UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA

T E S I S

para obtener el título de

Ingeniera Electrónica

de la Universidad de Buenos Aires

Presentada y defendida por
Melisa Coral HALSBAND

Métodos Eficientes para la Identificación de Patrones en Conjuntos de Señales Discretas

Directora de Tesis: Rosa WACHENCHAUZER
Co-directora de Tesis: Verónica BECHER

desarrollada en el Departamento de Computación de la Facultad de Ciencias Exactas y Naturales, Grupo KAPOW

defendida en diciembre de 2010

Agradecimientos

Índice general

Capítulo 1

Introducción

Índice

Esta tesis se interna en una de las intersecciones que existen entre la Electrónica y las Ciencias de la Computación. El procesamiento de señales discretas cuantizadas o digitales es para la computación el tratamiento de cadenas de caracteres, e involucra áreas como la teoría de la información, el análisis de Fourier, el reconocimiento de patrones, procesos estocásticos y complejidad computacional.

Además, es el producto de un trabajo fuertemente interdisciplinario entre estas ciencias duras y la -no siempre fluida- relación con la biología. La mayor parte de los datos a procesar y analizar provienen de avances técnicos recientes en la capacidad de analizar la información genética de los seres vivos, y el tratamiento de esta información requiere métodos adecuados a ella. Parte del trabajo realizado consistió en la traducción de necesidades abstractas formuladas desde el punto de vista de la genómica a definiciones cuyo cumplimiento pudiese ser comprobado y la correctitud de los métodos demostrada.

Una vez delineados los problemas, se proponen, describen y demuestran distintos métodos para el tratamiento de señales digitales o cadenas de símbolos atendiendo a los requerimientos que se derivan de las características de la entrada. Los criterios de diseño utilizados fueron formulados en base a distintos desafíos técnicos: a diferencia de la mayor parte de los enfoques de procesamiento de cadenas de ADN, basados en métodos heurísticos, en este trabajo se exige el hallazgo de resultados exhaustivos, sin comprometer la tratabilidad de los problemas. Por otro lado, se asumirá identidad de secuencias sólo en los casos en los que el ajuste es perfecto, sin contemplar índices de similitud o coincidencias inexactos.

Los algoritmos presentados en esta tesis forman parte del trabajo que el grupo KAPOW realiza en el marco del proyecto binacional MINCyT-MICINN AR009,"Dinámica de Homogeneización de genomas eucariotas", 2010-2011, dirigido por el Dr. Hernán Dopazo del Centro de Investigaciones Prícipe Felipe de Valencia, España, y la Dra. Verónica Becher.

Desde el punto de vista algorítmico el proyecto consiste en el diseño e implementación de una batería de algoritmos eficientes para el análisis comparativo de

secuencias genómicas intra e inter especies. El objetivo final de este proyecto es poner a prueba la hipótesis neutral de abundancia, y diversidad de elementos biológicos repetidos en el genoma eucariótico planteadas por [Venner 2009] y [Brookfield 2005].

1.1. Motivación

La aparición y accesibilidad de tecnologías de secuenciación de ADN de alta tasa de transferencia ha resultado en un enorme crecimiento de información genómica y metagenómica. A medida que más genomas son secuenciados, la eficiencia y escalabilidad de las herramientas para su análisis se vuelven necesarias. Aunque la capacidad de procesamiento también ha aumentado, y distintas formas de procesamiento de alto rendimiento se encuentran disponibles, los abordajes más directos adolecen de capacidad para tratar las variantes combinatorias que exigen los problemas.

Los gigantescos volúmenes de datos a ser analizados requieren tecnologías rápidas, dado que el costo de analizar los mayores conjuntos de datos actualmente exceden los costos de generarlos. Entre las funcionalidades necesarias para el análisis, una de las herramientas más costosas computacionalmente es la búsqueda de patrones o repeticiones dentro de las secuencias. A diferencia de la búsqueda o comparación de subsecuencias, que intentan alinear las secuencias local o globalmente, la búsqueda de patrones consiste en la identificación de subsecuencias que se encuentran repetidas más de una vez en el genoma, o en distintos genomas, y que cumplen ciertos criterios de maximalidad con respecto a las demás subsecuencias.

1.2. Objetivo

El objeto de la tesis consiste en desarrollar métodos para la identificación de repeticiones en genomas o metagenomas considerando *conjuntos* de secuencias. Es decir, la búsqueda de los fragmentos de ADN que se encuentren repetidos a lo largo de un conjunto de secuencias de nucleótidos, o aquellos que se encuentran repetidos en una o más secuencias pero son exclusivas de ellos, y no se encuentran en otro conjunto de referencia de secuencias. Técnicamente, entendiendo las secuencias de ADN como conjuntos de señales discretas cuantizadas de longitud arbitraria, los problemas consisten en identificar patrones dentro de las mismas atendiendo criterios de maximalidad que acoten el tamaño de la salida y atendiendo a requerimientos de tratabilidad en términos de tiempos de procesamiento y memoria consumida.

Se tratarán tres problemas distintos aunque relacionados:

- La búsqueda de *repeticiones supermaximales de un conjunto* de secuencias, subsecuencias de todos los elementos del conjunto que representan las máximas subsecuencias presentes en cada elemento del conjunto, es decir que tienen todos los elementos del conjunto en común.

- La búsqueda de *repeticiones maximales y supermaximales exclusivas de una secuencia respecto de un conjunto*, cadenas repetidas en una cadena fija, lo más

largas posibles, que no aparecen en ningún elemento del conjunto de referencia. Representan patrones de la cadena original que son propios o *exclusivos* de la misma.

- La búsqueda de *etiquetas minimales de un conjunto respecto de otro*, subcadenas lo más cortas posibles que se encuentren en todos los elementos del primer conjunto y en ninguno del segundo. Este problema toma características de los dos anteriores, aunque buscando cadenas minimales en lugar de maximales: de esta forma se hallan las subsecuencias más cortas que de alguna forma "identifican" a todos los elementos del primer conjunto, ya que una etiqueta minimal ocurre en un elemento si y sólo si éste pertenece al primer conjunto.

Estos tres problemas se tratarán por separado, pero utilizando estrategias análogas y compartiendo un juego de herramientas común en su resolución. Teniendo en cuenta las magnitudes de las entradas involucradas en las principales motivaciones de estos problemas, en todos los casos se tomará como principal restricción el uso de memoria, y en segundo lugar, la eficiencia en el consumo de recursos de procesador.

Además del análisis biológico sobre las repeticiones encontradas, el cálculo de indicadores estadísticos sobre los resultados obtenidos pueden interpretarse como índices de similitud o diferencias entre genomas o familias de especies y complejidad de la información contenida por las secuencias.

Mas allá de la motivación biotecnológica, dada la abstracción y generalidad del problema, existen otros campos del conocimiento en los que los métodos desarrollados pueden ser de utilidad. Actualmente no existen herramientas para el tratamiento de estos problemas en forma eficiente para grandes volúmenes de datos.

1.3. Antecedentes

En la comparación de Sahal et al. (2008) aparece una lista de los buscadores de repeticiones para genomas. Más allá de los métodos heurísticos y basados en bases de datos como RepeatMasker (2009), los métodos existentes utilizan las longitudes de las repeticiones como parámetros en cada pasada, y tienen una performance pobre en términos de tiempo y memoria; más aún, los métodos exaustivos producen salidas de tamaño inmanejable (cuadrático). Es posible hallar referencias a los métodos utilizados por diferentes autores y los resultados obtenidos en RepBase Update (*http://www.girinst.org/*, [Jurka 2005]).

En el año 2009 el grupo Kapow desarrolló métodos novedosos para la búsqueda de repeticiones exactas. En Becher et al., 2009 [Becher 2009] se presenta un algoritmo que computa todas las repeticiones perfectas en entradas de hasta 500 millones de bases de nucleótidos utilizando 8 GB de memoria. Éste está basado en la construcción de arreglos de sufijos y en un procedimiento novedoso para extraer todas las repeticiones perfectas en la entrada, sin restricciones sobre la distancia entre las mismas o en su longitud. Para entradas de tamaño n, la complejidad computacional del algoritmo es $\mathcal{O}(n \log n)$ y requiere espacio lineal.

1.4. Organización del Documento

El trabajo está organizado presentando los tres algoritmos propuestos en forma constructiva. El Capítulo 2 describe la nomenclatura utilizada a lo largo de la tesis y propone las definiciones necesarias para el planteo de los problemas. Además, introduce brevemente la nomenclatura bioinformática que será utilizada en la mayor parte de las pruebas. En el Capítulo 3 se presentan algunas estructuras de datos y herramientas que se utilizarán extensivamente a lo largo del trabajo, argumentando su necesidad y describiendo los mecanismos de construcción. Los problemas centrales a resolver se proponen en el Capítulo 4, donde se analizan resoluciones algorítmicas y se demuestra su corrección. En el Capítulo 5 se calcula el uso de recursos en tiempo y memoria de los métodos propuestos, así como las magnitudes de la información de salida. Finalmente, en el Capítulo 6 se documentan los experimentos realizados de medición de tiempos y un análisis preliminar sobre los resultados obtenidos en las pruebas sobre genomas de distintas especies animales.

Se incluye en el Anexo A el código fuente de los algoritmos implementados.

Capítulo 2

Notación y Definiciones

Índice

2.1. Definiciones Básicas

Se denomina *alfabeto* a un conjunto finito Σ de símbolos con un orden lexicográfico definido. Una señal discreta o *cadena* w es una secuencia de símbolos de Σ. En esta tesis trabajaremos únicamente con cadenas *finitas*, es decir, con secuencias finitas de símbolos de Σ. Se denomina Σ^* al conjunto de todas las cadenas finitas sobre Σ.

La cantidad de símbolos que componen una cadena w se denomina la *longitud* de la cadena y se representa como $|w|$. Las posiciones de cada símbolo en una cadena w se numeran de 1 a $|w|$, denotando $w[i]$ el símbolo en la posición i-ésima de w.

Se dice que una cadena w *precede lexicográficamente* a una cadena u si en la primera posición en la que las cadenas difieren, el símbolo en w precede al correspondiente en u. En general, para evitar ambigüedades en el orden de las cadenas, agregaremos al final de cada cadena un símbolo \$ no perteneciente al alfabeto, de modo que una cadena $w = w[1]w[2]...w[|w|]$ quedará como $w\$ = w[1]w[2]...w[|w|]\$$, incrementando en uno la longitud de la cadena. Como convención se considerará al símbolo \$ como lexicográficamente inferior a todos los símbolos del alfabeto.

Si $w = w[1]w[2]...w[|w|]$, entonces $u = w[i]w[i+1]...w[j]$ es la *subcadena* de w que comienza en la posición i y termina en la posición j de w, con $i \geq 1$ y $j \leq |w|$. Se dice que u *ocurre* en la posición i de w y se abrevia como $u = w[i...j]$. Si u es una subcadena de w, w es una *extensión* de u. Si u es una subcadena de w y u es distinto de w, se dice que u es una *subcadena propia*, y que w es una *extensión propia* de u.

Observación 1. *Si u ocurre en w, y w ocurre en v, entonces u ocurre en v. Es decir, la relación de ocurrencia en cadenas es transitiva.*

Un *prefijo* de w es una cadena inicial $u = w[1...i]$ de w, y se dice que w es una *extensión a derecha* de u. Si $u \neq w$, u es un *prefijo propio* de w. Análogamente, un

sufijo de w es una cadena final $u = w[j...|w|]$ de w y w es una *extensión a izquierda* de u. Si $u \neq w$, u es un *sufijo propio* de w. En este documento nos referiremos al j-ésimo sufijo de w como $w_j = w[j...|w|]$.

Observación 2. *Dada una cadena w, una subcadena de w es el prefijo de un sufijo de w.*

Dado un conjunto W de cadenas, se dice que *w ocurre en el conjunto W* si w ocurre en alguna de las cadenas de W.

2.2. Repeticiones Maximales

Siguiendo a Gusfield [Gusfield 1997], definimos repeticiones en una cadena:

Definición 1 (Repeticiones maximales [Gusfield 1997])**.** *Una* repetición maximal *u en una cadena w es una subcadena que ocurre más de una vez en w, y tal que todas sus extensiones propias ocurren menos veces que u en w.*

Definición 2 (Repeticiones supermaximales [Gusfield 1997])**.** *Una* repetición supermaximal *en una cadena w es una subcadena que ocurre más de una vez en w, y tal que todas sus extensiones propias ocurren a lo sumo una sola vez en w.*

Ejemplo 1 (Repeticiones maximales y supermaximales)**.** *La cadena* `catarata` *tiene como repetición maximal a* `ata` *que tiene dos ocurrencias en las posiciones* 2 *y* 6*, y también a* `a` *ya que tiene* 4 *ocurrencias, más que cualquiera de sus extensiones propias. La única repetición supermaximal de* `catarata` *es* `ata`*, ya que sus otras repeticiones tienen extensiones propias que ocurren más de una vez.*

Observación 3. *Toda repetición supermaximal de una cadena w también es repetición maximal de w.*

2.3. Definiciones Biológicas

Dado que la principal aplicación de los métodos descriptos es en el análisis de secuencias de ADN, enumeramos a continuación algunas nociones básicas de la disciplina.

El *genoma* de un organismo es la totalidad de información hereditaria que posee dicho organismo. La información del genoma se organiza en estructuras denominadas *cromosomas*: la cantidad de cromosomas es característica de cada especie.

En la mayor parte de las especies, la información en los cromosomas se codifica en una macromolécula de nombre *ácido desoxirribonucleico*, o *ADN*. El ADN se conforma como una secuencia de *nucleótidos*, cada uno de los cuales contiene una *base nitrogenada* -un compuesto orgánico aromático nitrogenado. Las bases nitrogenadas del ADN son cuatro: Adenina, Guanina, Timina y Citosina, abreviadas A, G, T, C respectivamente. Cada base nitrogenada tiene una base *complementaria* (se complementan A con T, y C con G), de manera que al aparearse una secuencia de

nucleótidos con una secuencia complementaria, se da forma a la geometría de doble hélice que caracteriza el ADN. Los pares de bases nitrogenadas apareadas componen la unidad básica de información genética -análoga al *bit* en informática-, y se abrevia *bp* (*base pair* en inglés).

La *secuenciación* de una molécula de ADN consiste en la obtención a través de métodos bioquímicos del orden de los nucleótidos en dicha secuencia. La longitud de la secuencia, es decir el tamaño de la señal, se mide en múltiplos de la unidad básica de información, el par de bases o *bp*: de esta manera 1 Kbp son mil pares de bases, 1 Mbp son un millón de pares y 1 Gbp son mil millones de pares de bases.

A modo de ejemplo, el genoma humano mide alrededor de 3,2 Gbp, divididos en sus 23 cromosomas.

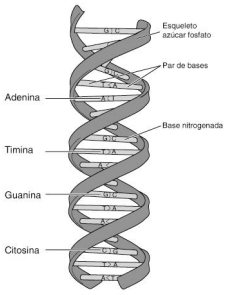

Figura 2.1: Representación esquemática de una cadena de ADN

Capítulo 3

Arreglos de Sufijos

Índice

3.1. Estructuras de Datos

Como demuestra Gusfield [Gusfield 1997], la estructura de datos más eficiente en términos asintóticos para tratar este tipo de problemas es el *árbol de sufijos*: los árboles de sufijos (en inglés llamados *suffix trees*) pueden construirse en tiempo lineal en la longitud de la cadena y requieren espacio en memoria también proporcional al largo de las cadenas a procesar. La resolución de los problemas planteados utilizando árboles de sufijos es posible y relativamente intuitiva, pero queda fuera del alcance de esta tesis.

