Universidad Michoacana de San Nicolás de Hidalgo
Facultad de Ingeniería Eléctrica

# ÍNDICES DE PROXIMIDAD EN EL RECONOCIMIENTO DE VOZ

**TESIS**

Que para obtener el grado de
**INGENIERO EN COMPUTACIÓN**

presenta
**José Martín Ruiz Pérez**

**Dr. José Antonio Camarena Ibarrola**
**Director de Tesis**

Abril 2013

*A mis Padres y mi Hermano*

# Resumen

El proceso de búsqueda en el reconocimiento de voz es costoso debido a la dimensión de los datos, para bases de datos formadas por decenas de miles de palabras la búsqueda secuencial resulta una estrategia ineficiente. En este documento se abordo el problema de encontrar una palabra en una base de datos haciendo uso de estructuras de datos llamadas índices de proximidad y la similaridad entre objetos. Los índices que se implementaron son el BKT, FQT, FHFQT, FQA y el basado en permutantes. Los tiempos de respuesta que entregaron los índices son mejores que los obtenidos con la búsqueda secuencial. Por otra parte, se realizo una comparación del desempeño de cada índice tanto en rapidez como en precisión. El índice basado en permutantes es capaz de identificar la consulta realizando un número reducido de comparaciones, aunque no se garantiza que se trate de la palabra correcta. Índices como el FQA reconocen la palabra buscada en un tiempo de menos del segundo y con mucha precisión.

# Contenido


Dedicatoria . . . . . III
Resumen . . . . . V
Contenido . . . . . VII
Lista de Figuras . . . . . IX
Lista de Algoritmos . . . . . XI

1. Introducción 1
   1.1. Planteamiento del Problema . . . . . 1
   1.2. Antecedentes . . . . . 2
   1.3. Motivación . . . . . 3
   1.4. Objetivos de la Tesis . . . . . 3
      1.4.1. Objetivo general . . . . . 3
      1.4.2. Objetivos particulares . . . . . 4
   1.5. Conclusiones del Capítulo . . . . . 4
   1.6. Descripción de Capítulos . . . . . 4

2. Búsquedas en espacios métricos 5
   2.1. Espacios Métricos . . . . . 5
      2.1.1. Norma Vectorial . . . . . 6
      2.1.2. Ejemplos de Espacios Métricos y métricas . . . . . 6
   2.2. Búsquedas por proximidad . . . . . 8
   2.3. Conclusiones del Capítulo . . . . . 9

3. Índices de proximidad 11
   3.1. Índices basados en pivotes . . . . . 11
      3.1.1. Burkhard-Keller Tree (BKT) . . . . . 11
      3.1.2. Fixed Queries Tree (FQT) . . . . . 13
      3.1.3. Fixed Height Fixed Queries Tree (FHFQT) . . . . . 14
      3.1.4. Fixed Queries Array (FQA) . . . . . 15
   3.2. Índice basado en Permutantes . . . . . 16
   3.3. Conclusiones del Capítulo . . . . . 17

4. Sistema de reconocimiento de palabras aisladas usando índices 19
4.1. Metodología del Sistema . . . . . 19
4.2. Implementación de los índices . . . . . 21
4.2.1. BKT . . . . . 21
4.2.2. FQT . . . . . 25
4.2.3. FHFQT . . . . . 29
4.2.4. FQA . . . . . 35
4.2.5. Permutaciones . . . . . 37
4.3. Conclusiones del Capítulo . . . . . 38

5. Experimentos y Resultados 39
5.1. Ambiente de trabajo . . . . . 39
5.2. Validaciones de la desigualdad triangular . . . . . 40
5.3. Comparación del tiempo de respuesta de los índices y la búsqueda secuencial 40
5.4. Comparación del tiempo de respuesta de los índices . . . . . 41
5.5. Comparación de la precisión de los índices . . . . . 42
5.6. Conclusiones del Capítulo . . . . . 43

6. Conclusiones y Trabajos Futuros 45
6.1. Conclusiones Generales . . . . . 45
6.2. Trabajos Futuros . . . . . 46

Referencias 47

Glosario 51

# Lista de Figuras


2.1. Desigualdad del triangulo. . . . . 6
2.2. Círculos unitarios. . . . . 7
2.3. Consulta de rango en $\mathbb{R}^2$. . . . . 8

3.1. BKT . . . . . 12
3.2. FQT . . . . . 14
3.3. FHFQT . . . . . 15
3.4. FQA vs FHFQT . . . . . 16
3.5. Permutaciones de los elementos de la BD. . . . . 17

4.1. Etapas del sistema. . . . . 20
4.2. Rango de distancias para cada arista, $b = 3$ y NumHijos= 4. . . . . 21
4.3. Diagrama UML de la clase BKT. . . . . 22
4.4. Diagrama UML de las clases FQT y FHFQT. . . . . 30
4.5. Insertar elementos en el FQT. (a) Se instancia y se inserta uno. (b) Se insertan dos y tres consecutivamente. (c) El *bucket* esta lleno por tanto se instancia un nodo interno, se elige aleatoriamente un elemento de la BD como $p_1$ y se verifica tantas veces sea necesario que los elementos se inserten sin problemas. (d) Se inserta ocho y como hay *bucket* lleno se repite el proceso anterior. . . 31
4.6. Diagrama UML de la clase FQA. . . . . 35
4.7. Diagrama UML de las clases del indice basado en permutantes. . . . . 37

5.1. Comparación del tiempo de respuesta usando búsqueda secuencial contra búsqueda indexada. . . . . 41
5.2. Comparación del tiempo de respuesta de cada índice. . . . . 42
5.3. Comparación de la precisión de cada índice. . . . . 43

# Lista de Algoritmos


1. Algoritmo para insertar un elemento en el árbol. . . . . 23
2. Algoritmo de búsqueda por rango. . . . . 24
3. Algoritmo para el auxiliar de la búsqueda por rango. . . . . 25
4. Pseuducódigo: Método que corrige la inserción si el *bucket* está lleno. . . . . 26
5. Pseudocódigo: Método que recorre el árbol e inserta. . . . . 27
6. Pseudocódigo del método Inserta. . . . . 28
7. Pseudocódigo: Método Busca. . . . . 30
8. Pseudocódigo: Método que busca nodos hoja y regresa el menor. . . . . 32
9. Pseudocódigo: Método que inserta un nuevo elemento al FHFQT. . . . . 33
10. Pseudocódigo: Método que recorre el árbol y se se posiciona en el penúltimo nivel para hacer la inserción. . . . . 34
11. Pseudocódigo del auxiliar de la búsqueda por rango en el FQA, regresa *elemmin*. 36

Capítulo 1

# Introducción

## 1.1. Planteamiento del Problema

El reconocimiento de voz es una área dentro del reconocimiento de patrones, y ha estado en la mira de la investigación desde mediados del siglo pasado [Davis52], la motivación que ha mantenido los ojos de los científicos en esta rama es llegar a la instancia donde una maquina pueda ser capaz de hacer reconocimiento continuo de gran vocabulario y también entienda su significado, como lo plantea [Lawrence Rabiner93]. A lo largo de este tiempo se han ido desarrollado sistemas capaces de reconocer vocales, consonantes, hasta llegar a sistemas que reconocen palabras aisladas y de estos pasar al reconocimiento continuo de voz. Todos los sistemas anteriores como cualquier proceso de reconocimiento de patrones pasan por la etapa de extracción de características de la señal de voz, estas pueden ser almacenadas en un diccionario* y son las que nos permiten realizar comparaciones contra una nueva entrada y así tomar una decisión dependiendo del parecido entre ellas. Para poder realizar la comparación se hace uso de una medida de similaridad o distancia; la alta dimensión de los vectores de características extraídos de la señal de voz (secuencias de vectores) hace que sea costoso el cómputo de la distancia, por tanto, si pensamos en diccionarios muy grandes, resulta prohibitivo hacer una búsqueda secuencial.

Es importante revisar estructuras de datos basadas en modelos matemáticos que

* Es la base de datos utilizada, contiene las las características extraídas de la señal de voz. Capítulo 5.

nos permiten reducir el tiempo de consulta para que sea posible la implementación de sistemas automáticos de reconocimiento de palabras eficientes.

## 1.2. Antecedentes

Como ya dijimos hace mas de medio siglo comenzó la investigación sobre el reconocimiento de voz, con forme pasaron los años esta área de la mano con el desarrollo de la tecnología generó grandes avances que derivaron en una gama de sistemas cada vez más robustos que nos permiten identificar perfectamente una palabra en un vocabulario ([Lawrence Rabiner93], [Lee11]).

Este paradigma esta ligado a los algoritmos de búsqueda. En el 73, Donald Knuth planteo el problema del correo [Knuth98] que consistía en dado un conjunto de puntos en el plano, encontrar el más cercano a un punto de consulta $q$ y en ese año Burkhard-Keller describía un índice para buscar archivos [Burkhard73]. En los años 90 se presentan en [Bahl92] y después en [Lee11] alternativas que aceleran las búsquedas haciendo uso de Modelos Ocultos de Markov (HMM) en combinación con un árbol donde en cada nodo interno se etiqueta un fonema y en cada hoja se encuentra una palabra pronunciada. Durante la consulta se podan las ramas dependiendo de un umbral basado en el parecido fonético con el segmento actual de la palabra. Esta técnica regresa una lista de elementos pequeña con los elementos mas parecidos a la palabra de entrada con la cual se realiza un comparación directa. También en estos años se comienzan a usar espacios métricos en bases de datos multimedia (e. g. voz). En este enfoque se toma la distancia como el valor a aproximar en una búsqueda de rango o del vecino más cercano ([Baeza-Yates94], [Baeza-Yates97], [Chávez01a], [Figueroa06]).

## 1.3. Motivación

La realización de sistemas de reconocimiento de voz eficientes busca como cualquier otra área de la investigación, ser una herramienta al alcance y para el beneficio de la sociedad. El fin de estos es contar con aplicaciones que permitan sobre todo la interacción del usuario con la máquina (de cualquier propósito) haciendo posible la manipulación de equipos como sistemas empotrados (automóviles, juguetes con fines didácticos, etc.), por medio de la voz humana, esto se vuelve crucial cuando pensamos en aplicaciones que son un apoyo para personas con discapacidades físicas.

Algunas aplicaciones están orientadas a organizaciones privadas o instituciones sin fines de lucro como instituciones de educación; en las primeras se utilizan sistemas de información donde se quiere dar un seguimiento a un usuario mediante sistemas telefónicos [JTA13], o sistemas de seguridad como identificación de individuos por su voz que son útiles en entornos donde se requiere tener un control del personal que tiene acceso a determinada área. También existen aplicaciones para manipular computadoras personales, dispositivos móviles, etcétera, que convierten las palabras pronunciadas en instrucciones especificas, estas aplicaciones son de propósito general es decir, van dirigidas para cualquier persona que cuente con el dispositivo. Además de *software* especializado como la *secretaria virtual*, que es capaz de ser entrenado por el usuario final y convertir su voz en texto [DRA13], potenciando actividades que se realizan comúnmente en determinados puestos laborales, este es otro aspecto en el que tiene impacto el reconocimiento de voz permitiendo al usuario agilizar sus tareas favoreciendo el desempeño y por tanto reduciendo el esfuerzo extra.

