UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

FACULTAD DE
INGENIERÍA
UDELAR

Algoritmos eficientes para la construcción de conjuntos óptimos de contextos bi–direccionales

Fernando Arsenio Fernández Barreiro

Tesis de Maestría presentada a la Facultad de Ingeniería de la Universidad de la República en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Informática.

Tutor

Marcelo Weinberger

Tribunal

Gonzalo Navarro
Álvaro Martín
Pedro Piñeyro

Montevideo, Uruguay
octubre de 2018

Tesis de maestría
Algoritmos eficientes para la construcción de conjuntos óptimos de contextos bi–direccionales

Fernando Fernández Barreiro
fefernan@fing.edu.uy

Director de Tesis: Dr. Marcelo Weinberger
Center for Science of Information
marcwein@ieee.org

Director Académico: Dr. Alberto Pardo
Universidad de la República, Montevideo, Uruguay
pardo@fing.edu.uy

Octubre, 2018

Resumen

Resumen La técnica de modelados por contextos usada en problemas de decisión secuencial, como compresión de datos y otros, descompone una secuencia dada entre un conjunto de subsecuencias que son tratadas independientemente. La descomposición se basa en la ocurrencia de ciertas cadenas de símbolos que son elementos de un conjunto finito de contextos. Cada contexto tiene asociado un peso dado por una pérdida numérica generada al procesar la subsecuencia. Por tal motivo es necesario encontrar un conjunto de contextos óptimo (que minimice la pérdida total). En esta tesis los contextos están formados por pares de cadenas y se denominan contextos bi–direccionales.

La contribución de este trabajo es presentar estructuras de datos y algoritmos computacionalmente eficientes para encontrar un conjunto óptimo de contextos cuando la longitud de los contextos no está acotada.

Se demuestra que el conjunto de todos estos contextos bi–direccionales se puede partir en clases de equivalencia y alcanza con procesar un elemento representativo de cada clase.

La estructura presentada es una generalización de árboles de sufijos compactos. Es un grafo enraizado, dirigido, acíclico, donde cada vértice está etiquetado por el elemento representativo de una clase. Recíprocamente a cada clase le corresponde un vértice de este grafo. Las aristas dirigidas (al igual que en los árboles compactos de sufijos) están etiquetados por cadenas de símbolos, con las que se extiende el contexto representado por el vértice origen. Algunos vértices no tienen aristas salientes. Cada uno de los otros vértices tiene dos conjuntos de hijos (cada uno de los cuales extiende una de las direcciones de los contextos).

También se describe un algoritmo que reduce la complejidad de orden cúbico a orden cuadrático (en el largo de la cadena). Para ello se hace uso de árboles de sufijos y el cálculo del ancestro común más profundo de varios nodos de un árbol.

Palabras clave: Modelado por contexto, contextos bi-direccionales, árboles de contexto, compresión, predicción, eliminación de ruido, árbol de sufijos, ancestro común más profundo, grafos.

Índice general

Capítulo 1

Introducción

En esta tesis se presenta una estructura de datos para manejar contextos bidireccionales que es una generalización de los árboles de sufijos, y un algoritmo eficiente para construirla. Es una generalización de árboles, en donde cada nodo representa un contexto y tiene dos conjuntos de hijos que representan extensiones de sus contextos. Resulta ser un caso particular de grafo enraizado, dirigido y acíclico.

La motivación es contribuir en la técnica de *modelado por contextos* usada en problemas como *compresión*, *predicción* y eliminación de ruido (*denoising*). En estas aplicaciones se procesan secuencias (también llamadas cadenas) de símbolos y para cada posición de esas secuencias se tiene en cuenta sus *contextos*, que son subsecuencias que rodean a la posición. Una subsecuencia de una cadena es una secuencia formada por los símbolos que están en un conjunto de posiciones de la secuencia original. Por ejemplo, dada la secuencia `abcab`, la subsecuencia correspondiente a las posiciones $\{1, 4\}$ es `aa`. Una subcadena es una subsecuencia determinada por posiciones consecutivas de la secuencia original, por ejemplo `bca`.

La secuencia a procesar se descompone en un conjunto de subsecuencias que son tratadas independientemente. La descomposición se basa en la ocurrencia de contextos en las posiciones de la secuencia. Los contextos pertenecen a un conjunto finito que debe cumplir las propiedades de ser disjunto y exhaustivo, lo cual implica que a cada posición de la secuencia le corresponde exactamente un contexto del conjunto. El procesamiento de la secuencia asigna a cada una de las subsecuencias un costo (pérdida numérica) que está determinado por la cantidad de ocurencias de cada símbolo. Por extensión, al contexto correspondiente a la subsecuencia se le puede asociar ese costo. Esto motiva a encontrar un conjunto de contextos que sea óptimo (esto es, que minimice el costo total) entre aquellos que cumplan las propiedades requeridas.

Es posible que el contexto esté formado por varios componentes. Otro aspecto

que es de interés es la longitud de los contextos. Puede estar acotada o no y puede ser la misma en todos los contextos o no. Si los contextos tienen varios componentes se puede exigir que el largo de los componentes sea el mismo. En el caso tratado en este documento los contextos tienen dos componentes, las longitudes pueden ser diferentes entre contextos y entre componentes de un mismo contexto y no se acota la longitud que pueden tomar. Esto último genera un problema de complejidad computacional y es lo que aquí se resuelve. Para lograrlo lo que se va a hacer es particionar el conjunto de contextos posibles en clases, de tal manera que los elementos de la misma clase tengan propiedades idénticas. Y se va a procesar un único contexto de cada clase.

1.1. Motivación del problema

En el modelado por contextos [1, 2, 3] se procesa una secuencia descomponiéndola en subsecuencias y procesando cada subsecuencia de manera independiente. El procesamiento de cada subsecuencia genera un costo y se busca una descomposición que minimice el costo total, que es la suma de los costos por cada subsecuencia. Las subsecuencias se forman seleccionando posiciones que tienen un contexto en común. Como ejemplo supongamos que los contextos de una posición son las subsecuencias, leídas de derecha a izquierda, que la preceden. De manera provisoria, excluimos la posición 1, ya que no hay subsecuencias que la preceden. Para resolver esto, en la Subsección 3.1 se agregará un símbolo marcador en la posición 0, que no ocurre en el resto de la secuencia. Sea `abcdbccz` la secuencia procesada. Algunos contextos posibles para la cuarta posición son `cb` y `cba`. El primero de estos contextos también precede a la séptima posición. Por lo tanto la subsecuencia asociada a `cb` es la formada por los elementos que están en las posiciones 4 y 7, `dc`. Por otro lado, el contexto `cba` sólo precede la cuarta posición, por lo que le corresponde la subsecuencia `d`. Entonces, lo que hay que decidir es cómo se elige el conjunto de contextos que determinan las subsecuencias.

El conjunto debe satisfacer dos propiedades: *exhaustiva* y *disjunta*, (que se verán en el Capítulo 3), que son condición para que cada posición de la secuencia procesada corresponda a exactamente una de las subsecuencias. A un conjunto que cumpla esas propiedades se le dice *conjunto válido*.

Volviendo al ejemplo anterior, la secuencia `abcdbccz`, supongamos que `cba` es uno de los contextos. Precede a la cuarta posición pero no a la séptima. Para que sea un elemento de un conjunto válido tiene que haber un contexto que preceda a la séptima posición, y no puede ser `c` o `cb` porque estos preceden también a la cuarta posición. Un contexto posible es `cbd`, ya que no precede a ninguna otra posición de la secuencia.

Un par de ejemplos de conjuntos válidos para la secuencia `abcdbccz` son

{a, b, cba, cbd, cc, d} y {a, ba, bd, cb, cc, d}.

Estos son conjuntos válidos específicos de la secuencia tratada, relativos a ella. Se debe tener en cuenta que en la bibliografía lo habitual es definir los conjuntos válidos para *cualquier* secuencia. En el ejemplo, si el alfabeto fuese $\{a, b, c, d, z\}$ un posible conjunto válido de contextos sería {a, b, ca, cba, cbb, cbc, cbd, cbz, cc, cd, cz, d, z}. Con esta definición de validez cada posición de *cualquier* secuencia es precedida por algún contexto del conjunto (recordamos que la secuencia se lee de derecha a izquierda), aunque puede haber contextos que no precedan ninguna posición.

Entre los conjuntos válidos, aquellos que generan un conjunto de subsecuencias para los cuales el costo total es mínimo se llaman *conjuntos óptimos*. Como ejemplo ilustrativo, sin relación con una aplicacion real, supongamos que a las secuencias de longitud uno se asocia un costo de valor 2, mientras que a las secuencias de dos símbolos corresponde un costo de valor 3 si ambos símbolos son iguales y de valor 5 si los símbolos son diferentes. Como se vio, el contexto cb precede a las posiciones cuarta y séptima y determina la subsecuencia dc. El costo que corresponde es 5. Pero para cubrir las mismas posiciones también se pueden elegir los contextos cba y cbd, los cuales determinan subsecuencias de longitud 1, por lo que el costo total asociado a esas posiciones sería 4. Entonces, un conjunto válido que incluya el contexto cb no puede ser óptimo, ya que "partiendo" ese contexto en el par {cba, cbd} se obtiene un conjunto con el que se asocia un costo menor. Lo opuesto sucede al considerar las posiciones tercera y sexta. Se puede elegir los contextos ba y bd que determinan subsecuencias de longitud 1 y, por lo tanto, con un costo asociado de valor 4. Pero también se puede elegir el contexto b. La subsecuencia determinada es cc, y el costo es 3. De donde, si el conjunto válido contiene el par {ba, bd}, es conveniente "unificar" esos contextos en el contexto b. El costo de {a, b, cba, cbd, cc, d} es 13 y el de {a, ba, bd, cb, cc, d} es 15.

Es conocido que un conjunto válido se puede representar con las hojas de un árbol m-ario siendo m la cantidad de símbolos del alfabeto sobre el que se define la secuencia procesada [2]. Si la longitud de los contextos es acotada por k, se conocen algoritmos que, podando el árbol m-ario perfecto de altura k, obtienen un árbol que representa un conjunto óptimo. A estos árboles se les denomina *árboles óptimos*. Si no se acota la longitud de los contextos, se conoce un algoritmo que también permite encontrar eficientemente un conjunto óptimo [4].

Se ha formalizado la definición de conjunto válido cuando los contextos están formados por pares de secuencias, no necesariamente de la misma longitud, dando la generalización de las propiedades exhaustiva y disjunta [5, 6, 7]. Se demostró que un conjunto válido en el llamado caso *bi–direccional* también puede representarse con las hojas de un árbol m-ario. Y se describió un algoritmo que encuentra un conjunto óptimo si se mantiene acotada la longitud de los contextos. Este es

un algoritmo que empieza en los contextos más largos y mediante programación dinámica va podando una estructura que está implícita y es un grafo.

La demostración de que en el caso bi–direccional un conjunto válido se representa mediante hojas de un árbol no es una simple formalización que generaliza el caso uni–direccional, ya que se ha demostrado que esto mismo no se cumple cuando la cantidad de componentes de los contextos es mayor a 2 [6, 7].

1.2. Definición del problema

En principio se puede suponer que establecer un límite en la longitud de los contextos reduce la información que se obtiene de la secuencia. Lo que acá nos proponemos es eliminar la cota en la longitud de los contextos. En realidad hay una cota implícita que es la longitud de la secuencia a procesar. Pero aplicar el algoritmo mencionado con esa cota implica complejidad de orden cúbico en la longitud de la secuencia. Entonces, el propósito de este trabajo es diseñar un algoritmo que encuentre un conjunto óptimo de contextos bi–direccionales cuando no se acota la longitud de los contextos, y que en el peor caso tenga complejidad polinómica de orden cuadrático.

La solución pasa por observar que para algunos contextos la información relevante al problema es la misma que para otros. Entonces, para cada conjunto de contextos con idéntica información, lo que se debe hacer es encontrar sólo uno de ellos, que los represente a todos. Lo que sucede con cada uno de los contextos de ese conjunto es que en cada posición de la secuencia en que ocurre, también ocurre su *contexto representativo*, del cual es *prefijo*. Esto se desarrolla en el Capítulo 4.

Se propondrá un algoritmo que encuentre estos contextos representativos. Además, se demostrará que cada contexto que ocurre en la secuencia de entrada está representado por uno de los contextos encontrados por el algoritmo. Parte del trabajo consiste en volver a definir árboles y árboles óptimos en los que los nodos no sean todos los contextos posibles, ni siquiera todos los que ocurren, sino sólo aquellos que representan a otros. También es necesario definir nuevas estructuras, grafos compactos con raíz, dirigidos y acíclicos, cuyos nodos corresponden a esos contextos representativos.

Tal como en la versión uni–direccional [7], el algoritmo empieza con los contextos de longitud máxima. Los contextos están formados por un par de componentes, que llamamos izquierdo y derecho. Todos los contextos izquierdos (derechos) son subcadenas de una cadena denominada *cadena de formación de contextos izquierdos (derechos)*. A cada posición de la secuencia procesada corresponde un sufijo de esta cadena de formación de contextos. Cada sufijo es la cadena formadora de contextos de su posición correspondiente. Los contextos son los prefijos de estos sufijos. Los contextos bi–direccionales de longitud máxima

son los que alcanzan el final de la cadena de formación de contextos correspondiente en cada una de los dos componentes. A los contextos de longitud máxima les llamamos *bi–sufijos*, porque sus dos componentes son sufijos de las cadenas de formación de contextos. La información relevante de un contexto, mencionada anteriormente, está determinada por los bi–sufijos de los cuales es prefijo. Hay una correspondencia entre los bi–sufijos y las posiciones de la secuencia de entrada, ya que cada posición define un contexto de longitud máxima.

En el caso uni–direccional un contexto se dice *ramificable* si ocurre en la cadena de formación de contextos al menos dos veces y el símbolo siguiente es diferente en ambas ocurrencias. Al plantear el ejemplo de la secuencia `abcdbccz` se vio que el contexto `cb` ocurre dos veces. En ese ejemplo la cadena de formación de contextos es `ccbdcba`. En esta cadena `cb` es ramificable porque una de las veces que ocurre es prefijo de `cba` y la otra de `cbd`. En el árbol de contextos que corresponde a la secuencia, hay un nodo que representa el contexto `cb` del cual salen dos ramas. En el caso bi–direccional le diremos *doblemente ramificable* o *bi–ramificable* si es ramificable en ambos direcciones. Veremos que los contextos representativos son los bi–sufijos y los bi–ramificables.

Mientras que es fácil encontrar los bi–sufijos, la complejidad del algoritmo aquí desarrollado consiste en encontrar los bi–ramificables. Todos los contextos izquierdos se pueden representar con un *árbol de sufijos*, y los contextos derechos con otro. Los contextos bi–direccionales representativos son un subconjunto del producto cartesiano de los nodos de esos árboles. Entre las hojas de ambos árboles hay una correspondencia uno a uno que determina los bi–sufijos y se obtiene de manera trivial. Empezando con esos contextos de longitud máxima se recorre desde las hojas hacia la raíz uno de los árboles, encontrando para cada nodo del árbol recorrido aquellos nodos del otro árbol con los que forma un contexto representativo. La principal operación que se usa es el cálculo del prefijo común más largo de varias secuencias. El costo de esta operación es proporcional a la longitud de las cadenas. Pero, si las cadenas son nodos de un árbol, como sucede en este caso, esa operación se puede transformar en la obtención del ancestro común más profundo o `lca` (por su denominación en inglés “lowest common ancestor”), que se puede hacer en tiempo constante después de un preprocesamiento del árbol.

Además de encontrar los nodos que corresponden a los contextos representativos hay que estructurarlos según relaciones padre-hijo, porque el objetivo es podar la estructura construida para obtener un árbol óptimo que está contenido en ella.

Las contribuciones fundamenteales de esta tesis fueron presentadas en seminarios y conferencias [8, 9].

En el Capítulo 2 se definen conceptos básicos que se usarán en el resto del documento. En el Capítulo 3 se describe el problema cuando se fija una cota a la longitud de los contextos, tanto en el caso uni como en el bi-direccional. En el Capítulo 4 se trata el caso no acotado. Los algoritmos y las estructuras se describen en el Capítulo 6. Previamente, en el Capítulo 5 se describe la versión uni–direccional del algoritmo principal. En el Apéndice A se describe la forma de resolver el cálculo del LCA.

Capítulo 2

Definiciones fundamentales

En este capítulo se presentan la definición y la notación de los principales conceptos con los que se tratará en el resto del documento.

Cadenas

A lo largo de todo el documento usaremos un *alfabeto finito* $\mathcal{A}$. Con $\mathcal{A}^*$ representamos el conjunto de todas las cadenas finitas de símbolos de $\mathcal{A}$, incluyendo la cadena de longitud nula, denotada por ϵ. Con $\mathcal{A}^+$ se simboliza el conjunto de cadenas finitas de longitud no nula. $\mathcal{A}^k$ es el conjunto de cadenas de longitud k y $\mathcal{A}^{\geq k}$ el de cadenas de longitud mayor o igual a k.

En algunas ocasiones se trabaja con pares ordenados de cadenas finitas. En general las dos cadenas del par no tienen por qué tener la misma longitud. Al primer componente le llamamos izquierdo y al segundo derecho. En este caso se dice que hay dos direcciones, izquierda y derecha, y lo definimos como caso *bi–direccional*. Por compatibilidad con esto, al caso simple le decimos también *uni–direccional*. Si el par de cadenas es (ϵ, ϵ), con un abuso de notación y asumiendo que no hay ambigüedad, lo denotaremos también por ϵ.

Siempre supondremos que procesamos una *secuencia* o *cadena* $\mathbf{x}$ de longitud n, $(x_1, x_2, \ldots, x_n)$. En el caso uni–direccional los elementos de $\mathbf{x}$ pertenecen a $\mathcal{A}$. En el caso bi–direccional hay aplicaciones, como compresión de señales de audio que provienen de dos canales, en que pertenecen a $\mathcal{A} \times \mathcal{A}$. Pero en otras, por ejemplo la eliminación de ruido, son elementos de $\mathcal{A}$; a pesar de eso corresponden al caso bi–direccional ya que para cada posición se tienen en cuenta dos conjuntos de símbolos, los que están antes y los que están después. Generalizando, le llamamos $\mathcal{X}$ al conjunto al que pertenecen los elementos de $\mathbf{x}$, ya sea $\mathcal{A}$ o $\mathcal{A} \times \mathcal{A}$. La secuencia $\mathbf{x}$ pertenece a $\mathcal{X}^*$, conjunto de secuencias finitas de elementos de $\mathcal{X}$. Además definimos $\mathcal{X}^\star$ para denotar el conjunto de contextos. En el caso uni–direccional es igual a $\mathcal{A}^*$ y a $\mathcal{X}^*$. Pero en el caso bi–direccional es $\mathcal{A}^* \times \mathcal{A}^*$, tanto para las aplicaciones en que $\mathcal{X}$ es $\mathcal{A}$ como para aquellas en las que $\mathcal{X}$ es $\mathcal{A} \times \mathcal{A}$.

Cada elemento de $\mathbf{x}$ está en una posición o *índice* perteneciente a $\{1, \ldots, n\}$. Dada una secuencia de índices $\mathcal{I} = (i_1, i_2, \ldots, i_h)$, con $1 \leq i_1 < i_2 < \cdots < i_h \leq n$, la *subsecuencia* de $\mathbf{x}$ indizada por $\mathcal{I}$ es la secuencia formada por los elementos de $\mathbf{x}$ que están en las posiciones de $\mathcal{I}$, esto es $(x_{i_1}, x_{i_2}, \ldots, x_{i_h})$.

Ejemplo 2.1. Supongamos que $\mathcal{A} = \{a, b, c, d, e\}$.

Sea la cadena `abcdeabcd`. Al conjunto índice $\mathcal{I} = \{2, 4\}$ corresponde la subsecuencia `bd`.

Para el caso bi–direccional en que los elementos de $\mathbf{x}$ pertenecen a $\mathcal{A} \times \mathcal{A}$, sea $\mathbf{x} = \binom{\texttt{abcdeabcd}}{\texttt{dcbaedcba}}$. Con esta notación en la segunda posición de $\mathbf{x}$ está el símbolo $(\texttt{b}, \texttt{c})$ de $\mathcal{X}$. A $\mathcal{I} = \{2, 4\}$ corresponde $\binom{\texttt{bd}}{\texttt{ca}}$.

Una *subcadena* es una subsecuencia definida por un conjunto cuyas posiciones son consecutivas: $(x_i, x_{i+1}, \ldots, x_j)$, con $i, j \in \{1, \ldots, n\}$. Esta subcadena se denota por $\mathbf{x}_i^j$. Si $i = j$ la subcadena tiene longitud 1. Si $i > j$ se considera que $\mathbf{x}_i^j$ es ϵ, la cadena vacía.

Para cualquier cadena $\mathbf{s}$ su longitud se denota con $|\mathbf{s}|$. El *reverso* de la cadena $\mathbf{s}$ es la misma cadena pero recorrida de derecha a izquierda y se denota por $\overline{\mathbf{s}}$. Si la secuencia es una subcadena $\mathbf{x}_i^j$, su reverso es $(x_j, x_{j-1}, \ldots, x_i)$, y usaremos la notación ${}_j^i\mathbf{x}$. En este caso la cadena es vacía si el superíndice es mayor que el subíndice.

