

Algoritmos y Estructuras de Datos



Arboles genéricos y Tries

1

ARBOLES



Definición formal:

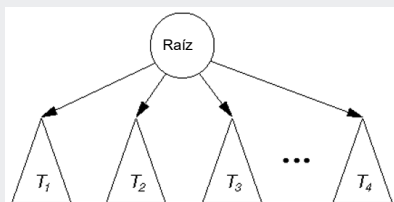
- conjunto finito T de uno o más nodos tales que:
 - a) existe un nodo especialmente designado, llamado raíz del árbol, raíz(T); y
 - b) Los restantes nodos (excluyendo la raíz), están distribuidos en $m \geq 0$ conjuntos disjuntos T_1, \dots, T_m , y cada uno de estos conjuntos es, a su vez, un árbol.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

Algoritmos y Estructuras de Datos

2

2

Un árbol visto recursivamente



Algoritmos y Estructuras de Datos

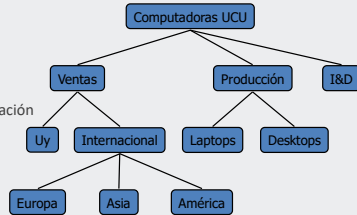
3

3

¿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



Algoritmos y Estructuras de Datos

4

4

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 T_j , de la raíz, que los contiene.

Algoritmos y Estructuras de Datos

5

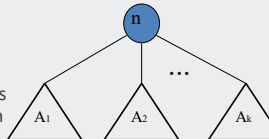
5

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.



Algoritmos y Estructuras de Datos

6

6



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.

Algoritmos y Estructuras de Datos

7

7



Recorrida en Preorden

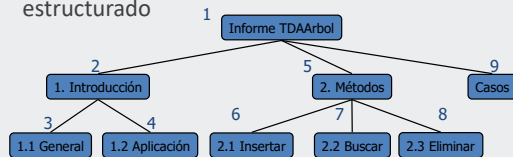
- En un recorrido en preorden, un nodo es visitado antes que sus descendientes
- Aplicación: imprimir un documento estructurado

Algoritmo *preOrden(v)*

visitar(v)

Para cada hijo *w* de *v*

preorden(w)



Algoritmos y Estructuras de Datos

8

8



Recorrida en Postorden

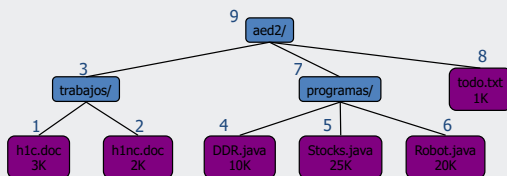
- En un recorrido en postorden, un nodo es visitado después de sus descendientes
- Aplicación: calcular el espacio usado por archivos en un directorio y sus subdirectorios

Algoritmo *postOrden(v)*

Para cada hijo *w* de *v*

postOrden(w)

visitar(v)

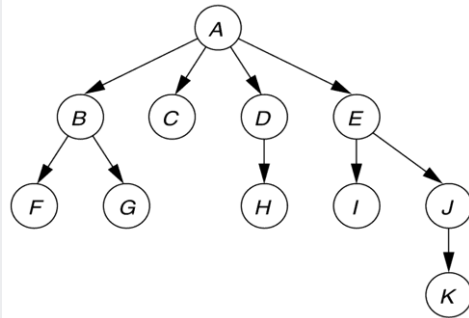


Algoritmos y Estructuras de Datos

9

9

Recorrer el siguiente árbol en
preorden y postorden

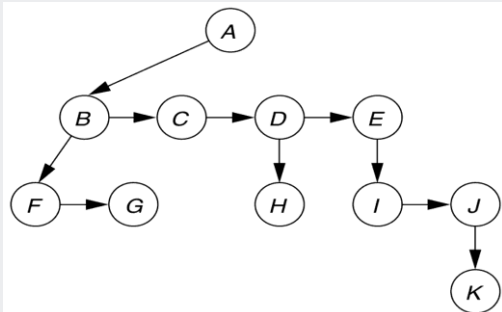


Algoritmos y Estructuras de Datos

10

10

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

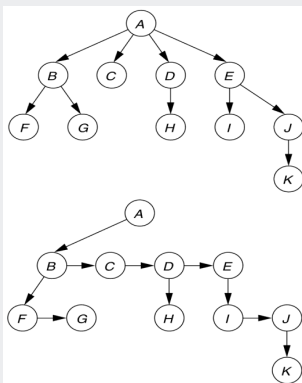


Algoritmos y Estructuras de Datos

11

11

Primer hijo – hermano derecho



Algoritmos y Estructuras de Datos

12

12

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
  
```

Algoritmos y Estructuras de Datos

13

13

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
  
```

Algoritmos y Estructuras de Datos

14

14

El TDA ARBOL. Operaciones a ser consideradas



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

Algoritmos y Estructuras de Datos

15

15

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 identificadas en 3.

