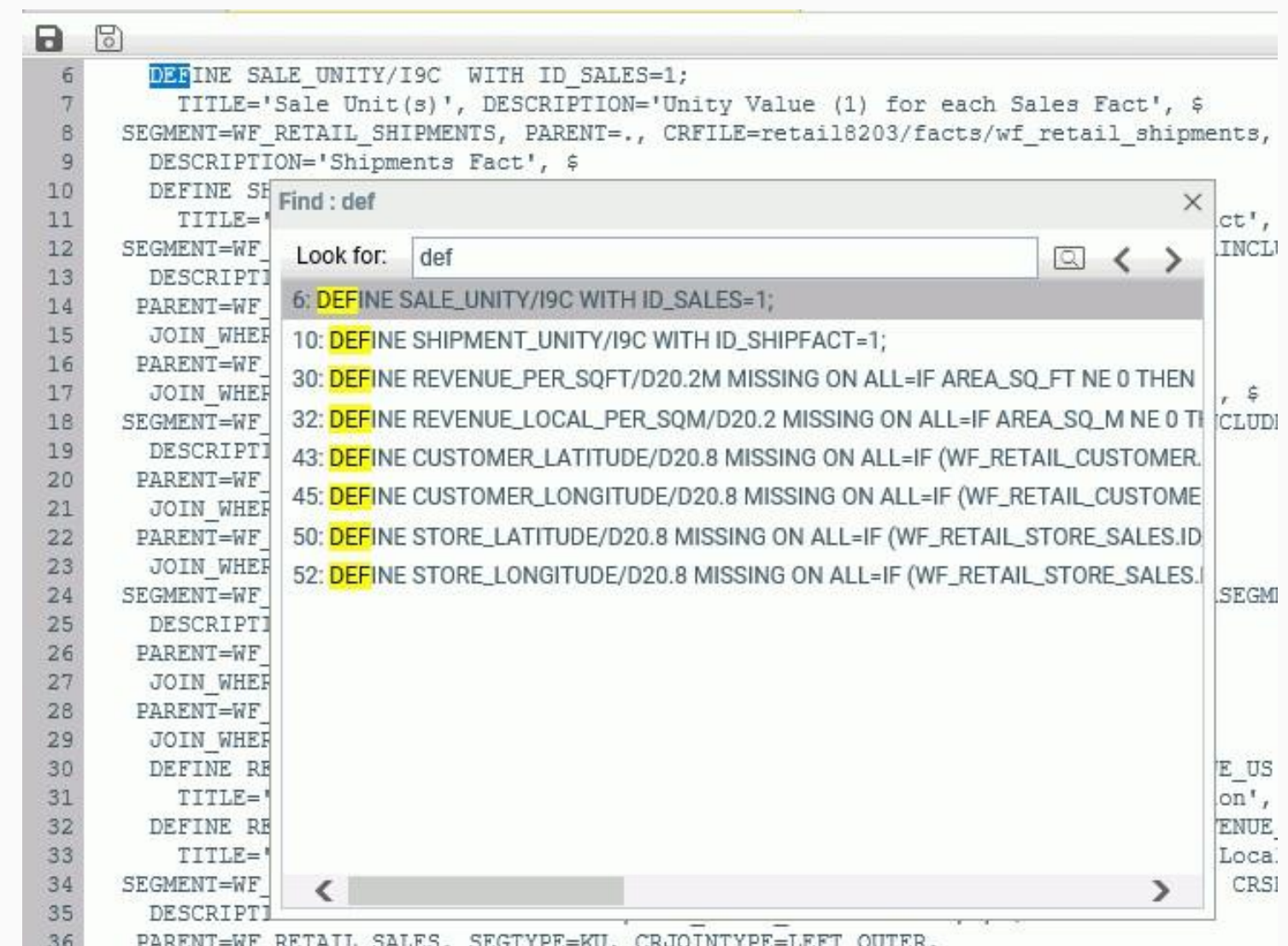


Welcome to Algorithms and Data Structures! - CS2100

String Matching

Consiste en encontrar un string corto (patrón), en un string largo (texto).

- Aplicaciones sobre editores de texto.



The screenshot shows a text editor window with a list of SQL-like statements. A search dialog box titled "Find : def" is open, displaying a list of matches. The search term "def" is entered in the "Look for:" field. The matches are listed as follows:

- 6: DEFINE SALE_UNITY/I9C WITH ID_SALES=1;
- 10: DEFINE SHIPMENT_UNITY/I9C WITH ID_SHIPFACT=1;
- 30: DEFINE REVENUE_PER_SQFT/D20.2M MISSING ON ALL=IF AREA_SQ_FT NE 0 THEN
- 32: DEFINE REVENUE_LOCAL_PER_SQM/D20.2 MISSING ON ALL=IF AREA_SQ_M NE 0 THEN
- 43: DEFINE CUSTOMER_LATITUDE/D20.8 MISSING ON ALL=IF (WF_RETAIL_CUSTOMER.
- 45: DEFINE CUSTOMER_LONGITUDE/D20.8 MISSING ON ALL=IF (WF_RETAIL_CUSTOME
- 50: DEFINE STORE_LATITUDE/D20.8 MISSING ON ALL=IF (WF_RETAIL_STORE_SALES.ID
- 52: DEFINE STORE_LONGITUDE/D20.8 MISSING ON ALL=IF (WF_RETAIL_STORE_SALES.I

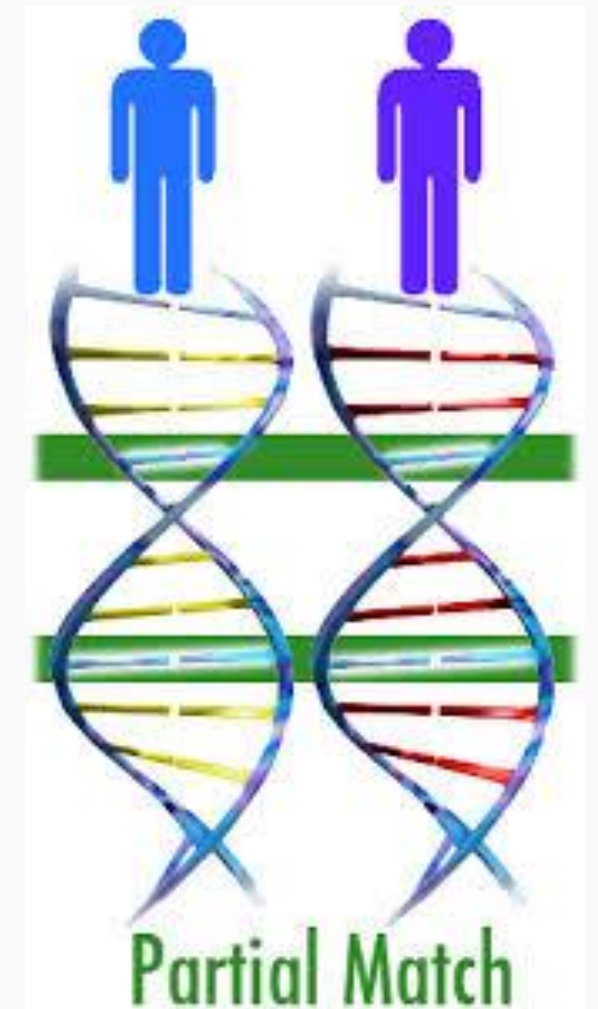
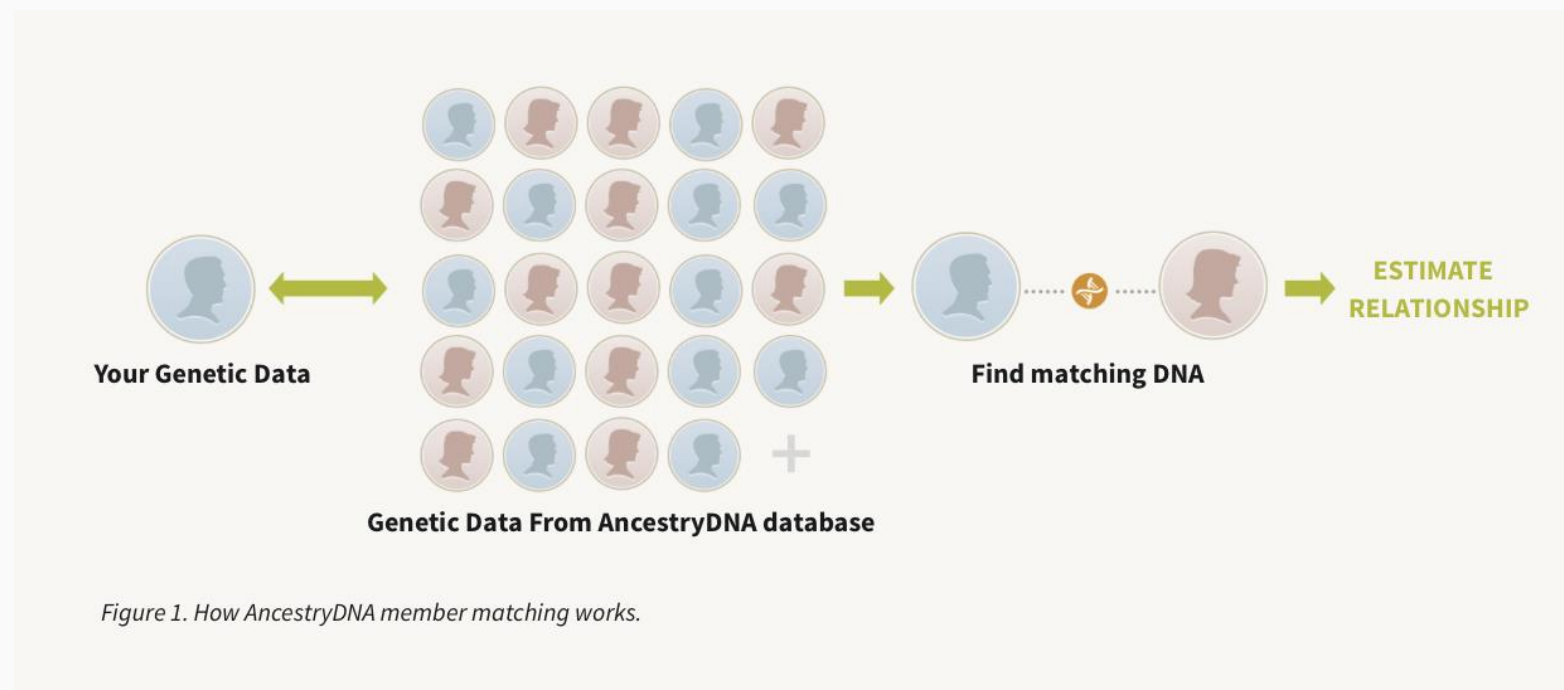
The text editor background shows the following code snippets:

```
6 DEFINE SALE_UNITY/I9C WITH ID_SALES=1;
7 TITLE='Sale Unit(s)', DESCRIPTION='Unity Value (1) for each Sales Fact', $
8 SEGMENT=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
9 DESCRIPTION='Shipments Fact', $
10 DEFINE SHIPMENT_UNITY/I9C WITH ID_SHIPFACT=1;
11 TITLE='Shipment Unit(s)', DESCRIPTION='Shipment Value (1) for each Sales Fact', $
12 SEGMENT=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
13 DESCRIPTION='Shipments Fact', $
14 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
15 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
16 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
17 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
18 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
19 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
20 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
21 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
22 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
23 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
24 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
25 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
26 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
27 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
28 JOIN_WHERE=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
29 PARENT=WF_RETAIL_SHIPMENTS, CRFILE=retail8203/facts/wf_retail_shipments,
30 DEFINE REVENUE_PER_SQFT/D20.2M MISSING ON ALL=IF AREA_SQ_FT NE 0 THEN
31 TITLE='Revenue per sqft', DESCRIPTION='Revenue per sqft', $
32 DEFINE REVENUE_LOCAL_PER_SQM/D20.2 MISSING ON ALL=IF AREA_SQ_M NE 0 THEN
33 TITLE='Revenue local per sqm', DESCRIPTION='Revenue local per sqm', $
34 SEGMENT=WF_RETAIL_SHIPMENTS, PARENT=., CRFILE=retail8203/facts/wf_retail_shipments,
35 DESCRIPTION='Shipments Fact', $
36 PARENT=WF_RETAIL_SALES, SEGTYPE=KII, CRJOINTYPE=LEFT OUTER.
```

String Matching

En bioinformática lo más frecuente es buscar un fragmento nuevo de ADN (un gen) en una colección de secuencias

- En este caso permitimos un cierto error, pero el pattern matching exacto es una subrutina.



String Matching

- Entrada
 - Dos strings sobre el alfabeto Σ
 - $T = \{t_1, t_2, \dots, t_n\}$
 - $P = \{p_1, p_2, \dots, p_m\}$
- Salida
 - El conjunto de posiciones de T donde aparece P.

Text : A A B A A C A A D A A B A A B A

Pattern : A A B A

A A B A

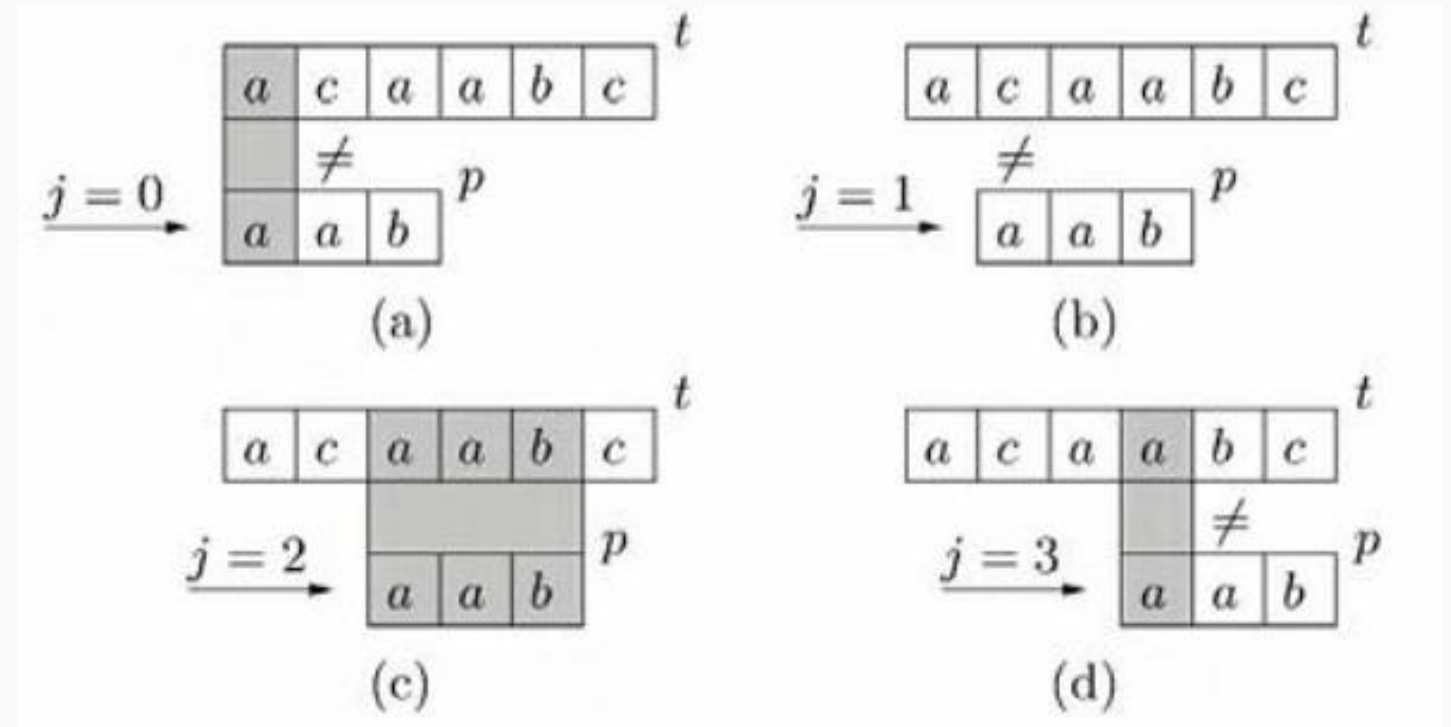
A A B A

A A B A A C A A D A A B A A B A
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A A B A

Pattern Found at 0, 9 and 12

String Matching (Algoritmo Intuitivo)

- StringMatching(Patron P, Texto T):



String Matching (Algoritmo Inocente)

- StringMatching(Patron P, Texto T):

- Result = {}
- For j=0 to n - m:
 - i = 0
 - While p[i] = T[i+j] and i < m :
 - i++
 - If i = m :
 - Result U {j}

¿Complejidad?

$O(mn)$

¿Cómo lo mejoramos?

String Matching (Algoritmo Boyer-Moore)

- Mueve el patrón sobre el texto al igual que el algoritmo anterior, con la diferencia de:
 - Cada comparación del patrón con el texto la hacemos empezando por el final
 - **Si podemos, nos movemos más de una posición en el texto.**
 - Se debe mantener guardado la posición de cada letra del patrón.
 - Ejemplo:

<i>texto</i>																
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C

<i>patrón</i>					
0	1	2	3	4	5
T	A	T	G	T	G

<i>positions</i>	
A	1
B	-1
..	-1
G	5
...	-1
T	4
...	-1

String Matching (Algoritmo Boyer-Moore)

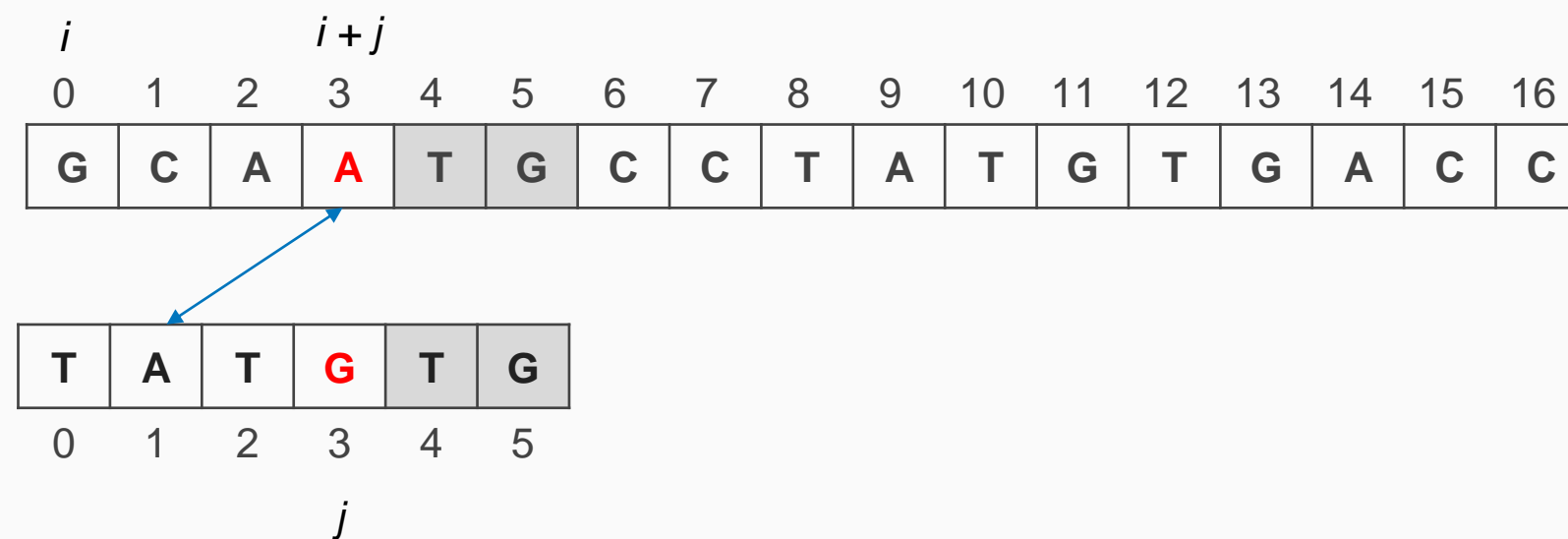
- Ejemplo

i					$i+j$											
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C

T	A	T	G	T	G
0	1	2	3	4	5
					j

String Matching (Algoritmo Boyer-Moore)

- Ejemplo



positions

A	1
B	-1
..	-1
G	5
...	-1
T	4
...	-1

String Matching (Algoritmo Boyer-Moore)

- Ejemplo

i				$i+j$												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C

T	A	T	G	T	G
0	1	2	3	4	5



$$3 - 1 = 2$$

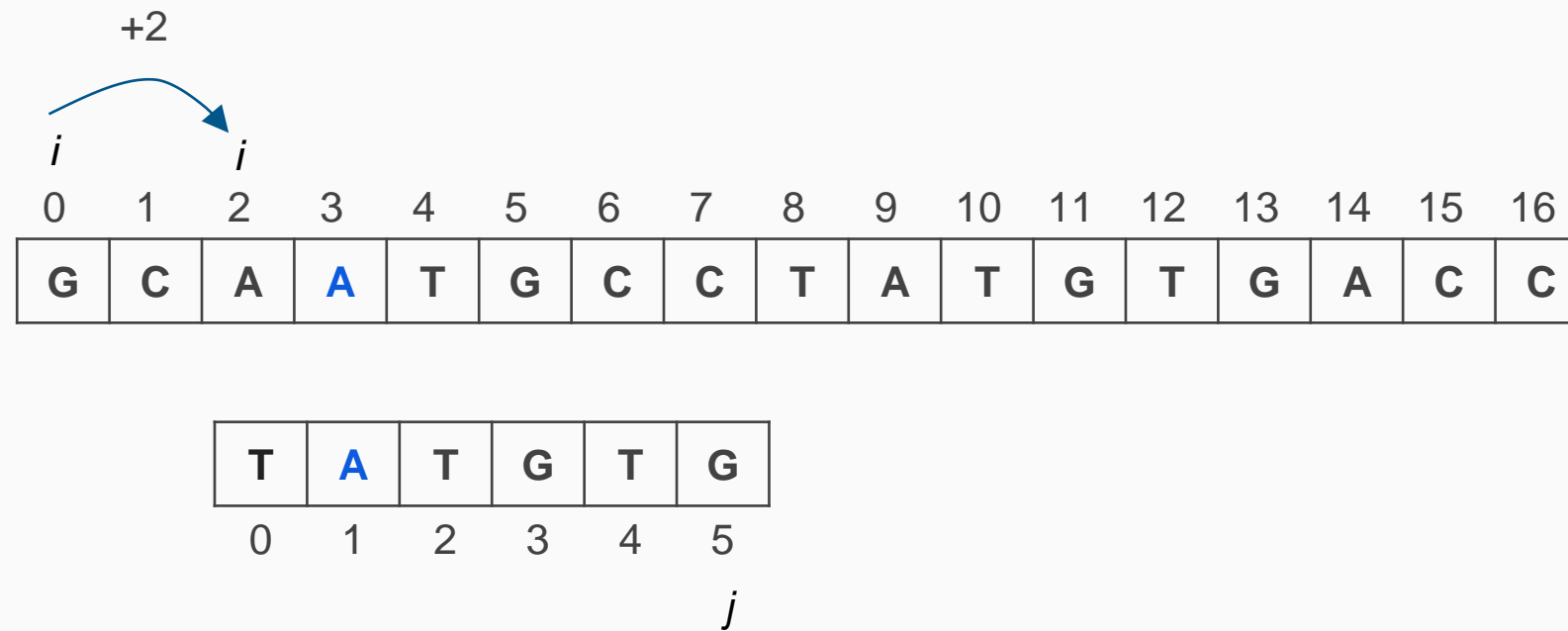
$$positions[A] = 1$$

positions

A	1
B	-1
..	-1
G	5
...	-1
T	4
...	-1

String Matching (Algoritmo Boyer-Moore)

- Ejemplo



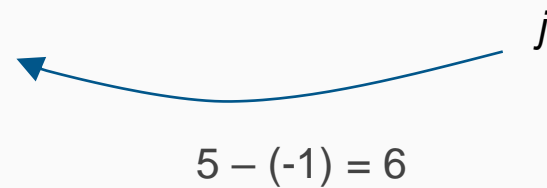
String Matching (Algoritmo Boyer-Moore)

- Ejemplo

		<i>i</i>														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C

T	A	T	G	T	C
0	1	2	3	4	5

$positions[C] = -1$

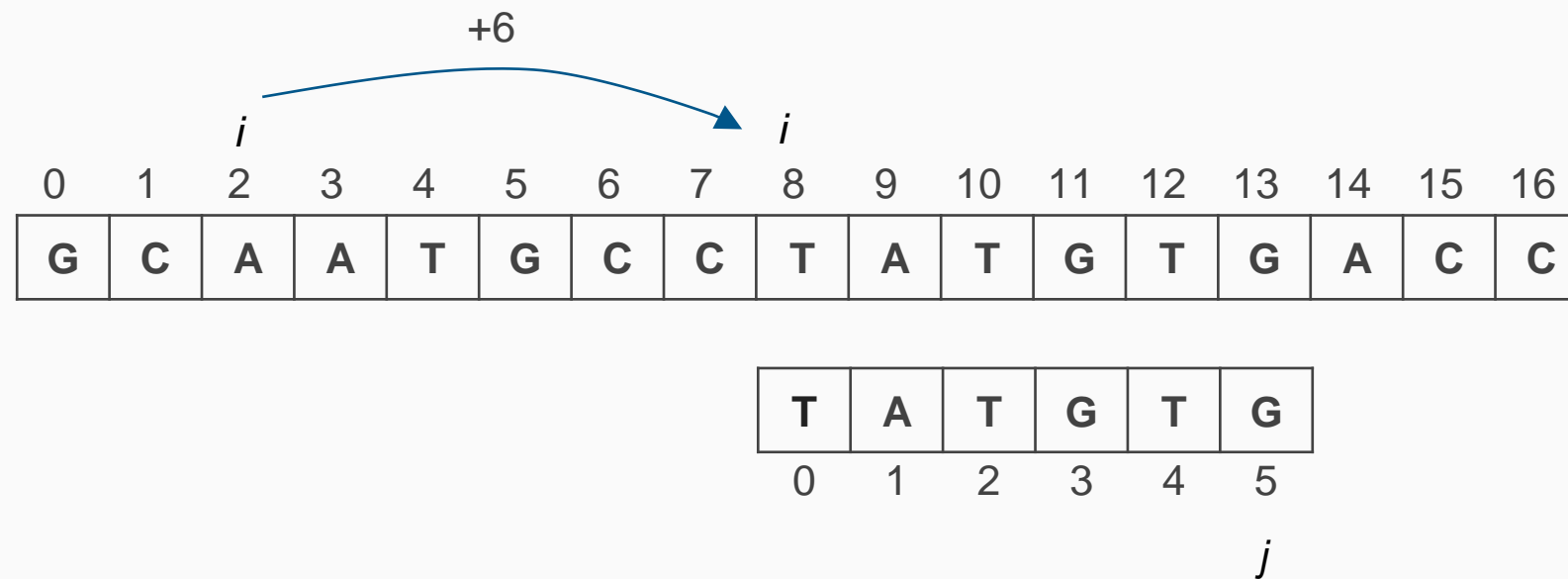


positions

A	1
B	-1
..	-1
G	5
...	-1
T	4
...	-1

String Matching (Algoritmo Boyer-Moore)

- Ejemplo



String Matching (Algoritmo Boyer-Moore)

StringMatching(Text, Pattern)

```
n = Text.size()
m = Pattern.size()
positions = buildPositions(Pattern)
results = []
for(i=0; i<n-m; )
    j=m-1
    while(Pattern[j] == Text[i+j] && j >= 0)
        j--
    if (j > 0)
        i = i + (j - positions[Text[i+j]])
    else
        results.add(i)
        i = i + m
return results
```

¿Complejidad?

$O(|\Sigma|) + O(n)$

TRIES

TDA Diccionario

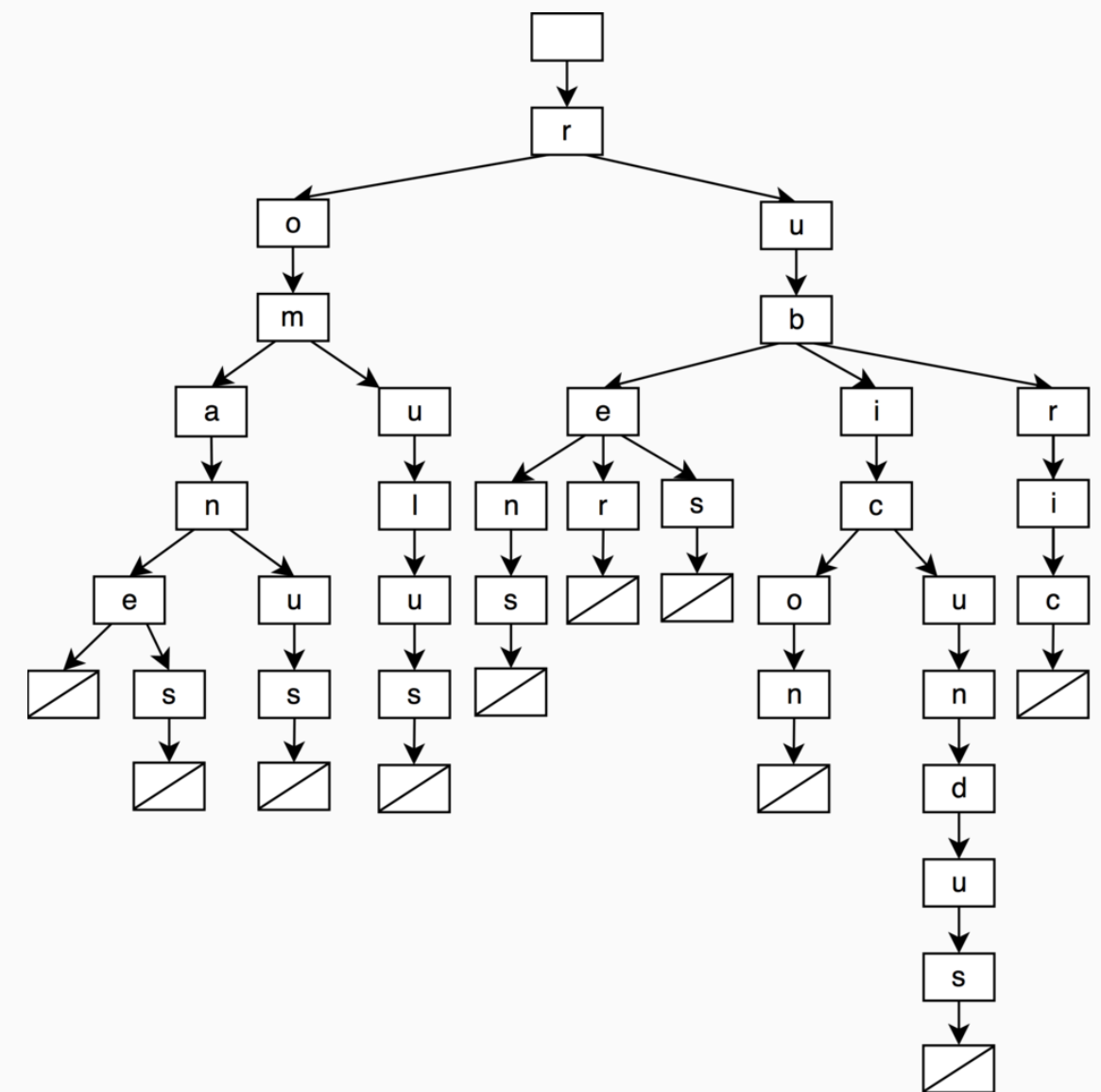
Tries

Los árboles de prefijos (*efficient information reTRIEval data structure*), es un tipo de árbol que es usualmente usado para almacenar caracteres. **Normalmente, un trie representa un diccionario de palabras**

A pesar de que cada nodo es utilizado para almacenar caracteres (también las aristas podrían representar los caracteres), los caminos entre estos representan palabras o partes de palabras. No olvidar el carácter de fin de cadena

Qué usos se les ocurren para las estructuras de datos que almacenan strings?

Autocompletar, editores de texto, procesadores de texto, etc



Tries (insertar)

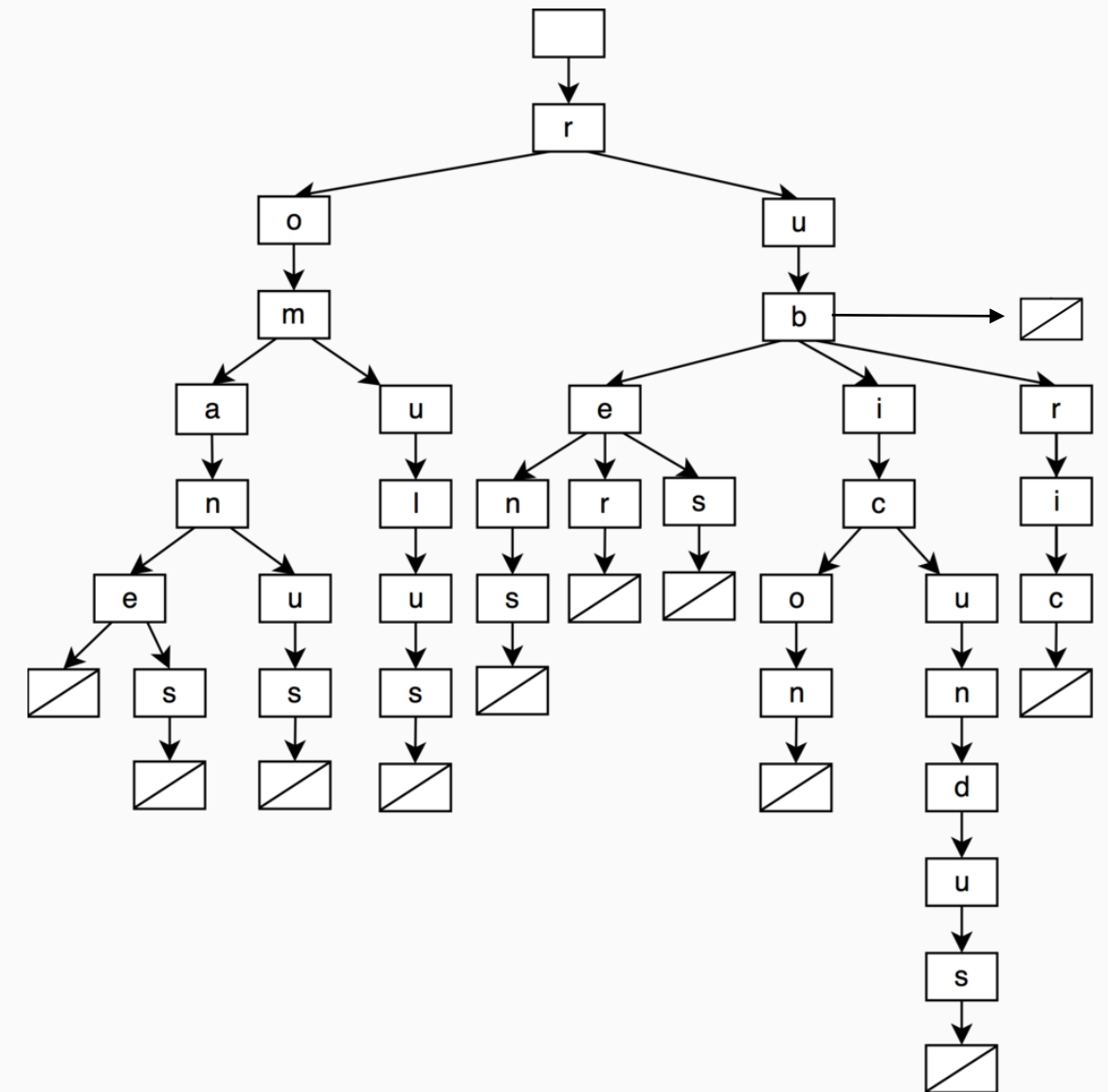
Cómo agregar una nueva palabra? $O(\text{WORD SIZE})$

1. Buscar el camino del string
2. Si se encuentra un null pointer, entonces se crea un nuevo nodo
3. Se adiciona una secuencia de hijos como letras faltan de la palabra, finalizando con el fin de cadena

En el diccionario actual tenemos palabras como:

ROMANE, ROMANES, ROMANUS, ROMULUS, RUBENS, RUBES, RUBER, RUBICON, RUBICUNDUS y RUBRIC

Cómo agregar RUB?



Tries

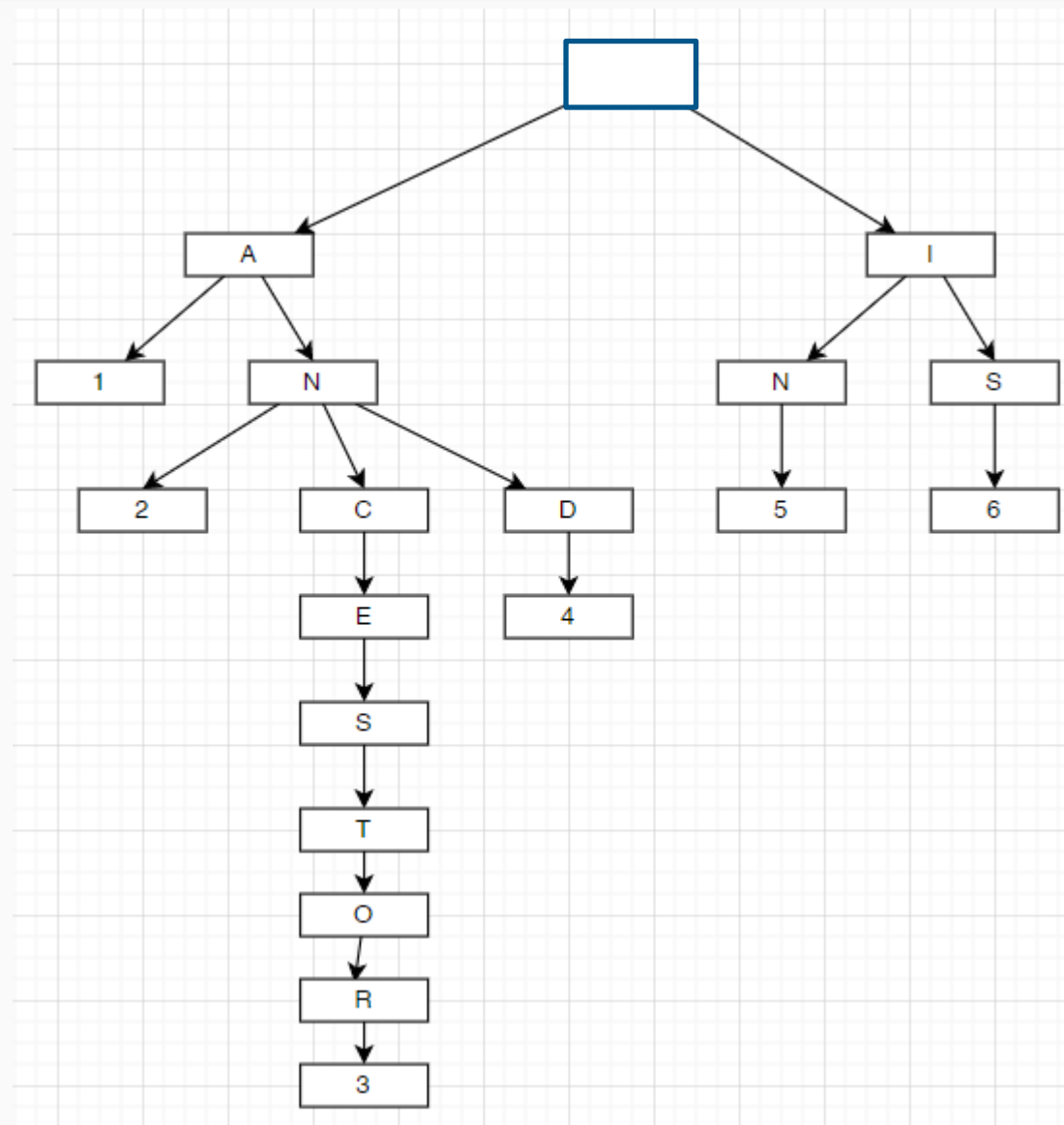
Entonces creemos el siguiente diccionario:

A
AN
ANCESTOR
AND
IN
IS
TAVERN
THERE
THE
TOWN

Tries

Entonces creamos el siguiente diccionario:

A
AN
ANCESTOR
AND
IN
IS
TAVERN
THERE
THE
TOWN

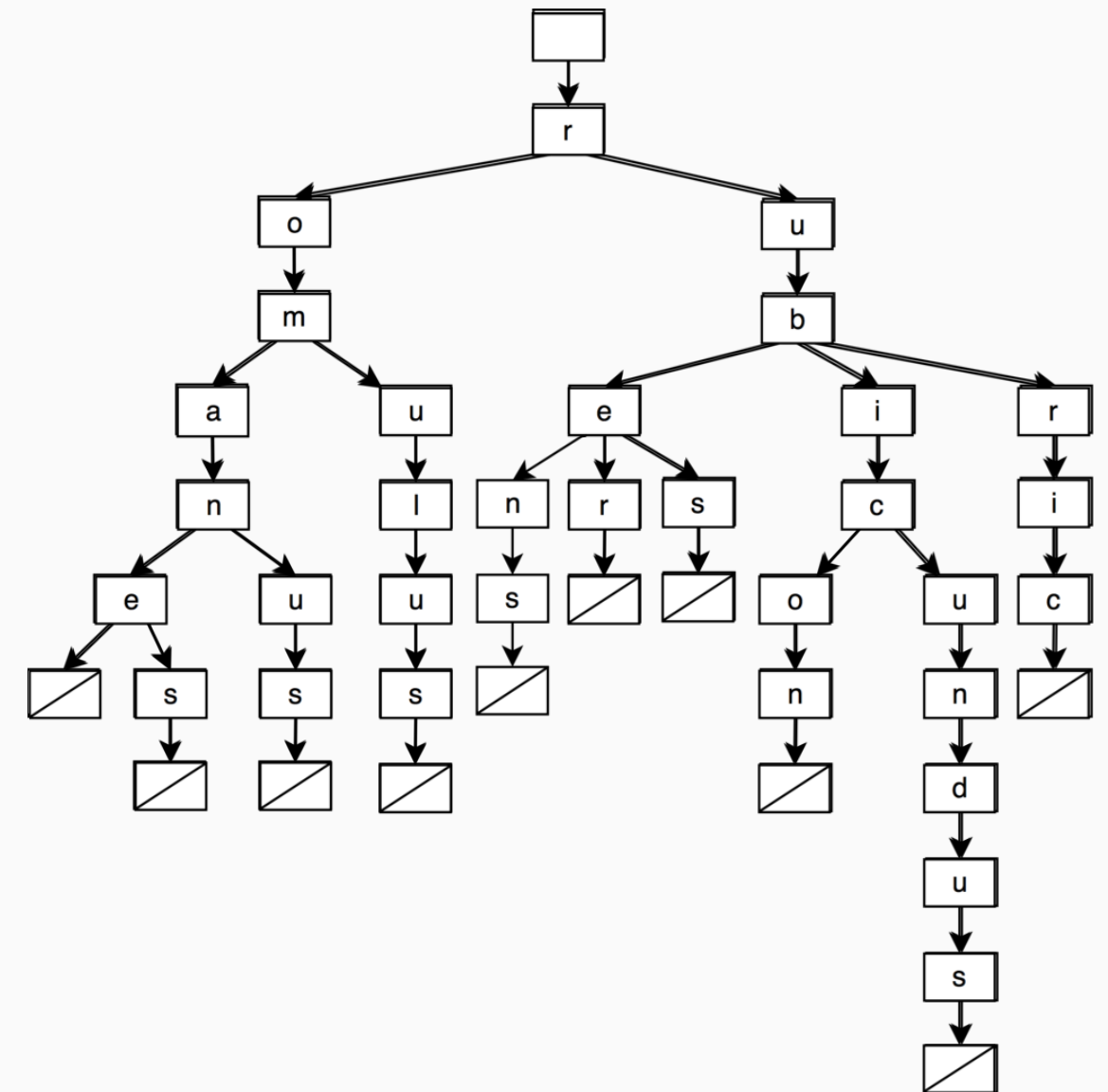


Tries (eliminar)

Cómo eliminar una palabra? $O(\text{WORD SIZE})$

1. Se busca la hoja "s" que represente nuestra palabra
2. Empezamos a eliminar los nodos desde "s" hacia el root hasta encontrar un nodo que tenga más de un hijo

Cómo eliminar RUBENS?

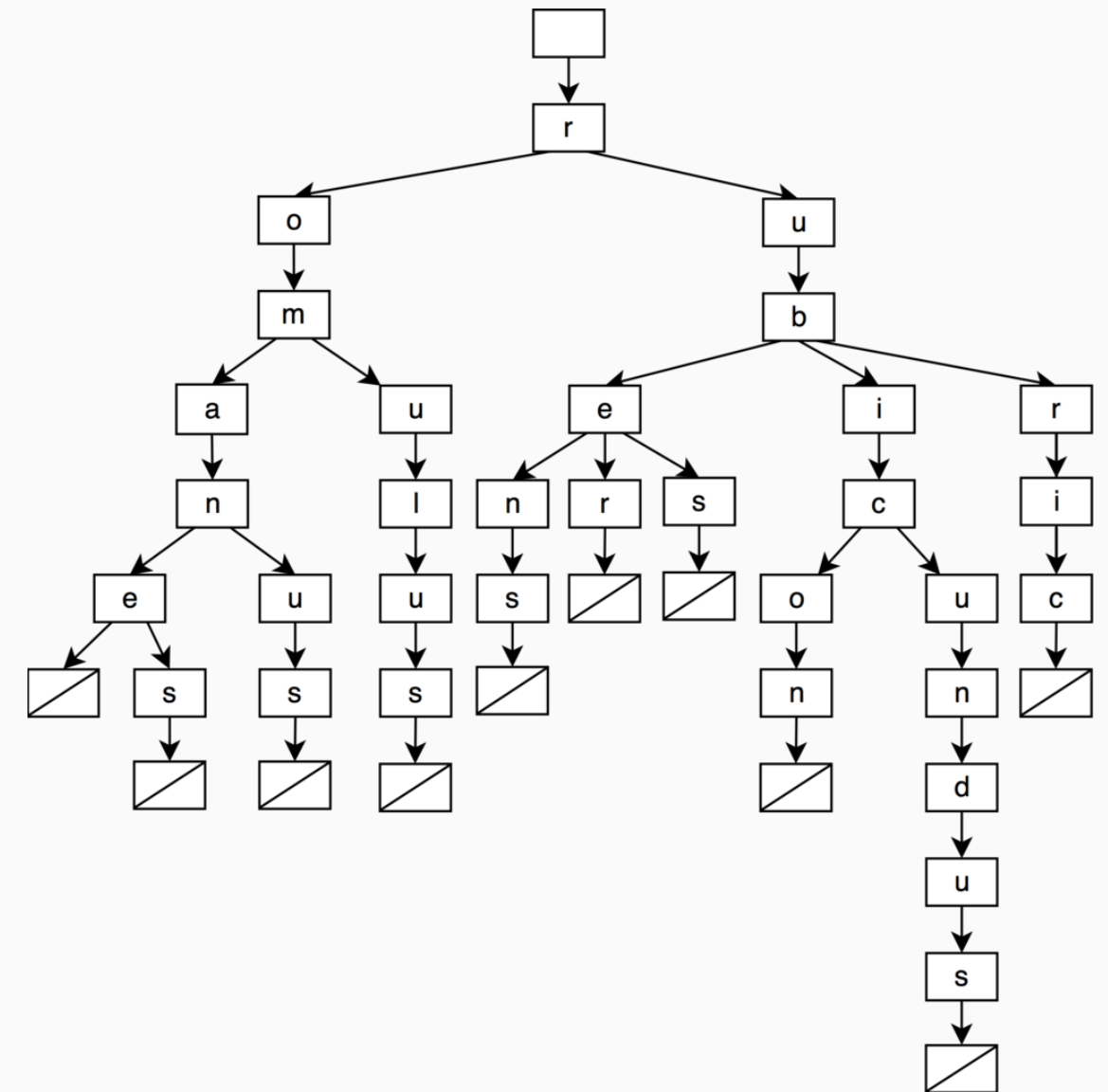


Tries (buscar)

Cómo buscar una palabra? $O(\text{WORD SIZE})$

1. Se va buscando cada carácter de la palabra en el árbol
2. Si llegamos a una nodo hoja que represente nuestra palabra, retornamos verdadero
3. De otra manera, la palabra no existe en el trie.

Buscar ROMANE?

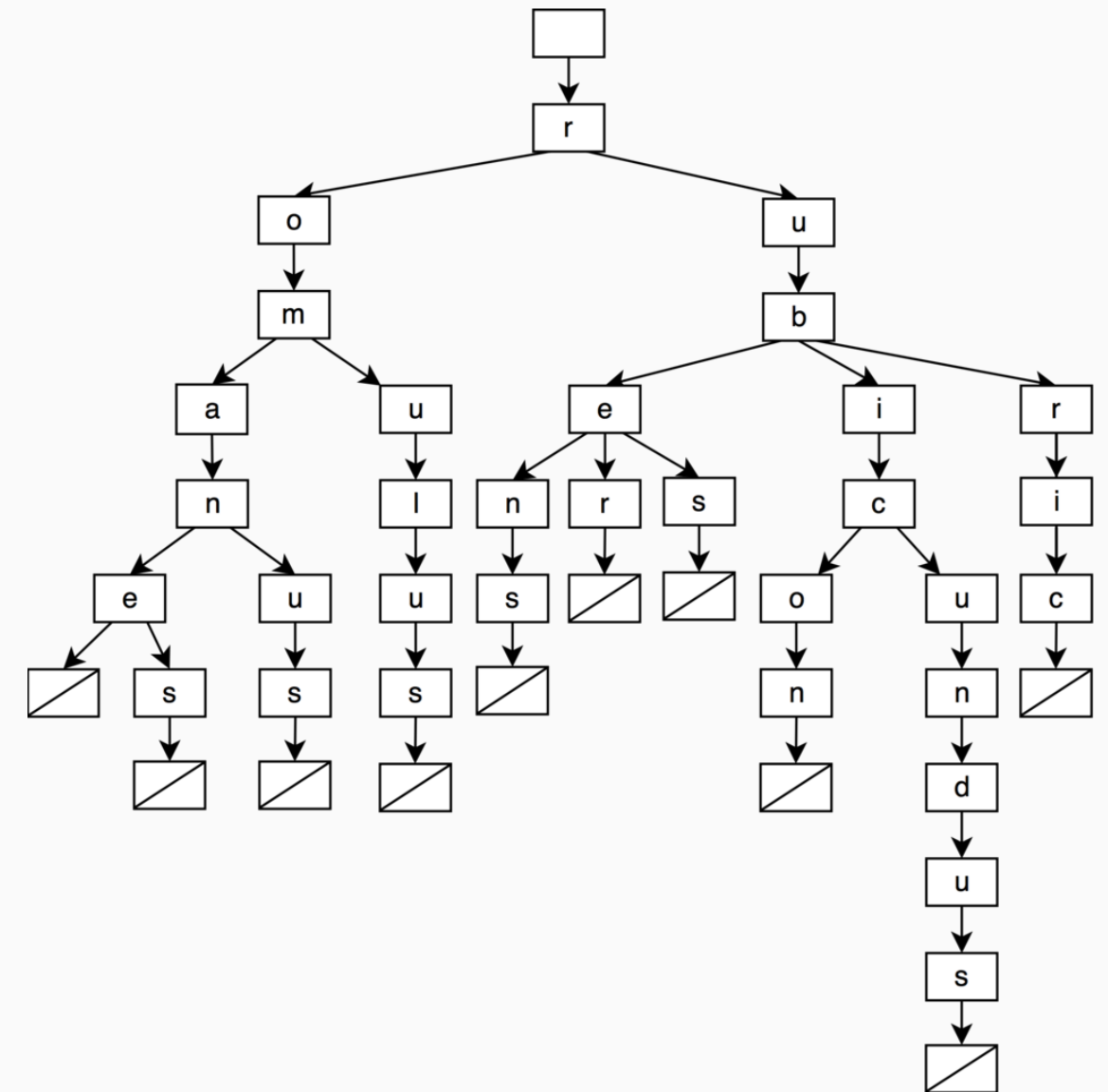


Tries (buscar)

Cómo buscar una palabra? $O(\text{WORD SIZE})$

1. Se va buscando cada carácter de la palabra en el árbol
2. Si llegamos a una nodo hoja que represente nuestra palabra, retornamos verdadero
3. De otra manera, la palabra no existe en el trie.

Buscar ROMANE?

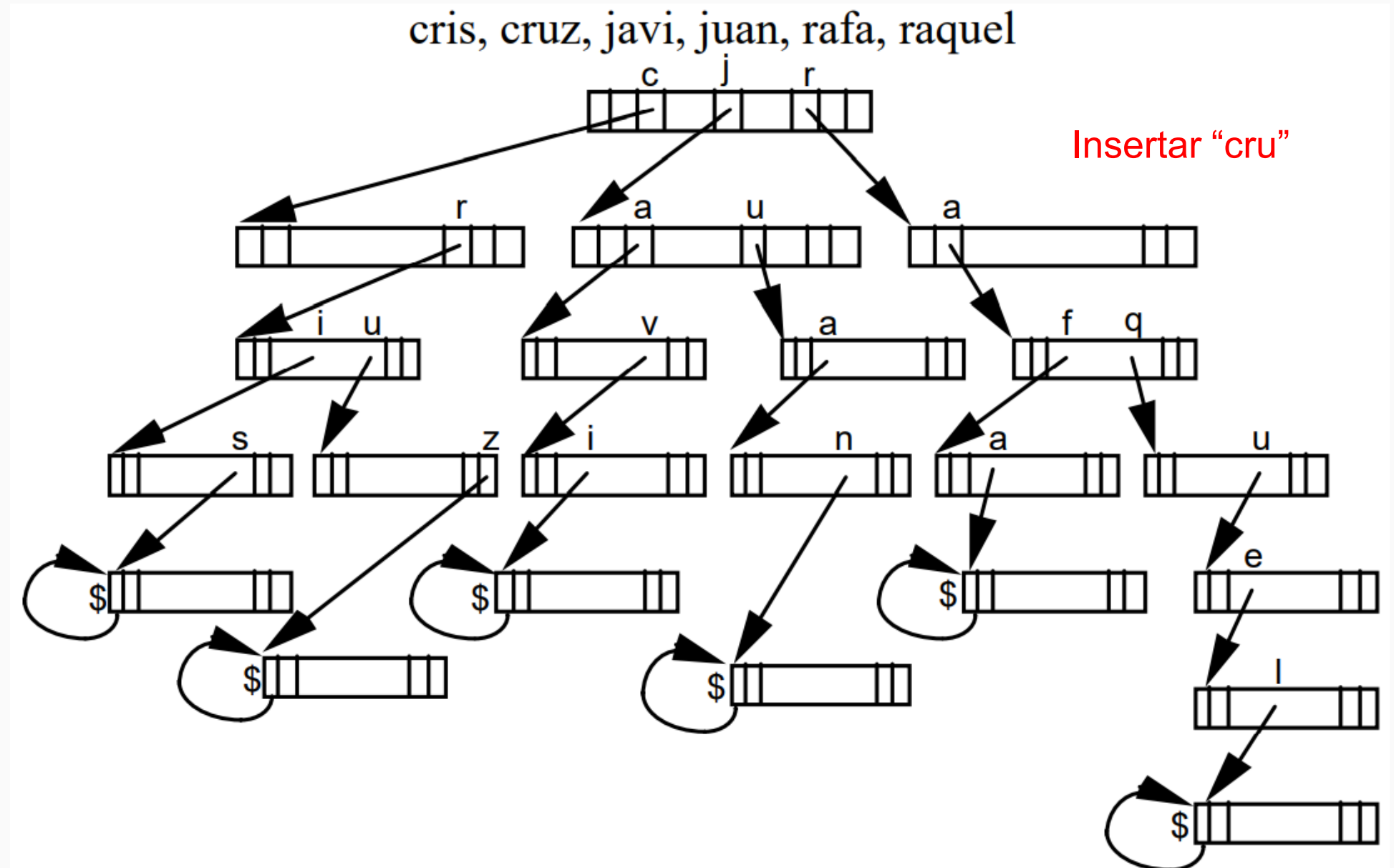


Tries (Ideas de Implementación)

1- Cada nodo como array

¿Problema?

Demasiado espacio



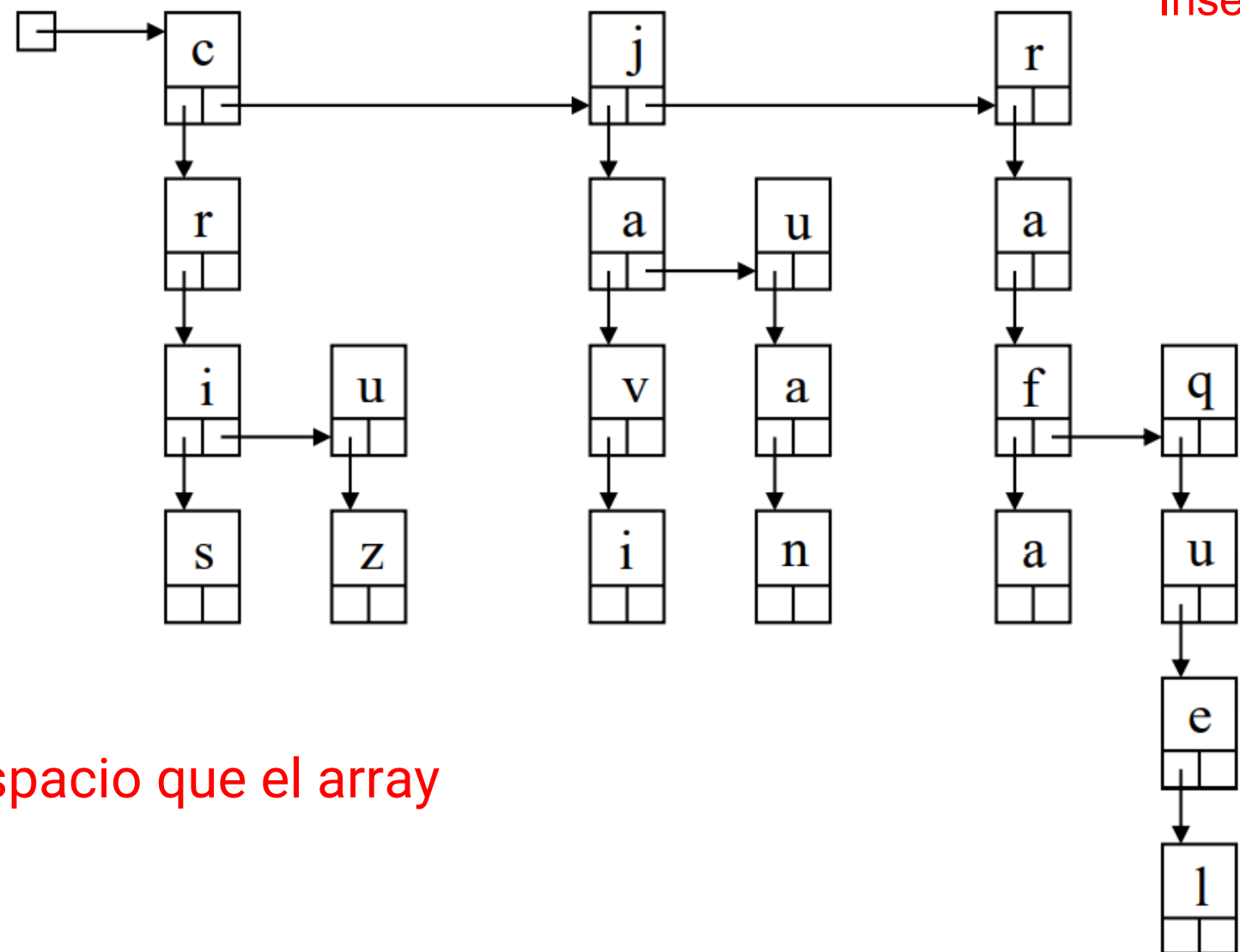
Tries (Ideas de Implementación)

2- Cada nodo como lista

¿Problema?

Mayor tiempo de búsqueda pero ocupa menos espacio que el array

cris, cruz, javi, juan, rafa, raquel



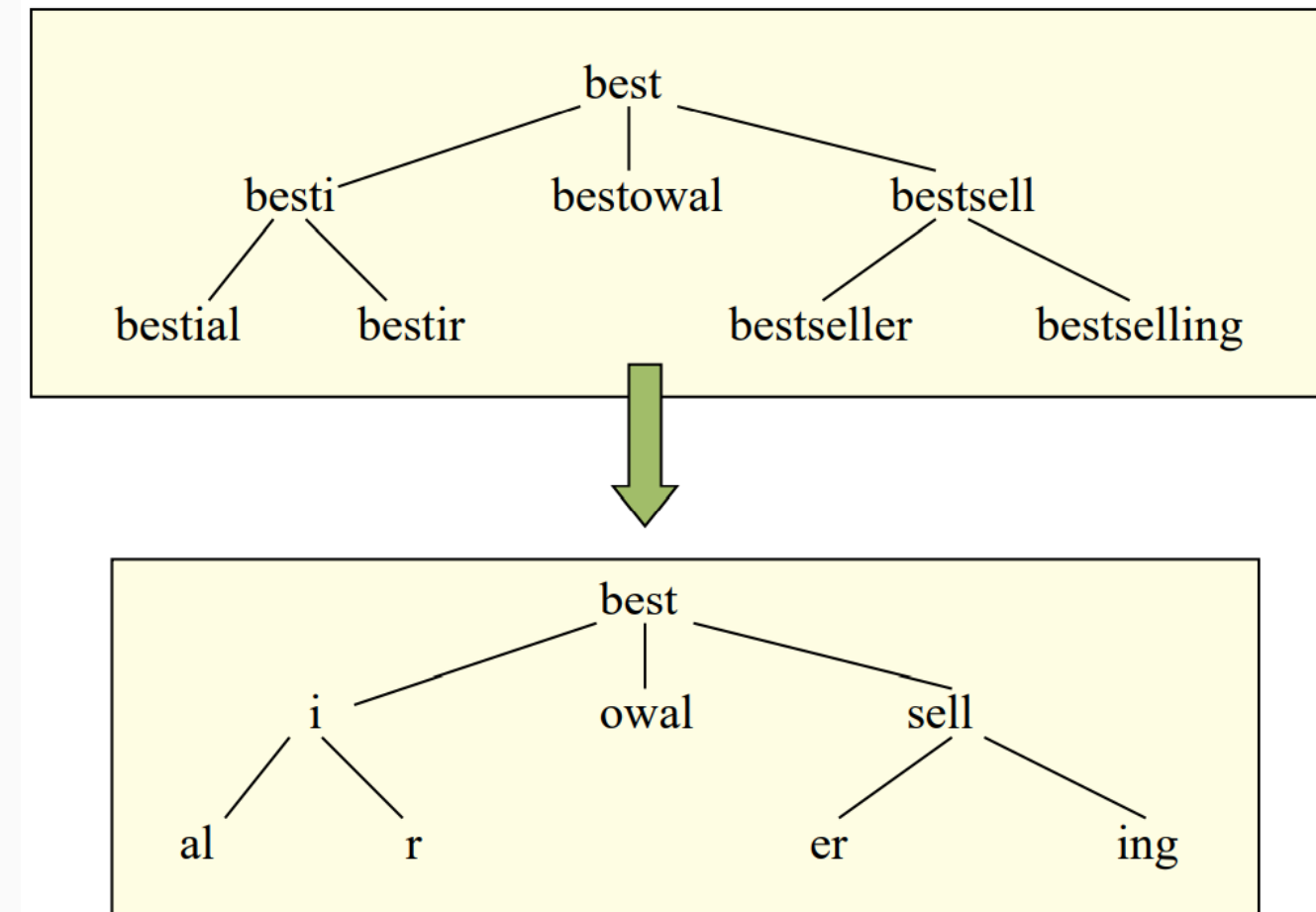
Patricia Tries (Compact Trie)

Es un tipo de trie que optimiza el espacio utilizado para representar el diccionario

El problema con los tries es que su conjunto de keys tiende a ser muy disperso. En muchos de los casos un nodo interno termina teniendo solo un descendiente

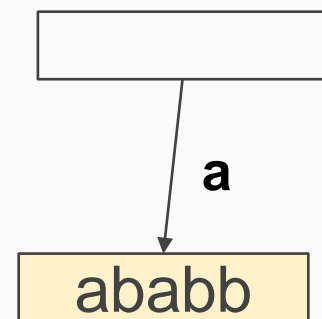
El problema anterior causa que los tries tengan una complejidad alta en términos de espacio

En los Patricia Tries, en vez de almacenar un solo carácter en un nodo, se almacena la mayor cantidad de caracteres posibles

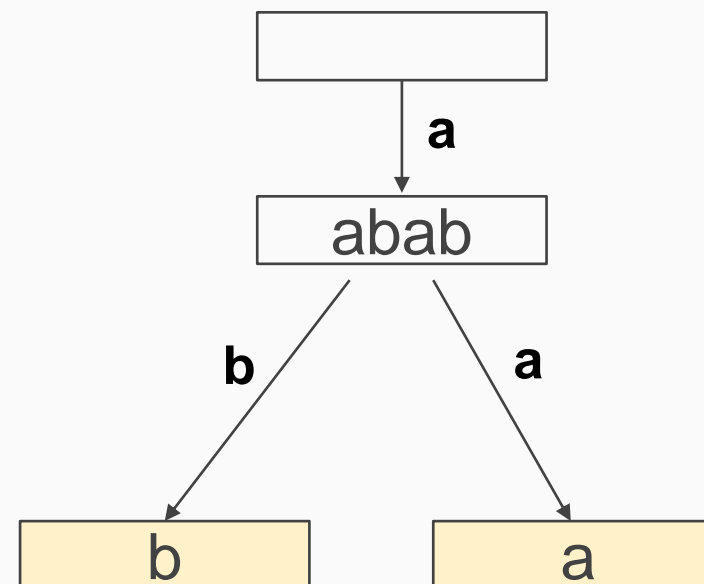


Patricia Tries (Insertar)

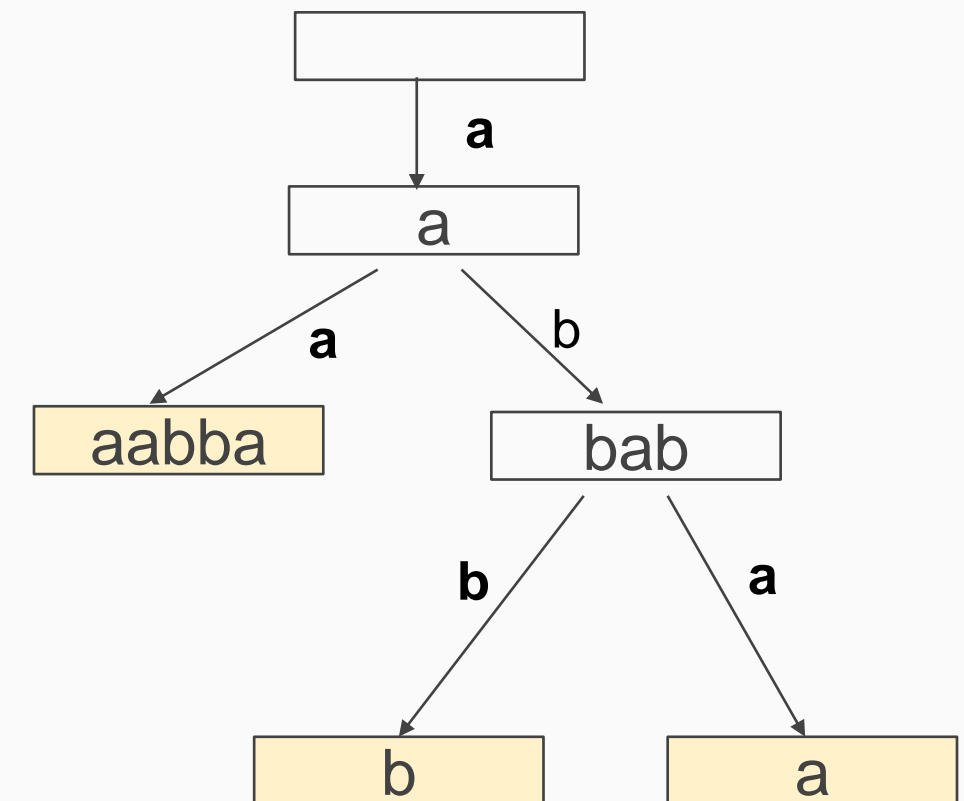
1) insertar "ababb"



2) insertar "ababa"

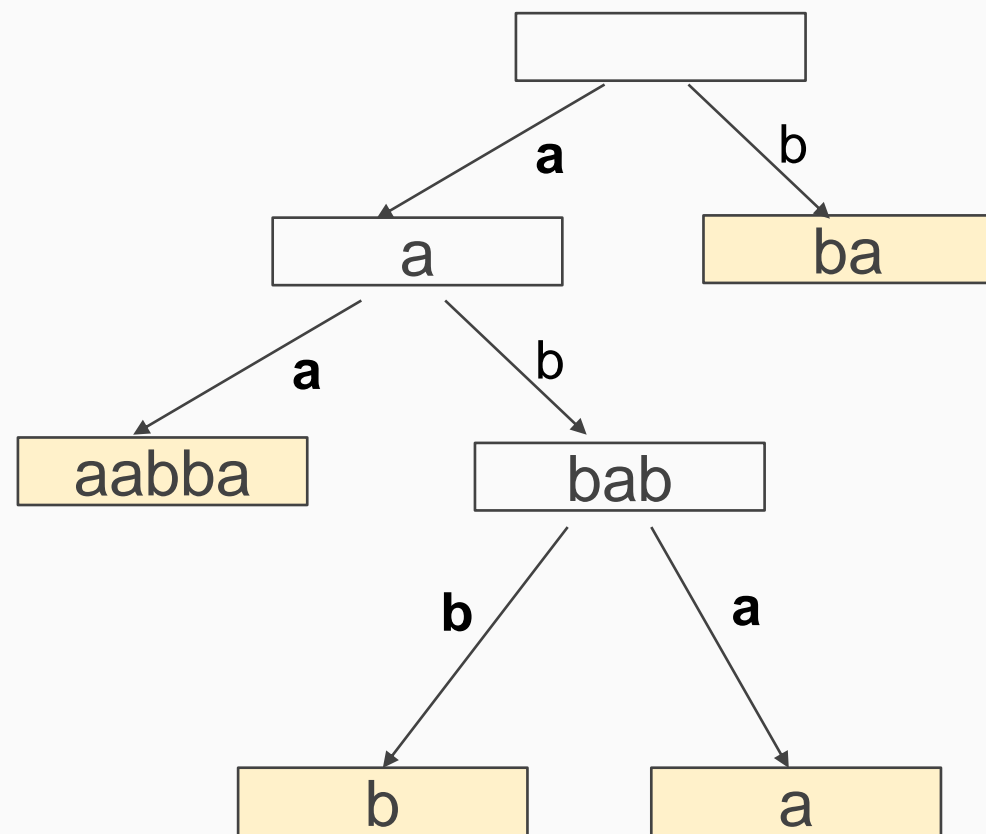


3) insertar "aaabba"

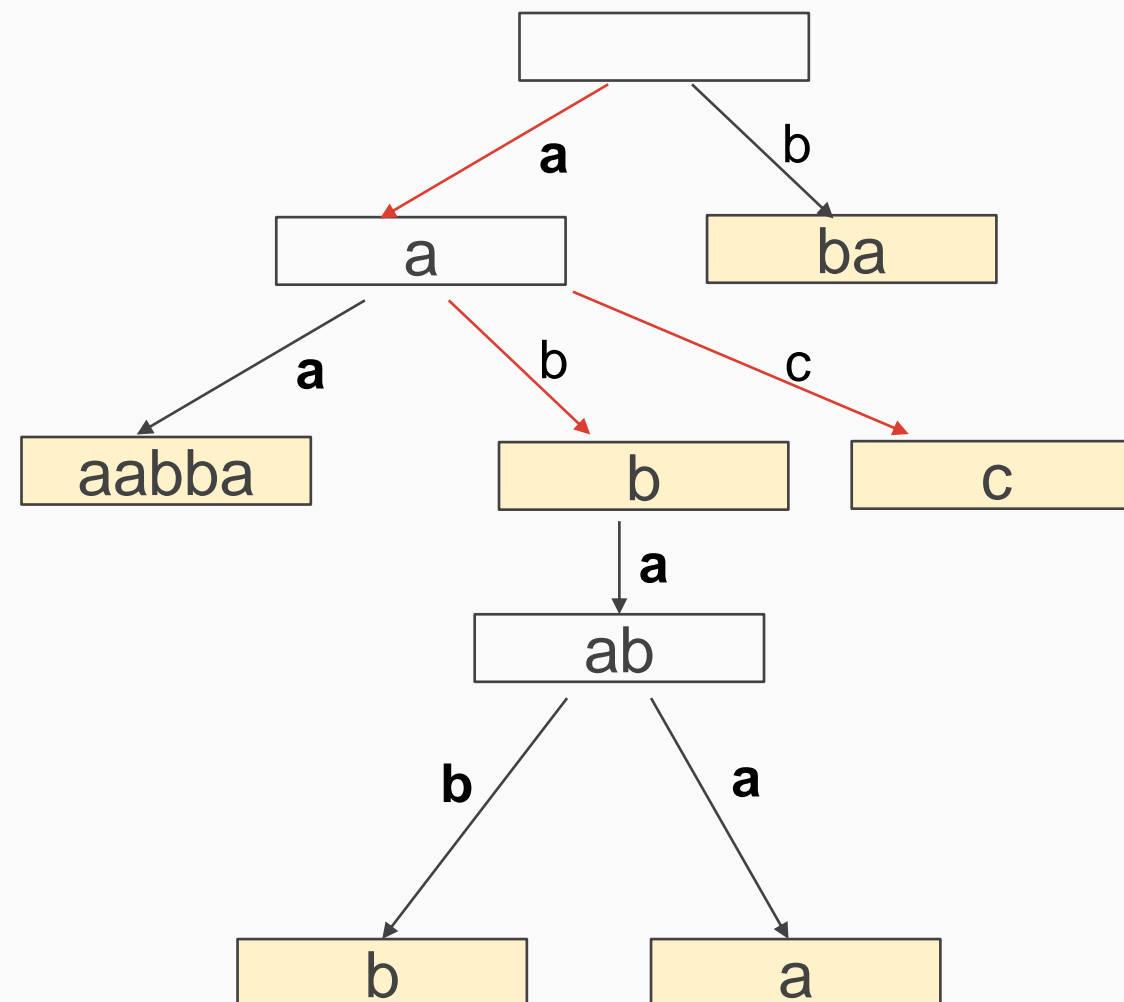


Patricia Tries (Insertar)

3) insertar "ba"



4) insertar "ab" y "ac"



Patricia Tries (Insertar)

Crear el siguiente diccionario:

TEST

TOASTER

TOASTED

TASTING

SLOW

SLOWLY

TESTIMONY

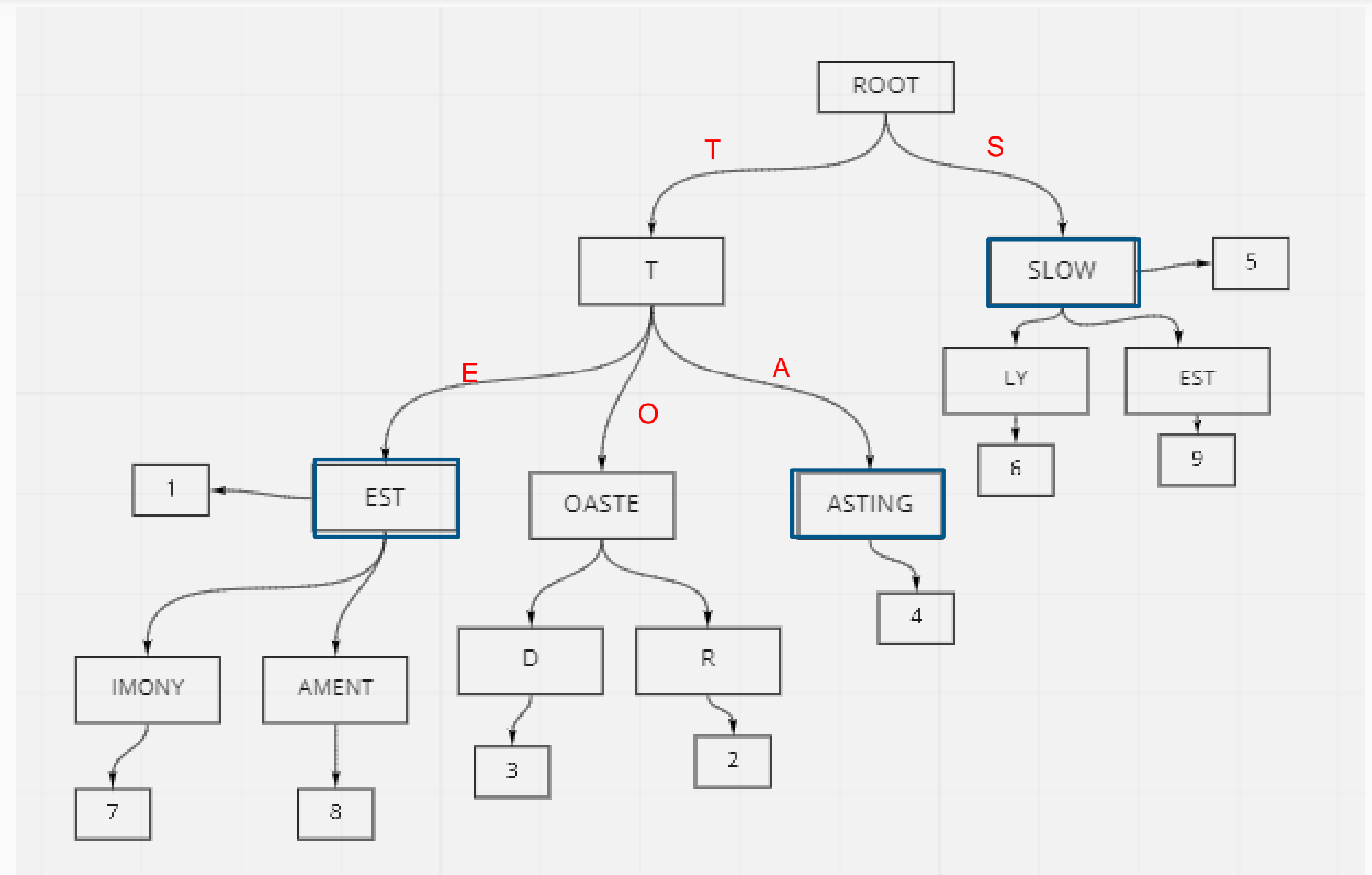
TESTAMENT

SLOWEST

Patricia Tries (Insertar)

Crear el siguiente diccionario:

TEST
TOASTER
TOASTED
TASTING
SLOW
SLOWLY
TESTIMONY
TESTAMENT
SLOWEST



Patricia Tries (Buscar y Eliminar)

¿Cómo sería la búsqueda?

Complejidad

¿Cómo sería la eliminación?

Complejidad

Patricia Tries (Idea de Implementación)

¿Cómo sería la estructura de datos?

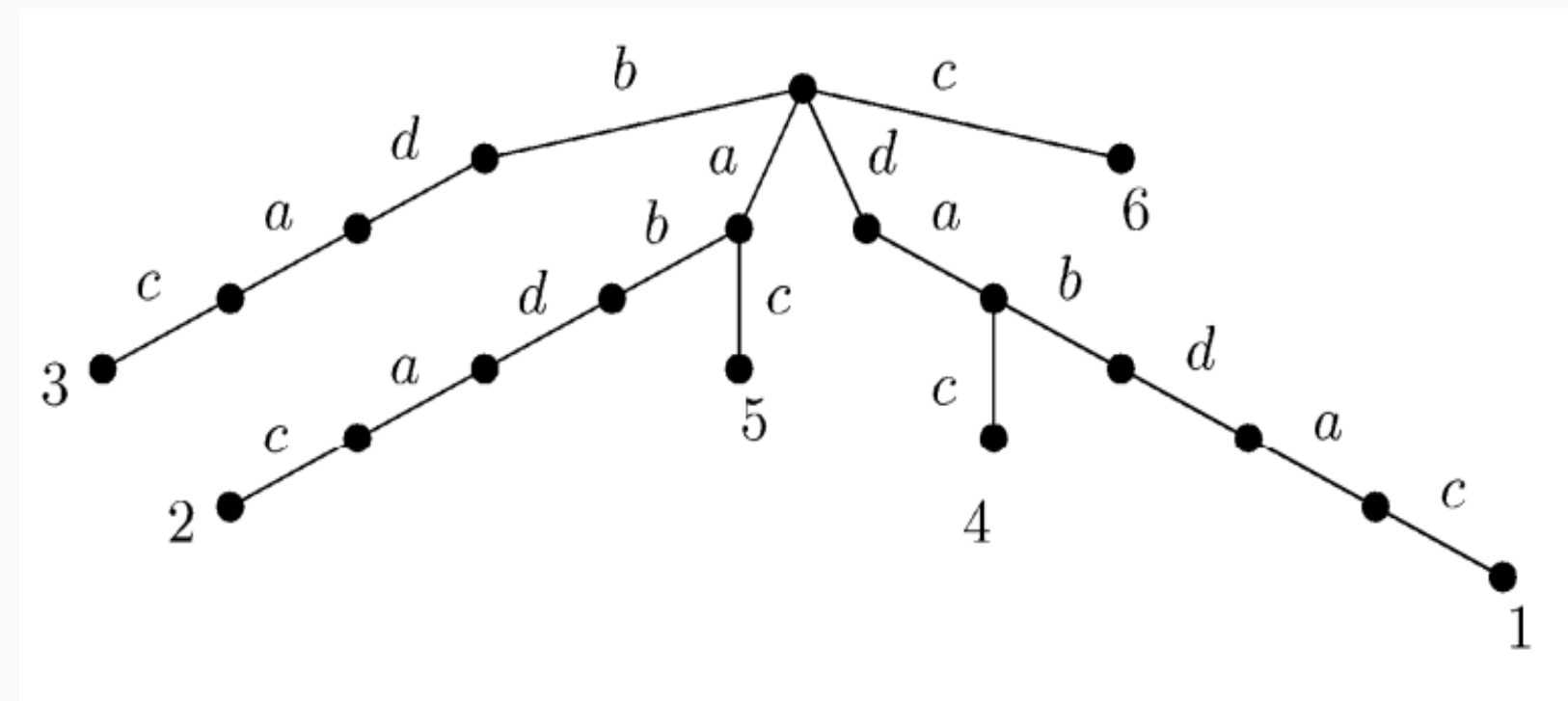
Suffix Tree

String Matching

String Matching (Suffix Tree)

Estructura eficientemente todos los sufijos del texto en un Trie

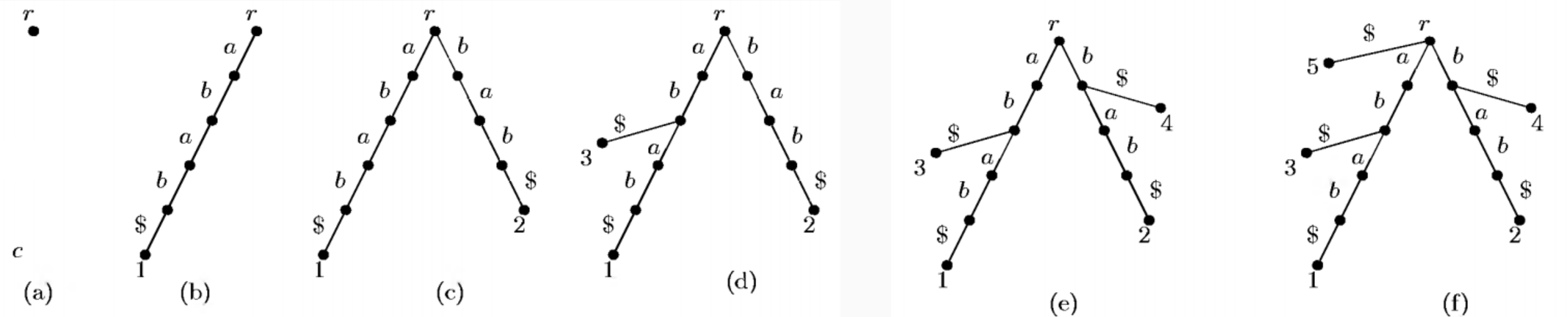
- Esto es útil cuando se quieren encontrar muchos patrones en el mismo texto (por ejemplo, muchos genes en el mismo DNA)
- El patrón P ocurre en T cuando P es el prefijo de algún sufijo de T
- Ejemplo, sea el suffix tree del texto $T = \text{dabdac}$



Como buscar el patrón $P = \text{"ac"}$, $P = \text{"bd"}$ y $P = \text{"da"}$

String Matching (Suffix Tree, Algoritmo)

Construcción del suffix tree para T="abab"



String Matching (Suffix Tree, Algoritmo)

BuildSuffixTree(Texto T)

- crear arbol vacio con solo raiz
 - for $i = 1 \dots n$
 - sufijo = t_i, t_{i+1}, \dots, t_n
 - empezar desde la raiz buscando un camino desde t_i hasta t_j
 - (conincidencia del prefijo en algun sufijo)
 - sea el nodo X de t_j
 - añadir al arbol los nodos restantes: t_{j+1} hasta t_n
 - agregar posicion como etiqueta del nodo hoja
- retornar el arbol

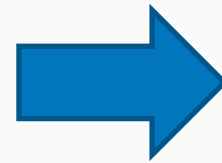
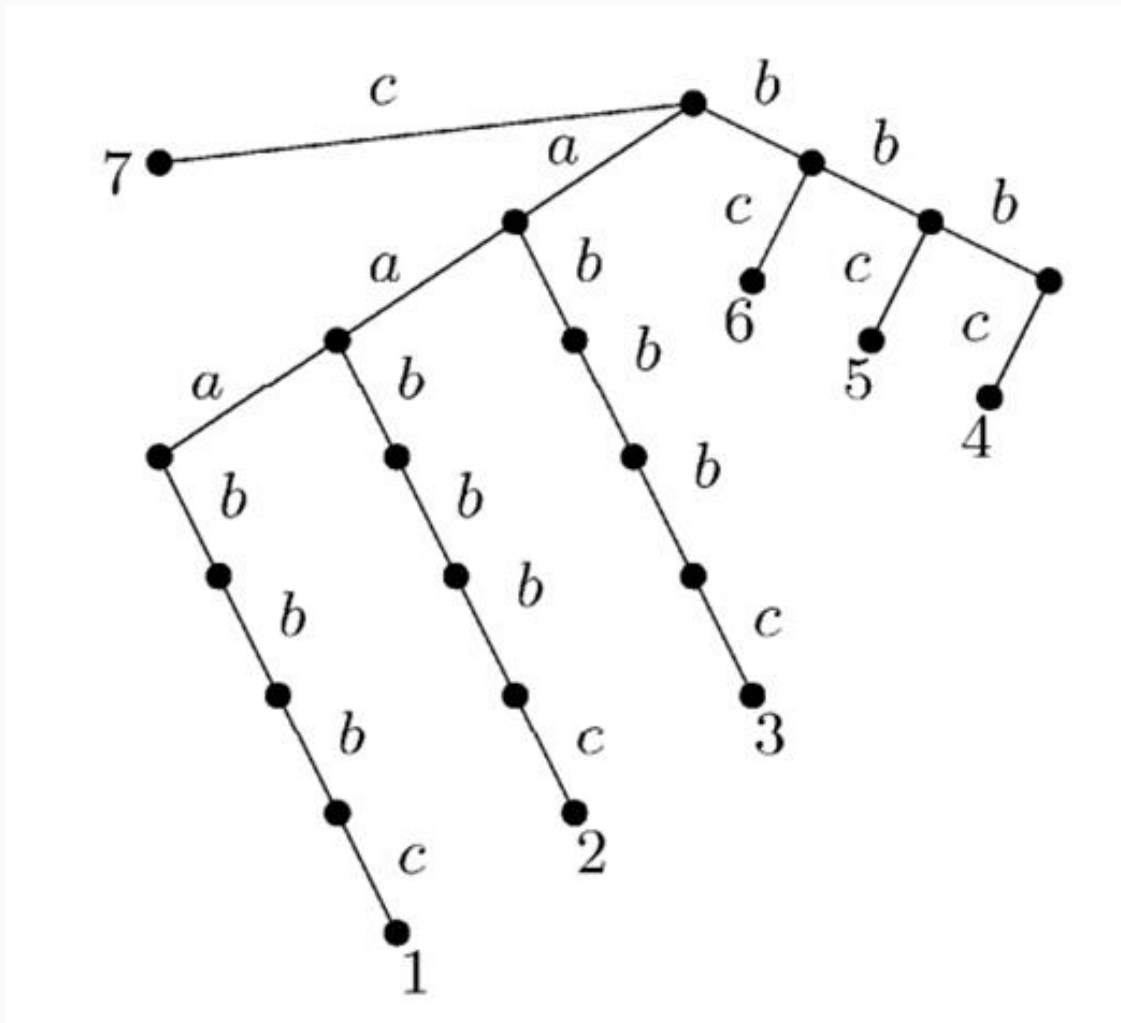
¿Cómo sería el nodo?

¿Cómo sería la búsqueda?

¿Complejidad?

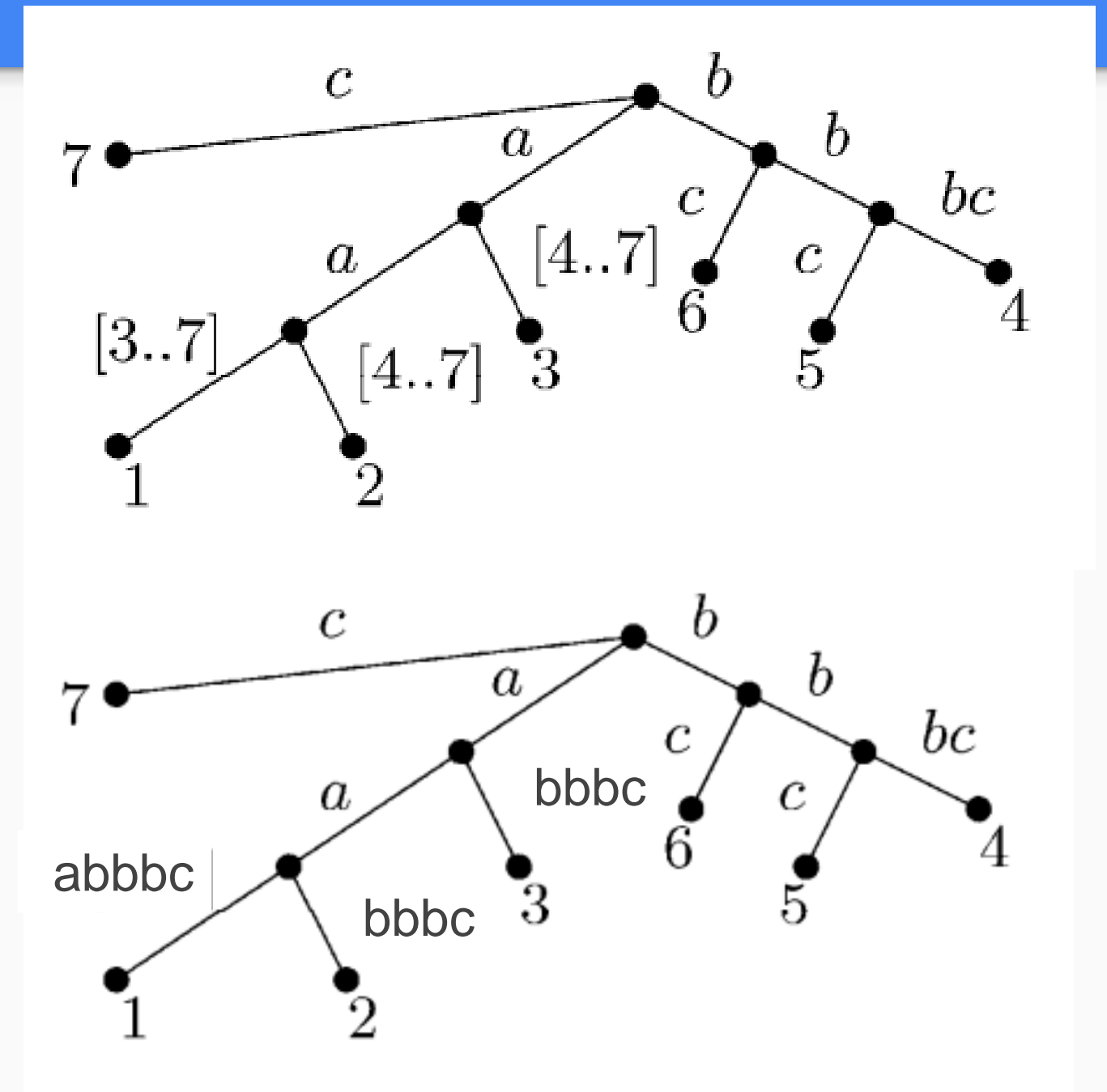
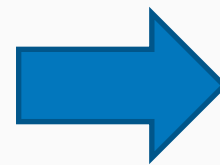
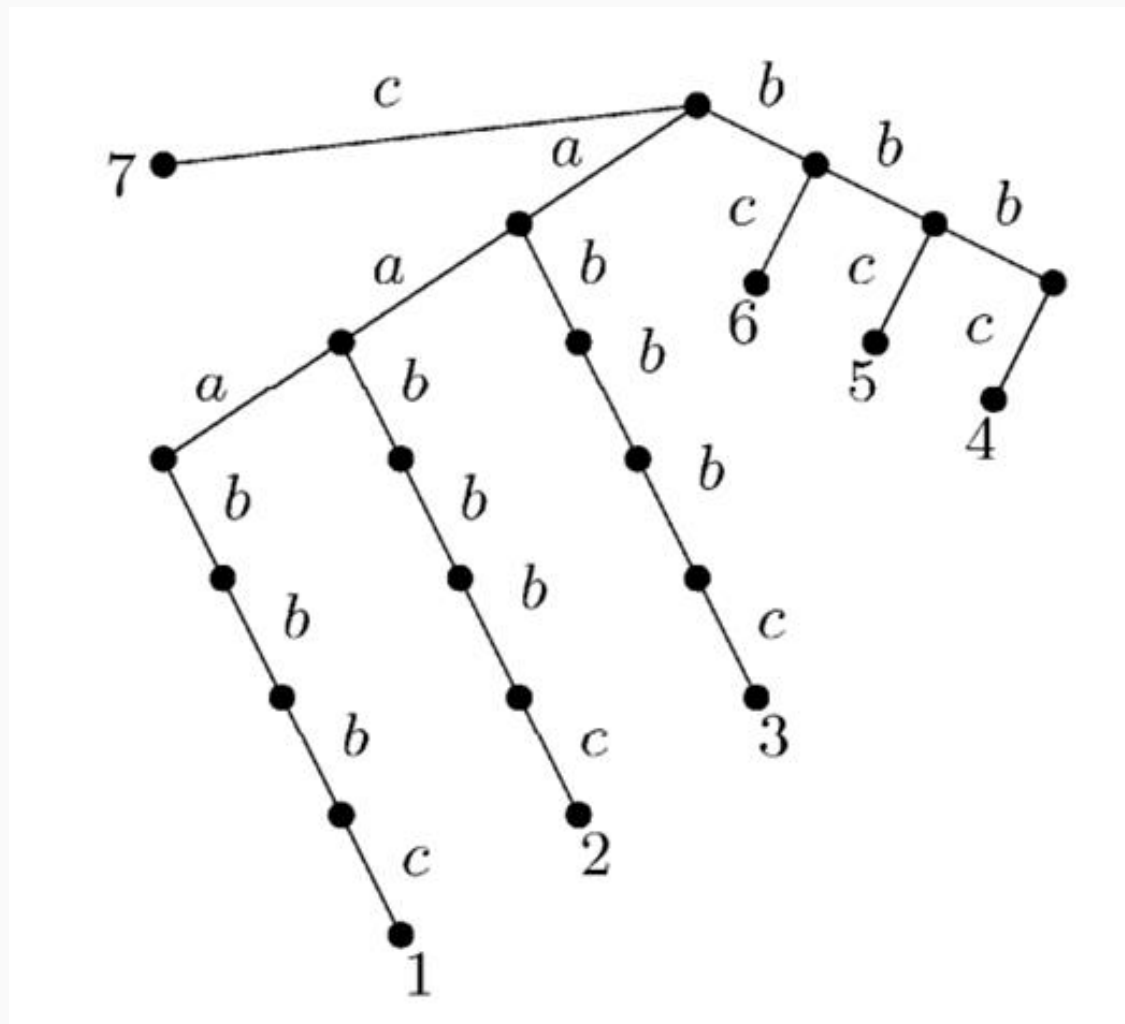
String Matching (Suffix Tree, Algoritmo)

Optimización: eliminar nodos con un solo hijo



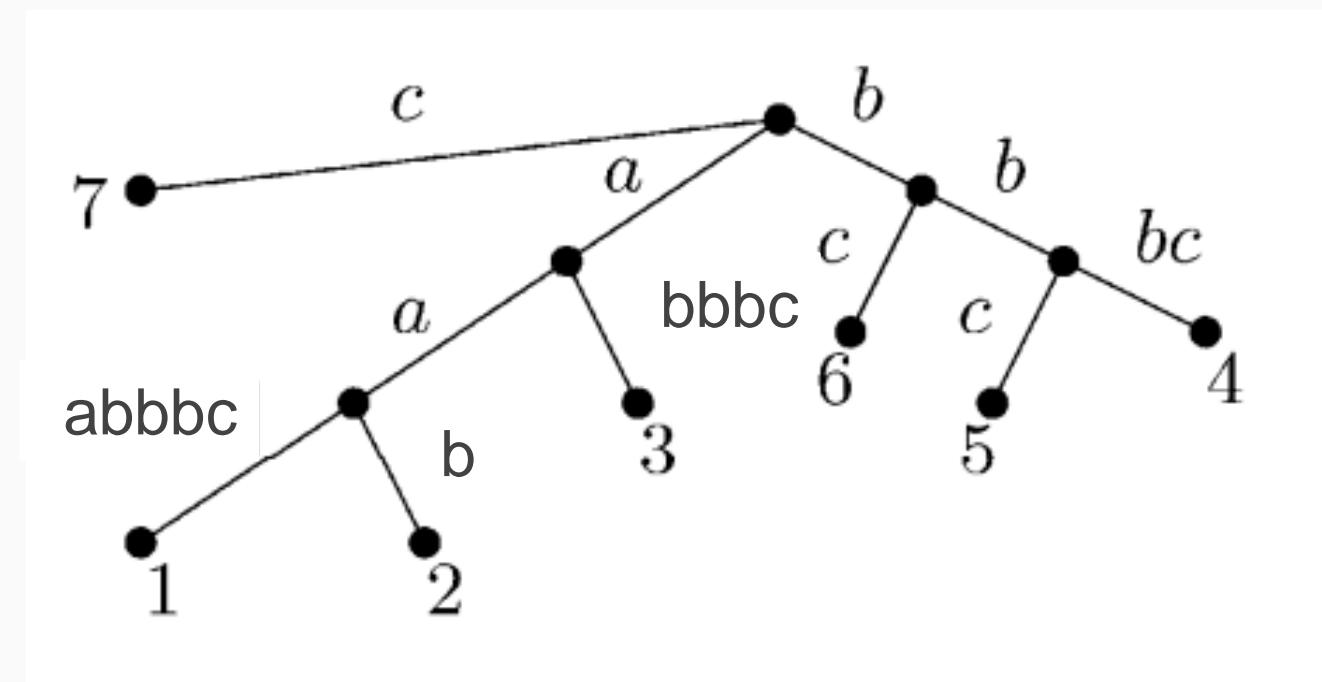
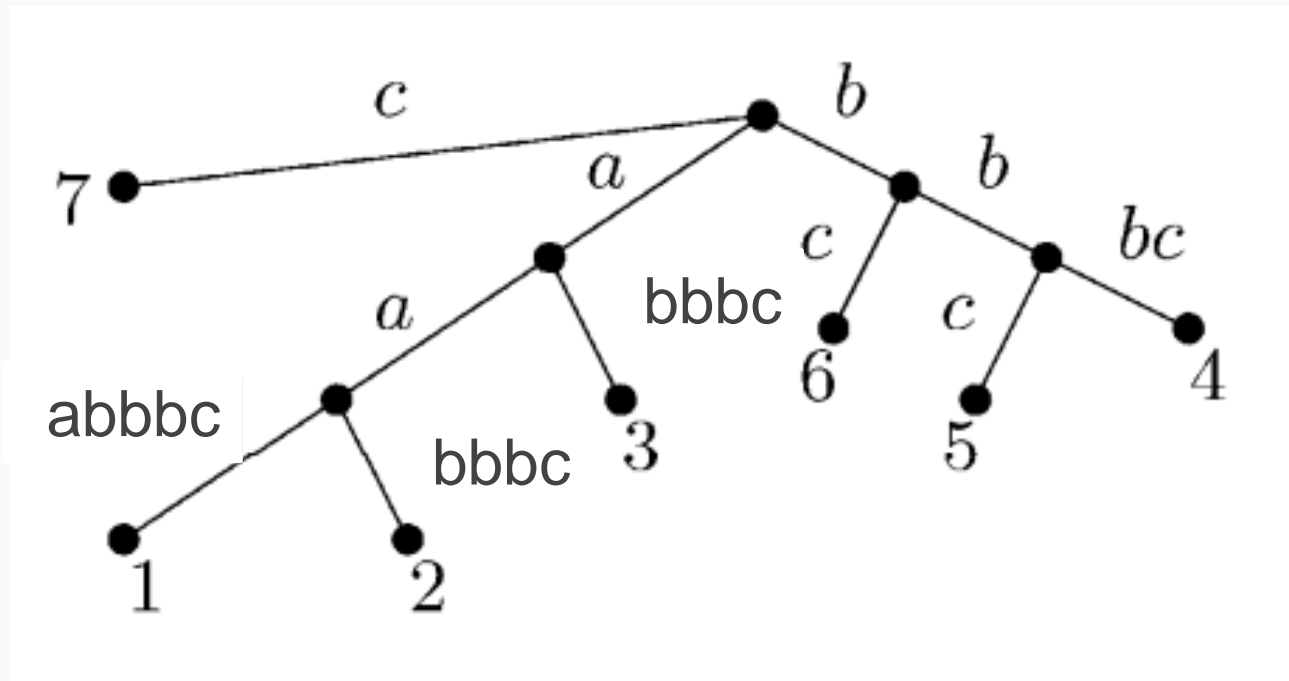
String Matching (Suffix Tree, Algoritmo)

Optimización: eliminar nodos con un solo hijo



String Matching (Suffix Tree, Algoritmo)

Optimización: eliminar nodos con un solo hijo



¿Insertar un nuevo sufijo?

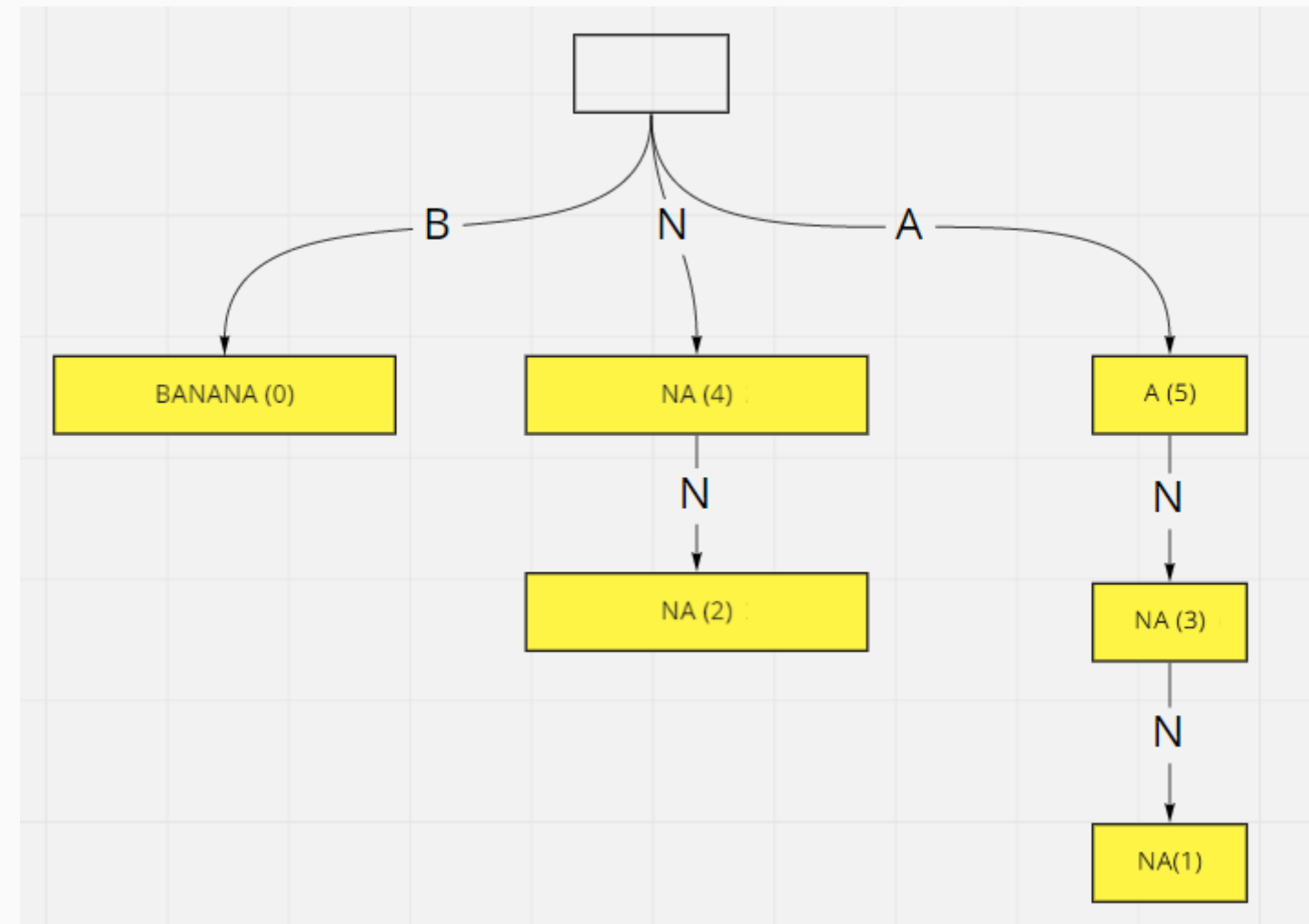
¿Eliminar un sufijo?

String Matching (Suffix Tree, Algoritmo)

Crear el diccionario para el siguiente texto "BANANA" y aplicar la búsqueda del patrón "ANA"

String Matching (Suffix Tree, Algoritmo)

Crear el siguiente diccionario para el siguiente texto
“BANANA” y aplicar la búsqueda del patrón “ANA”



String Matching (Suffix Tree, Algoritmo)

ConstruirArbolSufijosCompacto(Texto T):

¿Cómo sería la búsqueda?

¿Complejidad?

Welcome to Algorithms and Data Structures! - CS2100