Capítulo 1.

Capítulo 2.

Capítulo 3.

Capítulo 4.

Capítulo 5.
Aplicación de la Estructura S-D a la Recuperación de Información en Archivos Documentales

La localización de palabras en textos constituye una parte fundamental de un problema práctico de gran importancia: la organización, el almacenamiento y los sistemas de recuperación apropiados para el manejo de la información procedente de fuentes heterogéneas.

Existen dos aproximaciones a la localización de palabras en textos. En la primera, se efectúa la búsqueda directamente sobre el documento sin ningún tipo de preparación preliminar; si se trata con actas de longitud mediana o las consultas son algo complejas, esta manera de proceder puede resultar muy costosa en cuanto a tiempo de respuesta. Desde la segunda perspectiva, se someten previamente los escritos a un tratamiento; la intención es generar un índice que aumente la eficacia de los accesos al ejemplar.

Siguiendo este último enfoque, se utiliza como índice la estructura **S-D**, construida a partir del conjunto de las palabras diferentes que se obtienen del documento al descartar las palabras consideradas *vacías*.

El corpus que se va a indizar para llevar a cabo las localizaciones textuales es un Diccionario de Medicina, [JV87].

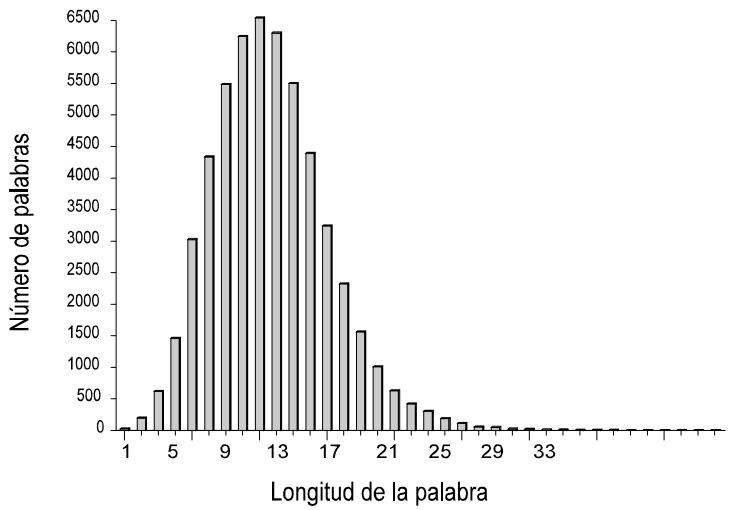

Figura 40

El Diccionario de Medicina constituye un texto formado por un total de 1.054.039 palabras de las cuales 603.152 se consideran *vacías* –584 es la cardinalidad de este conjunto [(1)]– y de las 450.887 restantes, solamente existen 56.297 palabras diferentes. Se considera que la tilde y la diéresis no suponen, a efectos de evaluación de **DIT** y **DL**, costo de edición alguno por tanto no se han tenido en cuenta los signos ortográficos en la construcción del índice, aunque en la respuesta a una petición se localiza la aparición, en el documento original, de todas las palabras con o sin signos ortográficos. En la figura 40, se muestra la distribución de frecuencias por longitudes de las 54.723 palabras diferentes consideradas para construir **S-D**. Los caracteres numéricos y obviamente los signos de puntuación no son considerados palabras y por ello no se admiten en las peticiones.

Se modifican los *nodos_SIT* añadiendo un puntero a cada *sinónimo_DIT*, de forma que señale a una lista –ordenada de menor a mayor– de posiciones en las que aparece esa palabra en el documento original, como se observa en la figura 41.

---

(1) En el Apéndice 1 aparece una lista de las mismas.

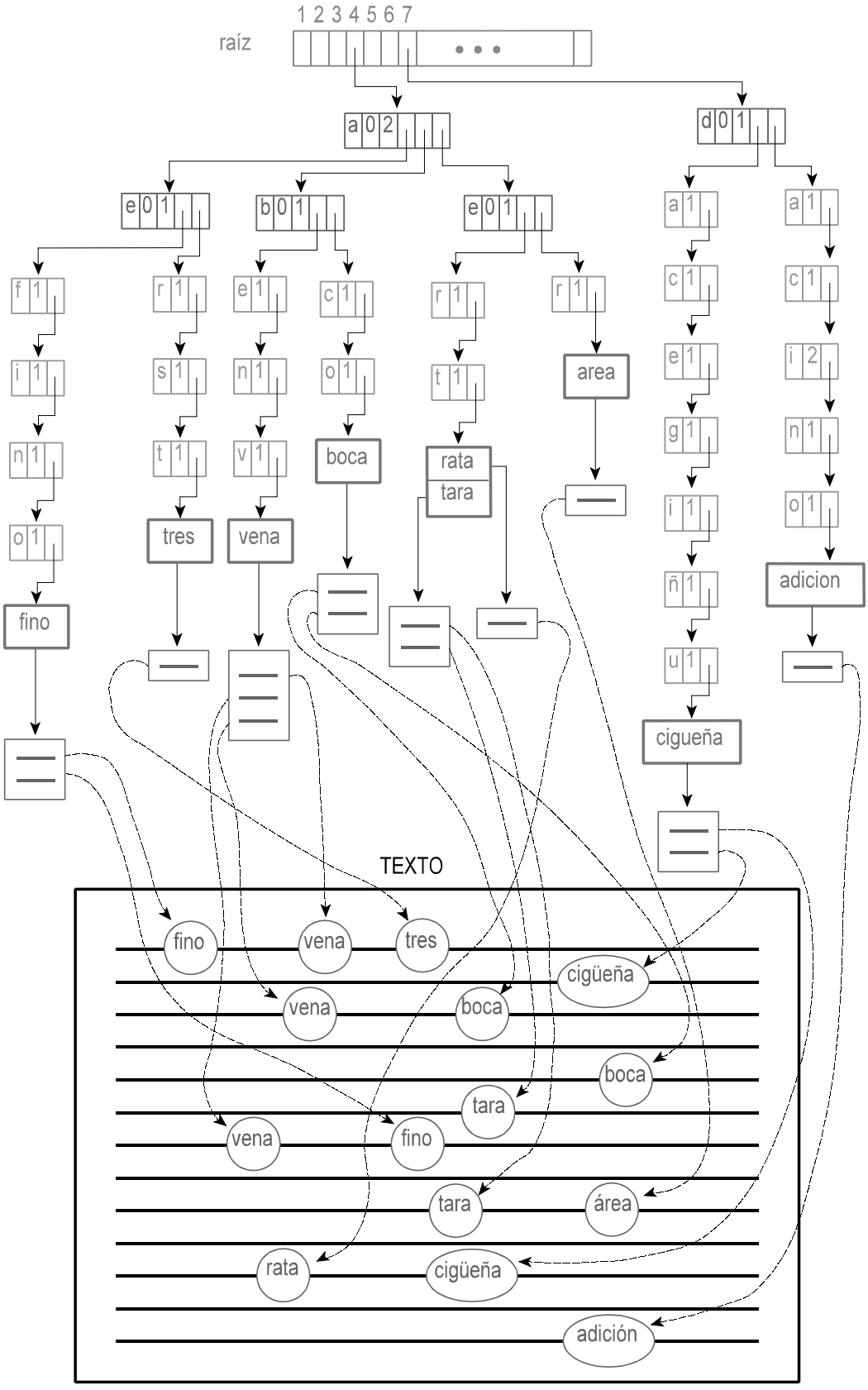
1 2 3 4 5 6 7
raíz
a 0 2
d 0 1
e 0 1
b 0 1
e 0 1
a 1
a 1
f 1
r 1
e 1
c 1
r 1
r 1
c 1
c 1
i 1
s 1
n 1
o 1
t 1
area
e 1
i 2
n 1
t 1
v 1
boca
rata
tara
g 1
n 1
o 1
o 1
tres
vena
i 1
adicion
fino
ñ 1
u 1
cigueña
TEXTO
fino
vena
tres
cigüeña
vena
boca
boca
tara
vena
fino
tara
área
rata
cigüeña
adición

Figura 41

Se estudiarán dos casos: en primer lugar cuando se dispone de todo el índice en memoria interna y a continuación cuando se hace necesaria la paginación de la estructura.

5.1. Tipos de Búsquedas en Archivos Documentales.

Sobre la estructura **S-D** así construida se permiten los siguientes tipos de búsquedas:

— *exacta*

— *más similares*

— *máscaras*

— *truncamientos*: *a la derecha, a la izquierda, a ambos lados*

— *con operadores booleanos*: *o, y, y_no.*

— *cercanía*

— *antecedencia*

— *párrafos*

— *sentencias*

— *frases*

— *compleja*

El Diccionario de Medicina está dividido de forma natural en *artículos* –en general *documentos*– donde cada uno se compone de una entrada y su

definición. La lista de posiciones asociada a cada *sinónimo_DIT* indica los distintos *artículos* en los que está presente. Si una palabra se repite dentro de un mismo *artículo,* su lista de posiciones hace referencia una sola vez a este *artículo.*

Las búsquedas de: *cercanía, antecedencia, párrafos, sentencias* y *frases,* así como las *complejas* que incluyan a alguna de las citadas, requieren el tratamiento de cada *artículo* candidato al conjunto respuesta con el fin de verificar las restricciones especificadas en la petición en cuanto a la situación de las palabras.

Cuando se solicita una búsqueda del tipo: *más similares, máscaras* o *truncamientos,* en las que el usuario no determina directamente las palabras a localizar, se muestran en la pantalla las palabras que satisfacen la petición permitiéndose la selección de un subconjunto de ellas; se funden entonces las listas de posiciones asociadas a las palabras seleccionadas para posteriormente visualizar los *artículos* correspondientes.

5.2. Búsqueda Exacta.

La búsqueda más elemental consiste en determinar todas las localizaciones de una palabra en el documento. La petición se expresa simplemente indicando cuál es la que se solicita. Por ejemplo: ***sida***, aparece en ocho *artículos.*

Algoritmo de búsqueda exacta:

```
Procedimiento Exacta (nodo)
{Obtiene la lista de posiciones asociada a la palabra de búsqueda}
    B: palabra de búsqueda
    encontrado: es falso cuando se invoca a este procedimiento; si se localiza B, se
    actualiza a verdadero
    frec: frecuencia en B del carácter asociado a nodo
si nodo ≠ nulo entonces
    si nodo es un nodo_SIT entonces
        j=1
        mientras (j≤nodo.ns) y (no encontrado) hacer
            si B = nodo.s(j) entonces
                encontrado=verdadero
                obtener la lista de posiciones asociada a nodo.s(j)
            si no
                j=j+1
            fin si
        fin mientras
    si no
        si es un nodo de la parte cadena entonces
            si vf(nodo.c) = nodo.f entonces
                Exacta(nodo.e)
            fin si
        si no  {es un nodo de la parte_árbol}
            frec=vf(nodo.cd)
            si (nodo.pp≤frec) y (frec≤nodo.pu) entonces Exacta(nodo.e(frec)) fin si
        fin si
    fin si
fin si
```

5.3. Búsqueda de las Palabras Más Similares.

La recuperación de las palabras *más similares* a una dada se realiza con el esquema de búsqueda *creciente* cuando se dispone de todo el índice en memoria interna y con el *decreciente* cuando se trabaja con la estructura paginada, ya que, en general, son los que presentan mejor rendimiento para cada una de esas situaciones.

El formato de la petición es: ***+cadena***. Por ejemplo, al solicitar ***+rida*** se obtienen: *ría, vida, rica, risa, sida, oída, brida.*

5.4. Búsqueda con Máscaras.

Si se desea hacer una búsqueda de una palabra en la que se desconocen caracteres en determinadas posiciones, se puede formular una petición en la que se sustituyan los caracteres desconocidos por *comodines*, *.

