# Indexando Bases de Datos no Estructurados

**Norma Herrera, Darío Ruano, Paola Azar, Daniel Welch**
Departamento de Informática
Universidad Nacional de San Luis, Argentina
{nherrera, dmruano, epazar, dwelch}@unsl.edu.ar

**Anabella De Battista, Andrés Pascal**
Departamento Ingeniería en Sistemas de Información
FRCU, Universidad Tecnológica Nacional, Entre Ríos, Argentina
{anadebattista, andrespascal22}@gmail.com

## Abstract

Las bases de datos han incluido la capacidad de almacenar datos no estructurados tales como imágenes, sonido, texto, video, etc. La problemática de almacenamiento y búsqueda en estos tipos de base de datos difiere de las bases de datos clásicas, dado que no es posible organizarlos en registros y campos, y aun cuando pudiera hacerse, la búsqueda exacta carece de interés. Es en este contexto donde surgen nuevos modelos de bases de datos capaces de cubrir las necesidades de almacenamiento y búsqueda de estas aplicaciones. Nuestro interés se basa en el diseño de índices eficientes para estas nuevas bases de datos.

## 1 Contexto

El presente trabajo se desarrolla en el ámbito de la línea Técnicas de Indexación para Datos no Estructurados del Proyecto Tecnologías Avanzadas de Bases de Datos (22/F814), cuyo objetivo es realizar investigación básica sobre manejo y recuperación eficiente de información no tradicional.

## 2 Introducción

La información disponible en formato digital aumenta día a día su tamaño de manera exponencial. Gran parte de esta información involucra el uso de datos no estructurados tales como imágenes, sonido, texto, video, etc. Debido a que no es posible organizar estos tipos de datos en registros y campos, las tecnologías tradicionales de bases de datos para almacenamiento y búsqueda de información no son adecuadas en este ámbito.

Es en este contexto donde surgen nuevos modelos de bases de datos capaces de cubrir las necesidades de almacenamiento y búsqueda de estas aplicaciones. Nuestro interés se basa en el diseño de índices eficientes para estas nuevas bases de datos. Describimos a continuación los modelos de bases de datos e índices sobre los que nos estamos trabajando.

**Bases de Datos de Texto.** Una base de datos de texto es un sistema que mantiene una colección grande de texto y que provee acceso rápido y seguro al mismo. Sin pérdida de generalidad, asumiremos que la base de datos de

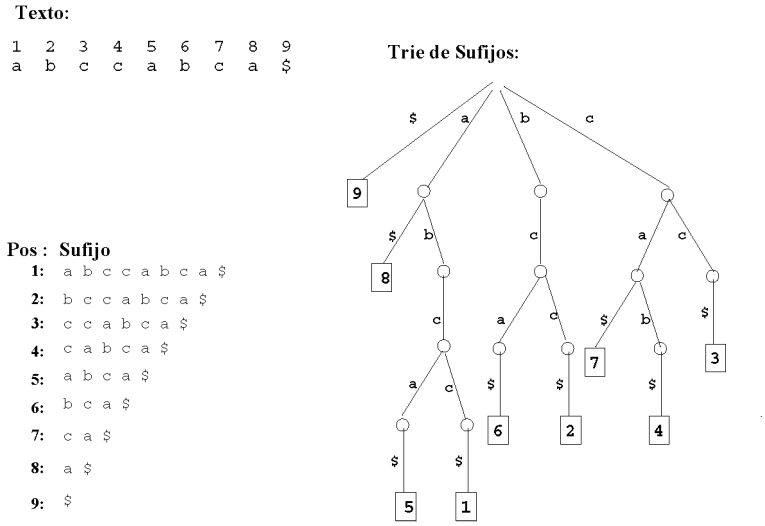


Figure 1: Un ejemplo de un texto, sus sufijos y su correspondiente trie de sufijos.

texto es un único texto $T$ que posiblemente se encuentra almacenado en varios archivos.

Una de las búsquedas más comunes en bases de datos de texto es la *búsqueda de un patrón*: el usuario ingresa un string $P$ (*patrón de búsqueda*) y el sistema retorna las ocurrencias del patrón $P$ en el texto $T$. Para resolver eficientemente estas búsquedas se hace necesario construir un índice que permita acelerar el proceso de búsqueda.

