



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DE ÍNDICE DINÁMICO PARA
COLECCIONES DE TEXTO Y APLICACIONES

PROPUESTA DE TESIS PARA OPTAR AL GRADO DE MAGÍSTER
EN CIENCIAS, MENCIÓN COMPUTACIÓN

FERNANDO KRELL LOY

PROFESOR GUÍA:
GONZALO NAVARRO BADINO

.....
Fernando Krell Loy
Alumno

.....
Gonzalo Navarro Badino
Prof. Guía

Este trabajo es financiado por el Instituto Milenio de
Dinámica Celular y Biotecnología (ICDB), P05-001-F, Mideplan, Chile

SANTIAGO DE CHILE
MAYO 2009

Índice

1. Introducción	1
2. Conceptos Básicos	3
2.1. Entropía de orden cero	3
2.2. Entropía de orden k	3
2.3. El problema de la secuencia dinámica	3
2.4. El problema de la colección dinámica de textos	4
2.5. Arreglos de sufijos	5
2.6. La transformada de Burrows-Wheeler	5
2.7. Wavelet trees	5
2.8. Codificación (c, o)	6
3. Objetivos	7
4. Metodología	8
5. Breve descripción de estructuras a implementar	9
5.1. Bitmaps dinámicos	9
5.2. Sumas parciales dinámicas	9
5.3. Bitmaps comprimidos dinámicos	9
5.4. Secuencias comprimidas dinámicas	9
5.5. Colecciones dinámicas de textos	10
Referencias	11

1. Introducción

Un problema común en computación es encontrar un patrón en un texto, siendo el texto considerablemente más largo que el patrón. Dado que hoy en día la cantidad de datos disponible como texto es enorme y crece con gran rapidez, este problema se vuelve muy interesante; cada vez existen más aplicaciones que necesitan encontrar información relevante en estos datos. La operación básica para obtener información es la búsqueda de patrones, por lo que encontrar una solución eficiente se hace extremadamente necesario.

Cuando el texto en donde buscar el patrón es grande una búsqueda secuencial es ineficiente e inútil. La solución a este problema es construir algún tipo de estructura de dato sobre el texto que funcione como índice y permita resolver las siguientes consultas:

- Encontrar el número de ocurrencias de un patrón en el texto.
- Localizar tales ocurrencias en el texto.

Los índices clásicos que resuelven estas consultas, aunque son rápidos, tienen la gran desventaja de ocupar bastante espacio (*Tries*, *Suffix Tries*, *Suffix Trees* y *Suffix Arrays*). Debido a esto en el último tiempo se ha hecho un gran esfuerzo en desarrollar índices que, aunque no sean tan rápidos, ocupan bastante menos espacio. Los mejores índices comprimidos existentes hoy en día son los llamados *auto-índices comprimidos*, debido a que ocupan espacio proporcional al texto, y además reemplazan el texto, es decir, cualquier trozo del texto puede ser obtenido directamente desde el índice sin utilizar el texto. Sin embargo, estos índices son válidos siempre y cuando el texto sea estático. Si el texto cambia el índice deja ser válido, por lo que se tendría que reconstruir el índice completamente, lo cual es extremadamente ineficiente. Esto complica su aplicabilidad en varios escenarios de interés.

En la última década se han estudiado índices que manejan colecciones de texto permitiendo insertar y eliminar textos [FM00, CHL04, CHLS07, MN08, GN08]. El más reciente de ellos (Mäkinen y Navarro 2008; González y Navarro 2008) es el que alcanza las mejores cotas para el espacio necesario y tiempo en responder consultas¹.

¹ Espacio: $nH_k + o(n \log \sigma)$ para cualquier $k \leq \alpha \log_{\sigma} n$ y $0 < \alpha < 1$.
Tiempo en contar patrones de largo m : $O(m \log n (1 + \frac{\log \sigma}{\log \log n}))$.
Cada ocurrencia puede ser encontrada en tiempo $O(\log^2 n (1 + \frac{\log \log n}{\log \sigma}))$.
Exponer un contexto de largo ℓ de un texto toma tiempo $O(\log^2 n (1 + \frac{\log \log n}{\log \sigma}) + \ell \log n (1 + \frac{\log \sigma}{\log \log n}))$.
Eliminar/insertar un texto toma $O(\log n (1 + \frac{\log \sigma}{\log \log n}))$ por símbolo.
En donde σ es el tamaño del alfabeto, H_k es la entropía de orden k y n es el largo total en caracteres de la colección.

