Tema 5.
Tipos Abstractos de Datos

Diseño de Algoritmos.

E.U. Politécnica
Curso 2004-2005
Departamento Lenguajes y Ciencias de la Computación.
Universidad de Málaga
José Luis Leiva Olivencia.
Despacho: I-326D (Edificio E.U.P)/ 3.2.41 (Teatinos-E.T.S.I.I.)

Introducción

Objetivos

- Especificación e Implementación de nuevas estructuras de datos ⇨ Técnica: Abstracción de Datos
- Tipos de Estructuras de Datos:

1) Datos organizados por Posición ⇨ Pilas , Colas y Listas
2) Datos organizados por Valor ⇨ Arboles Binarios

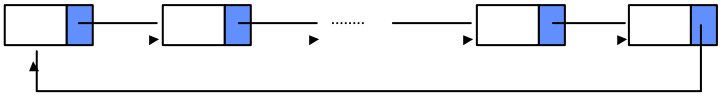
Introducción

- Estudio de las Estructuras de Datos:

- Definición de la Estructura de Datos e identificación de su Conjunto de Operaciones
- Desarrollo de diversas Implementaciones
- Presentación de Aplicaciones

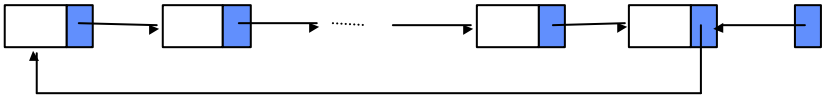
Listas enlazadas circulares

- Implementación estática o dinámica.
- El campo de enlace del último nodo apunta al primer nodo de la lista, en lugar de tener el valor **NULO.**
- No existe ni primer ni último nodo. Tenemos un anillo de elementos enlazados unos con otros.



Listas enlazadas circulares

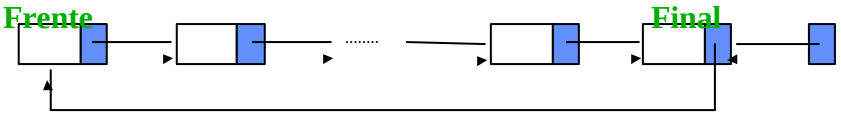
- Es conveniente, aunque no necesario, tener un enlace (puntero o índice) al último nodo lógico de la lista. Así podemos acceder facilmente a ambos extremos de la misma.



- Una lista circular vacía vendrá representada por un valor **NULO** (o `VALOR_NULO`).

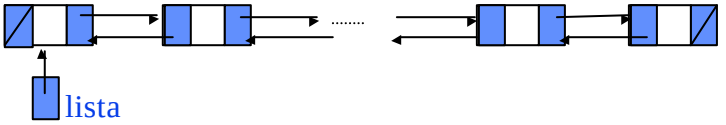
Listas enlazadas circulares

- 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.



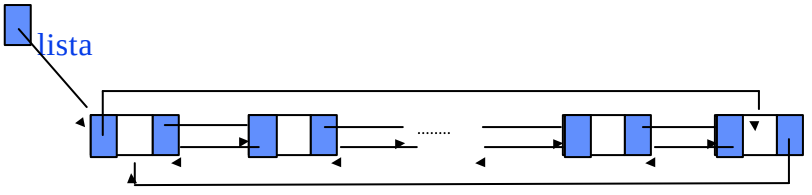
Listas doblemente enlazadas

- Es una lista enlazada en la que cada nodo tiene al menos tres campos:
  - 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.



