Algoritmos y Estructuras de Datos

Arboles genéricos y Tries

Universidad Católica del Uruguay

1

Arboles genéricos y tries

Resultados esperados del aprendizaje:
Al finalizar esta unidad de aprendizaje serás capaz de:

- Representar árboles genéricos y recorrerlos de diferentes maneras
- Identificar problemas reales en que sea conveniente utilizar árboles genéricos
- Analizar la utilidad y conveniencia de utilizar tries para diferentes aplicaciones reales.
- Identificar formas de representación eficientes de tries y tries comprimidos
- Implementar en lenguaje JAVA y utilizar tries en diferentes problemas reales - como la construcción de diccionarios e índices

2

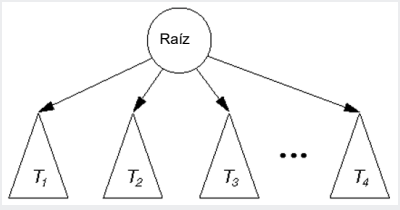
ARBOLES

Definición formal:

- conjunto finito T de uno o más nodos tales que:
  - a) existe un nodo especialmente designado, llamado raíz del árbol, raíz(T); y
  - b) Los restantes nodos (excluyendo la raíz), están distribuidos en m >= 0 conjuntos disjuntos T_1,.. ,T_m, y cada uno de estos conjuntos es, a su vez, un árbol. Los árboles T_1,.. ,T_m son llamados "subárboles" de la raíz.

3

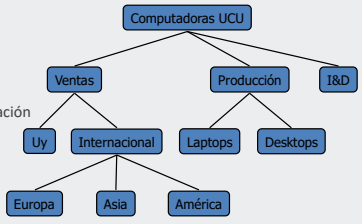
Un árbol visto recursivamente



4

¿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



5

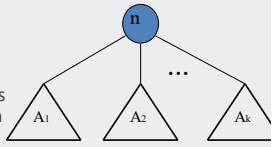
Terminología de árboles genéricos

- Grado: número de subárboles de un nodo.
- Nodo terminal y nodo interno: nodo terminal es el que tiene grado cero, también llamado “hoja”. Un nodo interno es aquél que no es terminal.
- Nivel de un nodo respecto a la raíz: la raíz tiene nivel 0, y otros nodos tienen un nivel que es superior en uno al que tiene con respecto al subárbol Tj, de la raíz, que los contiene.

6

Recorridos de los árboles genéricos

- Un recorrido visita los nodos del árbol de una forma sistemática
- Los recorridos u ordenamientos se definen recursivamente como sigue:
  - Si un árbol A es nulo, entonces el listado de los nodos de A en preorden, inorden o postorden, es la lista vacía.
  - Si A contiene un sólo nodo, entonces ese nodo constituye el listado en los tres órdenes.



7

Recorridos de Arboles genéricos

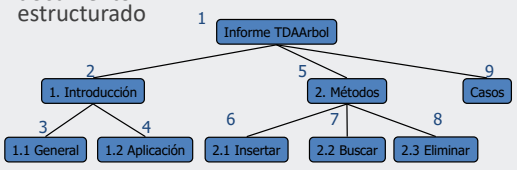
- Preorden : Raíz de A, seguido de los nodos de A1 en preorden, luego los de A2 en preorden, etc.
- Inorden : Nodos de A1 en inorden, luego la raíz, luego los nodos de los restantes subárboles en inorden.
- Postorden: Nodos de A1 en postorden, luego los de A2 en postorden, hasta el final, y luego la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

8

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)