Sean $\mathbf{s}, \mathbf{t}, \mathbf{u} \in \mathcal{A}^*$, con $\mathbf{s} = (s_1, s_2, \ldots, s_h)$, y $\mathbf{t} = (t_1, t_2, \ldots, t_k)$. Se denota $\mathbf{u} = \mathbf{st}$ a la *concatenación* de $\mathbf{s}$ y $\mathbf{t}$ definida por $\mathbf{u} = (s_1, s_2, \ldots, s_h, t_1, t_2, \ldots, t_k)$. Se dice que $\mathbf{s}$ es *prefijo* de $\mathbf{u}$, lo que se denota por $\mathbf{s} \preceq \mathbf{u}$. Si $\mathbf{t} \neq \epsilon$ las cadenas $\mathbf{s}$ y $\mathbf{u}$ son diferentes, y la relación se denomina *estricta*, y se denota por $\mathbf{s} \prec \mathbf{u}$. La cadena $\mathbf{t}$ es un *sufijo* de $\mathbf{u}$, y es sufijo estricto si $\mathbf{s} \neq \epsilon$. La cadena ϵ, es prefijo y sufijo de cualquier cadena. Las relaciones prefijo y sufijo definen un orden parcial entre las cadenas de $\mathcal{A}^*$.

La notación se extiende a conjuntos. Si $\mathcal{C}$ es un conjunto de cadenas, entonces $\mathbf{s} \preceq \mathcal{C}$ significa que $\mathbf{s}$ es prefijo de cada cadena de $\mathcal{C}$. Lo mismo vale para la relación estricta.

Las relaciones prefijo y sufijo se extienden a elementos de $\mathcal{X}^\star$. Sean $\mathbf{s} = (\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ y $\mathbf{t} = (\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$. Entonces $\mathbf{s}$ es prefijo de $\mathbf{t}$ si y sólo si $\mathbf{s}_\ell \preceq \mathbf{t}_\ell$ y $\mathbf{s}_\mathbf{r} \preceq \mathbf{t}_\mathbf{r}$.

En esta tesis, por comodidad en el lenguaje, tanto para el caso uni–direccional como bi–direccional, si $\mathbf{s}$ es prefijo de $\mathbf{t}$, decimos también que $\mathbf{t}$ es *extensión* de $\mathbf{s}$. Entonces en lugar de referirnos al "conjunto de cadenas (o pares de cadenas) de las que $\mathbf{s}$ es prefijo", nos referimos a "las extensiones de $\mathbf{s}$". Ese conjunto de extensiones lo denotamos con $\mathcal{P}(\mathbf{s})$. O sea

$$\mathcal{P}(\mathbf{s}) = \{\mathbf{t} \in \mathcal{X}^\star \colon \mathbf{s} \preceq \mathbf{t}\}. \tag{2.1}$$

También nos interesará considerar las extensiones de longitud igual a algún parámetro preestablecido. Para el caso bi–direccional, sean k_ℓ, k_r enteros no negativos, y $(\mathbf{s}_\ell, \mathbf{s_r})$ un elemento de $\mathcal{X}^\star$ que cumple $|\mathbf{s}_\ell| \leq k_\ell$ y $|\mathbf{s_r}| \leq k_r$. Denotamos por $\mathcal{P}_{(k_\ell,k_r)}(\mathbf{s}_\ell, \mathbf{s_r})$ a las extensiones de $(\mathbf{s}_\ell, \mathbf{s_r})$ tal que su componente izquierdo (resp. derecho) tiene longitud igual a k_ℓ (resp. k_r). Formalmente:

$$\mathcal{P}_{(k_\ell,k_r)}(\mathbf{s}_\ell, \mathbf{s_r}) = \left\{(\mathbf{t}_\ell, \mathbf{t_r}) \in \mathcal{A}^{k_\ell} \times \mathcal{A}^{k_r} \colon (\mathbf{s}_\ell, \mathbf{s_r}) \preceq (\mathbf{t}_\ell, \mathbf{t_r})\right\}. \tag{2.2}$$

El caso uni–direccional es una sencilla simplificación de la Expresión (2.2).

Las definiciones se extienden fácilmente a conjuntos de cadenas: una cadena es extensión de un conjunto de cadenas si es extensión de cada cadena del conjunto.

Definimos *extensiones mínimas* de $\mathbf{s} \in \mathcal{A}^*$ a las cadenas $\mathbf{s}a$ con $a \in \mathcal{A}$. Para el caso bi–direccional, las *extensiones mínimas por izquierda (resp. derecha)* de $(\mathbf{s}_\ell, \mathbf{s_r})$ son las cadenas $(\mathbf{s}_\ell a, \mathbf{s_r})$ (resp. $(\mathbf{s}_\ell, \mathbf{s_r}a)$) con $a \in \mathcal{A}$.

Si dadas dos cadenas, ninguna es prefijo de la otra, sus conjuntos de extensiones son disjuntos. Diremos que las dos cadenas son *disjuntas*.

Contextos

El problema a tratar en esta tesis incluye descomponer una secuencia $\mathbf{x}$ en un conjunto finito de subsecuencias, de tal manera que cada posición de $\mathbf{x}$ sea índice de exactamente una de las subsecuencias. Es decir, se trata de hacer una partición de $\{1, \ldots, n\}$ en clases, cada una de las cuales es el conjunto que indiza una subsecuencia.

Ejemplo 2.2. Sea, otra vez, la cadena $\mathbf{x} =$ `abcdeabcd`. Los conjuntos de índices $\{2, 4\}$, $\{1, 6, 7, 8\}$ y $\{3, 5, 9\}$ constituyen una partición de $\{1, \ldots, 9\}$. Con esta partición, $\mathbf{x}$ es descompuesta en `bd`, `aabc`, `ced`.

Una manera de formar las subsecuencias es utilizar contextos. De manera informal, un *contexto* de una posición de $\mathbf{x}$ es una secuencia de elementos que están en torno a la posición. La definición se hará rigurosa al precisar el significado de "estar en torno". Los contextos pueden estar constituidos por una sola cadena, o por dos, y en este caso los llamamos también *contextos bi–direccionales* o *bi–contextos*. En un bi–contexto a una de las cadenas la llamamos *contexto izquierdo* y a la otra *contexto derecho*. A los contextos de una sola cadena les llamamos también *simples* o uni–direccionales.

Dada la cadena $\mathbf{x}$, para cada posición en $\{1, \ldots, n\}$ se define su o sus *cadenas de formación de contextos*, que son cadenas de símbolos de $\mathcal{A}$. Los contextos de la posición son subsecuencias de la cadena de formación de contextos. En el caso de bi–contextos habrá dos cadenas de formación de contextos: *cadena de formación de contextos izquierdos*, y *cadena de formación de contextos derechos*. En esta

tesis los contextos de una posición dada serán los prefijos de su correspondiente cadena de formación de contextos. En general, las cadenas de formación de contextos serán subcadenas que preceden o suceden a la posición considerada. O sea, para la posición i pueden ser, si $\mathbf{x}$ pertenece a $\mathcal{A}^*$, las subcadenas ${}_{i-1}^{1}\mathbf{x}$ y $\mathbf{x}_{i+1}^{n}$. Cuando $\mathbf{x}$ pertenece a $(\mathcal{A} \times \mathcal{A})^*$, $\mathbf{x} = (\mathbf{x}_\ell, \mathbf{x}_r)$, pueden ser ${}_{i-1}^{1}[\mathbf{x}_\ell]$ y ${}_{i-1}^{1}[\mathbf{x}_r]$.

Por lo tanto, las cadenas de formación de contextos para cada posición son sufijos de una única cadena, ya sea ${}_{n-1}^{1}\mathbf{x}$ o $\mathbf{x}_2^n$. Entonces, en el caso uni–direccional tenemos una única cadena de formación de contextos que denotamos por $\mathbf{y}$, siendo $\mathbf{y} = {}_{n-1}^{1}\mathbf{x}$. En el caso bi–direccional hay una cadena de formación de contextos izquierdos, $\mathbf{y}_\ell$, y una cadena de formación de contextos derechos, $\mathbf{y}_\mathbf{r}$. Cada contexto, ya sea simple, izquierdo o derecho, es una subcadena de su respectiva cadena de formación.

Ejemplo 2.3. Sea $\mathbf{x}$ la cadena `abcdeabcd`. Una posible cadena de formación de contextos de la posición 6 es `edcba`, es decir la subcadena de $\mathbf{x}$ desde la posición 5 (=$6 - 1$) hasta la posición 1. La cadena de formación de contextos de $\mathbf{x}$ es $\mathbf{y} =$ `cbaedcba`, de la cual `edcba` es sufijo. Algunos contextos para la posición 6 son ϵ, `ed`, `edcba`, todos ellos prefijos de `edcba`.

Otra posible cadena de formación de contextos de la posición 6 es `bcd`, la subcadena de $\mathbf{x}$ desde 7 (=$6+1$) hasta $n = 9$, que es sufijo de `bcdeabcd`. En el caso bi–direccional esta cadena es $\mathbf{y}_\mathbf{r}$, la cadena de formación de contextos derechos de $\mathbf{x}$, mientras que $\mathbf{y}_\ell$ es `cbaedcba`. Un posible bi–contexto de la posición 6 es (ed, bcd).

Aquí se justifica la intuición de hablar de direcciones izquierda y derecha: los contextos izquierdos (resp. derechos) corresponden a cadenas que están a la izquierda (resp. derecha) de la posición.

Ejemplo 2.4. Como ejemplo de secuencia $\mathbf{x}$ perteneciente a $\mathcal{A}^* \times \mathcal{A}^*$ tomamos $\binom{\texttt{abcdeabcd}}{\texttt{dcbaedcba}}$. Para la posición 6 los componentes de ${}_{6-1}^{1}\mathbf{x}$ son `edcba` y `eabcd`. Estas son las cadenas de formación de contextos para la posición 6. Las cadenas de formación de contextos izquierdo y derecho de $\mathbf{x}$ son $\mathbf{y}_\ell =$ `cbaedcba` e $\mathbf{y}_\mathbf{r} =$ `bcdeabcd`.

Conjunto válido y conjunto óptimo

Sea una cadena $\mathbf{x}$ a procesar y $\mathbf{s}$ un contexto. Definimos el *conjunto índice* de $\mathbf{s}$ y lo denotamos por $\mathcal{I}(\mathbf{s})$ al conjunto de posiciones de $\mathbf{x}$ de las cuales $\mathbf{s}$ es contexto.

Un *conjunto válido de contextos*, $\mathcal{V}$, (relativo a una secuencia $\mathbf{x}$ dada) es un conjunto de contextos tales que sus conjuntos índices determinan una partición de $\{1, \ldots, n\}$. Esto es, $\{\mathcal{I}(\mathbf{s}) \colon \mathbf{s} \in \mathcal{V}\}$ es una partición de $\{1, \ldots, n\}$.

Es posible que para una secuencia $\mathbf{x}$ haya más de un conjunto válido de contextos. Cada contexto determina una subsecuencia, y con cada subsecuencia se asocia un valor numérico que representa un costo, que queda asociado a dicho contexto. Se quiere elegir un conjunto válido de contextos, relativo a la secuencia $\mathbf{x}$, de tal manera que resulte minimizada la suma de los costos de las subsecuencias.

La subsecuencia indizada por $\mathcal{I}(\mathbf{s})$ se denota por $\mathbf{x}(\mathbf{s})$. Para establecer el costo asociado a la subsecuencia es de interés conocer la cantidad de veces que cada elemento de $\mathcal{X}$ está en $\mathbf{x}(\mathbf{s})$. Esta información se mantiene con un *vector de conteos*, al que denotamos por $\mathbf{c}_{\mathbf{x}}(\mathbf{s})$, o $\mathbf{c}(\mathbf{s})$, si asumimos que está implícita la secuencia $\mathbf{x}$. O sea, para $a \in \mathcal{X}$

$$\mathbf{c}(\mathbf{s})[a] = |\{i \in \mathcal{I}(\mathbf{s}) \colon \mathbf{x}_i = a\}| \,. \tag{2.3}$$

Ejemplo 2.5. Sea `abbababbc` la secuencia $\mathbf{x}$ procesada. Si el contexto $\mathbf{s}$ es `ba`, entonces $\mathcal{I}(\mathbf{s}) = \{3, 6, 8\}$, y $\mathbf{x}(\mathbf{s}) =$ `bab`. Por lo tanto $\mathbf{c}(\mathbf{s})[a] = 1$, $\mathbf{c}(\mathbf{s})[b] = 2$ y $\mathbf{c}(\mathbf{s})[c] = 0$.

Suponemos que existe una función que a cada vector de conteos le asigna un valor numérico no negativo. Como el vector de conteos está determinado por una subsecuencia, que a su vez es generada por un contexto, podemos extender la definición de esa función al dominio de los contextos. Entonces, existe una función *peso*, $\mathtt{w}_{\mathbf{x}}$, que a cada contexto asigna un valor numérico, real para mayor generalidad, no negativo. Asumiendo que está implícita la secuencia procesada queda definida la función $\mathtt{w} \colon \mathcal{X}^\star \to \mathbb{R}^{\geq 0}$. Se extiende la definición de esta función a un conjunto de contextos: el peso de un conjunto de contextos es la suma de los pesos de los contextos del conjunto. Esto vale en particular para los conjuntos válidos. Sea $\mathcal{V}$ un conjunto válido de contextos relativo a $\mathbf{x}$. Entonces

$$\mathtt{w}(\mathcal{V}) = \sum_{\mathbf{s} \in \mathcal{V}} \mathtt{w}(\mathbf{s}). \tag{2.4}$$

Un *conjunto óptimo de contextos* relativo a $\mathbf{x}$ es un conjunto válido cuyo peso es menor o igual al peso de cualquier otro conjunto válido.

Árboles

Llamamos *árbol m–ario* a un árbol enraizado en que cada nodo interno tiene exactamente m hijos. Si $\mathcal{A}$ tiene m símbolos, entonces para cada nodo interno se puede etiquetar cada rama saliente del nodo interno con un símbolo diferente de $\mathcal{A}$. A cada nodo se le puede asociar una cadena de $\mathcal{A}^*$. Es la cadena que se forma concatenando las etiquetas de las ramas del camino que va desde la raíz hasta el propio nodo. A la raíz corresponde ϵ. Un árbol m–ario es *perfecto* si todas

las hojas están en el mismo nivel. Si la altura de un árbol $|\mathcal{A}|$–ario perfecto es k, entonces a sus $|\mathcal{A}|^k$ hojas se las puede poner en correspondencia con las cadenas de $\mathcal{A}^k$.

El procedimiento anterior de etiquetar ramas y nodos se puede generalizar. Las ramas se etiquetan con elementos de $\mathcal{A}^+$. Si a es el primer elemento de la etiqueta se dice que se trata de una a–rama. Un *$\mathcal{A}^+$–árbol* es un árbol enraizado cuyas ramas están etiquetadas con elementos de $\mathcal{A}^+$ y tal que cualquier nodo tiene a lo sumo una a–rama saliente para cada $a \in \mathcal{A}$. En general no son árboles $|\mathcal{A}|$–arios porque para cualquier nodo interno no necesariamente tiene que existir una a–rama para todo $a \in \mathcal{A}$. También en este caso los nodos se etiquetan con la concatenación de las etiquetas de las ramas del camino desde la raíz al nodo, con ϵ como etiqueta de la raíz. Por simplicidad, con un abuso de lenguaje, identificaremos al nodo con su etiqueta. La rama que va desde el nodo $\mathbf{s}$ al nodo $\mathbf{t}$ se denota por $\overrightarrow{\mathbf{st}}$.

Sea T un $\mathcal{A}^+$–árbol y $\mathcal{S}$ el conjunto de cadenas representadas por las hojas de T. Una cadena $\mathbf{w}$ es una palabra de $\mathcal{T}$ si $\mathbf{w} \preceq \mathbf{s}$ para algún $\mathbf{s} \in \mathcal{S}$.

Un *árbol de sufijos* $T_{\mathbf{x}}$ para una secuencia $\mathbf{x}$ es un $\mathcal{A}^+$–árbol cuyas hojas son los sufijos de $\mathbf{x}$. Cualquier subcadena de $\mathbf{x}$ es prefijo de un sufijo de $\mathbf{x}$ y por lo tanto es una palabra de $T_{\mathbf{x}}$.

En la Figura 2.1 hay un ejemplo de árbol de sufijos. La existencia de un árbol de sufijos está garantizada si el símbolo $\mathbf{x}_n$ no aparece en $\mathbf{x}_1^{n-1}$. De otro modo hay sufijos de $\mathbf{x}$ que son prefijos de otros sufijos de $\mathbf{x}$, y por lo tanto no es posible la correspondencia uno a uno entre sufijos y hojas. El problema se evita asumiendo que toda secuencia termina con una marca de finalización, $\#$, que no ocurre en las posiciones previas.

Un *árbol compacto de sufijos* es un árbol de sufijos cuyos nodos internos tienen más de un hijo. En esta tesis árbol de sufijos significa árbol compacto de sufijos.

Se conocen métodos para construcción de árboles de sufijo en tiempo lineal [10, 11, 12, 13].

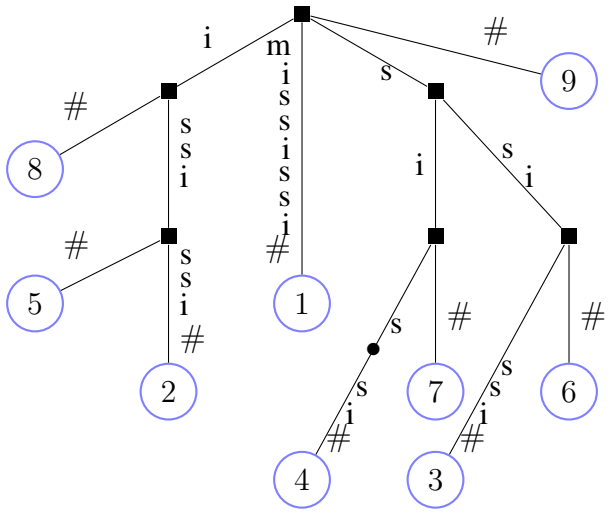


Figura 2.1: Árbol de sufijos para la secuencia '$mississi\#$'. Hay una hoja para cada sufijo de la secuencia. La concatenación de las etiquetas de las ramas en el camino desde la raíz a la hoja es igual al sufijo. Por ejemplo, el tercer sufijo es $ssissi\#$ y es la concatenación de s, si y $ssi\#$, el camino desde la raíz a la hoja 3. Cada etiqueta es una subcadena de la secuencia. Sin la marca de finalización '#' el quinto sufijo sería $issi$ y no habría una hoja para él, porque sería prefijo de otro sufijo, el segundo, $ississi$. Lo mismo sucede con los sufijos ssi y si. El octavo sufijo sería i, prefijo del segundo y quinto sufijos. Las ramas cuyas etiquetas tienen longitud mayor que 1 pueden ser fragmentadas. Por ejemplo, la rama entrante en la hoja 4 puede fragmentarse en s y $si\#$ creando un nuevo nodo. Pero este nuevo nodo tendría una única rama saliente y el árbol dejaría de ser compacto.

Capítulo 3

Fundamentos sobre conjuntos óptimos de contextos

En este capítulo se considera el problema de encontrar conjuntos óptimos de contextos de una secuencia $\mathbf{x}$ dada cuando la longitud de los contextos está acotada por valores predeterminados. En la Sección 3.1 se trata el caso uni–direccional y en la Sección 3.2 el caso bi–direccional. En el segundo caso se muestra una aplicación en la que los elementos de la secuencia $\mathbf{x}$ pertenecen a $\mathcal{A}$, y otra en la que pertenecen a $\mathcal{A} \times \mathcal{A}$. En todos los casos se establece con precisión a qué se le llama contexto.

3.1. Contextos uni–direccionales

Aplicaciones típicas de este caso son la compresión y la predicción.

La secuencia procesada $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ es simple, es decir, $x_i \in \mathcal{A}$, con $i \in \{1, \ldots, n\}$. Los contextos de cada posición son los reversos de las subcadenas de $\mathbf{x}$ que preceden inmediatamente a la posición. Para poder tratar la posición 1 vamos a suponer que hay un elemento x_0 cuyo valor es un símbolo marcador, $\#$, perteneciente a $\mathcal{A}$, que no ocurre en $\mathbf{x}$. Entonces, los contextos de la posición i son los prefijos de ${}_{i-1}^{0}\mathbf{x}$ $(={}_{i-1}^{1}\mathbf{x}\#)$, que es la cadena de formación de contextos para la posición i, que denotamos por $\mathbf{y}^{(i)}$. Así, para la posición 1 esa cadena es $\#$, y los contextos son ϵ y $\#$. Y para cualquier otra posición i, $\mathbf{y}^{(i)}$ es ${}_{i-1}^{0}\mathbf{x}$. Estas n cadenas son sufijos de una misma cadena.

Definición 3.1. La cadena de formación de contextos de $\mathbf{x}_1^n$ es $\mathbf{y} = {}_{n-1}^{0}\mathbf{x}$.

Por lo tanto, para $i \in \{1, \ldots, n\}$, la cadena de formación de contextos para la posición i es $\mathbf{y}_{n+1-i}^{n}$.