Listas doblemente enlazadas

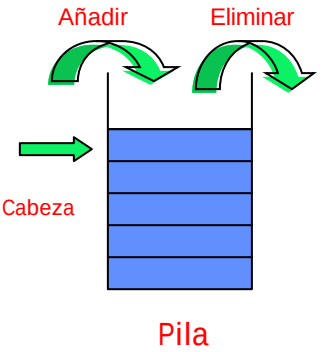
- Una lista doblemente enlazada puede modificarse para obtener una estructura circular de la misma



Pilas

Definición

- **Pila:** Grupo Ordenado, (de acuerdo al tiempo que llevan en la pila) de Elementos Homogéneos (todos del mismo tipo).
- **Acceso a la Pila:** añadir y eliminar elementos, SÓLO a través de la CABEZA de la Pila
- Estructura **LIFO** (**L**ast **I**nput **F**irst **O**utput)



Pilas. Operaciones

Conjunto de Operaciones

- **TipoPila Crear(void); /* vacía */**
- **/* Crea una pila vacía */**
- **int PilaVacia(TipoPila pila);**
- **/* Comprueba si la pila está vacía */**
- **int PilaLlena(TipoPila pila);**
- **/* Comprueba si la pila está llena */**

Pilas. Operaciones

Conjunto de Operaciones

- **void Sacar(TipoPila *pila, TipoElemento *elem);**
- **/* Saca un elemento. No comprueba antes si la pila está vacía */**
- **void Meter(TipoPila pila, TipoElemento elem);**
- **/* Mete un elemento en la pila. No comprueba si está llena. */**

Pilas. Implementación

Implementación

1) Con un Array

Array estructura adecuada ⇨ Elementos Homogéneos

Elementos almacenados de forma Secuencial

```
    const int MaxPila = 100; /* Dimension estimada */
/* TIPOS */
    typedef int TipoElemento; /* cualquiera */
    typedef struct{
        int Cabeza; /* Indice */
        TipoElemento elementos[100];
    }TipoPila;
```

Pilas. Implementación

Sólo es posible acceder a la Cabeza de la Pila

⇩

¿ Cómo es posible conocer la posición de la cabeza?

1) Variable entera "cabeza" ⇨Inconveniente: se ha de pasar como parámetro adicional a todas las operaciones sobre la pila
2) Extender el array, en pila[0] almacenaremos el índice del elemento que ocupa la cabeza actual

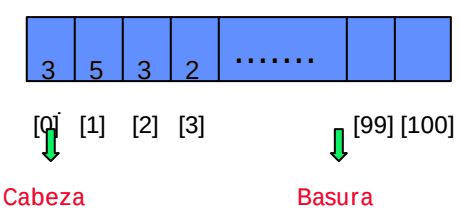
Pilas. Implementación

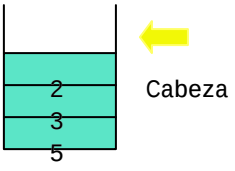
Constantes

```
Cabeza=0; MaxPila=100;
```

Tipos

```
TipoElemento TipoPila[MaxPila]
```

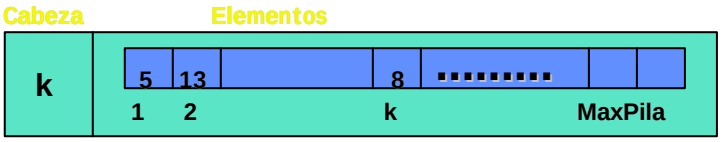




Pilas.Implementación

- Inconveniente: Solo es posible implementar una pila de enteros (no de cualquier otro tipo de datos)
- Solución:

```
Constantes
  MaxPila=100
Tipos
  TipoElemento=(*cualquier tipo de datos*)
  TipoPila=REGISTRO
            Cabeza:Z
              Elementos:ARRAY[1..MaxPila]DE
                        TipoElemento
```

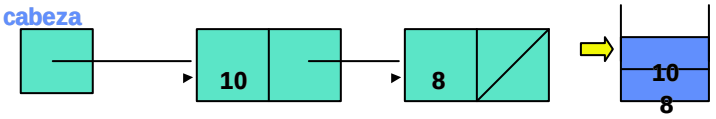


Pilas.Implementación

2) Con una Lista Enlazada de Punteros

- Comienzo de una lista enlazada ⇨ Cabeza de la Pila

```
/* TIPOS */
        typedef int TipoElemento; /*o cualquiera */
        typedef struct nodo {
                TipoElemento dato;
                struct nodo *sig;
        } TipoNodo;
        typedef TipoNodo *TipoPuntero;
        typedef TipoPuntero TipoPila;
```



