamos los siguientes una posición a la derecha.
- Al eliminar el elemento de la posición "i", desplazamos los siguientes una posición a la izquierda.

Listas posicionales Implementación





Listas posicionales Implementación

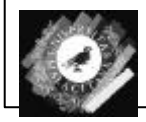


Listas posicionales Implementación

Dinámicas

- Utilizaremos ***punteros*** para crear la lista enlazada de elementos.

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



Tablas

- **Definición:** Una **tabla** es una colección de elementos homogéneos (del mismo tipo), especialmente diseñada para el acceso rápido a la información.

- Se puede acceder a cualquier elemento de la tabla a través de un campo clave.

Tablas

Operaciones sobre tablas:

Crear

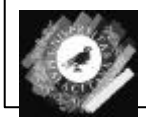
¿Esta vacía?

Insertar un elemento

Eliminar un elemento

Buscar un elemento por clave

Destruir



Tablas

```
INTERFAZ CLASE Tabla
      TIPOS
            TipoClave   // Definicion de tipos
            TipoElemento
      ALGORITMOS
            TipoClave Clave(E TipoElemento e)
      MÉTODOS
            Crear()
            B TablaVacia()
            B TablaLlena()
            Insertar(E TipoElemento elem;S B ok)
            Eliminar(E TipoClave clave; S B ok)
            Buscar(E TipoClave cl; S TipoElemento el; S B ok)
            Destruir()
FIN Tabla
```

Los tipos se definirán más tarde

Tablas

NIVEL DE UTILIZACIÓN

- Estructura muy utilizada.
- **Ejemplo:**

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

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

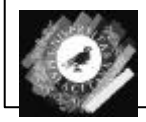
Tablas

Eficiencia de los métodos según la implementación

	Insertar	Eliminar	Buscar
Estática no ordenada	Constante	Lineal	Lineal
Estática ordenada	Lineal	Lineal	Lineal
Lista enlazada no ordenada	Constante	Lineal	Lineal
Lista enlazada ordenada	Lineal	Lineal	Lineal

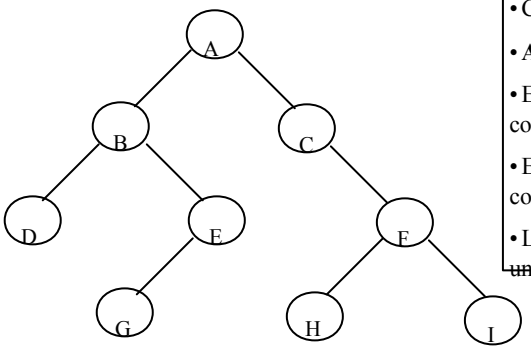
Árboles binarios. Conceptos

- **Definición**: es un conjunto finito de elementos que está vacío o está partido en tres subconjuntos disjuntos.
  - El primer subconjunto contiene un único elemento llamado la **raíz** del árbol binario.
  - Los otros dos subconjuntos son a su vez árboles binarios, llamados **subárboles izquierdo** y **derecho**.
- El subárbol izquierdo o derecho puede estar vacío.
- Cada elemento de un árbol binario se denomina **nodo**.



Árboles binarios. Conceptos

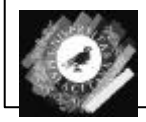
- Un método convencional para representar gráficamente un árbol binario es:



- 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.

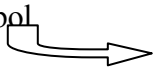
Árboles binarios. Conceptos

- 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.
- Dos nodos son **hermanos** si son los hijos izquierdo y derecho del mismo padre.



Árboles binarios. Conceptos

- **Árbol estrictamente binario**: árbol binario en que cada nodo no-hoja tiene subárboles 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 desde la raiz hasta una hoja.

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

Árboles binarios. Conceptos

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. Conceptos

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.

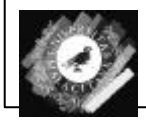
Árbol n-ario: conjunto finito de elementos que está vacío o está partido en **n+1** subconjuntos disjuntos.