Los contextos son las subcadenas de $\mathbf{y}$. Si el contexto $\mathbf{s}$ es una subcadena de $\mathbf{y}$ que empieza en la posición j de $\mathbf{y}$, entonces $\mathbf{y}_j^{j-1+|\mathbf{s}|}$ es un contexto de la posición $n+1-j$ de $\mathbf{x}$. El conjunto de posiciones de $\mathbf{x}$ de las que $\mathbf{s}$ es contexto es su conjunto índice, $\mathcal{I}(\mathbf{s})$. Nuestro propósito es construir conjuntos válidos de contextos, es decir, aquellos para los cuales los conjuntos índices forman una partición de $\{1,\ldots,n\}$.

Ejemplo 3.2. Supongamos que $\mathcal{V} = \{\mathtt{0}, \mathtt{10}, \mathtt{11}, \#\}$ es un conjunto de contextos y sea $\mathbf{x} = \mathtt{001010110}$, siendo $n = 9$. Entonces $\mathbf{y} = \mathtt{11010100}\#$.

x	#	0	0	1	0	1	0	1	1	0
pos		1	2	3	4	5	6	7	8	9
$x(0)$			0	1		1		1		
$x(10)$					0		0		1	
$x(11)$										0
$x(\#)$		0								

y	1	1	0	1	0	1	0	0	#
pos	1	2	3	4	5	6	7	8	9
pos x	9	8	7	6	5	4	3	2	1

Tenemos que $\mathcal{I}(\mathtt{0}) = \{\mathtt{2},\mathtt{3},\mathtt{5},\mathtt{7}\}$. Estas son las posiciones en $\mathbf{x}$ que cumplen que su símbolo precedente es $\mathtt{0}$. Esto se ve también porque mediante el mapeo $i \mapsto n+1-i$ obtenemos el conjunto $\{\mathtt{8},\mathtt{7},\mathtt{5},\mathtt{3}\}$, que son las posiciones de $\mathbf{y}$ en donde empieza la subcadena $\mathtt{0}$. Además $\mathcal{I}(\mathtt{10}) = \{\mathtt{4},\mathtt{6},\mathtt{8}\}$, porque en $\mathbf{y}$ esta cadena empieza en las posiciones $\{\mathtt{6},\mathtt{4},\mathtt{2}\}$. De la misma forma se llega a que $\mathcal{I}(\mathtt{11}) = \{\mathtt{9}\}$. Por último $\mathcal{I}(\#) = \{\mathtt{1}\}$. Se comprueba así que $\mathcal{V}$ es un conjunto válido ya que cada posición pertenece a exactamente uno de los conjuntos índices. La descomposición de $\mathbf{x}$ mediante $\mathcal{V}$ genera las subsecuencias $\mathbf{x}(\mathtt{0}) = (\mathtt{0111})$, $\mathbf{x}(\mathtt{10}) = (\mathtt{001})$, $\mathbf{x}(\mathtt{11}) = (\mathtt{0})$ y $\mathbf{x}(\#) = (\mathtt{0})$.

Si la secuencia fuese $\mathtt{0010100100}$, el conjunto $\mathcal{V}$ seguiría siendo válido aunque en este caso $\mathbf{x}(\mathtt{11})$ sería vacía.

Conjunto válido de contextos

Vamos a establecer una condición necesaria y suficiente que deben cumplir los contextos de un conjunto válido de contextos, $\mathcal{V}$.

Definición 3.3. Se denomina *conjunto libre de prefijos* un conjunto en el que ninguno de sus elementos es prefijo de otro ([14], capítulo 5).

Lema 3.4. *Un conjunto válido de contextos es libre de prefijos.*

Demostración. Supongamos que $\mathbf{s}$ y $\mathbf{t}$ son contextos de $\mathcal{V}$ y que $\mathbf{t}$ es prefijo de $\mathbf{s}$. Para cada posición i de $\mathcal{I}(\mathbf{s})$ el contexto $\mathbf{s}$ es prefijo de la cadena de formación de contextos $\mathbf{y}^{(i)}$ y, por transitividad, $\mathbf{t}$ también es prefijo de $\mathbf{y}^{(i)}$. Entonces, cada posición de $\mathcal{I}(\mathbf{s})$ también pertenece a $\mathcal{I}(\mathbf{t})$, por lo que $\mathcal{I}(\mathbf{s}) \subseteq \mathcal{I}(\mathbf{t})$. Por lo tanto, $\mathcal{V}$ no es un conjunto de contextos que genere una partición de las posiciones $\{1, \ldots, n\}$ por lo que no es válido. □

De esta manera se llega a una condición necesaria para que un conjunto sea válido, que se conoce como *propiedad de prefijo*, y es que ningún elemento del conjunto sea prefijo de otro:

$$\text{Si } \mathbf{s}, \mathbf{t} \in \mathcal{V} \text{ con } \mathbf{s} \neq \mathbf{t} \text{ entonces } \mathbf{s} \not\preceq \mathbf{t} \text{ y } \mathbf{t} \not\preceq \mathbf{s}. \tag{3.1}$$

Obviamente, esta no es una condición suficiente para que un conjunto sea válido. Además de que los conjuntos de posiciones sean disjuntos, se debe lograr la *exhaustividad*, esto es, que toda posición pertenezca a uno de ellos. A un conjunto de contextos tal lo llamamos *exhaustivo*. Sea k el máximo de las longitudes de los contextos de $\mathcal{V}$, $k = \max\{|\mathbf{s}| : \mathbf{s} \in \mathcal{V}\}$. Denotemos por $\mathcal{P}_k(\mathbf{s})$ el conjunto de extensiones de $\mathbf{s}$ de longitud k (versión uni–direccinal de la Expresión (2.2)):

$$\mathcal{P}_k(\mathbf{s}) = \left\{\mathbf{t} \in \mathcal{A}^k : \mathbf{s} \preceq \mathbf{t}\right\}.$$

Vamos a establecer una condición suficiente para que un conjunto de contextos sea exhaustivo.

En el resto de la tesis se impone como restricción que en cualquier contexto una ocurrencia del símbolo marcador $\#$ sólo puede darse en su última posición. Esta no es una restricción a los contextos reales ya que ese símbolo no ocurre en las secuencias y su presencia es sólo motivada para simplificar estructuras de datos y algoritmos.

Lema 3.5. *Sea $\mathcal{V}$ un conjunto de contextos, tal que $k = \max\{|\mathbf{s}| : \mathbf{s} \in \mathcal{V}\}$ y*

$$\bigcup_{\mathbf{s} \in \mathcal{V}} \mathcal{P}_k(\mathbf{s}) = \mathcal{A}^k. \tag{3.2}$$

Entonces se cumple la exhaustividad.

Demostración. Se debe probar que cada posición pertenece al conjunto de índices de un contexto, o sea, que hay un contexto en $\mathcal{V}$ que es prefijo de la cadena de formación de contextos de la posición, esto es, para cada $i \in \{1, \ldots, n\}$ $\exists\, \mathbf{s} \in \mathcal{V}, \mathbf{s} \preceq \mathbf{y}^{(i)}$. Si $\mathbf{y}^{(i)}$ tiene longitud k o más sea $\mathbf{t}$ su prefijo de longitud k. Entonces, según la Expresión (3.2), existe $\mathbf{s} \in \mathcal{V}, \mathbf{s} \preceq \mathbf{t}$. En otro caso, que la longitud de $\mathbf{y}^{(i)}$ sea menor a k implica que su último símbolo es $\#$. Sea $\mathbf{t} = \mathbf{y}^{(i)}$. Como $\mathbf{t}$

termina en $\#$ ninguna extensión estricta suya puede ser un contexto. Y como sus extensiones de largo k deben tener, otra vez según la Expresión (3.2), un prefijo en $\mathcal{V}$, se concluye que también en este caso existe $\mathbf{s} \in \mathcal{V}, \mathbf{s} \preceq \mathbf{t}$. En ambos casos, por la propiedad transitiva de la relación prefijo, $\mathbf{s} \preceq \mathbf{y}^{(i)}$. □

Se deduce el corolario:

Corolario 3.6. *Un conjunto de contextos $\mathcal{V}$ es válido si la relación de extensión aplicada a los elementos $\mathbf{s}$ de $\mathcal{V}$ genera una partición del conjunto $\mathcal{P}_k(\epsilon) = \mathcal{A}^k$ en los conjuntos disjuntos $\mathcal{P}_k(\mathbf{s})$, siendo $k = \max\{|\mathbf{s}| : \mathbf{s} \in \mathcal{V}\}$.*

Esta es una condición suficiente pero no necesaria para que el conjunto $\mathcal{V}$ sea válido. Esto se debe a que alguno de los contextos podría no ocurrir en $\mathbf{x}$. Más adelante, en el Capítulo 4, se tratará esta posibilidad. Por ahora consideramos que un conjunto válido cumple (3.1) y (3.2).

Un conjunto válido de contextos se puede representar con las hojas de un árbol $|\mathcal{A}|$-ario y también se cumple el recíproco [2]. A estos árboles se les llama, por lo tanto, *árboles de contextos*. Se puede extender la definición de peso a árboles:

Definición 3.7. El *peso de un árbol* es la suma de los pesos de los contextos representados por las hojas.

O sea, el peso de un árbol es igual al del conjunto válido de contextos que representa (Ecuación (2.4)). Entonces, el problema de encontrar un conjunto óptimo de contextos se puede transformar en el problema de hallar un árbol de peso mínimo.

Supongamos que se establece k como una cota superior de las longitudes de los contextos. Entonces un árbol que represente un conjunto válido es un subárbol del árbol $|\mathcal{A}|$-ario perfecto de altura k.

En la Figura 3.1 vemos un ejemplo de conjunto válido de contextos y árbol de contextos.

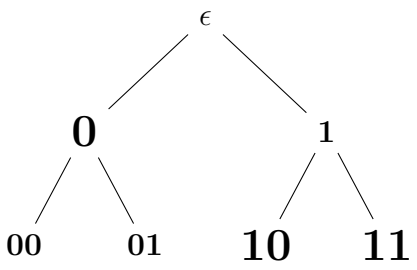


Figura 3.1: Árbol de contextos.
Sea $\mathcal{A} = \{0, 1\}$. Vemos un árbol perfecto de contextos para $k = 2$. El conjunto $\{0, 10\}$ no es exhaustivo para cualquier secuencia, porque 11 no es extensión de ningún elemento del conjunto. El conjunto $\{0, 10, 11, 01\}$ no es disjunto porque 01 es extensión de 0 y de 01. Un conjunto válido para cualquier secuencia es, por ejemplo, $\{0, 10, 11\}$ que está marcado en negrita en la figura.

Para encontrar el árbol óptimo para cada cadena $\mathbf{x}$ se aplica un algoritmo que empieza en las hojas del árbol perfecto. Se define el *peso óptimo* de cada contexto $\mathbf{s}$, $\mathtt{w}_{opt}(\mathbf{s})$. Si el nodo $\mathbf{s}$ es una hoja se calcula el peso con la función $\mathtt{w}(\mathbf{s})$, y $\mathtt{w}_{opt}(\mathbf{s}) = \mathtt{w}(\mathbf{s})$. Sea $\mathbf{s}$ un nodo interno del que ya fueron procesados sus hijos. Se calcula su peso óptimo como el mínimo entre el peso del contexto y la suma de los pesos óptimos de sus hijos: $\mathtt{w}_{opt}(\mathbf{s}) = \min\left\{\mathtt{w}(\mathbf{s}), \sum_{a\in\mathcal{A}} \mathtt{w}_{opt}(\mathbf{s}a)\right\}$. Se construye el árbol óptimo de ese nodo: si el mínimo es el primero de los valores, el árbol es podado y consiste en ese nodo. En el otro caso el árbol es el nodo junto con los árboles óptimos que tienen raíz en los hijos del nodo considerado. En el ejemplo de la Figura 3.1 esto último es lo que pasa en los nodos ϵ y $\mathtt{1}$ si el conjunto representado es el óptimo. En el caso en que los dos valores sean iguales se elige de manera determinística una de las opciones, por ejemplo, el árbol podado.

Un árbol que corresponde a un conjunto válido describe de manera recursiva la partición del conjunto $\mathcal{P}_k(\epsilon)$. Si el árbol no es sólo una hoja, en la raíz se obtienen los $|\mathcal{A}|$ conjuntos $\mathcal{P}_k(a)$ para cada $a \in \mathcal{A}$. Recursivamente, en cada nodo interno $\mathbf{s}$ se establece la partición de $\mathcal{P}_k(\mathbf{s})$ en los conjuntos $\mathcal{P}_k(\mathbf{s}a)$.

3.2. Contextos bi–direccionales

Se ha generalizado el modelado por contextos, considerando contextos compuestos por múltiples componentes profundizando en el caso *bi–direccional* [5, 6, 7]. Para cada posición i de $\mathbf{x}$ se dispone de dos cadenas formadoras de contexto, ${\mathbf{y}_\ell}^{(i)}$ y ${\mathbf{y}_\mathbf{r}}^{(i)}$, llamadas "izquierda" y "derecha" respectivamente. Los *contextos bi-direccionales*, o *bi–contextos* son pares ordenados de contextos, $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$.

En la generalización más directa del caso uni–direccional, la secuencia $\mathbf{x}$ consta de dos pistas, $\mathbf{x} = (\mathbf{x}_\ell, \mathbf{x}_r)$, de tal modo que $\mathbf{x}_i = ([\mathbf{x}_\ell]_i, [\mathbf{x}_r]_i)$. Tal como en el caso uni–direccional se supone que hay un elemento en la posición 0, $\mathbf{x}_0 = (\#, \#)$, siendo $\#$ un elemento de $\mathcal{A}$ que no ocurre en la cadena de entrada. Las cadenas de formación de contextos son $\mathbf{y}_\ell = {}_{n-1}^{0}[\mathbf{x}_\ell]$, y $\mathbf{y}_\mathbf{r} = {}_{n-1}^{0}[\mathbf{x}_r]$.

En la aplicación eliminación de ruido, los elementos de $\mathbf{x}$ son símbolos del alfabeto. Para cada posición i, los contextos izquierdos son los reversos de las subcadenas que preceden la posición i, mientras que los contextos derechos son las subcadenas que empiezan en $i+1$. También aquí se asume que el elemento $\#$ de $\mathcal{A}$ no ocurre en la cadena y se supone $\mathbf{x}_0 = \#$ y $\mathbf{x}_{n+1} = \#$. Entonces $\mathbf{y}_\ell$ es ${}_{n-1}^{0}\mathbf{x} = {}_{n-1}^{1}\mathbf{x}\#$, como antes, pero $\mathbf{y}_\mathbf{r}$ es $\mathbf{x}_2^{n+1} = \mathbf{x}_2^n\#$ y los contextos derechos de la posición i, con $1 \le i \le n$, son los prefijos de $[\mathbf{y}_\mathbf{r}]_i^n$.

El contexto $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ *ocurre* en la posición i si es prefijo de $({\mathbf{y}_\ell}^{(i)}, {\mathbf{y}_\mathbf{r}}^{(i)})$. Las posiciones en que ocurre se denotan por $\mathcal{I}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ y la subsecuencia indizada por esas posiciones es $\mathbf{x}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$.

Se mantiene que un conjunto de contextos $\mathcal{V}$ es válido si genera una partición de las posiciones $\{1, \ldots, n\}$. La condición para que esto se cumpla es la generalización de las propiedades del caso uni–direccional. Esto es:

(1) $\mathcal{V}$ debe ser un conjunto libre de prefijos, lo que implica que los conjuntos de extensiones sean disjuntos por pares: $\mathcal{P}(\mathbf{s}_\ell, \mathbf{s_r}) \cap \mathcal{P}(\mathbf{t}_\ell, \mathbf{t_r}) = \emptyset$ si $(\mathbf{s}_\ell, \mathbf{s_r}) \neq (\mathbf{t}_\ell, \mathbf{t_r})$, con $(\mathbf{s}_\ell, \mathbf{s_r}), (\mathbf{t}_\ell, \mathbf{t_r}) \in \mathcal{V}$;

(2) $\mathcal{V}$ debe ser exhaustivo, esto es, si k_ℓ (resp. k_r) es la mayor longitud de todos los componentes izquierdos (resp. derechos) de los contextos de $\mathcal{V}$, entonces cada contexto con longitud de componente izquierdo (resp. derecho) igual a k_ℓ (resp. k_r) es extensión de algún elemento de $\mathcal{V}$: $\mathcal{P}_{(k_\ell,k_r)}(\epsilon, \epsilon) \subseteq \bigcup_{(\mathbf{s}_\ell,\mathbf{s_r})\in\mathcal{V}} \mathcal{P}(\mathbf{s}_\ell, \mathbf{s_r})$. En el resto de esta sección k_ℓ, k_r tienen el significado aquí definido.

Se ha demostrado [5, 6, 7] que, tal como en el caso uni–direccional, un conjunto válido de contextos bi–direccionales puede representarse mediante las hojas de un árbol cuyos nodos pertenecen ahora a $\mathcal{A}^* \times \mathcal{A}^*$, con (ϵ, ϵ) como raíz. También se cumple el recíproco, un árbol de contextos bi–direccionales representa un conjunto válido de contextos bi–direccionales. Sin embargo, la estructura de este árbol es diferente al del caso uni–direccional. Los *árboles de contextos bi–direccionales* cumplen que cada nodo interno tiene como hijos o bien todas sus extensiones mínimas por izquierda o bien todas sus extensiones mínimas por derecha. O sea, si $(\mathbf{s}_\ell, \mathbf{s_r})$ no es una hoja, sus hijos son o bien $\{(\mathbf{s}_\ell a, \mathbf{s_r}) : a \in \mathcal{A}\}$, o bien $\{(\mathbf{s}_\ell, \mathbf{s_r} a) : a \in \mathcal{A}\}$.

A diferencia del caso uni–direccional, un conjunto válido puede ser representado por más de un árbol.

Por ejemplo, el conjunto $\{(\mathtt{0}, \mathtt{0}), (\mathtt{1}, \mathtt{01}), (\mathtt{10}, \mathtt{1}), (\mathtt{01}, \mathtt{1}), (\mathtt{1}, \mathtt{1})\}$, con $\mathcal{A} = \{0, 1\}$, no es exhaustivo porque ningún contexto es prefijo de $(\mathtt{10}, \mathtt{00})$. Y no es libre de prefijos porque $(\mathtt{10}, \mathtt{1})$ es extensión de $(\mathtt{1}, \mathtt{1})$ y $(\mathtt{10}, \mathtt{1})$.

El conjunto $\{(\mathtt{0}, \mathtt{0}), (\mathtt{1}, \mathtt{00}), (\mathtt{1}, \mathtt{01}), (\mathtt{00}, \mathtt{1}), (\mathtt{01}, \mathtt{1}), (\mathtt{1}, \mathtt{1})\}$ es válido. Y puede ser representado por dos árboles, como se muestra en la Figura 3.2.

Como en el caso uni–direccional un conjunto válido puede describirse con una secuencia de particiones de las extensiones pertenecientes a $\mathcal{P}_{(k_\ell,k_r)}(\epsilon, \epsilon)$ de los nodos del árbol que representa al conjunto. Ahora hay tres posibles particiones. La "no partición", que es lo que se hace en las hojas del árbol, y las particiones izquierda y derecha.

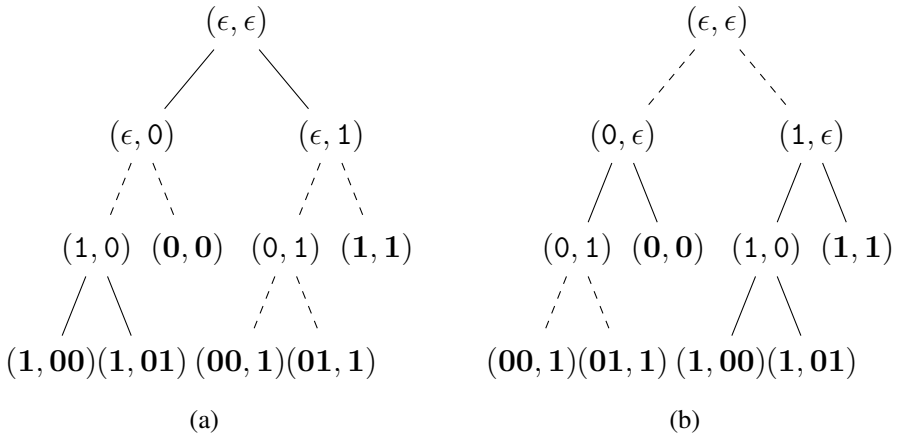


Figura 3.2: Dos árboles de contextos bi–direccionales que representan el mismo conjunto de contextos, que son las hojas de los árboles, iguales en ambos. En trazo continuo se muestran las particiones derechas y en trazo discontinuo las particiones izquierdas.

Podado bi–direccional

Tal como en el caso uni–direccional las subsecuencias $\mathbf{x}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ determinadas por los contextos $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ inducen un costo por lo que a los contextos se les asigna un peso $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. El peso del conjunto válido de contextos es la suma de los pesos de los contextos y queremos encontrar un conjunto óptimo, esto es, de peso mínimo.

Con $\mathcal{T}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ representamos los subárboles con raíz en $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ de los árboles de contextos bi–direccionales. Las hojas de un árbol $T \in \mathcal{T}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ son contextos que forman un subconjunto de un conjunto válido. Los elementos de este subconjunto son los contextos del conjunto válido que son extensión de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. Esto motiva la siguiente definición.