Sin embargo, tales cotas son netamente teóricas, por lo que se hace interesante implementar el índice para obtener resultados experimentales y una estructura que permita construir aplicaciones prácticas. Es por esto que el trabajo de tesis de magíster consistirá principalmente en su implementación. Para implementar el índice se debe estudiar cómo llevar a la práctica las estructuras de datos teóricas propuestas para construir el índice, lo que puede causar que las cotas teóricas para espacio requerido y para los tiempos de las consultas encontradas no se cumplan. Por tal motivo, uno de los principales desafíos en el trabajo es lograr una mínima diferencia entre las cotas teóricas y los resultados efectivos.

Otro problema que limita bastante la aplicabilidad de los índices comprimidos estáticos es que necesitan construir un índice clásico no comprimido primero, lo que podría no caber en RAM. Un índice dinámico comprimido permite construir fácilmente un índice estático comprimido sin necesidad de construir la solución no comprimida. La forma de hacer esto es insertar en el índice dinámico todos los textos uno a uno.

En resumen, el trabajo consiste en implementar un índice comprimido para colecciones texto que permita insertar y eliminar secuencias a la colección con bajo costo y sin alterar en gran medida el índice, permitiendo al índice mantener su funcionalidad. Posteriormente el índice será utilizado para calcular la transformada de Burrows-Wheeler [BW94] y para construir índices estáticos usando poco espacio. Dependiendo del tiempo se estudiará también la posibilidad de usarlo para construir un arreglo de sufijos [MM93, GBYS92] en memoria secundaria.

El trabajo tiene bastante interés en el área de la biología computacional. Su desarrollo permitirá mantener colecciones dinámicas comprimidas de secuencias de nucleótidos o de aminoácidos y manejarlas en memoria principal para identificar patrones, realizar búsquedas de similitud, etc. El índice a desarrollar permitirá analizar secuencias moleculares en forma más eficiente y con mayor cantidad de datos.

2. Conceptos Básicos

2.1. Entropía de orden cero

Sea S una secuencia de símbolos pertenecientes a un alfabeto $\Sigma = (c_1, \dots, c_\sigma)$. Sea n_{c_i} la cantidad de ocurrencias del símbolo c_i en S y $n = |S|$. La entropía de orden 0 de la secuencia S se define como:

$$H_0(S) = \sum_{i=1}^{\sigma} \frac{n_{c_i}}{n} \log \frac{n}{n_{c_i}}$$

2.2. Entropía de orden k

Sea S una secuencia de símbolos pertenecientes a un alfabeto $\Sigma = (c_1, \dots, c_\sigma)$. La entropía de orden k de la secuencia S se define como:

$$H_k(S) = \sum_{x \in \Sigma^k} \frac{n_x}{n} H_0(S|x)$$

En donde n_x es la cantidad de ocurrencias de la subsecuencia x en S y $S|x$ denota la concatenación de los símbolos que aparecen inmediatamente antes de las n_x ocurrencias en S .

Nota: Es posible demostrar que $0 \leq H_k \leq H_{k-1} \leq \dots \leq H_1 \leq H_0 \leq \log \sigma$. H_k es la cota inferior al número de bits necesitados para comprimir S usando un compresor que codifica cada carácter considerando solamente el contexto de k caracteres que lo suceden en S .

2.3. El problema de la secuencia dinámica

Dado un alfabeto Σ , el problema consiste en mantener una secuencia $A = a_1 \dots a_n, a_i \in \Sigma$ que permita las siguientes operaciones:

- $read(A, i)$ retorna el símbolo a_i .
- $rank_c(A, i)$ retorna el número de ocurrencias del símbolo c en $A[1, i]$.
- $select_c(A, i)$ retorna el índice de la i -ésima ocurrencia del símbolo c .
- $insert(A, c, i)$ agrega el símbolo $c \in \Sigma$ entre a_{i-1} y a_i .
- $delete(A, i)$ elimina a_i de la secuencia.

En el caso especial en donde $\Sigma = \{0, 1\}$ este problema también se denomina “el problema del vector dinámico de bits”.

2.4. El problema de la colección dinámica de textos

El problema consiste en mantener una colección dinámica \mathcal{C} de textos $\{T_1, T_2, \dots, T_m\}$ en donde cada $T_i \in \Sigma^+$, permitiendo las siguientes operaciones:

- $count(\mathcal{C}, P)$ retorna el número de veces que el patrón P ocurre como una subsecuencia en la colección.
- $locate(\mathcal{C}, P)$ retorna las posiciones de las ocurrencias de P en la colección.
- $substring(\mathcal{C}, j, \ell, r)$ retorna $T_j[\ell, r]$.
- $j = insert(\mathcal{C}, T)$ inserta el texto T en la colección, retornando un valor j que permite identificar el texto en la colección ($T = T_j$).
- $delete(\mathcal{C}, j)$ elimina T_j de la colección.

2.5. Arreglos de sufijos

El arreglo de sufijos [MM93, GBYS92] de un texto $T_{1,n}$ es un arreglo $A[1, n]$ que contiene una permutación del intervalo $[1, n]$, tal que $T_{A[i],n} < T_{A[i+1],n}$ para todo $1 \leq i < n$, en donde “ $<$ ” entre secuencias corresponde al orden lexicográfico.

2.6. La transformada de Burrows-Wheeler

La transformada de Burrows-Wheeler [BW94] (ahora en adelante *BWT* por sus siglas en inglés) de un texto es una permutación de sus caracteres formada al recorrer el arreglo de sufijos asociado al texto y concatenando el carácter que precede a cada sufijo.

Formalmente, dado un texto $T[1, n]$ y su arreglo de sufijos $A[1, n]$, la *BWT* de T , $T^{bwt}[1, n]$, es definida como $T^{bwt}[i] = T[A[i] - 1]$, excepto cuando $A[i] = 1$, donde es $T^{bwt}[i] = T[n]$.

2.7. Wavelet trees

Un *wavelet tree* es una estructura de datos propuesta por Grossi, Gupta y Vitter [GGV03] que permite encontrar las ocurrencias de un carácter en una secuencia. La estructura se define como:

1. La raíz contiene un vector de bits en donde las posiciones en 1 marcan los caracteres en la secuencia que descienden a la derecha. Esos caracteres son concatenados para formar la secuencia que corresponde al hijo derecho de la raíz.
2. Los caracteres en las posiciones marcadas en 0 en el vector de bit de la raíz, forman la secuencia correspondiente al hijo izquierdo
3. Recursivamente se sigue el proceso hasta tener un sólo símbolo del alfabeto. Por lo tanto las hojas corresponden a símbolos en el alfabeto.

Si el tamaño del alfabeto es σ , la altura del *wavelet tree* es a lo más $\log \sigma$. Para contar la cantidad de ocurrencias de un símbolo en la secuencia, se requiere buscar en el árbol la hoja que contiene al símbolo, realizando *rank* en el vector de bits de cada nodo, como muestra un ejemplo en la figura 1.

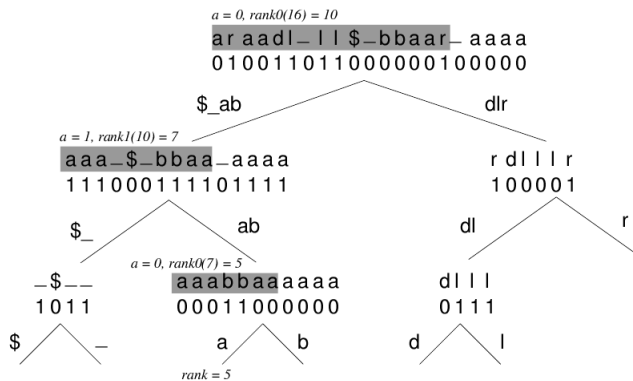


Figura 1: Un *wavelet tree* para la secuencia $S = \text{''araadl_ll\$_bbaar_aaaa''}$ que muestra la solución a la consulta de ocurrencias del carácter a hasta la 16va posición. Sólo los vectores de bits son almacenados.

2.8. Codificación (c, o)

Sea $t \in \mathbb{N}$. Se dice que una secuencia de bits de tamaño t pertenece a una *clase* c si tiene exactamente c bits en 1, para $0 \leq c \leq t$. Para cada clase c existe una tabla universal G_c de $\binom{t}{c}$ entradas que contiene las $\binom{t}{c}$ secuencias con c bits en 1. Una secuencia de t bits puede ser codificada con el par (c, o) , siendo c la clase de la secuencia y o el índice en la tabla G_c . Por lo tanto un bloque de tamaño t de clase c requiere solamente de $\log(t+1) + \log \binom{t}{c}$ bits. Notar que para decodificar un par (c, o) se deben tener precalculadas todas las tablas G_c , las cuales suman $O(\sqrt{n} \text{polylog}(n))$ bits.

Esta codificación se usa para representar un bitmap $B[1, n]$ en $nH_0(B) + o(n)$ bits [RRS07]. Cuando se aplica a los bitmaps de un *wavelet tree* de una secuencia $S[1, n]$, se obtienen $nH_0 + o(n \log \sigma)$ bits. Si S es la *BWT* de $T[1, n]$, el espacio es $nH_k(T) + o(n \log \sigma)$ bits, para $k \leq \alpha \log_\sigma n$, $0 \leq \alpha < 1$ [MN07].

3. Objetivos

El objetivo general de la tesis de magíster es implementar una estructura de datos que resuelva el problema de la colección dinámica de textos para luego ocuparlo en la construcción de la *BWT* y del índice estático en espacio comprimido.

Los objetivos específicos son:

- Implementar una estructura para resolver el problema del vector dinámico de bits.
- Implementar una estructura comprimida para resolver el problema del vector dinámico de bits.
- Implementar una estructura comprimida para resolver el problema de la secuencia dinámica de símbolos.
- Implementar un índice dinámico que resuelva el problema de la colección dinámica de textos.
- Construir la transformada de Burrows-Wheeler en espacio comprimido.
- Ocupar el índice para construir el índice estático en espacio comprimido.

4. Metodología

Para la implementación del índice es necesario implementar previamente algunas estructuras más básicas. La primera estructura a implementar es el bitmap dinámico en forma plana (no comprimida). Luego se extenderá esta estructura para manejarla en forma comprimida. Posteriormente se desarrollará una estructura para manejar secuencias comprimidas de símbolos que permita realizar operaciones de búsqueda, acceso, inserción y eliminación de símbolos con bajo costo en tiempo y en espacio. Por lo tanto, la tesis consistirá en desarrollar los siguientes puntos:

1. Implementación de estructura para secuencias de bits dinámicas.
2. Implementación de estructura comprimida para secuencias de bits dinámicas, mediante el uso de codificación (c, o) como representación física de los bloques de bits.
3. Implementación de estructura comprimida para secuencias dinámicas de símbolos, mediante un *wavelet tree* de bitmaps comprimidos dinámicos
4. Utilizar la secuencia dinámica de símbolos sobre la *BWT* de una colección de textos y estructuras auxiliares para implementar el índice comprimido dinámico para la colección.
5. Ocupar el índice dinámico para construir la transformada de Burrows-Wheeler en espacio comprimido.
6. Expandir la construcción anterior para obtener un índice estático en espacio comprimido.
7. Pruebas experimentales y aplicaciones.

Dependiendo el tiempo se explorará la construcción de arreglos de sufijos en memoria secundaria mediante el algoritmo de Gonnet et al. 1992 [GBYS92].

5. Breve descripción de estructuras a implementar

5.1. Bitmaps dinámicos

El bitmap dinámico es una estructura que entrega una solución al problema del vector dinámico de bits. Se basa en mantener un árbol balanceado en donde cada nodo contiene el número de bits mantenidos por su subárbol y la cantidad de bits en 1 en su subárbol.

La idea básica es dividir la secuencia de bits A en superbloques y bloques. Cada superbloque S mantiene $s = f(n) \log n$ bits ($f(n) = O(\log n)$) y corresponde a una hoja en el árbol. A la vez, cada superbloque está dividido en exactamente $2f(n)$ bloques que mantienen en forma plana (no comprimida) $t = (\log n)/2$ bits.

5.2. Sumas parciales dinámicas

La estructura que permite solucionar el problema de las sumas parciales buscables dinámica es básicamente la misma estructura que el bitmap dinámico, pero en donde cada bloque contiene $t = (\log n)/(2k)$ números. Los superbloques siguen manteniendo el mismo tamaño por lo que contienen $s = f(n)(\log n)/k$ números. Cada nodo contiene la cantidad de números de su subárbol y la suma de ellos.

5.3. Bitmaps comprimidos dinámicos

Un bitmap comprimido dinámico es esencialmente un bitmap dinámico en donde las hojas no contienen la secuencia de bits en forma plana, sino que se comprime usando algún tipo de codificación. Mäkinen y Navarro en 2008 proponen utilizar dos diferentes codificaciones dependiendo de la naturaleza de la secuencia. Si los bits en 1 son poco densos, entonces conviene utilizar *Gap Encoding*, en el caso de que los 1's sean densos ocupar *Block Identifier Encoding* (codificación (c, o)).

5.4. Secuencias comprimidas dinámicas

El caso del bitmap comprimido dinámico que utiliza *Block Identifier Encoding* (o codificación (c, o)) puede ser extendido para mantener secuencias de símbolos sobre un alfabeto Σ . La diferencia está en que los nodos contienen el número de ocurrencias de cada símbolo en su subárbol y la codificación utiliza en las hojas es la propuesta en [FMMN07] en donde se extiende la codificación para manejar secuencias no binarias.

5.5. Colecciones dinámicas de textos

Conceptualmente, se debe construir un arreglo de sufijos \mathbf{SA} para los textos que liste todos los sufijos de todos los textos en orden lexicográfico. $\mathbf{SA}[i] = (j, l)$ si el sufijo $T_j[l..]$ es el i -ésimo sufijo ordenado lexicográficamente. Para encontrar un patrón P se determina el intervalo $[x, y]$ tal que cada sufijo desde $\mathbf{SA}[x]$ hasta $\mathbf{SA}[y]$ tiene a P como prefijo. Por lo tanto $\mathbf{SA}[x], \mathbf{SA}[x + 1], \dots, \mathbf{SA}[y]$ da todas las posiciones de P en la colección.

Sin embargo, por restricciones de espacio no se puede almacenar \mathbf{SA} (pues ocupa $O(n \log n)$ bits). Por lo que, una mejor idea es utilizar un FM-index [FM00] para representar \mathbf{SA} implícitamente. Debido a que el FM-index requiere de una función *count*, se puede ocupar la estructura de la sección 5.4 sobre la transformada de Burrows-Wheeler [BW94] de los textos.

La estructura anterior permite contar las ocurrencias de un patrón, pero no permite obtenerlas. Para este propósito se requiere de una estructura adicional *MARK* que permite obtener $\mathbf{SA}[x]$ para cualquier x en forma eficiente (ver estructura en [CHL04]).

Referencias

- [BW94] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, 1994.
- [CHL04] Ho-Leung Chan, Wing-Kai Hon, and Tak-Wah Lam. Compressed index for a dynamic collection of texts. In *15th Proc. Combinatorial Pattern Matching, LNCS 3109*, pages 445–456, 2004.
- [CHLS07] Ho-Leung Chan, Wing-Kai Hon, Tak-Wah Lam, and Kunihiko Sadakane. Compressed indexes for dynamic text collections. *ACM Transactions on Algorithms*, 3(2):21, 2007.
- [FM00] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proc. of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 390–398. IEEE Computer Society, 2000.
- [FMMN07] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Transaction on Algorithms*, 3(2):20, 2007.
- [GBYS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text: Pat trees and Pat arrays. In *Information Retrieval: Data Structures and Algorithms*, pages 66–82. Prentice-Hall, Inc., 1992.
- [GGV03] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850. Society for Industrial and Applied Mathematics, 2003.
- [GN08] Rodrigo González and Gonzalo Navarro. Rank/select on dynamic compressed sequences and applications. *Theoretical Computer Science*, 2008. To appear.
- [MM93] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
- [MN07] Veli Mäkinen and Gonzalo Navarro. Implicit compression boosting with applications to self-indexing. In *SPIRE*, pages 229–241, 2007.
- [MN08] Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Transactions on Algorithms*, 4(3):1–38, 2008.

- [RRS07] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transaction on Algorithms*, 3(4):43, 2007.