



Programación 3

Cursada 2013

Prof. Catalina Mostaccio
Prof. Alejandra Schiavoni

Ingeniería en Computación
Facultad de Informática - UNLP
2013

1

Listas



Listas

- *Una lista es una estructura de datos en donde los objetos están ubicados en forma secuencial.*
- *Puede estar implementada a través de:*
 - *un arreglo*
 - *una estructura dinámica (punteros)*



Listas

- *La lista puede estar ordenada o no*
 - *Si está ordenada, los elementos se ubican siguiendo el orden de las claves almacenadas en la lista*
 - *Si está desordenada, los elementos pueden aparecer en cualquier orden*

Listas

➤ *La lista implementada con una estructura dinámica puede ser:*

- *Simplemente enlazada: cada elemento de la lista tiene un puntero al siguiente*
- *Doblemente enlazada: cada elemento de la lista tiene dos punteros, uno al elemento anterior y otro al siguiente*

Listas: operaciones

- `comenzar()` // Se prepara para iterar los elementos de la lista.
- `proximo()` // Avanza al próximo elemento de la lista.
- `fin()` // Determina si llegó o no al final de la lista.
- `elemento()` // Retorna el elemento actual.
- `elemento(Integer pos)` // Retorna el elemento de la posición indicada (la posición va desde 0 hasta n-1).
- `agregar(Integer elem)` // Agrega el elemento en la posición actual y retorna true si pudo agregar y false si no pudo agregar.
- `agregar(Integer elem, Integer pos)` // Agrega el elemento en la posición indicada y retorna true si pudo agregar y false; si no pudo agregar.

Listas: operaciones (cont.)

- `eliminar()` // Elimina el elemento actual y retorna true si pudo eliminar y false en caso contrario.
- `eliminar(Integer pos)` // Elimina el elemento de la posición indicada y retorna true si pudo eliminar y false en caso contrario.
- `esVacia()` // Retorna true si la lista está vacía, false en caso contrario.
- `incluye(Integer elem)` // Retorna true si elem está contenido en la lista, false en caso contrario.
- `tamano()` // Retorna la longitud de la lista.

Listas

➤ *Los métodos para representar las listas a través de estructuras dinámicas se pueden extender para otras estructuras de datos homogéneas, por ejemplo los árboles.*



Árboles Binarios



Agenda

- ❖ Definición
- ❖ Descripción y terminología
- ❖ Representaciones
- ❖ Recorridos
- ❖ Tiempo de ejecución de los recorridos
- ❖ Aplicación: Árboles de expresión

Árbol Binario: Definición

- *Un árbol binario es una colección de nodos, tal que:*
 - *puede estar vacía*
 - *puede estar formada por un nodo distinguido R , llamado **raíz** y dos sub-árboles T_1 y T_2 , donde la raíz de cada subárbol T_i está conectado a R por medio de una arista*

Descripción y terminología

- Cada nodo puede tener a lo sumo dos nodos hijos.
- Cuando un nodo no tiene ningún hijo se denomina **hoja**.
- Los nodos que tienen el mismo nodo padre se denominan **hermanos**.

Descripción y terminología

➤ Conceptos a usar:

- **Camino**: desde n_1 hasta n_k , es una secuencia de nodos n_1, n_2, \dots, n_k tal que n_i es el padre de n_{i+1} , para $1 \leq i < k$. La longitud del camino es el número de aristas, es decir $k-1$. Existe un camino de longitud cero desde cada nodo a sí mismo. Existe un único camino desde la raíz a cada nodo.
- **Profundidad**: de n_i es la longitud del único camino desde la raíz hasta n_i . La raíz tiene profundidad cero.

Descripción y terminología

- **Grado** de n_i es el número de hijos del nodo n_i .
- **Altura** de n_i es la longitud del camino más largo desde n_i hasta una hoja. Las hojas tienen altura cero. La altura de un árbol es la altura del nodo raíz.
- **Ancestro/Descendiente**: si existe un camino desde n_1 a n_2 , se dice que n_1 es ancestro de n_2 y n_2 es descendiente de n_1 .

Descripción y terminología

- **Árbol binario lleno**: Dado un árbol binario T y altura h , diremos que T es *lleno* si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel (h).