La principal dificultad que se presenta al trabajar con árboles de sufijos cuando las entradas consisten en grandes volúmenes de datos son las elevadas constantes asociadas a la construcción de los mismos. En especial, los requerimientos de memoria se vuelven prohibitivos cuando las magnitudes de los datos ingresados son comparables a la cantidad de memoria disponible.

Para atender esta limitación en función de las necesidades de procesamiento de cadenas en forma íntegra para entradas significativas, como la secuencia formada por un cromosoma de largo en el orden de las gigabases, se requiere un enfoque algorítmico que contemple una mayor economía en los recursos de espacio de almacenamiento, aún al costo de comprometer ligeramente los tiempos de ejecución.

Las estructuras propuestas para cumplir esta función son los *arreglos de sufijos*, arreglos de enteros que representan el orden lexicográfico de todos los sufijos de una cadena. Los arreglos de sufijos son estructuras más sencillas que los árboles de sufijos que mantienen algunas de sus propiedades y carecen de otras. Para una

descripción más detallada y fundamentación de las ventajas del uso de arreglos de sufijos, ver [Puglisi 2007]. Seguimos la implementación propuesta por Larsson y Sadakane [Larsson 2007].

Una estructura complementaria de los arreglos de sufijos son los arreglos de *máximo prefijo común*. A través de éstos, es posible realizar muchas de las funcionalidades de los árboles de sufijos sobre los arreglos de sufijos. Para la construcción de los arreglos de máximo prefijo común seguimos la implementación de Kasai et al. [Kasai 2001].

3.2. Construcción de Arreglos de Sufijos

Definición 3 (Arreglo de Sufijos [Larsson 2007]). *Dada una cadena* $w = w[1....|w-1|]\$$ *terminada en un caracter* \$ *no perteneciente al alfabeto, el* arreglo de sufijos *de* w *es un arreglo de números* I_w *que consiste en la permutación de los números enteros* $[1, 2, ..., |w|]$*, que cumple la condición:* $I_w[i]$ *precede a* $I_w[j]$ *si y sólo si* w_i *precede lexicográficamente a* w_j*.*

Ejemplo 2 (Arreglo de Sufijos). *Los sufijos de la palabra* yarara\$ *están se ordenan lexicográficamente de la siguiente forma:*

i	*0*	*1*	*2*	*3*	*4*	*5*	*6*
w[i]	*y*	*a*	*r*	*a*	*r*	*a*	*$*

I_w	w_i
6	\$
5	*a*\$
3	*ara*\$
1	*arara*\$
4	*ra*\$
2	*rara*\$
0	*yarara*\$

Cada sufijo se representa con el número de la izquierda, que identifica la posición en la que el sufijo empieza.

En otras palabras, el arreglo de sufijos de una cadena w indica el orden en que se deben recorrer los sufijos de w de modo que estén lexicográficamente ordenados, es decir que el problema de construir un arreglo de sufijos de una cadena w consiste en encontrar el orden lexicográfico de todos sus sufijos.

Según se describe en el artículo citado [Larsson 2007], la construcción de los arreglos de sufijos se puede realizar en forma eficiente según el algoritmo descripto a continuación.

3.2.1. Definiciones Preliminares

Definición 4 ($h-$orden). *Se dice que los sufijos de w están $h-$ordenados si están ordenados considerando el orden lexicográfico de sus primeros h símbolos.*

Ejemplo 3 ($h-$orden). *El siguiente conjunto de sufijos de la palabra* yarara\$

6 \$
5 *a*\$
1 *arara*\$
3 *ara*\$
2 *rara*\$
4 *ra*\$
0 *yarara*\$

está $1-$ordenado y $2-$ordenado, pero no $3-$ordenado, $4-$, $5-$, $6-$ o $7-$ordenado.

Observación 4. *Los sufijos de una cadena w están ordenados si y sólo si están $|w|-$ordenados.*

Definición 5 (Grupo de sufijos $h-$ordenados). *Dado un conjunto de sufijos de una cadena w que están $h-$ordenados, se denomina* grupo[1] *a un subconjunto de sufijos que comparte los primeros h símbolos.*

Observación 5. *En un conjunto $h-$ordenado, todos los elementos de cada grupo son contiguos.*

Proposición 1. *Las posiciones en el arreglo de sufijos de los elementos de un grupo en un conjunto $h-$ordenado serán siempre posiciones dentro del mismo grupo. En particular, si un grupo en un conjunto $h-$ordenado tiene un solo elemento, la posición de ese sufijo en el conjunto $h-$ordenado es la misma que tendrá en el arreglo de sufijos.*

Demostración. Si el conjunto está $h-$ordenado, todos los elementos que difieren en los primeros h elementos mantendrán el mismo orden relativo en el arreglo de sufijos. Por definición, si dos elementos están en distintos grupos, tienen alguna diferencia en sus primeros h elementos, por lo que al ordenar completamente el conjunto de sufijos, cada elemento podrá únicamente cambiar de posición a una posición en su mismo grupo. □

El *ordenamiento de cadenas ternario* [Bentley 1997] es un método de ordenamiento de cadenas basado en el $i-$ordenamiento de las cadenas en forma incremental para i entre 1 y $|w|$, a través del siguiente algoritmo:

1. Inicialmente, las cadenas están –trivialmente– $0-$ ordenadas ($h = 0$), y forman un único grupo.

[1] Sin relación alguna con los grupos como estructuras algebraicas.

2. Cada grupo con más de un elemento es $(h+1)-$ordenado internamente con el algoritmo *quicksort ternario*[2] utilizando como clave para ordenar el $(h+1)-$ésimo símbolo de cada sufijo. Los elementos que en cada llamada coinciden con el pivote conformarán un nuevo grupo en el conjunto $(h+1)-$ordenado.

3. Si $h = |w|$, o todos los grupos son unitarios, el conjunto está ordenado. En caso contrario, se incrementa h y se vuelve al paso 2.

Este algoritmo permite el ordenamiento de cualquier conjunto de cadenas en a lo sumo n pasadas, siendo n la longitud de la cadena más larga. La cantidad de pasadas se puede reducir a una cantidad proporcional al logaritmo de n a través de la siguiente observación, que explota el hecho de que en cada paso de $h-$ordenamiento, todos los sufijos tienen a su vez, todos sus sufijos $h-$ordenados.

Observación 6 ([Manber 1993]). *Si dado un conjunto $h-$ordenado, se ordena cada uno de sus grupos utilizando como clave, para cada sufijo w_i, la posición en el $h-$orden del sufijo w_{i+h}, el conjunto queda $2h-$ordenado.*

Es decir, dado un grupo que comparte los primeros h símbolos, en lugar de realizar una pasada ordenando de acuerdo al $(h+1)-$ésimo símbolo, es posible utilizar como clave el segundo bloque de h símbolos. Esto $2h-$ordena el conjunto ya que los $(h+1)-$ésimos símbolos de cada cadena fueron $h-$ordenados en el paso anterior, puesto que son sufijos de la misma cadena.

3.2.2. Algoritmo de Construcción de Arreglos de Sufijos

El algoritmo de construcción de los arreglos de sufijos consiste en la aplicación de la última observación al algoritmo de ordenamiento de cadenas ternario: en lugar de realizar $|w|$ pasadas de $h-$ordenamiento, incrementando h en cada una, la observación 6 permite realizar $\lceil \log_2(|w|) \rceil$ pasadas de $h-$ordenamiento, duplicando el valor de h en cada una.

La estructura general del algoritmo quedaría como sigue:

1. Se $1-$ordenan los sufijos utilizando quicksort ternario como en el algoritmo anterior, o algún método equivalente $(h = 1)$.

2. Cada grupo de más de un elemento es $2h-$ordenado internamente con el algoritmo quicksort ternario utilizando como clave para ordenar la posición del $(h+1)-$ésimo símbolo de cada sufijo (en caso de no existir, el sufijo ya se encontrará en un grupo unitario, en su posición final).

3. Si $h = \lceil \log_2(|w|) \rceil$, o todos los grupos son unitarios, el conjunto está ordenado. En caso contrario, se duplica h y se vuelve al paso 2.

[2] El *quicksort ternario* difiere del quicksort convencional en el hecho de que compara los elementos con el pivote si son mayores, menores, o iguales, y ubica a los elementos iguales al pivote en el medio del arreglo, junto a él. Su complejidad asintótica coincide con el quicksort convencional, pero para entradas con muchas repeticiones, es más performante.

Sin embargo, utilizar la como clave para ordenar la posición del $2h-$ésimo símbolo introduce dos dificultades: como cada posición es única, no hay empates con el pivote, con lo que no es posible reconocer los límites de los grupos correspondientes al $2h-$orden. Por otro lado, no son conocidas a priori las posiciones de cada sufijo, por lo que se debería realizar una búsqueda lineal en el arreglo I_w, lo que volvería el algoritmo bastante ineficiente.

Para resolver esto, se introduce un nuevo arreglo V_w, de longitud $|w|$, que también se irá actualizando en cada paso del algoritmo. $V_w[i]$ guardará para cada iteración, el *número de grupo* en el que se encuentra el sufijo que comienza en la posición i, y se utilizará como clave en las comparaciones en lugar de las posiciones de los sufijos. El número de un grupo en un conjunto $h-$ordenado es la última posición en el arreglo de sufijos en el que hay un elemento de ese grupo. Con esta modificación, el algoritmo puede acceder en tiempo constante al número de grupo de cada sufijo, y se garantiza que los grupos que queden igualados al pivote en cada paso del quicksort ternario formarán un nuevo grupo $2h-$ordenado.

Al final del proceso, cada sufijo forma un grupo $|w|-$ordenado, por lo que su número de grupo será la posición de dicho sufijo en el arreglo de sufijos I_w. Por lo tanto, al concluir el algoritmo, el arreglo V_w contendrá la permutación inversa de I_w, de modo tal que $V_w[I_w[i]] = i$ para todo i entre 1 y $|w|$.

La descripción del algoritmo más detallada, así como su demostración y optimización pueden encontrarse en [Larsson 2007].

3.3. Máximo Prefijo Común

Para dar cuenta de las similitudes entre distintas subcadenas de una cadena w es de utilidad conocer el máximo prefijo común entre dos sufijos cualesquiera de w. Para calcularlos, se define el arreglo de *máximo prefijo común* (LCP_w, de *Longest Common Prefix en inglés*) de w.

Definición 6 (Máximo Prefijo Común)**.** *La longitud del máximo prefijo común $lcp(w, s)$ entre dos cadenas w, s es la longitud del prefijo más largo de w que también es prefijo de w.*

Definición 7 (Arreglo de Máximo Prefijo Común)**.** *El arreglo de máximo prefijo común LCP_w de una cadena w con un arreglo de sufijos asociado I_w es un arreglo de enteros de longitud $|w| - 1$ que contiene en cada posición i el máximo prefijo común entre los sufijos $w_{I_w[i]}$ y $w_{I_w[i+1]}$, es decir $lcp(w_{I_w[i]}, w_{I_w[i+1]})$.*

Ejemplo 4 ($lcp(\cdot,\cdot)$)**.** *Por ejemplo,*
$lcp(yarara, mara) = 0$,
$lcp(\underline{ya}rara, \underline{ya}guarete) = 2$,
$lcp(\underline{mara}, \underline{mara}ña) = 4$.

Ejemplo 5 (LCP_w)**.** *El arreglo LCP_w de la cadena $w =$* `yarara` *es:*

LCP_w	I_w	w_i
0	6	\$
1	5	$a\$$
3	3	$ara\$$
0	1	$arara\$$
2	4	$ra\$$
0	2	$rara\$$
	0	$yarara\$$

Para la construcción del arreglo LCP_w seguimos el algoritmo propuesto por [Kasai 2001]. Éste se vale de las siguientes observaciones:

Observación 7. *Dado que el orden lexicográfico de los sufijos de w representado por el arreglo de sufijos I_w mantiene juntos a los sufijos que tienen prefijos similares, para cualquier par de sufijos w_i, w_j, se cumple: $lcp(w_i, w_j) = \text{mín}(LCP_w[i], LCP_w[i+1], ..., LCP_w[j-1])$.*

A partir de esta observación se deduce que si se conoce el valor de una $lcp(w_i, w_j)$ entre dos sufijos w_i, w_j de w, todos los valores que haya en el arreglos LCP_w entre los índices i y $j-1$ serán necesariamente mayores o iguales a $lcp(w_i, w_j)$.

Observación 8. *Si $lcp(w_i, w_j) \geq 1$ y $I_w[i] < I_w[j]$, entonces $I_w[i+1] < I_w[j+1]$.*

Es decir, si dos sufijos w_i, w_j de w tienen un prefijo común de longitud al menos 1 y w_i precede lexicográficamente a w_j, entonces respectivos sufijos siguientes w_{i+1} y w_{j+1} mantendrán el orden lexicográfico. Dado que comparten el primer símbolo, éste no afectará el orden lexicográfico de los sufijos siguientes.

Observación 9. *Si $lcp(w_i, w_j) \geq 1$, entonces $lcp(w_{i+1}, w_{j+1}) = lcp(w_i, w_j) - 1$*

Para sufijos que tienen un prefijo común de longitud no nula, sus respectivos sufijos siguientes tendrán un valor de *lcp* uno menor, ya que la operación es equivalente a borrar el primer símbolo de cada sufijo.

De estas observaciones se deduce lo siguiente: si el valor de $LCP_w[V_w[i]]$ es mayor a 1, entonces el valor de $LCP_w[V_w[i+1]]$ será al menos $LCP_w[V_w[i]]-1$. Por lo tanto, si se conoce el valor de $LCP_w[V_w[i]]$, para calcular el valor de $LCP_w[V_w[i+1]]$ bastará con comparar los símbolos a partir del $LCP_w[V_w[i]]-$ésimo entre w_{i+1} y el siguiente elemento en el arreglo de sufijos. Para aprovechar al máximo esta propiedad basta con calcular y asignar los valores de LCP_w recorriendo la cadena en el orden original con i de 1 a $|w|$. De esta manera, siempre se conocerá el valor de $LCP_w[V_w[i]]$ antes de calcular $LCP_w[V_w[i+1]]$.

El arreglo LCP_w se calcula entonces con el siguiente algoritmo:

1. Se asigna un valor h de *lcp acumulado* en 1.

2. Para cada sufijo de la cadena w_i, con i desde la posición 1 hasta $|w| - 1$, se calcula su valor de $LCP_w[V[i]]$ comparando desde el $h-$ésimo símbolo con el sufijo $w_{V[i]+1}$. Por cada símbolo que coincide, se incrementa h. Después de asignar el valor de LCP, se decrementa h si es mayor a 1.

Una descripción más precisa, su demostración y aplicaciones de la construcción del arreglo LCP_w se pueden encontrar en [Kasai 2001].

3.4. Búsqueda de Repeticiones Maximales y Supermaximales

Una solución eficiente a los problemas de la búsqueda exhaustiva de repeticiones maximales y supermaximales de una cadena w (Definiciones 1 y 2) se propone en [Becher 2009]. Se agregan las siguientes definiciones.