Algoritmos y Estructuras de Datos

16

16

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?

Algoritmos y Estructuras de Datos

17

17

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



Tarea más compleja:

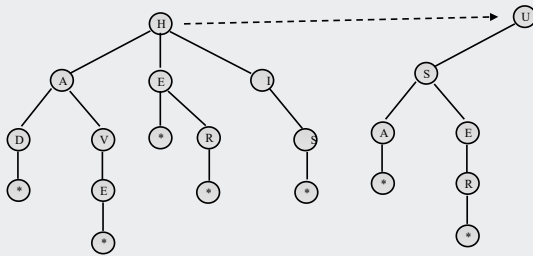
1. ¿Cómo implementar una funcionalidad que permita, dado un prefijo, listar todas las palabras del índice que tengan ese prefijo, con la información correspondiente?
2. ¿cuál sería el orden del tiempo de ejecución, para cada alternativa estructural?

Algoritmos y Estructuras de Datos

18

18

Búsqueda – tries como bosques



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

Algoritmos y Estructuras de Datos

19

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 .

Algoritmos y Estructuras de Datos

20

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

Algoritmos y Estructuras de Datos

21

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).

Algoritmos y Estructuras de Datos

22

22

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

Algoritmos y Estructuras de Datos

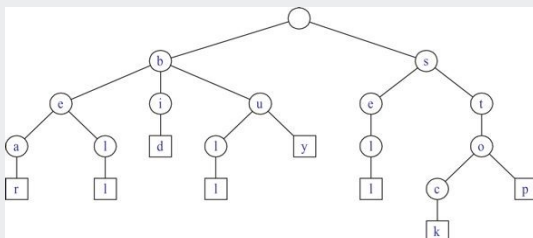
23

23

Trie estándar para las palabras:



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



- ¿costo de buscar "bet"?

Algoritmos y Estructuras de Datos

24

24

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!)

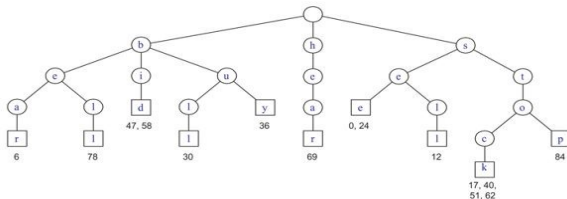
Algoritmos y Estructuras de Datos

28

28

s	e	e	a	b	e	a	r	?	s	e	l	l	s	t	o	c	k	!						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
s	e	e	a	b	u	l	l	?	b	u	y	s	t	o	c	k	!							
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46		
b	i	d	s	t	o	c	k	!	b	i	d	s	t	o	c	k	!							
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68			
h	e	a	r	t	h	e	b	e	l	l	?	s	t	o	p	!								
69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88					

(a)

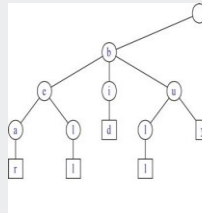


29

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

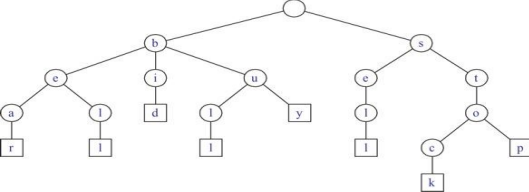
30

30

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.



31

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

Algoritmos y Estructuras de Datos

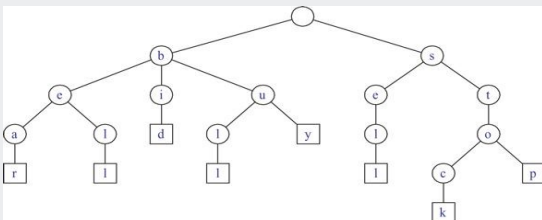
32

32

Inserción en tries



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



Algoritmos y Estructuras de Datos

33

33

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)

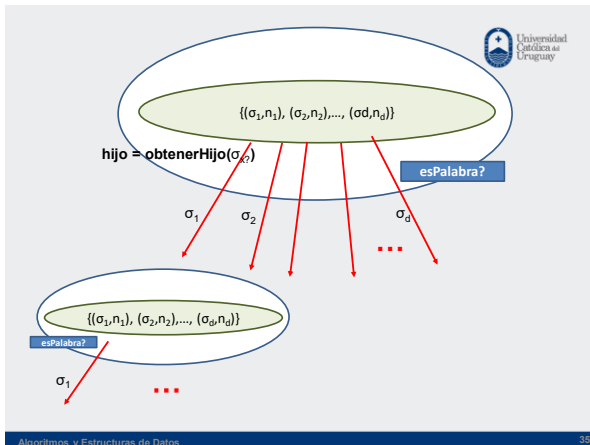
TnodoTrie hijo = obtenerHijo (caracter)

