TEMA 3. DICCIONARIOS. BÚSQUEDA CON TOLERANCIA

Contenidos basados en los materiales de otros cursos como los de Manning, Baeza, Jurafsky.

Contenidos

Bibliografía

A Introduction to Information Retrieval:

Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze.

Cambridge University Press, ***2009****.*

Capítulo 3

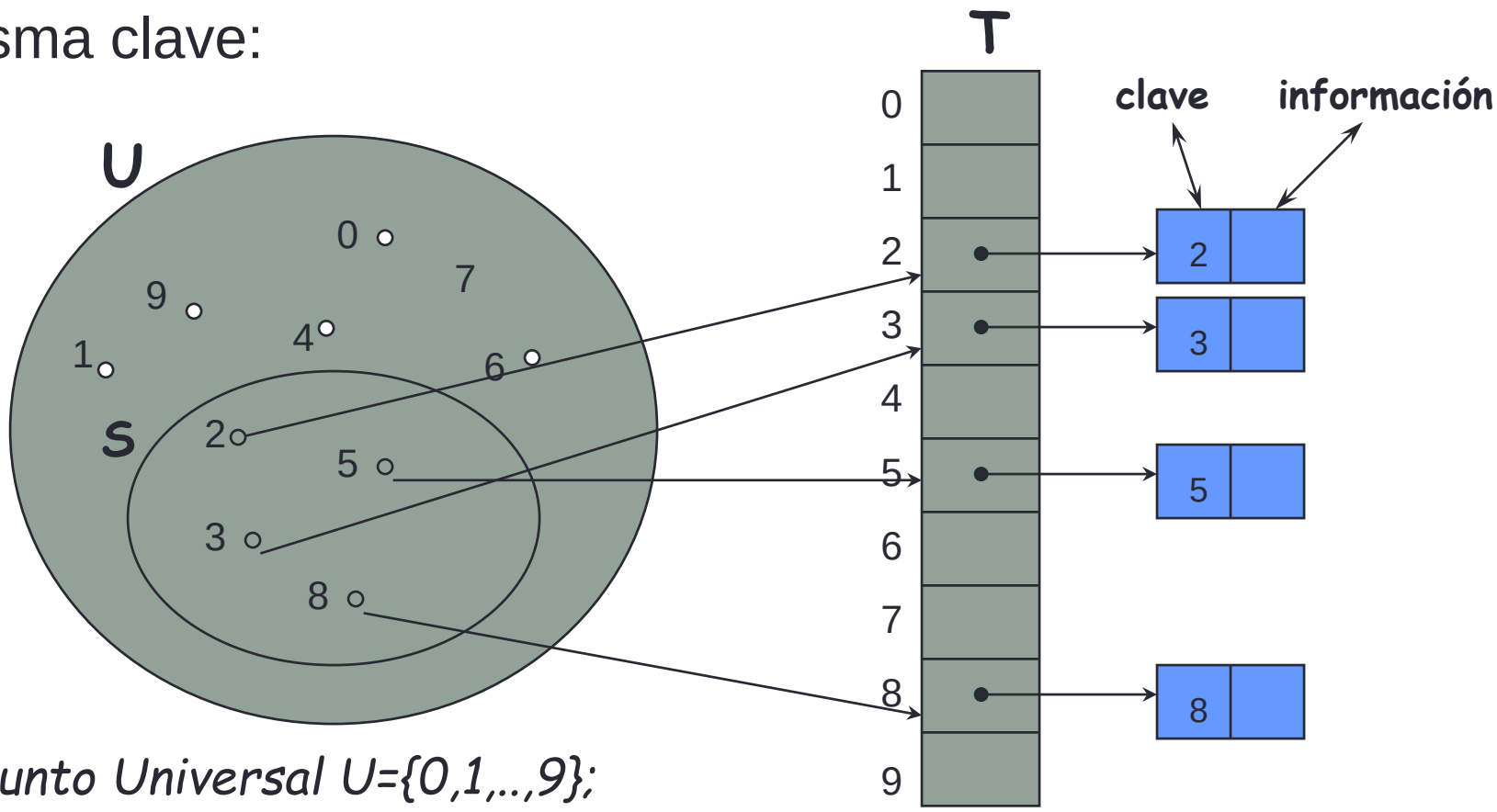
1.1 TIPOS DE DICCIONARIOS: TABLAS HASH

Tablas hash (o Tablas de dispersión)

- Estructura de datos especialmente diseñada para la implementación de DICCIONARIOS.
  - Se pueden conseguir los siguientes costes:
    - Búsqueda. O(1)
    - Inserción. O(1)
    - Borrado. O(1)
- Método: asociar una clave a cada elemento del dominio y asociarle un elemento de un vector.

Tablas hash

- Caso sencillo: el universo de valores posibles es pequeño
- Si |U|<<< y asumimos que no hay dos elementos con la misma clave:



Conjunto Universal U={0,1,..,9};
Conjunto a representar S={2,3,5,8}

Tablas hash

Caso más complejo:

Sea |U| el tamaño del conjunto universal y |S| el tamaño del conjunto a representar. El espacio en memoria para representar S es $O(|U|)$.

- Si |U|>>>> puede ser que no tengamos memoria suficiente para representar todos los elementos posibles
- Si |S|<<<<|U| se utiliza mucho espacio cuando el conjunto que hay que representar es pequeño.

Tablas hash

Para construir una Tabla Hash:

- Dividir el conjunto **U** en un número finito **B** de clases
- Usar un vector **T[0..B-1], con B<<|U|.**
- Cada elemento de **U** se identifica por una clave **k**.
- Definir una función hash que asocie a cada elemento de **U** (cada clave k) un valor entre **[0..B-1].**
- Un elemento de clave k se almacena en la posición **h(k)**.
- **T** es una **Tabla hash** y **h** es la **función hash.**
- A cada una de las **T[j]** se les llama **cubeta.**

Ejemplo de función hash: h(x) = x MOD B

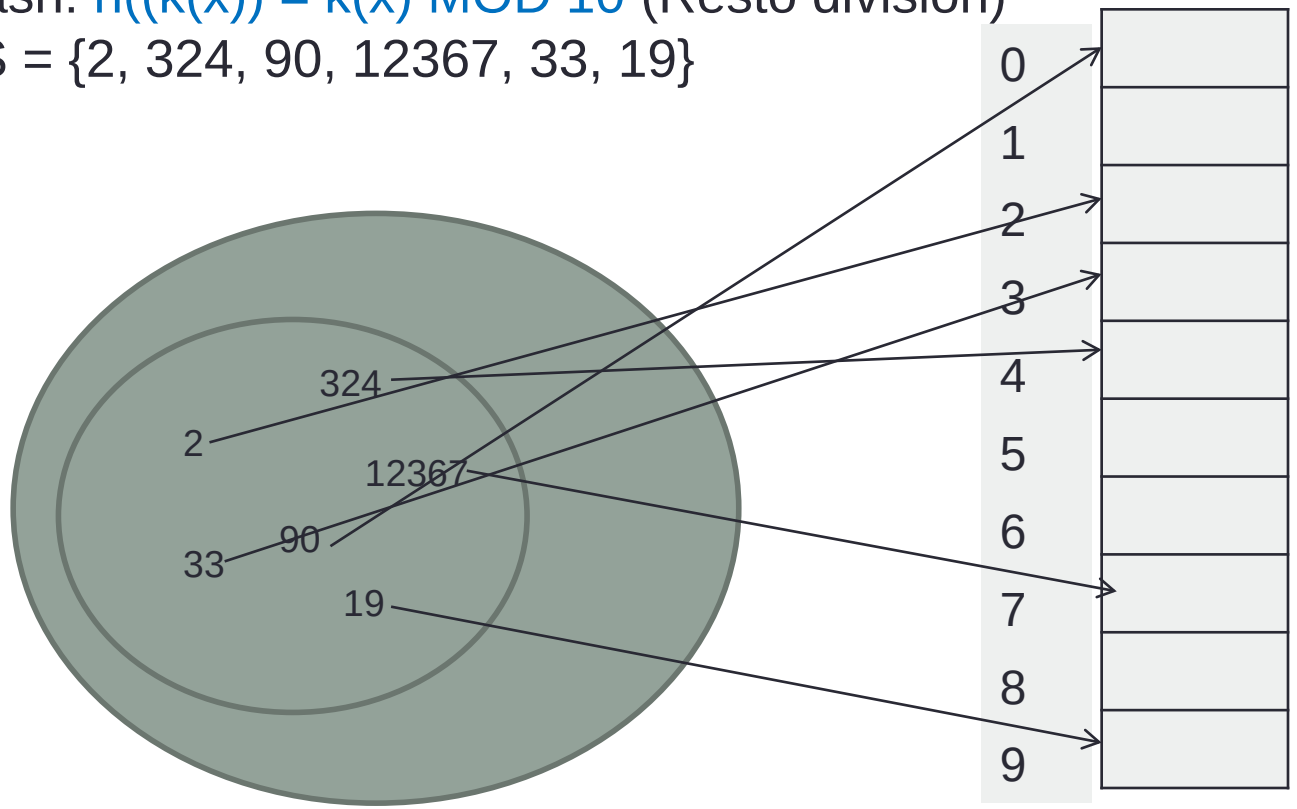
Características deseables:

- Debe ser fácil de calcular
- Debe minimizar el nº de colisiones
- Debe distribuir los elementos de forma aleatoria

Ejemplo: Representar elementos de Z^+

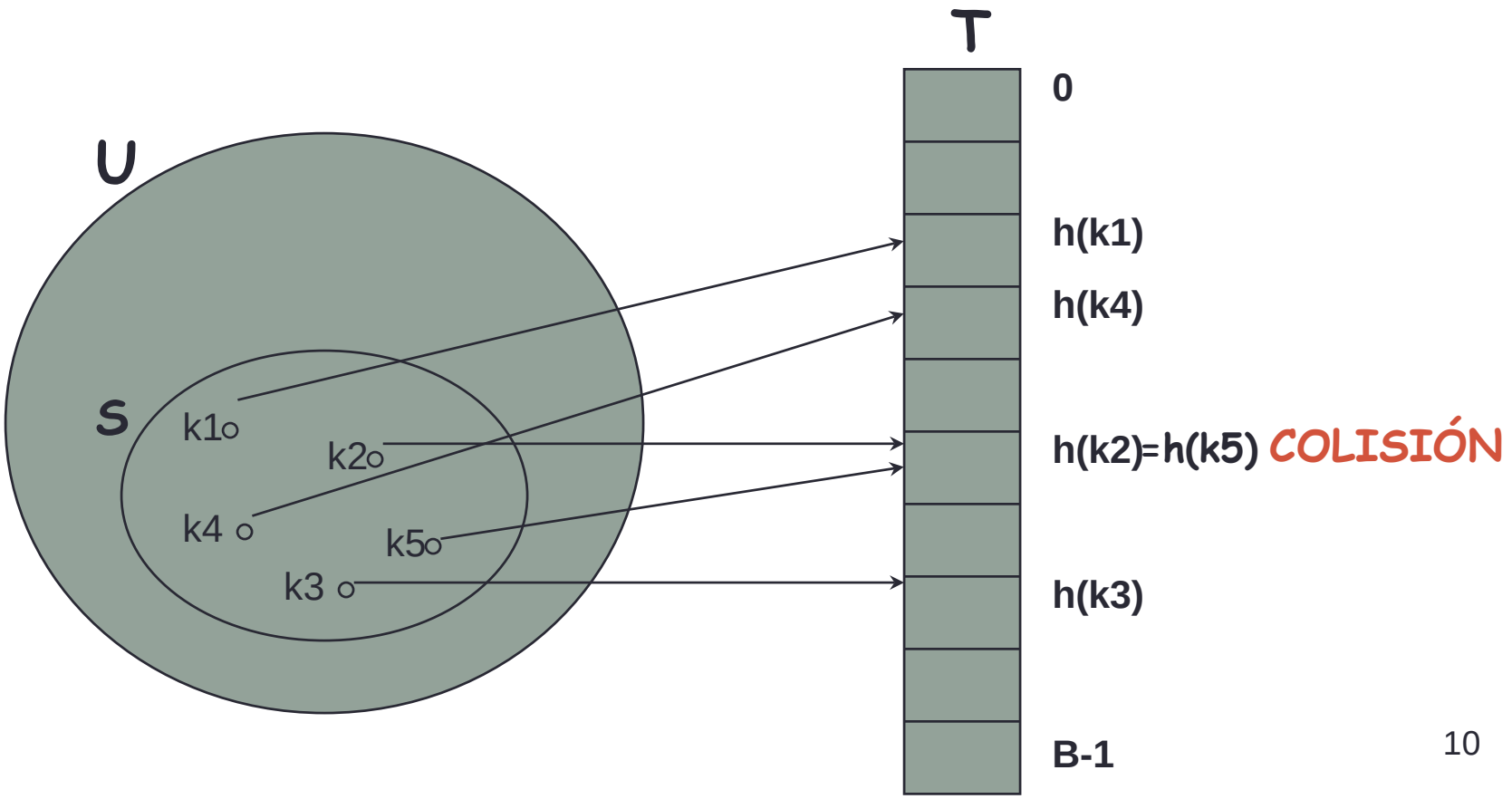
- U= Z^+, {enteros positivos}
- Clave del elemento x: k(x) = x (Es el mismo valor del número)
- Talla de la tabla Hash: B = 10
- Función hash: h((k(x)) = k(x) MOD 10 (Resto división)
- Conjunto S = {2, 324, 90, 12367, 33, 19}

h(2) = 2
h(324) = 4
h(90) = 0
h(12367) = 7
h(33) = 3
h(19) = 9



Función hash

h: U -> {0,1,2,..,B-1}, Coste O(1)



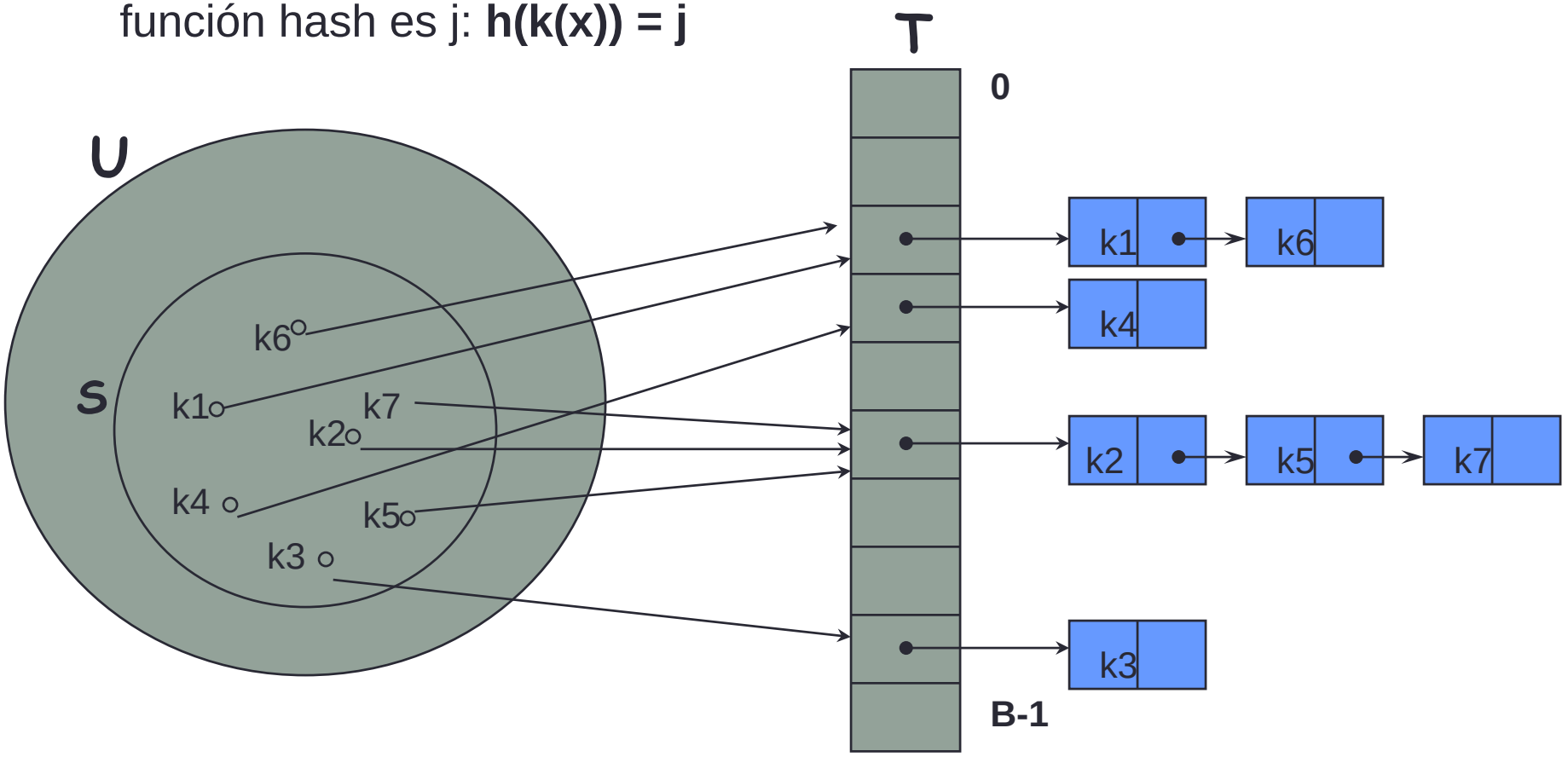
Tratamiento de colisiones

- El objetivo es definir una función hash que minimice el número de colisiones.
- Dos métodos para tratar las colisiones:
  - A. Por encadenamiento
  - B. Por direccionamiento abierto

A. Por encadenamiento

- Construir una lista de claves que tienen el mismo valor de la función hash

  T[j]: puntero a la cabeza de lista de aquellos elementos cuya función hash es j: **h(k(x)) = j**



B. Por direccionamiento abierto

Diferentes opciones de redirección (implementando circularidad):

A. Buscar secuencialmente a partir de *indiceHash* la siguiente posición libre de la tabla

B. Buscar sucesivamente las posiciones $indiceHash+1^2$, $indiceHash+2^2$, ..., $indiceHash+i^2$,.

C. Usar otras funciones: $h(x) = h1(x) + i * h2(x)$

Donde i = número de intentos

Por direccionamiento abierto….cont1

ESTRATEGIA DE REDISPERSIÓN LINEAL (“siguiente posición”):

- No eficiente. Larga secuencia de intentos

$h_i(x) = (h_{i-1}(x) + 1) \text{ MOD } B$

ESTRATEGIA DE REDISPERSIÓN ALEATORIA:

$h_i(x) = (h_{i-1}(x) + c) \text{ MOD } B$ con $c>1$

- Sigue produciendo AMONTONAMIENTO

(c y B no deben tener factores primos comunes mayores que 1)

ESTRATEGIA DE REDISPERSIÓN CON 2ª FUNCION HASH:

$h_i(x) = (h_{i-1}(x) + f(x)) \text{ MOD } B$

$f(x) = (x \text{ MOD } (B-1)) + 1$

(B debe ser primo)

Donde i = número de intentos

Ejercicio#1.

Completar la inserción en una tabla hash cerrada de tamaño *B=7*, con función hash *h(x)= x MOD B*, y con estrategia de redispersión 2ª función hash, los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

$h_i(x) = (h_{i-1}(x) + f(x)) \text{ MOD } B$ siendo $f(x) = (x \text{ MOD } (B-1)) + 1$

Ejercicio#1.

Completar la inserción en una tabla hash cerrada de tamaño *B=7*, con función hash *h(x)= x MOD B*, y con estrategia de redispersión 2ª función hash, los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

$h_i(x) = (h_{i-1}(x) + f(x)) \text{ MOD } B$ siendo $f(x) = (x \text{ MOD } (B-1)) + 1$

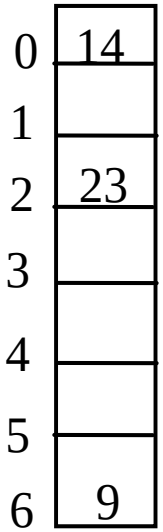
h(23)=23 MOD 7=***2***

h(14)=14 MOD 7=***0***

h(9)=9 MOD 7=2

f(9)=(9 MOD 6)+1=4

$h_1(9)$=(2+4) MOD 7=***6***



Ejercicio#1(Solución).

Insertar en una tabla hash los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

función hash $h(x)= x \ MOD \ B$

Redispersión $h_i(x) = (h_{i-1}(x) + f(x)) \ MOD \ B$ siendo $f(x) = (x \ MOD \ (B-1)) + 1$

h(6)=6 MOD 7=6

f(6)=(6 MOD 6)+1=1

$h_1(6)=(6+1) \ MOD \ 7=0$

$h_2(6)=(0+1) \ MOD \ 7=\mathbf{1}$

0	14
1	
2	23
3	
4	
5	
6	9

0	14
1	6
2	23
3	
4	
5	
6	9

Ejercicio#1(Solución).

Insertar en una tabla hash los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

función hash $h(x)= x \text{ MOD } B$

Redispersión $h_i(x) = (h_{i-1}(x) + f(x)) \text{ MOD } B$ siendo $f(x) = (x \text{ MOD } (B-1)) + 1$

h(6)=6 MOD 7=6

f(6)=(6 MOD 6)+1=1

$h_1(6)=(6+1) \text{ MOD } 7=0$

$h_2(6)=(0+1) \text{ MOD } 7=\mathbf{1}$

h(30)=30 MOD 7=2

f(30)=(30 MOD 6)+1=1

$h_1(30)=(2+1) \text{ MOD } 7=\mathbf{3}$

0	14
1	6
2	23
3	
4	
5	
6	9

0	14
1	6
2	23
3	30
4	
5	
6	9

Ejercicio#1(Solución).

Insertar en una tabla hash los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

función hash $h(x)= x \text{ MOD } B$

Redispersión $h_i(x) = (h_{i-1}(x) + f(x)) \text{ MOD } B$ siendo $f(x) = (x \text{ MOD } (B-1)) + 1$

h(6)=6 MOD 7=6

f(6)=(6 MOD 6)+1=1

$h_1(6)=(6+1) \text{ MOD } 7=0$

$h_2(6)=(0+1) \text{ MOD } 7=\mathbf{1}$

h(30)=30 MOD 7=2

f(30)=(30 MOD 6)+1=1

$h_1(30)=(2+1) \text{ MOD } 7=\mathbf{3}$

h(12)=12 MOD 7=***5***

0	14
1	6
2	23
3	30
4	
5	
6	9

0	14
1	6
2	23
3	30
4	
5	12
6	9

Ejercicio#1(Solución).

Insertar en una tabla hash los siguientes elementos: *23, 14, 9, 6, 30, 12, 18*

función hash $h(x)= x \text{ MOD } B$

Redispersión $h_i(x) = (h_{i-1}(x) + f(x)) \text{ MOD } B$ siendo $f(x) = (x \text{ MOD } (B-1)) + 1$

h(6)=6 MOD 7=6

f(6)=(6 MOD 6)+1=1

$h_1(6)=(6+1) \text{ MOD } 7=0$

$h_2(6)=(0+1) \text{ MOD } 7=\mathbf{1}$

h(30)=30 MOD 7=2

f(30)=(30 MOD 6)+1=1

$h_1(30)=(2+1) \text{ MOD } 7=\mathbf{3}$

h(12)=12 MOD 7=***5***

h(18)=18 MOD 7=***4***

0	14
1	6
2	23
3	30
4	18
5	12
6	9

Nº TOTAL DE INTENTOS HASTA LA CLAVE 18:

11

Se cuentan los *h(...)*.

Cómo obtener una clave de un elemento

(p.ej. de una cadena de caracteres)

1) Método sencillo: Sumar los códigos asociados a los caracteres

elemento		**clave**
casa	→	99 + 97 + 115 + 97 = 408
hola	→	104 + 111 + 108 + 97 = 420

Problema:

hola	→	104 + 111 + 108 + 97 = 420
teja	→	116 + 101 + 106 + 97 = 420

2) Funciones polinomiales: para mejorar la calidad de la función de dispersión se puede ponderar la posición de cada caracter dentro de la clave:

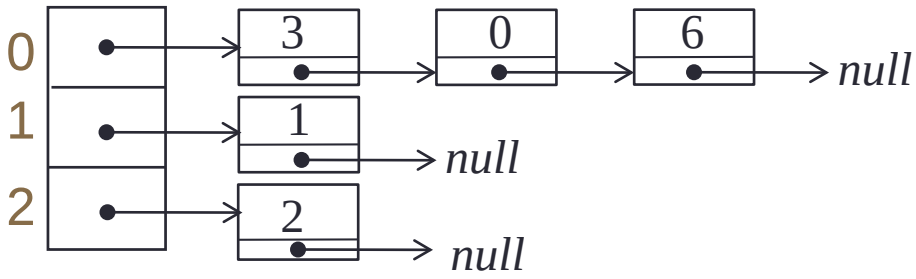
$$F(c) = c_0 \cdot a^{k-1} + c_1 \cdot a^{k-2} + \ldots + c_{k-2} \cdot a^1 + c_{k-1} \quad \text{, con } a>1.$$

Ejemplo con a=2

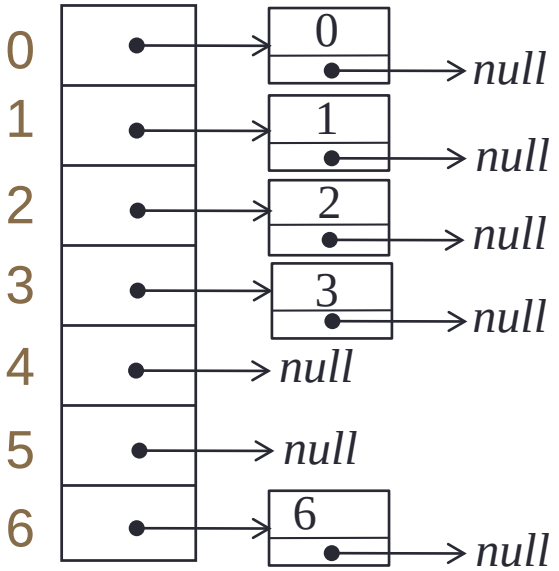
hola	→	$104 \cdot 2^3 + 111 \cdot 2^2 + 108 \cdot 2 + 97 = 1589$
teja	→	$116 \cdot 2^3 + 101 \cdot 2^2 + 106 \cdot 2 + 97 = 1641$

2.3. Rehashing.

- El número de colisiones puede crecer excesivamente si el Factor de Carga es demasiado alto. El Factor de Carga es el cociente entre el número de elementos que contiene la tabla y el número de cubetas.
- El ***rehashing*** consiste en incrementar el tamaño de la tabla hash, recolocando los elementos, reduciendo así su grado de ocupación



FC = 5 elementos/3 cubetas = 1.67



FC = 5 elementos/7 cubetas= 0.71

Tabla hash. Pros y Contras

- Operaciones en tiempo de consulta (on-line): Calcular el valor del hashing del término y localizarlo en la tabla resolviendo colisiones.
- Pros:
  - La localización de un elemento es más rápida que en un árbol. Puede ser de tiempo constante O(1)
- Contras:
  - No se pueden encontrar pequeñas variantes, ya que el hashing las puede colocar en lugares muy distantes (p.e. *resume* vs *résumé*).
  - No se pueden hacer búsquedas de prefijos (p.e. todos los términos que empiezan por "*automat*")
  - Necesidad de hacer un rehashing de todo el vocabulario periódicamente si el vocabulario crece contínuamente.

Ejercicio#2.

Completar la inserción en una tabla hash cerrada de tamaño B=11, con función hash H(x)= x MOD B, y con estrategia de redispersión 2ª función hash, los siguientes elementos:

51, 14, 3, 7, 18, 30.

$h_i(x) = (h_{i-1}(x) + k(x))$ MOD B siendo k (x) = (x MOD (B-1)) + 1

Ejercicio#3.

Dada la siguiente secuencia de claves: A, C, X, H, M, B, J, K

Se pide insertar la secuencia anterior en una tabla Hash de 7 cubetas, donde la función de hashing se calcula de la siguiente manera: a cada clave (letra) se le asigna un número correlativo al orden alfabético (k(A)=0, k(B)=1, k(C)=2…), considerando 27 letras. Y la función de dispersión de una letra x es:

H(x) = (k(x) mod 3)

Las colisiones se resuelven por encadenamiento.

1.2. TIPOS DE DICCIONARIOS: ÁRBOLES

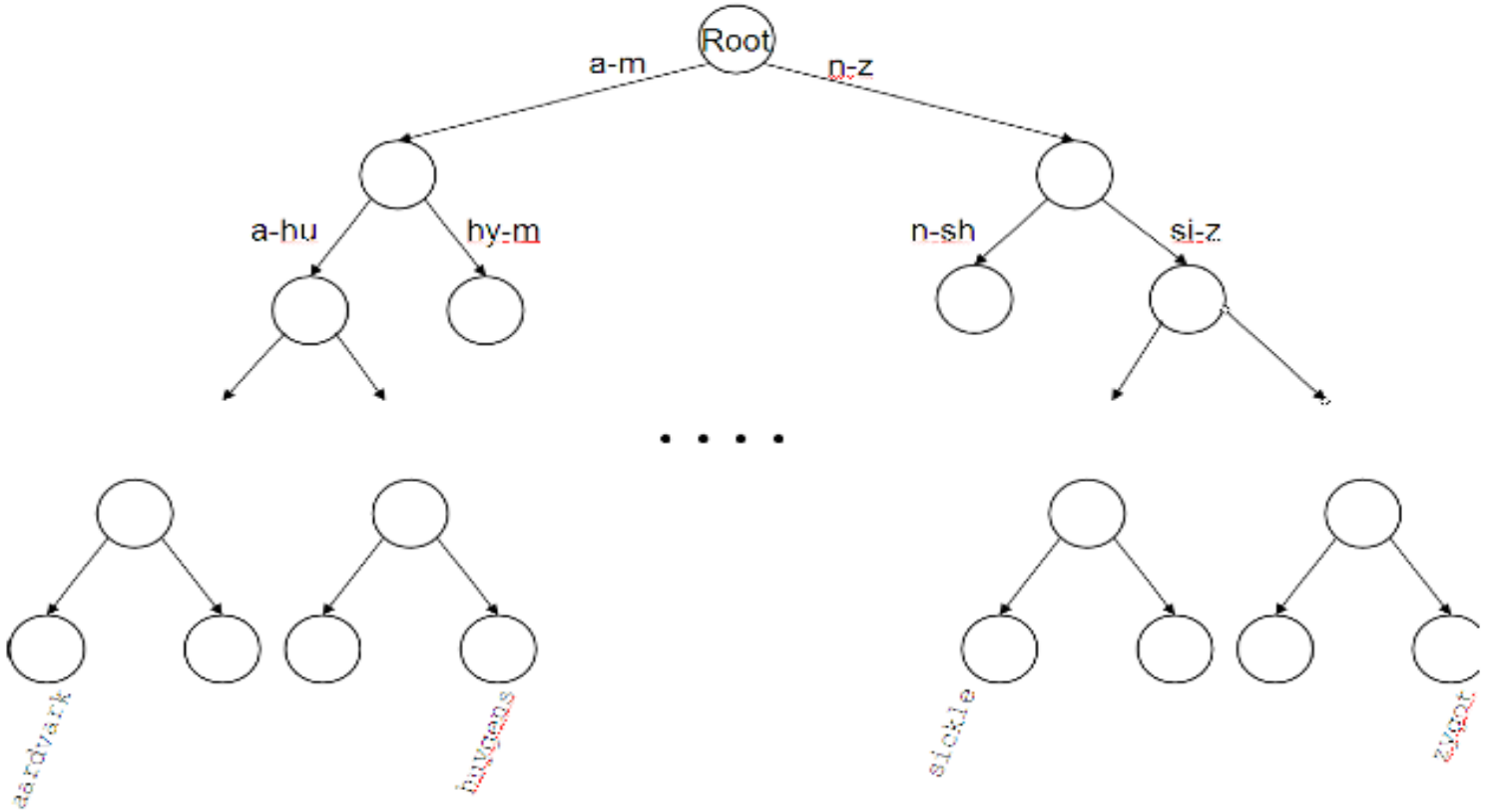
1.2.1. Árbol Binario de Búsqueda
1.2.2. B-Árbol

Árboles

- En los árboles la búsqueda es dirigida por una comparación que se hace en cada uno de sus nodos
- Por ello se requiere un orden estándar de los caracteres para hacer la comparación (mayor, menor, igual). Aunque hay lenguas que no tienen este orden alfabético no es ningún problema definirlo.
- El más simple es el Árbol Binario de Búsqueda.
- El más habitual es el B-Árbol.

1.2.1. Árbol Binario de Búsqueda

Ejemplo, Árbol Binario de Búsqueda con cadenas de caracteres



Árboles binarios de búsqueda. Pros y contras

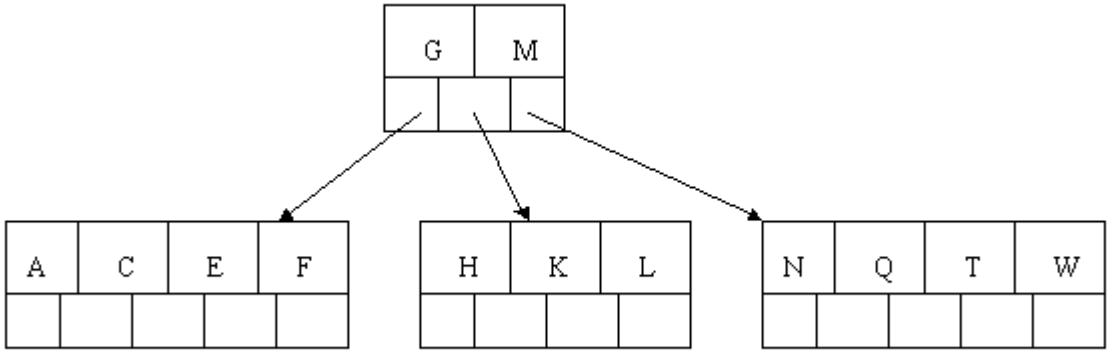
- Los árboles permiten resolver el problema de las búsquedas de prefijos (p.ej. encontrar todos los términos que comienzan por *automat*).
- La búsqueda es ligeramente más costosa que en la Tabla Hash: $O(\log M)$, donde M es la talla del vocabulario.
- El coste $O(\log M)$ sólo es cierto para árboles balanceados.
- Rebalancear árboles binarios es costoso.

1.2.2. B-Árbol

- Los B-árboles permiten que el número de subárboles de un nodo varíe entre un intervalo fijo, con ello mitigan el problema del rebalanceo.

- Búsquedas en un B-árbol: se realiza una (o varias) comparaciones en cada nodo para escoger el camino por el que descender.

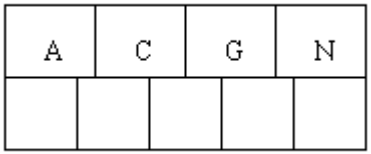
- Ejemplo: B-Árbol de orden 5 (máx 5 hijos y 4 claves).



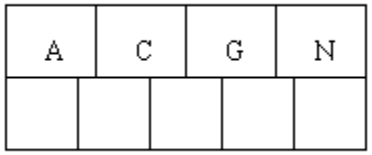
Inserción en un B-Árbol

Supongamos que queremos insertar los siguientes caracteres en un B-árbol de orden 5 (máximo de 5 hijos y 4 claves): C N G A H E K Q M F W L T Z D P R X Y S

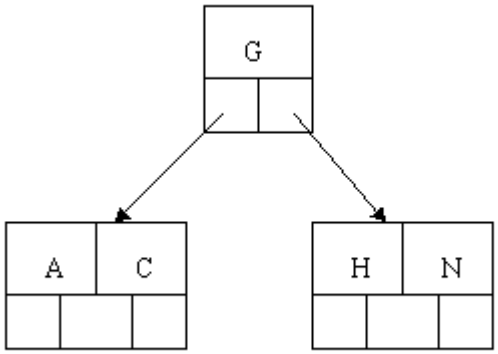
1. Insertar las cuatro primeras letras



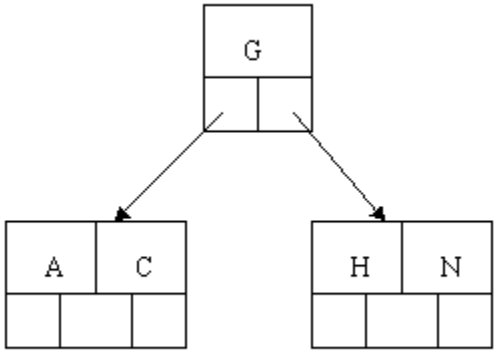
C N G A H E K Q M F W L T Z D P R X Y S



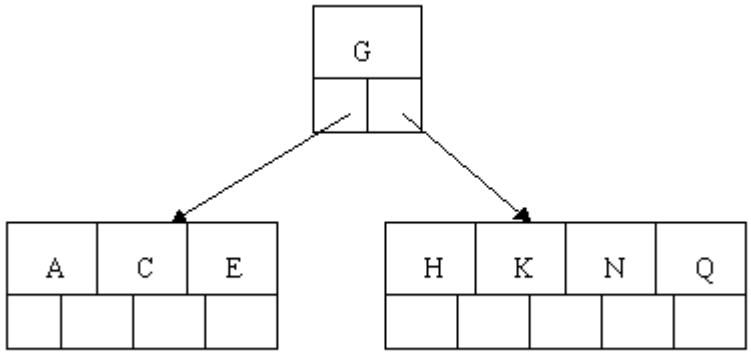
2. Como ya no hay sitio para la H, se desdobla este nodo en dos, que cuelgan de un nodo raíz, que es el valor de la mediana.



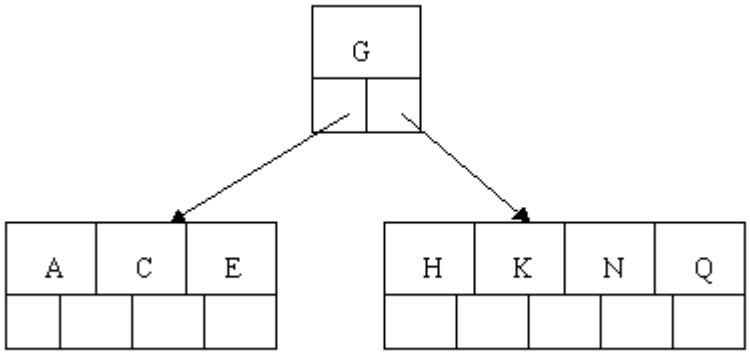
C N G A H E K Q M F W L T Z D P R X Y S



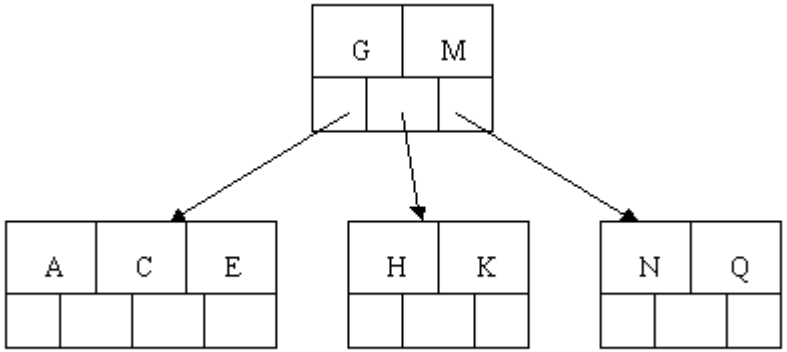
3. Para insertar E, K, y Q no hace falta dividir nodos.



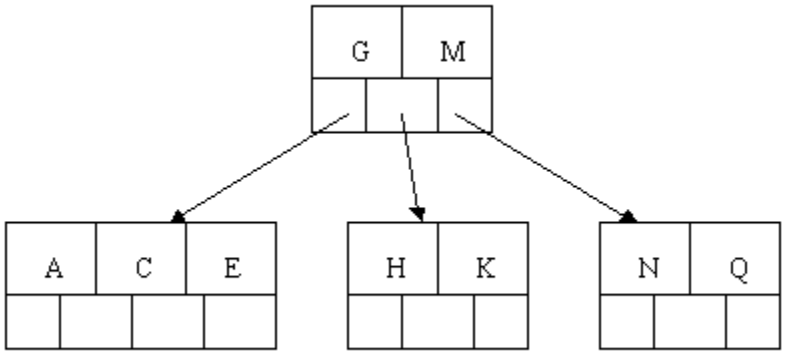
C N G A H E K Q M F W L T Z D P R X Y S



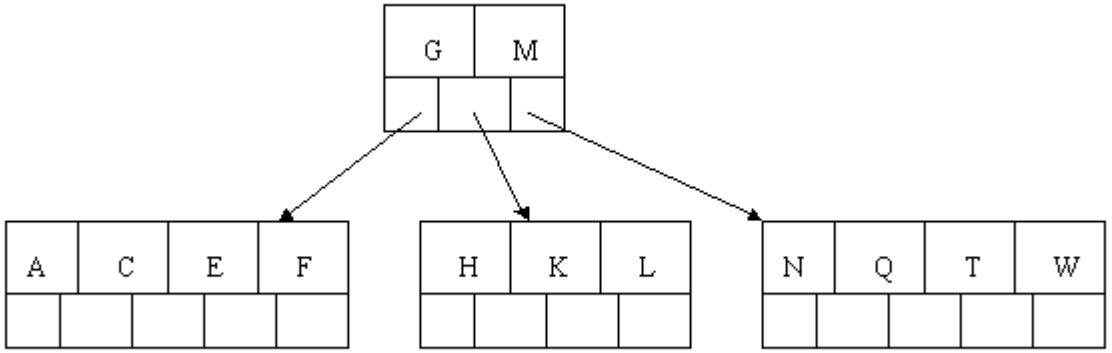
4. Insertar M requiere una división. En este caso la mediana es M, y es el que se sube al nodo padre.



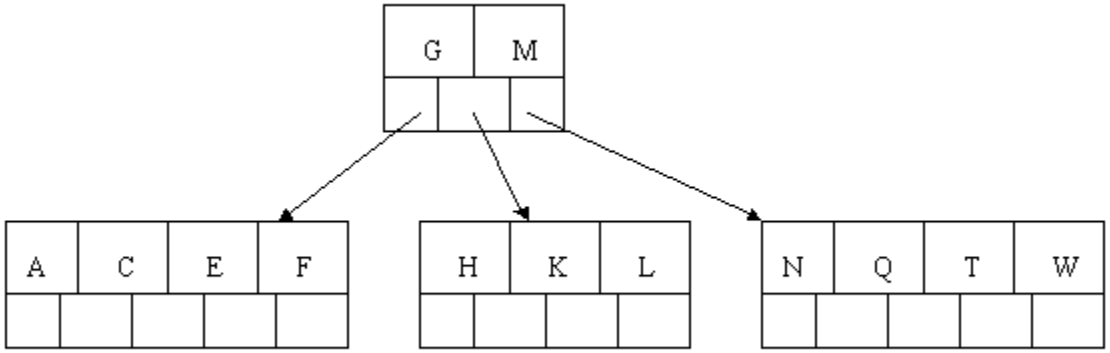
C N G A H E K Q M F W L T Z D P R X Y S



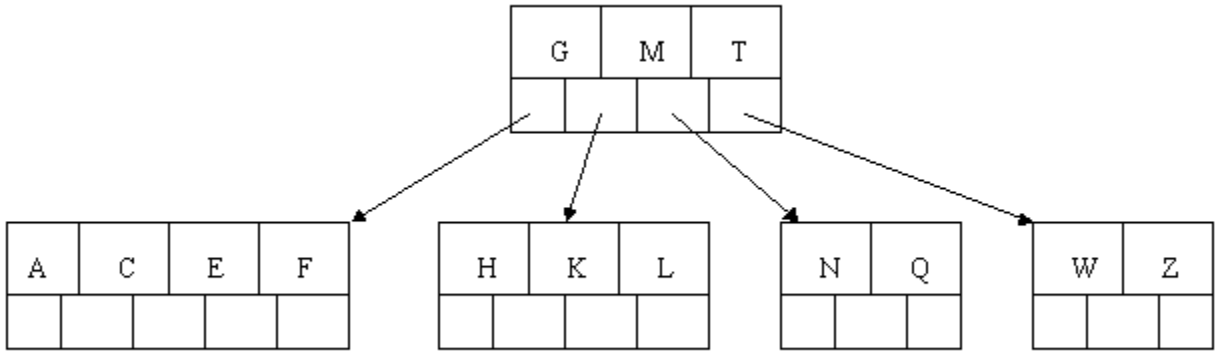
5. Las letras F, W, L, and T se añaden sin hacer divisiones.



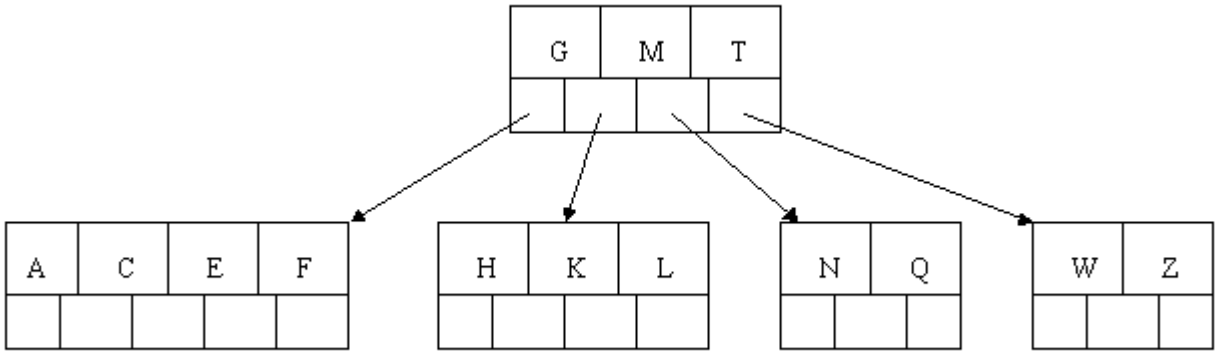
C N G A H E K Q M F W L T Z D P R X Y S



6. Al añadir la Z se produce una división.

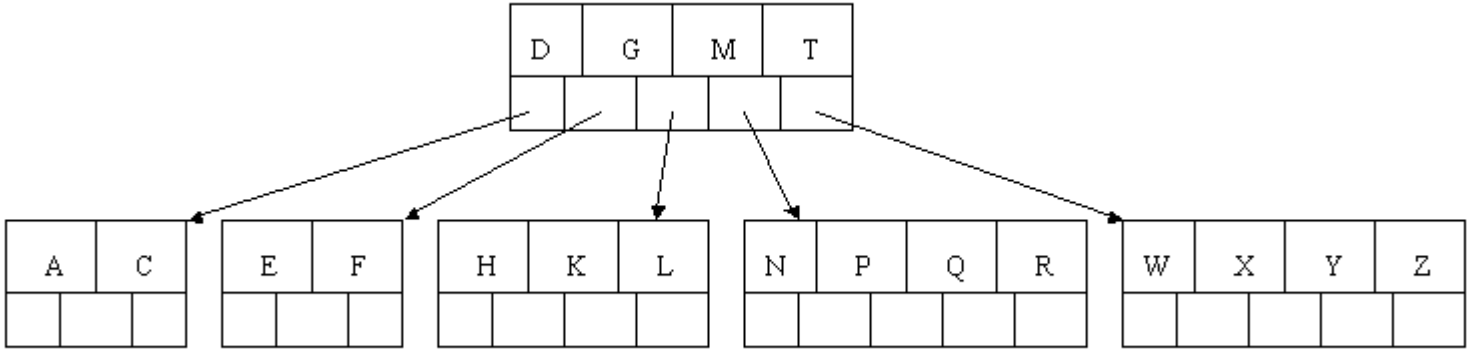


C N G A H E K Q M F W L T Z D P R X Y S

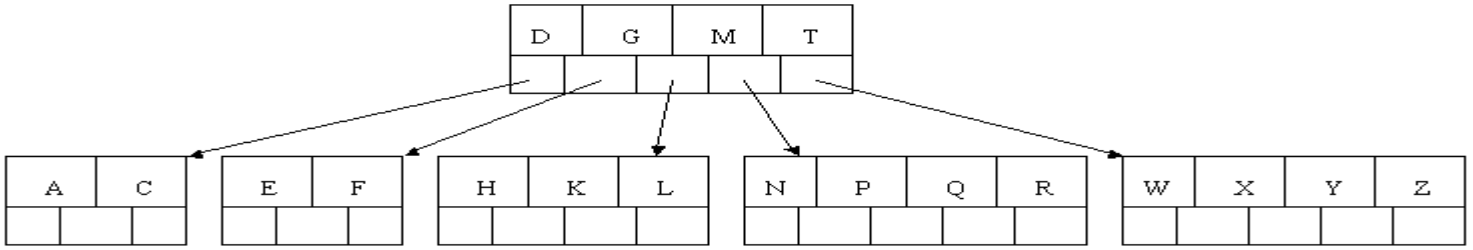


7. La inserción de D causa una división en la hoja de la izquierda. D es la mediana y por tanto se sube al nodo padre.

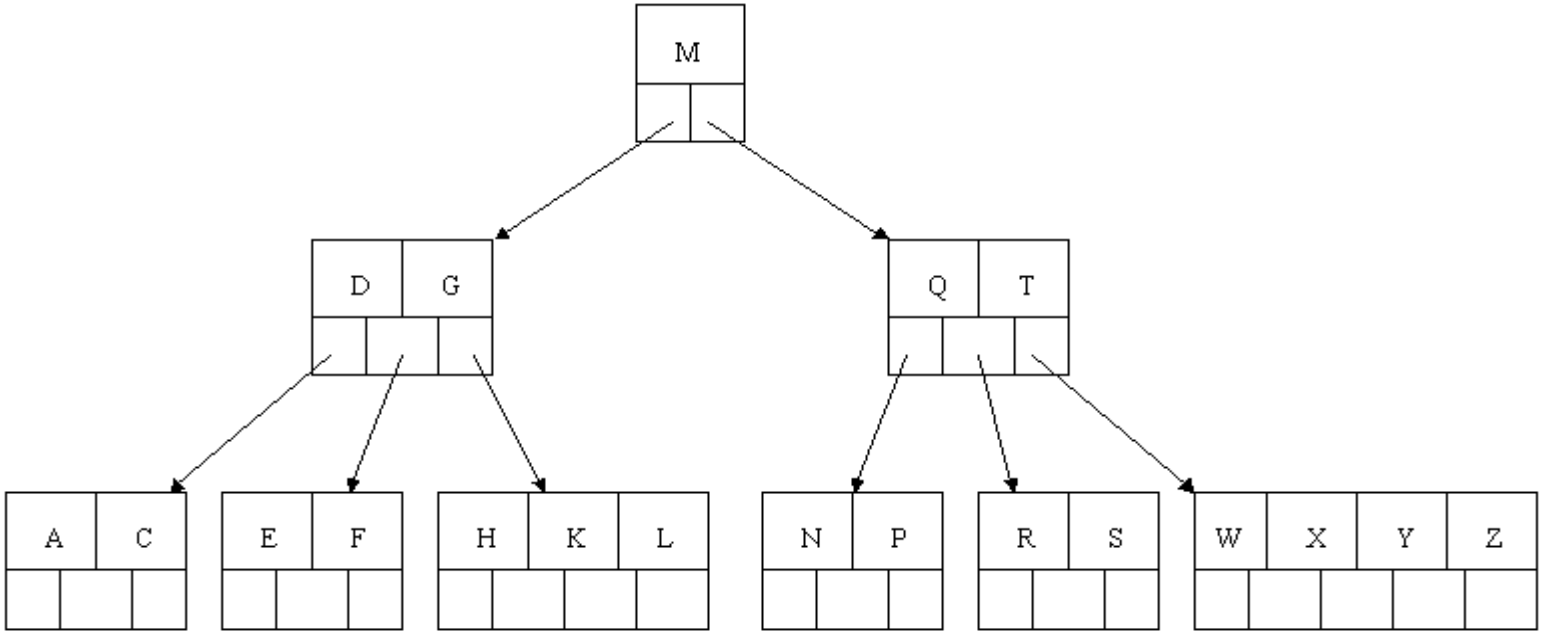
8. Las letras P, R, X,Y se añaden sin hacer divisiones.



C N G A H E K Q M F W L T Z D P R X Y S



9. Para añadir S, el nodo NPQR se divide enviando la mediana Q al padre. Sin embargo el padre está lleno, por tanto tiene que dividirse, enviado su mediana M hacia arriba creando un nuevo nodo raíz.



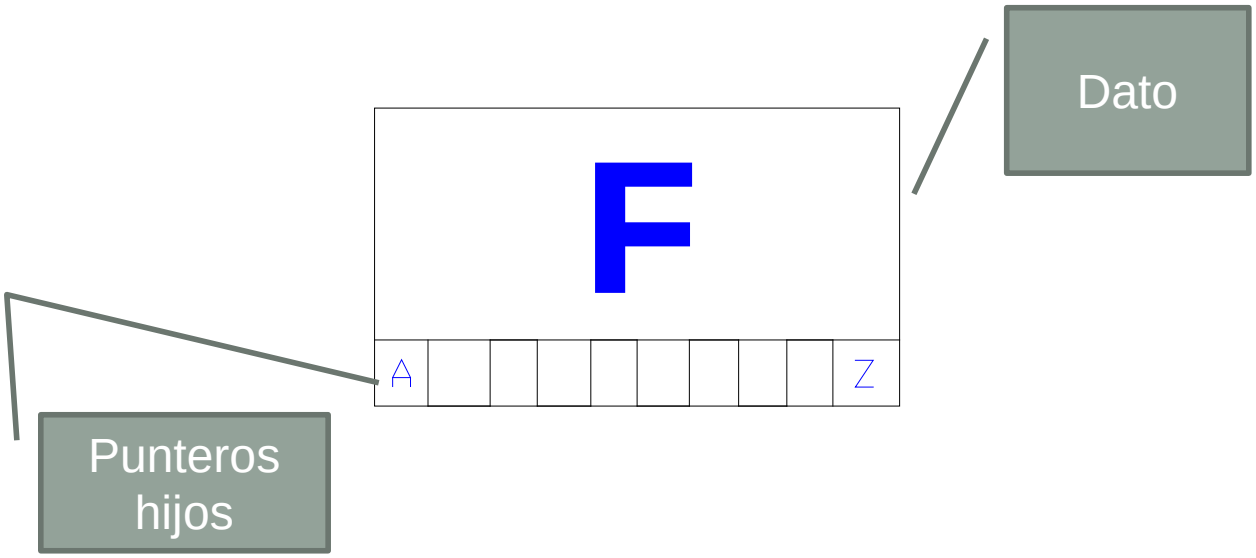
1.3. TIPOS DE DICCIONARIOS: TRIE

Trie

- Árbol para representar conjuntos de cadenas de caracteres u objetos (DICCIONARIO DE CADENAS):
- Cada nodo representa el prefijo de una palabra
- Los hijos de un nodo representan las cadenas que tiene a sus padres como prefijos
- Ventajas:
  - Búsquedas parciales (palabras que empiezan por “AR”)
  - No necesitan operación “redimensionar tabla”
  - Cada caracter se almacena una sola vez en los prefijos comunes
  - Complejidad en función de la longitud de la palabra y no en función del nº de palabras

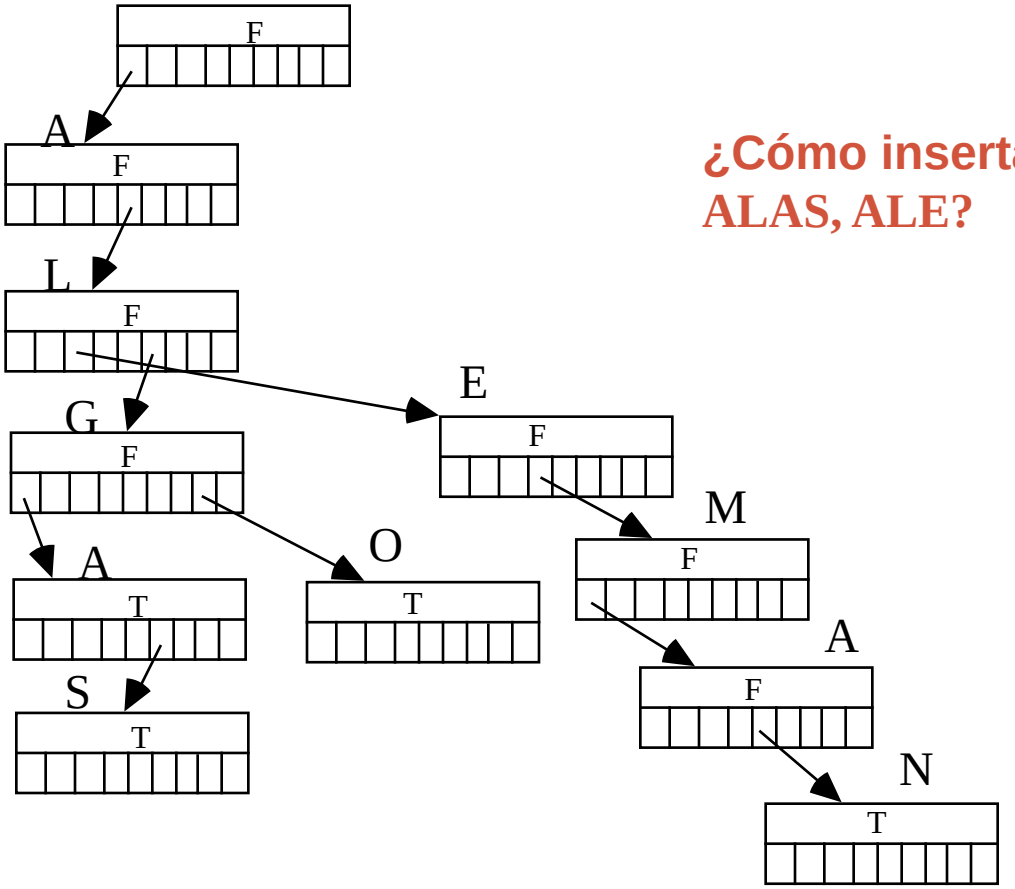
Nodo:

- **Dato**: campo booleano (**T** indica si es una palabra completa o **F** un prefijo)
- **Estructura de punteros** para todos los posibles hijos



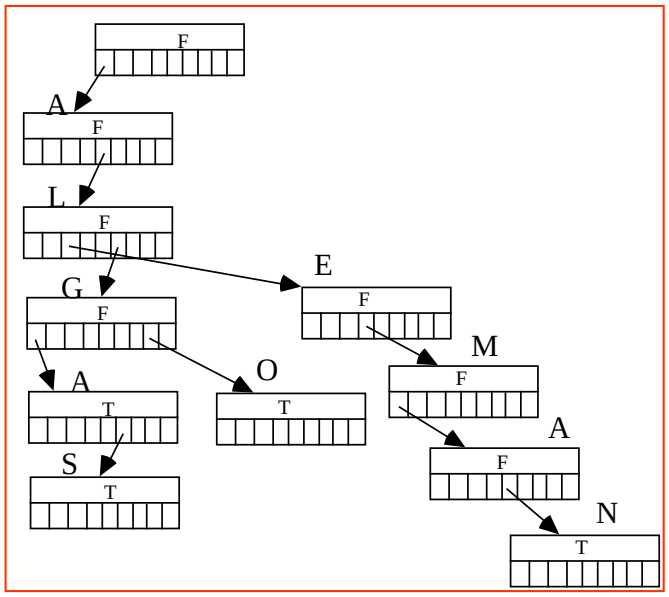
Crear: Constructor que pondrá la variable booleana a FALSE y los punteros nulos

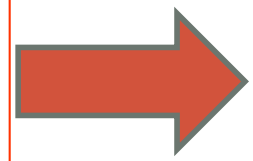
ALGA, ALGAS,
ALGO,
ALEMAN

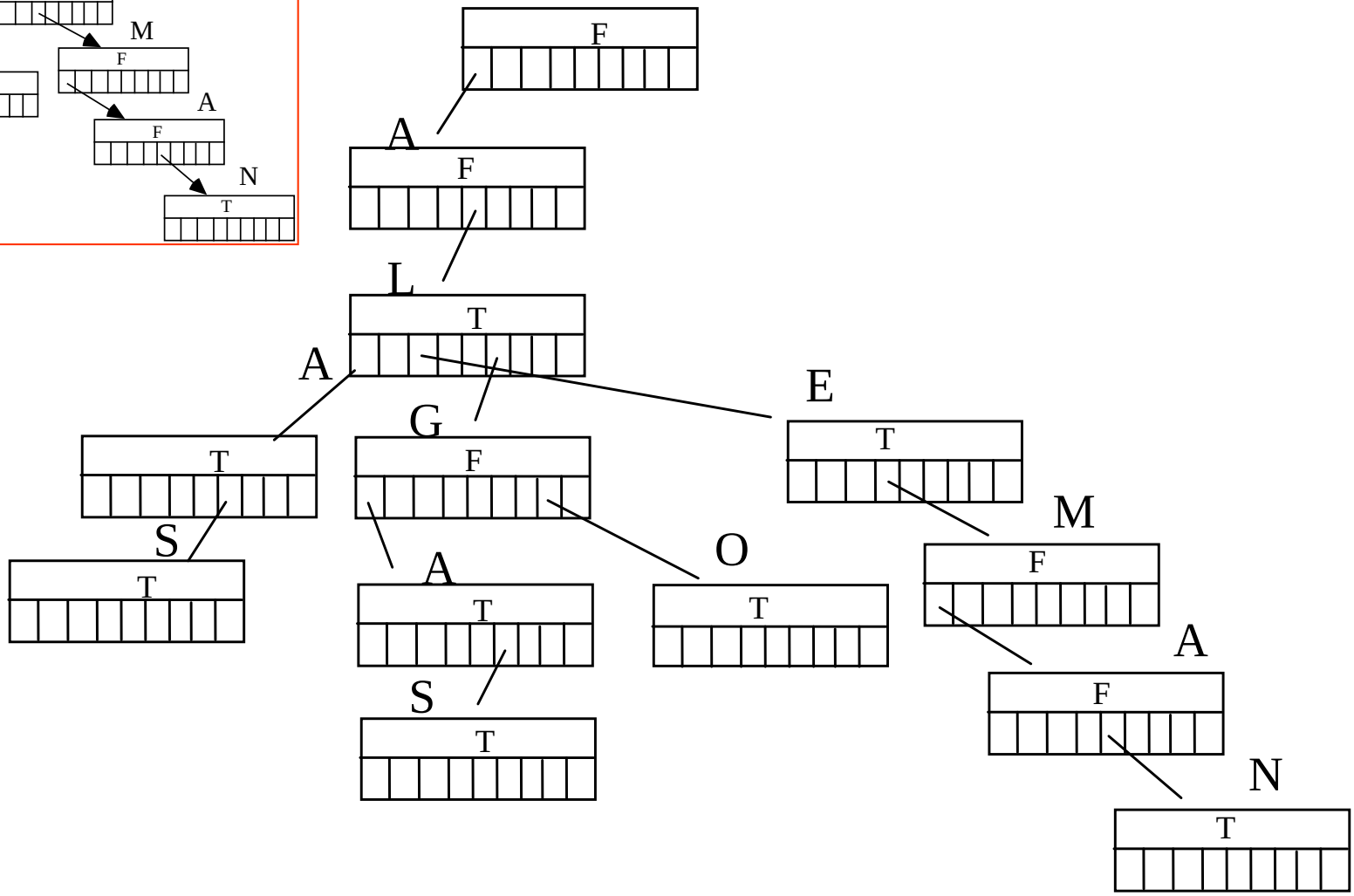


¿Cómo insertar AL, ALA, ALAS, ALE?

Insertar AL, ALA, ALAS, ALE







¿Pertenece la cadena *palabra* al diccionario?

```
ALGORITMO BÚSQUEDA_palabra_TRIE (palabra, T, Encontrado)
/*ENTRADA: palabra, T es el Trie
/*SALIDA: Encontrado: Boolean;
p ← T
Encontrado ← FALSE
mientras NOT EsVacio (p) AND NOT Encontrado
hacer    Si LONGITUD (palabra) = 0
         entonces   Encontrado ← Dato (p)
         sino       c ← OBTENER (palabra,1)
                    p ← p.HIJO (c)
                    Si NOT EsVacio (p)
                    entonces   palabra ← SUPRIMIR (palabra)
```

Donde:

Dato (p) : devuelve T (TRUE) o F (FALSE)
OBTENER (palabra,1) : devuelve primer carácter de ***palabra***
p.HIJO (c) : devuelve la posición del nodo hijo de **c**
SUPRIMIR (palabra): elimina el primer carácter de la *palabra*

2. BÚSQUEDA CON TOLERANCIA.

2.1 Indice Permuterm

2.2 Indice de k-gramas

Búsqueda con tolerancia

A menudo el usuario no está seguro de cómo se escribe una palabra o quiere buscar variantes de esa palabra.

Ejemplo1: se quiere buscar Sidney, pero no está seguro de si es ***Sidney*** o ***Sydney*** → buscar S*dney.

Ejemplo2: se quiere encontrar formas distintas de escribir la palabra ***color*** o ***colour*** → buscar col*r

Ejemplo3: se quiere buscar documentos que contienen ciertas variantes de un término, que podrían encontrarse si el sistema utilizara stemming, pero el usuario no lo sabe, p.ej. ***judicial, judiciary*** → buscar judicia*

*: representa cualquier cadena de caracteres

Wildcard query (consulta con comodín)

1) El * está al final de la palabra: mon*

Si el diccionario es un árbol es sencillo encontrar todos los términos W que tienen como prefijo *mon*. Luego se hacen |W| búsquedas de documentos que contienen los términos con ese prefijo.

2) Si * está al principio de la palabra: *mon

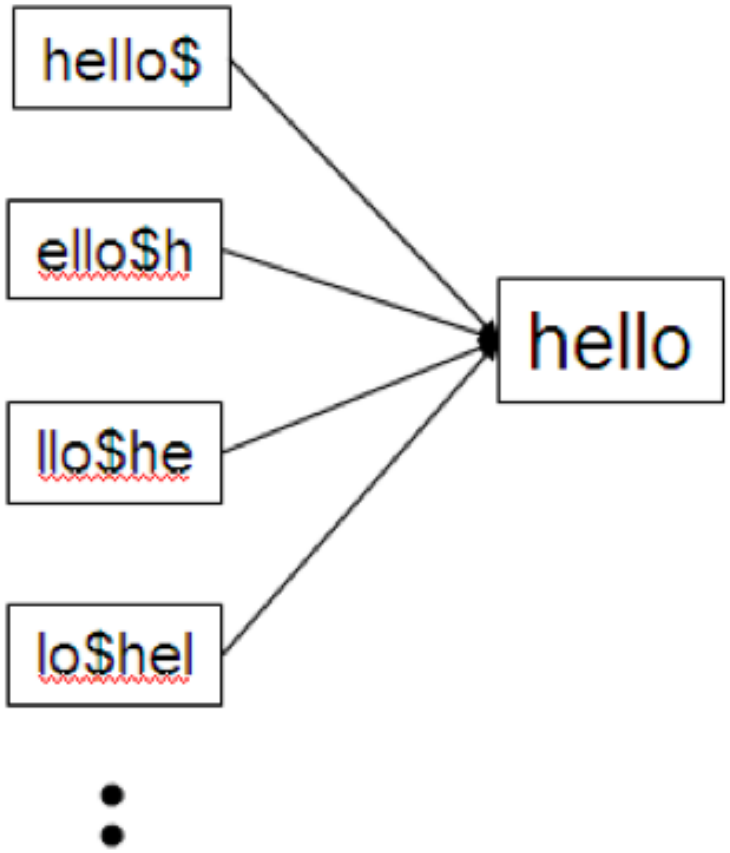
Se puede tener un B-árbol reverso que contiene el diccionario con las palabras escritas al revés.

3) Si el * está en medio de la palabra: se*mon

Podemos usar el árbol sencillo para encontrar las palabras cuyo prefijo es *se* y el B-árbol reverso para encontrar las palabras cuyo sufijo es *mon*. Luego se hace la intersección de los dos conjuntos obteniendo todas las palabras del diccionario que contienen el prefijo *se* y el sufijo *mon*.

2.1 Índice Permuterm

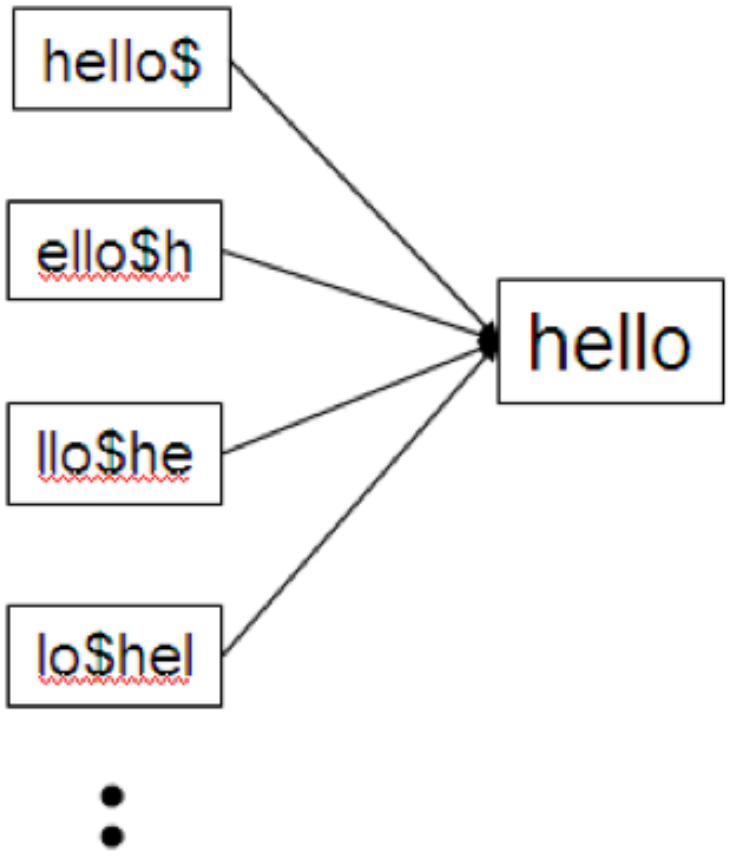
- Símbolo final del término: $
- Construye un Índice Permuterm con las diferentes rotaciones de cada término, todos enlazados al término original.
- Para la wildcard query m*n se rotaría apareciendo el * al final de la cadena buscando la cadena n$m* en el Índice Permuterm.
- A través de un árbol de búsqueda se dirigiría a términos como man o moron, entre otros.



Problema: incrementa sensiblemente la talla del diccionario.

Queries

- Para X → buscar X$
- Para X* → buscar $X*
- Para *X → buscar X$*
- Para *X* → buscar X*
- Para X*Y → buscar Y$X*



Query = hel*o (X=hel, Y=o) → buscar o$hel*

Ejercicio: ¿Cómo se construiría el índice permuterm de la palabra “John”? Explica el mecanismo de búsqueda para la wilcard query “J*n”.

El índice permuterm para el término John se construiría con las diferentes rotaciones del término:

John$

ohn$J

hn$Jo

n$Joh

$John

Y la búsqueda que se realiza es: n$J*

siguiendo la regla: Para buscar X*Y → buscar Y$X*

2.2 Índice de k-gramas

Enumerar todos los k-gramas (secuencias de k caracteres) que aparecen en cada término. Se añaden marcas de inicio y fin de palabra.

Ejemplo: "***April is the cruelest month***" se tienen los bigramas:

a,ap,pr,ri,il,l,i,is,s,t,th,he,e,$c,cr,ru,
ue,el,le,es,st,t$, m,mo,on,nt,h

El diccionario de un índice de k-gramas contiene todos los k-gramas que ocurren en los términos.

Se mantiene un segundo índice invertido desde los k-gramas a términos del diccionario que emparejan con ellos.

Ejemplo de posting list para el trigrama **etr**:

Procesar la wildcard query

Query ***re*ir:*** queremos buscar los documentos que contienen términos que empiecen por ***re*** y acaben en ***ir.***

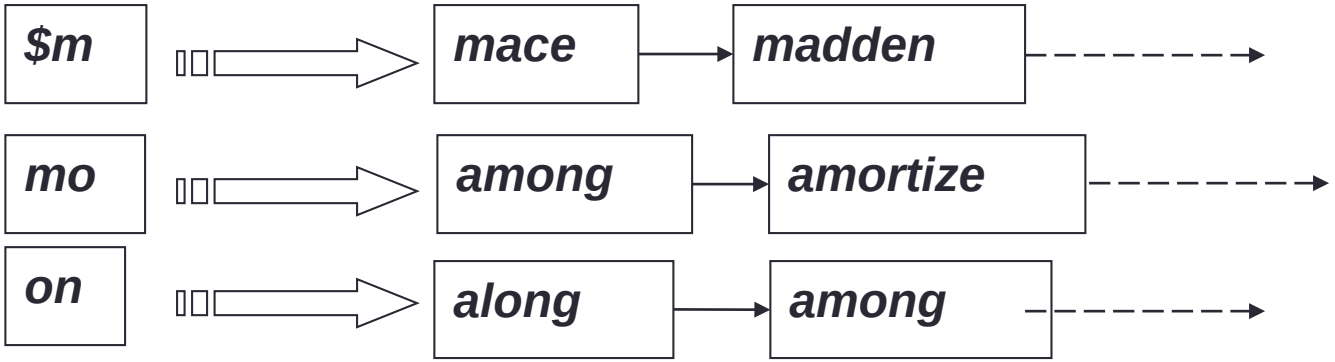
Ejecutar la consulta booleana: ***$re*** *AND* ***ir$***

1. En un Indice de 3-gramas se busca la lista de términos que emparejan, p.e. revivir y resistir.
2. En el Indice Invertido normal buscaremos los documentos que contienen estos términos.

Rápido, gestión del espacio eficiente (comparado con permuterm).

Procesar la wildcard query

- El índice *k*-gramas encuentra términos que contienen los *k*-gramas de la consulta (p.ej. *k*=2).



- Query ***mon**** se puede ejecutar como una expresión boleana
  - ***$m*** *AND* ***mo*** *AND* ***on***

- Podríamos haber obtenido ***moon***.
- Post-filtrar estos términos respecto a la query original.

3. CORRECCIÓN DE ERRORES

Corrección errores

Objetivo: corrección de errores en la consulta para obtener una respuesta correcta.

Ejemplo: el usuario teclea ***carot*** → corregir por ***carrot***

Aproximaciones:

1) Seleccionar la palabra más cercana: Distancia de edición.

2) Cuando hay más de una alternativa posible seleccionar la más común (mayor nº de ocurrencias en la colección o, en entornos web, la que mayor nº de usuarios han tecleado)

Ejemplo

El usuario teclea “graffe”

¿Cual es más cercana?

- graf
- graft
- grail
- giraffe

Necesitamos definir el concepto de “el más cercano”!!

Distancia de edición

La mínima distancia entre dos cadenas:

Es el número mínimo de operaciones de edición

- Inserción
- Borrado
- Sustitución

necesario para transformar una cadena en otra.

Calcular la mínima distancia de edición:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

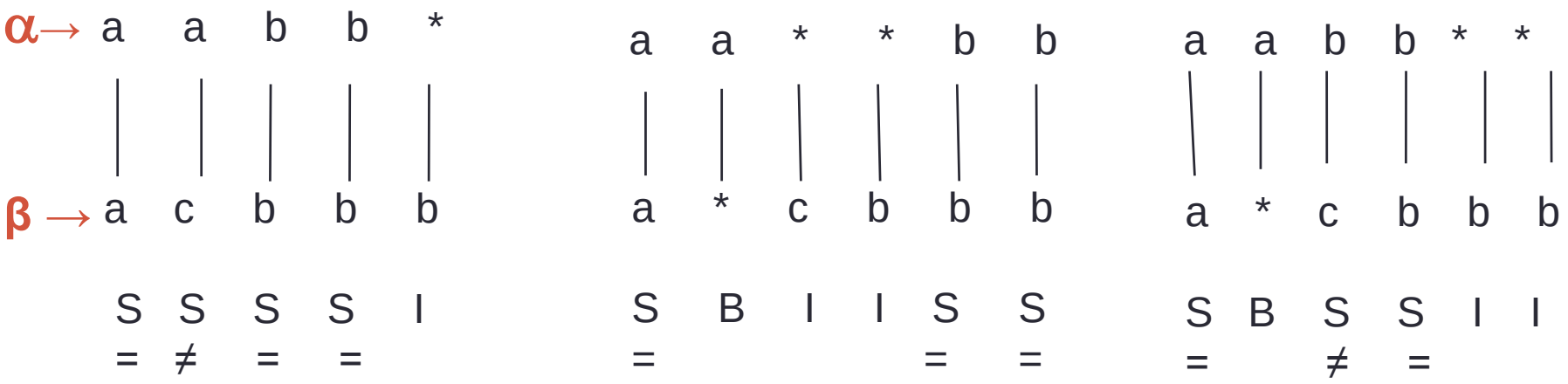
- La distancia será 5 si cada operación tiene coste 1.

Distancia: mínimo número de reglas de error para convertir la cadena **α** en **β**

$$D(\alpha, \beta) = \min_{\forall \Gamma}(N_s(\Gamma) + N_i(\Gamma) + N_B(\Gamma))$$

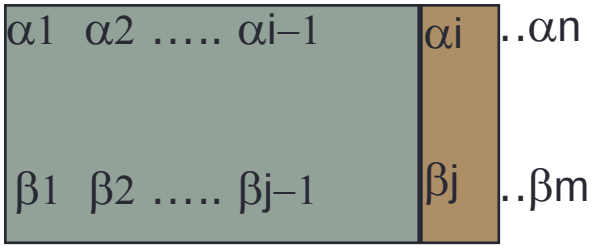
Como sólo se puede aplicar una regla de error a un carácter, una transformación corresponde con un "alineamiento" Γ entre las dos cadenas.

Ejemplo:

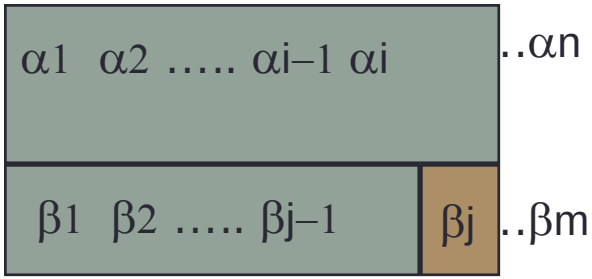


Solución recursiva:

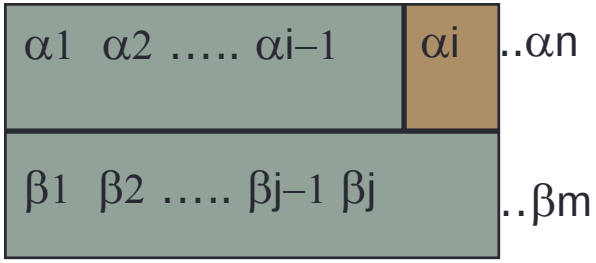
¿Cuál es la mejor forma de haber alineado hasta el carácter *i* de la cadena α y el carácter *j* de la cadena β?.



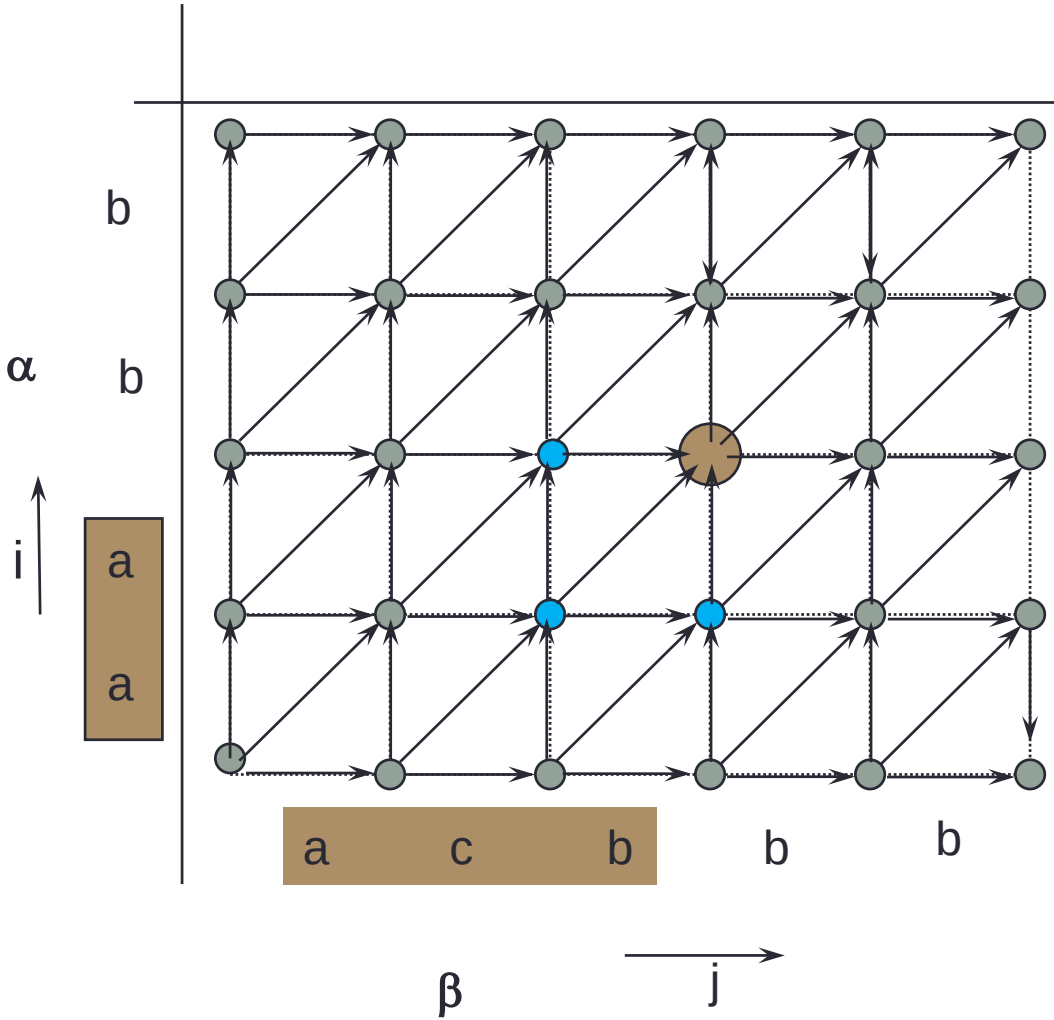
D(i - 1, j - 1) + Sust (α_i, β_j)

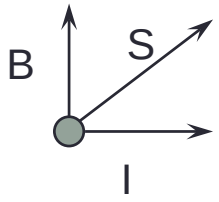


D(i, j - 1) + Ins (β_j)



D(i - 1, j) + Borr (α_i)





$$D[i,j]=\min\{D[i-1,j-1]+Sust(\alpha i,\beta j),$$
$$D[i,j-1]+Ins(\beta j),$$
$$D[i-1,j]+Borr(\alpha i)\ \}$$

Distancia de edición mínima (Levenshtein)

- Inicialización

  $D(i,0) = i$

  $D(0,j) = j$

- Cálculo de la matriz

  desde j←1 **hasta** $|\beta|$ **hacer**

  desde i← 1 **hasta** $|\alpha|$ **hacer**

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & /*\text{Borrado}*/ \\ D(i,j-1) + 1 & /*\text{Inserción}*/ \\ D(i-1,j-1) + \begin{cases} 1; \text{if } \alpha(i) \neq \beta(j) & /*\text{Sustitución}*/ \\ 0; \text{if } \alpha(i) = \beta(j) \end{cases} \end{cases}$$

- Terminación

  $D(|\alpha|, |\beta|)$ es la distancia

Levenshtein prueba todos los caminos posibles y se queda con el de mínimo valor.

Tabla de distancia de edición

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

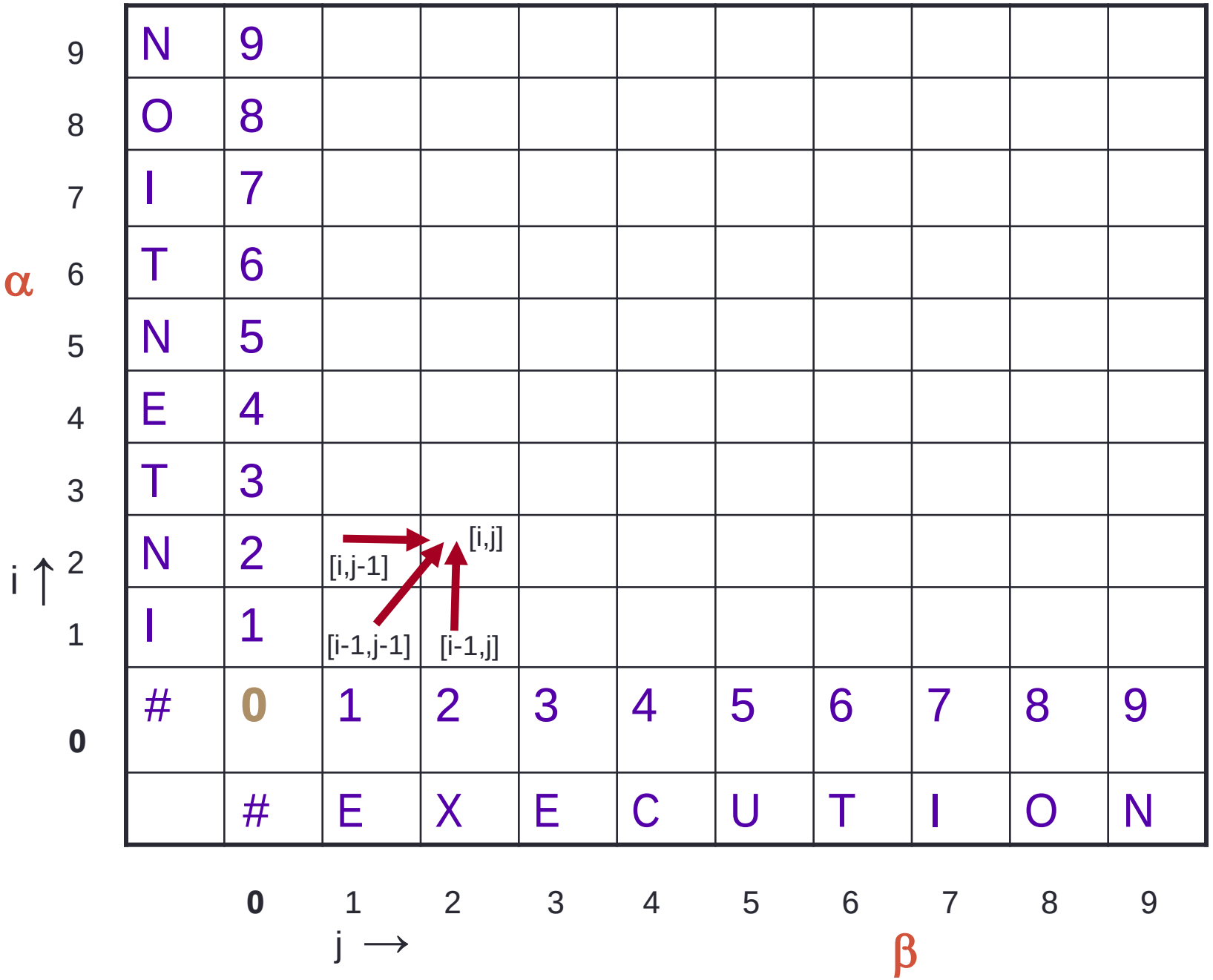
α

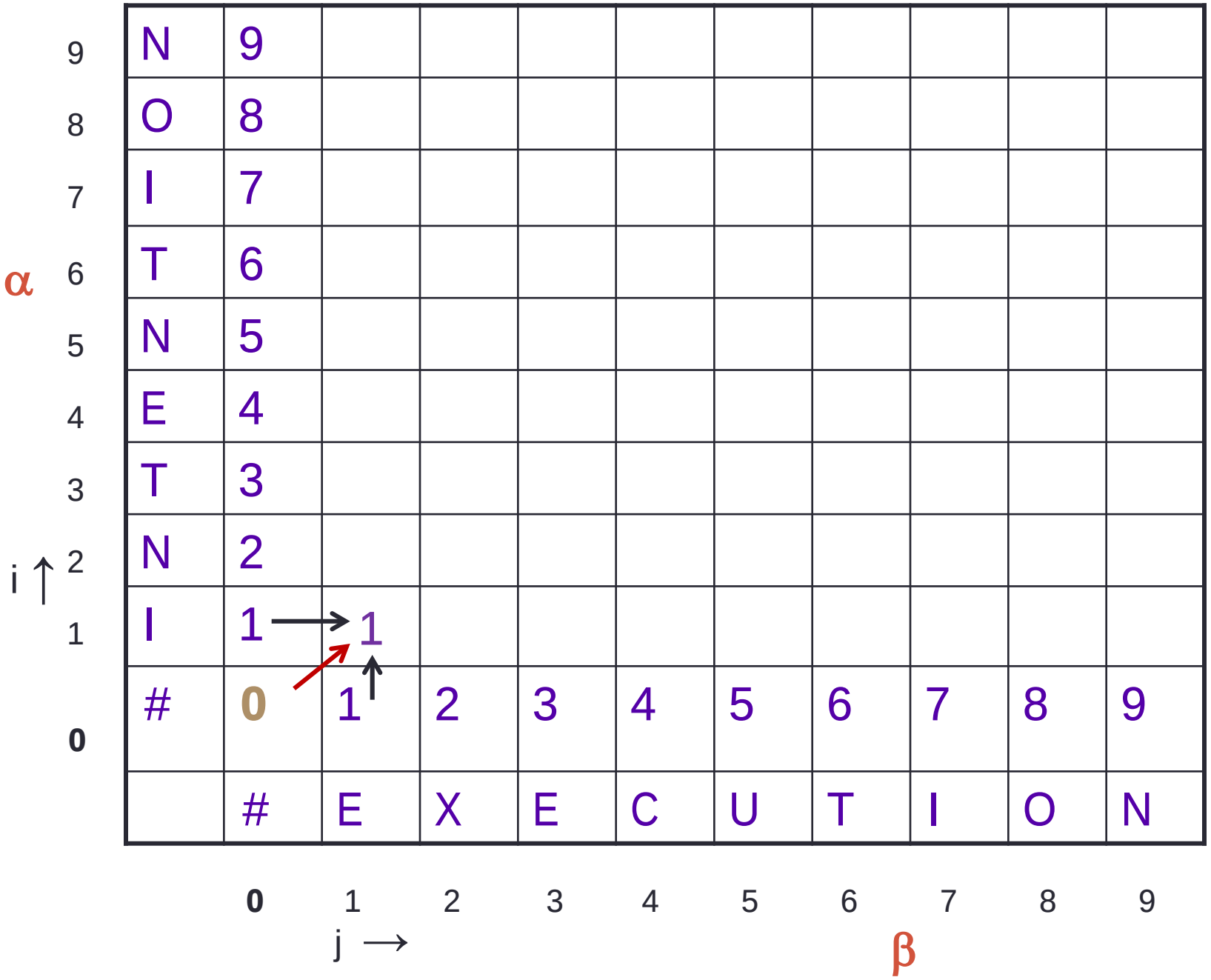
N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

i ↑

j →

β


9
8
7
α
6
5
4
3
i ↑
2
1
0
N 9
O 8
I 7
T 6
N 5
E 4
T 3
N 2
I 1
0 1 2 3 4 5 6 7 8 9
E X E C U T I O N
[i,j]
[i,j-1]
[i-1,j-1]
[i-1,j]
0 1 2 3 4 5 6 7 8 9
j →
β


N 9
O 8
I 7
T 6
N 5
E 4
T 3
N 2
I 1 1
0 1 2 3 4 5 6 7 8 9
E X E C U T I O N
9 8 7 6 5 4 3 2 1 0
α
i
0 1 2 3 4 5 6 7 8 9
j
β

9	N	9	8							
8	O	8	7							
7	I	7	6							
α 6	T	6	5							
5	N	5	4							
4	E	4	3							
3	T	3	3							
i ↑ 2	N	2	2							
1	I	1	1							
0	#	**0**	1	2	3	4	5	6	7	8	9
		#	E	X	E	C	U	T	I	O	N

0 1 2 3 4 5 6 7 8 9

j →

β

9	N	9	8	8	8	8	8	8	7	6	5
8	O	8	7	7	7	7	7	7	6	5	6
7	I	7	6	6	6	6	6	6	5	6	7
6	T	6	5	5	5	5	5	5	6	7	8
5	N	5	4	4	4	4	5	6	7	7	7
4	E	4	3	4	3	4	5	6	6	7	8
3	T	3	3	3	3	4	5	5	6	7	8
2	N	2	2	2	3	4	5	6	7	7	7
1	I	1	1	2	3	4	5	6	6	7	8
0	#	0	1	2	3	4	5	6	7	8	9
		#	E	X	E	C	U	T	I	O	N

α

i ↑

0 1 2 3 4 5 6 7 8 9

j →

β

9	N	9	8	8	8	8	8	8	7	6	**5**
8	O	8	7	7	7	7	7	7	6	**5**	6
7	I	7	6	6	6	6	6	6	**5**	6	7
6	T	6	5	5	5	5	5	**5**	6	7	8
5	N	5	4	4	4	4	**5**	6	7	7	7
4	E	4	3	4	**3**	**4**	5	6	6	7	8
3	T	3	3	**3**	3	4	5	5	6	7	8
2	N	2	**2**	2	3	4	5	6	7	7	7
1	I	**1**	1	2	3	4	5	6	6	7	8
0	#	**0**	1	2	3	4	5	6	7	8	9
		#	E	X	E	C	U	T	I	O	N
		0	1	2	3	4	5	6	7	8	9

α

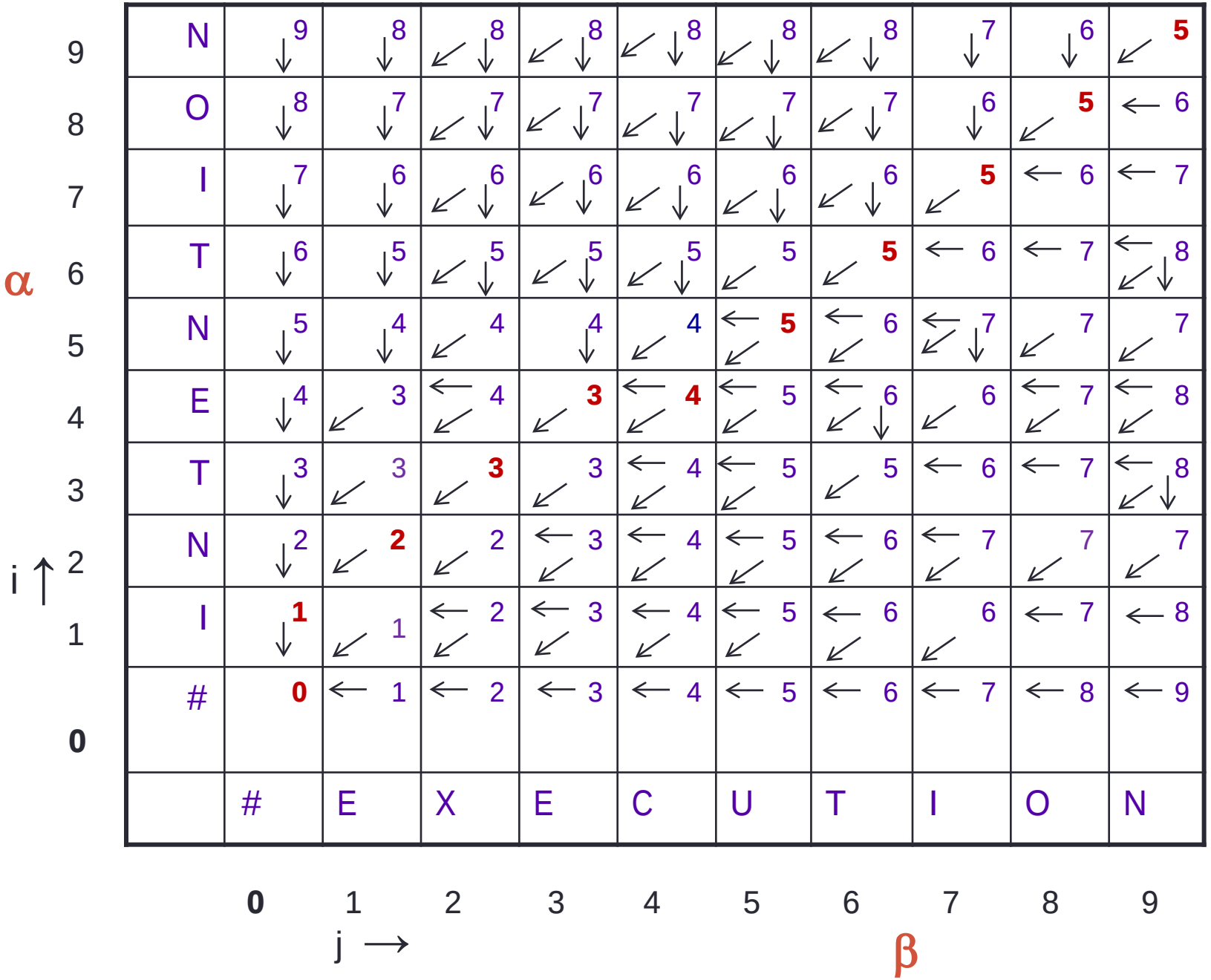
i ↑

j →

β

Cálculo de alineamientos

- La distancia de edición no es suficiente
  - Si necesitamos alinear cada caracter de las dos cadenas
- Podemos hacerlo manteniendo un “backtrace”
- Cada vez que entramos en una celda, recordar desde dónde venimos
- Cuando se alcanza el final,
  - Rastrear el camino desde la esquina superior derecha para leer la alineación


α
i
9
8
7
6
5
4
3
2
1
0
N 9 8 8 8 8 8 8 7 6 5
O 8 7 7 7 7 7 7 6 5 6
I 7 6 6 6 6 6 6 5 6 7
T 6 5 5 5 5 5 5 6 7 8
N 5 4 4 4 4 5 6 7 7 7
E 4 3 4 3 4 5 6 6 7 8
T 3 3 3 3 4 5 5 6 7 8
N 2 2 2 3 4 5 6 7 7 7
I 1 1 2 3 4 5 6 6 7 8
0 1 2 3 4 5 6 7 8 9
E X E C U T I O N
0 1 2 3 4 5 6 7 8 9
j
β

Añadir “Backtrace” a la distancia de edición mínima

- Inicio:

  `D(i,0) = i  D(0,j) = j`

Terminación:

`D(|α| , |β| ) distancia`

- Cálculo de las matrices:

  desde j←1 **hasta** | β | **hacer**

  desde i←1 **hasta** | α | **hacer**

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 1; \text{ if } \alpha(i) \neq \beta(j) \\ 0; \text{ if } \alpha(i) = \beta(j) \end{cases} & \text{substitution} \end{cases}$$

$$ptr(i,j) = \begin{cases} LEFT & \text{insertion} \\ DOWN & \text{deletion} \\ DIAG & \text{substitution} \end{cases}$$

Resultado del “Backtrace”

- Dos cadenas y su alineamiento:

```
I N T E * N T I O N
| | | | | | | | |
* E X E C U T I O N
```

Rendimiento

- Tiempo: O (|α| x |β|)
- Espacio: O (|α| x |β|)
- Backtrace: O (|α| + |β|)