Pilas. Aplicaciones

Aplicaciones

- Ejemplo1: Leer una secuencia de caracteres desde teclado e imprimirla al revés
- Ejemplo2: Verificar si una cadena de caracteres está balanceada en paréntesis o no

  abc(defg(ijk))(l(mn)op)qr ⇒ SI

  abc(def))ghij(kl)m ⇒ NO
- Ejemplo3: Reconocimiento del Lenguaje,

  L={W$W´ / W es una cadena de caracteres y W´es su inversa} (Suponemos que $ no está ni en W ni en W´)

Pilas. Aplicaciones

- Aplicaciones complejas que se pueden solucionar con pilas: **Expresiones Algebraicas**

  Operadores: +, -, *, /

  Operandos: Letras mayúsculas
- **Notación Infija:**
- El operador binario está situado entre sus dos operandos ⇒ A+ B
- Inconveniente: Son necesarias reglas de precedencia y uso de paréntesis para evitar ambigüedades ⇒ A+B*C

Pilas. Aplicaciones

Notación Prefija

- El operador binario esta situado justo antes de sus dos operandos ⇒ +AB
- Gramática:

<expr_pref>::=<letra>|<operador> <expr_pref><expr_pref>

<letra> ::= A| B ....|Z

<operador> ::= + | - | * | /

- Ejemplos:

A+(B*C) ⇒ +A*BC

(A+B)*C ⇒ *+ABC

Notación Postfija

- El operador binario está situado justo después de sus dos operandos ⇒ AB+
- Gramática:

<exp_post>::=<letra>|<expr_post> <exp_post><operador>

<letra> ::=A| B ....|Z

<operador> ::= + | - | * | /

- Ejemplos:

A+(B*C) ⇒ ABC*+

(A+B)*C ⇒ AB+C*

Pilas. Aplicaciones

- Ventaja: Usando expresiones prefijas y postfijas no son necesarias reglas de precedencia, ni uso de paréntesis.

  Las gramáticas que las generan son muy simples, y los algoritmos que las reconocen y evaluan muy fáciles

- Ejemplo 4: Algoritmo que evalúa una expresión en notación Postfija

  1)Usaremos una pila

  2)La expresión postfija se almacenará en un array de caracteres y será correcta

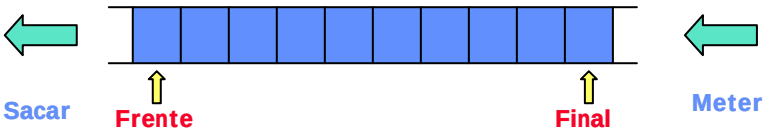
  3)Los operadores serán: +, -, * y /

  4)Los operandos serán letras mayúsculas (a cada una le podemos asignar un valor)

Colas

Definición

- Cola: es un grupo ordenado (con respecto al tiempo que llevan en él) de elementos homogéneos (todos del mismo Tipo)
- Acceso: los elementos se añaden por un extremo (final) y se sacan por el otro extremo (frente)
- Estructura **FIFO** (**F**irst **I**nput **F**irst **O**utput)



Colas. Operaciones

Conjunto de Operaciones

```
TipoCola  Crear();
/* Crea y devuelve una cola vacía */
int ColaVacía(Tipocola  Cola);
/* Devuelve 1 sólo si “cola” está vacía */
int ColaLlena(TipoCola Cola);
/* Devuelve 1 sólo si “cola” está llena */
int Sacar(Tipocola Cola,        TipoElemento *elem);
/* Saca el siguiente elemento del frente y lo pone en “elem” */
int Meter(TipoCola Cola, TipoElemento elem);
/* Mete “elem” al final de la cola */
```