Es decir, recursivamente, T es *lleno* si :

- 1.- T es un nodo simple (árbol binario lleno de altura 0), o
- 2.- T es de altura h y sus sub-árboles son llenos de altura $h-1$.

Descripción y terminología

- **Árbol binario completo**: Dado un árbol binario T y altura h , diremos que T es *completo* si es *lleno* de altura $h-1$ y el nivel h se completa de izquierda a derecha.
- **Cantidad de nodos en un árbol binario lleno**:
Sea T un árbol binario lleno de altura h , la cantidad de nodos N es $(2^{h+1} - 1)$
- **Cantidad de nodos en un árbol binario completo**:
Sea T un árbol binario completo de altura h , la cantidad de nodos N varía entre (2^h) y $(2^{h+1} - 1)$

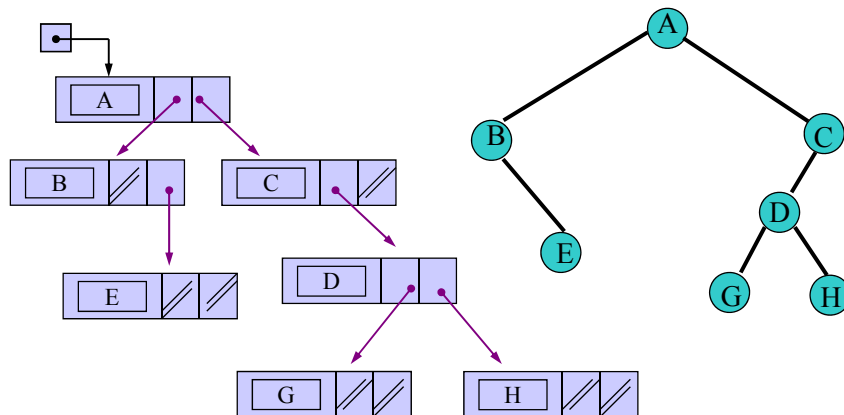
Representación

Hijo Izquierdo - Hijo Derecho

- ✓ Cada nodo tiene:
 - Información propia del nodo
 - Referencia a su hijo izquierdo
 - Referencia a su hijo derecho
- ✓ Puede implementarse a través de:
 - Arreglos
 - Punteros

Representación

Hijo Izquierdo - Hijo Derecho



Recorridos

- **Preorden**
 - Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.
- **Inorden**
 - Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho
- **Postorden**
 - Se procesan primero los hijos, izquierdo y derecho, y luego la raíz
- **Por niveles**
 - Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

Recorrido: Preorden

```
public void preorden() {  
    imprimir (this.getRaiz().getDato());  
    if(!this.getHijoIzquierdo().esVacio()){  
        this.getHijoIzquierdo.preorden();  
    }  
    if(!this.getHijoDerecho().esVacio()){  
        this.getHijoDerecho.preorden();  
    }  
}
```

Recorrido Preorden: Tiempo de Ejecución

Considerando un árbol binario **lleno** y altura h

$$T(n) = \begin{cases} c & n = 1 \\ c + 2 T((n-1)/2) & n > 1 \end{cases}$$

$T(n)$ es $O(n)$

Recorrido Preorden: Tiempo de Ejecución

Otra forma: expresando en función de la altura

$$T(h) = \begin{cases} c & h = 0 \\ c + 2 T(h-1) & h > 0 \end{cases}$$

$T(n)$ es $O(n)$

Recorrido: Por niveles

```
public void porNiveles() {
    Cola q = new Cola();
    ArbolBinario arbol;
    q.encolar(this);
    while (!q.esVacia()) {
        arbol = q.desencolar();
        imprimir arbol.getRaiz().getDato();
        if (!arbol.getHijoIzquierdo().esVacio()){
            q.encolar(arbol.getHijoIzquierdo());
        }
        if (!arbol.getHijoDerecho().esVacio()){
            q.encolar(arbol.getHijoDerecho());
        }
    }
}
```