Definición 3.8. El conjunto de contextos $\mathcal{V}$ es un *conjunto válido relativo a* $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ si el conjunto $\left\{\mathcal{P}_{(k_\ell,k_r)}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) : (\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) \in \mathcal{V}\right\}$ es una partición de $\mathcal{P}_{(k_\ell,k_r)}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, donde k_ℓ (resp. k_r) es el máximo de las longitudes de los componentes izquierdos (resp. derechos) de los elementos de $\mathcal{V}$.

Es claro que debe cumplirse $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \preceq \mathcal{V}$.

También podemos definir un *conjunto óptimo relativo a* $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ como uno que tenga peso mínimo entre los conjuntos válidos relativos a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$.

Obviamente, un conjunto válido y un conjunto óptimo como fue definido antes es un conjunto válido relativo a (ϵ, ϵ) y un conjunto óptimo relativo a (ϵ, ϵ) respectivamente. Esto sugiere un método para encontrar un conjunto óptimo (o

sea, un conjunto óptimo relativo a la raíz de un árbol) si se puede demostrar que se cumple el principio de optimalidad. Se empieza con los contextos más largos, para cada uno de los cuales el conjunto óptimo relativo a él es el propio contexto. Para cualquier otro contexto se va a conocer un conjunto óptimo relativo a cada una de sus extensiones mínimas, tanto izquierdas como derechas. Haciendo valer el principio de optimalidad se obtiene, mediante un algoritmo de programación dinámica [15], el conjunto óptimo relativo al contexto procesado.

Con $\mathcal{T}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$ se denota al subconjunto de $\mathcal{T}(\mathbf{s}_\ell, \mathbf{s_r})$ cuyo peso es mínimo, siendo el peso de un árbol igual a la suma de los pesos de sus hojas, esto es, el peso del conjunto válido que representa. Las hojas de un árbol de $\mathcal{T}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$ forman un conjunto óptimo relativo a $(\mathbf{s}_\ell, \mathbf{s_r})$. Como ya fue notado, $\mathcal{T}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$ puede tener más de un elemento, aún en el caso de que haya un único conjunto válido de peso mínimo. Cualquiera de los árboles pertenecientes a $\mathcal{T}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$ puede ser denotado por $T_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$. Al peso de los árboles de $\mathcal{T}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$ lo denotamos por $\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$.

Se ha demostrado [16, 6, 7] que se cumple el principio de optimalidad que nos permite establecer el peso óptimo de un contexto conociendo el de sus extensiones mínimas:

$$\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s_r}) = \text{mín} \left\{ \mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r}), \sum_{a \in \mathcal{A}} \mathtt{w}_{opt}(\mathbf{s}_\ell a, \mathbf{s_r}), \sum_{a \in \mathcal{A}} \mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s_r} a) \right\}. \tag{3.3}$$

Si en el segundo término $|\mathbf{s}_\ell a| > k_\ell$, o sea, si el contexto no cumple la restricción establecida, entonces se considera que $\mathtt{w}_{opt}(\mathbf{s}_\ell a, \mathbf{s_r}) = \infty$. Lo mismo sucede en el tercer término si $|\mathbf{s_r} a| > k_r$.

Se debe establecer una regla para elegir el mínimo en caso de que más de uno de estos valores lo sea. Y entonces se puede definir la función **split** para cada contexto bi–direccional que refleja cual es la opción que permite obtener el peso óptimo. El valor de esta función es {'N', 'L', 'R'} según que el óptimo se alcance respectivamente mediante la "no partición", la partición izquierda o la partición derecha.

Entonces, como en el caso uni–direccional se puede establecer un algoritmo que mediante programación dinámica encuentra un árbol óptimo. Este es el Algoritmo 1.

Cuando $j = 0$ se cumple que $(\mathbf{s}_\ell, \mathbf{s_r}) = (\epsilon, \epsilon)$ y por lo tanto se obtiene el árbol óptimo. Los árboles se están describiendo mediante sus nodos, sin mencionar las ramas que quedan implícitamente definidas. Al tener reglas de desempate en cada conjunto $\mathcal{T}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$ hay un árbol que el algoritmo obtiene de manera determinística que se denota por $T_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$.

El peso de un contexto es una función del vector de conteos del contexto. En el primer bucle del Algoritmo 1 se calcula el peso de los contextos de longitud

Algoritmo 1 Árbol óptimo de contextos bi–direccionales.

Para cada $(\mathbf{s}_\ell, \mathbf{s_r}) \in \mathcal{A}^{k_\ell} \times \mathcal{A}^{k_r}$
  calcular $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r})$, y asignar $\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$= $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r})$
  $T_{opt}(\mathbf{s}_\ell, \mathbf{s_r}) = \{(\mathbf{s}_\ell, \mathbf{s_r})\}$
Para $j = k_\ell + k_r - 1$ **hasta** 0
  Para cada $(\mathbf{s}_\ell, \mathbf{s_r})$ tal que $|\mathbf{s}_\ell| + |\mathbf{s_r}| = j, |\mathbf{s}_\ell| \leq k_\ell, |\mathbf{s_r}| \leq k_r$
    calcular $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r})$ y, mediante la ecuación (3.3), $\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s_r})$
    obtener $\mathbf{split}(\mathbf{s}_\ell,\mathbf{s_r})$
    $T_{opt}(\mathbf{s}_\ell, \mathbf{s_r}) =$

$$\begin{cases} \{(\mathbf{s}_\ell, \mathbf{s_r})\} & \text{si } \mathbf{split}(\mathbf{s}_\ell, \mathbf{s_r}) = \text{'N'}, \\ \{(\mathbf{s}_\ell, \mathbf{s_r})\} \cup \bigcup_{a \in \mathcal{A}} T_{opt}(\mathbf{s}_\ell a, \mathbf{s_r}) & \text{si } \mathbf{split}(\mathbf{s}_\ell, \mathbf{s_r}) = \text{'L'}, \\ \{(\mathbf{s}_\ell, \mathbf{s_r})\} \cup \bigcup_{a \in \mathcal{A}} T_{opt}(\mathbf{s}_\ell, \mathbf{s_r} a) & \text{si } \mathbf{split}(\mathbf{s}_\ell, \mathbf{s_r}) = \text{'R'}. \end{cases}$$

máxima, por lo cual debe obtenerse el vector de conteos mediante inspección de $\mathbf{x}$. Pero para el resto de los contextos no es necesario porque el vector se obtiene a partir de los contextos más largos.

$$\mathbf{c}(\mathbf{s}_\ell, \mathbf{s_r}) = \sum_{a \in \mathcal{A}} \mathbf{c}(\mathbf{s}_\ell a, \mathbf{s_r}) = \sum_{a \in \mathcal{A}} \mathbf{c}(\mathbf{s}_\ell, \mathbf{s_r} a). \tag{3.4}$$

Hay que notar una diferencia con el caso uni–direccional. En aquel caso el algoritmo que obtiene el árbol óptimo recorre desde abajo hacia arriba, desde los contextos largos hacia los cortos, un árbol, de hecho un árbol perfecto. En la generalización bi–direccional, la estructura implícita que se está recorriendo es una generalización del árbol perfecto. Lo llamamos *grafo de contextos bi–direccionales*. Cada nodo interno tiene dos conjuntos de hijos. Si los dos componentes de un nodo son diferentes de ϵ, entonces el nodo tiene dos ramas entrantes en él. El nodo $(\mathbf{s}_\ell a, \mathbf{s_r} b)$ desciende de $(\mathbf{s}_\ell a, \mathbf{s_r})$ y de $(\mathbf{s}_\ell, \mathbf{s_r} b)$, o sea, tiene dos "padres". Respecto al primero está incluido en la partición derecha, y respecto al segundo en la partición izquierda.

Advertimos otra diferencia con el caso uni–direccional que, como se verá en el Capítulo 4, determina la complejidad de obtener un conjunto óptimo. En una partición de un nodo interno, digamos la izquierda, no están incluidas todas las extensiones estrictas, sino sólo las extensiones estrictas en el componente izquierdo. Por ejemplo en la Figura 3.2(b), en la raíz se elige la partición izquierda y, como consecuencia, el contexto $(\epsilon, 0)$ no está incluido en la partición. Sin embargo las extensiones $\mathcal{P}_{(k_\ell,k_r)}(\epsilon, 0)$ de $(\epsilon, 0)$, con $k_\ell, k_r \geq 2$, si están incluidos en la

partición. Esto es porque ese conjunto de extensiones son también extensión de otros contextos.

Se debe destacar la importancia de poder representar un conjunto válido mediante un árbol porque existen ejemplos para 3 direcciones en que eso no es cierto [7] y se ha demostrado que ese no es un caso aislado para multi–direccionalidad mayor a 2 [6].

Lo demostrado en el Corolario 3.6 y su equivalente en el marco bi–direccional corresponde a conjuntos válidos para cualquier secuencia. En lo que sigue el concepto conjunto válido es aplicado para cada secuencia específica y se excluyen del conjunto aquellos contextos que no ocurren en la secuencia.

Capítulo 4

Contextos Representativos

En este capítulo nos planteamos el problema de eliminar la restricción que en el primer bucle del Algoritmo 1 se impone a la longitud de los contextos. Esto en realidad significa que el límite sea n. El problema es que el algoritmo, aún si se simplifica procesando sólo los contextos que ocurren, en el caso bi–direccional es de complejidad cúbica, ya que para cada una de las n posiciones hay $\Theta(n^2)$ contextos.

En el caso uni–direccional se puede resolver el problema, y la complejidad cuadrática de la solución simplista se reduce a lineal utilizando árboles de sufijos [4]. Tal como se describirá en el Capítulo 6, aquí utilizaremos esa misma herramienta y además algoritmos para calcular el prefijo común más largo de los contextos representados en los árboles.

Veremos que los contextos se pueden separar en clases de tal modo que todos los elementos de la misma clase tengan las mismas propiedades, determinadas por ocurrir en exactamente las mismas posiciones de $\mathbf{x}$. Entonces, en lugar de procesar todos los contextos que ocurren se procesa sólo uno por clase, el representativo.

Vamos a caracterizar las clases y los contextos representativos. Hay un equivalente a los contextos de longitud máxima con el cual comienza el Algoritmo 1. Son aquellos que no pueden extenderse porque alcanzan el final de la cadena de formación de contextos. Los contextos de la misma clase son aquellos que son prefijos de los mismos contextos de longitud máxima. Estos contextos de longitud máxima son la generalización de las hojas de los árboles de sufijos. La generalización de los nodos internos se obtiene como un subconjunto del producto cartesiano de los nodos internos de los árboles de sufijos que corresponden a las dos cadenas de formación de contextos. Demostraremos que en la búsqueda de un conjunto óptimo de contextos para la secuencia $\mathbf{x}$ dada alcanza con considerar sólo los contextos representativos.

4.1. Caso uni–direccional

Para el caso uni–direccional habíamos visto que las cadenas de formación de contextos de todas las posiciones son sufijos de una misma cadena: el reverso de $\mathbf{x}$ con el agregado de la marca de finalización $\#$, obteniendo ${}_{n-1}^{0}\mathbf{x}(= {}_{n-1}^{1}\mathbf{x}\#)$. A esa cadena, la denotamos por $\mathbf{y}$, y la llamamos la *cadena de formación de contextos* de $\mathbf{x}$. Por lo tanto, los contextos de $\mathbf{x}_i$ son los prefijos de $\mathbf{y}_{n+1-i}^{n}$. Decimos que un contexto $\mathbf{s}$ *ocurre* en la posición i si $\mathbf{s} \preceq \mathbf{y}_{n+1-i}^{n}$.

Definición 4.1. Para cada i, $1 \leq i \leq n$, el contexto de máxima longitud que ocurre en i es llamado *contexto sufijo*.

El agregado de la marca de finalización es para poder representar la cadena $\mathbf{y}$ con un árbol de sufijos, $T_{\mathbf{y}}$. Los contextos sufijos terminan en $\#$ y, como ya se vio, están en correspondencia uno a uno con las hojas de $T_{\mathbf{y}}$.

Un contexto puede ocurrir en diferentes posiciones, y por lo tanto puede ser prefijo de varios contextos sufijos. Recordemos que llamamos conjunto índice de $\mathbf{s}$ y lo denotamos con $\mathcal{I}(\mathbf{s})$ al conjunto de posiciones donde ocurre $\mathbf{s}$.

Definición 4.2. El *conjunto de contextos sufijo* de $\mathbf{y}$ determinado por el contexto $\mathbf{s}$, denotado por $\mathcal{S}(\mathbf{s})$, es el conjunto de sufijos de $\mathbf{y}$ que extienden $\mathbf{s}$. Esto es

$$\mathcal{S}(\mathbf{s}) = \left\{\mathbf{y}_{n+1-i}^{n}, i \in \mathcal{I}(\mathbf{s})\right\}.$$

Entonces $\mathbf{s}$ ocurre si y sólo si $\mathcal{S}(\mathbf{s})$ no es vacío.

Como $\mathbf{s} \preceq \mathcal{S}(\mathbf{s})$ y la relación prefijo es transitiva, se tiene que

$$\mathbf{t} \preceq \mathbf{s} \Rightarrow \mathbf{t} \preceq \mathcal{S}(\mathbf{s}) \Rightarrow \mathcal{S}(\mathbf{s}) \subseteq \mathcal{S}(\mathbf{t}). \tag{4.1}$$

El recíproco no se cumple en general porque puede haber varios contextos con el mismo conjunto sufijo. Esto es lo que permitirá reducir la complejidad porque sólo hará falta procesar uno de ellos.

Ejemplo 4.3. Sea $\mathbf{x} = \texttt{1110101}$, y por lo tanto $\mathbf{y} = \texttt{010111}\#$. El contexto $\texttt{011}$ ocurre en la posición 5 (ya que $\texttt{011} = x_4x_3x_2$), por lo que $\mathcal{S}(\texttt{011}) = \{\texttt{0111}\#\}$. Sea $\mathbf{s} = \texttt{0}$ un prefijo de $\texttt{011}$. Entonces, $\mathcal{I}(\mathbf{s})$ es $\{5, 7\}$, de donde $\mathcal{S}(\mathbf{s}) = \{\texttt{0111}\#, \texttt{010111}\#\} \supseteq \mathcal{S}(\texttt{011})$.

Para ver que no se cumple el recíproco de (4.1) sea $\mathbf{t} = \texttt{01}$. Tenemos que $\mathbf{t}$ ocurre en las mismas posiciones que $\mathbf{s}$. Por lo tanto se cumple $\mathcal{S}(\mathbf{s}) \subseteq \mathcal{S}(\mathbf{t})$, aunque $\mathbf{t} \not\preceq \mathbf{s}$.

Como hay una correspondencia entre las posiciones y los sufijos, es inmediato el siguiente teorema.

Teorema 4.4. *Se cumple* $\mathcal{S}(\mathbf{s}) = \mathcal{S}(\mathbf{t})$ *si y sólo si* $\mathcal{I}(\mathbf{s}) = \mathcal{I}(\mathbf{t})$.

La relación de igualdad entre conjuntos de sufijos define clases de equivalencia.

Definición 4.5. La *partición por igualdad de conjuntos de sufijos* de la secuencia $\mathbf{x}$, es la partición en clases del conjunto de contextos que ocurren, de tal manera que todos los elementos de una clase tienen el mismo conjunto de sufijos.

Denotamos la partición por Π, y una clase genérica por $\mathcal{C}$. La clase a la cual pertenece el contexto $\mathbf{s}$ se denota por $\mathcal{C}(\mathbf{s})$. Además, denotamos por $\mathcal{S}(\mathcal{C})$ al conjunto de contextos sufijos de los elementos de la clase $\mathcal{C}$.

Ejemplo 4.6. Según lo visto en el Ejemplo 4.3, `0` y `01` pertenecen a una clase $\mathcal{C}$, tal que $\mathcal{S}(\mathcal{C}) = \{\texttt{0111\#}, \texttt{010111\#}\}$. Ya sabíamos que $\mathcal{S}(\texttt{011})$ está contenido de manera estricta en $\mathcal{S}(\mathcal{C})$, por lo que `011` no está en $\mathcal{C}$. Se puede ver que $\mathcal{S}(\texttt{010}) = \{\texttt{010111\#}\}$, y por lo tanto, tampoco `010` pertenece a $\mathcal{C}$. Entonces ninguna extensión estricta de `01` puede pertenecer a $\mathcal{C}$ (ya que no pertencen ni `010` ni `011` y según la Expresión (4.1) las conjuntos de sufijos de las extensiones de estos contextos están contenidos en $\mathcal{S}(\texttt{010})$ o $\mathcal{S}(\texttt{011})$). Por otro lado `00` no ocurre en $\mathbf{x}$. Finalmente $\mathcal{S}(\epsilon)$ es un conjunto que contiene siete elementos, que son todos los sufijos de $\mathbf{y}$, por lo que ϵ no pertenece a $\mathcal{C}$. Entonces en la clase $\mathcal{C}$ no hay otros contextos, esto es, $\mathcal{C} = \{\texttt{0}, \texttt{01}\}$.

Una clase queda determinada por un conjunto de contextos sufijos. Pero cualquier conjunto de contextos sufijos no determina una clase.

Teorema 4.7. *Dada una clase* $\mathcal{C}$ *de* Π *todos los elementos de* $\mathcal{S}(\mathcal{C})$ *deben tener un prefijo común, que no debe ser prefijo de ningún otro contexto sufijo.*

Demostración. La primera parte de la afirmación es evidente ya que $\epsilon \preceq \mathcal{S}(\mathcal{C})$. Sea $\mathbf{s}$ el más largo de los contextos tal que $\mathbf{s} \preceq \mathcal{S}(\mathcal{C})$. Supongamos que además existiera un contexto sufijo $\mathbf{t}$, tal que $\mathbf{t} \notin \mathcal{S}(\mathcal{C})$ y $\mathbf{s} \preceq \mathbf{t}$.

Por ser $\mathbf{s}$ el más largo de los prefijos comunes de $\mathcal{S}(\mathcal{C})$, para toda extensión estricta de $\mathbf{s}$, $\mathbf{s}a, a \in \mathcal{A}$, existe algún elemento de $\mathcal{S}(\mathcal{C})$, $\mathbf{s}b\mathbf{v}, b \in \mathcal{A}, b \neq a$, del cual no es prefijo. Por lo tanto $\mathbf{s}a$ no pertenece a $\mathcal{C}$. Entonces, los elementos de $\mathcal{C}$ deben ser prefijos de $\mathbf{s}$. Pero todos los prefijos de $\mathbf{s}$, con este incluido, son también prefijos de $\mathbf{t}$. Esto contradice la hipótesis de que $\mathbf{t} \notin \mathcal{S}(\mathcal{C})$. □

Ejemplo 4.8. En la Figura 4.1 se muestra el árbol de sufijos $T_{\mathbf{y}}$ correspondiente a la cadena de formación de contextos $\mathbf{y} = \texttt{010111\#}$ del Ejemplo 4.3. El conjunto de sufijos representado por las hojas 1 y 3 determinan la clase $\{\texttt{0}, \texttt{01}\}$.

La hoja 3 determina la clase $\{\texttt{011}, \texttt{0111}, \texttt{0111\#}\}$.

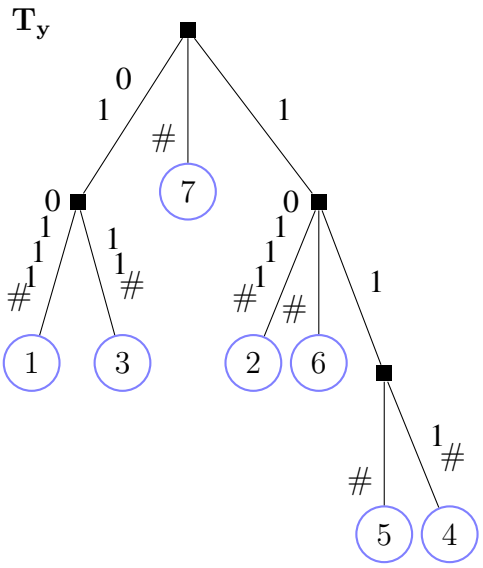


Figura 4.1: Árbol de la cadena de formación de contextos de $\mathbf{y} = 010111\#$.

Ahora veamos que los sufijos correspondientes a las hojas 2, 4 y 5 no determinan una clase. Los únicos contextos que son prefijos de estos tres sufijos son ϵ y 1. Entonces estos son los únicos elementos posibles de la hipotética clase. Pero 1 es también prefijo del sufijo correspondiente a la hoja 6. Y ϵ, además es también prefijo de 1, 3, 6 y 7. O sea que ningún contexto es prefijo de 2, 4 y 5 pero de ningún otro sufijo. Por otra parte 2, 4, 5 y 6 determinan la clase $\{1\}$.

Vamos a caracterizar las clases y definir los contextos representativos.

Cada rama $\overrightarrow{\mathbf{ts}}$ de $T_\mathbf{y}$ define una cadena $\mathbf{u} = u_1 u_2 \dots u_j, j \geq 1$, tal que $\mathbf{s} = \mathbf{tu}$. Por lo tanto $\overrightarrow{\mathbf{ts}}$ representa al conjunto de contextos $\mathcal{C} = \{\mathbf{tu}_1^i, 1 \leq i \leq j\}$. Se debe notar que $\mathbf{s}$ $(= \mathbf{tu}_1^j)$ pertenece a este conjunto.