Mientras que en bases de datos tradicionales los índices ocupan menos espacio que el conjunto de datos indexado, en las bases de datos de texto el índice ocupa más espacio que el texto, pudiendo necesitar de 4 a 20 veces el tamaño del mismo [4, 8]. Una alternativa para reducir el espacio ocupado por el índice es buscar una representación compacta del mismo, manteniendo las facilidades de navegación sobre la estructura. Pero en grandes colecciones de texto, el índice aún comprimido suele ser demasiado grande como para residir en memoria principal [5, 6]. Por esta razón, el estudio de índices comprimidos y en memoria secundaria para texto es un tema de interés

Dado un texto $T = t_1, \ldots, t_n$ sobre un alfabeto $\Sigma$ de tamaño $\sigma$, donde $t_n = \$ \notin \Sigma$ es un símbolo menor en orden lexicográfico que cualquier otro símbolo de $\Sigma$, definimos ***sufijo de T*** a cualquier string de la forma $T_{i,n} = t_i, \ldots, t_n$ con $i = 1..n$. Cada sufijo $T_{i,n}$ se identifica unívocamente por $i$; llamaremos al valor $i$ ***índice del sufijo*** $T_{i,n}$.

Un ***trie de sufijos***[4] es un *trie* construido sobre el conjunto de todos los sufijos del texto, en el cual cada hoja mantiene el índice del sufijo que esa hoja representa. La figura 1 muestra un ejemplo de un texto, los sufijos del texto con la posición del texto donde cada sufijo comienza y su correspondiente trie de sufijos.

La representación habitual de un trie consiste en mantener en cada nodo los punteros a sus hijos, junto con el rótulo correspondiente a cada uno de ellos [7].

En [9] se presenta una nueva representación de un trie de sufijos que permite reducir el espacio necesario para almacenar el índice, eliminando la necesidad de mantener los punteros explícitos a los hijos. Esta representación tiene la ventaja de permitir un posterior proceso de paginado para manejar eficientemente el trie de sufijos en memoria secundaria.

**Bases de Datos Métrico-Temporales.** Este modelo permite almacenar objetos no estructurados con tiempos de vigencia asociados y realizar consultas por similitud y por tiempo

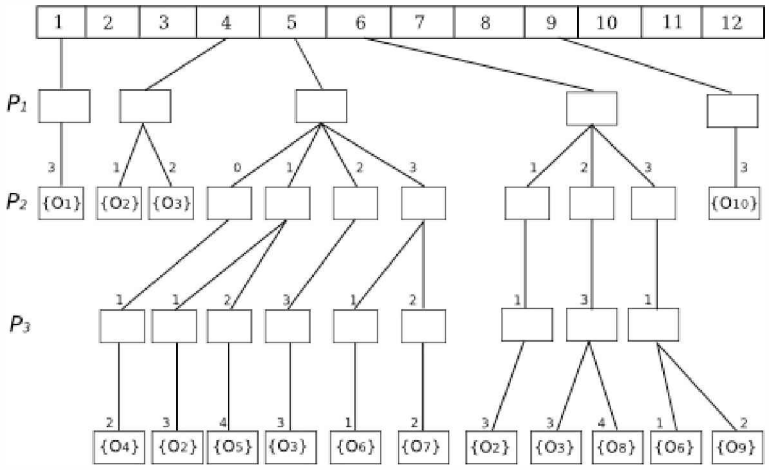


Figure 2: Un ejemplo de un H-FHQT

en forma simultánea. Formalmente un *Espacio Métrico-Temporal* es un par $(U, d)$, donde $U = O \times N \times N$, y la función $d$ es de la forma $d : O \times O \rightarrow R^+$. Cada elemento $u \in U$ es una triupla $(obj, t_i, t_f)$, donde $obj$ es un objeto (por ejemplo, una imagen, sonido, cadena, etc) y $[t_i, t_f]$ es el intervalo de vigencia de $obj$. La función de distancia $d$, que mide la similitud entre dos objetos, cumple con las propiedades de una métrica (positividad, simetría y desigualdad triangular).