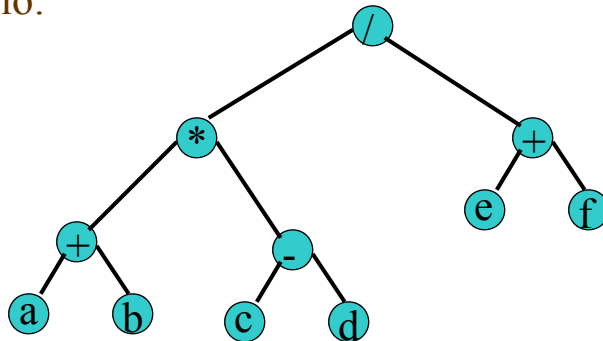
Árbol de Expresión

Es un árbol binario asociado a una expresión aritmética

- Nodos internos representan operadores
- Nodos externos (hojas) representan operandos

Árbol de Expresión

Ejemplo:



Árbol de Expresión

Recorriendo el árbol, obtenemos:

Inorden: $((a + b) * (c - d)) / (e + f)$

Preorden: $/*+ab-cd+ef$

Postorden: $ab+cd-*ef+ /$

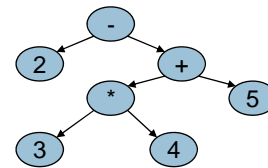
Construcción de un árbol de expresión

A partir de una:

✓ Expresión postfija

✓ Expresión prefija

✓ Expresión infija



Construcción de un árbol de expresión a partir de una expresión *postfija*

Algoritmo:

tomar un carácter de la expresión

mientras (existe carácter) hacer

si es un operando → creo un nodo y lo apilo.

si es un operador (lo tomo como la raíz de los dos últimos nodos creados)

→ - creo un nodo R,

- desapilo y lo pongo como hijo derecho de R

- desapilo y lo pongo como hijo izquierdo de R

- apilo R.

tomo otro carácter

fin

Construcción de un árbol de expresión a partir de una expresión *prefija*

Algoritmo:

ArbolExpresión (A: *ArbolBin*, exp: string)

si exp nulo \rightarrow nada.

si es un operador \rightarrow - creo un nodo raíz R

- *ArbolExpresión* (subArbIzq de R, exp
(sin 1º carácter))

- *ArbolExpresión* (subArbDer de R, exp
(sin 1º carácter))

si es un operando \rightarrow creo un nodo (hoja)

Construcción de un árbol de expresión a partir de una expresión *infija*

Expresión infija

Se usa una pila y se tiene en cuenta la precedencia de los operadores

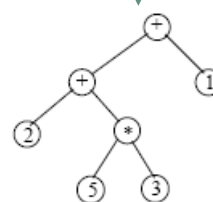
Expresión postfija

Se usa la estrategia anterior

Árbol de Expresión

2+5*3+1

253*+1+



-Convertir una **exp. infija** en árbol de expresión : se debe convertir la exp. infija en postfija y a partir de ésta, construir el árbol de expresión.

Estrategia del Algoritmo para convertir exp. infija en postfija :

- a) si es un operando → se coloca en la salida.
- b) si es un operador → se maneja una pila según la prioridad de este operador en relación al operador de la pila

operador con \geq prioridad – se apila

operador con $<$ prioridad – se desapila elemento

colocándolo en la salida, hasta encontrar uno de menor prioridad (si se encuentra de = prioridad también se desapila), luego se apila el operador.

- c) cuando se llega al final de la expresión, se desapilan todos los elementos llevándolos a la salida, hasta que la pila quede vacía.

- Consideraciones tenidas en cuenta en el Algoritmo:

Prioridad de los operadores de menor a mayor:

$+$ y $-$, $*$ y $/$

Los “ (“ siempre se apilan como si tuvieran la mayor prioridad y se desapilan **sólo** cuando aparece un “) ”.

Algoritmo:

tomar un carácter

mientras (existe carácter) hacer

 si es un operando → se coloca en la salida.

 si es un operador →

 - "(" se apila operador

 - ")" se desapila elemento y se lleva a la salida hasta encontrar "(", que se desapila pero no va a la salida.

 - "+", "*", "-", "/"

 Si el operador es de mayor prioridad que el tope se apila

 si no

 se desapila elemento y se lleva a la salida hasta encontrar operador con menor y luego se apila el operador. (según b) de la Estrategia)

 tomo otro carácter

fin

se desapilan los elementos y se llevan a la salida hasta que se vacíe la pila.

Ejercitación

Árbol binario de expresión

Ejercicio 1.