Teorema 4.9. *Sea* $\mathbf{s}$ *un nodo de* $T_\mathbf{y}$*, distinto de la raíz, y* $\mathbf{t}$ *su nodo padre. Los contextos representados por* $\overrightarrow{\mathbf{ts}}$ *son una clase.*

Demostración. Las hojas (sufijos) que descienden de esos contextos son las mismas. Falta probar que ningún otro contexto es prefijo de exactamente esas hojas. Otros contextos que también son prefijos de esas hojas son $\mathbf{t}$ y sus prefijos. Pero estos contextos son además prefijos de otras hojas. Si $\mathbf{s}$ es una hoja el teorema queda probado. En otro caso cualquier par de extensiones mínimas de $\mathbf{s}$ son disjuntas y por lo tanto ninguna de ellas ni sus extensiones son prefijos de las mismas hojas. □

Corolario 4.10. *Hay una clase de* Π *para cada nodo de* $T_{\mathbf{y}}$.

Demostración. Cada nodo distinto de la raíz identifica la rama que entra en él y por lo tanto a una clase. El nodo raíz es otra clase ya que es el único contexto del cual descienden todas las hojas. □

Vemos que el conjunto $\mathcal{C}$ acepta un orden total en que cada contexto es prefijo del siguiente. Cualquier contexto de $\mathcal{C}$ puede representar la clase. Se puede elegir $\mathbf{t}u_1$ [4], que es prefijo (el prefijo común más largo) de todos los elementos de la clase. Para nuestra generalización al caso bi–direccional tendremos que elegir otro porque no habrá, en general, un contexto que sea prefijo de todos los otros de la clase. Anticipamos que el que elegimos es el equivalente a $\mathbf{s}$, que es la extensión común más corta (concepto que se presentará en la Definición 4.15) de todos los elementos de la clase.

Definición 4.11. El *contexto representativo* de la clase $\mathcal{C}$, denotado por $\mathsf{R}(\mathcal{C})$ es el contexto $\mathbf{s}$, que es la extensión de todos los contextos de la clase. También, por extensión, definimos el representativo del contexto $\mathbf{u}$ y lo denotamos por $\mathsf{R}(\mathbf{u})$ como el representativo de la clase a la que pertenece $\mathbf{u}$.

4.2. Generalización Bi–direccional

En el caso bi–direccional asumimos que, al igual que en el uni–direccional, todas las cadenas ${\mathbf{y}_\ell}^{(i)}$ y ${\mathbf{y}_\mathbf{r}}^{(i)}$ son sufijos respectivamente de $\mathbf{y}_\ell$ y $\mathbf{y}_\mathbf{r}$ llamadas *cadenas de formación de contextos izquierda y derecha* respectivamente. De esta manera todos los contextos izquierdos están representados en un árbol de sufijos, $T_{\mathbf{y}_\ell}$ y todos los contextos derechos están representados en $T_{\mathbf{y}_\mathbf{r}}$.

Definición 4.12. Los *contextos bi–sufijo* son los n contextos $\left({\mathbf{y}_\ell}^{(i)}, {\mathbf{y}_\mathbf{r}}^{(i)}\right)$, con $i \in \{1, \ldots, n\}$. Cada uno de estos es el contexto más largo para una de las posiciones.

Recordamos que el contexto bi–direccional $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ ocurre en i si es prefijo de $\left({\mathbf{y}_\ell}^{(i)}, {\mathbf{y}_\mathbf{r}}^{(i)}\right)$. Definimos $\mathcal{I}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ como en el caso uni–direccional. Tal como en el caso uni–direccional se define el *conjunto de sufijos* $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, esto es, el conjunto de contextos bi–sufijo que son extensión de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. Definimos asimismo la partición por igualdad de conjuntos de sufijos Π del conjunto de todos los contextos bi–direccionales que ocurren en $\mathbf{x}$. De esta manera si $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ y $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ pertenecen a la misma clase $\mathcal{C}$, entonces $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = \mathcal{S}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) = \mathcal{S}(\mathcal{C})$. La clase a la que pertenece $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ la denotamos $\mathcal{C}(\mathbf{s}_\ell,\mathbf{s}_\mathbf{r})$. Además $\mathcal{S}$ denota al conjunto de todos los bi–sufijos. Cualquier contexto bi–direccional que ocurre es prefijo de un elemento de $\mathcal{S}$. También se cumple la generalización de la Expresión (4.1):

$$(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) \preceq (\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \Rightarrow (\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) \preceq \mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \Rightarrow \mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \subseteq \mathcal{S}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}). \tag{4.2}$$

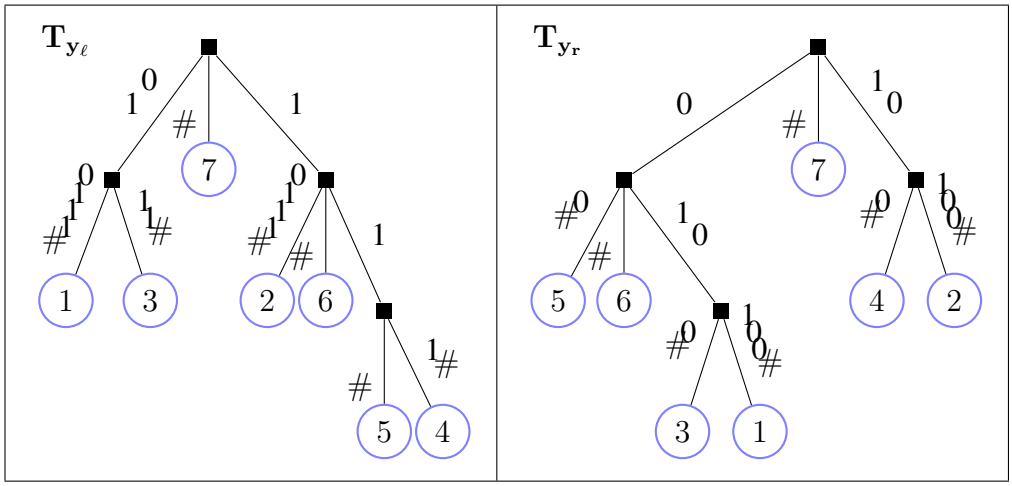


Figura 4.2: Árboles de las cadenas de formación de contextos para la cadena $\mathbf{x} = \begin{pmatrix} \texttt{1110101} \\ \texttt{0010101} \end{pmatrix}$.

Notemos que un bi–sufijo está determinado por dos contextos que son sufijos en sus respectivas cadenas de formación de contextos, y por lo tanto hojas en sus respectivos árboles de sufijos. Por lo tanto la relación bi–sufijos induce una correspondencia uno a uno entre las n hojas de $T_{\mathbf{y}_\ell}$ y las n hojas de $T_{\mathbf{y_r}}$.

Ejemplo 4.13. Sea $\mathbf{x} = \begin{pmatrix} \texttt{1110101} \\ \texttt{0010101} \end{pmatrix}$, de donde $\begin{pmatrix} \mathbf{y}_\ell \\ \mathbf{y_r} \end{pmatrix} = \begin{pmatrix} \texttt{010111\#} \\ \texttt{010100\#} \end{pmatrix}$. Los contextos $(\texttt{01}, \texttt{0})$ y $(\texttt{0}, \texttt{010})$ ocurren ambos en las mismas posiciones, 5 y 7, de $\mathbf{x}$ y por lo tanto pertenecen a la misma clase $\mathcal{C}$.

En la Figura 4.2 se muestran los árboles de sufijos de las cadenas formadoras de contextos.

4.3. Contextos Representativos

Mientras que en el caso uni–direccional, como ya se mencionó, los contextos que pertenecen a la misma clase $\mathcal{C}$ satisfacen una relación de orden total, esto *no* sucede en el caso bi–direccional. Más aún, no existe un contexto bi–direccional que sea prefijo de todos los otros de la clase. Sin embargo, como veremos en el Lema 4.17, cada clase contiene un elemento que es una extensión común de todos

los contextos bi–direccionales de la clase. Antes de establecer el lema veremos un ejemplo y agregaremos una definición.

Ejemplo 4.14. En el Ejemplo 4.13, además de $(\mathtt{01}, \mathtt{0})$ y $(\mathtt{0}, \mathtt{010})$, también pertenecen a la clase $\mathcal{C}$ los contextos bi–direccionales $(\mathtt{0}, \epsilon)$, $(\mathtt{01}, \epsilon)$, $(\mathtt{0}, \mathtt{0})$, $(\epsilon, \mathtt{01})$, $(\epsilon, \mathtt{010})$, $(\mathtt{0}, \mathtt{01})$, $(\mathtt{01}, \mathtt{01})$, $(\mathtt{01}, \mathtt{010})$. La presencia de $(\mathtt{0}, \epsilon)$ y $(\epsilon, \mathtt{01})$ evidencia que no hay un contexto bi–direccional, distinto de (ϵ, ϵ), que sea prefijo de todos los demás. Por otra parte, el contexto bi–direccional $(\mathtt{01}, \mathtt{010})$ es una extensión de todos los contextos bi–direccionales de $\mathcal{C}$.

Definimos la *extensión común más corta* de dos contextos uni–direccionales $\mathbf{s}$ y $\mathbf{t}$ no disjuntos como el más largo de ellos. En un conjunto de contextos no-disjuntos por pares la extensión común más corta es el más largo de los contextos. La motivación de esta definición es su generalización.

Definición 4.15. Sean $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ y $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ contextos bi–direccionales no–disjuntos La *extensión común más corta* de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ y $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ es el contexto bi–direccional cuyo componente izquierdo (resp. derecho) es la extensión común más corta entre $\mathbf{s}_\ell$ y $\mathbf{t}_\ell$ (resp. $\mathbf{s}_\mathbf{r}$ y $\mathbf{t}_\mathbf{r}$). En un conjunto de contextos bi–direccionales cuyos elementos son no–disjuntos por pares también se define la extensión común más corta. El componente izquierdo (resp. derecho) es la extensión común más corta de los componentes izquierdos (resp. derechos) de los contextos del conjunto.

Ejemplo 4.16. La extensión común más corta de los contextos bi–direccionales $(\mathtt{01}, \mathtt{0})$ y $(\mathtt{0}, \mathtt{010})$ es $(\mathtt{01}, \mathtt{010})$.

Las relaciones prefijo y extensión son relaciones de orden parcial. Los bi–sufijos que determinan una clase $\mathcal{C}$ de Π son sucesores (en la relación de orden extensión) de todos los contextos de $\mathcal{C}$. Por lo tanto el conjunto de sucesores de $\mathcal{C}$ no es vacío. Con el siguiente lema establecemos que el conjunto de todos los sucesores de los elementos de la clase, tiene un mínimo, que es, por lo tanto, el supremo de la clase. Además, ese supremo pertenece a la clase, por lo que la clase tiene un contexto máximo.

Lema 4.17. *Sea $\mathcal{C}$ una clase de Π y sea $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ el prefijo común más largo de $\mathcal{S}(\mathcal{C})$. Entonces, $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ pertenece a $\mathcal{C}$ y es la extensión común más corta de $\mathcal{C}$.*

Demostración. Sea $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ un contexto cualquiera de $\mathcal{C}$. Como $\mathbf{s}_\ell$ es el prefijo común más largo de los componentes izquierdos de los bi–sufijos $\mathcal{S}(\mathcal{C})$ y $\mathbf{t}_\ell$ es

también prefijo de esos componentes, se tiene que $\mathbf{t}_\ell$ es prefijo de $\mathbf{s}_\ell$. De consideraciones análogas para los componentes derechos se llega a $\mathcal{C} \preceq (\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, lo que prueba que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es una extensión de todos los elementos de $\mathcal{C}$.

Entonces, por (4.2), $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \subseteq \mathcal{S}(\mathcal{C})$. Además se cumple que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \preceq \mathcal{S}(\mathcal{C})$ y por lo tanto $\mathcal{S}(\mathcal{C}) \subseteq \mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. De donde $\mathcal{S}(\mathcal{C}) = \mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, lo que significa que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ pertenece a $\mathcal{C}$.

Queda por probar que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es la más corta de las extensiones comunes de $\mathcal{C}$. Para esto basta observar que como $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ pertenece a $\mathcal{C}$, la extensión común más corta de $\mathcal{C}$ también debe ser extensión de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. □

El contexto $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ definido en el Lema 4.17 será el elegido como el representativo de la clase y denotado por $\mathsf{R}(\mathcal{C})$. Para cualquier contexto $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ en $\mathcal{C}$, también se considera que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es su representativo, lo cual se denota por $\mathsf{R}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$. Para un contexto que no ocurre no se define su representativo. El conjunto de todos los representativos se denota por R.

Si $\mathcal{S}(\mathcal{C})$ tiene un único elemento, el contexto representativo de $\mathcal{C}$ es ese único elemento. Hay n clases en estas condiciones, una para cada bi–sufijo. Estos contextos representativos se obtienen de manera trivial. En el caso uni–direccional ese único elemento de $\mathcal{S}(\mathcal{C})$ es una hoja del árbol de sufijos, y la clase la constituyen los contextos que son representados por la rama que entra en la hoja. Con el objetivo de obtener el contexto representativo de otras clases, presentaremos una propiedad que ese contexto debe cumplir.

Sea $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ un contexto bi–direccional que no es bi–sufijo, y por lo tanto $\mathbf{s}_\ell$ o $\mathbf{s}_\mathbf{r}$ no son sufijos. Sin pérdida de generalidad supongamos que $\mathbf{s}_\mathbf{r}$ no es sufijo. Entonces, para cada bi–sufijo en $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ hay una única extensión mínima por derecha de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ que es prefijo de ese bi–sufijo. Por lo tanto, podemos particionar $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ según las extensiones mínimas por derecha de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ que ocurren. Si esta partición tiene más de un elemento, decimos que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es un *contexto ramificable por la derecha*. O sea que ocurren $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}a)$ y $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}b)$, con $a \neq b$. Es fácil ver que si esto sucede, $\mathbf{s}_\mathbf{r}$ es un nodo interno del árbol de sufijos de $\mathbf{y}_\mathbf{r}$. De manera similar se define *contexto ramificable por la izquierda* y se ve que $\mathbf{s}_\ell$ es nodo interno de $\mathbf{y}_\ell$.

Lema 4.18. *Si $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es ramificable por izquierda (resp. derecha) entonces $\mathbf{s}_\mathbf{r}$ (resp. $\mathbf{s}_\ell$) no es sufijo.*

Demostración. Si $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es ramificable ocurre al menos dos veces. Pero si $\mathbf{s}_\mathbf{r}$ fuese sufijo estaría en correspondencia uno a uno con una posición de $\mathbf{x}$ porque termina en $\#$. Por lo tanto no puede ser sufijo. □

Como se mencionó en el caso uni–direccional, un nodo en un árbol de sufijos identifica unívocamente una clase $\mathcal{C}$ de Π. Vamos a extender esa noción al caso bi–direccional.

Definición 4.19. Un contexto ramificable tanto por izquierda como por derecha se denomina *doblemente ramificable*, o *bi–ramificable*.

Un contexto bi–ramificable es prefijo de dos contextos bi–direccionales en los que es ramificable por izquierda y por derecha a la vez, como se ve en el siguiente lema.

Lema 4.20. *El contexto* $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ *es bi–ramificable si y sólo si en la cadena de entrada* $\mathbf{x}$ *ocurren contextos* $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} b)$ *y* $(\mathbf{s}_\ell c, \mathbf{s}_\mathbf{r} d)$ *que cumplan* $a, b, c, d \in \mathcal{A}$*,* $a \neq c$ *y* $b \neq d$*.*

Demostración. La dirección *si* de la proposición se cumple por la Definición 4.19. Para demostrar la dirección *sólo si* veamos que por definición y por el Lema 4.18 como $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es ramificable por izquierda ocurren los bi–contextos $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} b)$ y $(\mathbf{s}_\ell a', \mathbf{s}_\mathbf{r} b')$, con $a \neq a'$. Si, además $b \neq b'$, la proposición queda probada. Por lo tanto suponemos $b = b'$.

Como $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es también ramificable por derecha, ocurren los bi–contextos $(\mathbf{s}_\ell c', \mathbf{s}_\mathbf{r} d')$ y $(\mathbf{s}_\ell c, \mathbf{s}_\mathbf{r} d)$, con $d \neq d'$. Por la misma razón que antes, suponemos $c = c'$.

Como $a \neq a'$, c es diferente de al menos uno de ellos. Sin pérdida de generalidad, suponemos $a \neq c$. Por la misma razón, b debe ser diferente a al menos uno de $\{d, d'\}$, y sin pérdida de generalidad elegimos $b \neq d$. De esta manera queda probada la proposición. □

El conjunto de extensiones bi–ramificables de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ se denota por $\mathcal{B}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. Y $\mathcal{B}$ denota a todos los bi–ramificables, o sea $\mathcal{B}(\epsilon, \epsilon)$. Por lo que hemos visto, $\mathcal{B}$ es un subconjunto del producto cartesiano de los nodos internos de los árboles de sufijos correspondientes a $\mathbf{y}_\ell$ y $\mathbf{y}_\mathbf{r}$.

Una extensión mínima $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$, que ocurre, de un contexto bi–ramificable $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ no puede pertenecer a la misma clase que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, porque el conjunto $\mathcal{S}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ está contenido de manera estricta en $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. Por lo tanto los contextos de $\mathcal{C}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ deben ser prefijos de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. En el Lema 4.17 habíamos establecido que lo que después denominamos representativo es la extensión de todos los contextos de la clase. Esto sugiere que los contextos bi–ramificables son representativos. Esto quedará demostrado con los Lemas 4.23 y 4.24.

Ejemplo 4.21. Siguiendo con el Ejemplo 4.3, $(\texttt{01}, \texttt{010})$ es el representativo de su clase y podemos ver que ocurren sus extensiones $(\texttt{010}, \texttt{0101})$ en la posición 7 y $(\texttt{011}, \texttt{0100})$ en la posición 5.

El Lema 4.23 relaciona contextos representativos con bi–sufijos y contextos bi–ramificables: $\mathsf{R} \subseteq \mathcal{S} \cup \mathcal{B}$.

Antes vemos un lema previo.

Lema 4.22. *El prefijo común más largo de un conjunto de contextos uni–direccionales es el prefijo común más largo de al menos un par de los contextos del conjunto.*

Demostración. Alcanza con mostrar un ejemplo: el par formado por el primero y el último en orden lexicográfico de los contextos del conjunto. □

Lema 4.23. *Un contexto representativo es o bien bi–sufijo, o bien bi–ramificable.*

Demostración. Un contexto representativo es la extensión común más corta de los elementos de una clase $\mathcal{C}$ de Π y el prefijo común más largo de $\mathcal{S}(\mathcal{C})$. Sea $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ el contexto representativo de $\mathcal{C}$.

Si $\mathcal{S}(\mathcal{C})$ tiene un solo bi–sufijo, entonces $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es ese bi–sufijo por ser trivialmente el prefijo común más largo. Si, en cambio, más de un bi–sufijo pertenece a $\mathcal{S}(\mathcal{C})$, entonces, por el Lema 4.22 hay al menos un par de bi–sufijos de $\mathcal{S}(\mathcal{C})$ para los cuales el prefijo común más largo de sus componentes izquierdos es $\mathbf{s}_\ell$. Entonces, como $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es prefijo de ambos bi–sufijos, es ramificable por izquierda. De manera similar se ve que es ramificable por derecha, por lo que es bi–ramificable. □

El Lema 4.23 establece una condición necesaria para que un contexto sea representativo. Del Lema 4.24 se deduce que sólo un contexto en cada clase satisface esta condición. Por lo tanto la condición es suficiente.

Lema 4.24. *Sean* $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ *y* $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ *dos contextos diferentes, cada uno de ellos bi–sufijo o bi–ramificable. Entonces,* $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \neq \mathcal{S}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$.

Demostración. Si las extensiones de los dos contextos son conjuntos disjuntos la proposición se cumple. En otro caso, o bien $\mathbf{t}_\ell$ es prefijo de $\mathbf{s}_\ell$, o bien $\mathbf{s}_\ell$ es prefijo de $\mathbf{t}_\ell$. Algo similar ocurre con los contextos derechos. Como los dos contextos son diferentes, en alguno de los dos casos la relación es estricta. Sin pérdida de generalidad supongamos $\mathbf{t}_\ell \prec \mathbf{s}_\ell$. Esto significa que $\mathbf{t}_\ell\, a \preceq \mathbf{s}_\ell$ para algún $a \in \mathcal{A}$. Además $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ no puede ser bi–sufijo y por lo tanto es bi–ramificable. Esto significa que debe ocurrir $(\mathbf{t}_\ell b, \mathbf{t}_\mathbf{r})$, con $b \neq a$. Por lo tanto $\mathcal{S}(\mathbf{t}_\ell b, \mathbf{t}_\mathbf{r})$ no es vacío, y es disjunto con $\mathcal{S}(\mathbf{t}_\ell a, \mathbf{s}_\mathbf{r})$, de donde $\mathcal{S}(\mathbf{t}_\ell b, \mathbf{t}_\mathbf{r}) \cap \mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = \emptyset$, porque $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \subseteq \mathcal{S}(\mathbf{t}_\ell a, \mathbf{s}_\mathbf{r})$. Y como $\mathcal{S}(\mathbf{t}_\ell b, \mathbf{t}_\mathbf{r}) \subset \mathcal{S}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ se concluye $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \neq \mathcal{S}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$.