Colas. Implementación

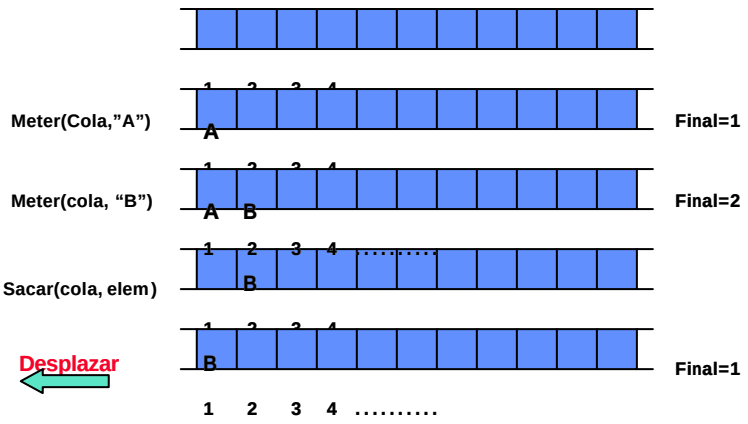
Implementación

1) Con un Array

- Se deja fijo el frente de la cola y se mueve el final a medida que se añaden nuevos elementos (Idem Pila)
- Las operaciones Meter, Crear, ColaVacia y ColaLlena se implementan de una forma similar a sus análogas en Pilas
- La operación de Sacar es mas complicada: cada vez que saquemos un elemento de la cola se han de desplazar el resto una posición en el array, para hacer coincidir el frente con la primera posición del array
- Ventaja ⇨ Simplicidad
- Inconveniente ⇨ Ineficiente (colas con muchos elementos o elementos grandes)

Colas. Implementación

- Ejemplo:



Diseño de Algoritmos

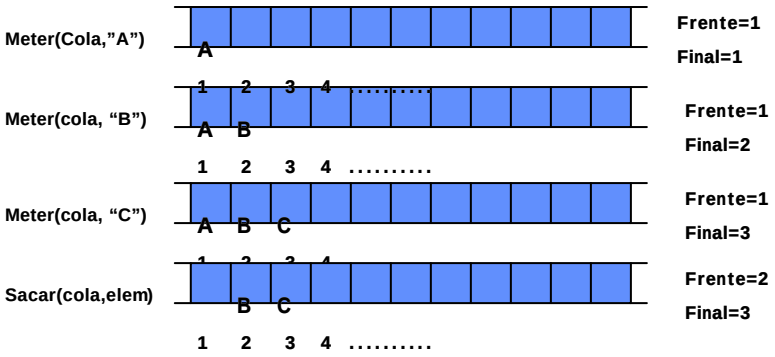
Colas. Implementación

Solución:

- Utilizar un indice para el frente y otro para el final y permitir que ambos fluctuen por el array
- Ventaja: operación Sacar más sencilla
- Inconveniente: Es posible que final sea igual a Maxcola (última casilla del array) y que la cola no esté llena

Colas. Implementación

- Ejemplo:

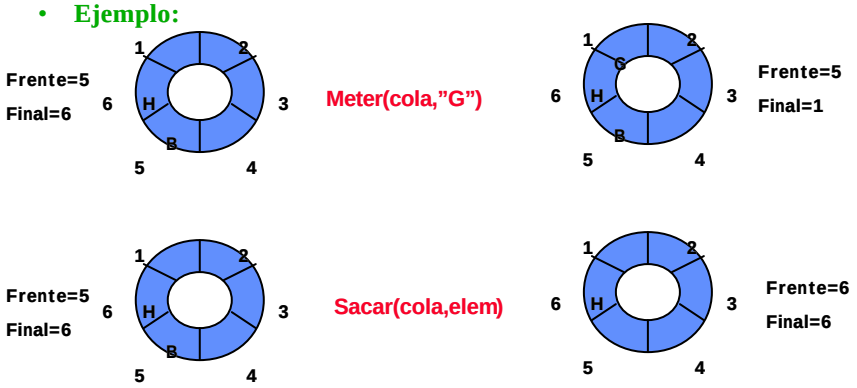


Colas. Implementación

- Solución:
- Tratar al array como una **Estructura Circular**, donde la última posición va seguida de la primera ⇨ Evitamos que el final de la cola alcance el final físico del array y no esté llena
- Operación Meter ⇨ Añade elementos a las posiciones del array e incrementa el índice final
- Operación Sacar ⇨ Más sencilla. Sólo se incrementa el índice frente a la siguiente posición