Un nuevo tipo de consulta son las denominadas métrico-temporales que se definen formalmente en símbolos como: $(q, r, t_{iq}, t_{fq})_d = \{o/(o, t_{io}, t_{fo}) \in X \wedge d(q, o) \leq r \wedge (t_{io} \leq t_{fq}) \wedge (t_{iq} \leq t_{fo})\}$ Esta consulta implica buscar todos los objetos $o$ de la base de datos que estén a una distancia a lo más $r$ de $q$, y cuyo tiempo asociado $t$ se solapa con el tiempo de la consulta.

Varios índices métrico-temporales se han propuesto en este ámbito, todos estos índices fueron desarrollados para ser eficientes en memoria principal. El *Historical-FHQT*(H-FHQT) [3] es un índice métrico-temporal que utiliza tanto la componente métrica como la temporal para resolver eficientemente búsquedas métrico-temporales. Este índice consiste en una lista de instantes válidos donde cada uno contiene un FHQT que indexa a todos los objetos vigentes en dicho instante. Los FHQT tienen distintas profundidades, es decir distintas cantidades de pivotes, en función de la cantidad de elementos que deban indexar.

Si bien en un H-FHQT la cantidad de pivotes en distintos instantes de tiempo varía, siempre se trabaja sobre el mismo conjunto base de pivotes; esto significa que si en el instante $i$ necesito $k_i$ pivotes y en el instante $j$ necesito $k_j$ pivotes con $k_i < k_j$, entonces los primeros $k_i$ pivotes de ambos instantes son iguales.

La figura 2 muestra un ejemplo de un H-FHQT construido sobre el intervalo de tiempo [1..12]. Se puede observar que el objeto $o_2$ está vigente en el intervalo de tiempo [4..6] y se encuentra a distancia 1 de $p_1$, a distancia 1 de $p_2$ y a distancia 3 de $p_3$.

# 3 Líneas de Investigación

## 3.1 Bases de Datos de Texto

Como mencionáramos en la sección anterior, en [9] se presenta una nueva representación de un trie de sufijos que permite reducir el espacio necesario para almacenar el índice, eliminando la necesidad de mantener los punteros explícitos a los hijos. Esta representación tiene la ventaja de permitir un posterior proceso de paginado para manejar eficientemente el trie

de sufijos en memoria secundaria.

Hemos diseñado e implementado una técnica de paginación para este índice basándonos en la representación secuencial del mismo. Nuestra técnica de paginación consiste en una extensión para árboles r-arios del paginado utilizado en el Compact Pat Tree [2]. Al igual que el algoritmo presentado en [2], nuestra técnica de paginado también particiona el árbol en componentes conexas, denominadas *partes*, cada una de las cuales se almacena en una página de disco.

El algoritmo procede en forma bottom-up tratando de condensar en una única parte un nodo con uno o más de los subárboles que dependen de él. En este proceso de particionado las decisiones se toman en base a la profundidad de cada nodo involucrado, donde la profundidad indica la cantidad de accesos a disco que deberá realizar el proceso de búsqueda para llegar desde esa parte a una hoja del árbol.

Para particionar un árbol, el algoritmo comienza asignando cada hoja a una parte con profundidad 1 y luego, en forma bottom-up, procesa cada uno de los nodos de este árbol r-ario según las reglas que se explican a continuación.

Sea $x$ el nodo corriente a procesar. Se ordenan los hijos del $x$ de mayor a menor según su profundidad, y para aquellos hijos de igual profundidad se ordenan de menor a mayor según su tamaño. Se pueden presentar los siguientes casos:

**Caso 1:** $x$ y su primer hijo de mayor profundidad $d$ entran en una página de disco:

• Se coloca en una misma parte $x$ y tantos hijos de $x$ como entren en una página, teniendo en cuenta en este proceso el orden ya establecido. Si ocurre que un hijo de $x$ tiene profundidad $j$ y no entra en la página, los siguientes hijos con profundidad $j$ tampoco entrarán (porque están ordenados por tamaños). En este caso se prosigue con los hijos con profundidad menor que $j$.

• Se aplica el mismo proceso hasta recorrer todos los grupos de distintas profundidades que existan para los hijos de $x$.

• Se cierran las partes de aquellos hijos que no conforman la nueva parte creada.