□

Los Lemas 4.17, 4.23, y 4.24 implican el siguiente teorema, que establece $\mathsf{R} = \mathcal{S} \cup \mathcal{B}$.

Teorema 4.25. *El conjunto de contextos representativos es la unión del conjunto de bi–sufijos y el conjunto de contextos bi–ramificables.*

4.3.1. Conjunto óptimo

Ahora aplicamos la noción de contexto representativo para la búsqueda de un conjunto óptimo de contextos para una función de peso dada. En la Sección 3.2, habíamos dado la Definición 3.8 de conjunto válido relativo a un contexto. Vamos a adaptarla para tratar sólo con contextos representativos. Es evidente que uno de los conjuntos válidos de contextos relativo a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es el conjunto de sus extensiones mínimas izquierdas, $\{\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} : a \in \mathcal{A}\}$. Algunas de esas extensiones no ocurren y otras no son contextos representativos. Ahora vemos otro conjunto válido de gran utilidad. Está compuesto por los $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ que no ocurren y los representativos de los $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ que ocurren. Es decir, es la unión de

$$\{(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) : a \in \mathcal{A}, \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) = \emptyset\} \text{ y } \{\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) : a \in \mathcal{A}, \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) \neq \emptyset\}\,.$$

El primero de los conjuntos no ocurre, por lo que no hay bi–sufijos que sean extensión suya. Si $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ ocurre, entonces $\mathcal{S}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ no es vacío y es igual a $\mathcal{S}(\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}))$. Y cada contexto de $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es extensión de exactamente un elemento del segundo conjunto. Esto motiva a dar la siguiente definición.

Definición 4.26. Un conjunto de contextos $\mathcal{V}$ es *compacto relativo* a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, si es extensión de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, cada uno de sus elementos es representativo, y cada elemento de $\mathcal{S}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es extensión de exactamente un elemento de $\mathcal{V}$.

Debe quedar claro que en esta definición el contexto $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ no tiene por qué ser representativo.

Definición 4.27. Un conjunto de contextos $\mathcal{V}$ es *óptimo compacto relativo a* $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ si tiene peso mínimo entre todos los conjuntos compactos relativos a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, donde el peso de un conjunto de contextos es definido como la suma de los pesos de sus elementos.

Nota 4.28. Se debe notar que dado un conjunto óptimo compacto relativo a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ se puede construir un conjunto que lo contenga y sea óptimo relativo a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ (según la Definición 3.8 y siguientes). Cada contexto que se agrega no ocurre y por lo tanto su vector de conteos es nulo así como también lo es su peso.

Teorema 4.29. *Sea $\mathcal{C}$ una clase de Π. Entonces, un conjunto óptimo compacto de contextos relativo a $\mathsf{R}(\mathcal{C})$ es también óptimo compacto relativo a cualquier otro contexto en $\mathcal{C}$.*

Demostración. Alcanza con ver que cualquier conjunto compacto relativo a cualquier $(\mathbf{s}_\ell, \mathbf{s_r})$ perteneciente a $\mathcal{C}$ es también compacto relativo a $\mathsf{R}(\mathcal{C})$. Y esto se cumple porque cualquier contexto representativo que sea extensión de $(\mathbf{s}_\ell, \mathbf{s_r})$ también lo es de $\mathsf{R}(\mathcal{C})$. □

De este teorema y de la Nota 4.28 se deriva el siguiente corolario.

Corolario 4.30. *El problema de encontrar un conjunto óptimo de contextos es equivalente a encontrar un conjunto óptimo compacto.*

Debe notarse que un conjunto óptimo compacto es, según la Definición 4.26 un subconjunto de los contextos representativos, que a su vez, según el Teorema 4.25 son los contextos bi–sufijos y bi–ramificables.

Bases para el nuevo algoritmo Ahora vamos a reformular expresiones basándonos en el actual marco de sólo trabajar con contextos representativos.

De la Ecuación (3.3) se sabe que el peso óptimo de un contexto $(\mathbf{s}_\ell, \mathbf{s_r})$ se obtiene del mínimo entre tres valores: la función $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r})$, la suma de los pesos óptimos de las extensiones mínimas izquierdas y la suma de los pesos óptimos de las extensiones mínimas derechas. Para cada contexto representativo $(\mathbf{s}_\ell, \mathbf{s_r})$, un conjunto válido relativo a él, trivial, es el propio $(\mathbf{s}_\ell, \mathbf{s_r})$. Ahora definimos otros dos conjuntos válidos relativos a $(\mathbf{s}_\ell, \mathbf{s_r})$.

Definición 4.31. El *conjunto válido mínimo por izquierda (resp. derecha) relativo* a $(\mathbf{s}_\ell, \mathbf{s_r})$ consiste en los contextos representativos de las extensiones mínimas izquierdas (resp. derechas) que ocurren.

O sea, son los conjuntos

$$\{\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s_r}) : a \in \mathcal{A}, \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s_r}) \neq \emptyset\} \tag{4.3}$$

y

$$\{\mathsf{R}(\mathbf{s}_\ell, \mathbf{s_r} a) : a \in \mathcal{A}, \mathcal{I}(\mathbf{s}_\ell, \mathbf{s_r} a) \neq \emptyset\}\,. \tag{4.4}$$

Primero vamos a ver que el vector de conteos de $(\mathbf{s}_\ell, \mathbf{s_r})$ se puede calcular sumando los vectores de conteos de cualquiera de estos conjuntos. La Ecuación (3.4) establece que el vector de conteos es la suma de los vectores de conteos de las extensiones mínimas izquierdas o derechas.

Esa suma se puede separar en dos, una con las extensiones que no ocurren y otra con las que ocurren:

$$\mathbf{c}(\mathbf{s}_\ell, \mathbf{s_r}) = \sum_{a \in \mathcal{A}} \mathbf{c}(\mathbf{s}_\ell a, \mathbf{s_r}) = \sum_{\substack{a \in \mathcal{A}, \\ \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s_r}) = \emptyset}} \mathbf{c}(\mathbf{s}_\ell a, \mathbf{s_r}) + \sum_{\substack{a \in \mathcal{A}, \\ \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s_r}) \neq \emptyset}} \mathbf{c}(\mathbf{s}_\ell a, \mathbf{s_r}).$$

Como los vectores de los contextos que no ocurren son nulos, la primera suma es 0. El vector de conteos de un contexto es igual al de su representativo porque ocurren en las mismas posiciones. Por lo tanto en la segunda suma los contextos pueden reemplazarse por sus representativos. Entonces (y procediendo de manera similar con las extensiones mínimas por derecha) se llega a una nueva expresión para calcular el vector de conteos de un contexto si se conoce los vectores de conteos de sus extensiones.

$$\mathbf{c}(\mathbf{s}_\ell, \mathbf{s_r}) = \sum_{\substack{a \in \mathcal{A}, \\ \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s_r}) \neq \emptyset}} \mathbf{c}(\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s_r})) = \sum_{\substack{a \in \mathcal{A}, \\ \mathcal{I}(\mathbf{s}_\ell, \mathbf{s_r} a) \neq \emptyset}} \mathbf{c}(\mathsf{R}(\mathbf{s}_\ell, \mathbf{s_r} a)). \tag{4.5}$$

Finalmente adaptamos la optimalidad expresada en (3.3). El razonamiento es similar al anterior. Cada una de las sumas se separa en dos. Una incluye las extensiones mínimas que no ocurren, que tienen peso óptimo 0, y por lo tanto esa suma es 0. La otra suma incluye las extensiones que ocurren. Del Teorema 4.29 deducimos que el peso óptimo de un contexto es igual al de su representativo. Por lo tanto hacemos la sustitución de cada extensión mínima con su representativo. Por otro lado, el cálculo de $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r})$ toma como parámetro el vector de conteos, que vimos que se calcula correctamente, por lo que no es afectado por la restricción de sólo tratar con contextos representativos. Entonces, definiendo

$$\begin{aligned} N &= \mathtt{w}(\mathbf{s}_\ell, \mathbf{s_r}) \\ L &= \sum_{\substack{a \in \mathcal{A}, \\ \mathcal{I}(\mathbf{s}_\ell a, \mathbf{s_r}) \neq \emptyset}} \mathtt{w}_{opt}(\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s_r})) \\ R &= \sum_{\substack{a \in \mathcal{A}, \\ \mathcal{I}(\mathbf{s}_\ell, \mathbf{s_r} a) \neq \emptyset}} \mathtt{w}_{opt}(\mathsf{R}(\mathbf{s}_\ell, \mathbf{s_r} a)) \end{aligned}$$

llegamos a la expresión de optimalidad entre contextos representativos:

$$\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s_r}) = \text{mín}\left\{N, L, R\right\}. \tag{4.6}$$

Y entonces se puede decidir cuál es la partición que debe aplicarse a $(\mathbf{s}_\ell, \mathbf{s_r})$ para obtener un conjunto óptimo relativo a él.

$$\mathbf{split}(\mathbf{s}_\ell, \mathbf{s_r}) = \mathbf{argmin}\left\{N, L, R\right\} \tag{4.7}$$

donde la función **argmin** devuelve 'N', 'L' o 'R' si el mínimo de $\{N, L, R\}$ es N, L o R respectivamente y resolviendo los empates de alguna manera determinística.

Capítulo 5

Subárboles Compactos

En este capítulo vamos a describir un método eficiente para hallar todos los contextos representativos correspondientes a una secuencia $\mathbf{x}$. El método utiliza los árboles de sufijos de las cadenas formadoras de contextos. Comienza obteniendo los bi–sufijos y luego recorre uno de los árboles en post-orden, obteniendo para cada nodo todos los nodos del otro árbol con los cuales forma un contexto representativo. Se verá que la cantidad de contextos representativos es $O(n \cdot h)$, siendo h el mínimo de las alturas de los dos árboles y n el largo de la secuencia $\mathbf{x}$.

Por cada nodo del árbol recorrido se encontrará un conjunto de nodos del otro árbol, al que se le puede dar una estructura de árbol. Se van a definir operaciones entre estos árboles. En el Capítulo 6 se define una clase de grafos y se describe su construcción en la Sección 6.2. Esa construcción puede verse como una generalización de las operaciones sobre árboles que aquí se definen.

Vamos a dar una definición de `lca`, el ancestro común más profundo de los nodos de un árbol, que es una importante operación entre nodos de un árbol. En el Apéndice A veremos que se puede calcular en tiempo constante.

5.1. Árboles

En esta sección trataremos los árboles de manera estructural, sin imponer ningún tipo de etiquetas en las ramas. Vamos a definir dos operaciones entre árboles que en la Sección 5.2 serán usadas para encontrar los contextos representativos.

Sea T un árbol. Denotamos con $\mathbf{V}(T)$ al conjunto de los nodos del árbol, que es la unión disjunta del conjunto de sus nodos internos, denotado $\mathbf{B}(T)$, y el conjunto de sus hojas, denotado $\mathbf{L}(T)$. Si T es el árbol vacío, cada uno de estos conjuntos es vacío. Si el nodo v es ancestro del nodo w lo denotamos por $v \preceq w$, y en otro caso por $v \not\preceq w$. Si w es hijo de v la rama que conecta v y w se denota

$\overrightarrow{vw}$, tal como se hizo al definir árboles en el Capítulo 2, aunque en esta sección las ramas no están etiquetadas. La función $\mathbf{root}(T)$ identifica el nodo que es la raíz del árbol T. Y $\mathbf{children}(T, v)$, con $v \in \mathbf{V}(T)$, es el conjunto de nodos de T que son los hijos del nodo v. El árbol vacío se representa con $\perp$.

Un *árbol compacto* es un árbol en que cada nodo interno tiene dos o más hijos. En esta tesis sólo se consideran árboles compactos.

5.1.1. Subárboles Compactos

Definición 5.1. Sean v y w nodos de un árbol B, con $v \preceq w$. Se dice que $\overset{\rightsquigarrow}{vw}$ es la *compresión del camino* $\overrightarrow{u_1u_2}\overrightarrow{u_2u_3}\cdots\overrightarrow{u_{n-1}u_n}$ de B, donde $v = u_1, w = u_n$, y $\overrightarrow{u_iu_{i+1}}$ es una rama de B, con $1 \leq i \leq n-1$.

Definición 5.2. Los caminos $\overset{\rightsquigarrow}{vy}$ y $\overset{\rightsquigarrow}{vz}$ de un árbol son disjuntos si no tienen ramas en común.

Esto significa que si $\overset{\rightsquigarrow}{vy}$ y $\overset{\rightsquigarrow}{vz}$ son disjuntos, entonces $\overset{\rightsquigarrow}{vy} = \overrightarrow{vu_2}\cdots\overrightarrow{u_hy}$ y $\overset{\rightsquigarrow}{vz} = \overrightarrow{vw_2}\cdots\overrightarrow{w_kz}$, con $u_2 \neq w_2$.

Definición 5.3. Un *subárbol compacto*, T, del árbol compacto B, al que denominamos *árbol base*, es un árbol compacto cuyos nodos pertenecen a $\mathbf{V}(B)$, las ramas son caminos de B, y todo par de ramas salientes de un nodo de T son disjuntas al ser consideradas como caminos de B.

Un ejemplo se ve en la Figura 5.1.

Según la definición, las ramas del subárbol no tienen por qué ser ramas del árbol base. Como ejemplo, en la Figura 5.1, la rama $\overrightarrow{\gamma 5}$ en T es la compresión del camino $\overrightarrow{\gamma\delta}\ \overrightarrow{\delta 5}$ en B.

De la definición de subárbol compacto se deduce que los hijos de un nodo u en el subárbol son descendientes de hijos diferentes del mismo nodo u en el árbol base.

Entonces, formalmente T es un subárbol compacto de B si y sólo si

$\mathbf{V}(T) \subseteq \mathbf{V}(B)$,
Si $v \in \mathbf{B}(T)$ entonces $|\mathbf{children}(T, v)| \geq 2$,
Si $v, w \in \mathbf{children}(T, u), v \neq w$ entonces
$\exists\, v', w' \in \mathbf{children}(B, u), v' \neq w'$, tal que $v' \preceq v, w' \preceq w$.

Denotamos con B^C a la colección de subárboles compactos del árbol base B.

Si u es un nodo de B, con $B(u)$ denotamos el subárbol compacto de B formado por todos los nodos de B que descienden de u. Obviamente, la raíz de este subárbol es u.

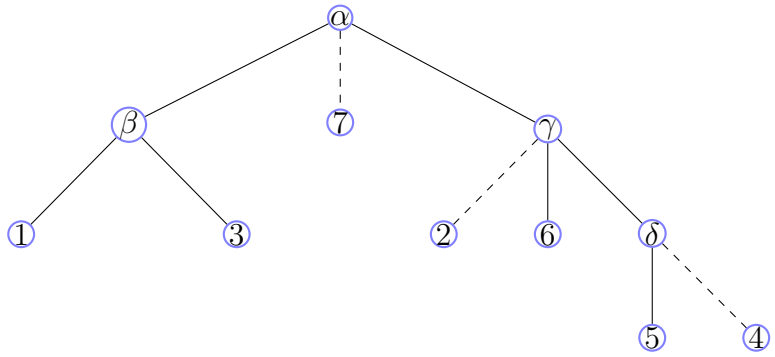


Figura 5.1: Subárbol compacto. En trazo continuo se ve un subárbol compacto sobre un árbol base que se completa en trazo discontinuo. Las hojas 7,2 y 4 y el nodo interno δ del árbol base no son nodos del subárbol. Hay que notar que si δ perteneciera al subárbol, este no sería compacto. La rama del subárbol que va desde γ hasta 5 es la compresión del camino que va desde γ hasta δ y desde δ hasta 5 en el árbol base.

Sea T un subárbol compacto de B y v un nodo de B que es descendiente de la raíz de T. Más formalmente, $T \in B^C$, $v \in \mathbf{V}(B)$, $\mathbf{root}(T) \preceq v$. La definición de $T(v)$ es igual a la de $B(v)$, es decir, es el subárbol formado por los nodos de T que descienden de v. Es posible que v no pertenezca a T, y esto genera dos casos. En uno de ellos ningún nodo de T desciende de v, por lo que $T(v)$ es $\perp$. En el otro caso todos los nodos de T que descienden de v lo hacen por la misma rama de T, siendo la raíz de $T(v)$ el menos profundo de esos nodos.

Ejemplo 5.4. En la Figura 5.1 $T(\beta)$ es igual a $B(\beta)$. $T(\gamma)$ es el subárbol con los nodos γ, 6 y 5. Los dos casos en los que v no pertenece a T se ejemplifican con $T(\delta)$, que es solamente la hoja 5 y $T(4)$ que es $\perp$.

5.1.2. LCA

El `lca` de dos nodos de un árbol es el ancestro común de ambos que está a mayor profundidad. De manera obvia extendemos la definición de `lca` a un conjunto de nodos.

Definición 5.5. El `lca` ("lowest common ancestor") de un conjunto de dos o más nodos de un árbol T es el nodo que está a mayor profundidad en el árbol entre los que son ancestros a todos los nodos del conjunto.

Formalmente,

$$\begin{aligned}
&\mathbf{lca} : \mathbf{set}(\mathbf{V}(T)) \to \mathbf{B}(T)\\
&\text{Precondición} : |S| \geq 2,\\
&\mathbf{lca}(S) \preceq S,\\
&v \in \mathbf{V}(T), v \preceq S \Rightarrow\ v \preceq \mathbf{lca}(S).
\end{aligned}$$

Ejemplo 5.6. En la Figura 5.1 β es el `lca` de las nodos 1 y 3; α es el `lca` de los nodos 1, 7 y γ; γ es el `lca` de 2, γ y 6.

Nota 5.7. El `lca` de dos elementos del conjunto S no necesariamente es el `lca` del conjunto. Lo que sí se cumple es que hay al menos un par de elementos de S cuyo `lca` es el `lca` del conjunto. Esto es trivial si $\mathbf{lca}$(S) pertenece a S. En el otro caso, si no fuera así significaría que todos los nodos del conjunto descienden de $\mathbf{lca}$(S) por la misma rama, por lo que el hijo de $\mathbf{lca}$(S) según esa rama sería un ancestro común más profundo de los nodos de S, en contra de lo supuesto. Un ejemplo es el par formado por el primero y el último de los nodos de S en una recorrida euleriana ("depth first"). Esto tiene importancia desde el punto de vista de la eficiencia porque seleccionando adecuadamente un par de elementos se puede obtener el `lca` de todo el conjunto. Aunque en esta sección los nodos de los árboles no están asociados a contextos se debe notar la relación con el Lema 4.22.

Ejemplo 5.8. El `lca` de los nodos 1, 3 y γ es α que también es el `lca` de 1 y γ (aunque no es el `lca` de los nodos 1 y 3); el `lca` de 2, γ y 6 es γ que es el `lca` de 2 y 6.

5.1.3. LCA generalizado

El `lca` es un nodo. Según lo que se estableció en la Nota 5.7 puede haber más de un nodo que sea `lca` de algún par de nodos del conjunto S. Por lo tanto podemos definir una nueva operación que generalice `lca` y devuelva el conjunto de nodos que son `lca` de algún par de esos nodos. En el caso particular en que el `lca` de cada par de nodos de S sea el mismo la nueva operación es la operación `lca` ya definida.

Vamos a extender esa generalización en otro aspecto. El `lca` se define sobre el conjunto de nodos. Pero como un nodo es la raíz de un subárbol, la definición podría haberse hecho sobre el conjunto de subárboles. Al considerarlo de esta manera la generalización de la operación puede obtener no sólo él o los `lca` de las raíces, sino el de todos los posibles pares de nodos tomados de diferentes subárboles.

Ejemplo 5.9. En el árbol base de la Figura 5.1 consideremos los subárboles definidos por los nodos $\{\alpha, 3, 6\}$ y $\{\delta, 4, 5\}$. La generalización del `lca` es el conjunto $\{\alpha, \gamma\}$. Esto es porque el `lca` de los nodos α o 3 con cualquiera de los del otro conjunto es α, mientras que el `lca` del nodo 6 con los otros es γ.

Aquí generalizamos el concepto `lca` para considerar una colección de conjuntos de nodos.

Definición 5.10. El *`lca` generalizado*, `glca`, de una colección de conjuntos de nodos de un árbol es el conjunto de nodos formado por el `lca` de todos los pares de nodos tomados de distintos conjuntos. Si hay menos de dos conjuntos de nodos, el resultado es $\emptyset$.

Nota 5.11. Un mismo nodo v puede pertenecer a varios conjuntos de la colección. En ese caso, $v = \mathbf{lca}(v, v)$ pertenece al `glca`.

Lo que nos interesa es la restricción de esta operación cuando cada conjunto de nodos tiene estructura de subárbol compacto. La importancia de restringir el `glca` a subárboles compactos proviene de que, como veremos en la Sección 5.2.1, con esta operación se obtienen los contextos representativos. Sólo trataremos con subárboles compactos cuyas hojas son hojas del árbol base, y además, los conjuntos de hojas son disjuntos por pares.

