UT5 - PD2

Ejercicio #1

(1) Vamos a usar un Trie.
Cada nodo del Trie debe almacenar:
→ Un Array de 26 punteros (hijos [0...25])
→ Un campo esPalabra, que es booleano, para saber si el camino hasta ese nodo corresponde a una palabra.
→ Una lista de int "páginas" donde se guardarán todas las páginas en donde aparece la infame palabra.

(2) Descripción

(a) Se recorre el texto palabra a palabra, y para cada palabra W junto con su número de página P, tenemos a hacer:
→ Me paro en la raíz del trie, y para CADA caracter pp de W:
- Calculo el indice (Index = pp - 'a')
- Si NO TIENE hijos, creo un nuevo nodo.
- Avanzo a nodo = nodo.hijos[index]

Una vez se haga dicha tarea con todos los caracteres pp de la palabra W, marco el campo esPalabra = True y añado p a nodo.páginas.

PRONTO

(b) Pre:
- p es int >= 1
- W está en minúsculas
- Trie inicializado Raiz != null

(c) Método InsertarPalabra(W, p) (p → páginas, W → Palabras)

Nodo ← trie.raiz
Para i desde 0 hasta W.lenght - 1
{ pp ← W[i]
Index ← Pp - 'a'

Si nodo.hijos[index] == null
{ nodo.hijos[index] ← NewNodoTrie() }
// Creo un nuevo nodo si no existiesen los hijos
Fin si
nodo ← nodos.hijos[index] }
Fin Para

nodo.esPalabra ← True // Esto marca el fin de la Palabra

Si p no está en nodo.paginas
{ nodo.paginas.Add(p) }
Fin Si
Fin método

Método ConstruirIndice(Raiz, listaPaginas)
// listaPaginas es la lista en donde estan las PAGINAS y Palabras
Para CADA (W, p) en listaPaginas
{ InsertarPalabra(W, p) }
Fin Para
Fin método

Post: → Existe en el Trie un camino desde la raíz hasta cada W

→ Cada palabra está identificada por el nodo W que tiene esPalabra = true

→ La lista páginas del nodo W contiene p.

→ insertarPalabra()

(d) Tiempo de ejecución O(w), pues para insertar, recorre w caracteres. Cada acceso a hijos[index] es O(1); y añadir a la lista de páginas es O(1).

→ ConstruirIndice()

Es O(n), siendo n la suma de las longitudes de todas las palabras de la lista listaPaginas.

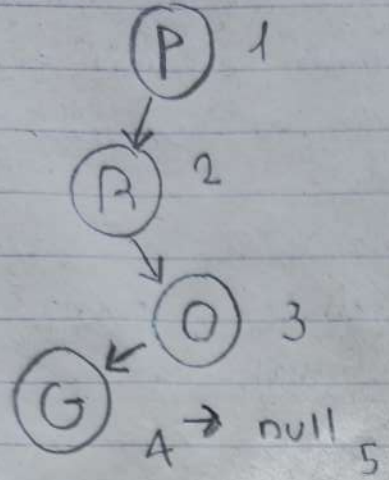
(3) Responder preguntas

¿Comparaciones con "Programa"?

P-R-O-G-R-A-M-A = 8 → El trie compara tantas veces como el largo de la palabra

¿ Proselitismo ?

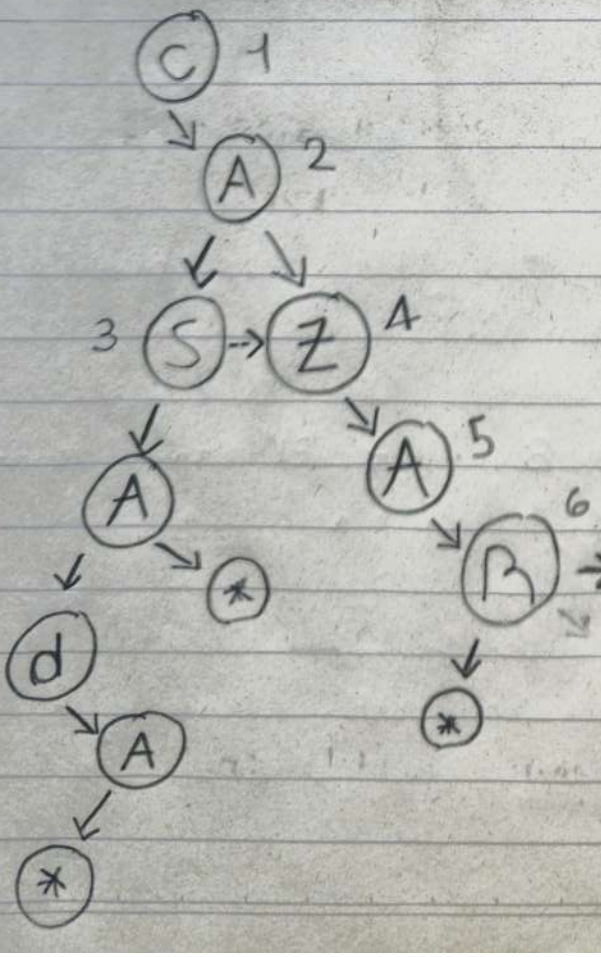
P-R-O-S-E-L-i-t-i-S-M-O (12) : m



!! Compara con G, no encuentra "S", busca a la derecha y no tiene hermanos.
Se podría haber dado de 4 a 5 comparaciones.
Como no encuentra "S" Retorna null.

Al no ser palabra, no pone esPalabra = true

¿ Cazadores ? m: 9



!! Compara con B, y busca en la lista de hermanos en busca de la "d"
7 comparaciones

Seguimos con UT5_PD2.

¿ Cuál es la Altura del trie resultante ?

Veamos cual es la palabra mas larga.
En nuestro caso: PROGRAMACION

=> 5 : 12 : Atura del Arbol (trie)

¿ Cuál es su tamaño ?

Vamo' a ver el numero total de nodos.

ALA : 3
ALIMAÑA : 5 (Compante 2)
ALABASTRO : 6 (Compante 3)
PERRO : 5
PERA : 4
ALIMENTO : 5 (Compante 3)
CASA : 4 (Compante
CASADA : 2 (Compante 4)
CAZAR : 3 (Compante 2)
PROGRAMA : 8
PROGRAMACIÓN : 4 (Compante 8)
PROGRAMAR : 1 (Compante 8)

tamaño total : 52 nodos

Ejercicio 2

Pseudocod:

```
buscarPaginas ( trie , palabrucha )
    nodo = trie.raiz
    Para (cada letra en palabrucha) hacer
        Si (letra no está en nodo.hijos) entonce
            .Retornar "Palabrucha no está en libro"
        Fin si
        nodo = nodo.hijos [ letra ]
    Si (nodo.paginas está vacio) entonce
        Retornar "la palabra no está en el libro"
    Fin si
    Retornar nodo.páginas
Fin
```

Pre : . Palabrucha debe estar compuesta por letras (a → z)
. El trie debe estar previamente construido c inicializado con palabras y sus páginas.

Post : Devuelve el conjunto de páginas donde se encuentra la palabra, si existe

Descripción : El Algoritmo busca una palabra en un trie ya construido. Recorre cada letra de la palabra. Si en algun momento no se encuentra la letra entre los hijos del nodo actual, la palabra no está. Si llega al nodo final de la palabra y esta contiene páginas, retorna ese conjunto. De lo contrario, indica que la palabra no está