Definición 8 (Repeticiones maximales a izquierda (respectivamente derecha))**.** *Una* repetición maximal a izquierda (respectivamente derecha) u *en una cadena* w *es una subcadena que ocurre más de una vez en* w*, y tal que todas sus extensiones propias a izquierda (respectivamente derecha) ocurren menos veces que* u *en* w*.*

Definición 9 (Repeticiones supermaximales (respectivamente derecha))**.** *Una* repetición supermaximal a izquierda (repectivamente derecha) *en una cadena* w *es una subcadena que ocurre más de una vez en* w*, y tal que todas sus extensiones propias a izquierda (respectivamente derecha) ocurren a lo sumo una sola vez en* w*.*

Observación 10. *Una repetición* u *de* w *es maximal (respectivamente supermaximal) si y sólo si es maximal (respectivamente supermaximal) a izquierda y a derecha.*

3.4.1. Repeticiones Supermaximales

Sean una cadena w, su arreglo de sufijos I_w y su arreglo de máximo prefijo común LCP_w. Si u es una repetición supermaximal de w con n ocurrencias, por la Observación 2 deben existir exactamente n sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ de w que tienen a u como prefijo. Dado que $lcp(w_{k_i}, w_{k_j}) \geq |u|\ \forall i, j$ y por la Observación 7, todos los sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ deben ser consecutivos en el arreglo de sufijos I_w.

Además, todos los valores del arreglo LCP_w entre los sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ deben ser exactamente $|u|$, ya que todos tienen a u como prefijo, y si hubiese un valor mayor, existiría una cadena u' extensión propia de u a derecha que ocurriría más de una vez en w, lo que contradice la hipótesis de que u es una repetición supermaximal.

Entonces, dado un conjunto de sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ consecutivos en I_w con igual valor $|u|$ de LCP_w (tales que $LCP[k_1] = LCP[k_1] = ... = LCP[k_{n-1}] = |u|$), se evalúa la condición de supermaximalidad como sigue: La supermaximalidad a derecha consiste en que no existan otras repeticiones u' de w que sean extensiones

propias de u a derecha. Si existiesen tales u', por la Observación 7, alguna de ellas debería ser adyacente a w_{k_1} o a w_{k_n}. Por lo tanto, basta con verificar que los valores de $LCP_w[k_1 - 1]$ y $LCP_w[k_n]$ sean menores a $|u|$. En otras palabras, una subcadena u de w es supermaximal a derecha si es el prefijo de un sufijo que se encuentra en un máximo local del arreglo LCP_w, y su longitud es dicho valor de LCP_w.

Una repetición de w es supermaximal a izquierda si no hay extensiones propias a derecha u' que ocurran más de una vez en w. Para comprobar la supermaximalidad a izquierda, basta verificar las posiciones anteriores de todos los sufijos $w[k_1-1], w[k_2-1], ..., w[k_n-1]$ y comprobar si hay coincidencias. Si son todos distintos, la repetición es supermaximal a izquierda. Esto puede hacerse a través un arreglo de longitud $|\Sigma|$ en el que se asigna a cada posición un símbolo del alfabeto, marcando las posiciones que ocurren como anteriores a cada sufijo.

Entonces, para calcular las repeticiones supermaximales de w se sigue el siguiente algoritmo:

1. Se calculan los arreglos de sufijos I_w y de máxima subcadena común LCP_w de w.

2. Para cada máximo local de LCP_w, si no hay coincidencias entre todos los símbolos anteriores a los sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ que comprende el máximo local, se reporta como un intervalo de I_w en el que hay una repetición supermaximal de longitud $LCP[V[k_i]]$.

3.4.2. Repeticiones Maximales

Al igual que en el caso de las repeticiones supermaximales, para buscar repeticiones maximales en una cadena w se calculan los arreglos de sufijos I_w y de máximo prefio común LCP_w. Si u es una repetición maximal de w con n ocurrencias, por la Observación 2 deben existir exactamente n sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ de w que tienen a u como prefijo. Dado que $lcp(w_{k_i}, w_{k_j}) \geq |u| \forall i, j$ y por la Observación 7, todos los sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ deben ser consecutivos en el arreglo de sufijos I_w.

Todos los valores de LCP_w de estos sufijos serán, naturalmente, mayores o iguales a $|u|$, y al menos uno de ellos debe ser igual a $|u|$, ya que en caso contrario habría una extensión propia a derecha u' de u con la misma cantidad de ocurrencias en w. En otras palabras, toda repetición u de w con n ocurrencias será maximal a derecha si y sólo si es el prefijo de todos los sufijos en un intervalo de I_w donde el mínimo valor del arreglo LCP_w en ese intervalo coincida con $|u|$.

Dado el intervalo de sufijos en I_w en los que ocurre una cadena u tales que el mínimo valor de LCP_w es $|u|$, la maximalidad a izquierda de u puede comprobarse fácilmente: si existe una extensión propia a izquierda de u que ocurre la misma cantidad n de veces que u, entonces todos los símbolos anteriores a $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ en w deben coincidir. En ese caso, los sufijos $\{w_{k_1-1}, w_{k_2-1}, ..., w_{k_n-1}\}$ deberán ser también consecutivos en I_w ya que si hubiese un sufijo intermedio w_i, entonces w_{i+1} debería tener como prefijo a u con lo que estaría en el intervalo $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$.

En caso contrario, o bien los símbolos anteriores a w_{k_1} y w_{k_n} serán distintos o entre w_{k_1} y w_{k_n} habrá menos de n elementos: entonces, u será maximal a izquierda a menos que $w[k_1 - 1] = w[k_n - 1]$ y $V[k_n - 1] - V[k_1 - 1] = n$.

Para hallar las repeticiones maximales de w se recorren las longitudes l de las subcadenas en forma ascendente entre 1 (o la mínima longitud que se quiera considerar) obteniendo todos los máximos intervalos de I_w que tienen al menos l símbolos en común, y se verifican las condiciones de maximalidad. Para hallar los máximos intervalos que comparten l símbolos en forma eficiente se utiliza una estructura de árbol binario completo en el que las hojas representan las posiciones de I_w, en el que se implementan los métodos *máximo menor que* y *mínimo mayor que*, que requieren una cantidad de operaciones logarítmica respecto de la longitud de w.

El algoritmo para la búsqueda de repeticiones maximales queda como sigue:

1. Se calculan los arreglos de sufijos I_w y de máxima subcadena común LCP_w de w. Se fija una mínima longitud de las repeticiones a detectar en l.

2. Se buscan todos los máximos intervalos de sufijos $\{w_{k_1}, w_{k_2}, ..., w_{k_n}\}$ que tienen prefijos en común de al menos l símbolos, y tales que exista algún i entre 1 y $n - 1$ tal que $LCP_w[k_i] = l$. Para cada uno, si se verifican las condiciones de maximalidad a izquierda, se reporta como un intervalo de I_w en el que hay una repetición supermaximal de longitud $LCP[V[k_i]]$.

3. Se incrementa l y se vuelve al paso 2.

Para una descripción más detallada de los algoritmos, cálculo de complejidad computacional y requerimientos de espacio de almacenamiento, ver [Becher 2009].

Capítulo 4

Algoritmos sobre Conjuntos

Índice

La resolución de problemas sobre *conjuntos* de cadenas introduce una dificultad relacionada con los recursos de espacio en memoria necesarios para relacionar todos los elementos de los conjuntos. En [Babenko 2008], Maxim Babenko y Tatiana Starikovskaya resuelven con un elegante método un problema similar a uno de los que trataremos en este capítulo, la búsqueda de *repeticiones supermaximales en un conjunto*. Sin embargo, el algoritmo requiere poder procesar todos los elementos del conjunto en forma simultánea por lo que todas las cadenas deben encontrarse simultáneamente accesibles en memoria. Esto puede ser excesivamente demandante si las cadenas de la entrada son largas, y especialmente si los conjuntos a procesar tienen muchos elementos.

En función de esta limitación se propone en la sección 4.1 un algoritmo básico que crea una estructura que permite trabajar con una cantidad acotada de cadenas a la vez, almacenando la información relevante. Esta estructura será utilizada luego en las secciones 4.3, 4.2 y 4.4 para resolver los problemas de *repeticiones supermaximales en un conjunto*, *repeticiones maximales y supermaximales exclusivas*, y el cálculo de *etiquetas minimales* respectivamente.

4.1. Algoritmo Básico

Para poder trabajar con una cantidad arbitraria de cadenas en un conjunto, procesando de a una a la vez, definiremos un arreglo que servirá para conservar la información de una cadena s respecto de una cadena *base* w.

Definición 10 (Arreglo de Máxima Subcadena Común). *Dadas dos cadenas w y s, se define como* arreglo de máxima subcadena común *a un arreglo $m_{w\$s}$ de longitud $|w|$ que cumple, para cada i entre 1 y $|w|$, que $m_{w\$s}[V_w[i]]$ contiene la longitud del prefijo más largo de w_i que es subcadena de s, donde $V_w[i]$ es la permutación inversa del arreglo de sufijos I_w de w.*

Ejemplo 6 (Arreglo de Máxima Subcadena Común). *Para las cadenas $w =$ `yarara` y $s =$ `catarata`, el arreglo de máxima subcadena común $m_{w\$s}$ es como sigue (se subrayan las subcadenas de s):*

$m_{w\$s}$	w_i
0	\$
1	$\underline{a}$\$
3	$\underline{ara}$\$
3	$\underline{ara}ra$\$
2	$\underline{ra}$\$
2	$\underline{ra}ra$\$
0	$yarara$\$

Notar que el arreglo $m_{w\$s}$ se encuentra indexado por el orden correspondiente al arreglo de sufijos, y no por el orden original de los sufijos en la cadena.

Dado que una subcadena de s es un prefijo de un sufijo de s, el elemento $i-$ésimo del arreglo de máxima subcadena común $m_{w\$s}$ entre w y s puede calcularse como

$$m_{w\$s}[V[i]] = \max_{1 \leq j \leq |s|} lcp(w_i, s_j).$$

El arreglo de máxima subcadena común entre w y s se construye a partir del arreglo de sufijos y el arreglo $LCP_{w\$s}$ de la concatenación de ambas cadenas $w\$s$ con el separador \$ entre ambos, un símbolo que no se encuentra en el alfabeto (y que también es distinto del símbolo que se agrega al final de la concatenación, implícito en adelante).

Proposición 2. *Si $I_{w\$s}$ es el arreglo de sufijos de $w\$s$ y $LCP_{w\$s}$ es el arreglo de máximo prefijo común de $I_{w\$s}$, para cada posición i entre 1 y $|w|$, $m_{w\$s}[i]$ puede calcularse como el máximo entre $lcp(w_i, s_{ant})$ y $lcp(w_i, s_{sig})$, donde s_{ant} y s_{sig} son los sufijos de s más cercanos a $(w\$s)_i$ hacia atrás y hacia adelante respectivamente en el arreglo de sufijos de $w\$s$.*

Notar que todos los sufijos de s coinciden con algún sufijo de $w\$s$, y que todos los sufijos de w son prefijos de algún sufijo de $w\$s$, por lo que a todos los fines prácticos w_i coincide con $(w\$s)_i$.

Demostración. Por la observación 7, los valores de *lcp* entre dos sufijos de una cadena son monótonos decrecientes a medida que los sufijos se alejan en el arreglo de sufijos. Por lo tanto, el maximo sufijo común entre w_i y cualquier sufijo de s estará entre los dos sufijos de s más próximos a w_i en el arreglo de sufijos. □

También por la observación 7, para calcular el valor de $lcp(w_i, s_{ant})$ y $lcp(w_i, s_{sig})$ basta con calcular el mínimo valor de $LCP_{w\$s}$ entre los intervalos correspondientes, es decir, si $s_{ant} = (w\$s)_j$ y $s_{sig} = (w\$s)_k$,

$$lcp(w_i, s_{ant}) = \text{mín}(LCP_{w\$s}[V[j]], LCP_{w\$s}[V[j]+1], ..., LCP_{w\$s}[V[i]-1])$$

$$lcp(w_i, s_{sig}) = \text{mín}(LCP_{w\$s}[V[i]], LCP_{w\$s}[V[i]+1], ..., LCP_{w\$s}[V[k]-1])$$

Por lo tanto,

$$\begin{aligned} m_{w\$s}[V[i]] = \text{máx}(\, &\text{mín}(LCP_{w\$s}[V[j]], LCP_{w\$s}[V[j]+1], ..., LCP_{w\$s}[V[i]-1]), \\ &\text{mín}(LCP_{w\$s}[V[i]], LCP_{w\$s}[V[i]+1], ..., LCP_{w\$s}[V[k]-1])) \end{aligned} \tag{4.1}$$

Entonces, el arreglo $m_{w\$s}$ se puede construir recorriendo el arreglo de sufijos $I_{w\$s}$ de la cadena $w\$s$ y su respectivo arreglo de máximo prefijo común $LCP_{w\$s}$, y actualizando para cada posición de $I_{w\$s}$ que corresponda a un sufijo de w el valor de $m_{w\$s}[V[i]]$ a través de la fórmula 4.1. Esto se puede implementar realizando dos pasadas (una descendente y otra ascendente) que vayan calculando los mínimos de los intervalos entre posiciones que correspondan a sufijos de s y asignen el mayor valor de las dos pasadas, como se ilustra en el Algoritmo 1.

La pertenencia de cada sufijo de $w\$s$ a w o a s puede verificarse en el arreglo de sufijos $I_{w\$s}$ en forma inmediata según la siguiente observación:

Observación 11. *Un sufijo de la concatenación $w\$s$ en la posición i del arreglo de sufijos $I_{w\$s}$ es un índice de w si y sólo si $I_{w\$s}[i] \leq |w|$, y es un índice de s si y sólo si $I_{w\$s}[i] > |w|$.*

4.2. Repeticiones Maximales y Supermaximales Exclusivas

El primer problema a resolver es el de la búsqueda de *repeticiones maximales y supermaximales de una cadena respecto de un conjunto.* Las mismas se definen como sigue:

Definición 11 (Repeticiones maximales exclusivas respecto de un conjunto)**.** *Dados un conjunto de cadenas W y una cadena w, una* repetición maximal exclusiva de w respecto del conjunto W *es una repetición maximal de w que no ocurre en ningún elemento de W.*

Algoritmo 1 máxima_subcadena_común (**Entrada:** cadena w, cadena s **Salida:** arreglo m)

Se inicializa el arreglo $m[1..|w|]$ en 0
$I :=$ arreglo de sufijos de $w\$s$
$LCP :=$ arreglo de máximo prefijo común de $w\$s$
para todo i desde 2 a $|w\$s| - 1$ tal que $I[i]$ es un índice de w **hacer**
  si $I[i-1]$ es un índice de w **entonces**
    $m[I[i]] := \text{mín}(m[I[i-1]], LCP[i-1])$
  en otro caso
    —$I[i-1]$ es un índice en s—
    $m[I[i]] := LCP[i-1]$
para todo i desde $|w\$s| - 2$ a 1 tal que $I[i]$ es un índice de w **hacer**
  si $I[i+1]$ es un índice de w **entonces**
    $acc := \text{mín}(acc, LCP[i]))$
  en otro caso
    —$I[i+1]$ es un índice de s—
    $acc := LCP[i]$
  $m[I[i]] := \text{máx}(m[I[i]], acc)$

Definición 12 (Repeticiones supermaximales exclusivas respecto de un conjunto)**.** *Dados un conjunto de cadenas X y una cadena w, una* repetición supermaximal exclusiva de w respecto del conjunto X *es una repetición supermaximal de w que no ocurre en ningún elemento de X.*

Ejemplo 7 (Repeticiones maximales y supermaximales)**.** *Dados los conjuntos $X = \{\texttt{yarara}, \texttt{mara}, \texttt{tararira}, \texttt{araña}\}$ y $Y = \{\texttt{loro}, \texttt{gata}\}$, y la cadena $w = \texttt{catarata}$, w no tiene repeticiones maximales ni supermaximales exclusivas respecto de Y, ya que todas sus repeticiones maximales y supermaximales ocurren en algún elemento de Y. La única repetición supermaximal exclusiva de w respecto de X es $\texttt{ata}$, que también es la única repetición maximal exclusiva, dado que la otra repetición maximal de w ocurre en algún elemento de X.*

4.2.1. Análisis

Las repeticiones maximales y supermaximales de una cadena w respecto de un conjunto X son, de todas las repeticiones maximales o supermaximales de w, aquellas que *no* ocurren en X. Para obtenerlas, se calculan primero las repeticiones maximales y supermaximales de w utilizando el algoritmo descripto en [Becher 2009], y se reportarán aquellas que no ocurran en X.