## 1.4. Objetivos de la Tesis

### 1.4.1. Objetivo general

Realizar una comparación de las diferentes estrategias de búsqueda rápida en diccionarios conformados por secuencias de vectores de características (y su etiqueta correspondiente) extraídos previamente de un conjunto de palabras conocidas por el sistema de reconocimiento.

### 1.4.2. Objetivos particulares

- Implementar sistemas de reconocimiento de palabras aisladas que hagan uso de índices de proximidad específicos.
- Comparar los desempeños de los diferentes índices en cuanto a tiempo de respuesta y precisión.

## 1.5. Conclusiones del Capítulo

Las búsquedas por proximidad desde su planteamiento en 1973 han mostrado ser la mejor alternativa para encontrar en un conjunto de datos muy extensos donde los miembros tienen dimensiones grandes (bases de datos de palabras), un conjunto reducido que contenga los elementos más parecidos a nuestro elemento de consulta y en un tiempo menor debido a que no se valida la totalidad de los elementos.

## 1.6. Descripción de Capítulos

El **Capítulo 2** da una introducción a las bases teóricas que apoyan a los índices: modelos matemáticos, como lo son los espacios métricos, así como a los algoritmos básicos para realizar búsquedas o consultas en ellos.

El **Capítulo 3** expone diferentes índices, con el fin de dar a conocer su estructura ademas de sus posibles ventajas y desventajas.

El **Capítulo 4** describe la propuesta de un sistema de reconocimiento de palabras aisladas que es extendido para utilizar índices de proximidad.

El **Capítulo 5** muestra los resultados que se obtuvieron después de haber hecho los experimentos de búsqueda por fuerza bruta y en la que se usan los índices, así como comparaciones de precisión y rapidez en la respuesta entre ellos.

El **Capítulo 6** presenta las conclusiones del trabajo, mejoras que se pueden hacer al sistema y aplicaciones de los índices para otro tipo datos.

Capítulo 2

# Búsquedas en espacios métricos

## 2.1. Espacios Métricos

Un espacio métrico es un conjunto de objetos que tiene asociada una métrica (función distancia) la cual establece una distancia entre dos integrantes.

Definición formal:
Un espacio métrico es un par ordenado $(M, d)$ donde $M$ es un conjunto de objetos y $d$ es una métrica en $M$ y se define de la siguiente manera; para $x, y, z \in M$, $d$ debe cumplir las siguientes propiedades:

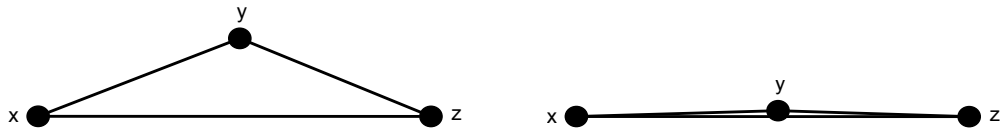
1. Estricta positividad $d(x, y) \geq 0$

2. Simetría $d(x, y) = d(y, x)$

3. Reflexividad $d(x, x) = 0$

4. Desigualdad triangular $d(x, z) \leq d(x, y) + d(y, z)$

La desigualdad del triangulo expresa que la distancia de $x$ a $z$ puede ser a lo más igual que su distancia a través de $y$, la Figura 2.1 ilustra la condición anterior.




**(a)** $d(x,z) < d(x,y) + d(y,z)$ **(b)** $d(x,z) \approx d(x,y) + d(y,z)$

Figura 2.1: Desigualdad del triangulo.

### 2.1.1. Norma Vectorial

La norma es una función u operador que asigna un tamaño o longitud estrictamente positiva a todos los vectores en un espacio vectorial. Este operador nace de la necesidad particular de áreas de la ciencia como Física y Geometría. Un ejemplo de una definición de norma es la norma-$p$ que se define a continuación, para $x \in M$ donde $M$ es un espacio vectorial la norma-$p$ de $x$ es:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p} \tag{2.1}$$

### 2.1.2. Ejemplos de Espacios Métricos y métricas

Un espacio vectorial normado (es decir que cuenta con una función norma), es un ejemplo de espacio métrico, esto es debido a que las normas en espacios vectoriales son equivalentes a ciertos tipos de métricas, que son las homogéneas e invariantes a la traslación, definimos esta condición como sigue $d(x,y) = \|y - x\|$. En otras palabras cada norma determina una métrica, y algunas métricas determinan una norma. Un ejemplo de esto seria decir que el espacio norma-$p$ Ec. 2.1 o espacio $L_p$ ($p = 1, 2, \cdots, \infty$) es un espacio métrico y su métrica también llamada distancia de Minkowski está definida por la Ec. 2.2.

$$d_p(x,y) = \left( \sum_{i=1}^{\delta} |x_i - y_i|^p \right)^{1/p} \quad \text{donde } \delta \text{ es el número de dimensiones de } x \text{ y } y \tag{2.2}$$

Las distancias Manhattan, Euclidiana y de Chebyshev o máxima son norma-1, norma-2 y norma-$\infty$ respectivamente es decir son casos particulares de la distancia $L_p$ y en la Figura 2.2 se muestran sus círculos unitarios. La distancia máxima se define a continuación:

$$d_\infty(x,y) = max_{i=1}^{n} |x_i - y_i| \tag{2.3}$$

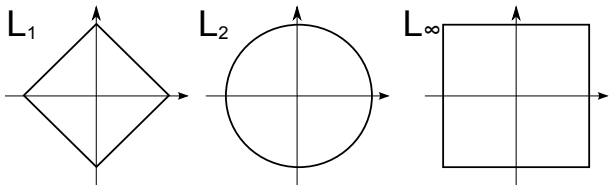



Figura 2.2: Círculos unitarios.

La distancia Coseno (la definición fue estudiada en [Camarena11]) es una función distancia que mide el angulo entre vectores, el producto punto entre dos vectores se define a continuación:

$$x \cdot y = |x||y| \cos\theta \tag{2.4}$$

donde $\theta$ es el angulo menor que hay entre los vectores $x$ y $y$

$$\cos\theta = \frac{X \cdot Y}{|x||y|} = \frac{x_1y_1 + x_1y_1 + \cdots + x_ny_n}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}} \tag{2.5}$$

$$\cos\theta = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{2.6}$$

Hasta ahora esta función es una medida de similitud y no precisamente una distancia ya que mientras menos se parecen los vectores esta nos entrega valores más cercanos a 0, como en el rango de la función cos el valor mas grande es 1.0 podemos entonces convertirla en distancia como sigue:

$$d_{\cos}(x, y) = 1 - |\cos\theta| = 1 - \left| \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \right| \tag{2.7}$$

## Distancias Semimétricas

Una distancia semimétrica es aquella que no cumple con la desigualdad del triangulo, sin embargo en este grupo existen distancias que tienen una fuerte relación con esta propiedad, es decir, que en la mayoría de los casos se cumple que para tres objetos $A$, $B$ y $C$, $d(A, C) \leq d(A, B) + d(B, C)$.

El *Dynamic Time Warping* (existe una descripción sobre esta distancia en [Camarena11]) (DTW) es una distancia muy usada en el reconocimiento de palabras y en la identificación

de individuos [Rabiner78]. El motivo de su uso es que nuestros objetos en la base de datos no son de las mismas dimensiones por lo que se requiere realizar un alineamiento para poder obtener un valor de distancia. El DTW se construye usando una métrica esto con la finalidad de cumplir con la desigualdad del triangulo. En el Capítulo 5 se realiza un experimento para validar que el DTW cumple fuertemente con la desigualdad del triangulo.

## Preludio a la siguiente sección

En la sección anterior establecimos los modelos matemáticos en los que están basados nuestros elementos de BD. Ahora necesitamos métodos que nos permitan hacer búsquedas rápidas pero también precisas en estos modelos, y para eso usaremos las basadas en proximidad a una consulta.

## 2.2. Búsquedas por proximidad

Las búsquedas por proximidad o similaridad se pueden entender de dos tipos. Definimos la BD como un conjunto $\mathbb{U} \subseteq \mathbb{X}$ de tamaño $n$, y definimos la consulta como $q$, que es un elemento de $\mathbb{X}$.

**Consulta de rango**, dado un radio $r$, se regresan todos los elementos dentro de este, e. g., $(q,r)_d = \{u \in \mathbb{U} : d(q,u) \leq r\}$. En este trabajo haremos uso de la búsqueda por rango (que es la más común) para consultar a la BD. La Figura 2.3 muestra como funcionaría en un espacio euclidiano bidimensional.

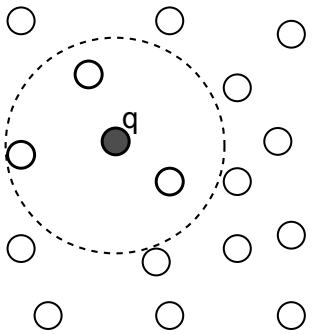



Figura 2.3: Consulta de rango en $\mathbb{R}^2$.

**Consulta del vecino más cercano**, regresa los elementos más cercanos a la consulta, e. g., $nn(q)_d = \{u \in \mathbb{U} : d(q,u) \leq d(q,v),\ \forall v \in \mathbb{U}\}$. Es directo extenderlo a $k$ vecinos.

## 2.3. Conclusiones del Capítulo

En la sección anterior se presentaron tipos de búsquedas que resuelven de una mejor manera las búsquedas en espacios métricos dejando a un lado la búsqueda por fuerza bruta o búsqueda secuencial, esto debido a que el computo de la distancia entre dos elementos de multiples dimensiones es muy costoso, ahora extrapolando esta comparación a BD de millones de elementos resulta indeseable tal implementación. Existen estructuras de datos como el BKT, FQT, FHFQT, FQA y el índice basado en permutantes que nos permiten ademas de almacenar en memoria nuestros elementos de BD usar las búsquedas por proximidad.

Capítulo 3

# Índices de proximidad

Los índices de proximidad son estructuras de datos en las cuales nuestros elementos de BD son almacenados siguiendo cierto orden o recorrido, es decir, de tal manera que se harán notablemente menos comparaciones, *UNIFYING MODEL* [Chávez01b].

Existen varios tipos de índices, los basados en pivotes o llaves que son los elementos con los que se compararan los demás para saber que camino seguir o que región visitar. Los basados en permutantes que fungen una función muy similar a los pivotes en el sentido de que son una referencia tanto para indexar como para consultar, pero en esta familia se usan para crear permutaciones de ellos para cada miembro de BD. Los basados en familias de funciones hash, en estos se piensa que si un vector es muy parecido a otro hay una probabilidad muy grande de que sean mapeados al mismo *bucket* dentro de una tabla hash.