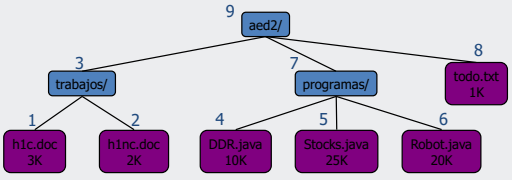


9

Recorrida en Postorden

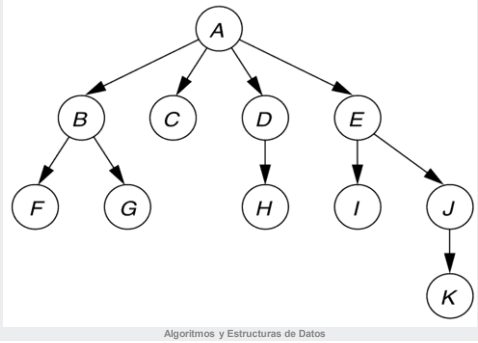
- En un recorrido en postorden, un nodo es visitado después de sus descendentes
- Aplicación: calcular el espacio usado por archivos en un directorio y sus subdirectorios

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



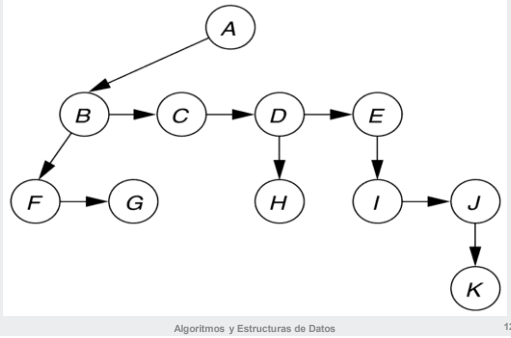
10

Recorrer el siguiente árbol en preorden y postorden



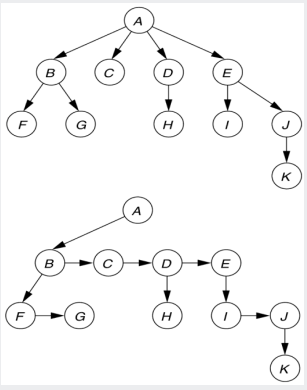
11

Representación de árboles genéricos Primer hijo – siguiente hermano / hermano derecho



12

Primer hijo – hermano derecho



13

Recorridas de árboles genéricos - preorden

TArbolGenerico.preOrden;
COM
 SI Raiz <> nulo
  Raiz.preOrden;
 FINSI
FIN

TNodoArbolGenerico.preOrden;
COM
 Imprimir(etiqueta);
 unHijo ← PrimerHijo;
 MIENTRAS unHijo <> nulo hacer
  unHijo.preOrden;
  unHijo ← unHijo.HermanoDerecho;
 FIN MIENTRAS
FIN

14

Recorridas de árboles genéricos - postorden

TArbolGenerico.postOrden;
COM
 SI Raiz <> nulo
  Raiz.postOrden
 FINSI
FIN

TNodoArbolGenerico.postOrden;
COM
 unHijo ← PrimerHijo
 MIENTRAS unHijo <> nulo hacer
  unHijo.postOrden
  unHijo ← unHijo.HermanoDerecho
 FIN MIENTRAS
 Imprimir(etiqueta)
FIN

15

El TDA ARBOL. Operaciones a ser consideradas

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

16

Búsqueda

- Por comparación de claves
  - Lo que hemos visto en Listas y Arboles hasta ahora
- "Digital", por comparación de los dígitos o caracteres individuales de la clave
  - Tries
- Por transformación de clave
  - "hashing"

17

Caso de Estudio: índice de un libro muy grande (o conjunto de documentos) (1)

- Supongamos que deseamos construir el índice para un libro grande (ej., La Biblia), con millones de palabras
- Tareas que tenemos que desarrollar:
  1. ¿qué es un índice? ¿para qué sirve?
  2. "leer" el libro, separar las palabras, crear las estructuras para representarlas, así como sus ocurrencias (números de página) – discutir alternativas estructurales
  3. Proveer funcionalidades para buscar una palabra e indicar cuántas veces está en el libro y en qué Páginas
  4. analizar órdenes de ejecución en función de las estructuras identifcadas en 3.

18

Caso de Estudio: índice de un libro muy grande (o conjunto de documentos) (2)

- Estructuras
  - ¿Listas? ¿Arboles?
  - ¿cómo serían los nodos para almacenar la info?
- Operaciones
  - ¿cómo buscaríamos en ese "índice"?
- ¿Cuál sería el orden del tiempo de ejecución?