Realizar una búsqueda de cada repetición en el conjunto X podría resultar muy costoso –aún implementado en forma eficiente– ya que el recorrido de la totalidad de repeticiones maximales de una cadena w crece, a lo sumo, cuadráticamente con la longitud de w [Becher 2009]. Por lo tanto, se construirá un arreglo M que funcionará como un *filtro* de las ocurrencias en X. El arreglo M se define, a partir de los arreglos

de máxima subcadena común $m_{w\$x}$ entre w y todos los elementos x de X, sin que sea necesario almacenar los arreglos $m_{w\$x}$ más que temporalmente: sólo se mantendrán los valores máximos de cada posición.

Definición 13. *Dada una cadena w y un conjunto de cadenas X, se define el arreglo M de longitud $|w|$ que cumple para cada posición i,*

$$M[i] = \max_{x \in X} m_{w\$x}[i]$$

Dado que se recorren todos los arreglos $m_{w\$x}$ para cada $x \in X$, el arreglo M representa, para cada posición i de w, la longitud del prefijo más largo de w_i que ocurre en algún elemento de X. El arreglo M puede obtenerse fácilmente iterando sobre todos los arreglos $m_{w\$x}$ y manteniendo para cada posición siempre el máximo valor.

Una vez construido el arreglo M, para cada repetición u maximal o supermaximal de w, se chequea si ésta ocurre o no en X. Si u es una repetición maximal o supermaximal que ocurre en la posición i de w, y además ocurre en X, habrá algún elemento x para el cual $m_{w\$x}[i] \geq |r|$, por lo que $M[i] \geq |r|$. Por el contrario, si u no ocurre en X, todos los $m_{w\$x}[i]$ serán menores que la longitud de la repetición u. Por lo tanto, para comprobar si u es una repetición maximal o supermaximal exclusiva con respecto a X basta con comparar la longitud de u con el valor del arreglo M en la posición de alguna de sus ocurrencias.

4.2.2. Algoritmo de Repeticiones Maximales y Supermaximales Exclusivas

De acuerdo a lo desarrollado en la sección anterior, queda demostrado que las repeticiones maximales y supermaximales de una cadena w respecto de un conjunto X pueden obtenerse a partir de los Algoritmos 2 y 3.

Algoritmo 2 maximales_exclusivas (**Entrada:** cadena w, conjunto de cadenas X)

```
Se inicializa el arreglo M[1..|w|] en 0
para todo x ∈ X hacer
  m :=máxima_subcadena_común(w, x)
  para i := 1 a |w| hacer
    M[i] := máx(M[i], m[i])
para todo u repetición maximal de w hacer
  — Obtenido mediante el algoritmo findmaxr —
  i := primera ocurrencia de u en w
  si |u| > M[i] entonces
    reportar u
```

Algoritmo 3 supermaximales_exclusivas (**Entrada:** cadena w, conjunto de cadenas X)

```
Se inicializa el arreglo M[1..|w|] en 0
para todo x ∈ X hacer
  m :=máxima_subcadena_común(w, x)
  para i := 1 a |w| hacer
    M[i] := máx(M[i], m[i])
para todo u repetición maximal de w hacer
  — Obtenido mediante el algoritmo findsmaxr —
  i := primera ocurrencia de u en w
  si |u| > M[i] entonces
    reportar u
```

4.3. Repeticiones Supermaximales en un Conjunto

Al tratar con conjuntos de cadenas, puede ser de interés conocer qué subcadenas son comunes a todos los elementos del conjunto, con algún criterio de maximalidad. Para esto, se propone la siguiente definición de *repeticiones supermaximales en un conjunto*:

Definición 14 (Repeticiones supermaximales en un conjunto)**.** *Dado un conjunto de cadenas W, una* repetición supermaximal en el conjunto *es una cadena u que ocurre en todos los elementos de W, y tal que ninguna de sus extensiones propias ocurre también en todos los elementos de W.*

Ejemplo 8 (Repeticiones supermaximales en conjuntos)**.** *Si $W = \{\texttt{yararas}, \texttt{maras}, \texttt{tarariras}, \texttt{arañas}\}$ las repeticiones supermaximales de W son* `ara` *y* `as`*. $U = \{\texttt{loro}, \texttt{gata}\}$ no tiene cadenas supermaximales.*

4.3.1. Análisis

Para calcular las repeticiones maximales en un conjunto W se elige un elemento w de W (convenientemente, uno de mínima longitud), que será tomado como cadena *base*, y se lo excluirá del conjunto trabajando en adelante con el conjunto $X = W \setminus \{w\}$. A continuación se construirá un arreglo N, *dual* al arreglo M definido en el problema anterior, también a partir de los arreglos de máxima subcadena común $m_{w\$x}$ de de w y cada uno de los demás elementos x de X, pero en este caso reteniendo los valores mínimos:

Definición 15. *Dada una cadena w y un conjunto de cadenas X, se define el arreglo N de longitud $|w|$ que cumple para cada posición i,*

$$N[i] = \min_{x \in X} m_{w\$x}[i]$$

Manteniendo el mínimo valor de cada posición de los arreglos de máxima subcadena se garantiza que los prefijos de w_i de longitud menor o igual a $N[i]$ ocurren

en todos los elementos de W, y que cualquier prefijo de w_i de longitud mayor a $N[i]$ no ocurrirá en algún elemento de W. En otras palabras, para cada posición i, $w[i...i+N[V[i]]-1]$ es el prefijo de w_i más largo que ocurre en todos los elementos de W.

Por otro lado, si u es una repetición supermaximal de W, u debe ocurrir en todos los elementos de W, y en particular ocurrirá en alguna posición i de w. Por definición de repetición supermaximal, para cada $x \in X$ se deberá cumplir que $m_{w\$x}[i] \geq |w|$, ya que w es un prefijo de w_i, y también ocurre en x. Como esto debe cumplirse para todo x de X, se deriva que

$$N[k] = \min_{x \in X} m_{w\$x}[i] \geq |u|$$

Lema 1. *Si u es una repetición supermaximal de W tal que u es prefijo de w_i, entonces se cumple que $N[i] = |u|$.*

Demostración. $N[i] \geq |u|$ quedó demostrado en el párrafo anterior. Para demostrar que el mínimo efectivamente se alcanza en el menor de los $m_{w\$x}[i]$, se supondrá que $N[i] > |u|$, es decir que para todo $x \in X$, $m_{w\$x}[i] > |u|$. En tal caso, habrá una extensión propia a derecha u' de w que es un prefijo de w_i y para todo $x \in X$, u' ocurre en x, lo que contradice la hipótesis de que u era una repetición supermaximal. □

Con esto podemos afirmar que para toda repetición supermaximal u de W habrá una posición i de w tal que $u = w[i...i+N[V[i]]-1]$, pero no que todas las subcadenas $w[i...i+N[V[i]]-1]$ serán repeticiones supermaximales de W ya que pueden existir extensiones a derecha en otras posiciones o extensiones a izquierda que también ocurran en todos los elementos de W.

Para abordar estas cuestiones en más detalle, utilizaremos las siguientes definiciones:

Definición 16 (Repeticiones supermaximales a izquierda (respectivamente derecha) en un conjunto)**.** *Dado un conjunto de cadenas W, una* repetición supermaximal a izquierda (respectivamente derecha) en el conjunto *es una cadena u que ocurre en todos los elementos de W, y tal que ninguna de sus extensiones propias a izquierda (respectivamente derecha) ocurre también en todos los elementos de W.*

Observación 12. *Una cadena u es una repetición supermaximal en un conjunto W si y sólo si es repetición supermaximal a izquierda y a derecha en el conjunto W.*

4.3.2. Supermaximalidad a izquierda

Sean W un conjunto de cadenas, $w \in W$ una cadena de longitud mínima que se toma como base, y $X = W \setminus w$. Si u es una repetición supermaximal a izquierda de W que ocurre en la posición i de w, ninguna de sus extensiones propias a izquierda debe ocurrir en todos los elementos de X. Por lo tanto, el prefijo más largo de w_{i-1} que ocurre en todas las cadenas de W debe tener longitud menor o igual a

$|u|$. Además, si u ocurre en w en otra posición i', también deberá cumplirse que el prefijo más largo de $w_{i'-1}$ que ocurre en todas las cadenas de W debe tener longitud menor o igual a $|u|$.

Recíprocamente, si para todas las ocurrencias i de una subcadena u de w, se cumple $N[V[i-1]] \leq N[V[i]]$, entonces no hay ninguna extensión a izquierda de u en w que además ocurra en todos los elementos de W.

Lema 2 (Supermaximalidad a izquierda)**.** *Sean W un conjunto de cadenas, $w \in W$ una cadena de longitud mínima que se toma como base, y $X = W \backslash w$. Una subcadena u de w es supermaximal a izquierda en W si y sólo si para todas las posiciones i de w en las que ocurre u se cumple que $N[V[i-1]] \leq N[V[i]]$.*

4.3.3. Supermaximalidad a derecha

La supermaximalidad de una subcadena u en W no es tan inmediata, y requiere definir una partición del arreglo de sufijos de w en intervalos representativos de una única subcadena. Si I_w es el arreglo de sufijos de w, LCP_w es el arreglo de máximos prefijos comunes de I_w y V_w es la permutación inversa de I_w, definimos la siguiente partición en intervalos:

Definición 17. *Dada una cadena w, w_i y w_j estarán en el mismo intervalo si y sólo si $N[V[i]] = N[V[j]]$, y* $\min(LCP[V[i]], LCP[V[i]+1], ..., LCP[V[j]]) \geq N[V[i]]$.

Esta partición agrupa a todos los sufijos que comparten el mismo valor de N, y cuyo valor de LCP es mayor o igual al valor de N: es decir, agrupa a todos los sufijos que tienen *los mismos máximos prefijos* que ocurren en todos los elementos de X y que son contiguos en el arreglo de sufijos I_w. La igual longitud entre prefijos queda garantizada por el igual índice en el arreglo N, y la identidad entre prefijos es asegurada por la condición de superar al valor de N en el arreglo LCP. Dado que los prefijos de longitud $N[V[i]]$ son iguales en todos los elementos del intervalo, se puede asociar dicho prefijo $w[i...i+N[V[i]]-1]$ a cada intervalo.

La condición de supermaximalidad a derecha puede analizarse de la siguiente forma: si un prefijo u de w_i de longitud $N[V[i]]$ no es supermaximal a derecha, debe existir una extensión propia a derecha u' de u, de longitud $|u'| > |u|$ que sea prefijo de otro sufijo w_j, y cuyo valor $N[V[j]]$ debe superar a $N[i]$. Como u es prefijo de u', por la observación 7 todos los valores de LCP entre $V[i]$ y $V[j]$ deben ser mayores o iguales a $|u|$, es decir, todos los sufijos entre $V[i]$ y $V[j]$ tendrán como prefijo a u. En particular, si w_i pertenece al intervalo $[l, l']$, $LCP[l-1]$ o $LCP[l']$ (dependiendo de la ubicación de j) serán mayores o iguales a $|u|$[1].

[1] En realidad, $LCP[l-1]$ o $LCP[l']$ deberán ser estrictamente iguales a $|u|$. Suponiendo sin pérdida de generalidad que $V[i] < V[j]$, debe cumplirse $LCP[l'] = |u|$ ya que en caso contrario $w_{l'}$ tendría más símbolos que $|u|$ en común con un intervalo que tiene asociado un $N[V[j]] > N[V[i]]$ (esta desigualdad se cumple por la definición 17 bajo la condición de que el valor de LCP supera al de N). Esto implicaría que el sufijo $w_{l'}$ tendría al menos $|u|+1$ símbolos en común con el sufijo $w_{I[V[l']+1]}$. Necesariamente, $w_{l'}$ tendría que tener entonces un valor asociado $N[l'] > |u|$, lo que contradice la hipótesis.

Sin embargo, esta observación no es relevante en el algoritmo.

Recíprocamente, si en los dos intervalos adyacentes al de w_i se cumple que para cualquier sufijo w_k del intervalo $lcp(w_i, w_k) < |u| = N[V[i]]$, entonces u deberá ser necesariamente supermaximal a derecha, pues no hay otros sufijos que tengan a u de prefijo (éstos deberían ser contiguos en el arreglo de sufijos). También por la observación 7, los valores $lcp(w_i, w_k)$ para los intervalos adyacentes al intervalo $[l, l']$ al que pertenece w_i serán menores o iguales a los valores del arreglo $LCP[l-1]$ y $LCP[l']$. Arribamos entonces a la condición necesaria y suficiente de maximalidad a derecha:

Lema 3 (Supermaximalidad a derecha)**.** *Sean W un conjunto de cadenas, $w \in W$ una cadena de longitud mínima que se toma como base, y $X = W \setminus w$. Una subcadena u de w es supermaximal a derecha en W si y sólo si es el prefijo de longitud $N[V[i]]$ de un sufijo w_i que se encuentre en un intervalo $[l, l']$ tal que $LCP[l-1] < |u|$ y $LCP[l'] < |u|$.*

4.3.4. Algoritmo de Repeticiones Supermaximales en un Conjunto

A partir de las definiciones y condiciones desarrolladas se propone el Algoritmo 4 para hallar las repeticiones supermaximales de un conjunto W.

Algoritmo 4 supermaximales_conjunto (**Entrada:** conjunto de cadenas W)

```
w:= una cadena de longitud mínima de W
X := W \ {w} —se remueve w de W—
Se inicializa el arreglo N[1..|w|] en |w|
para todo x ∈ X hacer
  m :=máxima_subcadena_común(w, x)
  para i := 1 a |w| hacer
    N[i] := mín(N[i], m[i])
I := arreglo de sufijos de w
V := permutación inversa de I
LCP := arreglo de máximo prefijo común de I
para todo intervalo [l, l'] de acuerdo a la definición 17 hacer
  si (l = 1 o LCP[l − 1] < N[l]) y
     (l' = |w| o LCP[l'] < N[l']) entonces
    si para todo i in [l, l'], (i = 1 or N[i − 1] ≤ N[i]) entonces
      reportar w[I[i]...I[i] + N[i] − 1]
```

El chequeo de la definición de intervalo puede hacerse en forma inmediata recorriendo los valores de LCP y N en el sentido del arreglo de sufijos y evaluando las condiciones necesarias.