## 3.1. Índices basados en pivotes

### 3.1.1. Burkhard-Keller Tree (BKT)

El BKT [Burkhard73], es una estructura de datos de árbol $m$-ario, este árbol dado que todos sus nodos son pivotes cada que se avanza en él se debe volver a calcular una distancia. Para mejorar el desempeño de un búsqueda por proximidad en las siguientes secciones se presentan estructuras de datos que ahorran comparaciones.

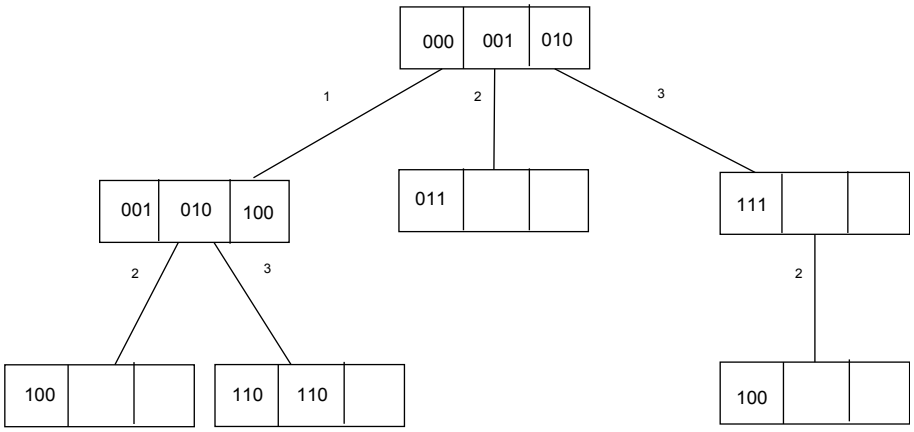



Figura 3.1: BKT

**Definición**

Sea $\mathbb{U}$ el conjunto de todos los elementos posibles que puede abarcar nuestro problema. Asumiremos que $\mathbb{X} \subset \mathbb{U}$ es un conjunto grande y que $\mathbb{U}$ es más grande que $\mathbb{X}$. Sea $d$ una función distancia que satisfaga la desigualdad del triangulo y asumamos que el conjunto de todas las distancias posibles es un conjunto finito $\{d_1,\ d_2,\ \ldots,\ d_s\}$. El BKT formado a partir de un conjunto $\mathbb{X}$ y una función $d$ cumple las siguientes características. Todos los elementos de $\mathbb{X}$ son pivotes a excepción de las hojas y si $\mathbb{X}$ contiene solamente $b$ ($b$ es el tamaño del *bucket*) elementos, entonces el árbol consiste en solamente un nodo hoja conteniendo todos los elementos.

La Figura 3.1 muestra un ejemplo de un BKT con $b = 3$, sus elementos son cadenas de bits, la métrica asociada a este espacio es la distancia de Hamming $d_H$, el primer elemento del *bucket* es el pivote por convención. Para insertar 011, se verifica si hay espacio en el *bucket* de la raíz, debido a que no hay espacio se calcula $d_H(011, 000)$ y se desciende por la rama etiquetada por el valor que regreso $d_H$. En el nodo actual no hay elementos por tanto ahí se inserta.

La búsqueda de una consulta $q$ consiste en calcular la distancia entre $q$ y el pivote del nodo raíz (primer elemento de ese *bucket*), y desciende por la rama que tenga etiquetada la misma distancia, este proceso se repite para cada nodo (pivote) al que descendamos y termina hasta encontrar una hoja. Esto para una búsqueda exacta, para una búsqueda por

proximidad resulta un poco más complicado debido a su naturaleza recursiva. Primero al igual que la búsqueda exacta se compara con el pivote del nodo raíz $d(q, p_1) = d_1$, todos las ramas que estén dentro del rango $d_1 - r \leq d \leq d_1 + r$ serán recorridas recursivamente.

### 3.1.2. Fixed Queries Tree (FQT)

En [Baeza-Yates94] se presenta una estructura de datos basada en árboles de búsqueda, este índice presenta una mejor alternativa sobre el BKT debido a que solo se cuenta con un pivote por cada nivel del árbol, esto reduce el número de distancias evaluadas ya que para una consulta se almacena un vector con las distancias de esta hacia todos los pivotes.

#### Definición

Sea $\mathbb{U}$ el conjunto de todos los elementos posibles que puede abarcar nuestro problema. Asumiremos que $\mathbb{X} \subset \mathbb{U}$ es un conjunto grande y que $\mathbb{U}$ es más grande que $\mathbb{X}$. La diferencia entre un BKT y un FQT es que todos los nodos internos en cierto nivel tienen asociado el mismo pivote. Cuando hablemos de elementos nos referimos a los miembros de $\mathbb{X}$. Sea $d$ una función distancia que satisfaga la desigualdad del triangulo y asumamos que el conjunto de todas las distancias posibles es un conjunto finito $\{d_1,\ d_2,\ \ldots,\ d_s\}$. Un FQT para un conjunto de elementos $\mathbb{X}$ y una función distancia $d$ es un árbol que satisface las siguientes cuatro propiedades:

- Todos los elementos de $\mathbb{X}$ están asociados con las hojas.
- Si $\mathbb{X}$ contiene solamente $b$ elementos ($b$ es el tamaño del *bucket*), entonces el árbol consiste en solamente un nodo hoja conteniendo todos los elementos.
- Todos los nodos internos de profundidad $r$ están asociados con el mismo pivote $p_r$. Los pivotes pueden ser seleccionados aleatoriamente, también pueden ser miembros de $\mathbb{X}$.
- Cada nodo interno $v$ es una raíz de un FQT valido asociado con el conjunto $\mathbb{X}_v \subset \mathbb{X}$. Un nodo interno $v$ con pivote $p_r$ tiene un subarbol por cada conjunto no vacío $\mathbb{X}_i$

definido por $\mathbb{X}_i = \{x \in X_v \ : \ d(p_r, x) = d_i \geq 0\}$.

La figura 3.2 muestra un ejemplo de un FQT con $b = 3$. Para insertar 1110 se creo un nodo interno y se eligió aleatoriamente de $\mathbb{X}$ un nuevo pivote, debido a que el *bucket* estaba lleno. Los elementos que pertenecían al *bucket* y el nuevo elemento son reubicados calculando $d_H$ con $p_2$.

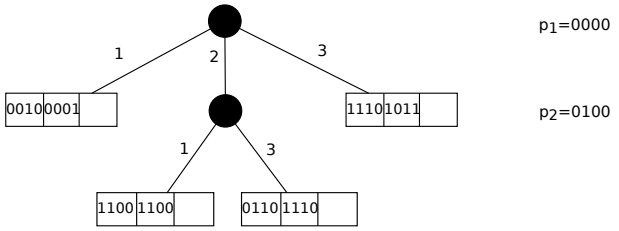



Figura 3.2: FQT

La búsqueda de una consulta $q$ consiste en calcular la distancia entre $q$ y los $k$ pivotes, y descender por la rama que este etiquetada con la distancia hacia el $i$-ésimo pivote en ese nivel, por ejemplo para el primer pivote $p_1$ obtenemos $d_1 = d(q, p_1)$ descendemos por tal rama, y así sucesivamente, termina hasta encontrar una hoja. Esto para una búsqueda exacta, para una búsqueda por proximidad al igual que en el BKT hacemos uso de la recursividad para regresar los elementos que encontremos durante el recorrido. Primero obtenemos un vector con las distancias hacia todos los pivotes $(d(q, p_1), \ldots, d(q, p_k))$, en el árbol para cada nivel $i$ descendemos recursivamente por todas las ramas que estén etiquetadas con una distancia $d$ dentro del rango $d_i - r \leq d \leq d_i + r$.

### 3.1.3. Fixed Height Fixed Queries Tree (FHFQT)

El FHFQT [Baeza-Yates97], es una mejora al FQT, la cual propone fijar la altura para todas las ramas (todas las hojas estarán al mismo nivel) y aumentarla, a pesar de que se está aumentando el número de distancias evaluadas, el costo es pequeño ya que se cuenta con la regla de un pivote por nivel y esto provocaría que, aun para un tamaño de *bucket* de 1, los elementos se distribuyen en las hojas. Imaginemos que se construyera un FQT y que varias ramas de este crecieran un número considerable de veces más que las otras, para

ciertas consultas la respuesta sería mucho más rápida y para otros mucho más lenta. Otra mejora remarcable de este índice es que consigue balancear el árbol para igualar tiempos de respuesta.

La figura 3.3 muestra un ejemplo de un FHFQT con $b = 3$. Para insertar 0110 se desciende por las ramas que tengan etiquetado el valor $d_H(0110, p_i)$. Una vez que se haya alcanzado el ultimo nivel se inserta 0110.

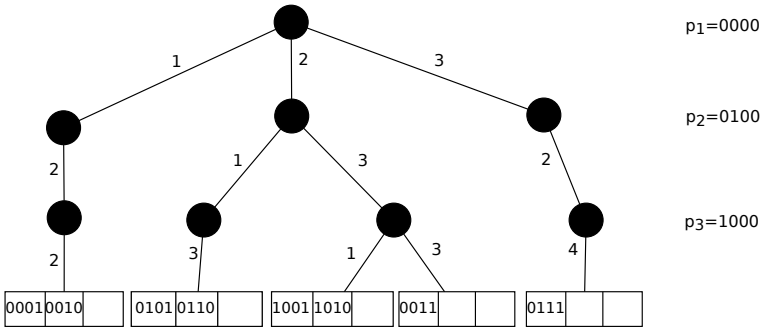



Figura 3.3: FHFQT

Los algoritmos de búsqueda tanto exacta como por proximidad en este árbol son heredados del FQT.

### 3.1.4. Fixed Queries Array (FQA)

El FQA [Chávez01a], es una estructura de datos basada en un arreglo como su nombre lo dice, a parte de utilizar poca memoria la cual está en función del número de pivotes elegidos, ademas esta flexibilidad beneficia que los elementos estén mas ordenados y por ende la consulta sea mas eficiente.

Primero que nada se asume que el rango de distancias que se pueden obtener es finito. Partiendo de esto para cada elemento en la BD se guarda una lista con las distancias hacia los pivotes. Esta lista es una secuencia de $k$ enteros. El índice almacena los elementos y los ordena lexicográficamente, es decir en una primera instancia son ordenados en orden creciente respecto a su distancia al primer pivote, aquellos que se encuentren a la misma distancia en ese nivel son ordenados en base a la distancia al segundo pivote y así sucesivamente. Una vez terminado el ordenamiento podemos hacer una analogía entre el FHFQT

y el FQA, la Figura 3.4 muestra tal semejanza. Es por esto que el FQA tiene una fuerte relación con el FHFQT, inclusive el FQA hereda del FHFQT el algoritmo de búsqueda. Cada nodo del FHFQT es un rango de celdas en el FQA. Si un nodo desciende de otro en el árbol, su rango es un subrango de otro en el arreglo. Entonces podemos decir que el movimiento espejo de descender de un nodo a un hijo en el árbol corresponde a hacer una búsqueda binaria del nuevo rango dentro del actual. Esta búsqueda únicamente agrega un factor $O(\log n)$ al tiempo de respuesta.