En una búsqueda con *máscaras*, se recuperan todos los *artículos* en los que se encuentra una palabra de longitud igual a la especificada y que difiera solamente en los caracteres indicados por los asteriscos. Por ejemplo para ***t*m*r*** se recuperan los *artículos* en los que aparecen las palabras *tumor, temor, tomar*.

Una búsqueda con *máscaras* se lleva a cabo explorando el subárbol correspondiente a la longitud de la petición, teniendo como objetivo inicial la recuperación de todas las palabras cuya **DIT** con la palabra de búsqueda es igual al número de asteriscos (**DTM**=número de asteriscos).

El vector de frecuencias, **vf**, contiene la frecuencia de cada carácter alfabético especificado en la petición y cero para cualquier otro. En cada nodo de la *parte_árbol*, la frecuencia en **vf** del carácter asociado al nodo define el encadenamiento de descenso inicial; las alternativas se toman a partir de este punto sólo hacia la derecha, mientras no se supere el umbral **DTM**. Cada vez que se alcanza un *nodo_SIT*, se comprueba si cada sinónimo tiene los caracteres especificados en la petición en las posiciones correspondientes.

Algoritmo de búsqueda con máscaras:

```
Procedimiento Máscara (B)
{Realiza una búsqueda con máscaras}
    B: cadena de búsqueda, formada por caracteres del alfabeto y asteriscos
    vf(.): vector de frecuencias de los caracteres de B, excluyendo los asteriscos
    DTM: número de asteriscos de la petición
    nms: número de palabras que verifican la petición
para c='a' hasta 'z' hacer
    vf(c)=0
fin para
DTM=0
para i=1 hasta │B│ hacer
    si B<i> = '*' entonces
        DTM=DTM+1
    si no
        vf(B<i>)=vf(B<i>)+1
    fin si
fin para
nms=0
cadit=0
Masc_pa(raiz(│B│),cadit,│B│-DTM)
```

```
Procedimiento Masc_pa (nodo,cadit,cbnt)
{Explora la parte_árbol de la estructura S-D}
    nodo: nodo actual
    cadit: componentes acumuladas de DIT
    cbnt: número de caracteres de B no tratados
    frec: frecuencia en B del carácter asociado a nodo
    dfm: diferencia de frecuencias máxima
    finf, fsup: frecuencia mínima y máxima alcanzadas en la selección de alternativas
    falt: frecuencia alternativa
    vcadit: valor de las componentes de DIT para la alternativa actual
    ditm: DIT mínima utilizada en la poda
si nodo ≠ nulo entonces
    si nodo es un nodo_SIT entonces
        cadit=cadit+cbnt
        si cadit ≤ DTM entonces
            Masc_nS(nodo)
        fin si
    si no
        si es un nodo de la parte cadena entonces
            Masc_pc(nodo,cadit,cbnt)
        si no  {es un nodo de la parte_árbol}
            frec=vf(nodo.cd)
            cbnt=cbnt-frec
            dfm=DTM-cadit
            finf=frec
            fsup=frec+dfm
            si finf < nodo.pp entonces
                finf=nodo.pp
            fin si
            si fsup > nodo.pu entonces
                fsup=nodo.pu
            fin si
            para falt=finf hasta fsup hacer
                vcadit=cadit+abs(frec-falt)
                Masc_pa(nodo.e(falt),vcadit,cbnt)
            fin para
        fin si
    fin si
fin si
```

```
Procedimiento Masc_nS (nodo)
{Trata los nodos_SIT}
para j=1 hasta nodo.ns hacer
    i=1
    igualcar=verdadero
    mientras (i≤ |B| ) y igualcar hacer
        si (B<i>≠'*') y (B<i>≠nodo.s(j)<i>) entonces
            igualcar=falso
        si no
            i=i+1
        fin si
    fin mientras
    si i > |B| entonces
        añadir nodo.s(j) al conjunto de palabras que verifican la petición
        nms=nms+1
    fin si
fin para


Procedimiento Masc_pc (nodo,cadit,cbnt)
{Examina la parte_cadena}
si nodo es un nodo_SIT entonces
    cadit=cadit+cbnt
    si cadit ≤ DTM entonces
        Masc_nS(nodo)
    fin si
si no
    cadit=cadit+abs(nodo.f-vf(nodo.c))
    cbnt=cbnt-vf(nodo.c)
    si cadit ≤ DTM entonces
        Masc_pc(nodo.e,cadit,cbnt)
    fin si
fin si
```

5.5. Búsquedas con Truncamientos.

Las búsquedas con *truncamientos: a la derecha, a la izquierda* y *a ambos lados* permiten recuperar *artículos* que contienen palabras que empiezan, terminan o presentan alguna coincidencia central con la cadena de caracteres sobre el alfabeto, *cadena*, explicitada en la petición.

Cada tipo de *truncamiento* se expresa de la siguiente forma:

a la derecha: ***cadena!***

a la izquierda: ***!cadena***

a ambos lados: ***!cadena!***

Por ejemplo:

a la derecha: ***tos!***, se obtienen: *tos, tose, tosa, toser, tosco, tosca, toscos, toscas, tostada.*

a la izquierda: ***!tipo***, se recuperan: *subtipo, ojotipo, idiotipo, optotipo, fenotipo, genotipo, serotipo, fagotipo, prototipo, cariotipo, inmunotipo, somatotipo.*

a ambos lados: ***!cubo!***, se localizan: *cubo, cuboide, cuboides, cuboideo, cuboidal, cuboidea, cuboideum, cuboideos, cuboideas, calcaneocuboidea, calcaneocuboideo, calcaneocuboideas.*

Para realizar la búsqueda correspondiente a cualquiera de los tres tipos de *truncamientos* se examinan las palabras con longitud igual o mayor a la cadena especificada en la petición. La exploración de cada longitud se realiza aprovechando el procedimiento de búsqueda con *máscaras* –donde el número de comodines depende de la diferencia de longitudes– y modificando el tratamiento de los *nodos_SIT*.

Algoritmo de búsqueda con truncamiento a la derecha:

```
Procedimiento Truncamiento (B)
    B: cadena de búsqueda especificada en la petición
    vf(.): vector de frecuencias de la cadena de búsqueda
    DTM: diferencia entre la longitud actual y la de la cadena de búsqueda
para c='a' hasta 'z' hacer
    vf(c)=0
fin para
para i=1 hasta |B| hacer
```

```
    vf(B<i>)=vf(B<i>)+1
fin para
DTM=0
nms=0
para lalt=|B| hasta longmax hacer
    cadit=0
    Masc_pa(raiz(lalt),cadit,|B|)
    DTM=DTM+1
fin para
```

El procedimiento **Masc_pa**, en este caso, en vez de invocar al procedimiento **Masc_nS** llama al **Truncder_nS** que selecciona los *sinónimos_DIT* que poseen la cadena de búsqueda como prefijo.

```
Procedimiento Truncder_nS (nodo)
{Trata los nodos_SIT en una búsqueda con truncamiento a la derecha}
para j=1 hasta nodo.ns hacer
    i=1
    mientras (i≤|B|) y (B<i>=nodo.s(j)<i>) hacer
            i=i+1
    fin mientras
    si i > |B| entonces
        añadir nodo.s(j) al conjunto de palabras que verifican la petición
        nms=nms+1
    fin si
fin para
```

Algoritmo de búsqueda con truncamiento a la izquierda:

El algoritmo de *truncamiento a la izquierda* sólo se diferencia del anterior en que invoca al procedimiento **Truncizq_nS** –selecciona los *sinónimos_DIT* que poseen la cadena de búsqueda como subfijo– en vez de al **Truncder_nS**.

```
Procedimiento Truncizq_nS (nodo)
{Trata los nodos_SIT en una búsqueda con truncamiento a la izquierda}
para j=1 hasta nodo.ns hacer
    i=|B|
    mientras (i>0) y (B<i>=nodo.s(j)<lalt-|B|+i>) hacer
            i=i-1
    fin mientras
```

```
    si i = 0 entonces
        añadir nodo.s(j) al conjunto de palabras que verifican la petición
        nms=nms+1
    fin si
fin para
```

Algoritmo de búsqueda con truncamiento a ambos lados:

Se utiliza el procedimiento **Truncamiento** con un nuevo tratamiento de los *nodos_SIT*, **Truncamb_ns**; selecciona los *sinónimos_DIT* que poseen la cadena de búsqueda como infijo –busca la primera coincidencia entre el sinónimo y la cadena de búsqueda.

```
Procedimiento Truncamb_nS (nodo)
{Trata los nodos_SIT en una búsqueda con truncamiento a ambos lados}
para j=1 hasta nodo.ns hacer
    t=lalt-|B|
    r=0
    valido=falso
    mientras (r≤t) y (no valido) hacer
        i=1
        mientras (i≤|B|) y (B<i>=nodo.s(j)<r+i>) hacer
            i=i+1
        fin mientras
        si i > |B| entonces
            añadir nodo.s(j) al conjunto de palabras que verifican la petición
            nms=nms+1
            valido=verdadero
        si no
            r=r+1
        fin si
    fin mientras
fin para
```

5.6. Búsquedas con Operadores Booleanos.

Se pueden realizar búsquedas que combinen varias cadenas utilizando como conectores los operadores lógicos: *o*, *y*, *y_no*.

5.6.1. Búsqueda Disyuntiva.

Una búsqueda *disyuntiva* proporciona todos los *artículos* en los que aparece al menos una de las palabras solicitadas en la petición. Se solicita de la siguiente forma: ***cadena1 o cadena2***, donde *cadena1* y *cadena2* son peticiones *exactas, más similares, máscaras* o *truncamientos.* Por ejemplo: ***cáncer o +tos***.

Este proceso se realiza invocando al procedimiento de búsqueda correspondiente a cada petición especificada uniendo posteriormente las listas de posiciones asociadas.

Algoritmo de unión de las listas de posiciones:

```
Procedimiento Unión (list1,list2,long1,long2)
{Une dos listas de posiciones}
    long1, long2: longitudes de las listas de posiciones
    list1(.), list2(.): listas de posiciones que se van a unir
    list3(.): lista en la que se almacena ordenadamente la fusión
    nrd: número de elementos en list3
i=1; j=1; nrd=0
mientras (i≤long1) y (j≤long2) hacer
    si list1(i) < list2(j) entonces
        nrd=nrd+1
        list3(nrd)=list1(i)
        i=i+1
    si no
        nrd=nrd+1
        list3(nrd)=list2(j)
        si list1(i) = list2(j) entonces i=i+1 fin si
        j=j+1
    fin si
fin mientras
mientras i ≤ long1 hacer
    nrd=nrd+1
    list3(nrd)=list1(i)
    i=i+1
fin mientras
mientras j ≤ long2 hacer
    nrd=nrd+1
    list3(nrd)=list2(j)
    j=j+1
fin mientras
```

Análisis del algoritmo en el peor caso:

La cantidad total de tiempo que consume este algoritmo está en relación directa con el número de comparaciones que realiza. El peor caso ocurre cuando para situar cada elemento de la segunda lista en la de fusión se requieren dos comparaciones y para situar cada elemento de la primera, excepto el último, es necesaria una comparación. Esta situación se verifica cuando las posiciones de las dos listas a fundir son todas distintas y no se agota ninguna de ellas hasta el último instante. Si se desean unir **s** listas, se invoca al procedimiento para las dos primeras y se mezcla cada una de las *s-2* restantes con el resultado previo. El número máximo de comparaciones en función del número de listas se muestra en la tabla 4 –se denota por $\mathbf{n_i}$ la longitud de la **i**-ésima lista– donde se ha supuesto sin pérdida de generalidad que la primera lista se corresponde con la fusión previa y la segunda con la nueva lista a mezclar.