19

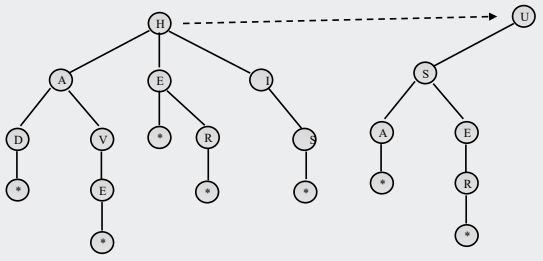
Caso de Estudio: índice de un libro muy grande (o conjunto de documentos) (2)

Tarea más compleja:

1. ¿Cómo implemenar una funcionalidad que permita, dado un prefijo, listar todas las palabras del índice que tengan ese prefijo, con la información correspondiente?
2. ¿cuál sería el orden del tiempo de ejecución, para cada alternativa estructural?

20

Búsqueda – tries como bosques



Por ejemplo, los arboles correspondiente a todas las claves que comienzan con "H" y "U" serían:

21

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***.

22

Tries

- Por lo tanto, un trie ***T*** representa las strings de ***S*** con caminos desde la raíz hasta los nodos externos de ***T***
- Enfoque adecuado cuando se realiza una serie de consultas sobre un texto fijo.
- Estructura apropiada para almacenar cadenas para soportar **comparación de patrones** rápida (proporcional al tamaño del patrón).
- En una aplicación de recuperación de la información, (ej.: secuencia de ADN en una base de datos genómica), la entrada es una colección ***S*** de strings, definidas mediante un alfabeto determinado

23

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).

24

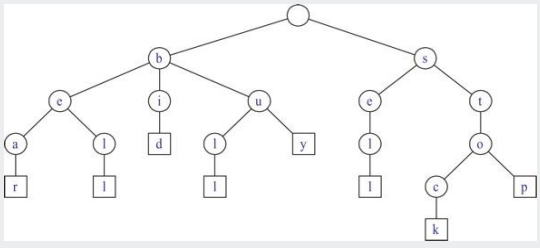
Trie estándar

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

25

Trie estándar para las palabras:

- bear, bell, bid, bull, buy, sell, stock, stop



- ¿costo de buscar "bet"?

26

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"**

27

Búsqueda en Tries

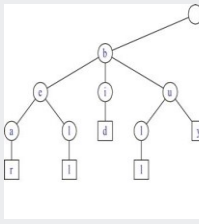
Dos operaciones principales:

- 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

28

Búsqueda en trie – palabra completa

- Seguir el camino desde la raíz, indicado por los caracteres de la entrada
- Si el camino puede ser recorrido y termina en un nodo externo, la palabra está en el diccionario (ej. "**bull**")
- Si no se puede recorrer o termina en un nodo interno, la palabra no está en el diccionario (ejemplos **"be"**, "**bet**")
- ¿tiempo de ejecución para una string de tamaño *m*?



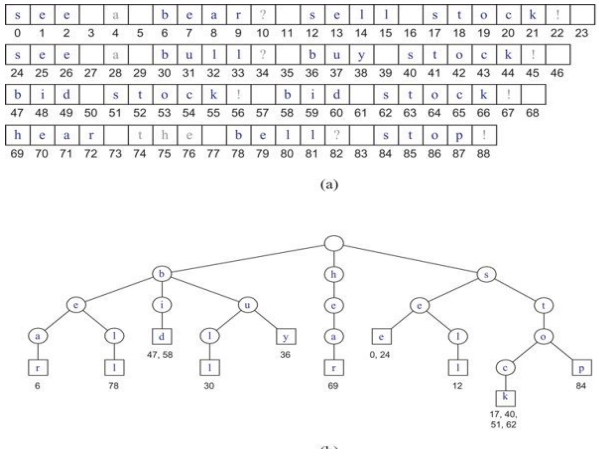
29

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!**

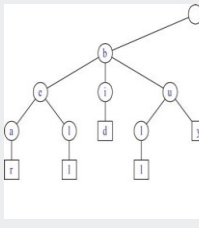
30



31

Búsqueda de prefijos en trie

- Seguir el camino desde la raíz, indicado por los caracteres de la entrada (prefijo)
- Similar a la búsqueda de palabras completas, pero termina en un nodo interno (ejemplos **"be"**, "**bu**")
- A partir de ese nodo, podemos encontrar todas las palabras de las cuales el patrón pasado es prefijo, simplemente recorriendo para todos los subárboles.