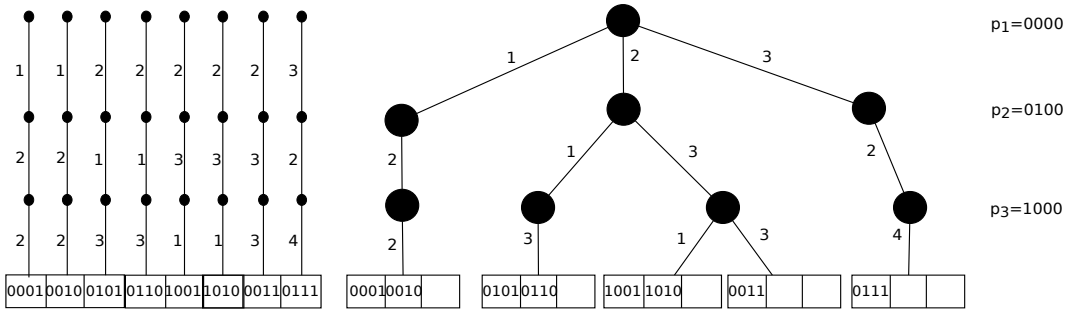



Figura 3.4: FQA vs FHFQT

La búsqueda de una consulta $q$ consiste en calcular la distancia entre $q$ y los $k$ pivotes, y emular el descender de un nodo a su hijo buscando el subrango (dentro del rango actual en el arreglo) de los elementos que tengan la etiqueta dada por la distancia hacia el $i$-ésimo pivote en ese nivel, por ejemplo para el primer pivote $p_1$ obtenemos $d_1 = d(q, p_1)$ y buscamos el subrango, y así sucesivamente, este proceso termina hasta haber alcanzado el $k$-ésimo pivote. Para una búsqueda por proximidad primero obtenemos el vector de distancias hacia los pivotes $(d(q, p_1), \ldots, d(q, p_k))$, para cada nivel buscaremos recursivamente el subrango (dentro del rango actual) que esté a una distancia $r$ de $q$ es decir $d_i - r \leq d \leq d_i + r$, regresando todos los elementos cada vez que visitemos una hoja es decir cuando nos encontremos en el último nivel del árbol.

## 3.2. Índice basado en Permutantes

Un índice basado en permutaciones [Chávez08], es una alternativa a los basados en pivotes, en este índice los elementos de referencia se llaman permutantes y cada objeto de

BD ordena el espacio como lo percibe dependiendo de la similaridad hacia los permutantes.

Inicialmente se seleccionan (aleatoriamente) un conjunto $\mathbb{P} = \{p_1,\ p_2,\ \ldots,\ p_k\} \subseteq \mathbb{U}$, $|\mathbb{P}| = k$. Cada elemento de BD $u \in \mathbb{U}$ define un preorden $\leq_u$ en los elementos de $\mathbb{P}$ definido por la distancia a $u$: $p_1 \leq_u p_2 \Leftrightarrow d(u, p_1) \leq d(u, p_2)$. Se asocia $\leq_u$ a una permutación $\prod_u$ de $p_1,\ p_2,\ \ldots,\ p_k$, para este caso donde $p_i \leq_u p_{i+1}$. Los elementos a la misma distancia toman un lugar consistente en la permutación. Sea $\prod^{-1}(p)$ la posición del permutante $p$ en $\prod$. La Figura 3.5 muestra la representación del proceso descrito anteriormente en $\mathbb{R}^2$.

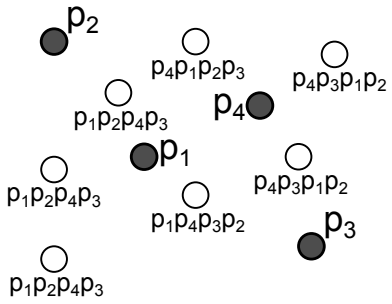



Figura 3.5: Permutaciones de los elementos de la BD.

Los algoritmos de búsqueda [Chávez08], nos presentan dos perspectivas, el que nos ocupa es el probabilista el cual consiste en una primera instancia obtener la permutación de la consulta $q$, después para cada elemento de la BD mediremos la similaridad de su permutación contra la de $q$ usando Spearman Rho Ec. (3.1), a continuación se ordenan los elementos respecto a esta medida, esto con el fin de comenzar a comparar $q$ con los elementos que están mas cercanos a ella y así reducir el costo computacional en términos del número de distancias computadas permitiendo terminar la búsqueda más rapidamente.

$$S_\rho(\prod_q, \prod_u) = \left(\sum_{i=1}^{k} |\prod_u^{-1}(p_i) - \prod_q^{-1}(p_i)|^2\right)^{1/2} \tag{3.1}$$

## 3.3. Conclusiones del Capítulo

La estructura de la mayoría de los índices basados en pivotes es un árbol, por lo tanto el recorrido tanto para insertar como para buscar, es similar por ejemplo al de un árbol de búsqueda. El FQA a pesar de no ser un árbol homologa el comportamiento de este

para realizar la búsqueda. El índice basado en permutaciones usa un esquema probabilista para determinar que elementos son más trascendentes para la consulta y comienza a realizar comparaciones con ellos. En este trabajo se implementaran los índices que se explicaron en este Capítulo. Es importante mencionar que dado que se usara el DTW como distancia, los índices usaran la similaridad para dar una respuesta aproximada a la consulta.

Capítulo 4

# Sistema de reconocimiento de palabras aisladas usando índices

El Capítulo 3 presenta la estructura y operaciones básicas de los diferentes índices, en este Capítulo se expondrán estas mismas características desde el enfoque de su implementación.

## 4.1. Metodología del Sistema

El sistema de reconocimiento de palabras aisladas fue implementado con el lenguaje de programación orientado a objetos **Java**[*], está conformado por tres etapas principales que ademas denotan el flujo de ejecución del mismo, Figura 4.1:

1. Etapa de captura, digitalización y segmentación.
2. Etapa adaptable de extracción de características (Implementación de varios esquemas).
3. Etapa adaptable (Escribe al diccionario o consulta al diccionario).

La primera etapa entrega una señal de voz ya digitalizada como entrada a la siguiente.

[*] http://www.java.com/

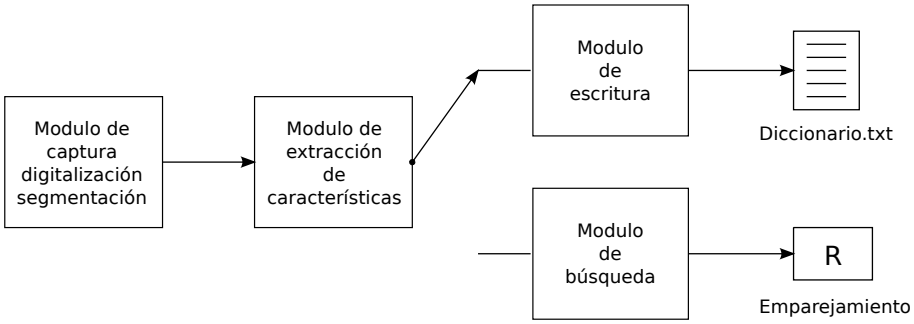



Figura 4.1: Etapas del sistema.

En la segunda etapa entra en ejecución el modulo* de extracción de características, existen varios esquemas para extraer características tales como:

- Energías por bandas de Bark (Reconocimiento por espectrogramas).
- MFCC's (Reconocimiento usando los MFCC's).
- LPC (Reconocimiento basado en coeficientes LPC).

Estos son implementados en el modulo de extracción de características. Para este trabajo se usaron los MFCC's (método estudiado en [Camarena11]) para crear el diccionario.

La tercera etapa ejecuta dos módulos pero no ambos a la vez y define el contexto de nuestro sistema ya que esta puede trabajar en la creación o en la consulta de nuestro diccionario o BD. Es decir como si realizara una demultiplexación entre los módulos de escritura y el de búsqueda.

El modulo de búsqueda es el encargado de implementar el método `busca` (consulta) al diccionario. Una implementación para este modulo podría ser una búsqueda secuencial [Camarena11], la cual resulta ineficiente para nuestro propósito debido a la naturaleza de nuestros datos. En lugar de esto el método va a ser implementado por la función `busca` de un índice.

* Cada modulo implementa un método

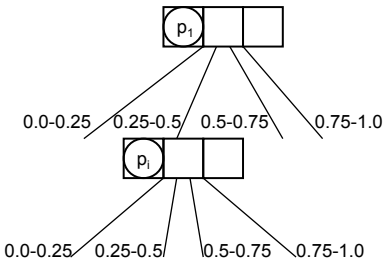



Figura 4.2: Rango de distancias para cada arista, $b = 3$ y NumHijos= 4.

## 4.2. Implementación de los índices

### 4.2.1. BKT

Un BKT es un árbol que tiene $n$ hijos ademas de un tamaño de *bucket*, también es importante decir que cada arista no esta etiquetada por un solo número entero. El valor de las distancias va a ser normalizado de 0.0 a 1.0 utilizando los valores extremos que son observados durante la creación de la BD. Por tanto cada arista estará etiquetada por un rango que sera un múltiplo de la razón de 1.0 entre $n$ hijos, Figura 4.2. A continuación se muestra la estructura propuesta en lenguaje Java solo por brindar más claridad en la implementación que se hizo así como los diagramas UML de la clase, Figura 4.3:

```
public class BucketBKT {
  String palabra;
  double[][] datos;
}

public class NodoBKT {
  BucketBKT[] bucket;
  double rango;
  NodoBKT[] hijos;

  public NodoBKT(int nh, int b) {
    bucket = new BucketBKT[b];
    hijos = new NodoBKT[nh];
    rango = 1.0 / hijos.length;
  }
}

// Clase BKT
public class BKT {
  NodoBKT raiz;

  public BKT(int nh, int b) {
    raiz = new NodoBKT(nh, b);
  }
}
```

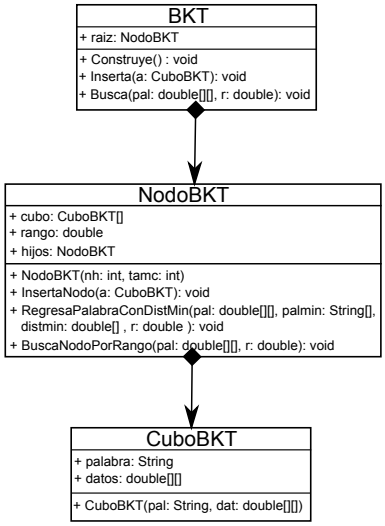



Figura 4.3: Diagrama UML de la clase BKT.

Del código y los diagramas vemos la declaración de los métodos que nos interesan, `InsertaNodo`, `BuscaNodoPorRango` y el auxiliar `RegresaPalabraConDistMin` que son para la creación del indice y consulta a la BD respectivamente, a continuación se listan los algoritmos definidos para cada uno de ellos, Algoritmo 1, 2 y 3.