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

Árboles binarios. Interfaz

Operaciones sobre árboles:

Crear árbol vacío	Devolver el contenido del nodo raíz
Construir árbol con raíz, izquierdo y derecho	Devolver el subárbol derecho
¿Está vacío?	Devolver el subárbol derecho
	Destruir



Árboles binarios. Interfaz

```
INTERFAZ CLASE ArbolBin
TIPOS
      TipoElemento = (* Cualquier tipo de datos *)
      TipoRecorrido = (preorden, inorden, postorden)
MÉTODOS
      Crear()
      Construir(E TipoElemento x,E ArbolBin hijoizq, hijoder)
      B ArbolVacio()
      TipoElemento Raíz()
      ArbolBin Izq()
      ArbolBin Der()
      Destruir()
      Imprimir(E TipoRecorrido rec)
```

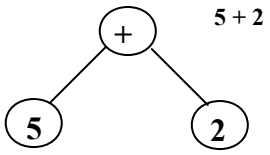
Árboles binarios. Utilización

NIVEL DE UTILIZACIÓN

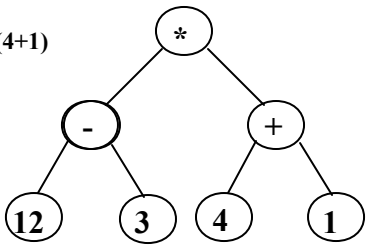
- Estructura de datos muy útil cuando se deben tomar decisiones de "dos caminos"
- Muy utilizado en aplicaciones relacionadas con expresiones aritméticas.

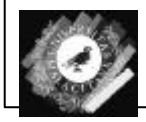
Ejemplos:

5 + 2



(12-3)*(4+1)





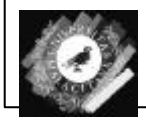
Árboles binarios. Utilización

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

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

Árboles binarios. Utilización

```
ALGORITMO R Eval(E ArbolBin arbol)
VARIABLES
     TipoElemento dato
     R result
INICIO
     dato = Info(arbol)
     SI dato.contenido == Operando ENTONCES
          result = dato.val
     SINO
          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
```

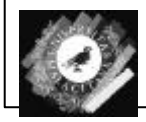
Árboles binarios. Implementación

```
IMPLEMENTACIÓN CLASE ArbolBin
TIPOS
     REGISTRO NodoABin
               TipoElemento dato
               NodoABin *izq, *der
     FINREGISTRO
ATRIBUTOS
     NodoABin *nrz //puntero al nodo raíz del árbol
MÉTODOS
     Crear()
     INICIO
          nrz = NULO
     FIN
```

Árboles binarios. Implementación

```
B ArbolVacio()
INICIO
     DEVOLVER (nrz == NULO)
FIN

Construir(E TipoElemento x, E ArbolBin hijoizq, hijoder)
INICIO
     Asignar(nrz)
     nrz->dato = x
     nrz->izq = hijoizq.nrz
     nrz->der = hijoder.nrz
FIN
```

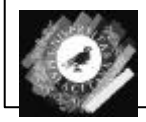



Árboles binarios. Implementación

```
TipoElemento Raíz()
VARIABLES
     TipoElemento r
INICIO
     SI NO ArbolVacio() ENTONCES
          r = nrz->dato
     FINSI
     DEVOLVER r
FIN Raíz
```

Árboles binarios. Implementación

```
ArbolBin Izq()
VARIABLES
     ArbolBin i
INICIO
     SI NO ArbolVacio() ENTONCES
          i.nrz = nrz->izq
     FIN
     DEVOLVER i
FIN Izq
```

Árboles binarios. Implementación

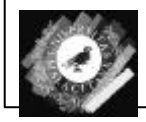
```
ArbolBin Der()
VARIABLES
     ArbolBin d
INICIO
     SI NO ArbolVacio() ENTONCES
          d.nzr = nrz->der
     FIN
     DEVOLVER d
FIN Der
```

Árboles binarios. Implementación

```
ALGORITMO LiberarMemoria(ES NodoABin a)
INICIO
     SI a <> NULO ENTONCES
          LiberarMemoria(a->izq)
          LiberarMemoria(a->der)
          Liberar (a)
     FINSI
FIN LiberarMemoria

Destruir()
INICIO
     LiberarMemoria(nrz)
FIN Destruir

FIN  ArbolBin
```

Árboles binarios. Recorridos

Consideraciones acerca de la operación de recorrido.

- Recorrer un árbol es "visitar" todos sus nodos para llevar a cabo algún proceso como por ejemplo imprimir los elementos que contiene.
- ¿Cómo recorrer los elementos de un árbol? ¿en qué orden?.

Árboles binarios. Recorridos

- Para recorrer un árbol binario, podemos hacerlo de tres formas distintas:

a) Recorrido **InOrden**.