La siguiente es la especificación de `glca` cumpliendo estas restricciones:

$$\mathbf{glca} : \mathbf{set}(B^C) \to \mathbf{set}(\mathbf{B}(B))$$

Precondiciones :

$\mathbf{L}(T) \subseteq \mathbf{L}(B)$, para todo $T \in S$,

$\mathbf{L}(T_1) \cap \mathbf{L}(T_2) = \emptyset$, con $T_1, T_2 \in S, T_1 \neq T_2$,

$$\mathbf{glca}(S) = \begin{cases} \emptyset & \text{si } |\{T \neq \perp : T \in S\}| < 2 \\ V & \text{en otro caso, donde} \\ & v \in V \Leftrightarrow \exists\, v_i \in \mathbf{V}(T_i), v_j \in \mathbf{V}(T_j) \text{ tal que } v = \mathbf{lca}(v_i, v_j), \\ & \qquad \text{con } T_i, T_j \in S, T_i \neq T_j. \end{cases}$$

Es posible obtener el `glca` de manera eficiente. Primero se puede ver que no es necesario obtener el `lca` de cada par de nodos. Se obtiene el mismo resultado restringiendo la operación a los pares de hojas, ya que la última expresión puede sustituirse por

$$v \in V \Leftrightarrow \exists\, l_i \in \mathbf{L}(T_i), l_j \in \mathbf{L}(T_j) \text{ tal que } v = \mathbf{lca}(l_i, l_j), \\ \text{con } T_i, T_j \in S, T_i \neq T_j,$$

porque para $v_i \in \mathbf{V}(T_i)$ existe $l_i \in \mathbf{L}(T_i)$ que cumple $v_i \preceq l_i$ y para $v_j \in \mathbf{V}(T_j)$ existe $l_j \in \mathbf{L}(T_j)$ que cumple $v_j \preceq l_j$, y $\mathbf{lca}(v_i, v_j) = \mathbf{lca}(l_i, l_j)$.

Aún así, ese procedimiento es ineficiente porque algunos nodos del `glca` son el `lca` de varios pares de hojas y se obtendrían varias veces.

Vamos a demostrar que la siguiente especificación, que elimina las ineficiencias mencionadas, describe el `glca`.

$$\mathbf{glca}(S) = \begin{cases} \emptyset \quad \text{si } |S| < 2 \\ \{v\} \bigcup_{w \in \mathbf{children}(B,v)} \mathbf{glca}\left\{T(w) : T \in S, T(w) \neq \perp\right\} \\ \qquad \text{en otro caso, donde } v = \mathbf{lca}\left\{\mathbf{root}(T) : T \in S\right\}. \end{cases} \tag{5.1}$$

Lema 5.12. *La Expresión* (5.1) *es el* `glca` *del conjunto* S *de subárboles compactos del árbol base* B.

Demostración. Está claro que todo nodo obtenido usando esta especificación pertenece al `glca` porque es el resultado de aplicar el `lca` a nodos de diferentes subárboles. Para ver el recíproco, esto es, que el `lca` de todo par de hojas de diferentes subárboles pertenece al conjunto obtenido, sea $z = \mathbf{lca}(l_i, l_j)$, donde l_i y l_j son respectivamente hojas de los subárboles T_i y T_j de S, con $T_i \neq T_j$. Hay dos casos posibles. Uno es que z sea igual a v. Si es así el `lca` de las dos hojas pertenece al resultado obtenido. En el otro caso, z pertenece a $B(w)$ para algún w en $\mathbf{children}(B, v)$, y $T_i(w)$ y $T_j(w)$ son no vacíos. Entonces, l_i y l_j son hojas de diferentes subárboles en la obtención del `glca` de los subárboles que descienden de w. □

Esta especificación induce de manera directa un algoritmo recursivo para obtener el `glca`. Como se ve, cada nodo del `glca` se obtiene sólo una vez, por lo que, la complejidad del algoritmo, medida en cantidad de veces que se invoca la función `lca` es igual a la cantidad de nodos del `glca`. En el Apéndice A se verá que el cálculo del `lca` se puede hacer en $O(1)$. Como consecuencia, la complejidad de obtener el `glca` es $O(|\texttt{glca}|)$.

Para consideraciones de búsqueda de eficiencia en los algoritmos notemos algunos casos particulares de $T(w)$ en la Ecuación (5.1).

$$T(w) = \begin{cases} T & \text{si } w \preceq \mathbf{root}(T), \\ \perp & \text{si } w' \preceq \mathbf{root}(T) \\ & \quad \text{para algún } w' \in \mathbf{children}(B, v), w' \neq w, \\ \neq T, \text{ tal vez } \perp & \text{si } v = \mathbf{root}(T). \end{cases}$$

En el tercer caso la raíz del árbol T es el `glca` de S. En ese caso es incierto en qué contribuye T a la llamada recursiva. El aporte es un subárbol estricto de T, tal vez $\perp$. Si, en cambio, $v \neq \mathbf{root}(T)$ estamos en los otros dos casos. En el primero, el árbol completo es descendiente de w por lo que todos sus nodos quedan involucrados en los cálculos. En el segundo caso ningún nodo de T desciende de w.

5.1.4. Árbol soporte

Un subárbol compacto del árbol base B queda determinado por sus hojas. Esto es porque cualquier nodo interno es el `lca` de algún par de hojas que descienden de él según ramas diferentes. Recíprocamente, el `lca` de cualquier par de hojas es un nodo interno. El subárbol es la clausura del conjunto de hojas bajo la operación `lca`. Esto es, empezando con un conjunto que contiene las hojas, se obtiene el `lca` de un par de elementos del conjunto y el resultado se agrega al conjunto. El proceso termina cuando todo posible par da como resultado un elemento que ya está en el conjunto. Entonces, es el mínimo subárbol compacto que contiene todo el conjunto de hojas. No cualquier conjunto de nodos de B puede ser un conjunto de hojas de un subárbol compacto. Nos limitaremos a subárboles cuyas hojas sean hojas de B. Con esta restricción vamos a dar la definición de árbol soporte.

Definición 5.13. El *árbol soporte* de un conjunto de hojas L del árbol base B, es el subárbol compacto de B cuyas hojas son exactamente L.

Podemos interpretar el árbol soporte como una función sobre conjuntos de hojas:

$$\begin{aligned}&\mathbf{soporte} : \mathbf{set}(\mathbf{L}(B)) \to B^C\\ &\mathbf{soporte}(L) = T \text{ tal que}\\ &\mathbf{V}(T) = \bigcup_{l_i, l_j \in L} \mathbf{lca}(l_i, l_j)\end{aligned}$$

Se cumple que $\mathbf{L}(T)$ es L. Estas hojas se obtienen cuando $l_i = l_j$. Si L es vacío el resultado es el árbol vacío.

Ejemplo 5.14. En la Figura 5.1 lo que se ve en trazo continuo es el árbol soporte de las hojas 1, 3, 6 y 5.

5.1.5. Unión de subárboles

Para el proceso de obtención de los contextos representativos vamos a definir otra operación entre subárboles compactos. Esta definición está restringida a

subárboles para los cuales el conjunto de hojas de todos ellos satisface la propiedad de que dado cualquier par de esas hojas de los subárboles, al ser consideradas como nodos del árbol base, ninguna es ancestro de la otra. Esto es, para todo par v_1, v_2 de hojas de los subárboles se cumple que en B, $v_1 \not\preceq v_2$, $v_2 \not\preceq v_1$. Esto se cumple con las restricciones que hemos impuesto, esto es, que las hojas de los subárboles son hojas del árbol base y los conjuntos de hojas de cada par de subárboles son disjuntos.

Definición 5.15. Llamamos *unión de subárboles compactos* de un árbol base al árbol soporte de la unión de los conjuntos de hojas de los subárboles.

Formalmente:

$$\begin{aligned}&\mathbf{union} : \mathbf{set}(B^C) \to B^C\\ &\text{Precondición} : |S| \geq 2\\ &\mathbf{union}(S) = \mathbf{soporte}\left(\bigcup_{T\in S} \mathbf{L}(T)\right).\end{aligned}$$

Con la condición impuesta de que las hojas de los subárboles son hojas del árbol base, esta operación es cerrada, es decir, el resultado es también un subárbol cuyas hojas son hojas del árbol base.

Como ya fue establecido al definir árbol soporte, es posible obtener la unión mediante la clausura `lca` del conjunto de hojas. Además habrá que establecer las ramas. Una primera simplificación de este algoritmo es notar que, con el `lca` de dos hojas del mismo subárbol se obtiene un nodo interno que ya se conocía. Entonces, para obtener todos los nodos de la unión basta con restringir el cálculo a hojas de diferentes subárboles. Y esto es obtener el `glca` de los subárboles. O sea que

$$\mathbf{V}(\mathbf{union}(S)) = \mathbf{glca}(S) \cup \bigcup_{T\in S} \mathbf{V}(T). \tag{5.2}$$

Con el procedimiento descrito es posible obtener repetidas veces el mismo nodo. Una versión más eficiente es una variante de la solución vista para el `glca` en la Ecuación (5.1). Se describe en el Algoritmo 2. La generalización para el caso bi–direccional, en el que se obtiene un grafo, se verá en el Algoritmo 5

Se va a demostrar la correctitud de este algoritmo.

Teorema 5.16. *El Algoritmo 2 devuelve la unión de los subárboles pertenecientes al conjunto S.*

Demostración. Por la Ecuación (5.2) hay que demostrar que la estructura devuelta, U, contiene todos los nodos del `glca` y de todos los árboles de S. Además hay que probar que la estructura de ramas es correcta.

Algoritmo 2 $\mathbf{union}(S : \mathbf{set}(B^C))$

1: **Salida:** $U : B^C$
2: **Precondición:** $|S| \geq 2$
3: $v = \mathbf{root}(U) \leftarrow \mathbf{lca}\,\{\mathbf{root}(T) : T \in S\}$
4: **Para cada** $w \in \mathbf{children}(B, v)$
5: $\quad S' \leftarrow \emptyset$
6: $\quad$ **Para cada** $T \in S$
7: $\quad\quad$ **Si** $T(w) \neq \perp$ **entonces**
8: $\quad\quad\quad S' \leftarrow S' \cup \{T(w)\}$
9: $\quad$ **Si** $S' \neq \emptyset$ **entonces**
10: $\quad\quad U(w) \leftarrow \begin{cases} T' & \text{si } S' = \{T'\} \\ \mathbf{union}(S') & \text{si } |S'| \geq 2. \end{cases}$
11: **devuelve** U

Como puede verse comparando con la Ecuación (5.1) los nodos del `glca` son los que resultan de obtener en la línea 3 el `lca` de las raíces de los subárboles en todas las llamadas recursivas de la línea 10.

Para cada $T \in S$, su raíz, **root**(T), pertenece a U si es igual a v. En otro caso **root**(T) desciende de algún hijo w de v por lo que es raíz de $T(w)$. Como tal, en la línea 10 o bien, si es el único árbol en S', es incluido en U, junto a todos sus descendientes, o bien es la raíz de uno de los árboles de la llamada recursiva, en una instancia con árboles más chicos, por lo que en alguna de las llamadas va a ser incluido en U. Volviendo al primer caso, $\mathbf{root}(T) = v$, cada hijo de **root**(T) es raíz de alguno de los $T(w)$, por lo que valen para ellos las consideraciones anteriores y serán incluidos en U.

En cuanto a la estructura de ramas, basta ver que $U(w)$, el subárbol de U, que desciende de v por la misma rama que une a v con w está formado sólo por descendientes de w, ya que por definición esto lo cumplen todos los árboles $T(w)$. □

La complejidad de este algoritmo, medida en la cantidad de veces que se invoca la función **lca**, es la misma que la de obtener el `glca`. Para ello se impone como precondición que el conjunto S tenga más de un elemento para evitar las recorridas de los árboles que ya fueron construidos.

Usaremos el árbol $T_{\mathbf{y}_\ell}$ de la Figura 4.2 como ejemplo de árbol base. En la Figura 5.2, en la parte de arriba, vemos tres subárboles de $T_{\mathbf{y}_\ell}$. Con trazo discontinuo aparece el árbol base. Dos de los subárboles son hojas de $T_{\mathbf{y}_\ell}$. La raíz de la unión es el `lca` de las raíces de los tres subárboles, que es la raíz de $T_{\mathbf{y}_\ell}$. Descendiendo de esta raíz por la rama izquierda sólo está el tercer subárbol, por lo cual se establece una rama hacia él. Nada desciende por la rama del medio. Por la

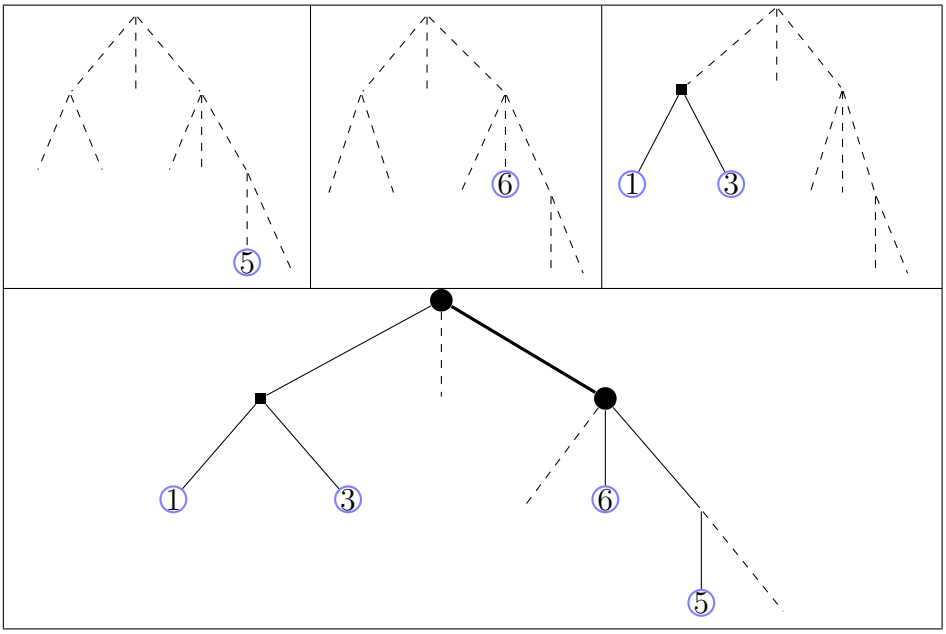


Figura 5.2: Unión de subárboles

rama derecha descienden los otros dos subárboles, por lo que hay que hacer una llamada recursiva, obtener el `lca` entre las raíces de ellos y finalmente establecer las ramas hacia ellos.

El resultado de la unión se ve en la Figura 5.2. Los nodos que son el resultado de la obtención de `lca`, que es el `glca`, se muestran con un círculo grande, y las ramas que los unen con un trazo grueso. En este ejemplo, el `glca` coincide con los nodos que pertenecen a la unión y no a alguno de los subárboles. Esto no siempre es así, ya que es posible que un elemento del `glca` sea uno de los nodos de los cuales se obtiene el `lca`, tal como se verá en el ejemplo de la Figura 5.4. Esta unión tiene 4 hojas, una cantidad menor, 3, de nodos internos y la cantidad de nodos calculados es aún menor, 2. Con esta descripción alcanzó con la obtención de dos `lca` para obtener esos nodos. Con la versión en que se calcula el `lca` de cada par de hojas provenientes de subárboles diferentes hacen falta cinco cálculos.

En general, la cantidad de nodos del `glca` es obviamente no mayor que el de nodos internos de la unión, que a su vez, por tratarse de árboles compactos, es menor que la cantidad de hojas. Esto es

$$|\mathbf{glca}(S)| < |\mathbf{L}(\mathbf{union}(S))| = \left| \bigcup_{T \in S} \mathbf{L}(T) \right|. \tag{5.3}$$

Por lo tanto, y asumiendo que el `lca` se puede calcular en tiempo constante (lo cual es posible como se verá en el Apéndice A), tenemos que la cantidad de

hojas de los subárboles es una cota superior de la complejidad de la obtención del `glca` y la unión de subárboles.

5.2. Búsqueda de contextos representativos

Volvemos al problema planteado en esta tesis y nos proponemos encontrar los contextos representativos, que sabemos que son los bi–ramificables y los bi–sufijos, $\mathsf{R} = \mathcal{B} \cup \mathcal{S}$. Además estableceremos una cota superior sobre la cantidad de esos contextos.

Los árboles de sufijos correspondientes a las cadenas de formación de contextos izquierda y derecha son, respectivamente, $T_{\mathbf{y}_\ell}$ y $T_{\mathbf{y_r}}$. Nuevamente identificamos a los nodos con los contextos que representan.

5.2.1. Obtención de los contextos representativos

El conjunto R es un subconjunto del producto cartesiano de $\mathbf{V}(T_{\mathbf{y}_\ell})$ y $\mathbf{V}(T_{\mathbf{y_r}})$. Considerando R como una relación, a cada elemento $\mathbf{s_r}$ de $\mathbf{V}(T_{\mathbf{y_r}})$ le corresponde un subconjunto de $\mathbf{V}(T_{\mathbf{y}_\ell})$ que denotamos $\mathsf{R}_L(\mathbf{s_r})$. Es el conjunto de subcadenas de $\mathbf{y}_\ell$ que forman contexto representativo con $\mathbf{s_r}$.

Si encontramos los conjuntos $\mathsf{R}_L(\mathbf{s_r})$ para todos los nodos $\mathbf{s_r}$ de $T_{\mathbf{y_r}}$ habremos encontrado todos los representativos. Sabemos que los bi–sufijos se obtienen trivialmente de una correspondencia uno a uno entre $\mathbf{L}(T_{\mathbf{y}_\ell})$ y $\mathbf{L}(T_{\mathbf{y_r}})$. Por otra parte el conjunto de los bi–ramificables, $\mathcal{B}$, es un subconjunto de $\mathbf{B}(T_{\mathbf{y}_\ell}) \times \mathbf{B}(T_{\mathbf{y_r}})$. Los vamos a obtener mediante el `glca` de subárboles compactos de $T_{\mathbf{y}_\ell}$. Los subárboles compactos los obtenemos recursivamente mediante `union` de otros subárboles compactos.

Recordemos que $T_{\mathbf{y_r}}(\mathbf{s_r})$ es el subárbol de $T_{\mathbf{y_r}}$ con raíz en $\mathbf{s_r}$. Llamamos $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ al conjunto de nodos de $T_{\mathbf{y}_\ell}$ relacionados mediante $\mathsf{R}_L(.)$ con algún nodo de $T_{\mathbf{y_r}}(\mathbf{s_r})$.

Teorema 5.17. *Para todo nodo* $\mathbf{s_r}$ *de* $T_{\mathbf{y_r}}$, $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ *es un subárbol compacto de* $T_{\mathbf{y}_\ell}$. *Si, además,* $\mathbf{s_r}$ *es un nodo interno, llamamos* S *a la colección de subárboles compactos* $T^C_{\mathbf{y}_\ell}(.)$ *correspondientes a los hijos de* $\mathbf{s_r}$ *en* $T_{\mathbf{y_r}}$. *Entonces,* $\mathsf{R}_L(\mathbf{s_r})$ *es* $\mathbf{glca}(S)$ *y* $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ *es igual a* $\mathbf{union}(S)$.

Demostración. La demostración se hace por inducción en la altura de $T_{\mathbf{y_r}}(\mathbf{s_r})$. Si la altura de $T_{\mathbf{y_r}}(\mathbf{s_r})$ es 0, esto es, si $\mathbf{s_r}$ es una hoja, entonces tanto $\mathsf{R}_L(\mathbf{s_r})$ como $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ son la hoja $\mathbf{s}_\ell$ de $T_{\mathbf{y}_\ell}$, tal que $(\mathbf{s}_\ell, \mathbf{s_r})$ es bi–sufijo, por lo que la proposición se cumple de manera trivial.

Si $\mathbf{s_r}$ es un nodo interno aceptamos como hipótesis inductiva que el teorema se cumple para todos los hijos de $\mathbf{s_r}$, ya que las alturas de los árboles de los que

son raíces son menores que la altura de $T_{\mathbf{y_r}}(\mathbf{s_r})$. Supongamos que $\mathbf{s}_\ell$ pertenece a $\mathsf{R}_L(\mathbf{s_r})$. Esto significa que $(\mathbf{s}_\ell, \mathbf{s_r})$ es bi–ramificable y esto, como consecuencia del Lema 4.20, a su vez implica que ocurren dos bi–sufijos $(\mathbf{s}_\ell a\mathbf{u}_\ell, \mathbf{s_r}b\mathbf{u_r})$, $(\mathbf{s}_\ell c\mathbf{v}_\ell, \mathbf{s_r}d\mathbf{v_r})$, con $a, b, c, d \in \mathcal{A}$, $a \neq c, b \neq d$. Por la hipótesis inductiva, el componente izquierdo del primero de estos bi–sufijos es una hoja en el subárbol compacto correspondiente al hijo de $\mathbf{s_r}$ que desciende por la rama b (que es $\mathsf{R}(\mathbf{s_r}b)$, el repesentativo en el caso uni–direccional de $\mathbf{s_r}$b). Esto es, $\mathbf{s}_\ell a\mathbf{u}_\ell \in \mathbf{L}(T^C_{\mathbf{y}_\ell}(\mathsf{R}(\mathbf{s_r}b)))$. Y por la misma razón $\mathbf{s}_\ell c\mathbf{v}_\ell \in \mathbf{L}(T^C_{\mathbf{y}_\ell}(\mathsf{R}(\mathbf{s_r}d)))$. Como $b \neq d$, $\mathsf{R}(\mathbf{s_r}b)$ y $\mathsf{R}(\mathbf{s_r}d)$ son hijos diferentes de $\mathbf{s_r}$, por lo que sus correspondientes árboles en S son diferentes. Entonces $\mathbf{s}_\ell a\mathbf{u}_\ell$ y $\mathbf{s}_\ell c\mathbf{v}_\ell$ son hojas de dos subárboles compactos diferentes de S, por lo que el `lca` de ambas, $\mathbf{s}_\ell$, pertenece al `glca` de los subárboles.