### Inserción

El proceso de inserción está basado en un recorrido recursivo que empieza visitando el nodo padre, después avanza por su hijos de izquierda a derecha. El nuevo elemento debe insertarse en la posición del *bucket* que esté desocupada, en el caso de que el *bucket* este lleno el nuevo elemento se posicionara en la arista que le sea asignada una vez tomada la distancia con respecto al pivote en ese nodo. Los pivotes en el BKT son los elementos en el primer espacio del *bucket*.

En las lineas 3-9 del Algoritmo 1 se busca un espacio vacío para colocar el nuevo elemento en el nodo actual, en caso de que esto se logre con éxito la bandera `bvi` es puesta en falso para que el método regrese. Las lineas 11-19 realizan el proceso de descender por las ramas que cumplan la desigualdad $A_j \leq d(p_i, x) < A_{j+1}$ de que la distancia del elemento nuevo al $i$-ésimo pivote está entre el rango de la $j$-ésima arista.

**Algoritmo 1** Algoritmo para insertar un elemento en el árbol.

```
InsertaNodo(a)
 1  bvi ← verdadero
 2  para j ← 0 hasta b
 3    {
 4    si bucket[j] == NULO
 5       entonces bucket[j] ← a
 6                bvi ← falso
 7                sal del ciclo
 8    }
 9    si bvi == verdadero
10       entonces para j ← 0 hasta NumHijos
11                   {
12                   si A_j ≤ d(p_i, x) < A_{j+1}
13                      entonces si hijos[j] == NULO
14                                   entonces hijos[j] ← nuevo nodo
15                                            hijos[j].InsertaNodo(a)
16                                            regresar
17                                   sino     hijos[j].InsertaNodo(a)
18                   }
```

**Búsqueda**

La búsqueda no es la convencional para un árbol aunque sigue un proceso recursivo muy parecido, en cada nodo conservamos el elemento del *bucket* con la distancia mínima a la consulta `q` y se procede buscando en cada rama que alcance el rango de tolerancia, guardando en cada nodo que se visite el elemento más cercano a `q`.

**Algoritmo 2** Algoritmo de búsqueda por rango.

BuscaNodoPorRango$(q, r)$

1 $elemmin \leftarrow dmin\{[bucket[0], bucket[1], \ldots, bucket[b-1]], q\}$
2 **para** $j \leftarrow 0$ **hasta** $NumHijos$
3 {
4 **si** $A_j \leq d(p_i, q) - r < A_{j+1}$ ||
5 $A_j \leq d(p_i, q) < A_{j+1}$ ||
6 $A_j \leq d(p_i, q) + r < A_{j+1}$
7 **entonces** $hijos[i].RegresaPalabraConDistMin(q, elemmin, r)$
8 }
9 $Mensaje(\text{"}La\ palabra\ prununciada\ fue:\ \text{"} + Palabra(elemmin))$

Las lineas 2 de los Algoritmos 2 y 3 conservan el elemento con la distancia mínima a `q`. Estos dos algoritmos realizan la misma tarea únicamente el método llamado en el Algoritmo 2 es un auxiliar. Las lineas 3-9 verifican que la j-ésima arista cumpla con $A_j \leq d(p_i, q) - r < A_{j+1} \vee A_j \leq d(p_i, q) < A_{j+1} \vee A_j \leq d(p_i, q) + r < A_{j+1}$ para poder avanzar por ella al siguiente nodo haciendo la llamada recursiva, el método regresa cuando no hay aristas que cumplan con la desigualdad. En la linea 10 del Algoritmo 2 se muestra la etiqueta del elemento regresado.

**Algoritmo 3** Algoritmo para el auxiliar de la búsqueda por rango.

```
RegresaPalabraConDistMin(q, elemmin, r)
1  elemmin ← dmin{[elemmin, bucket[0], bucket[1], ..., bucket[b − 1]], q}
2  para j ← 0 hasta NumHijos
3    {
4    si A_j ≤ d(p_i, q) − r < A_{j+1}  ||
5       A_j ≤ d(p_i, q) < A_{j+1}  ||
6       A_j ≤ d(p_i, q) + r < A_{j+1}
7       entonces hijos[i].RegresaPalabraConDistMin(q, elemmin, r)
8    }
```

### 4.2.2. FQT

La implementación de este árbol requiere un diseño más elaborado debido al hecho de que cada nodo pude ser tanto nodo hoja como un nodo interno, para simular este comportamiento se hizo uso de una clase abstracta y así poder instanciar el nodo según sea conveniente, el diagrama de clases se muestra en la Figura 4.4.

#### Inserción

El procedimiento `ColocaHoja` es uno de dos procesos recursivos que se llevan a cabo en la inserción. En el Algoritmo 4 se insertan uno por uno los elementos del *bucket* donde hubo colisión más el nuevo elemento, si falla esto se crea una nueva instancia de nodo interno y se llama recursivamente la función con esta lista de elementos, el procedimiento termina hasta que se inserten todos.

El procedimiento `RecorreHoja` funciona como auxiliar (realiza esencialmente lo mismo que `Inserta`) y es parte de la recursión indirecta que se realiza en el FQT, es el encargado de recorrer el árbol avanzando por las aristas que cumplan la desigualdad $A_j \leq d(p_i, u_n) < A_{j+1}$, línea 4 del Algoritmo 5.

**Algoritmo 4** Pseudocódigo: Método que corrige la inserción si el *bucket* está lleno.

```
ColocaHoja(bucket, u_n)
 1  para u_n, u en bucket
 2   {
 3   para j ← 0 hasta NumHijos
 4     {
 5     si A_j ≤ d(p_i, u) < A_{j+1}
 6        entonces si hijos[j] == NULO
 7                    entonces hijos[j] ← nuevo nodoHoja
 8                              hijos[j].InsertaHoja(u)
 9                 si hijos[j] == nodoHoja
10                    entonces si !InsertaHoja(u)
11                                 entonces aux ← nuevo nodoInterno
12                                           bucket_n ← hijos[j].ObtenerBucket()
13                                           aux.ColocaHoja(bucket_n, u)
14                                           hijos[j] ← aux
15     }
16   }
```

**Algoritmo 5** Pseudocódigo: Método que recorre el árbol e inserta.

```
RecorreArbol(u_n)
 1  para j ← 0 hasta NumHijos
 2    {
 3    si A_j ≤ d(p_i, u_n) < A_{j+1}
 4       entonces si hijos[j] == NULO
 5                    entonces hijos[j] ← nuevo nodoHoja
 6                              hijos[j].InsertaHoja(u_n)
 7                 si hijos[j] == nodoHoja
 8                    entonces si !InsertaHoja(u_n)
 9                                 entonces aux ← nuevo nodoInterno
10                                           bucket_n ← hijos[j].ObtenerBucket()
11                                           aux.ColocaHoja(bucket_n, u_n)
12                                           hijos[j] ← aux
13                 si hijos[j] == nodoInterno
14                    entonces aux ← nuevo nodoInterno
15                              aux.RecorreArbol(u_n)
16                              hijos[j] ← aux
17    }
```

**Algoritmo 6** Pseudocódigo del método Inserta.

```
INSERTA(h)
 1  si raiz == NULO
 2     entonces raiz ← nuevo nodoHoja
 3              raiz.InsertaHoja(h)
 4     sino     si raiz == nodoHoja
 5                 entonces si !raiz.InsertaHoja(h)
 6                             entonces aux ← nuevo nodoInterno
 7                                      aux.ColocaHoja(bucket, h)
 8                                      raiz ← aux
 9     sino     si raiz == nodoInterno
10                 entonces aux ← nuevo nodoInterno
11                          aux.RecorreArbol(h)
12                          raiz ← aux
```

El procedimiento `Inserta` (Algoritmo 6) realiza las siguientes tareas. En las lineas 1-3 hace de la raíz un nodo hoja si es que el árbol esta vacío y procede a insertar ahí el nuevo elemento, en otro caso en las lineas 4-8 si actualmente la raíz es un nodo hoja se intenta insertar el elemento en el nodo, si falla se llama al método `ColocaHoja` para insertar correctamente tanto los elementos del *bucket* como el nuevo. Las lineas 9-12 si es nodo interno se llama a `RecorreArbol` para recorrer el árbol. En la Figura 4.5 se ilustra los casos que se pueden dar al insertar elementos en el FQT.

**Búsqueda**

La métodos de búsqueda tanto del BKT como del FQT son muy parecidos debido a que siguen el mismo principio, con la diferencia de que en el FQT para una consulta inmediatamente se calcula el vector de distancias hacia los pivotes y después se procede con el recorrido por las aristas para las que se cumple que su rango de distancias está dentro del rango de tolerancia. Al final este método regresa el elemento que tiene la distancia menor hacia la consulta. El pseudocódigo del método `Busca` (Algoritmo 7) en las lineas 3-5 realiza una búsqueda secuencial en el *bucket* de la raíz si es nodo hoja y muestra la etiqueta del elemento regresado. Las lineas 6-9 en caso de que sea un nodo interno hacen la llamada recursiva del método auxiliar `BuscaHojaPorRango` (Algoritmo 8) que se encarga de descender por las aristas que cumplan la desigualdad planteada en las lineas 4-6, si el hijo es hoja se realiza la búsqueda del elemento con menor distancia a la consulta, de otro modo si es un nodo interno se repiten los dos pasos anteriores.

### 4.2.3. FHFQT

Este indice hereda su estructura del FQT (Figura 4.4) y casi toda su funcionalidad pero no completamente, este primero requiere un método especifico para llevar a cabo la inserción de un nuevo elemento. El método de búsqueda es heredado del FQT.

**Inserción**

Como es sabido el FHFQT tiene una altura fija por lo tanto el procedimiento `Inserta` al momento de insertar verifica que tipo de instancia es la raíz y que altura se

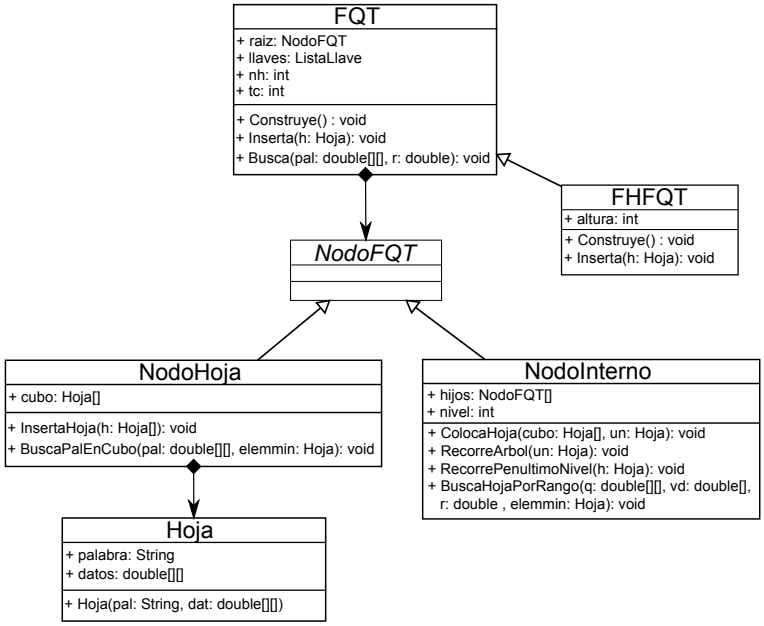
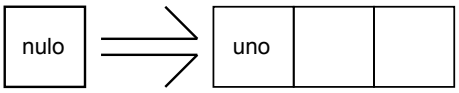



Figura 4.4: Diagrama UML de las clases FQT y FHFQT.