Número de listas	Número máximo de comparaciones
2	n_1+2*n_2-1
3	$2*n_1+3*n_2+2*n_3-2$
4	$3*n_1+4*n_2+3*n_3+2*n_4-3$
5	$4*n_1+5*n_2+4*n_3+3*n_4+2*n_5-4$
...	...
s	$(s-1)*n_1+s*n_2+(s-1)*n_3+...+2*n_s-(s-1)$

Tabla 4

Dado que:

$$(s-1)*n_1+s*n_2+(s-1)*n_3+...+2*n_s-(s-1) < s*(n_1+n_2+n_3+...+n_s) < s^2*n$$

siendo

$$n = máx\{n_1, n_2, n_3,..., n_s\} \text{ y } s \geq 2$$

se tiene que la complejidad de cálculo en el peor caso es $O(s^2*n)$.♦

El análisis realizado indica que para obtener un mejor rendimiento es conveniente fundir las listas en orden creciente de longitudes –los coeficientes que afectan a las longitudes son decrecientes a partir de la segunda lista.

Se ha obtenido que las listas de posiciones de la estructura tienen una longitud promedio de 7; la máxima que asciende a 724 y está asociada a la palabra *sangre*.

5.6.2. Búsqueda Conjuntiva.

La respuesta de una petición *conjuntiva* está formada por aquellos *artículos* que contienen una de las palabras solicitadas en la primera petición y otra de la segunda. Esta petición se formula como: ***cadena1 y cadena2***, donde *cadena1* y *cadena2* son peticiones *exactas*, *más similares*, *máscaras* o *truncamientos*. Por ejemplo: ***cáncer y tos!***.

Esta búsqueda se lleva a cabo realizando la intersección de las listas de posiciones que proporciona cada una de las peticiones especificadas.

Algoritmo de intersección de las listas de posiciones:

```
Procedimiento Intersección (list1,list2,long1,long2)
{Realiza la intersección de dos listas de posiciones}
    long1, long2: longitudes de las listas de posiciones
    list1(.), list2(.): listas de posiciones a intersectar. El resultado se almacena en list1
    nrc: número de elementos de la intersección
j=1; nrc=0
mientras (i≤long1) y (j≤long2) hacer
    si list1(i)>list2(j) entonces
        j=j+1
    si no
        si list1(i) = list2(j) entonces
            nrc=nrc+1
            list1(nrc)=list1(i)
            j=j+1
```

```
        fin si
        i=i+1
    fin si
fin mientras
long1=nrc
```

Análisis del algoritmo en el peor caso:

De forma análoga al anterior, el tiempo que requiere este algoritmo está en relación directa con el número de comparaciones. Cuando todos los elementos de la primera lista están situados al final de la segunda, se alcanza el peor caso, ya que se realiza una comparación para cada elemento no común de la segunda lista y dos comparaciones para los comunes. La intersección de **s** listas se lleva a cabo invocando al procedimiento para las dos primeras y cada una de las *s-2* restantes se intersectan con el resultado previo. La tabla 5 muestra el número máximo de comparaciones en función del número de listas a tratar.

Número de listas	Número máximo de comparaciones
2	n_2+n_1
3	$n_2+n_3+2*n_1$
4	$n_2+n_3+n_4+3*n_1$
5	$n_2+n_3+n_4+n_5+4*n_1$
...	...
s	$n_2+n_3+...+n_s+(s-1)*n_1$

Tabla 5

Dado que:

$$n_2+n_3+...+n_s+(s-1)*n_1 < (s-1)*n+(s-1)*n < 2*(s-1)*n < 2*s*n$$

siendo

$$n = máx\{n_1, n_2, n_3,..., n_s\} \text{ y } s \geq 2$$

se tiene que la complejidad de cálculo en el peor caso es $O(s*n)$.♦

El análisis refleja que si la primera lista es la más corta se obtiene un mejor rendimiento –es la única longitud afectada por un coeficiente mayor que uno.

5.6.3. Búsqueda con Operador *y_no*.

En una búsqueda de una petición de la forma ***cadena1 y_no cadena2*** se obtienen aquellos *artículos* en los que se encuentra alguna palabra solicitada en la petición situada a la izquierda del operador y que a su vez no contienen a ninguna palabra de las solicitadas a la derecha. Por ejemplo: ***tos y_no fiebre***.

La forma de conseguir los *artículos* deseados consiste en eliminar de la lista de posiciones de la izquierda las que se encuentren en la lista de la derecha.

Algoritmo de diferencia de listas de posiciones:

```
Procedimiento Diferencia (list1,list2,long1,long2)
{Elimina de list1 las posiciones que se encuentren en list2}
    long1, long2: longitudes de las listas de posiciones
    list1(.), list2(.): listas de posiciones a intersectar. El resultado se almacena en list1
    nrc: número de elementos de la intersección
i=1; j=1; nrc=1
mientras (i≤long1) y (j≤long2) hacer
    si list1(i)>list2(j) entonces
        j=j+1
    si no
        si list1(i) ≠ list2(j) entonces
            list1(nrc)=list1(i)
            nrc=nrc+1
        si no
            j=j+1
        fin si
        i=i+1
    fin si
fin mientras
mientras i ≤ long1 hacer
    list1(nrc)=list1(i)
    nrc=nrc+1
    i=i+1
fin mientras
```

long1=nrc

Análisis del algoritmo para el peor caso:

De forma análoga a los anteriores, el tiempo que requiere este algoritmo está en relación directa con el número de comparaciones. Sean $\mathbf{n_1}$ y $\mathbf{n_2}$ las longitudes respectivas de las listas. Si $n_1 \geq n_2$, cuando todos los elementos de la segunda lista están situados al final de la primera, se alcanza el peor caso, ya que se realizan dos comparaciones por cada elemento de la primera lista, $2*n_1$; si $n_1 < n_2$, se alcanza el peor caso cuando los elementos de la primera lista se encuentran situados al final de la segunda, ya que se realiza una comparación para los elementos no comunes de la segunda lista y dos para los comunes, $n_2-n_1+2*n_1<2*n_2$. En cualquier caso, el número de comparaciones está acotado por $2*n$, siendo $n=máximo\{n_1,n_2\}$, por tanto la complejidad de cálculo, en el peor caso, es $O(n)$.♦

5.7. Búsquedas de Cercanía y Antecedencia.

Se denominan búsquedas con condiciones topológicas ya que delimitan el *entorno,* medido en número de palabras, en el que se tienen que encontrar las palabras especificadas en la petición.

5.7.1. Cercanía.

La búsqueda de *cercanía* se caracteriza por limitar el número máximo de palabras que pueden existir entre las dos especificadas. Las

peticiones se expresan como: ***cadena1 c/n cadena2,*** donde *cadena1* y *cadena2* deben ser peticiones *exactas* y **n** es un entero positivo que indica la posición más alejada posible desde una cadena a la otra. Por ejemplo: ***fiebre c/5 aguda***, seleccionará aquellos *artículos* en los que se han localizado *fiebre* y *aguda*, verificándose además que entre ellas existen como mucho *n-1* palabras.

Este proceso se realiza eliminando aquellos *artículos* que no cumplen las condiciones de *cercanía* de la lista generada por la búsqueda *conjuntiva* de las dos palabras especificadas.

5.7.2. Antecedencia.

La *antecedencia* presenta una restricción adicional con respecto a la búsqueda anterior ya que establece además el orden de aparición entre las palabras. Las peticiones tienen la forma: ***cadena1 a/n cadena2***. Por ejemplo: ***nervio a/3 facial***, selecciona los *artículos* en los que aparece *nervio* seguida de *facial* y entre ellas no hay más de dos palabras.

5.8. Búsquedas en Párrafos y Sentencias.

En las búsquedas anteriores, el *artículo* es el ámbito de búsqueda por defecto, se puede restringir dicho ámbito a un *párrafo* o a una *sentencia*..

5.8.1. Párrafos.

La búsqueda en *párrafos* es una alternativa a la de *cercanía* entre palabras en la que se requiere que las palabras especificadas se encuentren en el mismo *párrafo*. Se solicita de la siguiente forma: ***cadena1 p/ cadena2***. Por ejemplo: ***pies p/ manos***, selecciona aquellos *artículos* en los que aparecen, en un mismo *párrafo*, las palabras *pies* y *manos*.

Este proceso se lleva a cabo de forma análoga a la búsqueda de *cercanía*.

5.8.2. Sentencias.

Es como la búsqueda en *párrafos* pero restringida a una *sentencia*. La petición se expresa: ***cadena1 s/ cadena2***. Por ejemplo: ***pies s/ manos***, selecciona aquellos *artículos* en los que aparecen, en la misma *sentencia*, las palabras *pies* y *manos*.

5.9. Búsqueda de Frases.

El objetivo de la búsqueda de *frases* es la localización de los *artículos* en los que se encuentra la *frase* especificada. La *frase* se denota entre comillas. Por ejemplo: "***de pies y manos***".

Este proceso se lleva a cabo mediante una búsqueda *conjuntiva* de las palabras no *vacías* de la *frase* y la ulterior comprobación de la existencia de toda la *frase* en los *artículos* preseleccionados.

5.10. Búsqueda Compleja.

Cualquier combinación de los anteriores tipos de búsqueda en una expresión utilizando los conectores lógicos –*y, o, y_no*– es a lo que se denomina una petición compleja. El orden de prioridad de los conectores es de izquierda a derecha. Si se desea alterar esta prioridad, se deben utilizar paréntesis y en el caso de que existan paréntesis anidados, los más internos son los que se resolverán en primer lugar. Por ejemplo: ***s**a y ((fiebre c/2 aguda) o tos!) y_no sana***.

La obtención de la respuesta de una búsqueda compleja con paréntesis anidados se realiza resolviendo las expresiones desde las más profundas hacia afuera. A partir del paréntesis abierto, se actúa de izquierda a derecha ejecutando cada búsqueda *no_booleana* y resolviendo los *operadores lógicos* una vez que se alcanza el paréntesis cerrado. Por defecto, se considera que la petición *compleja* está encerrada entre paréntesis.

5.10.1. Búsqueda en la que intervienen Peticiones Anteriores.

Iniciada una sesión de consultas y tras haber realizado una serie de peticiones, es posible modificar una petición *previa* para confeccionar otra más *compleja*. El formato ***@n*** indica que la **n**-ésima consulta realizada forma parte de la nueva petición. Supóngase que se desea añadir a la quinta petición, cuya expresión de contenido es: ***cáncer y tumor***, un nuevo término: ***y_no pulmón***, la nueva petición se escribe de la siguiente forma: ***@5 y_no pulmón***.

También se pueden hacer *combinaciones de peticiones previas* sin añadir ningún término nuevo, utilizando los *operadores booleanos*. Por ejemplo: ***@2 y (@4 o @6)***, combina los resultados de las consultas segunda, cuarta y sexta.

5.11. Analizador Sintáctico de la Petición.

El *analizador sintáctico* cumple un doble cometido, ya que ha de identificar cada tipo de petición –diferenciando las componentes y los *conectores lógicos* en el caso de las *complejas*– y, a la vez, determinar la *correctitud sintáctica.*

Se consideran erróneas las siguientes situaciones:

— Si alguna petición *no_booleana* presenta un formato incorrecto. Ya sea porque se especifica una búsqueda no permitida –por ejemplo: ***t*m!***, ***+rida c/9 tos!***–; o bien porque la sintaxis es incompleta –por ejemplo: si se detecta la presencia de **/** precedida de **a** o **c** y no le sigue un valor entero o se especifica una *frase* y falta cerrar las comillas, etc.

— Si se utilizan *conectores no permitidos*.

— Si después de un *operador lógico* no existe una búsqueda *no_booleana* o un paréntesis abierto.

— Si se especifica la búsqueda *exacta* de una palabra considerada como *vacía*.

— Si existe un paréntesis abierto para el que no se encuentra el correspondiente paréntesis cerrado.

En caso de que se detecte un error del usuario, se informará al mismo de ello y se señalará en la pantalla la posición en la que se ha encontrado el fallo.

Para *analizar* la petición se recorre de izquierda a derecha identificando las diversas búsquedas *no_booleanas*, *operadores lógicos* y paréntesis, comprobándose que no ocurre ninguna de las situaciones indicadas anteriormente.

Una vez que se ha verificado que la petición es *sintácticamente correcta* e identificado sus componentes, se pasa a la fase de ejecución de la búsqueda.

5.12. Optimizador de la Petición.

Se construye un *optimizador* como complemento al *analizador*. La idea es llegar a la solución de una búsqueda *compleja* en el menor tiempo posible. Se pretende, siempre que sea factible, calcular una *petición equivalente* a la solicitada por el usuario de forma que, aunque los resultados obtenidos a partir de ambas sean los mismos, esta última *petición* se resuelva de forma más rápida, minimizando así el tiempo de respuesta del sistema.

Por una parte existen los procedimientos comentados con anterioridad que agilizan la ejecución de peticiones del tipo:

cadena1 o cadena2 o cadena3 o cadena4

cadena1 y cadena2 y cadena3 y cadena4

seleccionando la lista de posiciones más corta para comenzar la *unión* o la *intersección* de las mismas.

Adicionalmente se contempla la simplificación de aquellas peticiones en las que es aplicable la propiedad *distributiva* –teniendo en cuenta la *conmutatividad*–, por ejemplo:

(cadena1 y cadena2) o (cadena3 y cadena1)

se puede reducir a:

cadena1 y (cadena2 o cadena3)

y

(cadena1 o cadena2) y (cadena3 o cadena1)

se puede reducir a:

cadena1 o (cadena2 y cadena3)

donde los términos *cadena1*, *cadena2* y *cadena3* denotan cualquier tipo de búsqueda permitida.

5.13. Resultados Experimentales. Ubicación del Índice en Memoria Interna.

La ocupación del Diccionario de Medicina –1.054.039 palabras– es de 7.708.577 Bytes; la estructura **S-D** correspondiente ocupa 2.666.898 Bytes[(2)]; y las listas de posiciones, asociadas a cada una de las 54.273 palabras del índice, 1.550.740 Bytes.

El objetivo que se persigue es estudiar el comportamiento de la estructura **S-D** frente a las peticiones de respuesta múltiple, es decir, *truncamientos*, *máscaras* y *más similares*. Se mide, en función de la longitud de la cadena especificada, el tiempo requerido para cada tipo de búsqueda y, en cada caso, el número de palabras que verifican la petición –cardinalidad de la respuesta. El tiempo de búsqueda se mide desde el momento en que el usuario realiza la petición hasta que se muestran en pantalla las palabras que la satisfacen.

Inicialmente se comparan los tres tipos de *truncamientos*. Se generan 18 ficheros de 50 cadenas cada uno, desde longitud 2 hasta 19, para cada uno de los tipos de *truncamiento* con el fin de ser utilizados como argumento de búsqueda. Se elige una palabra al azar y se sortea entre uno y

---

(2) Un carácter ocupa 1 Byte, un entero 4 Bytes y un puntero 4 Bytes.

la mitad de su longitud para obtener el número de caracteres a eliminar; la longitud restante y la posición a partir de la que se extraen los caracteres (al principio, al final o a ambos lados) indican el fichero que le corresponde. En el caso del *truncamiento a ambos lados*, se ha de seleccionar el número de caracteres a eliminar por el principio y por el final, de tal forma que no coincida con ninguno de los otros dos tipos de *truncamiento*.

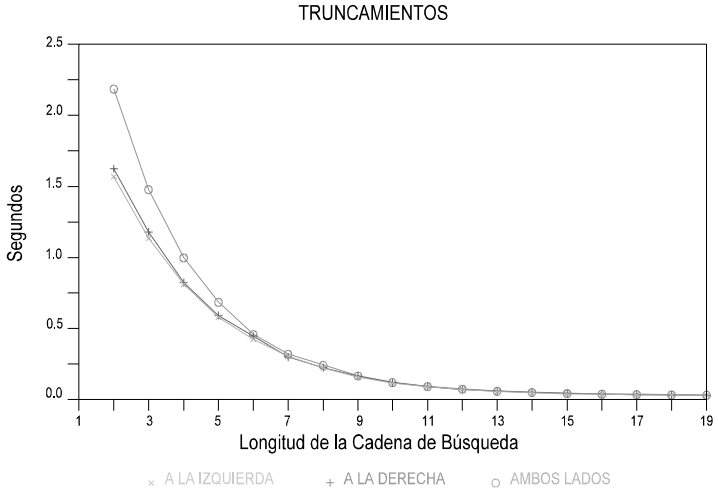

Figura 42

☞ El tiempo requerido para la búsqueda de los distintos tipos de *truncamientos*, figura 42, decrece al aumentar la longitud de la cadena de búsqueda debido a la disminución correlativa de la estructura que ha de ser explorada.

☞ El *truncamiento a ambos lados* presenta un tiempo ligeramente superior a los otros dos –prácticamente coincidentes– debido a que requiere un tratamiento algo más costoso de los *sinónimos_DIT*, cuya importancia crece en función del número de *nodos_SIT* visitados; la diferencia se acorta al crecer la longitud de la cadena de búsqueda, figura 42.

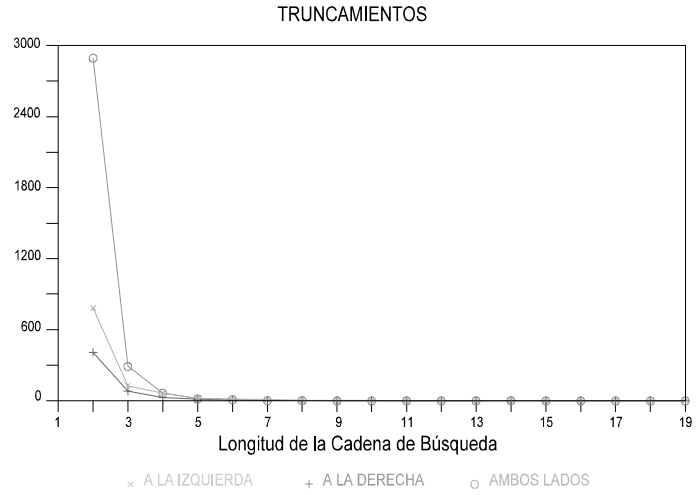

Figura 43

☞ La multiplicidad de la respuesta, figura 43, es significativa para longitudes menores que 10. De forma análoga al tiempo de respuesta, la cardinalidad decrece asintóticamente –hacia una única respuesta.

En el estudio de la búsqueda con *máscaras*, se utilizan 15 ficheros de 50 cadenas cada uno, de longitud 3 a 17. Para crearlos, se elige una palabra al azar –su longitud indica el fichero al que se va a asignar– se sortea el número de asteriscos entre uno y la mitad de su longitud, y por último, se seleccionan aleatoriamente las posiciones de éstos.

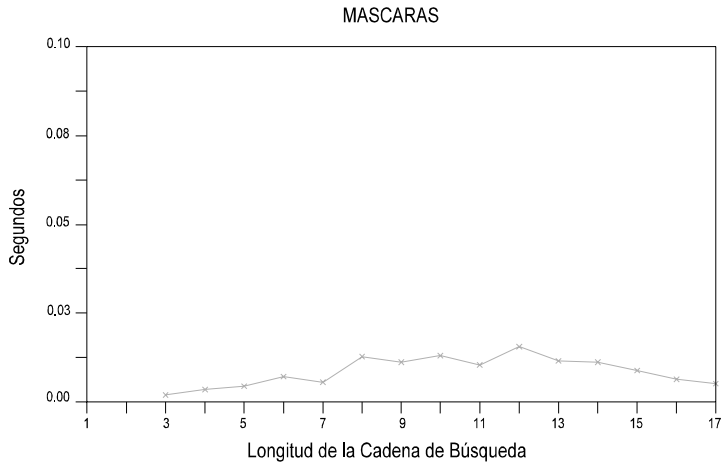

Figura 44

- ☞ La búsqueda con *máscaras* es la más rápida de los tres tipos de búsqueda con respuesta múltiple, puesto que realiza una exploración de la estructura limitada al subárbol correspondiente a la longitud de la cadena de búsqueda.
- ☞ El tiempo requerido en la búsqueda con *máscaras* está influido por el número de palabras cuya longitud coincide con la de la cadena de búsqueda como se muestra en la figura 44.

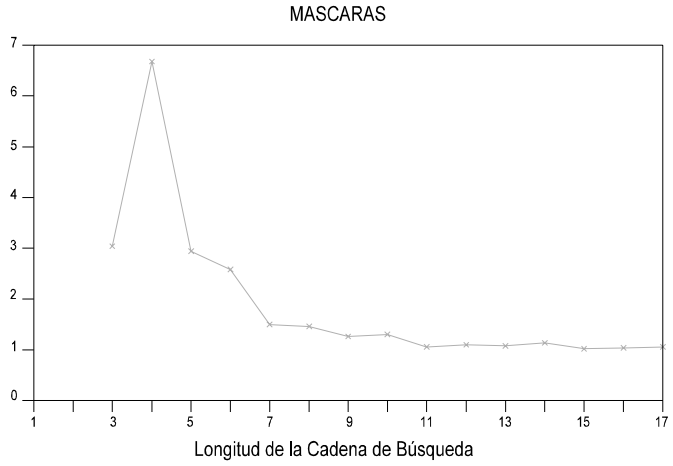

Figura 45

- ☞ La cardinalidad de la respuesta, figura 45, es mayor para pequeñas longitudes, haciéndose asintótica a uno con el incremento de la longitud.

Para la búsqueda de *más similares*, se selecciona al azar una palabra del Diccionario de Medicina y se transforma aplicándole diversas operaciones de edición, elegidas aleatoriamente, hasta obtener una cadena con una **DL** determinada –*DL=2*, *DL=4* o *DL=6*– respecto a la palabra original. **DL** es como máximo igual a la mitad de la longitud de la palabra correcta. La longitud de la palabra distorsionada y el valor de **DL** indican el fichero al que se asigna. Cada fichero contiene 50 cadenas.

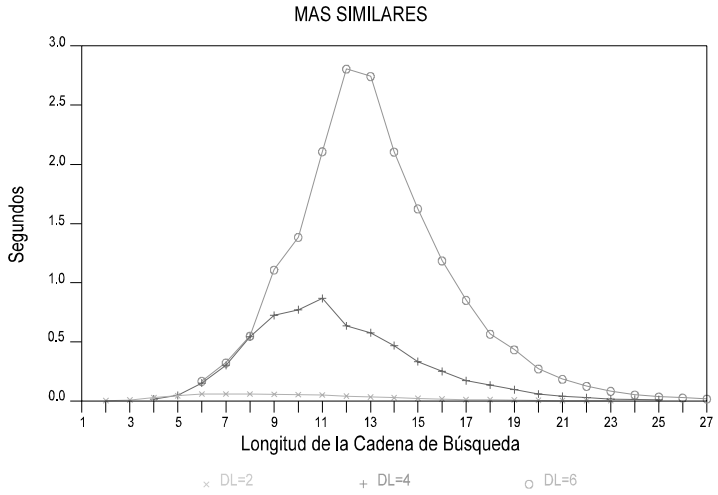

Figura 46

☞ La búsqueda de *más similares* consume un tiempo que está directamente relacionado con la parte de la estructura explorada y con el cálculo de **DL**. Las diferencias para los distintos valores de **DL** se deben tanto a la parte de la estructura implicada como al número de **DL** evaluadas, figura 46.