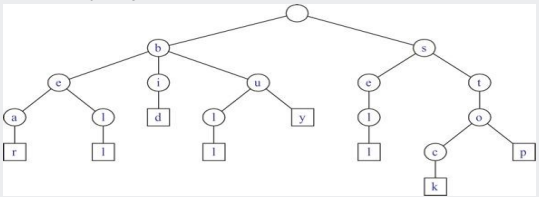


Algoritmos y Estructuras de Datos 32

32

Búsqueda en tries

- Representar el siguiente trie utilizando carácter especial de fin de string "*"
- Buscar en el mismo las siguientes palabras:
  "seal", "buy", "buyer", "head", "stop"
- Indicar todas las palabras del trie que tengan "be" como prefijo.



33

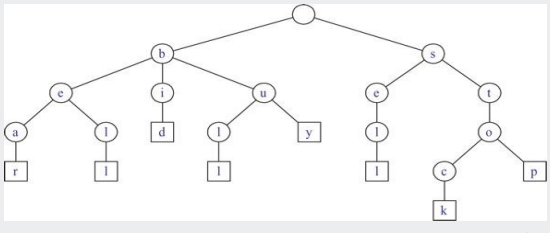
Inserción en tries

- **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

34

Inserción en tries

- Representar el siguiente trie utilizando carácter especial de fin de string **"*"**
- Insertar las siguientes palabras:
  - "beat", "seat", "buyer", "head", "heal", "sea"



35

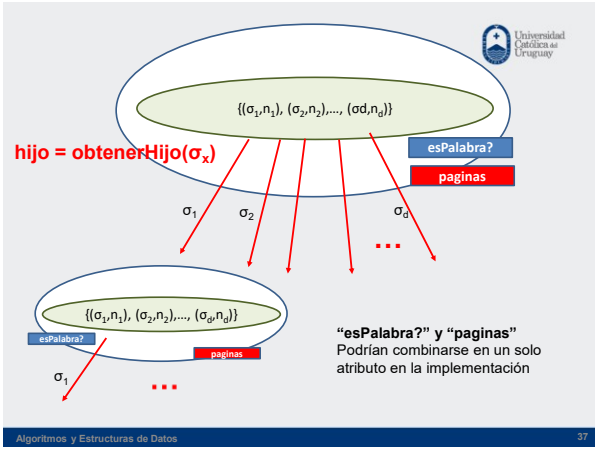
Representación del nodo del trie…?

- Cada nodo puede tener hasta ***d*** (tamaño del alfabeto) subárboles, o sea, hasta ***d*** referencias a nodos trie (una para cada posible carácter del alfabeto)
- La forma en que realizamos esta representación impacta en el consumo de memoria….
- Operativamente, lo que nos interesa es, dado un carácter del alfabeto, obtener la referencia al subárbol correspondiente (si existe)

  TnodoTrie hijo = obtenerHijo (caracter)

- *Mapas <k,v>*
  - *Ejemplo mapa(caracter,nodoTrie)*

36



37

Para cada alfabeto una representación… ¿cómo realizamos "obtenerHijo(caracter)"?

- Griego
  - "*αλγόριθμοι*"
- a..z (en inglés, minúsculas, 26 letras)
  - *"alabastro"*
- Alfabeto español estándar
  - "*Mañanitas"*
- Bases nitrogenadas de cadenas genómicas
  - "TACGAACCGAGCATTAACAGT"
- Números telefónicos (Código de país, Código de ciudad, Código de área, número) {0,1,…9)
  - +598 2 487 2717
- Codificación moderna universal de caracteres ***UNICODE***

38

Representaciones…

- Nodos internos con **vectores** de ***d*** referencias a subárboles
- Nodos internos con ***listas encadenadas*** de referencias a subárboles
- Árboles?
- Tablas de hash? (las veremos en la próxima unidad)
- ¿Otras más eficientes?