**Algoritmo 7** Pseudocódigo: Método Busca.

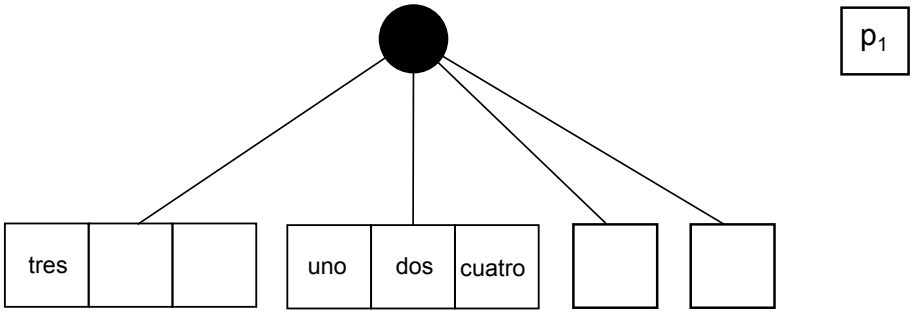
BUSCA$(pal, r)$

1 **si** $raiz ==$ NULO
2 **entonces** $Mensaje(\text{"}La\ palabra\ prununciada\ no\ esta\ en\ la\ BD.\text{"})$
3 **si** $raiz == nodoHoja$
4 **entonces** $raiz.BuscaPalEnBucket(pal, elemmin)$
5 $Mensaje(\text{"}La\ palabra\ prununciada\ fue:\ \text{"} + Palabra(elemmin))$
6 **si** $raiz == nodoInterno$
7 **entonces** $ObtenerVectorDistancias(vd, pal)$
8 $raiz.BuscaHojaPorRango(pal, vd, r, elemmin)$
9 $Mensaje(\text{"}La\ palabra\ prununciada\ fue:\ \text{"} + Palabra(elemmin))$

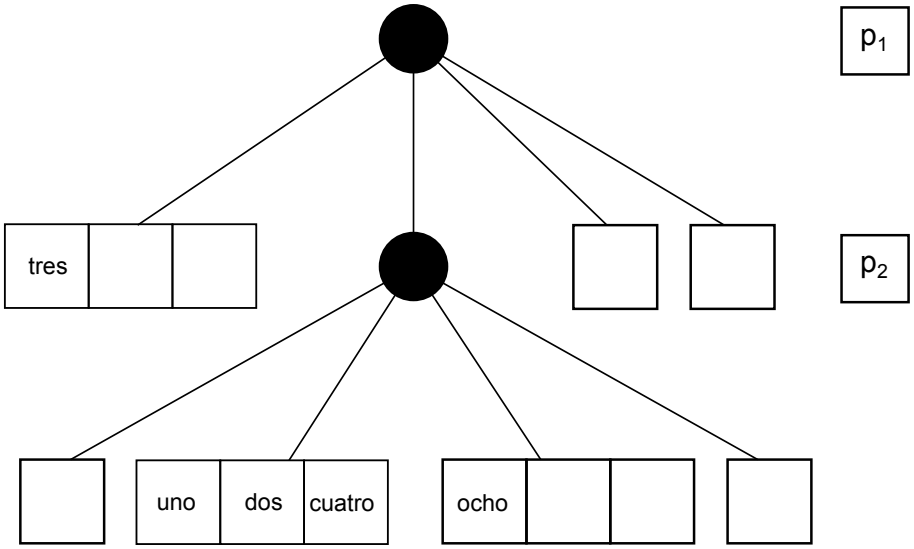



**(a)** ***uno insertado***


**(b)** ***dos, tres insertados***




**(c)** ***cuatro insertado***




**(d)** ***ocho insertado***

Figura 4.5: Insertar elementos en el FQT. (a) Se instancia y se inserta uno. (b) Se insertan dos y tres consecutivamente. (c) El *bucket* esta lleno por tanto se instancia un nodo interno, se elige aleatoriamente un elemento de la BD como $p_1$ y se verifica tantas veces sea necesario que los elementos se inserten sin problemas. (d) Se inserta ocho y como hay *bucket* lleno se repite el proceso anterior.

**Algoritmo 8** Pseudocódigo: Método que busca nodos hoja y regresa el menor.

```
BuscaHojaPorRango(q, vd, r, elemmin)
 1  para j ← 0 hasta NumHijos
 2   {
 3   si A_j ≤ d(p_i, q) − r < A_{j+1}  ||
 4      A_j ≤ d(p_i, q) < A_{j+1}  ||
 5      A_j ≤ d(p_i, q) + r < A_{j+1}
 6      entonces si hijos[j] == nodoHoja
 7                  entonces hijos[j].BuscaPalEnBucket(q, elemmin)
 8               si hijos[j] == nodoInterno
 9                  entonces hijos[j].BuscaHojaPorRango(q, vd, r, elemmin)
10   }
```

definió en la creación, si no hay elementos y la altura es igual a 0 entonces se instancia el primer nodo como hoja y se lleva a cabo la inserción, de otro modo si la altura es mayor a 0 se llama a `RecorrePenultimoNivel`. Si es un nodo hoja, en ese nodo se almacena el nuevo elemento (lineas 8 y 9), de otro modo si el nodo raíz es interno se llama a un método auxiliar que tiene como tarea recorrer el árbol hasta colocarse en el penúltimo nivel lineas 10-13, una vez ahí realiza la inserción, el pseudocódigo de estos métodos son los Algoritmos 9 y 10. `RecorrePenultimoNivel` avanza por las aristas dentro del rango de tolerancia, si el nivel del nodo actual es igual a la altura del árbol entonces se inserta el nuevo elemento en el $j$-ésimo hijo lineas 5-10 (en caso de que no esté instanciado se crea el nodo hoja); si no se ha alcanzado el penúltimo nivel se avanza por las ramas haciendo la llamada recursiva lineas 11-17.

**Algoritmo 9** Pseudocódigo: Método que inserta un nuevo elemento al FHFQT.

```
INSERTA(h)
 1  si raiz == NULO
 2     entonces si altura == 0
 3                 entonces raiz ← nuevo nodoHoja
 4                           raiz.InsertaHoja(h)
 5           si altura > 0
 6                 entonces raiz ← nuevo nodoInterno
 7                           raiz.RecorrePenultimoNivel(h)
 8     sino     si raiz == nodoHoja
 9                 entonces raiz.InsertaHoja(h)
10     sino     si raiz == nodoInterno
11                 entonces aux ← nuevo nodoInterno
12                           aux.RecorrePenultimoNivel(h)
13                           raiz ← aux
```

**Algoritmo 10** Pseudocódigo: Método que recorre el árbol y se se posiciona en el penúltimo nivel para hacer la inserción.

```
RecorrePenultimoNivel(h)
 1  j
 2  para j ← 0 hasta NumHijos
 3    {
 4    si A_j ≤ d(p_i, x) < A_{j+1}
 5       entonces si i == altura
 6                      entonces si hijos[j] == NULO
 7                                  entonces hijos[j] ← nuevo nodoHoja
 8                                            hijos[j].InsertaHoja(h)
 9                                  sino       si hijos[j] == nodoHoja
10                                                entonces hijos[j].InsertaHoja(h)
11                      sino       si hijos[j] == NULO
12                                  entonces aux ← nuevo nodoInterno
13                                            aux.RecorrePenultimoNivel(h)
14                                            hijos[j] ← aux
15                                  si hijos[j] == nodoInterno
16                                  entonces aux.RecorrePenultimoNivel(h)
17                                            hijos[j] ← aux
18    }
```

### 4.2.4. FQA

Dado que este índice es un arreglo y cualquier lenguaje de programación cuenta con esta estructura, su implementación no conlleva una elaboración de una estructura y procedimientos complejos, por ejemplo para insertar un nuevo elemento basta con encapsular los datos y realizar la asignación entre el $k$-ésimo elemento de la BD y la $k$-ésima posición del arreglo. Otra ventaja es que como el tamaño del arreglo es de exactamente el número de elementos que tenemos en nuestra BD entonces no hay desperdicio de memoria. En la Figura 4.6 se muestra el diagrama de clases del FQA.

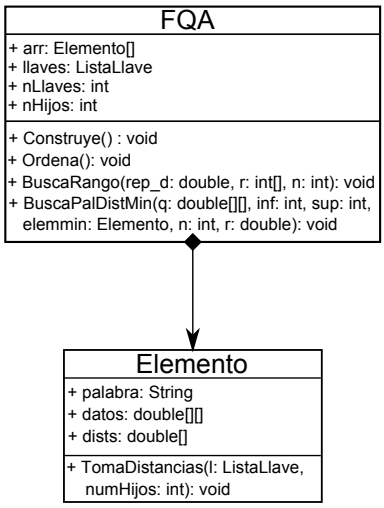



Figura 4.6: Diagrama UML de la clase FQA.

#### Búsqueda

El procedimiento `Busca` únicamente encapsula a la consulta `q`, crea el vector de distancias hacia pivotes y manda llamar a `BuscaPalDistMin` que es el procedimiento encargado de buscar al elemento con la distancia menor a `q`.

El procedimiento `BuscaPalDistMin` (Algoritmo 11) en la linea 1 verifica si se está en el ultimo nivel, si es el caso se busca en el rango que dictan los apuntadores `inf` y `sup`, el elemento con la distancia menor a la consulta $q$ linea 2, de otro modo en cada nivel se busca el rango de elementos que tienen etiquetada una distancia $d$, donde $d$ toma valores

**Algoritmo 11** Pseudocódigo del auxiliar de la búsqueda por rango en el FQA, regresa *elemmin*.

```
BuscaPalDistMin(q, inf, sup, elemmin, r)
 1  si i == NumLlaves
 2     entonces elemmin ← dmin{[I_inf, I_inf+1, I_inf+2, ..., I_sup], q}
 3     sino     para j ← 0 hasta NumHijos
 4                 {
 5                 si A_j ≤ d(p_i, q) − r < A_j+1  ||
 6                    A_j ≤ d(p_i, q) < A_j+1  ||
 7                    A_j ≤ d(p_i, q) + r < A_j+1
 8                    entonces inf_n ← inf
 9                             sup_n ← sup
10                             BuscaRango(j * 1/nh, inf_n, sup_n) //Búsqueda binaria
11                             BuscaPalDistMin(q, inf_n, sup_n, elemmin, r)
12                 }
```

desde $d(p_i, q) - r$ hasta $d(p_i, q) + r$, y se manda llamar el procedimiento recursivamente con los valores de los apuntadores actualizados lineas 8-11, es así como se simula el recorrido por un árbol en el FQA.