Teorema 1 (Supermaximalidad en Conjuntos)**.** *El Algoritmo 4 calcula todas las repeticiones supermaximales en un conjunto W.*

Demostración. El Lema 1 asegura que todas las repeticiones supermaximales serán prefijo de algún sufijo w_i de w, de longitud $N[V[i]]$. El algoritmo verifica la condición

de supermaximalidad a derecha derivada en el Lema 3 y luego chequea la supermaximalidad a izquierda de acuerdo al Lema 2 en cada intervalo. Queda asegurado que en todo intervalo para el que se evalúe esta condición estarán todas las ocurrencias de la subcadena evaluada, ya que sólo se chequean los intervalos que representan cadenas supermaximales a derecha. □

4.4. Etiquetas Minimales

Al superponer los dos problemas anteriores, sin las exigencias de maximalidad, se delimita el problema de búsqueda de las *etiquetas* de un conjunto de cadenas con respecto a otro:

Definición 18 (Etiquetas). *Dados dos conjuntos de cadenas U y W, una* etiqueta *de U con respecto a W es una cadena que ocurre en todos los elementos de U, y no ocurre en ningún elemento de W.*

En especial, se hará énfasis en un criterio de minimalidad para las etiquetas:

Definición 19 (Etiquetas Minimales). *Dados dos conjuntos de cadenas U y W, una* etiqueta minimal *de U con respecto a W es una etiqueta de U con respecto a W tal que ninguna de sus subcadenas propias es etiqueta de U con respecto a W.*

Ejemplo 9 (Etiquetas y Etiquetas Minimales). *Dados los conjuntos $W = \{\texttt{yarara}, \texttt{mara}, \texttt{tararira}, \texttt{araña}\}$ y $U = \{\texttt{loro}, \texttt{gata}\}$, las etiquetas de W respecto de U son* `ar`*,* `ra` *y* `ara`*. De éstas,* `ar` *y* `ra` *son etiquetas minimales de W respecto de U.*

El problema de etiquetas minimales se puede interpretar como la búsqueda de todas las subcadenas minimales que *representan* o caracterizan a los elementos del primer conjunto (por ocurrir en todos sus elementos) en contraposición al segundo.

4.4.1. Análisis

Para resolver el problema de etiquetas minimales, en primer lugar será necesario contar con una representación de todas las cadenas que ocurren en todos los elementos del primer conjunto W. Para esto se procederá en forma similar a la resolución de las repeticiones supermaximales del conjunto: de igual manera que en la sección 4.3.1 se elige un elemento de longitud mínima del conjunto w, y se define un nuevo conjunto que lo excluye $X = W \setminus w$. Sobre w y X se define el arreglo N (Definición 15), que representa la longitud del prefijo más largo de cada sufijo de w que ocurre en todos los elementos de X. En otras palabras, para toda posición i de w, todo prefijo $w[i...i+n-1]$ de w_i de longitud $n \leq N[V[i]]$ ocurrirá en todos los elementos de W.

Por otro lado, para almacenar información sobre las subcadenas de w que ocurren en el conjunto U, también se construirá un arreglo M (Definición 13) sobre w y U, que representa la longitud del máximo prefijo de cada sufijo de w que ocurre en algún

elemento de U. Este arreglo también puede interpretarse como sigue: el prefijo de w_i de longitud $M[V[i]]+1$, es decir $w[i...i+M[V[i]]+1]$, es el prefijo de w_i más corto que *no* ocurre en ningún u de U. Por lo tanto, para cada posición i de w, todo prefijo $w[i...i+n-1]$ de w_i de longitud $n > M[V[i]]$ no ocurrirá en ningún $u \in U$.

A partir de estas derivaciones se infiere el siguiente lema:

Lema 4 (Etiquetas)**.** *Si para una posición i de w se cumple que $M[V[i]] < N[V[i]]$, en i comienzan exactamente $N[V[i]] - M[V[i]]$ etiquetas de longitudes $M[V[i]]+1, M[V[i]]+2, ..., N[V[i]]$. Si por el contrario $M[V[i]] \geq N[V[i]]$, entonces en i no comienza ninguna etiqueta.*

Demostración. Por definición de N todos los prefijos de w_i de longitud menor o igual a $N[V[i]]$ ocurrirán en todos los elementos de W. Por definición de M, todos los prefijos de w_i de longitud mayor a $M[V[i]]$ no ocurrirán en ningún elemento de U. Por lo tanto, por definición de etiqueta, todos los prefijos de w_i de longitud mayor a $M[V[i]]$ y menor o igual a $N[V[i]]$ serán etiquetas de W con respecto a U. Para ver que en i no comienza ninguna otra etiqueta basta ver que todo prefijo de w_i de longitud menor o igual a $M[V[i]]$ ocurrirá por definición de M en algún elemento de U. En forma similar, todo prefijo de w_i de longitud mayor a $N[V[i]]$ no ocurrirá en algún elemento de W, por lo que no será una etiqueta. Por lo tanto, si el valor de $M[V[i]]$ es mayor o igual a $N[V[i]]$, no habrá ninguna etiqueta que comience en la posición i de w. □

Sin profundizar mucho en las exigencias de minimalidad, a partir de la definición de etiquetas minimales y del lema anterior puede hacerse la siguiente observación:

Observación 13. *Si en la posición i de w comienza una etiqueta minimal de W respecto de U, ésta es de longitud a lo sumo $M[V[i]]+1$.*

Esta condición es necesaria pero no suficiente para que una subcadena de w sea etiqueta minimal de W con respecto a U. Utilizaremos definiciones de minimalidad a izquierda y a derecha análogas a las utilizadas en el problema de repeticiones supermaximales en un conjunto:

Definición 20 (Etiquetas minimales a izquierda (respectivamente derecha))**.** *Dados dos conjuntos de cadenas U y W, una* etiqueta minimal a izquierda (respectivamente derecha) *de U con respecto a W es una etiqueta de U con respecto a W tal que ninguno de sus sufijos propios (prefijos propios) es etiqueta de U con respecto a W.*

A continuación, se enuncian las condiciones suficientes de minimalidad a derecha y necesarias y suficientes de minimalidad a izquierda para etiquetas de un conjunto W respecto de otro U.

Lema 5 (Minimalidad a derecha)**.** *Si en una posición i de w comienza al menos una etiqueta, la subcadena $w[i...i+M[V[i]]]$ es una etiqueta minimal a derecha.*

Demostración. Supongamos que, por el contrario, la subcadena $u = w[i...i+M[V[i]]]$ no es minimal a derecha. Esto querría decir que existe un prefijo propio u' de u que también es etiqueta de W respecto de U. Dado que u' es prefijo de u, también lo será de w_i, por lo que será una etiqueta que ocurre en la posición i de w. Pero según el Lema 4, si una etiqueta ocurre en la posición i, su longitud debe ser al menos $M[V[i]] + 1$, lo que contradice la hipótesis. □

Lema 6 (Minimalidad a izquierda)**.** *Una etiqueta $u = w[i...i + M[V[i]]$ que ocurre en una posición i de w es minimal a izquierda si y sólo si $M[V[i]] \leq M[V[i + 1]]$.*

Demostración. Si $u = w[i...i + M[V[i]]]$ es minimal a izquierda, ninguno de sus sufijos propios será también etiqueta de W respecto de U. En especial, $u' = w[i + 1...i + M[V[i]]]$ no debe ser etiqueta de W respecto de U. Dado que u es etiqueta de W respecto de U, u ocurre en todos los elementos de W, y al ser u' subcadena de u, también ocurre u' en todos los elementos de u', por la transitividad de la ocurrencia (Observación 1). Por lo tanto, si u' no es etiqueta de W respecto de U se deberá a que u' ocurre en algún elemento de U. Esto implica que $|u'| \leq M[V[i + 1]]$. Ya que u' es una cadena un símbolo más corta que u, $|u'| = M[V[i]]$.

Recíprocamente, si u no es minimal a izquierda, debe cumplirse que alguno de sus sufijos propios u' sea también etiqueta de W respecto de U. Por la transitividad de la relación de ocurrencia, todos los sufijos de u ocurrirán en todos los elementos de W, ya que u es una etiqueta. Por la misma razón, si u' no ocurre en ningún elemento de U, todas sus extensiones a izquierda tampoco ocurrirán. Por lo tanto, todos los sufijos de u que sean extensiones a izquierda de u' serán también etiquetas de W respecto de U. En particular, $u'' = w[i+1...i+M[V[i]]]$, de longitud $|u''| = M[V[i]]$ deberá ser también etiqueta de W respecto de U. Por el Lema 4, si u'' es etiqueta, entonces debe cumplirse que $M[V[i + 1]] < |u''| = M[V[i]]$. □

4.4.2. Algoritmo de Etiquetas Minimales

Para hallar todas las etiquetas minimales de un conjunto de cadenas W respecto de otro U se calculan los arreglos N respecto de w y $X = W \setminus w$, y M respecto de w y U. Luego, podrían verificarse las condiciones necesarias y suficientes obtenidas en la sección anterior para cada sufijo w_i de w de longitud $M[V[i]] + 1$. Sin embargo, además de la correcta detección de etiquetas minimales, será importante que el algoritmo reporte una sola vez cada etiqueta, aunque ésta tenga más de una ocurrencia.

Por la Observación 7, todas las ocurrencias de la misma etiqueta u estarán juntas en el arreglo de sufijos I_w de w, y tendrán valores de LCP asociados menores o iguales a $|u|$. Por lo tanto, para evitar la repetición de etiquetas, basta descartar aquellas etiquetas u que ocurran en posiciones i tales que los valores $LCP[V[i] - 1]$ sean mayores o iguales a $|u|$.

A partir de las definiciones y condiciones desarrolladas se propone el Algoritmo 5 para hallar las etiquetas minimales de un conjunto de cadenas W respecto de otro U:

Algoritmo 5 etiquetas_minimales (**Entrada:** conjunto de cadenas W, conjunto de cadenas U)

```
w:= una cadena de longitud mínima de W
X := W \ {w} —se remueve w de W—
Se inicializa el arreglo N[1..|w|] en |w|
para todo x ∈ X hacer
  m :=máxima_subcadena_común(w, x)
  para i := 1 a |w| hacer
    N[i] := mín(N[i], m[i])
Se inicializa el arreglo M[1..|w|] en 0
para todo x ∈ U hacer
  m :=máxima_subcadena_común(w, x)
  para i := 1 a |w| hacer
    M[i] := máx(M[i], m[i])
I := arreglo de sufijos de w
V := permutación inversa de I
LCP := arreglo de máximo prefijo común de I
para i := 1 a |w| hacer
  si M[V[i]] < N[V[i]] y
     (V[i] = |w| o M[V[i]] ≤ M[V[i] + 1]) y
     (V[i] = 1 o LCP[V[i] − 1] ≤ M[V[i]]) entonces
    reportar w[i...i + M[V[i]]]
```

Teorema 2 (Etiquetas Minimales). *El Algoritmo 5 calcula todas las etiquetas minimales de un conjunto de cadenas W respecto de otro U.*

Demostración. Inmediato de los Lemas 4, 5 y 6. □

Capítulo 5

Complejidad Computacional

Índice

Dado que la principal motivación de los algoritmos desarrollados surge del procesamiento de señales genéticas, que suelen ser entradas de magnitudes significativas respecto de la memoria disponible en las computadoras utilizadas en el procesamiento de las mismas, es imperativo el cálculo de la memoria necesaria para el procesamiento de cada uno de los algoritmos propuestos. Asimismo, el tamaño de la entrada es suficientemente grande como para que –por ejemplo– un tiempo de ejecución proporcional al cuadrado del mismo vuelva el problema intratable en términos prácticos.

Para evaluar el comportamiento asintótico de los algoritmos utilizaremos la notación de Landau, conocida como *O grande ($\mathcal{O}$)*.

Definición 21 (O grande ($\mathcal{O}$))**.** *Dadas dos funciones $f : \mathbb{N} \to \mathbb{N}$ y $g : \mathbb{N} \to \mathbb{N}$, se dice que $f(n)$ es $\mathcal{O}(g(n))$ o $f(n) \in \mathcal{O}(g(n))$ si y sólo si existen constantes $n_0 \in \mathbb{N}$ y $k \in \mathbb{R}^+$ tales que $k\ |g(n)| \geq |f(n)|\ \forall n \geq n_0$.*

Esta notación significa que para valores suficientemente altos de n, la función f no crecerá más que como un múltiplo de la función g.

También utilizaremos en forma automática, y sin demostración, las siguientes propiedades:

$$k \in \mathbb{R},\ f(n) \in \mathcal{O}(g(n)) \Rightarrow k\ f(n) \in \mathcal{O}(g(n))$$
$$f_1(n) \in \mathcal{O}(g_1(n)),\ f_2(n) \in \mathcal{O}(g_2(n)) \Rightarrow f_1(n) + f_2(n) \in \text{máx}(\mathcal{O}(g_1(n)), \mathcal{O}(g_2(n)))$$
$$f_1(n) \in \mathcal{O}(g_1(n)),\ f_2(n) \in \mathcal{O}(g_2(n)) \Rightarrow f_1(n)\ f_2(n) \in \mathcal{O}(g_1(n)\ g_2(n))$$

Como se acostumbra en la bibliografía de algoritmos, se expresará la complejidad computacional asumiendo que los valores enteros se pueden almacenar en una unidad de memoria, y que las adiciones y productos entre enteros se realizan en tiempo constante ($\mathcal{O}(1)$). Estas presunciones resultan razonables para las magnitudes consideradas en los casos de interés, que involucran valores enteros que entran

en los tamaños de palabra de los procesadores actuales. Si bien los algoritmos son escalables para entradas de cualquier tamaño, los resultados obtenidos en esta sección serán válidos sólo si el tamaño de la entrada se mantiene dentro de las cantidades representables por una palabra del procesador. En otro caso, se deberá incorporar un factor logarítmico asociado a todas las operaciones entre enteros.

Para los algoritmos propuestos, calcularemos los recursos asociados necesarios en tiempo y memoria en función del tamaño de la entrada. Como se verá adelante, ninguno de los algoritmos requerirá más que $\mathcal{O}(m)$ memoria ni correrá en más de $\mathcal{O}(n \log m)$ tiempo, siendo n el tamaño total de la entrada, y m la longitud de la cadena más larga.

En los análisis lo largo del capítulo se trabajará siempre con logaritmos en base 2 salvo indicación explícita (esto en cualquier caso no modifica los resultados asintóticos obtenidos).

5.1. Algoritmo Básico

Proposición 3. *Dadas dos cadenas w y s, el algoritmo* **máxima_subcadena_común** *(Algoritmo 1) puede computarse en tiempo $\mathcal{O}(n \log n)$ y requiere $(|w| + |s| + 1)(2 \beth + \log |\mathcal{A}|) + |w| \beth + \mathcal{O}(1)$ bits de memoria –complejidad espacial lineal–, donde $n = |w| + |s|$ y $\beth$ es el tamaño de palabra del procesador.*

Demostración. El algoritmo **máxima_subcadena_común** tiene esencialmente dos partes: la construcción del arreglo de sufijos $I_{w\$s}$ y su arreglo de máximo prefijo común $LCP_{w\$s}$ asociado a dos cadenas w y s, y el recorrido de dichos arreglos para construir el arreglo de máxima subcadena común $m_{w\$s}$. La construcción del arreglo de sufijos toma $\mathcal{O}(n \log n)$ tiempo [Larsson 2007], y el arreglo de máximo prefijo común, $\mathcal{O}(n)$ [Kasai 2001]. El cálculo de la máxima subcadena común $m_{w\$s}$ se compone de dos recorridos de los arreglos $I_{w\$s}$ y $LCP_{w\$s}$ en los que sólo se realizan operaciones que toman tiempo constante (en particular es constante el tiempo requerido por el chequeo de pertenencia a cada sufijo por la Observación 11), por lo que esta parte toma tiempo $\mathcal{O}(n)$. Por lo tanto, el algoritmo se ejecuta en $\mathcal{O}(n \log n)$ tiempo en total.