39

Búsqueda en tries

En la clase **NodoTrie buscar (string unaPalabra):** devuelve el nodo que corresponde al último carácter del argumento, el nodo nulo si ese argumento no está en el TRIE

```
Comienzo
    nodoActual ← this
    Para cada caracter car de unaPalabra hacer
        unHijo ← nodoActual.obtenerHijo (car) //depende de la estructura del nodo
        Si unHijo = nulo entonces
            devolver nulo
        Sino
            nodoActual ← unHijo
        fin si
    fin para cada
    Si nodoActual.esFinDePalabra entonces
        devolver nodoActual
    sino
        devolver nulo
    fin si
Fin
```


42

Inserción en tries

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

```
Comienzo
    nodoActual ← this
    Para cada caracter car de unaPalabra hacer
        unHijo ← nodoActual.obtenerHijo (car)
        Si unHijo = nulo entonces
            unHijo ← crear nuevo nodo trie
            nodoActual.agregar (unHijo, car) // depende de la estructura
        fin si
        nodoActual ← unHijo
    fin para cada
    nodoActual.esFindePalabra ← VERDADERO
Fin
```


43

Imprimir todas las strings del trie…

En la clase **NodoTrie imprimir (string cadena, nodo trie nodoActual):** imprime por consola todas las palabras contenidas en el subárbol que tiene como raíz al nodo pasado inicialmente, concatenadas con la cadena inicial. Si la cadena inicial es vacía y el nodo inicial es la raíz del trie, imprime todas las palabras del trie.

```
Comienzo
    Si nodoActual no nulo entonces
        Si nodoActual.esFinDePalabra entonces
            dar salida por pantalla a la cadena
        Fin si
        Para cada hijo de nodoActual hacer
            imprimir (cadena + carácter del hijo , hijo)
        Fin para cada
    Fin si
Fin
```


44

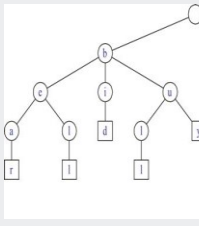
Búsqueda de prefijos

- ¿existe?
- ¿devolver todas las strings de las cuales es prefijo?

45

Búsqueda de prefijos en trie

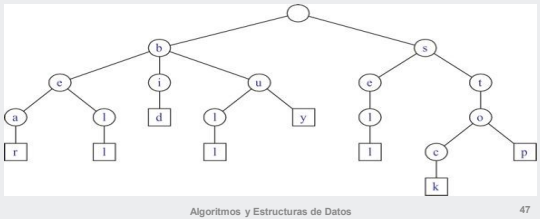
- Seguir el camino desde la raíz, indicado por los caracteres de la entrada (prefijo)
- Similar a la búsqueda de palabras completas, pero termina en un nodo interno (ejemplos **"be"**, "**bu**")
- A partir de ese nodo, podemos encontrar todas las palabras de las cuales el patrón pasado es prefijo, simplemente recorriendo para todos los subárboles.



46

Búsqueda de strings que tengan un cierto prefijo…

- Indicar todas las palabras del trie que tengan "be" como prefijo.



47

Trabajo de aplicación 3

- Ejercicio 0
  - Seudocódigo abstracto para buscar por prefijo
- Ejercicio 1
  - "predecir" o "autocompletar" (búsqueda de todas las strings del trie que tengan una cierta cadena, pasada por parámetro, como prefijo)

48

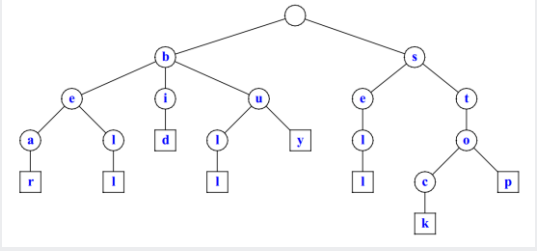
En perspectiva: Métodos de Búsqueda…

Varias formas de realizar búsquedas de información en estructuras de datos:

- Búsqueda por *comparación* de claves (listas, árboles)
- Búsqueda "*digital*", dirigida por los "dígitos" de la clave (tries)
- Búsqueda por *transformación* de claves (próxima UT)

53

¿y cuánta memoria ocupa este trie?



54

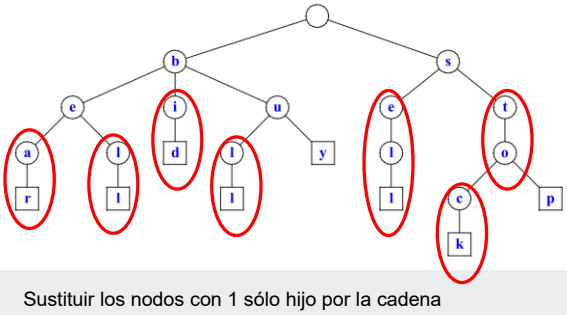
Trie comprimido – "*patricia*"

"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

55

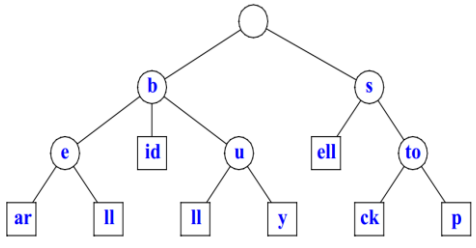
Trie comprimido – "*patricia*"



Sustituir los nodos con 1 sólo hijo por la cadena correspondiente….

56

Trie comprimido – "*patricia*" *¿cómo sería?*



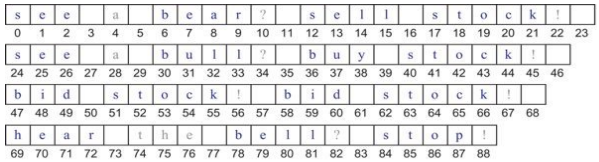
- ¿pero no se ocupa memoria al guardar estas cadenas?

57

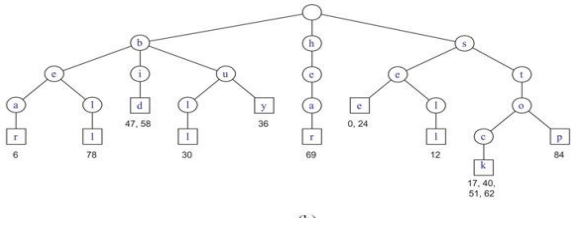
Trie comprimido – "patricia"

- ventajoso sólo cuando se lo utiliza como una estructura de índice auxiliar
- colección de strings que ya está almacenada
- no se le requiere que almacene realmente todos los caracteres
- **S** en un array de strings S[0], S[1], ..., S[s - 1].
- Tríada **(i,j,k)** representa etiqueta **X = S[i][j..k]**
- **X** es la substring de **S[i]** que consiste de los caracteres desde el **j-esimo** al **k-esimo** inclusive

58

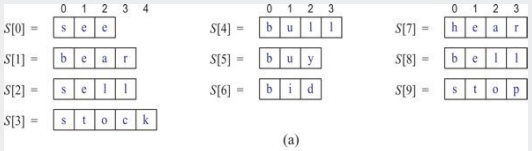


(a)

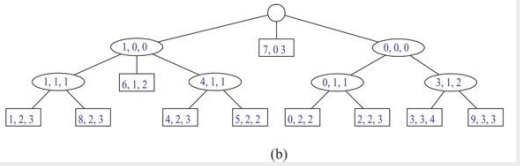


59

"patricia" – representación compacta



(a)



(b)

60

Implementaciones de Patricia

- org.apache.commons.collections4.trie

Class PatriciaTrie<E>

61

Trabajo de Aplicación 4

62

TA4 EJERCICIO 1

- ¿cómo es la estructura de datos utilizada?
- ¿cuánta memoria se ocupa, en relación al tamaño de los datos de entrada?

63