- Mapas $\langle k, v \rangle$
– Ejemplo `mapa(caracter, nodoTrie)`

Algoritmos y Estructuras de Datos

34

34



Algoritmos y Estructuras de Datos

35

35

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
– “TACGAACCGAGCATTACAGT”
- 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**

Algoritmos y Estructuras de Datos

36

36

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?

Algoritmos y Estructuras de Datos

37

37

Búsqueda e inserción en tries



Se puede implementar con un

- Enfoque iterativo: iterando por cada letra de la palabra
- Enfoque recursivo: avanzando recursivamente a cada nodo del árbol

Algoritmos y Estructuras de Datos

38

38

Búsqueda e inserción en tries



- Enfoque iterativo: iterando por cada letra de la palabra

Algoritmos y Estructuras de Datos

39

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
  
```

Algoritmos y Estructuras de Datos

40

40

Insertión 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
  
```

Algoritmos y Estructuras de Datos

41

41

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
  
```

Algoritmos y Estructuras de Datos

42

42

Búsqueda de prefijos



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

Algoritmos y Estructuras de Datos

43

Búsqueda e inserción en tries



Enfoque recursivo: avanzando recursivamente a cada nodo del árbol

Algoritmos y Estructuras de Datos

44

Trie utilizando árboles genéricos (Primer hijo – Hermano derecho)



BUSCAR UNA PALABRA EN UN TRIE

Buscar (unaPalabra) : devuelve un tipo lógico

COM

Carácter ← unaPalabra.PrimerO

unaPalabra.EliminarPrimerO

HijoActual ← PrimerHijo

Mientras HijoActual <> nulo y HijoActual.Etiqueta <> Carácter hacer

HijoActual ← HijoActual.HermanoDerecho

Fin mientras

Si HijoActual = nulo entonces

Devolver FALSO

Si HijoActual.Etiqueta = "*" entonces

Devolver VERDADERO

Sino

Devolver HijoActual.Buscar(unaPalabra)

Finsi

Finsi

FIN

Algoritmos y Estructuras de Datos

45

Inserción en un TRIE implementado como árbol genérico (Primer hijo – Hermano Derecho)



Insertar (unaPalabra): TNodeArbol

COM

Carácter ← unaPalabra.PrimerO

unaPalabra.EliminarPrimerO

Si PrimerHijo = nulo entonces

HijoPorElCualSeguir ← CrearNuevoNodo con etiqueta igual a carácter

PrimerHijo ← HijoPorElCualSeguir

Sino

HijoPorElCualSeguir ← PrimerHijo.BuscaOInserta(Carácter)

Finsi

Si Carácter <> "*" entonces

HijoPorElCualSeguir.Insertar (unaPalabra)

Finsi

FIN

BuscaOInserta (Carácter):

Recorre la lista de hermanos encabezada por el primer hijo, buscando un nodo con etiqueta igual a "Carácter". Si lo encuentra, devuelve el nodo. Si no lo encuentra, inserta ordenado en esa lista de hermanos un nodo con etiqueta igual a "carácter" y lo devuelve.

Algoritmos y Estructuras de Datos

46

46

Eliminación de una palabra en un TRIE implementado como árbol genérico



Eliminar (unaPalabra): devuelve un tipo lógico

COM

Carácter ← unaPalabra.PrimerO

unaPalabra.EliminarPrimerO

HijoActual ← PrimerHijo

Mientras HijoActual <> nulo y HijoActual.etiqueta <> carácter hacer

HijoActual ← HijoActual.HermanoDerecho

Fin mientras

Si HijoActual = nulo entonces

Devuelve FALSO

Sino Si Carácter = "*" entonces // llegó al final del camino