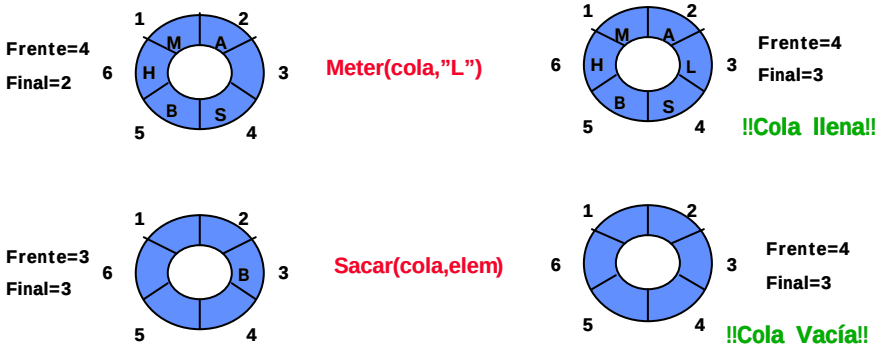
Colas. Implementación

- **Ejemplo:**



Colas. Implementación

¿Como sabemos si la cola está vacía o llena?



Colas. Implementación

- Solución:
- 1) Disponer de otra variable ⇨ Contabilizará los elementos almacenados en la cola

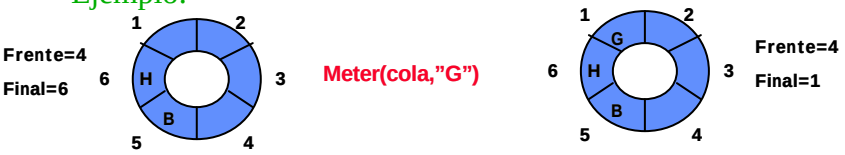
  Variable=0 ⇨ Cola vacía

  Variable=MaxCola ⇨ Cola llena

  Inconveniente: añade más procesamiento a las operaciones Meter y Sacar
- 2) Frente apunte a la casilla del array que precede a la del elemento frente de la cola ⇨ Solución elegida

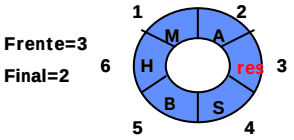
Colas. Implementación

- Ejemplo:



¿Como saber si la cola está llena? Es necesario que la posición a la que apunta frente en cada momento este Reservada

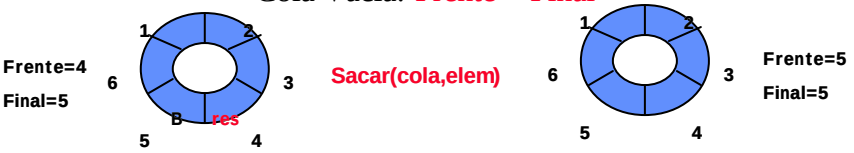
Cola Llena: final+1 =frente



Colas. Implementación

¿Como saber si la cola está vacía?

Cola Vacía: Frente = Final

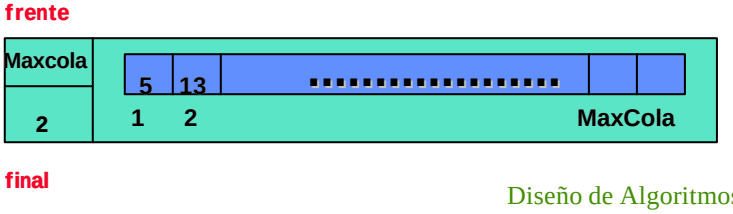


- Crear la cola (inicializarla vacía): frente =Maxcola (índice del array que precede al elemento frente de la cola) y final=Maxcola ⇨ Cola Vacía correcto (frente = final)

Colas. Implementación

- Agrupamos en un registro los índices frente y final, junto con el array que contendrá los elementos de la cola