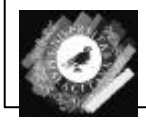
Pasos:
1) Recorrer el subárbol izquierdo en InOrden
2) "Visitar" el valor del nodo raiz y procesarlo
3) Recorrer el subárbol derecho en InOrden

b) Recorrido **PreOrden**.

Pasos:
1) "Visitar" el valor del nodo raiz y procesarlo
2) Recorrer el subárbol izquierdo en PreOrden
3) Recorrer el subárbol derecho en PreOrden

c) Recorrido **PostOrden**.

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



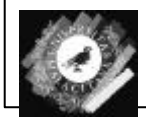
Árboles binarios. Recorridos

```
ALGORITMO Rec_InOrden(E ArbolBin arbol)
INICIO
     SI (arbol <> NULO) ENTONCES
          Rec_InOrden(arbol.Izq())
          RecNodo(arbol.Raíz())
          Rec_InOrden(arbol.Der())
     FINSI
FIN Rec_InOrden
Algoritmo Rec_PreOrden(E ArbolBin arbol)
INICIO
     SI (arbol <> NULO) ENTONCES
          RecNodo(arbol.Raíz())
          Rec_PreOrden(arbol.Izq())
          Rec_PreOrden(arbol.Der())
     FINSI
FIN Rec_PreOrden
```

Árboles binarios. Recorridos

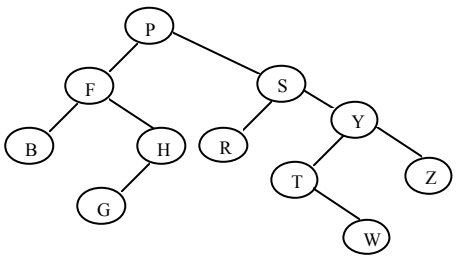
```
ALGORITMO Rec_PostOrden(E ArbolBin arbol)
INICIO
     SI (arbol <> NULO) ENTONCES
          Rec_PostOrden(arbol.Izq())
          Rec_PostOrden(arbol.Der())
          RecNodo(arbol.Raíz())
     FINSI
FIN Rec_PostOrden

ALGORITMO Recorrer(E ArbolBin arbol, E TipoRecorrido rec)
INICIO
     CASO rec SEA
          inorden: Rec_InOrden(arbol)
          preorden: Rec_PreOrden(arbol)
          postorden: Rec_PostOrden(arbol)
     FINCASO
FIN Recorrer
```

Árboles binarios. Recorridos

Ejemplo:



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

Árboles binarios de búsqueda. Definición

La estructura lista enlazada es lineal ⇨ ¿Búsqueda?

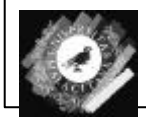
La estructura árbol ⟹ ¿Búsqueda más eficiente?

⇩

Siempre que los datos se distribuyan de forma adecuada

⇙

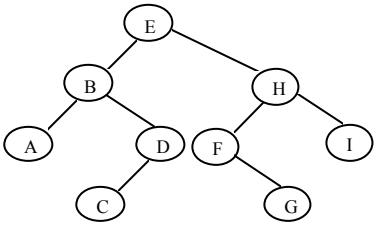
Árbol binario de búsqueda



Árboles binarios de búsqueda. Definición

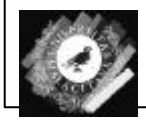
- Definición: árbol binario en el que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores.

Ejemplo:



Árboles binarios de búsqueda. Interfaz

```
INTERFAZ CLASE ABBusqueda
TIPOS
     // DEFINICIÓN DE TipoClave
     // DEFINICIÓN DE TipoElemento
MÉTODOS
     Crear()
     Destruir()
     Insertar(E TipoElemento dato)
     Suprimir(E TipoClave C)
     Buscar(E TipoClave C, S TipoElemento dato, S B EnArbol)
```

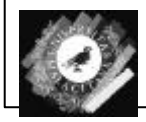



Árboles binarios de búsqueda. Utilización

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

- 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 teclado.
- 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.
  - Palabra.
  - Páginas ⟸ Caso de operación **InsertarP** o **AñadirP**

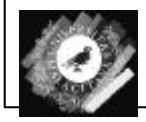


Árboles binarios de búsqueda. Utilización

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

Árboles binarios de búsqueda. Utilización

```
ALGORITMO ActualizarIndice(ES ABBusqueda indice)
VARIABLES
     TipoCódigo codigo
     TipoElemento elem
     N pag
     TipoClave palabra
     B encontrada
INICIO
     leerCódigo(codigo)
     MIENTRAS (codigo <> FinalizarP) HACER
          CASO MAY(codigo) SEA
               InsertarP:
                    leer(elem.clave)
                    elem.pag.total = 0
                    leer(pag)
```