PrimerHijo.Quitar(HijoActual)

Sino

HijoActual.Eliminar(unaPalabra) // sigue recorriendo el camino

Si HijoActual.EsHoja entonces // se fija si hay que quitar el nodo

PrimerHijo.Quitar(HijoActual)

Fin si

Fin si

Devuelve VERDADERO

Finsi

FIN

Quitar (unNodo):

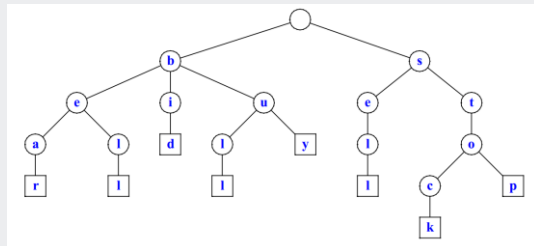
Recorre la lista de hermanos encabezada por el primer hijo, buscando unNodo y lo elimina de esa lista.

Algoritmos y Estructuras de Datos

47

47

¿y cuánta memoria ocupa este trie?



Algoritmos y Estructuras de Datos

48

48

Trie comprimido – “patricia”

“Practical Algorithm to Retrieve Information Coded in Alphanumeric”



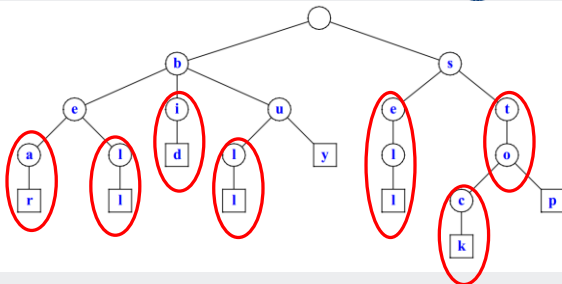
- Ver que muchos nodos del trie tienen sólo un hijo....
- El trie comprimido T asegura que todos los nodos tengan **al menos dos hijos** (y como máximo d)
- La cantidad de nodos internos de T con L hojas será como máximo $L-1$
- Si s es la cantidad de strings en S , entonces el tamaño de T será $O(s)$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

Algoritmos y Estructuras de Datos

49

49

Trie comprimido – “patricia”



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

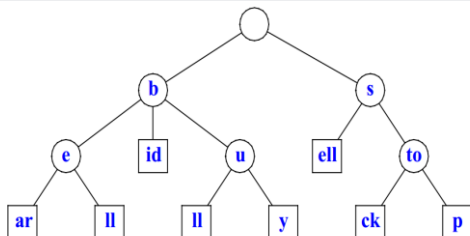
Algoritmos y Estructuras de Datos

50

50

Trie comprimido – “patricia”

¿cómo sería?



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

Algoritmos y Estructuras de Datos

51

51

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 $S[0], S[1], \dots, S[s - 1]$.
- 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

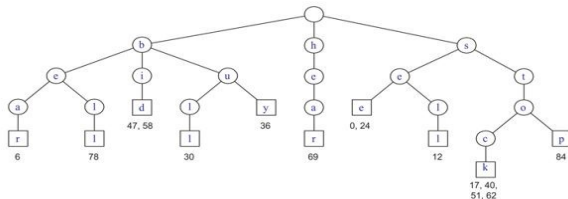
Algoritmos y Estructuras de Datos

52

52

s	e	e		a		b	e	a	r	?		s	e	i	l	l		s	t	o	e	k	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s	e	e		a		b	u	i	l	?		b	u	y		s	t	o	e	k	!		
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	
b	i	d		s	t	o	e	k	!			b	i	d		s	t	o	e	k	!		
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68		
h	e	a	r		t	h	e		b	e	!	?		s	t	o	p	!					
69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88				

(a)



53

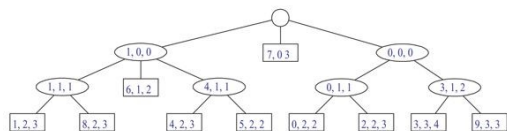
“patricia” – representación compacta



The diagram illustrates the steps of the bubble sort algorithm on the array [s, e, e, l, l]. Each step shows the array with indices 0 to 4 above it. Elements are enclosed in boxes, and arrows indicate comparisons or swaps.

- S[0]**: Initial array: [s, e, e, l, l].
- S[1]**: First pass, first swap: [b, e, e, a, r].
- S[2]**: First pass, second swap: [s, e, l, l, l].
- S[3]**: First pass, third swap: [s, t, o, c, k].
- S[4]**: Second pass, first swap: [b, u, l, l, l].
- S[5]**: Second pass, second swap: [b, i, d, l, l].
- S[6]**: Third pass, first swap: [h, e, a, r, l].
- S[7]**: Third pass, second swap: [b, e, l, l, r].
- S[8]**: Third pass, third swap: [s, t, o, c, p].
- S[9]**: Final sorted array: [s, t, o, c, p].

(a)



(b)

Algoritmos y Estructuras de Datos

34

54

Implementaciones de Patricia



- org.apache.commons.collections4.trie

Class PatriciaTrie<E>

Algoritmos y Estructuras de Datos

55

55

Trabajo de Aplicación 4



Algoritmos y Estructuras de Datos

56

56

TA4 EJERCICIO 1



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

Algoritmos y Estructuras de Datos

57

57