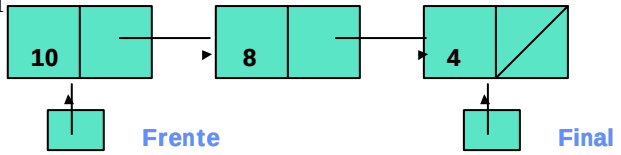
```
typedef int TipoElemento /* U otro cualquiera */
typedef struct{
        TipoElemento elementos[MaxCola];
        unsigned frente; /* Frente: [0..MaxCola-1] */
        int final; /* Final: [0..MaxCola-1] */
}TipoCola;
```



Colas. Implementación

2) Con listas enlazadas con Punteros

- Usamos dos variables de tipo puntero, frente y final, que apunten a los nodos que contienen los elementos frente y final

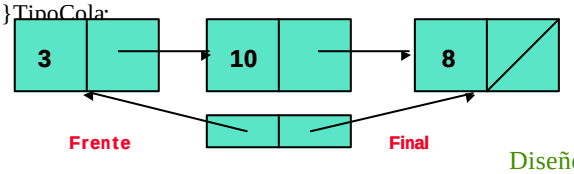


- ¿Que sucedería si intercambiáramos las posiciones de frente y final en la lista enlazada?

Colas. Implementación

- Agrupamos las variables frente y final en un registro

```
typedef int TipoElemento; /* O cualquiera */
typedef struct nodo{
                TipoElemento valor;
                struct nodo *sig;
    }TipoNodo;
typedef TipoNodo *TipoPuntero;
typedef struct{
                TipoPuntero frente;
                TipoPuntero final;
    }TipoCola;
```

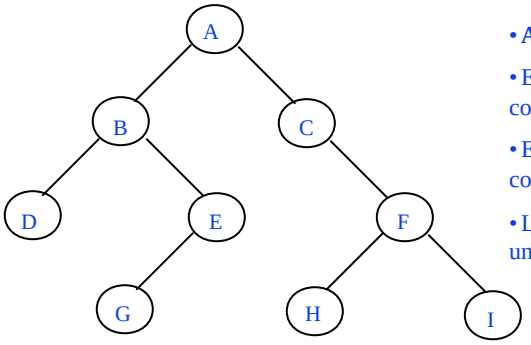


Árboles binarios

- **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

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



- Consta de 9 nodos.
- **A** es el nodo raíz.
- El subárbol izquierdo tiene como nodo raíz **B**.
- El subárbol derecho tiene **C** como raíz.
- 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.
- Dos nodos son **hermanos** si son los hijos izquierdo y derecho del mismo padre.

Árboles binarios

- **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**: longitud del camino que une dicho nodo con la raíz. 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 raíz hasta una hoja.
- **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 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 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

Operaciones sobre árboles:

Crear árbol vacío

Crear árbol dada su raíz,

y sus hijos Derecho

e Izquierdo

¿Está vacío?

¿Está lleno?

Devolver el contenido del nodo raíz

Devolver el subárbol derecho

Devolver el subárbol izquierdo

Destruir

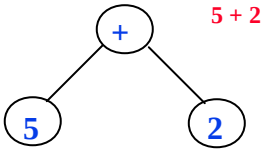
Árboles binarios

NIVEL DE UTILIZACIÓN

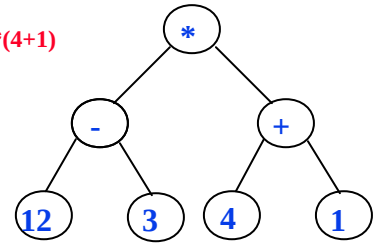
- Estructura de datos muy útil cuando se deben tomar decisiones de "dos caminos"
- Muy utilizado en aplicaciones relacionadas con expresiones aritméticas.

Ejemplos:

5 + 2



(12-3)*(4+1)



Árboles binarios de búsqueda

La estructura lista enlazada es lineal → ¿Búsqueda?

La estructura árbol → ¿Búsqueda más eficiente?