✓ Dada la siguiente expresión postfija : $IJK++AB* C - *$, dibuje su correspondiente árbol binario de expresión

✓ Convierta la expresión $((a + b) + c * (d + e) + f) * (g + h)$ en expresión prefija

Ejercicio 2.

✓ Dada la siguiente expresión prefija : $*+I+JK-C*AB$, dibuje su correspondiente árbol binario de expresión

✓ Convierta la expresión $((a + b) + c * (d + e) + f) * (g + h)$ en expresión postfija



Árboles Generales



Agenda

- ❖ Definición
- ❖ Descripción y terminología
- ❖ Ejemplos
- ❖ Representaciones
- ❖ Recorridos
- ❖ Tiempo de ejecución de los recorridos

Definición

➤ *Un árbol es una colección de nodos, tal que:*

- *puede estar vacía. (Árbol vacío)*
- *puede estar formada por un nodo distinguido R , llamado **raíz**, y un conjunto de árboles T_1, T_2, \dots, T_k , $k \geq 0$ (subárboles), donde la raíz de cada subárbol T_i está conectado a R por medio de una arista*

Descripción y terminología

- **Grado** del árbol es el grado del nodo con mayor grado.
- **Árbol lleno**: Dado un árbol T de grado k y altura h , diremos que T es *lleno* si cada nodo interno tiene grado k y todas las hojas están en el mismo nivel (h).

Es decir, recursivamente, T es *lleno* si :

- 1.- T es un nodo simple (árbol lleno de altura 0), o
- 2.- T es de altura h y todos sus sub-árboles son llenos de altura $h-1$.

Descripción y terminología

- **Árbol completo:** Dado un árbol T de grado k y altura h , diremos que T es *completo* si es lleno de altura $h-1$ y el nivel h se completa de izquierda a derecha.
- **Cantidad de nodos en un árbol lleno:**
Sea T un árbol lleno de grado k y altura h , la cantidad de nodos N es $(k^{h+1} - 1) / (k - 1)$

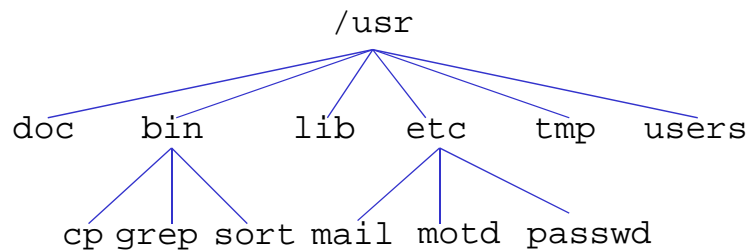
Descripción y terminología

- **Cantidad de nodos en un árbol completo:**
Sea T un árbol completo de grado k y altura h , la cantidad de nodos N varía entre $(k^h + k - 2) / (k - 1)$ y $(k^{h+1} - 1) / (k - 1)$

Ejemplos

- ✓ Organigrama de una empresa
- ✓ Árboles genealógicos
- ✓ Taxonomía que clasifica organismos
- ✓ Organización de un libro en capítulos y secciones
- ✓ Sistemas de archivos

Ejemplo: Sistema de archivos



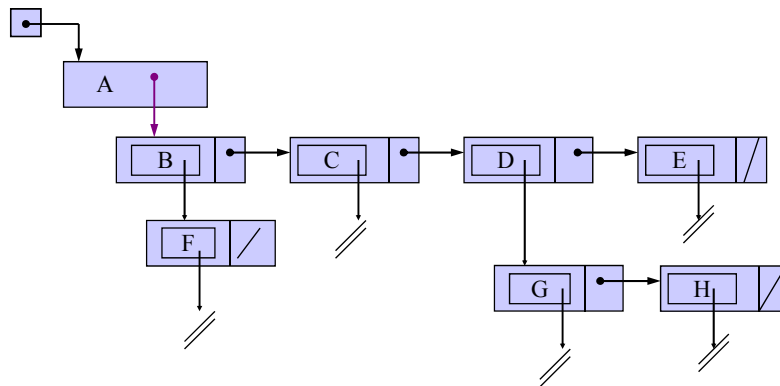
Representaciones

- ✓ Lista de hijos
 - Cada nodo tiene:
 - Información propia del nodo
 - Una lista de todos sus hijos
- ✓ Hijo más izquierdo y hermano derecho
 - Cada nodo tiene:
 - Información propia del nodo
 - Referencia al hijo más izquierdo
 - Referencia al hermano derecho

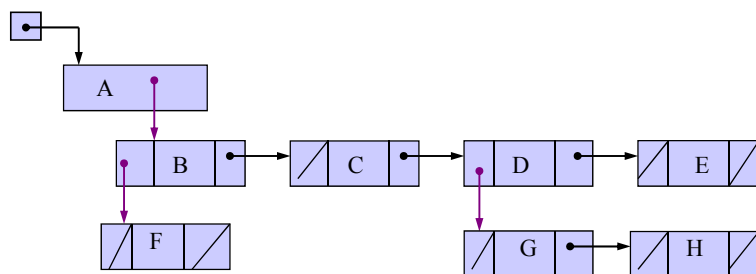
Representación: Lista de hijos

- ✓ La lista de hijos, puede estar implementada a través de:
 - Arreglos
 - Desventaja: espacio ocupado
 - Listas dinámicas
 - Mayor flexibilidad en el uso

Representación: Lista de hijos



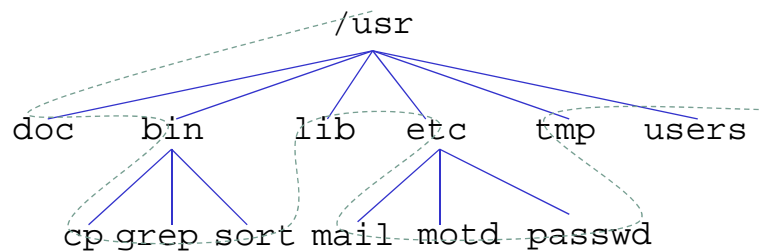
Representación: Hijo más izquierdo y hermano derecho



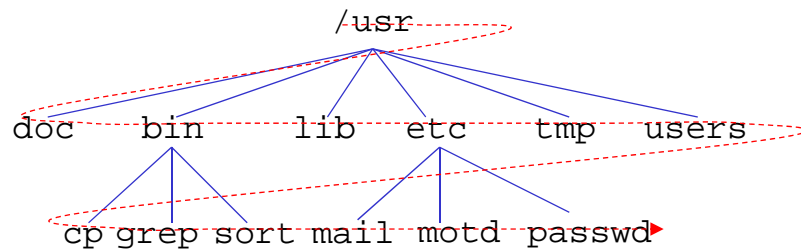
Recorridos

- **Preorden**
 - Se procesa primero la raíz y luego los hijos
- **Inorden**
 - Se procesa el primer hijo, luego la raíz y por último los restantes hijos
- **Postorden**
 - Se procesan primero los hijos y luego la raíz
- **Por niveles**
 - Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

Recorrido: Preorden



Recorrido: Por niveles



Recorrido: Preorden

```
public void preorden() {
    imprimir (this.getRaiz().getDato());
    Lista hijos;
    hijos = this.getHijos();
    hijos.comenzar()
        while (!hijos.fin()) {
            hijos.actual().preOrden();
            hijos.siguiete();
        }
}
```

Recorrido Preorden: Tiempo de Ejecución

Considerando un árbol lleno de grado k y altura h

$$T(n) = \begin{cases} c & n = 1 \\ c + k T((n-1)/k) & n > 1 \end{cases}$$

$T(n)$ es $O(n)$

Recorrido Preorden: Tiempo de Ejecución

Otra forma: expresando en función de la altura

$$T(h) = \begin{cases} c & h = 0 \\ c + k T(h-1) & h > 0 \end{cases}$$

$T(n)$ es $O(n)$

Recorrido: Por niveles

```
porNiveles() {  
    Lista hijos;  
    ArbolGeneral arbol;  
    Cola cola = new Cola();  
    cola.encolar(this);  
    while (!cola.esVacia()) {  
        arbol = cola.desencolar();  
        imprimir (this.getRaiz().getDato());  
        hijos = this.getListaHijos();  
        hijos.comenzar();  
        while (!hijos.fin()) {  
            cola.encolar(hijos.actual());  
            hijos.siguiente();  
        }  
    }  
}
```