En cuanto al cálculo de memoria necesaria, la entrada del algoritmo insume $(|w|+|s|) \log |\mathcal{A}|$ bits. El arreglo de sufijos $I_{w\$s}$ y el arreglo de máximo prefijo común $LCP_{w\$s}$ requieren $|w\$s| \beth$ memoria cada uno, y la salida, el arreglo $m_{w\$s}$, ocupa $|w| \beth$. Más allá de una cantidad constante de variables auxiliares, no se utilizan otras estructuras por lo que el espacio total requerido es de $(|w| + |s| + 1)(2 \beth + \log |\mathcal{A}|) + |w| \beth + \mathcal{O}(1)$ bits. □

5.2. Arreglos M y N

Proposición 4. *Dados una cadena w y un conjunto de cadenas X,*

1. *La construcción del arreglo M de w respecto de X insume $\mathcal{O}(n \log m)$ tiempo, y requiere $m\ (2 \beth + \log |\mathcal{A}|) + 2\ |w| \beth + \mathcal{O}(1)$ bits, donde $n = |w| + \sum_{x \in X} |x|$*

(es decir, el tamaño de la entrada) y $m = \max\{|x| : x \in X \cup \{w\}\}$ *(la longitud de la cadena más larga).*

2. *Los mismos requerimientos de tiempo y memoria son necesarios para la construcción del arreglo* N *de* w *respecto de* X*.*

Demostración. Los arreglos M o N y el arreglo de máxima subcadena común $m_{w\$x}$ (uno solo, ya que cada $m_{w\$x}$ sobrescribe al siguiente una vez que se actualiza el valor de M o N) requieren cada uno $|w| \sqsupseteq$ bits. De acuerdo a la Proposición 3, cada cálculo de arreglo de máxima subcadena común $m_{w\$x}$ utiliza temporalmente un espacio de $(|w| + |x|)(2 \sqsupseteq + \log|\mathcal{A}|)$, además de espacio constante para variables locales. La construcción de los arreglos M y N sólo acumula los valores máximos o mínimos, no utilizando estructuras adicionales, por lo que el espacio en memoria requerido en total es $(m + |w|)\ (2 \sqsupseteq + \log|\mathcal{A}|) + 2\ |w| \sqsupseteq + \mathcal{O}(1)$ bits, donde m es la longitud de la cadena más larga en la entrada.

El tiempo requerido para construir los arreglos M y N está dominado por los tiempos de construcción de los arreglos de máxima subcadena común $m_{w\$x}$ para w con cada $x \in X$. Estos suman $\mathcal{O}(\sum_{x\in X}(|w|+|x|)\log(|w|+|x|))$. Estudiaremos cada uno por separado:

En el caso del arreglo N, la elección de la cadena w como una de mínima longitud garantiza que $|w| \leq |x|\ \forall x \in X$, por lo que el valor total de la expresión es

$$\sum_{x\in X}(|w|+|x|)\log(|w|+|x|) \leq \sum_{x\in X} 2|x|\log(2|x|) \leq \sum_{x\in X} 2|x|\log(2m)$$
$$\leq 2\log(2m)\sum_{x\in X}|x| \leq 2\log(2m)\ n \in \mathcal{O}(n\log m)$$

Para el caso del arreglo M, la cadena w no puede elegirse y por lo tanto ésta podría ser más larga que muchas cadenas de X, lo que podría aumentar la cota superior del tiempo de ejecución. Para evitar esto, se calculará el algoritmo sobre la cadena w y una variante X' del conjunto de cadenas X, construido como sigue: Originalmente, $X' = X$. Luego si dos elementos x_i, x_j de X' tienen longitud menor a $m/2$, éstos son reemplazados en X' por la concatenación de ambos $x_i\$x_j$ con un símbolo \$ fuera del alfabeto como separador, realizando esta operación hasta que no haya más de un elemento de longitud menor a $m/2$.

Dado que el arreglo M representa la longitud del prefijo más largo de cada sufijo de w que *ocurre en algún* elemento de X, puede verse que es equivalente calcular w respecto de X que w respecto de X' (ya que las únicas diferencias serán originadas por separadores fuera del alfabeto, aparecerán en X' las mismas subacadenas de w que en X). Por construcción de X', ninguno de sus elementos tendrá longitud mayor a m, y salvo a lo sumo uno de ellos, longitud mayor o igual a $m/2$, por lo que la cantidad de elementos $|X'|$ de X' será tal que $\frac{n-|w|}{m} \leq |X'| \leq \frac{n}{m/2}$.

Puede observarse también que $\sum_{x\in X'}|x| \leq |X| + \sum_{x\in X}|x| \leq 2\sum_{x\in X}|x|$, ya que X' consiste únicamente de cadenas de X concatenadas (el agregado de tamaño se deberá únicamente a los símbolos añadidos en la concatenación).

Entonces, el tiempo requerido por el algoritmo será

$$\sum_{x \in X'} (|w| + |x|) \log(|w| + |x|) \leq |X'| 2m \log(2m) \leq \frac{n}{m/2} 2m \log(2m)$$
$$= 4n \log(2m) \in \mathcal{O}(n \log m)$$

□

5.3. Repeticiones Maximales y Supermaximales Exclusivas

Primero probaremos que el tamaño de la salida de este problema se puede representar en espacio lineal, pues el tiempo de ejecución deberá ser suficiente para generar la salida al mismo. Para eso utilizaremos el siguiente teorema:

Teorema 3 ([Becher 2009]). *Para cualquier cadena w, el conjunto de todas las repeticiones maximales y todas sus ocurrencias pueden representarse en espacio $\mathcal{O}(|w|)$.*

Demostración. Cada repeticion maximal u de w se representa como una terna de enteros: las posiciones inicial y final en el arreglo de sufijos I_w en las que ocurre u, y la longitud de u, es decir que para cada repetición maximal el espacio utilizado en representarla es constante independientemente de su longitud o cantidad de ocurrencias. En cada iteración del algoritmo de búsqueda de repeticiones maximales descripto en la sección 3.4.2 y en [Becher 2009] se inhibe al menos una posición del arreglo de sufijos I_w, aquella que tiene el mínimo valor de LCP_w y define la longitud de la repetición. Por lo tanto, hay a lo sumo $|w|$ repeticiones maximales, cada una de las cuales se representa en espacio constante, con lo que para representar el conjunto de todas las repeticiones maximales y sus ocurrencias basta con $\mathcal{O}(|w|)$ bits. □

Proposición 5. *El conjunto de repeticiones maximales o supermaximales de w con respecto a un conjunto de cadenas X y todas sus ocurrencias en w pueden representarse en $\mathcal{O}(|w|)$ bits.*

Demostración. Por las Definiciones 11 y 12, el conjunto de repeticiones maximales exclusivas es un subconjunto de las repeticiones de w, y el conjunto de repeticiones supermaximales exclusivas es a su vez un subconjunto de éste. Por el teorema anterior, estas son representables en espacio $\mathcal{O}(|w|)$. □

Teorema 4. *El cálculo de las repeticiones maximales o supermaximales de una cadena w respecto de un conjunto X a través de los Algoritmos 2 y 3 requiere $\mathcal{O}(n \log m)$ tiempo de ejecución y $\mathcal{O}(m)$ memoria, donde $n = \sum_{x \in X} |x|$ y $m = \max\{|x| : x \in X\}$. Precisamente, requerirá $(m + |w|)\ (2 \sqsupseteq + \log |\mathcal{A}|) + 2\ |w| \sqsupseteq + \mathcal{O}(1)$ bits.*

Demostración. Los Algoritmos 2 y 3 consisten en el cálculo del arreglo M y el filtrado de las repeticiones maximales o supermaximales que ocurren en algún elemento de

X. El cálculo del arreglo M toma $\mathcal{O}(n \log m)$ tiempo y utiliza $\mathcal{O}(m)$ bits de memoria, como se prueba en la Proposición 4. Veremos que la construcción de este arreglo domina el resto del algoritmo de búsqueda de repeticiones.

Una vez calculado el arreglo M se computan las repeticiones maximales o supermaximales de la cadena w. Estos algoritmos toman ambos $\mathcal{O}(|w| \log |w|)$ tiempo de ejecución, como se demuestra en los Teoremas 15 y 23 de [Becher 2009], y utilizan $\mathcal{O}(m)$ espacio. El chequeo de ocurrencia o no en X se realiza en tiempo constante para cada ocurrencia, por lo que el tiempo total del filtrado es $\mathcal{O}(|w| \log |w|)$, que queda subsumido en el paso anterior.

Durante el filtrado, además del espacio requerido por el algoritmo de búsqueda de repeticiones en w únicamente es necesario el espacio para almacenar el arreglo M de tamaño $|w|$, por lo que el espacio necesario sigue siendo lineal en m. La cantidad exacta de memoria necesaria surgirá del máximo de las cantidades necesarias para la construcción del arreglo M y para el filtrado, operaciones que se realizan secuencialmente. Estos requerimientos, en bits, son:

Repeticiones maximales: (Teorema 15, [Becher 2009])	$\|w\|(4 \sqsubseteq + \log \|\mathcal{A}\| + 2) + \mathcal{O}(1)$
Repeticiones supermaximales: (Teorema 23, [Becher 2009])	$\|w\|(3 \sqsubseteq + \log \|\mathcal{A}\| + 2) + \|\mathcal{A}\| + \mathcal{O}(1)$
Arreglo M: (Proposición 4)	$(m + \|w\|)\ (2 \sqsubseteq + \log \|\mathcal{A}\|) + 2\ \|w\| \sqsubseteq + \mathcal{O}(1)$

Teniendo en cuenta que $m \geq |w|$ y asumiendo que $|w| \geq |\mathcal{A}|$, en todos los casos resulta el mayor requerimiento de memoria el necesario para la construcción del arreglo M. □

5.4. Repeticiones Supermaximales en un Conjunto

Al igual que en la sección anterior, requeriremos que el tamaño de la salida de este problema no supere los objetivos de complejidad temporal, por lo que demostraremos que todas las repeticiones supermaximales de un conjunto de cadenas W puede representarse en espacio lineal en la longitud del elemento más corto de W.

Proposición 6. *El conjunto de repeticiones supermaximales de un conjunto de cadenas W puede representarse en $\mathcal{O}(|w|)$ bits, donde w es una cadena de mínima longitud de W.*

Demostración. Todas las repeticiones supermaximales de W deben ocurrir en todos sus elementos. En especial, deben ocurrir todas en una cadena de longitud mínima w. Dado que no puede haber una cadena supermaximal de W que sea subcadena propia de otra, en cada posición i de w ocurrirá a lo sumo una repetición supermaximal de W. Como para representar la repetición supermaximal basta con una posición i en w y su longitud, pueden representarse todas las repeticiones supermaximales de W reproduciendo la cadena w, y a lo sumo $|w|$ pares posición-longitud, es decir que basta con $\mathcal{O}(|w|)$ espacio para representar la salida. □

Teorema 5. *El cálculo de las repeticiones supermaximales de un conjunto W a través del Algoritmo 4 requiere $\mathcal{O}(n \log m)$ tiempo de ejecución y $\mathcal{O}(m)$ memoria, donde $n = \sum_{x \in W} |x|$ y $m = \max\{|x| : x \in W\}$. Precisamente, requerirá $(m + |w|)\ (2 \sqsupseteq + \log |\mathcal{A}|) + 2\ |w| \sqsupseteq + \mathcal{O}(1)$ bits, donde w es una cadena de mínima longitud de W.*

Demostración. Sea w un elemento de mínima longitud de W y $X = W \setminus w$, como en el Algoritmo 4. El algoritmo se ejecuta en la adición de tiempos de la construcción de las estructuras N, I_w y LCP_w, sumado a lo que toma el ciclo que evalúa las condiciones de maximalidad en los intervalos definidos.

Según se demuestra en la Proposición 4, el tiempo de corrida del cálculo del arreglo N es $\mathcal{O}(n \log m)$. Este comportamiento asintótico domina los tiempos necesarios para la construcción de I_w, $(\mathcal{O}(|w| \log |w|))$, y LCP_w $(\mathcal{O}(|w|))$. El último ciclo verifica una cantidad fija de condiciones que requieren tiempo constante una sola vez para cada posición de w, por lo que el mismo se ejecuta en tiempo lineal en $|w|$, con lo que el tiempo total de ejecución del algoritmo es $\mathcal{O}(n \log m)$.

La memoria necesaria para la ejecución del Algoritmo 4 será la máxima entre las necesarias para la construcción de las estructura N, $m\ (2 \sqsupseteq + \log |\mathcal{A}|) + 2\ |w| \sqsupseteq + \mathcal{O}(1)$ bits por la Proposición 4 y el espacio necesario para almacenar los arreglos N, I_w, LCP y el contenido de w, es decir $|w|(3 \sqsupseteq + \log |\mathcal{A}|)$. La primera es claramente mayor, y es $\mathcal{O}(m)$.

□

5.5. Etiquetas Minimales

Nuevamente, comenzaremos por demostrar que el espacio en memoria ocupado por la salida no superará los objetivos de complejidad temporal del algoritmo.

Proposición 7. *El conjunto de etiquetas minimales de un conjunto de cadenas W con respecto a otro conjunto de cadenas U puede representarse en $\mathcal{O}(|w|)$ bits, donde w es una cadena de mínima longitud de W.*

Demostración. Todas las cadenas minimales de W deben ocurrir en todos sus elementos. En especial, deben ocurrir todas en una cadena de longitud mínima w. Debido a las exigencias de minimalidad, no puede haber una etiqueta minimal de W que sea extensión propia de otra, por lo que en cada posición i de w ocurrirá a lo sumo una etiqueta minimal de W. Como para representar la repetición supermaximal basta con una posición i en w y su longitud, pueden representarse todas las repeticiones supermaximales de W reproduciendo la cadena w, y a lo sumo $|w|$ pares posición-longitud, es decir que basta con $\mathcal{O}(|w|)$ espacio para representar la salida. □

Teorema 6. *El cálculo de las etiquetas minimales de un conjunto W respecto de otro conjunto de cadenas U a través del Algoritmo 5 requiere $\mathcal{O}(n \log m)$ tiempo de ejecución y $\mathcal{O}(m)$ memoria, donde $n = \sum_{x \in W} |x| + \sum_{x \in U} |x|$ y $m = \max\{|x| : x \in$*

$W \cup U\}$. *Precisamente, requerirá* $(m + |w|)\ (2 \sqsubseteq + \log|\mathcal{A}|) + 2\ |w| \sqsubseteq + \mathcal{O}(1)$ *bits, donde* w *es una cadena de mínima longitud de* W.