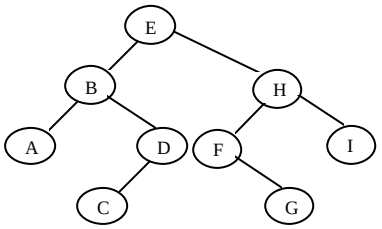
Siempre que los datos se distribuyan de forma adecuada

Árbol binario de búsqueda

Árboles binarios de búsqueda

- Definición: 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

Operaciones:

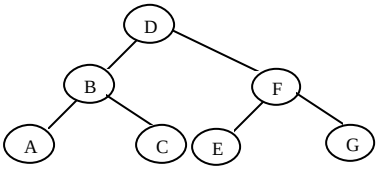
Crear
Buscar
Insertar
Suprimir
Imprimir

\+

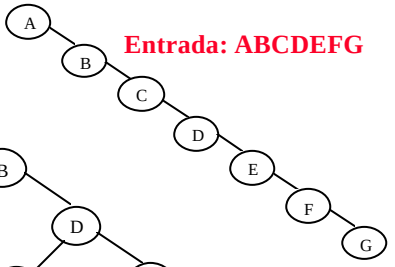
Las definidas para un árbol binario general

Árboles binarios de búsqueda

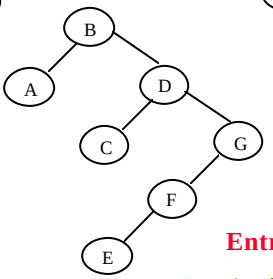
Consideraciones acerca de la operación de inserción.



Entrada: DBFACEG



Entrada: ABCDEFG



Entrada: BADCGFE

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

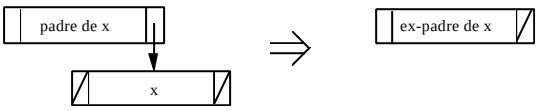
Árboles binarios de búsqueda

Consideraciones acerca de la operación de suprimir.

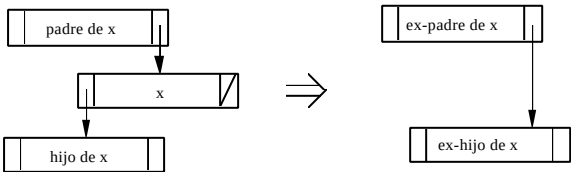
- Pasos:

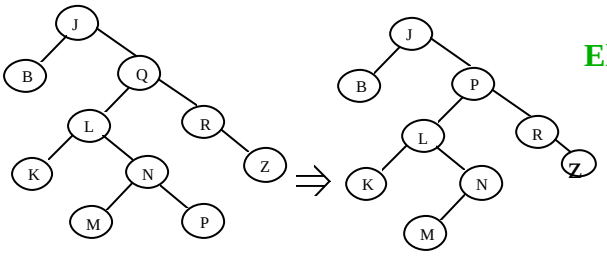
  1) Encontrar el nodo a suprimir. ⇒ Equivalente a Buscar

  2) Eliminarlo del árbol. ⇒ 3 casos



Árboles binarios de búsqueda





Eliminar Q

Árboles binarios de búsqueda

<u>Consideraciones acerca de la operación de imprimir</u>.

- 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 orden?

Árboles binarios de búsqueda

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

a) Recorrido **InOrden**.

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

b) Recorrido **PreOrden**.

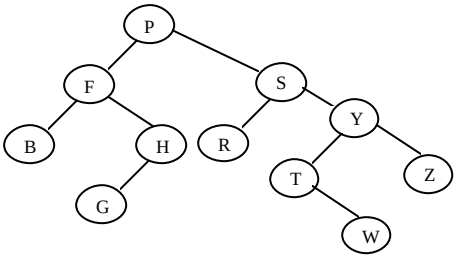
1) "Visitar" el valor del nodo raíz y procesarlo
2) Recorrer el subárbol izquierdo en PreOrden
3) Recorrer el subárbol derecho en PreOrden

c) Recorrido **PostOrden**.

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

Árboles binarios de búsqueda

Ejemplo:



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