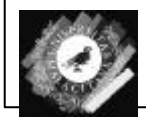



Árboles binarios de búsqueda. Utilización

```
                    MIENTRAS (pag <> 0) HACER
                          INC(elem.pag.total)
                          elem.pag.num[elem.pag.total] = pag
                          leer(pag)
                    FINMIENTRAS
                    indice.Insertar(elem)
               SuprimirP:
                    leer(palabra)
                     indice.Suprimir(palabra)
               AñadirP:
                    leer(palabra)
                indice.Buscar(palabra,elem,encontrada)
```

Árboles binarios de búsqueda. Utilización

```
                SI (NO encontrada) ENTONCES
                     escribir("palabra no está en indice")
                SINO
                     leer(pag)
                     MIENTRAS (pag <> 0) HACER
                         INC(elem.pag.total)
                         elem.pag.num[elem.pag.total] = pag
                         leer(pag)
                     FINMIENTRAS
                     indice.Suprimir(palabra)
                     indice.Insertar(elem)
                FINSI
          FINCASO
          leerCódigo(codigo)
     FINMIENTRAS
FIN
```

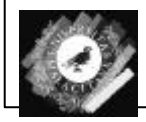
Árboles binarios de búsqueda. Implementación

```
IMPLEMENTACIÓN CLASE ABBusqueda
TIPOS
     REGISTRO NodoArbol
               NodoArbol *izq,der
               TipoElemento elem
               TipoClave    clave
     FINREGISTRO
ATRIBUTOS
     NodoArbol *nrz
MÉTODOS
     Crear()
     INICIO
          nrz = NULO
     FIN
```

Árboles binarios. Implementación

```
ALGORITMO LiberarMemoria(ES NodoArbol *a)
INICIO
     SI a <> NULO ENTONCES
          LiberarMemoria(a->izq)
          LiberarMemoria(a->der)
          Liberar (a)
     FINSI
FIN LiberarMemoria

Destruir()
INICIO
     LiberarMemoria(nrz)
FIN Destruir
```

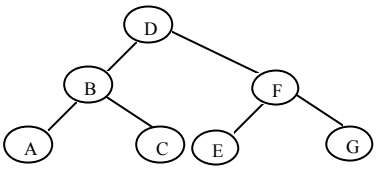



Árboles binarios de búsqueda. Implementación

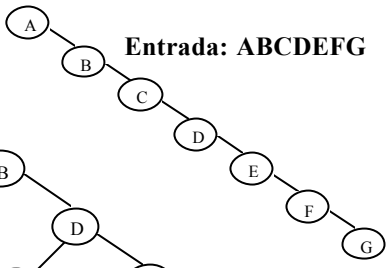
```
    Buscar(E TipoClave C, S TipoElemento dato, S B EnArbol)
    VARIABLES
         NodoArbol *aux = nrz
    INICIO
         EnArbol = FALSO
         MIENTRAS (aux<>NULO) Y (NO EnArbol)) HACER
                 SI (aux->clave == c) ENTONCES
                       EnArbol == VERDADERO
                       dato = arbol->elem
                 SINOSI (aux->clave > c) ENTONCES
                           aux = aux->izq
                       EN OTRO CASO
                           aux = aux->der
                       FINSI
                 FINSI
         FINMIENTRAS
    FIN Buscar
```

Árboles binarios de búsqueda. Implementación

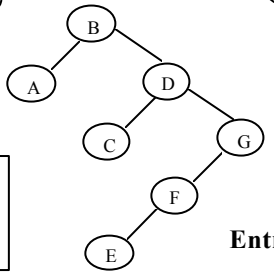
Consideraciones acerca de la operación de inserción.



Entrada: DBFACEG



Entrada: ABCDEFG



Entrada: BADCGFE

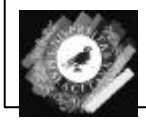
Los mismos datos, insertados en orden diferente, producirán árboles con formas o distribuciones de elementos muy distintas.

Árboles binarios de búsqueda. Implementación

```
Insertar(E TipoElemento dato)
VARIABLES
     NodoArbol *nuevonodo,*pav = nrz, *pret = NULO
     TipoClave clavenueva
INICIO
     Asignar(nuevonodo)
     nuevonodo->izq = NULO
     nuevonodo->der = NULO
     nuevonodo->elem = dato
     clavenueva = dato.clave
     MIENTRAS (pav <> NULO) HACER
          pret = pav
          SI (pav->elem.clave > clavenueva) ENTONCES
               pav = pav->izq
          SINO
               pav = pav->der
          FINSI
     FINMIENTRAS
```

Árboles binarios de búsqueda. Implementación

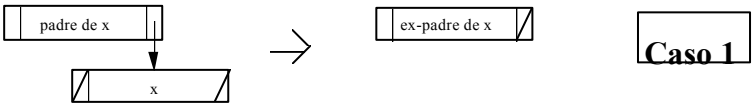
```
      SI (pret == NULO) ENTONCES
            arbol = nuevonodo
      EN OTRO CASO
          SI (pret->elem.clave > clavenueva) ENTONCES
               pret->izq = nuevonodo
          SINO
               pret->der = nuevonodo
          FINSI
      FINSI
FIN Insertar
```

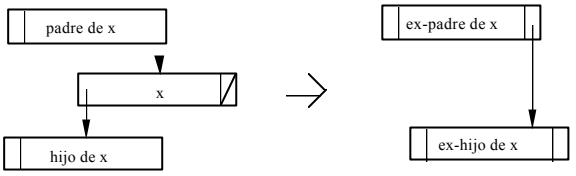
Árboles binarios de búsqueda. Implementación

Consideraciones acerca de la operación de suprimir.

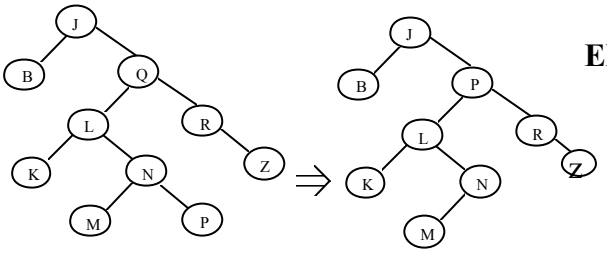
- Pasos:
  1) Encontrar el nodo a suprimir. ⇨ Equivalente a Buscar
  2) Eliminarlo del árbol. ⟹ 3 casos



Árboles binarios de búsqueda. Implementación



Caso 2



Eliminar Q

Caso 3

Árboles binarios de búsqueda. Implementación

```
ALGORIMTO SuprimirNodo(ES NodoArbol *na)
VARIABLES
         NodoArbol *temp,*ant
INICIO
     temp = na
     SI (na->der == NULO) ENTONCES
          na = na->izq
     SINO
          SI (na->izq == NULO) ENTONCES
               na = na->der
          SINO
               temp = na->izq
               ant  = na
               MIENTRAS (temp->der <> NULO HACER
                   anterior = temp
                   temp = temp->der
               FINMIENTRAS
```

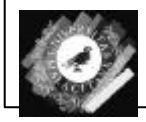
0 ó 1 hijo

1 hijo

2 hijos

Árboles binarios de búsqueda. Implementación

```
            na->elem = temp->elem
            SI (anterior == na) ENTONCES
                anterior->izq = temp->izq
            EN OTRO CASO
                anterior->der = temp->izq
            FINSI
        FINSI
    FINSI
    Liberar(temp)
FIN SuprimirNodo
```

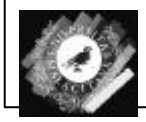



Árboles binarios de búsqueda. Implementación

```
    Suprimir(E TipoClave c)
    VARIABLES
        NodoArbol *pav, *pret
    INICIO
        pav = nrz
        pret = NULO
        MIENTRAS pav <> NULO Y pav->elem.clave <> c HACER
            pret = pav
            SI (pav->elem.clave > c) ENTONCES
                pav = pav->izq
            SINO
                pav = pav->der
            FINSI
        FINMIENTRAS
```

Árboles binarios de búsqueda. Implementación

```
        SI (pav <> NULO) ENTONCES
             SI (pav == nrz) ENTONCES
                 SuprimirNodo(nrz)
             SINO
                 SI (pret->izq == pav) ENTONCES
                       SuprimirNodo(pret->izq)
                 SINO
                       SuprimirNodo(pret->der)
                 FINSI
             FINSI
        FINSI
    FIN Suprimir
```

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.