Demostración. Sea w un elemento de mínima longitud de W y $X = W \setminus w$, como en el Algoritmo 5. El algoritmo se ejecuta en la adición de tiempos de la construcción de las estructuras N de W, M de w respecto de U, I_w y LCP_w, sumado a lo que toma el ciclo que evalúa si en cada posición comienza una etiqueta, y las condiciones de minimalidad y unicidad en w.

Según se demuestra en la Proposición 4, los tiempos de corrida de cálculo de los arreglos M y N son $\mathcal{O}(n' \log m')$ y $\mathcal{O}(n'' \log m'')$ respectivamente considerando $n' = |w| + \sum_{x \in U} |x|$, $m' = \text{máx}\{|x| : x \in U \cup \{w\}\}$ para la construcción de M, y $n'' = \sum_{x \in W} |x|$, $m'' = \text{máx}\{|x| : x \in W\}$ para N. Dado que $n' + n'' \leq 2n$ y $m = \text{máx}(m', m'')$, el tiempo total de la construcción de los arreglos M y N será $\mathcal{O}(n \log m)$. Este comportamiento asintótico domina los tiempos necesarios para la construcción de I_w, $(\mathcal{O}(|w| \log |w|))$, y LCP_w $(\mathcal{O}(|w|))$. Cada iteración del último ciclo se ejecuta claramente en tiempo constante, por lo que el mismo se ejecuta en tiempo lineal en $|w|$, con lo que el tiempo total de ejecución del algoritmo es $\mathcal{O}(n \log m)$.

La memoria necesaria para la ejecución del Algoritmo 5 será la máxima entre las necesarias para la construcción de las estructuras N y M, $m\ (2 \sqsubseteq + \log|\mathcal{A}|) + 2\ |w| \sqsubseteq + \mathcal{O}(1)$ bits por la Proposición 4 y el espacio necesario para almacenar los arreglos M, N, I_w, LCP y el contenido de w, es decir $|w|(4 \sqsubseteq + \log|\mathcal{A}|)$. Puede verse que la primera es mayor, y es $\mathcal{O}(m)$.

□

Capítulo 6

Resultados

Índice

En este capítulo se evaluarán los métodos descriptos tanto en su eficiencia como en su capacidad de arrojar información útil. En la sección 6.1 se expresan los resultados de tiempos de ejecución experimentales para distintos tipos de entradas, y en la sección 6.2 se evalúan algunos resultados obtenidos por los algoritmos sobre secuencias genómicas de distintas especies.

Los algoritmos fueron implementados en el lenguaje de programación C (ANSI C99) para computadoras de 32 o 64 bits. Se utilizó el compilador GCC versión 4.2.4 con optimización normal (opción `-O2`). Las pruebas fueron ejecutadas sobre uno de los procesadores de una unidad Intel® Core™2 Duo E6300, corriendo a 1.86 GHz con 8 GB de RAM (DDR2-800) sobre el sistema operativo Ubuntu Linux para 64 bits.

6.1. Tiempos de Ejecución

Para evaluar el desempeño de los algoritmos se realizaron una serie de pruebas de estrés con distintos tipos de entradas de tamaño significativo, incluyendo entradas con lenguaje natural, lenguajes artificiales, casos de borde y naturalmente, secuencias biológicas. Se describen los conjuntos de archivos utilizados en el Cuadro 6.1.

Conjunto	Descripción
as	Dos archivos con la letra 'a' repetida 2 millones de veces y 65536 veces
txts-big	Los dos archivos más grandes del Canterbury Corpus[†]
txts	Cuatro archivos de textos del Canterbury Corpus
linux-hs	7043 Encabezados (archivos '.h') del kernel Linux, versión 2.6.31
linux-hs	9985 Implementaciones (archivos '.c') del kernel Linux, versión 2.6.31
HS-genome-big	Los cromosomas 1, 2 y 3 del humano, en formato FASTA (NCBI 36.49)
HS-genome	Los 24 cromosomas del humano, en formato FASTA (NCBI 36.49)

[†] El Canterbury Corpus es una colección de textos estándar para mediciones de algoritmos de compresión. Los utilizamos aquí debido a su longitud aunque los métodos descriptos en este trabajo se relacionan sólo indirectamente con técnicas de compresión.

Cuadro 6.1: Conjuntos de Archivos de Entrada

Para las pruebas, en cada corrida se eligió un elemento como *base* (la cadena w en los ejemplos de los capítulos anteriores). En el caso de la búsqueda de repeticiones maximales y supermaximales exclusivas, es la cadena en la que se buscan las repeticiones. En el caso de repeticiones supermaximales en un conjunto, y etiquetas minimales, la elección del elemento base puede afectar significativamente el tiempo de ejecución del algoritmo (ver Demostración de la Proposición 4), por lo que se eligieron los elementos de mayor y menor longitud del conjunto para apreciar esta diferencia. Los tamaños de los conjuntos utilizados y las bases elegidas en cada uno se detallan en el Cuadro 6.2.

Se ilustran en el Cuadro 6.3 los tiempos de la ejecución de los tres algoritmos para cada conjunto. En el caso del algoritmo de etiquetas minimales se dividió cada conjunto en dos particiones de similar cantidad de elementos. Se discriminó en las mediciones el tiempo insumido por cada parte del proceso: construcción de arreglos de sufijos (suma de todas las construcciones, independientemente del paso en el que se utilizan), cálculo de arreglo LCP, cálculo de los arreglos M y N (insumen el mismo tiempo), búsqueda de repeticiones maximales exclusivas, búsqueda de repeticiones supermaximales exclusivas, búsqueda de repeticiones supermaximales del conjunto y etiquetas minimales.

Los resultados de los experimentos se ajustan adecuadamente a las predicciones teóricas de comportamiento asintótico. Las primeras tres columanas del Cuadro 6.3 crecen en forma similar al tamaño total del conjunto analizado, mientras que las últimas cuatro crecen con el tamaño de la base elegida (mucho menor en la mayor parte de los casos).

Entrada	Conjunto	Total del Conjunto (bytes)	Base	Longitud de la Base (bytes)
1	as	2 065 536	a64K.txt	65536
2	as	2 065 536	a2M.txt	2 000 000
3	txts-big	6 520 792	world192.txt	2 473 400
4	txts-big	6 520 792	bible.txt	4 047 392
5	txts	6 798 060	asyoulik.txt	125 179
6	txts	6 798 060	bible.txt	4 047 392
7	linux-hs	54 582 944	genapic.h	22
8	linux-hs	54 582 944	me4000_firmware.h	781 415
9	linux-cs	197 594 330	regs.c	32
10	linux-cs	197 594 330	nls_cp949.c	875 265
11	HS-genome-big	2 975 638 422	chr3.actgn	200 851 322
12	HS-genome-big	2 975 638 422	chr1.actgn	252 811 345
13	HS-genome	3 139 901 384	chr21.actgn	48 817 465
14	HS-genome	3 139 901 384	chr1.actgn	252 811 345

Cuadro 6.2: Tamaño de las Entradas

En cuanto a las magnitudes, la primera columna concentra un muy alto porcentaje de los tiempos totales en todos los casos: es la única que representa una parte del procesamiento que involucra la totalidad de la entrada que no tiene complejidad lineal sino $\mathcal{O}(n \log n)$. En cuanto a las últimas columnas, la primera, correspondiente a la búsqueda de repeticiones maximales exclusivas, es claramente la más lenta, ya que es la única otra columna que representa un proceso que de complejidad $\mathcal{O}(n \log n)$.

Finalmente, pueden notarse las diferencias más contundentes en aquellos conjuntos de mayor diámetro[1], como en los conjuntos de archivos del código fuente del kernel Linux. En los casos en los que haya una gran diferencia entre las longitudes de los elementos de los conjuntos, una elección errónea del elemento base puede volver un problema intratable. Para estos casos se evidencia la necesidad de la elección del elemento de menor longitud en la construcción del arreglo N y la concatenación de cadenas para reducir sus diferencias relativas en la construcción del arreglo M.

6.2. Análisis e Interpretación

Sin realizar interpretaciones desde el punto de vista biológico, pues excedería el tema del presente trabajo, se desarrollaron una serie de pruebas sobre un conjunto de genomas de ocho especies de mamíferos y una aviar. Los genomas utilizados, extraídos de la base de datos de Ensembl[Ensembl 2009], se listan en el Cuadro 6.4.

[1] Se define diámetro de un conjunto como la mayor diferencia de longitudes entre todos los pares

Entrada	Arreglo de Sufijos	Arreglo LCP	Arreglos M/N
1	3.02	0.06	0.01
2	5.33	0.11	0.04
3	8.73	0.81	0.23
4	8.90	0.83	0.29
5	5.45	0.53	0.05
6	16.88	1.59	0.91
7	18.33	2.01	0.20
8	5 421.00	352.39	294.96
9	59.66	6.87	0.66
10	4 987.65	347.68	275.65
11	2 844.43	204.80	85.64
12	3 224.98	222.79	106.63
13	9 552.86	676.26	190.50
14	23 738.29	1 608.39	1 085.60

Entrada	Repeticiones Maximales Exclusivas	Repeticiones Supermax. Exclusivas	Repeticiones Supermax. de un Conjunto	Etiquetas Minimales
1	0.03	0.00	0.00	0.00
2	1.22	0.01	0.01	0.01
3	1.10	0.13	0.08	0.06
4	1.66	0.22	0.12	0.09
5	0.03	0.00	0.00	0.00
6	1.77	0.24	0.13	0.10
7	0.00	0.00	0.00	0.00
8	0.29	0.02	0.02	0.02
9	0.00	0.00	0.00	0.00
10	0.23	0.02	0.01	0.01
11	114.28	22.45	19.11	17.34
12	154.73	26.92	24.34	20.85
13	24.40	4.00	2.58	2.13
14	151.85	26.84	21.86	19.07

Cuadro 6.3: Tiempos de Ejecución de da Proceso (en segundos)

Especie	Nomenclatura binomial	Versión del Genoma
Humano	*Homo sapiens*	GRCh37
Chimpancé	*Pan troglodytes*	CHIMP2.1.52
Macaca	*Macaca mulatta*	MMUL_1.53
Vaca	*Bos taurus*	Btau_4.0.53
Perro	*Canis familiaris*	BROADD2.53
Caballo	*Equus caballus*	EquCab2.53
Ratón	*Mus musculus*	NCBIM37.53
Rata	*Rattus norvegicus*	RGSC3.4.53
Gallo	*Gallus gallus*	WASHUC2.53

Cuadro 6.4: Especies Analizadas

En adelante, trabajaremos con los conjuntos de cromosomas de cada especie excluyendo en general los cromosomas Y de cada una, ya que suelen ser un orden de magnitud menores al resto de los cromosomas. La inclusión de estos cromosomas no sería un problema técnico, pero se los omite para obtener resultados más significativos, especialmente en la búsqueda de repeticiones supermaximales de conjuntos y etiquetas minimales. Se reproducen a continuación resultados estadísticos, observaciones y conclusiones sobre los mismos.

6.2.1. Repeticiones Supermaximales en un Conjunto

Se computaron las repeticiones supermaximales de los siguientes conjuntos:

- Para las nueve especies anteriores, todos los cromosomas
- Todos los cromosomas de los mamíferos (humano, chimpancé, macaca, vaca, caballo, perro, rata, ratón)
- Todos los cromosomas de los primates (humano, chimpancé, macaca)
- Todos los cromosomas de los mamíferos domésticos (vaca, perro, caballo)
- Todos los cromosmas de los roedores (rata, ratón)

Se reproducen en el Cuadro 6.5 las longitudes de las repeticiones supermaximales más largas encontradas en cada conjunto. Naturalmente, la longitud de la repetición supermaximal más larga de cada conjunto de especies es siempre mayor que la longitud de la repetición supermaximal más larga de cada una de las especies que componen el conjunto.

Por ejemplo, puede verse que existe una secuencia en todos los cromosomas del humano (salvo el cromosoma Y) de 294 *bp* de longitud, mientras que existen fragmentos en común en todos los cromosomas de todos los primates de hasta 156 *bp*,

de elementos

Conjunto de cromosomas	Máxima repetición supermaximal
mamíferos	**47**
primates	**156**
humano	294
chimpancé	290
macaca	368
mamíferos domesticos	**47**
vaca	382
perro	481
caballo	365
roedores	**123**
ratón	2300
rata	1109
gallo	59

Cuadro 6.5: Longitud de las repeticiones supermaximales más largas

poco más de la mitad. En contraste, tanto en los mamíferos domésticos como en los roedores, la longitud de la mayor repetición supermaximal más larga de cada conjunto es alrededor de diez veces menor a la máxima repetición supermaximal de cada especie. Si estas relaciones se interpretasen como una medida de similitud entre especies, debería concluirse que las especies de primates analizadas son mucho más cercanas entre sí que los roedores. Tanto en los conjuntos de roedores como de mamíferos domésticos, todas las repeticiones supermaximales son *microsatélites*, fragmentos de repeticiones consecutivas o en forma de *tándem* de 1 a 6 *bp* que se encuentran en todos los organismos eucariotas en regiones generalmente no codificantes, mientras que la secuencia más larga en común de todos los primates corresponde a una parte de un retrotransposón L1. La repetición supermaximal más larga de los cromosomas del gallo no ocurre en ninguno de los cromosomas de ninguna de las ocho especies restantes.

Para ilustrar las variaciones de las repeticiones supermaximales en distintos conjuntos, se grafica en la Figura 6.1 la longitud de la mayor subcadena común que comienza en cada posición del cromosoma 1 del humano (es decir, el valor del arreglo N si se toma al cromosoma 1 del humano como base) en distintos conjuntos anidados: los cromosomas 1 del humano y su homólogo del chimpancé, éstos y el cromosoma 1 de la macaca, todos los cromosomas de los tres primates, todos los cromosomas de todos los mamíferos y todos los cromosmas de todos los vertebrados analizados. En el esquema, cada píxel representa aproximadamente 225 Kbp, de las que se tomó el mayor valor.

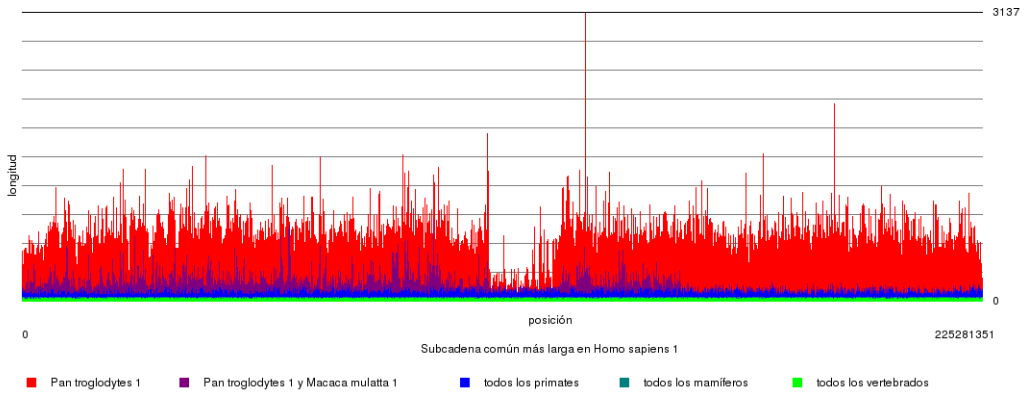

Figura 6.1: Longitud de la máxima subcadena común del primer cromosma del humano vs. distintos conjuntos

6.2.2. Repeticiones Maximales y Supermaximales Exclusivas

Se eligió el cromosoma 1 del humano para calcular sus repeticiones supermaximales exclusivas respecto de tres conjuntos: el de los cromosomas 1 del humano y su homólogo del chimpancé, el de todos los cromosomas de todos los primates, y el de todos los cromosomas de los 9 vertebrados. Se ilustran las cantidades obtenidas de cada longitud en la Figura 6.2, en escala logarítmica en ambos ejes.