• Si todos los hijos de mayor profundidad $d$ se han agregado a la nueva parte creada, se establece que esta nueva parte tiene profundidad $d$.

• Si algún hijo de mayor profundidad $d$ es cerrado, la profundidad de la nueva parte se establece en $d + 1$.

**Caso 2:** $x$ y su primer hijo de mayor profundidad $d$ no entran en una página de disco. En este caso se cierran todas las partes hijas y se crea una nueva parte para el nodo corriente con profundidad $d+1$, donde $d$ es el máximo de las profundidades de los hijos.

Los resultados obtenidos con esta primer técnica nos ha permitido analizar el funcionamiento de la misma y nos encontramos diseñando mejoras a la misma.

Paralelemamente estamos trabajando sobre técnicas de compresión para el trie de sufijos, con el obejtivo final de implementar una versión compacta y en memoria secundaria de este índice.

## 3.2 Modelo Métrico-Temporal

En este ámbito nuestro obejtivo es el diseño de índices en memoria secundaria. Específicamente hemos estamos trabajando con el H-FHQT. El H-FHQT consiste en una lista de los instantes válidos de tiempo, donde cada celda de la lista contiene un índice FHQT [1] con el que indexa todos los objetos vigentes en dicho instante. Nuestra técnica de paginación tiene los siguientes casos a considerar:

**Caso 1:** La lista de instantes de tiempos válidos entra en memoria primaria pero cada árbol correspondiente a cada instante de

tiempo reside en memoria secundaria. En este caso hay dos situaciones a tener en cuenta:

**1a:** Cada árbol FHQT correspondiente a cada instante de tiempo entra en una página de disco. En este caso la paginación es directa, haciendo corresponder cada FHQT con una página de de disco.

**1b:** Cada árbol FHQT correspondiente a cada instante de tiempo no entra en una página de disco, en este caso se procede a paginarlo con la técnica propuesta en la sección anterior.

**Caso 2:** Ni la lista de instantes de tiempos válidos nia cada uno de los árboles FHQT correspondientes a cada instante de tiempo entran memoria principal. En este caso utilizamos para la lista de instantes de tiempo válido un árbol B (dado que allí se buscará por igualdad) y para los árboles FHQT usamos la técnica de paginado explicada en la sección anterior.

Nos encontramos implementando esta primer versión de paginado del H-FHQT.

# 4 Resultados Esperados

Se espera obtener índices eficientes, tanto en espacio como en tiempo, para el procesamiento de consultas en bases de datos textuales y en bases de datos métrico temporales. Los mismos serán evaluados tanto analíticamente como empíricamente.

# 5 Recursos Humanos

El trabajo desarrollado en esta línea forma parte del desarrollo de un Trabajo Final de la Licenciatura, dos Tesis de Maestría y una Tesis de Doctorado, todas ellas en el ámbito de la Universidad Nacional de San Luis.

# References


[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[2] D. Clark and I. Munro. Efficient suffix tree on secondary storage. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 383–391, 1996.

[3] A. De Battista, A. Pascal, G. Gutierrez, and N. Herrera. Un nuevo índice métrico-temporal: el historical-fhqt. In *Actas del XIII Congreso Argentino de Ciencias de la Computación*, Corrientes, Agentina, 2007.

[4] G. H. Gonnet, R. Baeza-Yates, and T. Snider. *New indices for text: PAT trees and PAT arrays*, pages 66–82. Prentice Hall, New Jersey, 1992.

[5] R. González and G. Navarro. A compressed text index on secondary memory. In *Proc. 18th International Workshop on Combinatorial Algorithms (IWOCA)*, pages 80–91. College Publications, UK, 2007.

[6] R. González and G. Navarro. Compressed text indexes with fast locate. In *Proc. 18th Annual Symposium on Combinatorial Pattern Matching (CPM)*, LNCS 4580, pages 216–227, 2007.

[7] A. Thomo M. Barsky *, U. Stege. A survey of practical algorithms for suffix tree construction in external memory. In *Software: Practice and Experience*, 2010.

[8] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal of Computing*, 22(5):935–948, 1993.

[9] D. Ruano and N. Herrera. Representación secuencial de un trie de sufijos. In *XX Congreso Argentino de Ciencias de la Computación*, Buenos Aires, Argentina, 2014.