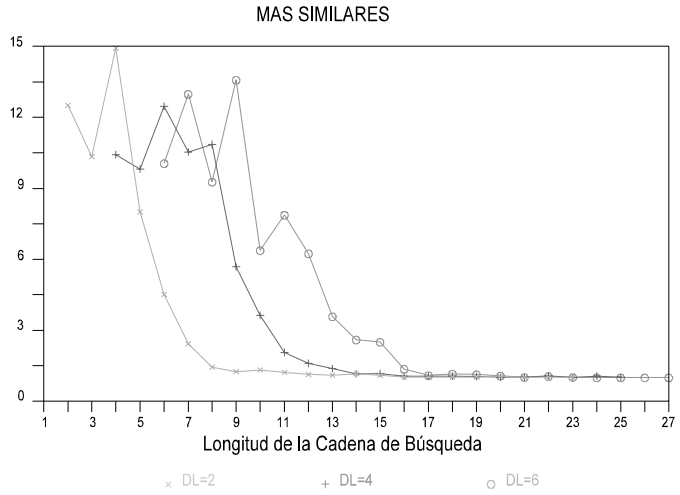

Figura 47

☞ La cardinalidad de la respuesta, figura 47, está determinada por la relación entre el valor de **DL** y la longitud de la cadena de búsqueda –encuentra un mayor número de palabras *más similares* a la cadena buscada para longitudes pequeñas y para mayores valores de **DL**–; sin embargo, el tiempo de búsqueda, figura 46, no depende de la cardinalidad de la respuesta sino del tiempo empleado en recorrer la estructura y en calcular **DL** –que no tiene relación significativa con la cardinalidad de la respuesta.

5.14. Resultados Experimentales. Índice Paginado.

Esta aplicación de la estructura **S-D** a la recuperación de información en archivos documentales se implanta en un **PC** 486 a 33mhz con disco a 19ms para hacerla accesible a un mayor número de usuarios. Para ello se utiliza la paginación en *preorden*, porque tal y como se ha comprobado es la más adecuada para el índice, con un *tamaño de página* de 4K –se obtiene una **S-D** de 538 páginas[(3)]– y un *tamaño de buffer* de 32K. La búsqueda de las palabras *más similares* se realiza con el esquema *decreciente* ya que en general, sobre la *estructura paginada,* presenta una mejor realización que el *creciente.*

[(3)] Un carácter ocupa 1 Byte, un entero 2 Bytes y un puntero 4 Bytes.

Inicialmente se repite el experimento realizado anteriormente en la memoria interna de un **HP9000-835**, con el fin de contrastar ésta realización con la obtenida utilizando memoria secundaria en el **PC**, es decir, se mide el tiempo requerido –en función de la longitud de la cadena de búsqueda– para cada tipo de búsqueda con respuesta múltiple, es decir, *truncamientos*, *máscaras* y *más similares*.

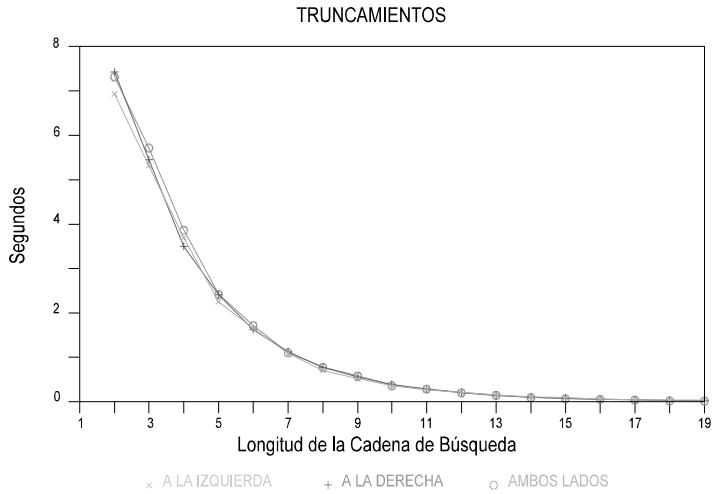

Figura 48

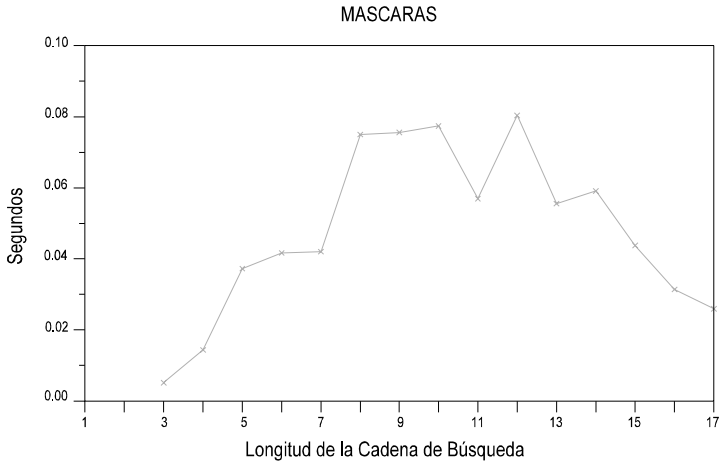

Figura 49

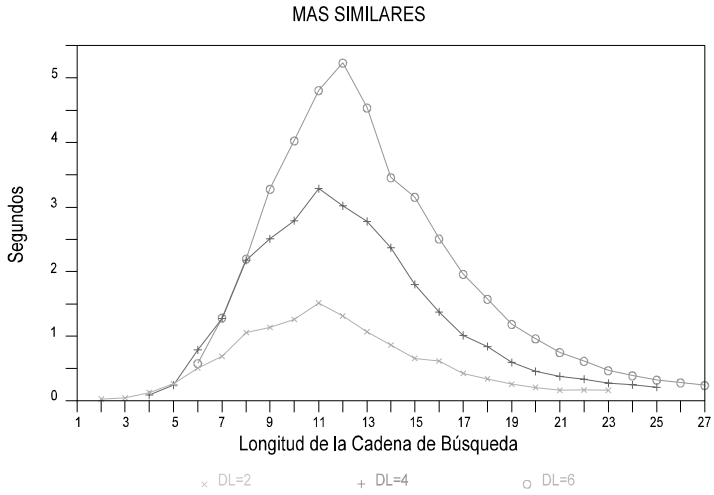

Figura 50

☞ Los resultados obtenidos en el **PC**, figuras 48, 49 y 50, son parecidos, en cuanto a la forma que presentan los tiempos de respuesta, a los correspondientes obtenidos en el **HP9000**, figuras 42, 44 y 46. Como era de esperar, los valores son cuantitativamente mayores debido fundamentalmente a los accesos a disco.

La segunda parte de este estudio experimental tiene como objetivo estudiar el tiempo necesario para resolver las peticiones permitidas desde que se formulan hasta antes de comenzar a visualizar los *artículos*.

a) Búsqueda *exacta*.

Se selecciona de forma aleatoria una muestra de 100 palabras de entre todas las distintas –sin tener en cuenta las *vacías*– que existen en el Diccionario de Medicina. Se mide el tiempo transcurrido desde que se solicita cada petición de la muestra hasta que se obtiene la confirmación de que hay respuesta. Se calcula el promedio y la desviación típica del tiempo que se

invierte en recuperar una palabra que está en el Diccionario de Medicina.

b) Búsqueda de las palabras *más similares* en función del valor de **DL**.

Este cálculo se realiza sobre dos muestras, una para *DL=2* y otra para *DL=4*, de 100 cadenas cada una. Para obtener una cadena, se selecciona al azar una palabra de entre todas las distintas del Diccionario de Medicina y se transforma aplicándole diversas operaciones de edición, elegidas aleatoriamente, hasta obtener una palabra con una de las **DL** predeterminada respecto a la original –en todo caso se limita **DL** a la mitad de la longitud de la palabra correcta. Se miden, independientemente, el tiempo de recuperación de las palabras *más similares* y el lapso necesario para realizar la unión de las listas de posiciones para cada petición sobre cada una de las dos muestras. Se calculan los promedios y las desviaciones típicas de cada uno de los tiempos medidos para los distintos valores de **DL** bajo las condiciones descritas anteriormente.

c) Búsqueda con *truncamientos*.

Se construyen tres muestras, una para cada tipo de *truncamiento*, de 100 peticiones cada una –que respeten al menos el 50% de los caracteres. Se elige al azar una palabra de entre todas las distintas del Diccionario de Medicina, se sortea entre uno y la mitad de su longitud para obtener el número de caracteres a eliminar, las posiciones de extracción de los caracteres (al principio, al final o a ambos lados) indican la

muestra que le corresponde. Para el *truncamiento a ambos lados* se selecciona aleatoriamente el número de caracteres a eliminar por el principio y por el final, no permitiéndose que estén todos a un mismo lado. Se mide, por una parte el tiempo de recuperación de las palabras que verifican la petición y por otra, el lapso necesario para realizar la unión de las listas de posiciones para cada petición sobre cada una de las tres muestras. Obteniéndose los promedios y las desviaciones típicas de cada uno de los tiempos medidos para los distintos tipos de *truncamiento*.

d) Búsqueda con *máscaras*.

Se seleccionan al azar 100 palabras de entre todas las distintas del Diccionario de Medicina, se construye una muestra de 100 peticiones –que especifiquen al menos el 50% de los caracteres. Para cada palabra se sortea el número de asteriscos entre uno y la mitad de su longitud y se seleccionan al azar las posiciones de éstos. Se miden el tiempo de recuperación de las palabras que verifican la petición y el lapso necesario para realizar la unión de las listas de posiciones para cada petición de la muestra. Se calculan el promedio y la desviación típica de los dos tiempos medidos.

Los resultados experimentales de los apartados **a**, **b**, **c** y **d** se muestran en la tabla 6. La primera y segunda columna corresponden al promedio y la desviación típica, respectivamente, del tiempo –medido en segundos– desde el momento en que el usuario realiza la petición hasta que se muestran en pantalla las palabras que la satisfacen. La tercera y cuarta

contienen respectivamente el promedio y la desviación típica del tiempo en segundos que se consume en la *unión* de las listas de posiciones asociadas a las palabras que forman parte de cada respuesta múltiple, en el caso de que el usuario las seleccionara todas. La quinta columna indica el promedio del número de palabras que verifican la petición, **NPR**.

		Tiempo promedio en el índice	Desviación típica del tiempo del índice	Tiempo promedio de unión de las listas	Desviación típica del tiempo de unión	NPR
Exacta		0.0322	0.04218	0.0000	0.00000	1
Máscaras		0.0663	0.06811	0.1164	0.22555	1.63
Truncamiento	Izquierd	0.8333	1.51744	0.5875	1.72712	10.68
	Derecho	0.9130	1.68815	0.7162	2.09274	13.3
	Ambos lados	0.9536	1.77283	0.9797	2.34481	73.55
Más similares	DL=2	1.1092	1.05868	0.2738	0.71312	3.14
	DL=4	2.0617	1.74955	0.3182	0.95049	3.14

Tabla 6

- La búsqueda que consume más tiempo en el índice, tabla 6, es la de *más similares*. El tiempo promedio de la *unión* depende del número de palabras que verifican la petición –además de la longitud de las listas de posiciones para cada una de ellas–; las búsquedas para las que se obtiene una mayor multiplicidad son las que tienen *truncamientos*, en particular el *truncamiento a ambos lados.*
- Para todos los tipos reflejados en la tabla 6, se obtienen mayores coeficientes de variación para los tiempos de unión de listas que para los de recorrido en el índice, debido a la variabilidad en la longitudes de las listas de posiciones.

☞ Se observa, tabla 6, que los valores más altos de las desviaciones típicas tanto en los tiempos de búsqueda como en los de unión de listas se presentan en las peticiones con *truncamientos*; la razón hay que buscarla en la gran variabilidad existente en el número de palabras del Diccionario de Medicina que verifican este tipo de peticiones.

e) Búsquedas con *operadores booleanos*.

A partir de las siete muestras construidas en los apartados **a**, **b**, **c** y **d**, se forman parejas entre los distintos tipos de búsqueda estudiados y se seleccionan 100 pares de peticiones para cada una de las combinaciones posibles, utilizándose éstas para estudiar cada uno de los *operadores booleanos*. Se mide el tiempo transcurrido en segundos desde que se formula la petición *booleana* hasta antes de comenzar a visualizar los *artículos*, se calcula el promedio y la desviación típica de dicho tiempo para cada combinación con cada uno de los *operadores booleanos*.

"Y"		Tiempo promedio de búsqueda						
		Exacta	Máscaras	Truncamiento			Más similares	
				Izquierdo	Derecho	Ambos lados	DL=2	DL=4
Exacta		0.0947	0.2276	1.4626	1.6641	2.0131	1.4710	2.4049
Máscaras		0.2252	0.3439	1.6809	1.8379	2.1437	1.5708	2.5348
Truncamiento	Izquierdo	1.4570	1.5885	2.7490	3.0315	3.3148	2.9930	3.9691
	Derecho	1.7247	1.8466	3.0601	3.1955	3.6493	3.2154	4.1837
	Ambos lados	1.9568	2.1165	3.3131	3.7107	3.8246	3.4667	4.4457
Más similares	DL=2	1.4034	1.5845	3.0368	3.1819	3.4637	2.7220	3.6633
	DL=4	2.3332	2.4893	4.0146	4.1964	4.4630	3.6838	4.595

Tabla 7

"Y"		Desviación típica del tiempo de búsqueda						
		Exacta	Máscaras	Truncamiento			Más similares	
				Izquierdo	Derecho	Ambos lados	DL=2	DL=4
Exacta		0.0674	0.2377	2.8616	3.4889	4.0614	1.1570	1.8297
Máscaras		0.2386	0.3251	3.0152	3.6551	4.1231	1.1919	1.8083
Truncamiento	Izquierdo	2.8517	2.9582	5.1383	6.1440	6.6046	3.2692	3.4573
	Derecho	3.4922	3.6356	6.1364	6.8282	7.8035	3.7233	3.7015
	Ambos lados	3.8854	4.0301	6.5415	8.0055	7.5716	4.1660	4.1666
Más similares	DL=2	1.1044	1.1714	3.3072	3.8612	4.2116	1.9289	2.6661
	DL=4	1.7911	1.7999	3.5552	3.7501	4.2623	2.6356	3.110

Tabla 8

"O"		Tiempo promedio de búsqueda						
		Exacta	Máscaras	Truncamiento			Más similares	
				Izquierdo	Derecho	Ambos lados	DL=2	DL=4
Exacta		0.1175	0.3173	1.6144	1.8477	2.1393	1.5500	2.4556
Máscaras		0.2780	0.4244	1.8632	2.0950	2.3525	1.6627	2.6094
Truncamiento	Izquierdo	1.5872	1.7594	2.9348	3.2839	3.5701	3.2397	4.1665
	Derecho	1.8874	2.0173	3.3043	3.5498	3.7843	3.4311	4.4023
	Ambos lados	2.1257	2.2800	3.5156	3.9055	4.1824	3.6900	4.6602
Más similares	DL=2	1.4902	1.6793	3.1524	3.4656	3.7299	2.8801	3.7928
	DL=4	2.4237	2.6105	4.2084	4.4579	4.6694	3.8305	4.738

Tabla 9

"O"		Desviación típica del tiempo de búsqueda						
		Exacta	Máscaras	Truncamiento			Más similares	
				Izquierdo	Derecho	Ambos lados	DL=2	DL=4
Exacta		0.0779	0.3244	3.1294	3.8754	4.2714	1.2300	1.8743
Máscaras		0.3004	0.4249	3.3140	4.2025	4.5065	1.2457	1.8410
Truncamiento	Izquierdo	3.0871	3.2153	5.5453	6.6077	7.0923	3.5989	3.8256
	Derecho	3.8548	3.9943	6.6105	7.5437	7.8930	4.0832	4.0675
	Ambos lados	4.2015	4.3412	6.9743	8.0797	8.3003	4.5431	4.5758
Más similares	DL=2	1.1956	1.2259	3.5949	4.2944	4.6550	1.9731	2.7134
	DL=4	1.8642	1.8453	3.8910	4.1977	4.6231	2.7585	3.157

Tabla 10

"Y_NO"		Tiempo promedio de búsqueda						
		Exacta	Máscaras	Truncamiento			Más similares	
				Izquierdo	Derecho	Ambos lados	DL=2	DL=4
Exacta		0.0864	0.2242	1.4642	1.7736	2.0569	1.4599	2.3715
Máscaras		0.2360	0.3504	1.6768	1.8740	2.1496	1.5689	2.5057
Truncamiento	Izquierdo	1.4730	1.6227	2.8232	3.0808	3.4003	2.9727	4.0398
	Derecho	1.7366	1.8554	3.1626	3.3016	3.6597	3.3152	4.2397
	Ambos lados	2.0117	2.1451	3.4047	3.8373	3.8705	3.5320	4.5038
Más similares	DL=2	1.4221	1.5967	3.0450	3.1612	3.4191	2.7574	3.7093
	DL=4	2.3460	2.5256	4.0947	4.2648	4.5022	3.7382	4.610

Tabla 11

"Y_NO"		Desviación típica del tiempo de búsqueda						
		Exacta	Máscaras	Truncamiento			Más similares	
				Izquierdo	Derecho	Ambos lados	DL=2	DL=4
Exacta		0.0747	0.2343	2.8987	3.6213	4.0876	1.1553	1.7861
Máscaras		0.2702	0.3434	3.0432	3.7177	4.1383	1.1600	1.7835
Truncamiento	Izquierdo	2.9166	3.0493	5.3774	6.3315	6.9510	3.3886	3.6391
	Derecho	3.6150	3.7311	6.5784	7.3318	7.7497	3.9116	3.7971
	Ambos lados	4.0466	4.1056	6.8224	8.0647	7.7743	4.2984	4.2626
Más similares	DL=2	1.1418	1.1745	3.3967	3.7600	4.1721	1.9165	2.7303
	DL=4	1.8005	1.8107	3.6863	3.87099	4.2800	2.7172	3.121

Tabla 12

☞ De las tres búsquedas *booleanas* la más costosa, en general, es la *disyuntiva*, tabla 9; la *conjuntiva* y la que incluye el *operador y_no* presentan realizaciones parecidas, tablas 7 y 11.

Las combinaciones que presentan un mayor tiempo de respuesta son aquellas en las que intervienen búsquedas de *más similares*.

- ☞ Las tablas 7-12 muestran una tendencia a la simetría teórica respecto a la diagonal principal que en la práctica no se alcanza debido a las fluctuaciones en la medición de los tiempos, aunque cuando se traten listas de igual tamaño y con el *operador y_no* puede existir alguna influencia debida al orden en el que se traten las listas de posiciones.
- ☞ Se puede comprobar el incremento del tiempo promedio de una *combinación* respecto a la suma del consumido por las búsquedas que intervienen en ella utilizando la tabla 6, junto con las tablas 7, 9 y 11.
- ☞ Se mantiene que los mayores coeficientes de variación –tablas 8, 10 y 12 en relación con las 7, 9 y 11– se obtienen, en general, para la resolución de combinaciones en las que intervienen *truncamientos* que son además las que presentan mayor variabilidad.

f) Búsqueda en *sentencias*.

Como muestra se seleccionan al azar parejas de palabras de entre las diferentes del Diccionario de Medicina, sin considerar las que no pertenezcan a un mismo artículo, hasta completar un total de 100.

g) Búsquedas de *cercanía* y *antecedencia*.

Se utilizan las mismas parejas del apartado anterior. Se fija el valor de **n**, siendo 5 para *antecedencia* y 20 para *cercanía*.

h) Búsqueda de *frases*.

Se selecciona aleatoriamente una muestra de 100 *frases* del Diccionario de Medicina formadas por entre cinco y doce palabras, y con no menos de cuatro palabras no *vacías* ni más de seis.

Para los apartados **f**, **g** y **h** se mide el tiempo desde que se formula la petición hasta antes de comenzar a visualizar los artículos –incluyendo la verificación, en cada artículo, de las restricciones especificadas en la petición en cuanto a la situación de las palabras.

Se calcula el promedio y la desviación típica del tiempo medido en segundos para cada uno de los apartados **f**, **g** y **h**, bajo las condiciones expuestas. Los resultados se muestran en la tabla 13.

	Tiempo promedio	Desviación típica del tiempo
Sentencias	0.1949	0.11274
Cercanía	0.2086	0.06851
Antecedencia	0.1895	0.06639
Frases	0.4248	0.16315

Tabla 13

☞ Los cuatro tipos de búsqueda de la tabla 13 requieren el acceso al *artículo* para comprobar si se cumplen las restricciones de la petición, a pesar de ello los resultados obtenidos son bastante satisfactorios. Se observa un tiempo algo mayor para la búsqueda de *frases* debido al mayor número de palabras implicadas. La variabilidad en general es pequeña para este tipo de búsquedas.

Capítulo 6.
Conclusiones y Principales Aportaciones

En este trabajo se propone una solución al problema de la búsqueda de las cadenas *más similares* a una dada –según la distancia de Levenshtein, **DL**– en un determinado diccionario de cadenas.

Como aportación esencial se ha introducido una nueva distancia denominada *distancia invariante trasposicional*, **DIT**, que se define en función de la frecuencia de aparición de cada carácter en ambas cadenas. Se ha analizado su costo computacional obteniéndose que es sensiblemente inferior al de **DL** y además se ha demostrado que $DIT(X,Y) \leq 2*DL(X,Y)$. Estas características permiten su aplicación a la construcción de un filtro adaptivo **DIT**│**DL** que tiene por misión reducir el número de cadenas del diccionario que han de soportar el cálculo de **DL** con la cadena de búsqueda.

El diccionario se organiza de manera genuina como un árbol en el cual se estructuran las componentes que intervienen en el cálculo de **DIT** –estructura **S-D**– con el fin de optimizar el cálculo **DIT** y a la vez disponer de una forma de uso más eficaz del filtro que seleccione las cadenas con las que se evalúa **DL**. Se propone como criterio de selección del carácter discriminante el de los *mínimos cuadrados*. Se comprueba la evolución de la relación ocupacional entre el índice y los datos con el incremento de la

cardinalidad del diccionario, obteniéndose una tendencia a la igualdad en las respectivas ocupaciones ello indica la bondad de la estructura.

El esquema de búsqueda de las cadenas *más similares*, desarrollado en este trabajo, que comienza con un valor del radio igual a infinito y posteriormente va disminuyendo, con el tratamiento de cadenas cada vez más próximas, ha sido denominado esquema *decreciente*.

Se ha estudiado un nuevo esquema de búsqueda denominado *creciente*, donde el radio de búsqueda, en oposición a la evolución clásica decreciente, sigue una línea de modificación creciente.

Se ha presentado una *familia de esquemas* definida en función del incremento del radio de búsqueda que, de una manera natural, evoluciona conceptualmente desde el *decreciente* al *creciente*.

En el esquema *creciente*, a diferencia del *decreciente*, existen nodos que se visitan más de una vez; en principio esta característica no es deseable, pero queda suficientemente compensada por la aproximación más acertada a las cadenas más próximas, como se ha visto –da lugar a una mejor realización en memoria interna.

La introducción de la poda **PP** ha supuesto, para ambos esquemas, una optimización en el recorrido del índice.

Se ha introducido una nueva distancia, **DS**, definida en función de la longitud de la subsecuencia común más larga; tiene en cuenta características posicionales de las cadenas que el índice no considera y posee un bajo costo computacional; se ha demostrado además que $DIT(X,Y) \leq 2*DS(X,Y) \leq 2*DL(X,Y)$. Todo ello permite utilizar **DS** como filtro de **DL** con lo que se obtiene una mejor realización.

Se han estudiado diversas formas de paginación de la estructura **S-D** para resolver la situación en la que debido a su tamaño no sea posible ubicarla en memoria interna. Con la paginación en *preorden* se han obtenido los mejores resultados para los dos esquemas de búsqueda. Se selecciona el esquema *decreciente* para la búsqueda de *más similares* en memoria secundaria ya que supera al *creciente* para $DL>2$ y la mejoría del *creciente* para $DL \leq 2$ se reduce a medida que aumenta el tamaño de la página.

Se ha aplicado la estructura **S-D** a la recuperación de información en texto libre con resultados altamente satisfactorios. La estructura se ha construido con el conjunto de palabras diferentes consideradas no *vacías*, extraídas de un texto de más de un millón de palabras; constituye un índice que permite además de la recuperación de las palabras *más similares* otros tipos de búsqueda habituales en texto libre.

El estudio previo de la paginación de la estructura **S-D** ha permitido implantar esta aplicación en un ordenador personal, suponiendo un incremento del tiempo de respuesta con respecto a la realización en memoria interna que se puede considerar muy aceptable.

La continuidad de este trabajo está en la extensión a la recuperación de información en bases de datos documentales de gran dinamicidad; el objetivo es desarrollar un sistema que basándose en una reestructuración parcial de la estructura permita la recuperación de información en un archivo de documentos que se está actualizando constantemente, implementándolo en un ambiente multitarea de forma que puedan llevarse a cabo tales reestructuraciones mientras el usuario realiza consultas. Otro aspecto a tratar en futuros trabajos consiste en llevar a cabo una

generalidad de los diversos tipos de búsqueda, teniendo en cuenta la organización interna de los diferentes documentos y la adaptación de la estructura a esta nueva situación.

Apéndices

Apéndice 1. Lista de Palabras Vacías

a
acción
ácido
activa
actividad
actúa
actualidad
actualmente
además
afección
afecta
agente
al
algo
alguien
algún
alguna
algunas
algunos
alrededor
ambas
ambos
ángulo
anterior
antes
aparece
aparecen
aparecer
aparición
aplica
aplicación
aplicase
aproximadamente
aquel
aquella
aquellas
aquellos
área
arriba
así
asocia
asociación
asociada
asociado
atrás
aumenta
aumento
aunque
ausencia
años
bajo
base
bastante
bien
borde
cada
cambio
cambios
capacidad
capaz
característica
características
característico
caracteriza
caracterizada
caracterizado
casi
caso
casos
causada
causas
cavidad
célula
células
central
centro
cierta
ciertas
cierto
ciertos
combinación
como
completamente
componente

comprende
compuesto
compuestos
común
con
condiciones
conducto
congénita
conjunto
conoce
consecuencia
considera
consiste
consistente
constituida
constituido
constituye
constituyen
contenido
contiene
contienen
continua
contra
corresponde
cree
crónica
cual
cuales
cualquier
cualquiera
cuando
cuanta
cuantas
cuanto
cuantos
cuatro
cuerpo
cursa
curso
cuya
cuyas
cuyo
cuyos
da
dan
dar
de
debajo
debe
deben
debida
debido
defecto
degeneración
del
delante
demás
denomina
denominada
denominado
dentro
depende
derivado
derivados
desarrolla
desarrollo
descrito
desde
después
determinación
determinada
determinadas
determinado
determinados
determinar
detrás
dícese
diferentes
dirección
disminución
distinguen
distribución
diversas
diversos
dolor
donde
dos
durante
e
efecto
efectos
ej
ejemplo
el
elementos
ella
ellas
ello
ellos
embargo
emplea
empleado
en
encuentra
encuentran
enfermedad
enfermedades
enfermo

enfermos
entre
es
eso
espacio
especial
especialmente
especie
específico
esta
están
estar
estas
este
esto
estos
estructura
estructuras
estudia
estudio
etc
existe
existen
existencia
extensión
externa
externo
extiende
extremo
factor
factores
falta
familiar
fase
fenómeno
fibras
fin
final
finalmente
forma
formación
formada
formado
forman
forman
formando
formas
frecuencia
frecuente
frecuentemente
frecuentes
fue
fuera
función
funciones
fundamental
fundamentalmente
general
generalmente
género
gr
gran
grande
grandes
grupo
ha
haber
habitualmente
hace
hacen
hacer
hacia
halla
hallan
han
hasta
hay
hecho
hueso
i
igual
importancia
importante
importantes
incluso
incluyen
indica
individuos
inferior
inflamación
insuficiencia
intensa
interior
interna
interno
izquierda
junto
la
lado
largo
las
lat.
lateral
laterales
le
les
lesión
lesiones

línea
llama
llamada
llamado
lo
local
localiza
localización
localizada
localizado
los
lugar
mal
manera
manifestaciones
manifiesta
maniobra
mas
material
mayor
mayoría
media
mediante
medida
medio
medios
menor
menos
menudo
método
mía
mías
mientras
mío
míos
misma
mismas
mismo
mismos
mitad
modo
mucha
muchas
mucho
muchos
múltiple
múltiples
músculo
músculos
muy
nada
nadie
ni
nivel
no
nombre
normal
normales
normalmente
nosotros
nuestra
nuestro
nuestros
número
o
observa
observan
observar
observarse
obtenido
ocasionalmente
ocasiones
ocurre
ocurrir
origen
origina
otra
otras
otro
otros
pacientes
para
parece
parte
partes
particularmente
partir
pasan
paso
pequeña
pequeñas
perdida
período
permite
pero
perteneciente
piel
plano
poca
pocas
poco
pocos
por
porción
porque
posee
poseen
posible

posición
posterior
posteriormente
presencia
presenta
presentación
presentan
presentar
presentarse
presente
primaria
primario
primer
primera
primeros
principal
principales
principalmente
probablemente
proceso
procesos
producción
produce
producen
producida
producido
produciendo
producir
producto
productos
progresiva
propia
propiedad
propiedades
propio
provoca
prueba
puede
pueden
pues
punto
que
queda
rápidamente
raramente
reacción
realiza
realizar
recibe
referencia
referirse
refiere
relación
relativamente
relativo
representa
respectivamente
respuesta
resultado
s
se
sea
secundaria
secundario
secundarios
seguida
según
segundo
semejante
sensación
ser
si
sido
siempre
siendo
significa
signo
sigue
simple
sin
síndrome
sino
síntoma
sirve
sistema
situación
situada
situadas
situado
sobre
solo
solución
son
su
suele
suelen
superficial
superficie
superficies
superior
superiores
sus
sustancia
sustancias
suya
suyas
suyo
suyos

tal
tales
tamaño
también
tan
tanta
tantas
tanto
tantos
tejido
tejidos
tendencia
tener
tercer
término
tiempo
tiene
tienen
tipo
tipos
toda
todas
todo
todos
total
totalmente
tras
trata
tratado
tratamiento
través
tres
tu
tuya
tuyas
tuyo
tuyos
u
un
una
unas
únicamente
unión
uno
unos
usualmente
utiliza
utilizada
utilizado
v
va
van
varias

variedad
varios
ve
veces
vez
vía
vosotros
vuestra
vuestro
vuestros
y
ya
yo
zona
zonas

Apéndice 2. Abreviaturas y Siglas

abs	Función valor absoluto
B	Cadena de búsqueda
C_DIT	Componentes de DIT
CA_DIT	Valor acumulado de las componentes de DIT en el recorrido del índice
D	Diccionario: conjunto de cadenas sobre un alfabeto dado
DIT	Distancia invariante trasposicional
DL	Distancia de Levenshtein
DLM	Mínima DL alcanzada
DS	Distancia de Santana
DTM	Radio de búsqueda DIT
IR	Incremento del radio de búsqueda en la familia de esquemas
Lsc	Función que calcula la longitud de la subsecuencia común más larga
LV	Matriz del cálculo optimizado de Landau y Vishkin de DL
MFU	Política para liberar del buffer la página menos frecuentemente usada
MRU	Política para liberar del buffer la página menos recientemente usada
NCD	Cardinalidad del diccionario
NNA	Número de nodos afectados
NNR	Número de veces que se revisitan los nodos
NPR	Multiplicidad de la respuesta: número de palabras que verifican la petición

PA	Poda de la estructura S-D que depende exclusivamente de los ancestros
PC	Ordenador personal
PEPS	Política para liberar del buffer la primera página que entró
PP	Poda predictiva sobre los sucesores en la estructura S-D
S-D	Estructura de Santana y Díaz que organiza las componentes de DIT
SB	Conjunto de cadenas más similares a B
SIT	Sinónimos invariantes trasposicionales
UEPS	Política para liberar del buffer la última página que entró
WF	Matriz del cálculo de Wagner y Fischer de DL

Referencias

[AL67] ALBERGA, C. N.: "String Similarity and Misspellings". Comm. ACM., Vol. 10 (5), 302/313, (1967).

[BE75] BENTLEY, J. L.: "Multidimensional Binary Search Trees used for Associative Searching". Comm. of ACM., Vol. 18, 9, 509/516, (1975).

[BE79] BENTLEY, J. L.: "Multidimensional Binary Search Trees in Database Applications". IEEE Transactions on Software Engineering, Vol. SE-5, 4, 333/340, (1979).

[BE85] BECKLEY, D. A.; EVENS, M. W.; RAMAN, V. K.: "Multikey Retrieval from K-D Trees and Quad-Trees". AT & T Bell Laboratories Naperville, Ill. Go 566, 1/29, (1985).

[BK73] BURKHARD, W. A.; KELLER, R. M.: "Some Approches to Best-Mach File Searching". Comm. ACM., 16, 4, (1973).

[BM77] BOYER, R. S.; MOORE, J. S.: "A Fast String Searching Algorithm". Comm. ACM., Vol. 20, 762/772, (1977).

[BY89] BAEZA-YATES, R.: "Efficient Text Searching". PhD thesis Dept. of Computer Science, Univ. of Waterloo, 1989. Also as Research Report Cs-89-17, (1989).

[CO91] COLE, R.: "Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm". 2nd Symp. on Discrete Algorithms, SIAM, S. Francisco, Cal., 224/233, (1991).

[DS90] DIAZ, M.; SANTANA, O.; RODRIGUEZ, J. C.: "Búsqueda de las Cadenas Más Similares: Esquema Decreciente con Radio de Búsqueda Ascendente, Esquema Creciente". XVI Conferencia Latinoamericana de Informática, Vol I, 90/97, (1990).

[DS93] DIAZ, M; SANTANA, O.; PEREZ, J.: "Distancia dependiente de la Subsecuencia Común Más Larga entre Cadenas de Caracteres". Aceptado para publicar en las Actas de las II Jornadas de Ingeniería de Sistemas Informáticos y Computación, Quito (Ecuador), (1993).

[FA85] FALOUTSOS, C.: "Access Methods for Text". ACM Comm. Surveys, 17, 49/74, (1985).

[FA88] FALOUTSOS, C.: "Signature Files: An Integrated Access Method for Text and Attributes Suitable for Optical Disk Storage". BIT, 28 (4), 736/754, (1988).

[FB74] FINKEL, R. A.; BENTLEY, J. L.: "Quad Trees a Data Structure for Retrieval on Composite Keys". Acta Informatica, 4, 1/9, Springer Verlag, (1974).

[FB77] FRIEDMAN, J. H.; BENTLEY, J. L.; FINKEL, R. A.: "An Algorithm for Finding Best Matches in Logarithmic Expected Time". ACM Transactions on Mathematical Software, Vol. 3, 3, 209/226, (1977).

[FB92] FRAKES, W. B. & BAEZA-YATES, R. editors: "Information Retrieval. Data Structures & Algorithms". Prentice Hall, Englewood Ciffs, New Jersey, (1992).

[FC84] FALOUTSOS, C.; CHRISTODOULAKIS, S.: "Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation". ACM TOOIS, 2 (4), 267/288, (1984).

[FO73] FORNEY, G. D.: "The Viterbi Algorithm". Proc. IEEE, Vol. 61, 268/278, (1973).

[GG86] GALIL, Z.; GIANCARLO, R.: "Improved String Matching with k Mismatches", SIGACT News, 17, 4, 52/54, (1986).

[GO87] GONNET, G.: "PAT 1.3: An Efficient Text Searching System, User's Manual", UW Centre for the New OED, University of Waterloo, (1987).

[HA71] HARRISON, M. C.: "Implementation of the Substring Test by Hashing". C. ACM, 14, 777/779, (1971).

[HD80] HALL, P. A. V.; DOWLING, G. R.: "Approximate String Matching". Computing Surveys., Vol. 12, 4, 381/402, (1980).

[HI75] HIRSCHBERG, D. S.: "A Linear Space Algorithm for Computing Maximal Common Subsequences", Communications of the ACM, Vol 18, 341/353, (1975).

[HO80] HORSPOOL, R. N.: "Practical Fast Searching in Strings". Software-Practice and Experience, 10, 501/506, (1980).

[HS77] HUNT, J. W.; SZYMANSKI, T. G.: "A Fast Algorithm for Computing Longest Common Subsequences", Communications of the ACM, Vol 20, 350/353, (1977).

[HS82] HULL, J. J.; SRIHARI, S. N.: "Experiments in Text Recognition with Binary N-Gram and Viterbi Algorithms". IEEE Trans. Pat. Anal. Mach. Intel., Vol. PAMI-4, 5, 520/530, (1982).

[HS91] HUME, A.; SUNDAY, D. M.: "Fast String Searching". AT&T Bell Labs Computing Science Technical Report Nº 156, (1991).

[IK82] ITO, T.; KIZAWA, M.: "Spelling Correction on a Hierarchically Organized File". Trans. Inst. Electron. Commun. Eng. Japan (in Japanese), Vol. J65-D, 1090/1091, (1982).

[IY84] ITAHASHI, S.; YOKOYAMA, S.: "Vocabulary Reduction Effectby Specifying Phoneme Sequences in Words". Trans. Inst. Electron. Commun. Eng. Japan (in Japanese), Vol. J67-D, 869/876, (1984).

[JV87] JOVEN, J.; VILLABONA, C.; JULIA, G.; GONZALEZ-HUIX, F.: "Diccionario de Medicina". Tercera edición, Editorial MARIN, S.A, (1987).

[KA84] KURITA, T.; AIZAWA, T.: "A Method for Correcting Errors on Japanese Words Input and its Application to Spoken Word Recognition with Large Vocabulary". J. Inform. Processing Soc. Jap., Vol. 25, 831/841, (1984).

[KC89] KUO, S.; CROSS, G. R.: "An Improved Algorithm to Find the Length of the Longest Common Subsequence of Two Strings", SIGIR FORUM, Vol 23, Numbers 3, 4, (1989).

[KD83] KANEKO, T.; DIXON, N. R.: "A Hierarchical Decision Approach to Large Vocabulary Discrete Utterance Recognition". IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-31, 1061/1066, (1983).

[KM77] KNUTH, D. E.; MORRIS, J.; PRATT, V.: "Fast Pattern Matching in Strings". SIAM J. on Computing, 6, 323/350, (1977).

[KO85] KASHYAP, R. L.; OOMMEN, B. J.: "Spelling Correction Using Probabilistic Methods". Pattern Recognition Letters, No. 2, 147/154, (1985).

[LA83] LARSON, P.: "A Method for Speeding Up Text Retrieval". Proceedings ACM SIGMOD, San Jose, California, 12, 117/123, (1983).

[LE66] LEVENSHTEIN, V. I.: "Binary Codes Capable of Correcting, Insertions and Reversals". Soviet Phys. Dokl. 10, 707/710, (1966).

[LV85a] LANDAU, G. M.; VISHKIN, U.: "Efficient String Matching in the Presence of Errors". Proc. 26th IEEE FOSC, 126/136, (1985).

[LV85b] LANDAU, G. M.; VISHKIN, U.: "Efficient String Matching with k Differences". TR-36/85, Departament of Computer Science, Tel Aviv Univ., Submitted for Journal Publication, (1985).

[LV86a] LANDAU, G. M.; VISHKIN, U.: "Efficient String Matching with k Mismatches". Theoretical Computer Science, 43, 239/249, (1986).

[LV86b] LANDAU, G. M.; VISHKIN, U.; NUSSINOV, R.: "An Efficient String Matching Algorithm with k Differences for Nucleotide and Amino Acid Sequences". Nucleic Acid Research 14 (1), 31/46, (1986).

[MM90] MANBER, U.; MYERS, G.: "Suffix Arrays: A New Method for On-line String Searches". 1st ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 319/327, (1990).

[MO70] MORGAN, H. L.: "Spelling Correction in System Programs". C. ACM, Vol. 13, 2, 90/94, (1970).

[NE75] NEUHOFF, D. L.: "The Viterbi Algorithm as an Aid in Text Recognition". IEEE Trans. Inform. Theory, Vol. IT-21, 222/226, (1975).

[OT76] OKUDA, T.; TANAKA, E.; KASAI, T.: "A Method for the Correction of Garbled Words Based on the Levenshtein Metric". IEEE Trans. Comput., Vol. C-25, 2, 172/177, (1976).

[RE71] RISEMAN, E. M.; EHRICH, R. W.: "Contextual Word Recognition Using Binary Digrams". IEEE Trans. Comput., Vol. C-20, 397/403, (1971).

[RH74] RISEMAN, E. M.; HANSON, A. R.: "A Contextual Post-Processing System for Error-Correcting Using Binary N-Grams". IEEE Trans. Comput., Vol. C-23, 5, 480/493, (1974).

[RI77] RIVEST, R. L.: "On the Worst-Case Behavior of String-Searching Algorithms". SIAM J. Comput., Vol. 6, 4, 669/674, (1977).

[SD87] SANTANA, O.; DIAZ, M.; MAYOR, O.; REYES, J.: "Esquemas y Estructura para la Búsqueda de las Palabras Más Similares a una Dada". XIII Conferencia Latinoamericana de Informática, Vol. II, 1169/1189, (1987).

[SD89] SANTANA, O.; DIAZ, M.; DUQUE, J. D.; RODRIGUEZ, G.: "Estructuración de las Componentes de la Distancia Invariante Trasposicional, DIT, con Compartición de la Zona No-Discriminante en la Búsqueda de las Cadenas Más Similares". XV Conferencia Latinoamericana de Informática, Vol. II, 335/341, (1989).

[SD92] SANTANA, O.; DIAZ, M.; BALLESTER, A.; SANTANA, F. J.: "Recuperación de Información en Diccionarios". VIII Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN), (1992).

[SH83] SHINGHAL, R.: "A Hybrid Algorithm for Contextual Text Recognition". Pattern Recognition, Vol. 16, 261/267, (1983).

[SP90] SANTANA. O.; PEREZ, J.; RODRIGUEZ, J. C.: "Increasing Radius Search Schemes for the Most Similar Strings on the Burkhard-Keller Tree". Cybernetics and Systems: An International Journal, (1990).

[SR90] SANTANA, O.; RODRIGUEZ, J. C.; DIAZ, M.: "Búsqueda de las Cadenas Más Similares: Incidencia de la Subsecuencia Común Más Larga en los Esquemas Decreciente y Creciente". XVI Conferencia Latinoamericana de Informática, Vol I, 98/104, (1990).

[SS85] SMALL, H.; SWEENEY, E.: "Clustering the Science Citation Index Using Cocitations. I. A Comparison of Methods". Scientometrics, 7, 391/409, (1985).

[ST79] SHINGHAL, R.; TOUSSAINT, G. T.: "A Bottom-Up and Top-Down Approach to Using Context in Text Recognition". Int. J. of Man-Mach. Studies, Vol. 11, 201/212, (1979).

[SU90] SUNDAY, D.: "A Very Fast Substring Search Algorithm". Communications of the ACM, 33 (8), 132/142, (1990).

[SZ73a] SZANSER, A. J.: "Automatic Error Correction of Natural Text: Part 1". Computer Science, No. 46. National Physics Laboratory, England, (1973).

[SZ73b] SZANSER, A. J.: "Bracketing Technique in Elastic Matching". Comput. J., Vol. 16, 2, 132/134, (1973).

[TK86] TANAKA, E.; KOHASHIGUCHI, T.; SHIMAMURA, K.: "High Speed Error Correcting Method for Substitution Errors". Trans. Inform. Processing Soc. Japan (in Japanese), Vol. 27, 177/182, (1986).

[TK87] TANAKA, E.; KOJIMA, Y.: "A high speed string correction method using a hierarchical file". IEEE Trans. Pattern Anal. Mach. Intell. Vol. PAMI-9, 6, 806/815, (1987).

[TT86] TANAKA, E.; TOYAMA, T.; KAWAI, S.: "High Speed Error Correction of Phoneme Sequences". Pattern Recognition, Vol. 19, 407/412, (1986).

[TU90] TARHIO, J.; UKKONEN, E.: "Boyer-Moore Approach to Approximate String Matching". Proceedings Scandinavian Workshop in Algorithmic Theory, SWAT'90, Lecture Notes in Computer Science, 447, Springer-Verlag, Bergen, Norway, 2, 348/359, (1990).

[UK83] UKKONEN, E.: "On Approximate String Matching". Proc. Int. Conf. Found. Comp. Theor., Lecture Notes in Computer Science 158, Springer-Verlag, 487/495, (1983).

[UK85] UKKONEN, E.: "Finding Approximate Pattern in Strings". J. of Algorithms, 6, 132/137, (1985).

[UL75] ULLMANN, J. R.: "A Binary N-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words". The Computer Journal, Vol. 20, 2, 141/147, (1975).

[UW90] UKKONEN, E.; WOOD, D.: "A Simple on-line Algorithm to Approximate String Matching". (Report A-1990-4), Helsinki, Finland, (1990).

[WF74] WAGNER, R. A.; FISCHER, M. J.: "The String-to-String Correction Problem". JACM, 21 (1), 168/173, (1974).

[WI88] WILLETT, P.: "Recent Trends in Hierarchic Document Clustering: A Critical Review". Information Processing & Management, 24 (5), 577/597, (1988).