### 4.2.5. Permutaciones

Este indice es implementado en un arreglo donde residen los elementos de la base de datos encapsulados y tiene como miembro una lista ligada donde se insertan los permutantes. Esta encapsulación agrega un arreglo para almacenar la permutación que es ordenada en orden creciente con respecto al valor de distancia. Cada elemento de la permutación también está encapsulado y almacena tanto un valor entero para identificar a cada permutante y un *double* que es la distancia de la palabra a ese permutante. La Figura 4.7 muestra el diagrama de clases del indice.

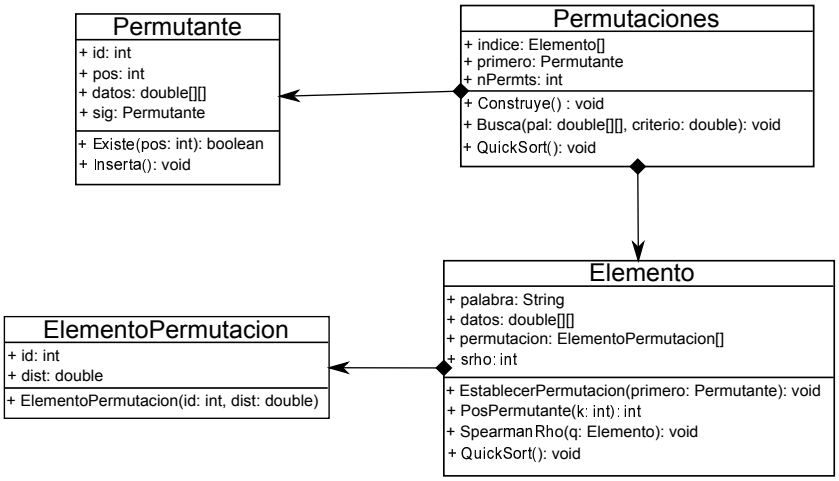



Figura 4.7: Diagrama UML de las clases del indice basado en permutantes.

El procedimiento `Busca` se realiza exactamente como se indica en la sección 3.2. La consulta es encapsulada y se inicializa su permutación para ser ordenada. Se calcula $S_\rho(u, q)$ para todo $u$ en la BD. Los elementos en el arreglo del indice son ordenados con respecto al valor de $S_\rho$ usando `QuickSort` [Cormen09]. Por ultimo se establece que porcentaje de la BD se va a revisar y se regresa el elemento con la distancia más pequeña a $q$.

## 4.3. Conclusiones del Capítulo

Se presento la implementación de los índices y los algoritmos de búsqueda, basándonos en la estructura definida en el Capítulo anterior para verificar si el índice presenta una mejor opción ante la búsqueda secuencial y poder realizar una comparación de desempeño entre los índices dentro del reconocimiento de palabras. En general la construcción de un índice basado en un árbol es compleja en comparación con los que utilizan un arreglo.

Capítulo 5

# Experimentos y Resultados

En esta sección se muestran los resultados experimentales en relación al tiempo de respuesta y precisión, además de una breve descripción del ambiente donde se ejecutó el sistema de reconocimiento. Por otra parte se muestran los resultados de la validación de la desigualdad del triangulo en el DTW.

## 5.1. Ambiente de trabajo

El sistema de reconocimiento fue ejecutado en una computadora con una memoria principal de 3.0 GB, así como un procesador con dos núcleos, soporte para cuatro subprocesos, conjunto de instrucciones de 64 bits a 2.13 GHz. Se usó una maquina virtual de Java* para arquitectura de 64 bits. Mientras se ejecutaba el sistema a la par estaba en ejecución el Netbeans IDE†.

En los experimentos se midió el tiempo que tarda en reconocer el método que realiza la búsqueda ya sea secuencial o indexada. La BD consta de 2000 palabras y su formato consta de una etiqueta con la palabra escrita textualmente seguida del número de marcos $nm$ que resultaron del proceso de segmentación de la señal de voz y por ultimo $nm$ renglones donde el primer valor es el $n$-ésimo marco y a continuación $k$ valores flotantes de doble precisión. Estos valores forman la matriz de características de la palabra pronunciada. A continuación se muestra la plantilla del formato:

* http://www.java.com/ † http://netbeans.org/

$\langle etiqueta \rangle \langle nm \rangle$

$\langle marco_1 \ \rangle, \langle valor\ doble \rangle, \langle valor\ doble \rangle, \cdots, \langle valor\ doble \rangle$

$\langle marco_2 \ \rangle, \langle valor\ doble \rangle, \langle valor\ doble \rangle, \cdots, \langle valor\ doble \rangle$

$\vdots \qquad \vdots \qquad \vdots \qquad \cdots \qquad \vdots$

$\langle marco_{nm} \rangle, \langle valor\ doble \rangle, \langle valor\ doble \rangle, \cdots, \langle valor\ doble \rangle$

Las características con las que se instanciaron los índices para hacer las pruebas son las siguientes:

- Los árboles fueron instanciados con 4 hijos y un cubo de 3 cada nodo.
- El FQA simula ser un árbol con 4 descendientes.
- Los FHFQT, FQA tienen una altura de 50.
- El algoritmo basado en permutaciones usa 50 permutantes.

Estos parámetros fueron elegidos para que todos los índices fueran probados en circunstancias similares.

## 5.2. Validaciones de la desigualdad triangular

La validación de la desigualdad del triangulo se realizo seleccionando aleatoriamente tres objetos de la BD, digamos elocuciones A,B y C, después se evalúan $d(A, B)$, $d(A, C)$ y $d(B, C)$ para verificar si se cumple que $d(A, C) <= d(A, B) + d(B, C)$. Este experimento se probó con $7,200$ tercias. El resultado obtenido fue que la desigualdad triangular se cumplió el $99.9\,\%$ de las veces y solo el $0.1\,\%$ no se cumplió.

## 5.3. Comparación del tiempo de respuesta de los índices y la búsqueda secuencial

En la Figura 5.1 se muestra la comparación entre la búsqueda indexada y la secuencial. De la gráfica se observa claramente como los índices se sitúan sobre una región

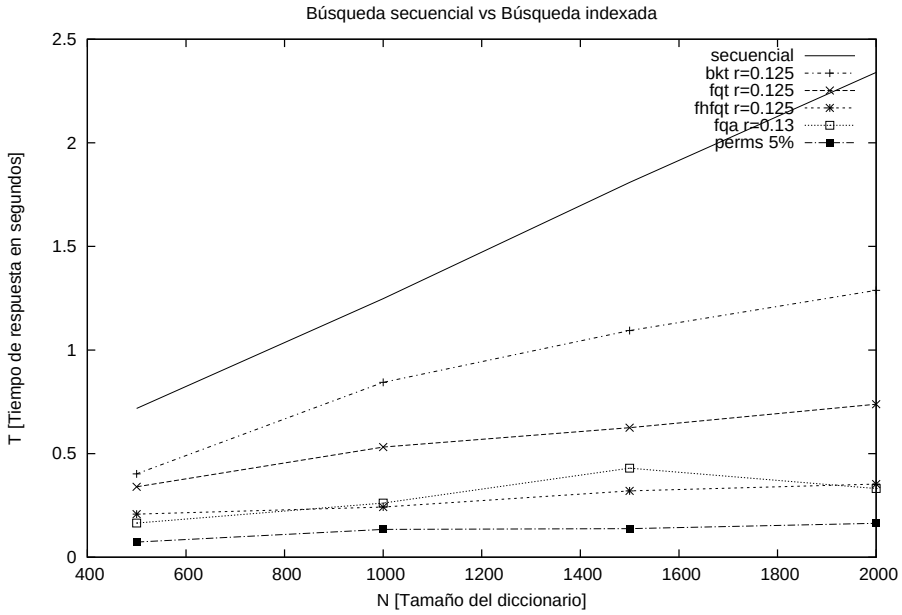



Figura 5.1: Comparación del tiempo de respuesta usando búsqueda secuencial contra búsqueda indexada.

por debajo de donde se coloca la curva de la búsqueda secuencial por tanto cualquier índice representa una mejor opción ante el procedimiento de fuerza bruta.

## 5.4. Comparación del tiempo de respuesta de los índices

Después de obtener resultados favorables al hacer uso de índices, en la Figura 5.2 se presentan los resultados de hacer los experimentos de medición de tiempo contra tamaño de diccionario, para cada método de búsqueda implementado en cada índice de proximidad. Se usó un rango de interés de 0.125 para agilizar la búsqueda y no perder precisión, contando con que cada nodo tiene cuatro descendientes. Para el índice basado en permutantes se estableció realizar comparaciones directas con el 5 % de la BD porque para ese porcentaje el índice es capaz de reconocer la palabra buscada.

El resultado arroja el siguiente orden en en cuanto a tiempo de respuesta. Los experimentos muestran que en la mayoría de los casos el índice basado en permutantes debido a que solo ocupa buscar en al menos 5 % de la BD para encontrar la palabra es el más rápido. El que le sigue es el FQA con tiempos muy aproximados. El FHFQT es el

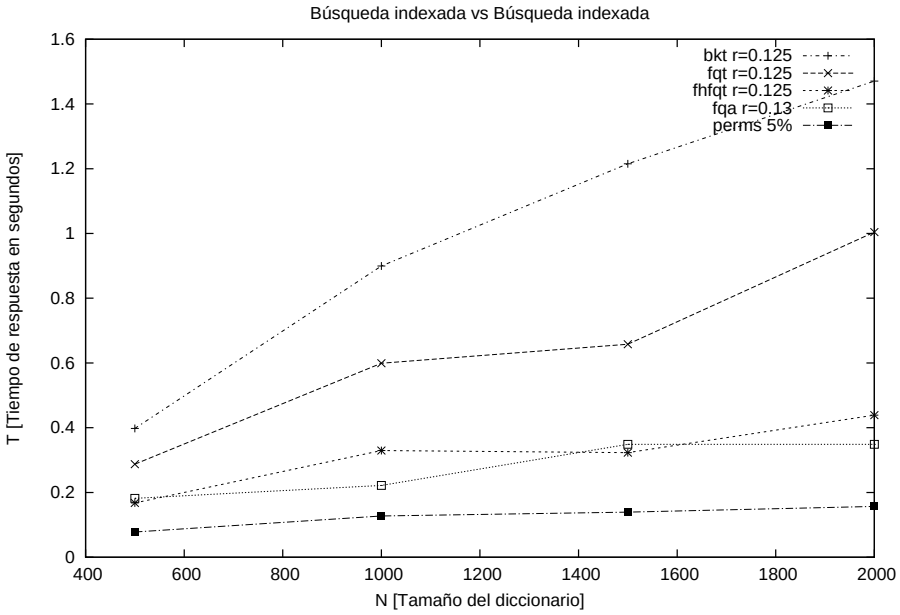



Figura 5.2: Comparación del tiempo de respuesta de cada índice.

siguiente más rápido, respondiendo muy parecido al FQA pero con un retraso muy notorio. El FQT tiene un tiempo de respuesta mejor que el BKT y en ambos índices cuando el diccionario es más grande sus tiempos se incrementan notablemente.