En la figura puede verse que la mayor parte de las repeticiones exclusivas son cortas: menos de 15000 son mayores a 100 *bp* de las más de dos millones de repeticiones supermaximales exclusivas del cromosoma 1 del humano respecto del cromosoma 1 del chimpancé. De éstas, sólo 325 son mayores a 1000 *bp*. La cantidad de repeticiones supermaximales exclusivas respecto de todos los restantes cromosomas del humano y todos los de los primates es sustancialmente menor: hay alrededor de 130000 de las cuales 9500 tienen longitud mayor a 100 y las mismas 325 tienen longitud mayor a 1000. Al buscar repeticiones exclusivas respecto de todos los cromosomas de las 9 especies, las cantidades no se alteran significativamente. Esto sugiere que las repeticiones supermaximales del primer cromosoma del humano que no son exclusivas, muy probablemente aparecerán en algún otro cromosoma del humano o de algún primate. Para todos los cromosomas del humano se observa un comportamiento cuantitativo similar.

6.2.3. Etiquetas Minimales

Para probar el algoritmo de etiquetas minimales se seleccionaron cuatro fragmentos del cromosoma 1 del humano que conforman un conjunto de *duplicaciones segmentales*. Las duplicaciones segmentales son secuencias extensas de bases de alta similitud entre sí, y poca cantidad de ocurrencias en el genoma. Tienen la particularidad de formar un porcentaje significativo del genoma del humano, aunque son bastante más raras en otras especies de mamíferos. Las duplicaciones segmentales son de interés para el diagnóstico de enfermedades ya que de acuerdo a su ubicación

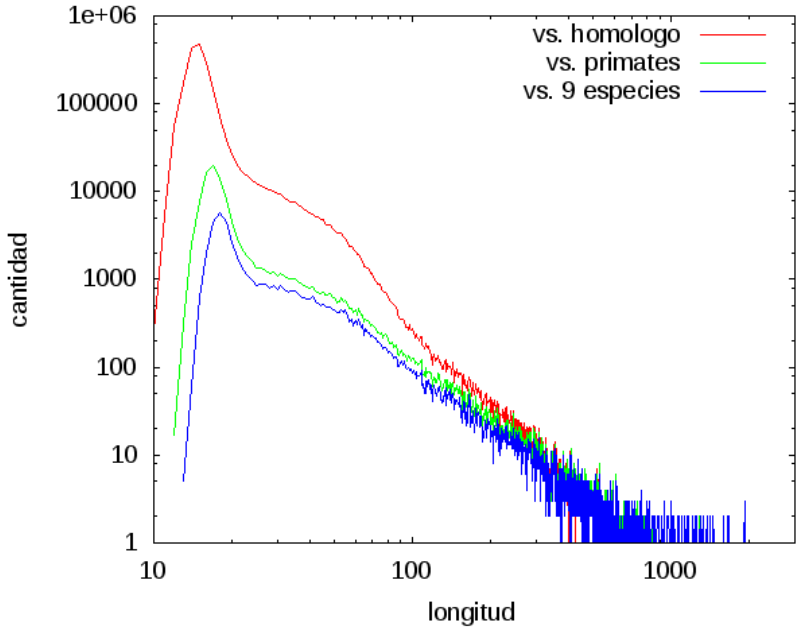

Figura 6.2: Cantidad de repeticiones supermaximales exclusivas del cromosoma 1 del humano respecto de distintos conjuntos

	Posición inicial	Posición final	Longitud (*bp*)
Copia nº 1	143644525	143771003	126479
Copia nº 2	144274483	144401745	127263
Copia nº 3	144451746	144526797	75052
Copia nº 4	147832033	147998780	166748

Cuadro 6.6: Conjunto de duplicaciones segmentales del cromosoma 1 del humano

y cantidad de ocurrencias, estas pueden dar lugar a afecciones genéticas. En este caso, se utilizó un conjunto de duplicaciones conformado por los segmentos indicados en el Cuadro 6.6

La suma de las longitudes de los segmentos no llega a las 500 *Kbp*, mientras que el resto del genoma supera las 252 *Mbp*. Estos segmentos, recortados del primer cromosma del humano, conforman el primer conjunto W del que se calculan las etiquetas minimales. El conjunto U de referencia está formado por el conjunto *diferencia* del cromosoma 1 respecto de los elementos de W, es decir, los cinco segmentos del cromosoma 1 que no forman parte de las duplicaciones segmentales.

A continuación, se computaron las etiquetas minimales del mismo conjunto W respecto de la unión del conjunto U con los restantes 22 cromosomas del humano. En el caso de los primeros conjuntos, se obtienen 8882 etiquetas minimales de longitudes entre 10 y 86 y una longitud promedio de las etiquetas de 14,9. Al agregar los

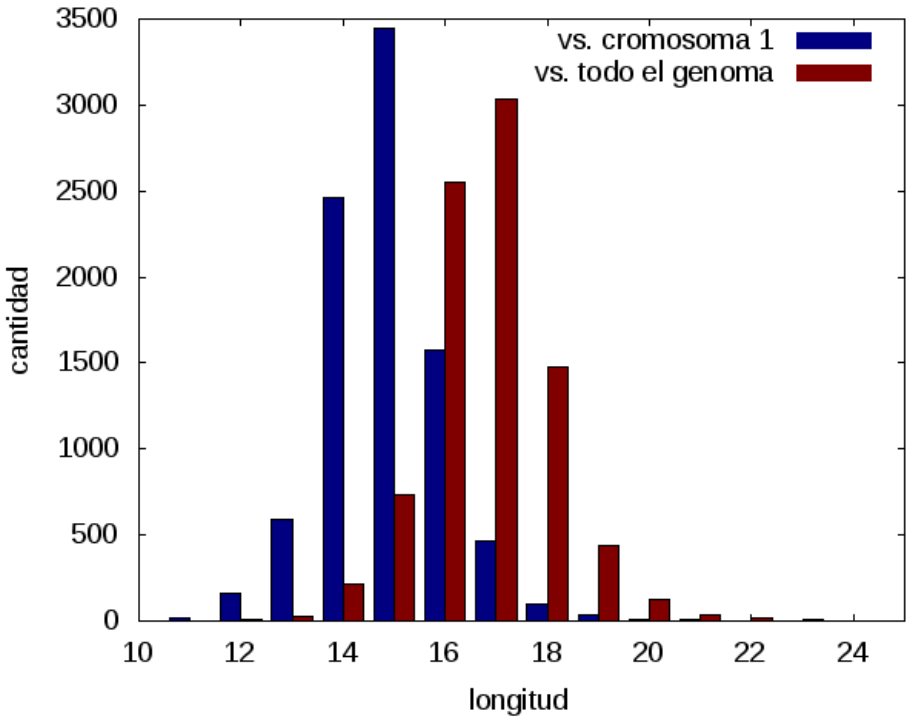

Figura 6.3: Cantidad de etiquetas minimales de un conjunto de duplicaciones segmentales

22 cromosomas al conjunto de referencia, se obtienen 8688 etiquetas minimales de longitudes entre 11 y 86, con longitud promedio 16,8. Las distribuciones de las longitudes se ilustran en la Figura 6.3 (no se grafican longitudes mayores a 24 ya que ninguna tiene más de 3 etiquetas, y no se llegan a distinguir en la escala).

Al comparar las etiquetas entre ambas corridas puede verse que el número de etiquetas no se redujo significativamente a pesar de que se agregaron secuencias sumando casi 3 *Gbp*. Las longitudes aumentan ligeramente: un promedio de menos de dos bases cada etiqueta. Analizando estos datos y el gráfico de distribución se puede presumir que las etiquetas en general se mantuvieron, aumentando en forma tenue su longitud, y posiblemente fundiéndose algunas etiquetas cercanas que tuviesen algo de superposición en una extensión de ambas. Esto indicaría que las etiquetas halladas efectivamente *identifican* el primer conjunto de duplicaciones segmentales, con lo que una búsqueda de alguna de las etiquetas en un cromosoma sería un indicador adecuado de la presencia de dicha duplicación segmental.

A modo de ejemplo final de posibles aplicaciones de este algoritmo, se ejecutó el algoritmo de etiquetas minimales a un conjunto de textos provenientes de libros de tres grandes autores de la literatura castellana, listados en el Cuadro 6.7.

Autor	Título	Tamaño (kB)
Jorge Luis Borges	El Aleph	233
	El Hacedor	83
	El Informe de Brodie	131
	El Libro de Arena	149
	Ficciones 1956	256
	Ficciones	267
	Historia Universal de la Infamia	121
	La Memoria de Shakespeare	55
	Seis Problemas para don Isidro Parodi	262
Julio Cortázar	62 Modelo para Armar	512
	Bestiario	187
	Fantomas contra las Multinacionales	54
	Historias de Cronopios y Famas	134
	Las Armas Secretas	294
	Rayuela	1022
	Todos los Fuegos el Fuego	288
	Un Tal Lucas	181
Gabriel García Márquez	Cien Años de Soledad	836
	Crónica de una Muerte Anunciada	173
	Doce Cuentos Peregrinos	302
	El Amor en los Tiempos del Cólera	846
	El Coronel no Tiene quien le Escriba	114
	El Otoño del Patriarca	509
	La Cándida Eréndira	92
	La Mala Hora	290
	Noticia de un Secuestro	578
	Ojos de Perro Azul	157
	Relato de un Náufrago	175
	Vivir para Contarla	1050

Cuadro 6.7: Libros para buscar etiquetas minimales

Los libros se agruparon por autor, considerando cada libro completo una única secuencia. A continuación, se corrió el algoritmo de etiquetas minimales para cada conjunto correspondiente a uno de los autores respecto de la unión de los otros conjuntos, es decir, se buscaron todas las palabras, partes de palabras o frases que ocurren al menos una vez en cada uno de los libros del autor, y en ninguno de los libros de los otros autores. Dado que los libros no forman parte de una historia común, las condiciones planteadas son bastante exigentes. Sin embargo, encontramos los siguientes resultados:

- Las etiquetas minimales de los libros de Borges son "`es fam`", "`Isl`", "`apócr`", "`l Fe`", "`mite l`" y "`ócrif`". Se comprueba que la palabra "`apócrifo`" ocurre en todos los libros citados de Borges en alguna de sus terminaciones, y en ninguno de los libros de los otros autores. La subcadena "`Isl`" aparece como prefijo de "`Islam`" e "`Islandia`".

- Las etiquetas minimales de los libros de Cortázar son "`de cuando en`", "`diota `", "`do en cuan`", "`e cuando en `", "`e telef`", "`está ta`", "`hicas`", "`ndo en cua`", "`o en cuand`", "`s chica`" y "`uando en c`". Se ven como características del estilo de Cortázar la construcción "`de cuando en cuando`", la referencia a "`chicas`" y la palabra "`idiota`".

- Las etiquetas minimales de los libros de García Márquez son "`Fue una`", "`a amarr`", "`marrado`" y "`ntensa`". Se encuentran formas del verbo "`amarrar`", poco utilizado por los autores argentinos, y la palabra "`intensa`", que más allá de su ocurrencia en todos los libros de García Márquez, puede resultar sorprendente su ausencia en los otros autores.

Capítulo 7

Conclusiones

Se propuso una familia de tres algoritmos para tratar las repeticiones en conjuntos de cadenas. La motivación de los mismos surge del análisis de genomas en bioinformática, aunque existen otros campos de aplicación donde los mismos pueden ser de utilidad. La principal ventaja de los métodos propuestos es la eficiencia obtenida en el compromiso de uso de recursos de tiempo y memoria, condición esencial para las aplicaciones que requieren procesar grandes volúmenes de datos como son las secuencias de ADN.

En particular, se destaca la propiedad de los métodos de requerir una cantidad de memoria proporcional a la más larga de las cadenas de los conjuntos, y de ser *independiente* de la cantidad de elementos de cada conjunto. Si todos los elementos procesados ocupan espacio menor a 1/17 de la memoria disponible, cualquiera de los algoritmos se podrá llevar a cabo en la memoria principal, siendo especialmente eficiente al permitirse prescindir del acceso a memoria secundaria (*swap* o memoria de intercambio). La originalidad de los algoritmos presentados reside en la capacidad de representar las repeticiones y conjuntos de subcadenas en forma eficiente a través de estructuras de datos diseñadas a tal fin.

Apéndice A

Implementaciones

A.1. Construcción del Arreglo de Sufijos

A.2. Máximo Prefijo Común

A.3. Algoritmo Básico

A.4. Repeticiones Maximales y Supermaximales Exclusivas y Supermaximales en un Conjunto

A.5. Etiquetas

Bibliografía

[Babenko 2008] Maxim Babenko and Tatiana Starikovskaya. *Computing Longest Common Substrings Via Suffix Arrays.* In CSR 2008, LNCS 5010, pages 64–75, Heidelberg, 2008. Springer-Verlag Berlin. 19

[Becher 2009] Verónica Becher, Alejandro Deymonnaz and Pablo Heiber. *Efficient repeat finding via suffix arrays.* submitted, 2009. 3, 15, 17, 22, 36, 37

[Bentley 1997] Jon L. Bentley and Robert Sedgewick. *Fast algorithms for sorting and searching strings.* In Proceedings of the eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 360–369, 1997. 11

[Brookfield 2005] John Brookfield. *The ecology of the genome - mobile DNA elements and their hosts.* Nat Rev Genet, vol. 6, pages 128–36, 2005. 2

[Ensembl 2009] Ensembl. *Ensembl*, 2009. http://nar.oxfordjournals.org/cgi/content/abstract/gkn828. 43

[Gusfield 1997] Dan Gusfield. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997. 6, 9

[Jurka 2005] Jurka, Kapitonov, Pavlicek, Klonowski, Kohany and Walichiewicz. *Repbase Update, a database of eukaryotic repetitive elements.* Cytogentic and Genome Research, vol. 110, pages 462–467, 2005. 3

[Kasai 2001] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa and Kunsoo Park. *Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications.* In Proc.12th Annual Symposium on Combinatorial Pattern Matching, pages 181–192, London, UK, 2001. Springer-Verlag. 10, 14, 15, 34

[Larsson 2007] N. Jesper Larsson and Kunihiko Sadakane. *Faster suffix sorting.* Theoretical Computer Science, vol. 387, no. 3, pages 258–272, 2007. 10, 13, 34

[Manber 1993] Udi Manber and Gene Myers. *Suffix arrays: a new method for on-line string searches.* SIAM Journal on Computing, vol. 22, no. 5, pages 935–948, 1993. SODA '90: Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms, 319–327, San Francisco, 1990. 12

[Puglisi 2007] Simon J. Puglisi, W. F. Smyth and Andrew H. Turpin. *A taxonomy of suffix array construction algorithms.* ACM Computing Surveys, vol. 39, no. 2, page 4, 2007. 10

[Venner 2009] Samuel Venner, Cédric Feschotte and Christian Biémont. *Dynamics of transposable elements: towards a community ecology of the genome.* Trends Genet, vol. 25, pages 317–323, 2009. 2

Resumen:
Palabras clave:

Abstract:
Keywords: