


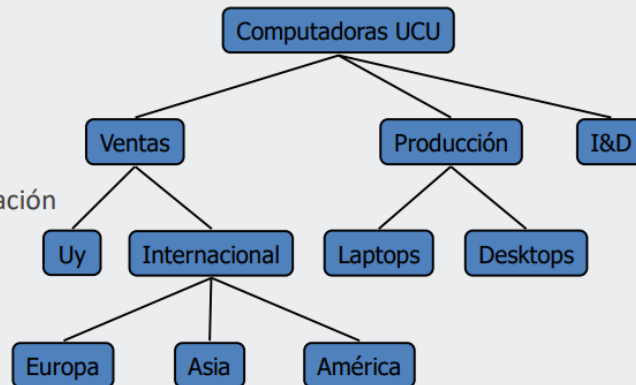
UT5 – Árboles genéricos y tries

```
def rec(node, s, i):
    if i == len(s):
        node.is_end = False
        return len(node.children) == 0
    else:
        next_deletion = rec(node.children[s[i]], s, i+1)
        if next_deletion:
            del node.children[s[i]]
    return next_deletion
        and not node.is_end
        and len(node.children) == 0
```



¿qué es un Arbol?

- En ciencias de la computación, un árbol es un modelo abstracto de una estructura jerárquica
- Un árbol consiste de nodos con una relación padre-hijo
- Aplicaciones:
 - Organigramas
 - Sistemas de archivos
 - Entornos de programación



Recorridos de Arboles genéricos

- Preorden : Raíz de A, seguido de los nodos de A1 en preorden, luego los de A2 en preorden, etc.
- Inorden : Nodos de A1 en inorden, luego la raíz, luego los nodos de los restantes subárboles en inorden.
- Postorden: Nodos de A1 en postorden, luego los de A2 en postorden, hasta el final, y luego la raíz.

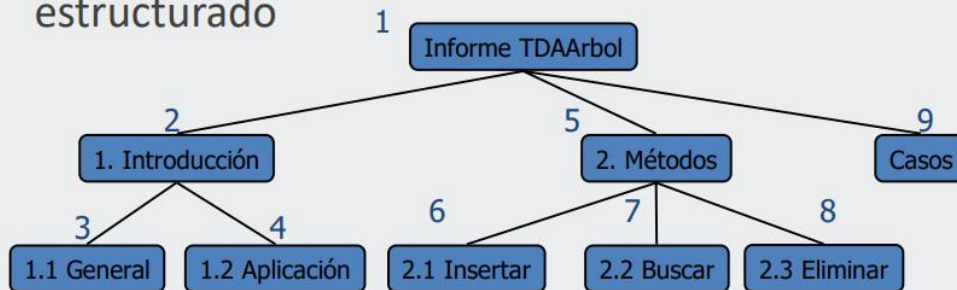
Recorrida en Preorden

- En un recorrido en preorden, un nodo es visitado antes que sus descendientes
- Aplicación: imprimir un documento estructurado

Algoritmo *preOrden*(*v*)

visitar(*v*)

Para cada hijo *w* de *v*
preorden (*w*)



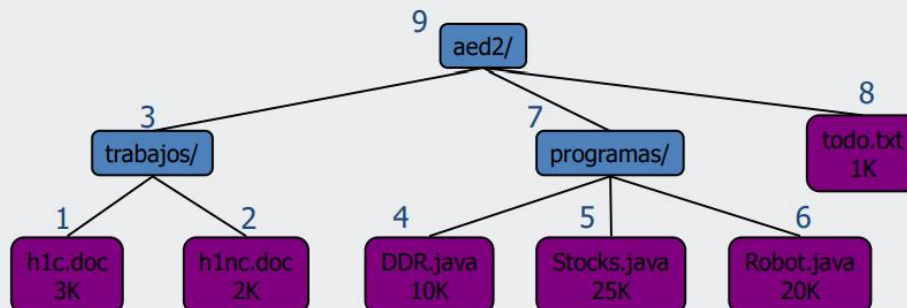
Recorrida en Postorden

- En un recorrido en postorden, un nodo es visitado después de sus descendientes
- Aplicación: calcular el espacio usado por archivos en un directorio y sus subdirectorios

Algoritmo *postOrden*(*v*)

Para cada hijo *w* de *v*
postOrden (*w*)

visitar(*v*)



Recorridas de árboles genéricos - preorden



TArbolGenerico.preOrden;

COM

SI Raiz <> nulo

Raiz.preOrden;

FINSI

FIN

TNodoArbolGenerico.preOrden;

COM

Imprimir(etiqueta);

unHijo \leftarrow PrimerHijo;

MIENTRAS unHijo <> nulo hacer

unHijo.preOrden;

unHijo \leftarrow unHijo.HermanoDerecho;

FIN MIENTRAS

FIN

El TDA ARBOL. Operaciones a ser consideradas



- Padre(unNodo).
- Hijolizquierdo(unNodo).
- HermanoDerecho(unNodo).
- Etiqueta(unNodo).
- Raiz.
- Anula.
- Otros??. (p.ej. Altura)

Tries



- Sea S un conjunto de s strings del alfabeto σ tal que ninguna es prefijo de otra (en el conjunto).
- Un **trie estándar** para S es un árbol **ordenado** T que cumple:
 - Cada nodo de T , excepto la raíz, tiene por etiqueta un caracter de σ .
 - El orden de los hijos de un nodo interno de T está determinado por el orden canónico del alfabeto σ .
 - T tiene s nodos externos, cada uno asociado con una string de S , de forma tal que la concatenación de las etiquetas de los nodos en el camino desde la raíz hasta un nodo externo v de T produce la string asociada con v .

Tries

- **Operaciones primarias:**
 - **Comparación de patrones** (strings o palabras completas)
 - **Comparación de prefijos** (dada una string X de entrada, encontrar todas las strings S que contienen a X como prefijo).
- **Importante: ninguna string en S es prefijo de otra string.**
 - Esto asegura que cada string de S está asociada en forma única con un **nodo externo** de T .
 - **Siempre se puede satisfacer esta condición, agregando un carácter especial al final de cada string. (ej: “*”)**
- Un nodo interno puede tener entre **1** y **d** hijos, donde **d** es el tamaño del alfabeto (ej 26 letras).

Trie estándar

- El trie estándar para el conjunto de strings S es un árbol **ordenado** tal que:
 - Cada nodo está etiquetado con un carácter
 - Los hijos de un nodo están ordenados alfabéticamente
 - Los caminos desde la raíz hasta los nodos externos nos dan las strings del conjunto S

Propiedades estructurales del trie



- Un **trie estándar** para almacenar una colección S de s strings de largo total n en base a un alfabeto de tamaño d tiene las siguientes propiedades:
 - Todo nodo interno de T tiene como máximo d hijos
 - T tiene s nodos externos (tantos como strings hay en S)
 - Altura de T : longitud de la string más larga de S
 - El número de nodos de T es $O(n)??$
- Puede ser usado para implementar un diccionario
 - Se comparan caracteres individuales en vez de strings completas
- Falta de eficiencia en la representación
 - **Trie comprimido** – trie “**Patricia**”

Búsqueda en Tries



Dos operaciones principales:

- Búsqueda de palabras completas
 - se desea determinar si un patrón dado está en una de las palabras del texto, exactamente.
 - Ejemplo: índice, al encontrar la palabra queremos indicar las páginas del libro en que se encuentra
- Búsqueda de prefijos
 - Dado un patrón (una string), determinar si es prefijo de palabras existentes en el trie
 - Devolver todas las palabras de las cuales es prefijo (ejemplo “autocompletar” incremental...)
 - Indicar la cantidad de palabras de las cuales es prefijo

Búsqueda en trie – palabra completa



- La búsqueda de palabras para un patrón de largo m toma un tiempo $O(d*m)$
- Alfabeto es de tamaño constante, (ej.: lenguajes naturales o cadenas ADN), una consulta toma un tiempo $O(m)$, proporcional al **tamaño del patrón (independiente del tamaño del texto!)**

30

Inserción en tries



- **Objetivo:** Insertar las strings una a la vez.
- **Condición:** Ninguna string de S es prefijo de otra
 - Siempre se puede satisfacer esta condición, agregando un carácter especial al final de cada string. (ej: “*”)
- **Insertar una string X :** primero tratamos de recorrer el camino asociado con X en T .
- X no está en T : pararemos en un nodo interno v .
- Crear nueva cadena de nodos descendientes de v para almacenar los restantes caracteres de X .
- **Tiempo para insertar X :** $O(d*m)$, m es el largo de X y d es el tamaño del alfabeto
- **Construcción del trie entero:** $O(d*n)$, n largo total de todas las strings de S

21

Inserción en tries



En la clase **NodoTrie insertar** (string unaPalabra) // eventualmente boolean u otro indicador?

Comienzo

nodoActual \leftarrow this

Para cada caracter car de unaPalabra hacer

unHijo \leftarrow nodoActual.**obtenerHijo** (car)

Si unHijo = nulo entonces

unHijo \leftarrow crear nuevo nodo trie

nodoActual.agregar (unHijo, car) // depende de la estructura

fin si

nodoActual \leftarrow unHijo

fin para cada

nodoActual.esFindePalabra \leftarrow VERDADERO

Fin

Trie comprimido – “patricia”

“Practical Algorithm to Retrieve Information
Coded in Alphanumeric”



- Ver que muchos nodos del trie tienen sólo un hijo....
- El trie comprimido T asegura que todos los nodos tengan **al menos dos hijos** (y como máximo d)
- La cantidad de nodos internos de T con L hojas será como máximo $L-1$
- Si s es la cantidad de strings en S , entonces el tamaño de T será $O(s)$
- Esto se asegura comprimiendo las cadenas de nodos que tienen sólo un hijo en aristas individuales

“patricia” – representación compacta