## 5.5. Comparación de la precisión de los índices

La precisión es un factor indispensable en el caso de la búsqueda por rango usada en los índices basados en pivotes este factor está relacionado con el valor que demos a $r$, si reducimos demasiado el radio de interés se pierde precisión. En el índice basado en permutantes con tan solo realizar una comparación directa con el 10 % o 5 % de la BD podemos obtener la palabra buscada, pero esto implica no tener la certeza de encontrar la palabra en cada intento.

Para medir la precisión se dicto una secuencia de 48 palabras al sistema para diccionarios de 500, 1000, 1500 y 2000 palabras y la Figura 5.3 muestra los resultados obtenidos. Como se pude apreciar los índices basados en pivotes tienen aproximadamente la misma precisión. El BKT es el más preciso y el que lo igualo en dos intancias de las cuatro

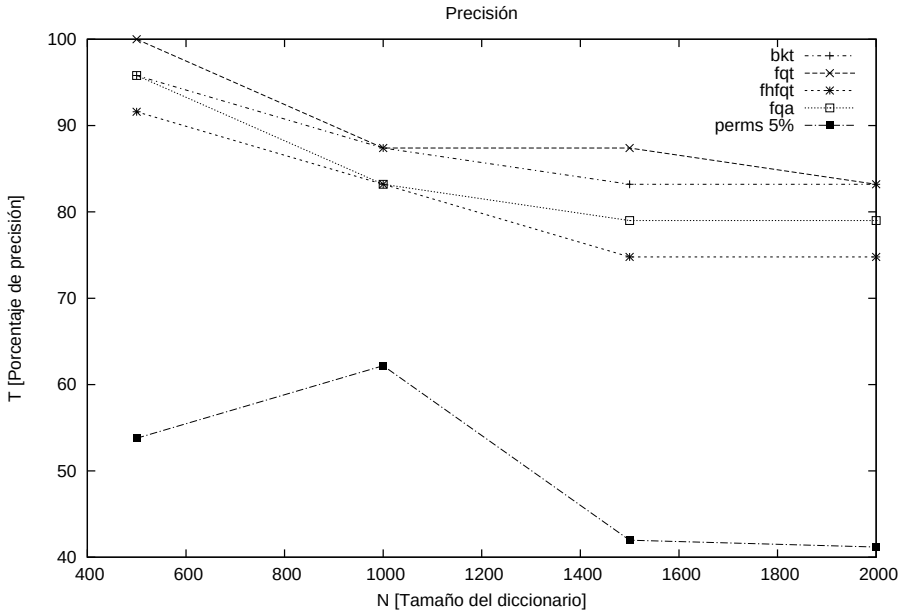



Figura 5.3: Comparación de la precisión de cada índice.

comparadas fue el FQA. El FQT y el FHFQT mostraron la misma precisión en la mayoría de las instancias. El índice basado en permutantes debido a que su algoritmo de búsqueda no hace uso de la desigualdad del triangulo no asegura reconocer la palabra buscada. Por lo anterior este índice tiene menos precisión. Una observación a resaltar es que todos los índices son más precisos conforme el tamaño del diccionario es más grande. Este fenómeno es mas notorio en los basados en pivotes.

## 5.6. Conclusiones del Capítulo

El DTW a pesar de no cumplir estrictamente la desigualdad triangular, tiene una fuerte relación con esta propiedad ya que en la mayoría de los casos la cumple. Esta característica del DTW lo hace muy útil en búsquedas por por proximidad que hacen un fuerte uso de la desigualdad del triangulo para quitar elementos del espacio de búsqueda. Los índices son una mejor opción en el reconocimiento de palabras porque tienen tiempos de respuesta más rápidos que la búsqueda secuencial. Los índices como el BKT y el FQT encuentran la palabra buscada en un tiempo mayor que el FHFQT, FQA y el basado en

permutantes pero tienen una mejor precisión. El índice basado en permutantes tiene tiempos de respuesta mejores que los basados en pivotes, pero tiene una precisión más baja.

Capítulo 6

# Conclusiones y Trabajos Futuros

## 6.1. Conclusiones Generales

En este trabajo se presentaron propuestas de la implementación de los índices estudiados en el Capítulo 3 y su adaptación a un sistema de reconocimiento de palabras aisladas para realizar comparaciones entre la búsqueda secuencial y las búsquedas por proximidad. Una vez que se optó como mejor alternativa el uso de índices se realizaron pruebas entre sus métodos de búsqueda así como experimentos de precisión para cada índice.

- La característica más deseada en un índice es que sea capaz de encontrar el elemento buscado realizando el menor número de cómputos de distancias debido a lo costosas que resultan. El índice basado en permutantes permite exactamente esta posibilidad ya que puede encontrar la palabra comparando directamente tan solo con el 5 % de la BD a pesar de no contar con la precisión de los índices basados en pivotes.

- Si la precisión se volviera un factor importante en nuestro sistema de reconocimiento, el FQA es el índice con tiempos de respuesta más rápidos que sus similares y cuenta con mucha precisión.

- Para una base de datos de 2000 palabras el FHFQT identifica la palabra buscada en tiempos muy próximos (en varios casos) a los que se obtienen usando el FQA.

- El FQT y el BKT son muy precisos y para bases de datos que no son tan grandes

como la nuestra son una buena alternativa porque tienen tiempos de respuesta de menos del segundo.

## 6.2. Trabajos Futuros

1. Trabajar en la investigación de índices de proximidad y algoritmos de búsquedas por proximidad en espacios métricos que reduzcan el tiempo de consulta a una base de datos multimedia.

2. Continuar con el estudio del reconocimiento de voz para poder implementar sistemas más eficientes de reconocimiento de palabras aisladas, sistemas de reconocimiento continuo de voz y sistemas de reconocimiento de música.

3. Probar índices basados en Hash. Locality Sensitive Hashing (LSH).

4. Dar seguimiento a la captura de palabras para aumentar el número de elementos de la BD.

# Referencias


[Baeza-Yates94] Baeza-Yates, R. A., Cunto, W., Manber, U., y Wu, S. Proximity matching using fixed-queries trees. *En* M. Crochemore y D. Gusfield, eds., *CPM*, tomo 807 de *Lecture Notes in Computer Science*, págs. 198–212. Springer, 1994. ISBN 3-540-58094-8.

[Baeza-Yates97] Baeza-Yates, R. *Searching: An Algorithmic Tour.*, tomo 37, págs. 331–359. Marcel Dekker, Inc., 1997.

[Bahl92] Bahl, L., Gopalakrishnan, P., Kanevsky, D., y Nahamoo, D. A fast admissible method for identifying a short list of candidate words. *Computer Speech & Language*, 6(3):215 – 224, 1992. ISSN 0885-2308. doi:10.1016/0885-2308(92)90018-Y.

[Burkhard73] Burkhard, W. A. y Keller, R. M. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, abr. 1973. ISSN 0001-0782. doi:10.1145/362003.362025.
URL `http://doi.acm.org/10.1145/362003.362025`

[Camarena11] Camarena, J. A. Notas de síntesis y reconocimiento de voz, 2011.

[Chávez01a] Chávez, E., Marroquín, J. L., y Navarro, G. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools Appl.*, 14(2):113–135, jun. 2001. ISSN 1380-7501. doi:10.1023/A:1011343115154.
URL `http://dx.doi.org/10.1023/A:1011343115154`

[Chávez01b] Chávez, E., Navarro, G., Baeza-Yates, R., y Marroquín, J. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.

[Chávez08] Chávez, E., Figueroa, K., y Navarro, G. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(9):1647–1658, 2008.

[Cormen09] Cormen, T. H., Leiserron, C. E., Rivest, R. L., y Stein, C. *Introduction to Algorithms*. MIT Press, 3ª ed$^{\underline{\text{ón}}}$., 2009. ISBN ISBN 978-0-262-03384-8 (hardcover : alk. paper)?ISBN 978-0-262-53305-8 (pbk. : alk. paper).

[Davis52] Davis, K. H., Biddulph, R., y Balashek, S. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, 1952. doi:10.1121/1.1906946.
URL `http://link.aip.org/link/?JAS/24/637/1`

[DRA13] Dragon naturallyspeaking. http://www.nuance.es/dragon/index.htm, 2013.

[Figueroa06] Figueroa, K., Chávez, E., y Navarro, G. Nuevos Índices para la búsqueda por proximidad en espacios métricos. *ENC 2006*, sep. 2006.

[JTA13] Java telephony api (jtapi). http://www.oracle.com/technetwork/java/jtapi-136088.html, 2013.

[Knuth98] Knuth, D. E. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998. ISBN 0-201-89685-0.

[Lawrence Rabiner93] Lawrence Rabiner, B.-H. J. *Fundamentals of speech recognition*. Prentice-Hall, 1ª ed$^{\underline{\text{ón}}}$., 1993. ISBN 0-13-285826-6.

[Lee11] Lee, C.-H., Soong, F. K., y Paliwal, K. K. *Automatic Speech and Speaker Recognition: Advanced Topics*. Springer Publishing Company, Incorporated, 1ª ed$^{\underline{\text{ón}}}$., 2011. ISBN 1461285909, 9781461285908.

[Rabiner78] Rabiner, L., Rosenberg, A., y Levinson, S. Considerations in dynamic time warping algorithms for discrete word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(6):575–582, 1978. ISSN 0096-3518. doi:10.1109/TASSP.1978.1163164.

# Glosario

**BD** *Base de Datos*

**BKT** *Burkhard-Keller Tree*

**FHFQT** *Fixed Height Fixed Queries Tree*

**FQA** *Fixed Queries Array*

**FQT** *Fixed Queries Tree*

**GB** *Giga-Bytes*

**GHz** *Giga-Hertz*

**HMM** *Hidden Markov Models (Modelos Ocultos de Markov)*

**LPC** *Linear Predictive Coding (Codificación Linear Predictiva)*

**LSH** *Locality Sensitive Hashing*

**MFCC** *Mel-Frequency Cepstral Coefficients (Coeficientes Cepstrales de Mel)*

# Índice

Índice basado en Permutantes, 17
Índices de proximidad, 11
Índices vs Búsqueda secuencial, 38
Índices vs Índices, 42

Búsquedas por proximidad, 8
BKT, 12

Consulta de rango, 8
Consulta del vecino más cercano, 8

Desigualdad del triangulo, 5

Espacio Métrico, 5
Experimentos y Resultados, 37

FHFQT, 15
FQA, 15
FQT, 13

Implementación BKT, 21
Implementación FHFQT, 32
Implementación FQA, 32
Implementación FQT, 25
Implementación permutaciones, 36

Norma Vectorial, 6

Precisión de los índices, 42

Sistema implementado, 19