Recíprocamente, supongamos que $\mathbf{s}_\ell$ pertenece al `glca` de los subárboles compactos correspondientes a hijos de $\mathbf{s_r}$, $T^C_{\mathbf{y}_\ell}(\mathsf{R}(\mathbf{s_r}b))$ y $T^C_{\mathbf{y}_\ell}(\mathsf{R}(\mathbf{s_r}d))$, con $b \neq d$. Esto significa que $\mathbf{s}_\ell$ es el `lca` de un par de hojas, tomadas una de cada árbol. Sean estas hojas $\mathbf{s}_\ell a\mathbf{u}_\ell$ y $\mathbf{s}_\ell c\mathbf{v}_\ell$ respectivamente. Como $\mathbf{s}_\ell$ es el `lca` de las hojas se cumple que $a \neq c$. Como $\mathbf{s}_\ell a\mathbf{u}_\ell$ es una hoja de $T^C_{\mathbf{y}_\ell}(\mathsf{R}(\mathbf{s_r}b))$ hay una hoja $\mathbf{s_r}b\mathbf{u_r}$ en $T_{\mathbf{y_r}}(\mathsf{R}(\mathbf{s_r}b))$ tal que $(\mathbf{s}_\ell a\mathbf{u}_\ell, \mathbf{s_r}b\mathbf{u_r})$ es un bi–sufijo. De manera similar se llega a que $(\mathbf{s}_\ell c\mathbf{v}_\ell, \mathbf{s_r}d\mathbf{v_r})$ es un bi–sufijo. Por lo tanto, por Definición 4.19, $(\mathbf{s}_\ell, \mathbf{s_r})$ es bi–ramificable, por lo que $\mathbf{s}_\ell \in \mathsf{R}_L(\mathbf{s_r})$.

Quedó demostrado que $\mathsf{R}_L(\mathbf{s_r})$ es $\mathbf{glca}(S)$. Por definición, los nodos de$T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ son la unión de $\mathsf{R}_L(\mathbf{s_r})$ y los nodos de S. Por lo tanto, según la Ecuación (5.2), los nodos de $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ son los mismos que los de $\mathbf{union}(S)$. Al tener los mismos nodos, si a $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$ lo consideramos como un árbol compacto es el árbol soporte de las mismas hojas que $\mathbf{union}(S)$ y por lo tanto es el mismo árbol. □

El Algoritmo 3 permite obtener $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$.

Algoritmo 3 $\mathbf{subarbol\_asociado}(\mathbf{s_r} : \mathbf{B}(T_{\mathbf{y_r}}))$

Salida: $T^C_{\mathbf{y}_\ell}(\mathbf{s_r})$
$S \leftarrow \emptyset$
Para cada $\mathbf{t_r} \in \mathbf{children}(T_{\mathbf{y_r}}, \mathbf{s_r})$

$$S \leftarrow S \bigcup \begin{cases} \mathbf{t}_\ell, \text{tal que } (\mathbf{t}_\ell, \mathbf{t_r}) \text{ es bi–sufijo} & \text{si } \mathbf{t_r} \in \mathbf{L}(T_{\mathbf{y_r}}) \\ \mathbf{subarbol\_asociado}(\mathbf{t_r}) & \text{si } \mathbf{t_r} \in \mathbf{B}(T_{\mathbf{y_r}}) \end{cases}$$

devuelve $\mathbf{union}(S)$

Como conocemos los subárboles compactos correspondientes a las hojas de $T_{\mathbf{y_r}}$, si recorremos $T_{\mathbf{y_r}}$ en post-orden, y para cada nodo obtenemos la `union` de los subárboles compactos correspondientes a sus hijos, al procesar la raíz de $T_{\mathbf{y_r}}$ habremos encontrado todos los contextos representativos. Y sabemos que para obtener la unión de subárboles alcanza con obtener el `glca` y conectar nodos mediante ramas.

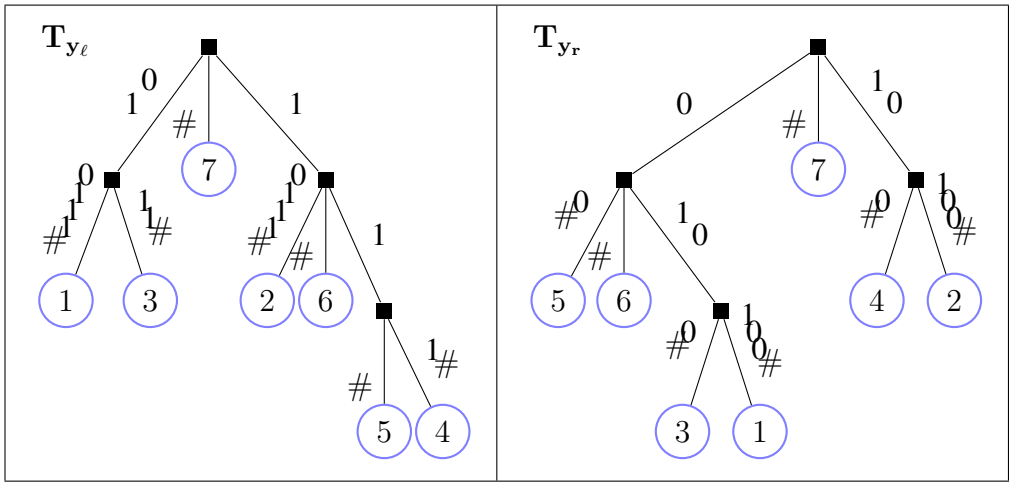


Figura 5.3: Árboles de las cadenas de formación de contextos de $\begin{pmatrix}\mathbf{y}_\ell\\ \mathbf{y}_\mathbf{r}\end{pmatrix} = \begin{pmatrix}\texttt{010111\#}\\ \texttt{010100\#}\end{pmatrix}$.

Ejemplo 5.18. Seguimos con el ejemplo de la Sección 4.2 : $\mathbf{x} = \begin{pmatrix}\texttt{1110101}\\ \texttt{0010101}\end{pmatrix}$, de donde $\begin{pmatrix}\mathbf{y}_\ell\\ \mathbf{y}_\mathbf{r}\end{pmatrix} = \begin{pmatrix}\texttt{010111\#}\\ \texttt{010100\#}\end{pmatrix}$, cuyos árboles de sufijos correspondientes repetimos en la Figura 5.3.

Para cada hoja i, $1 \leq i \leq 7$ de $T_{\mathbf{y}_\mathbf{r}}$, el subárbol compacto de $T_{\mathbf{y}_\ell}$ correspondiente es la hoja i de $T_{\mathbf{y}_\ell}$. El subárbol que corresponde al nodo $\texttt{010}$ de $T_{\mathbf{y}_\mathbf{r}}$ se obtiene mediante la unión de los subárboles correspondientes a las hojas 1 y 3, que son las hojas 1 y 3 respectivamente de $T_{\mathbf{y}_\ell}$. Esta unión tiene $\texttt{01}$ como raíz y sus dos hijos son las hojas mencionadas. El $\texttt{glca}$ es sólo la raíz, $\texttt{01}$. Entonces $(\texttt{01}, \texttt{010})$ es el único contexto bi–ramificable cuyo contexto derecho es $\texttt{010}$.

Después se procesa el nodo $\texttt{0}$ de $T_{\mathbf{y}_\mathbf{r}}$. Esto es lo que se vio en la Figura 5.2. Hay que notar que en esa figura la estructura del árbol base es la de $T_{\mathbf{y}_\ell}$, y que los tres subárboles de arriba son los correspondientes a los hijos de $\texttt{0}$ en $T_{\mathbf{y}_\mathbf{r}}$. El $\texttt{glca}$ está formada por dos nodos, entonces, por el Teorema 5.17, $\mathsf{R}_L(\texttt{0})$ es $\{\epsilon, 1\}$. Por lo tanto $(\epsilon, \texttt{0})$ y $(\texttt{1}, \texttt{0})$ también son bi– ramificables.

Con el nodo $\texttt{10}$ de $T_{\mathbf{y}_\mathbf{r}}$ se obtiene un subárbol de $T_{\mathbf{y}_\ell}$ con raíz en $\texttt{1}$ cuyos hijos son las hojas 2 y 4. El contexto $(\texttt{1}, \texttt{10})$ es el único bi–ramificable.

Finalmente se procesa la raíz de $T_{\mathbf{y}_\mathbf{r}}$. Esto se ve en la Figura 5.4.

En ese ejemplo se ve que dos de los tres nodos que se calcularon pertenecen a alguno o algunos de los subárboles que constituyen el parámetro.

En la parte de arriba están los subárboles correspondientes a los tres hijos de ϵ

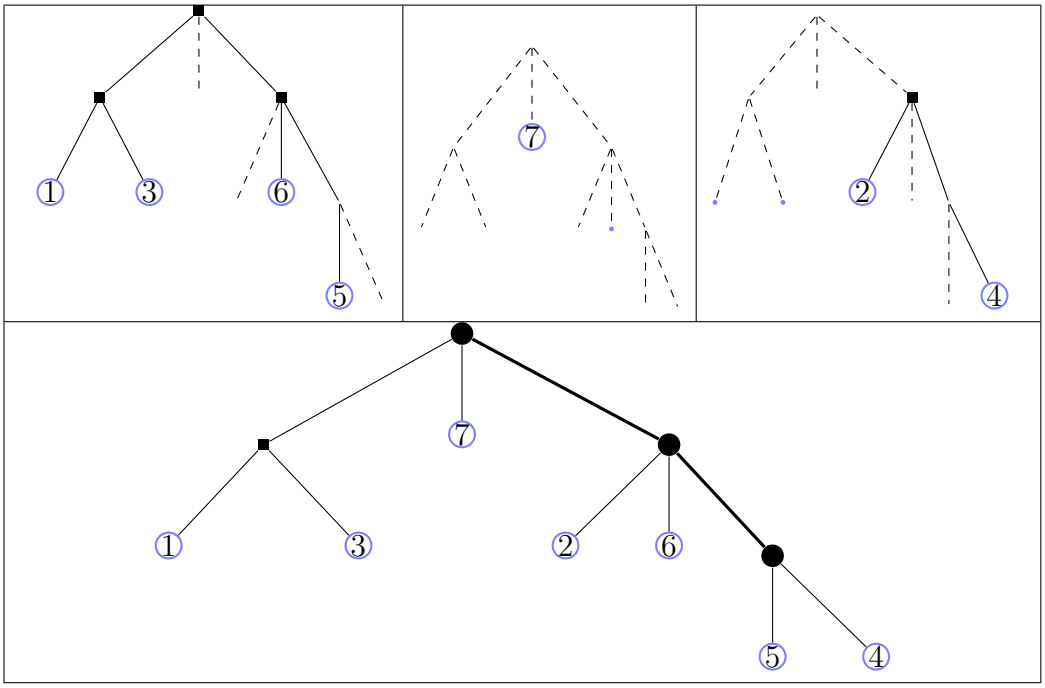


Figura 5.4: Subárbol asociado a la raíz

en $T_{\mathbf{y_r}}$. El `lca` de las raíces de los subárboles es ϵ, por lo que este nodo pertenece al `glca`, lo cual se indica en la figura con el círculo. Por la rama `0`, sólo el primer subárbol tiene descendientes, por lo tanto alcanza con establecer una rama hacia `01`, que es el hijo por esa rama. El subárbol del medio es descendiente por la rama $\#$ y los otros dos no tienen descendencia por esa rama. Finalmente el primer y tercer subárbol son los que tienen nodos que descienden de la raíz obtenida por la rama `1`. Para el primer subárbol los descendientes forman el árbol soporte de las hojas 6 y 5, mientras que al tercer subárbol corresponde el soporte de las hojas 2 y 4. El `lca` de las raíces de estos dos subárboles es el nodo `1`, que por lo tanto, pertenece al `glca`. De este nodo descienden por la rama `0` sólo la hoja 2 del tercer subárbol y por la rama $\#$ sólo la hoja 6 del primer subárbol. Pero por la rama `1` descienden hojas en ambos subárboles, la 5 del primero y la 4 del tercero. Por lo tanto se debe obtener el `lca` de ambas hojas que es `11`. Entonces se completó el conjunto $\mathcal{B}$ de bi–ramificables con (ϵ, ϵ), $(\mathtt{1}, \epsilon)$ y $(\mathtt{11}, \epsilon)$.

5.2.2. Cantidad de contextos representativos

Ahora vamos a establecer una cota acerca de la cantidad de contextos representativos.

Denotamos por $h(\cdot)$ la cantidad de niveles de un árbol.

Veremos que la cantidad de contextos representativos está acotada superiormente por $n \times h(T_{\mathbf{y_r}})$ porque en cada nivel de $T_{\mathbf{y_r}}$ hay a lo sumo n contextos representativos.

Lema 5.19. *La cantidad de contextos representativos encontrados con el método de la Sección 5.2.1 está acotada superiormente por* $n \times h(T_{\mathbf{y_r}})$.

Demostración. Notemos que por cada nodo $\mathbf{s_r}$ de $T_{\mathbf{y_r}}$ se cumple que la cantidad de contextos representativos está acotado por la cantidad de hojas que en $T_{\mathbf{y_r}}$ descienden de $\mathbf{s_r}$, esto es

$$|\mathsf{R}_L(\mathbf{s_r})| < |\mathbf{L}(T_{\mathbf{y_r}}(\mathbf{s_r}))|\,,$$

lo que se deduce de la Ecuación (5.3) ya que en el Teorema 5.17 hemos probado que $\mathsf{R}_L(\mathbf{s_r})$ es el `glca` del conjunto S de subárboles $T^C_{\mathbf{y}_\ell}(.)$ correspondientes a los hijos de $\mathbf{s_r}$, y sabemos que las hojas que descienden de $\mathbf{s_r}$ están en correspondencia uno a uno con las de los subárboles en S.

Como los conjuntos de hojas que descienden de nodos del mismo nivel de $T_{\mathbf{y_r}}$ son disjuntos, y el número de hojas en $T_{\mathbf{y_r}}$ es el largo n de la cadena, llegamos a que n es una cota superior ajustada de la cantidad de hojas que descienden de todos los nodos del nivel.

Se concluye lo propuesto al tratar cada nivel de $T_{\mathbf{y_r}}$. □

Finalmente ajustamos la cota de cantidad de contextos representativos.

Teorema 5.20. *La cantidad de contextos representativos correspondientes a la secuencia* $\mathbf{x}$ *está acotada superiormente por* $n \cdot \min\{h(T_{\mathbf{y}_\ell}), h(T_{\mathbf{y_r}})\}$*, donde* $T_{\mathbf{y}_\ell}$ *y* $T_{\mathbf{y_r}}$ *son los árboles de sufijos de las cadenas de formación de contextos izquierda y derecha respectivamente correspondientes a* $\mathbf{x}$.

Demostración. En esta sección hemos descrito la obtención de los contextos representativos mediante una recorrida de $T_{\mathbf{y_r}}$. Podríamos haber hecho algo similar recorriendo $T_{\mathbf{y}_\ell}$. Para esta recorrida se puede establecer un postulado análogo al Lema 5.19. Se obtienen entonces dos expresiones que son cota superior de la cantidad de contextos representativas, lo cual prueba el enunciado del teorema. □

En la Sección 6.4 veremos que construir un grafo con todos los contextos representativos es lo que determina la complejidad de la obtención de un conjunto óptimo de contextos, y que cada contexto representativo se obtiene en tiempo constante. Por lo tanto el resultado que aquí se obtuvo determina la complejidad del algoritmo completo.

>

Capítulo 6

Grafos Compactos de Contextos Bi–direccionales

En la Sección 3.2 se estableció un algoritmo que permite obtener un conjunto óptimo de contextos bi–direccionales para el caso de longitudes acotadas. Se hacía notar que se obtenía un árbol que de manera implícta recorría un grafo, denominado grafo de contextos bi–direccionales. En este capítulo se describe el grafo, junto con su construcción y podado, correspondiente al caso de longitudes no acotadas.

Aquel grafo tenía un vértice para cada contexto posible, incluso los que no ocurrían. Del Corolario 4.30 sabemos que podemos restringir la atención a contextos bi–ramificables y a bi–sufijos. En este capítulo definimos una clase de grafos, los grafos compactos de contextos bi–direccionales cuyos vértices corresponden a esos contextos. Entonces, el problema en cadenas de símbolos de encontrar los contextos representativos y un conjunto óptimo de contextos es trasladado al problema de construir un grafo y podarlo. El resultado del podado es un árbol compacto cuyas hojas forman un conjunto óptimo de contextos bi–direccionales.

La construcción del grafo se basa en la obtención de los contextos representativos vista en la Sección 5.2.1. En esa sección se vio que aplicando el Algoritmo 3 a cada nodo $\mathbf{s}$ de $T_{\mathbf{y_r}}$ se obtiene un conjunto de nodos de $T_{\mathbf{y}_\ell}$ que forman un subárbol compacto asociado a $\mathbf{s}$. Lo que falta para completar el grafo es establecer las aristas entre los nodos de los subárboles asociados a cada nodo de $T_{\mathbf{y_r}}$. Para ello, se verá el Algoritmo 4 que es una generalización del Algoritmo 3 mediante el cual para cada nodo de $T_{\mathbf{y_r}}$ se obtiene un grafo en lugar de un árbol.

En la Sección 6.2 se describe formalmente la construcción de un grafo. El podado se realiza junto con esa construcción como se explica en la Sección 6.3, en donde también se demuestra que se obtiene un conjunto óptimo de contextos bi–direccionales. Se estudia la complejidad en la Sección 6.4. En la Sección 6.5 se describen resultados experimentales.

6.1. Definición

Un *grafo compacto de contextos bi–direccionales* (o CBDCG por su nombre en inglés, "compact bi-directional context graph") es un grafo enraizado, dirigido y acíclico, definido para cada secuencia $\mathbf{x}$, de tal manera que hay una correspondencia uno a uno entre los vértices del grafo y los contextos representativos de $\mathbf{x}$. Con un abuso de notación a los vértices sumideros los llamaremos 'hojas', a los que tienen aristas salientes 'internos' y a los destinos de las aristas salientes, 'hijos'. La raíz está etiquetada con (ϵ, ϵ), los vértices internos corresponden a los bi–ramificables y las hojas representan los bi–sufijos. Cada vértice interno tiene dos conjuntos de hijos, ambos con más de un elemento. Para uno de los conjuntos, el de los *hijos izquierdos* del vértice, se establece una correspondencia uno a uno con las extensiones mínimas por izquierda que ocurren del vértice. Más precisamente, el hijo es el contexto representativo de la extensión mínima que le corresponde. O sea, si $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es un vértice interno y $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ es una extensión que ocurre, entonces el hijo de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ que corresponde a $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ es $\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$. Obviamente, cada uno de los componentes izquierdos de las aristas que salen de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ hacia sus hijos izquierdos comienzan con un símbolo diferente. Está claro que el conjunto de hijos izquierdos de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ es un conjunto válido relativo a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, precisamente el conjunto válido mínimo por izquierda, definido en la Expresión (4.3). Consideraciones análogas valen para el conjunto de *hijos derechos* del vértice. Las aristas están etiquetadas con un par ordenado de cadenas sobre $\mathcal{A}^*$ y al menos uno de sus componentes no es ϵ. Entonces, un CBDCG puede ser considerado como una generalización de los árboles compactos. Notemos, sin embargo que un vértice de un CBDCG puede ser hijo derecho de más de un vértice, y al mismo tiempo ser hijo izquierdo de los mismos u otros vértices.

En la Figura 6.1 se ve el grafo correspondiente al Ejemplo 5.18. Por ejemplo $(\mathtt{1}, \mathtt{10})$ es hijo derecho tanto de (ϵ, ϵ) como de $(\mathtt{1}, \epsilon)$. El bi–sufijo 6 es hijo derecho de $(\epsilon, \mathtt{0})$ e hijo derecho e izquierdo de $(\mathtt{1}, \mathtt{0})$.

Ejemplo 6.1. Supongamos que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ y $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} c)$ son contextos bi–ramificables, y por lo tanto vértices del grafo. El vértice $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} c)$ es hijo derecho de $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ y también sería hijo derecho de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ si no ocurriera $(\mathbf{s}_\ell b, \mathbf{s}_\mathbf{r} c)$ con $b \neq a$. Vamos a ver un ejemplo para la aplicación eliminación de ruido en el que sucede esto (esta aplicación limita la libertad en la elección de las cadenas formadoras de contextos $\mathbf{y}_\ell$ e $\mathbf{y}_\mathbf{r}$, y es por lo tanto un ejemplo más fuerte). Para que el ejemplo sea más sencillo usamos el alfabeto $\{0, 1\}$, tomamos $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = (\epsilon, \epsilon)$ y no tendremos en cuenta la primera y la última posición para no necesitar el símbolo marcador. Sea $\mathbf{x} = \mathtt{00000101011}$. Comprobamos que

- (ϵ, ϵ) es bi–ramificable porque ocurren $(\mathtt{0}, \mathtt{0})$ (posiciones 2, 3, 4 y 8) y $(\mathtt{1}, \mathtt{1})$ (posiciones 7 y 9).

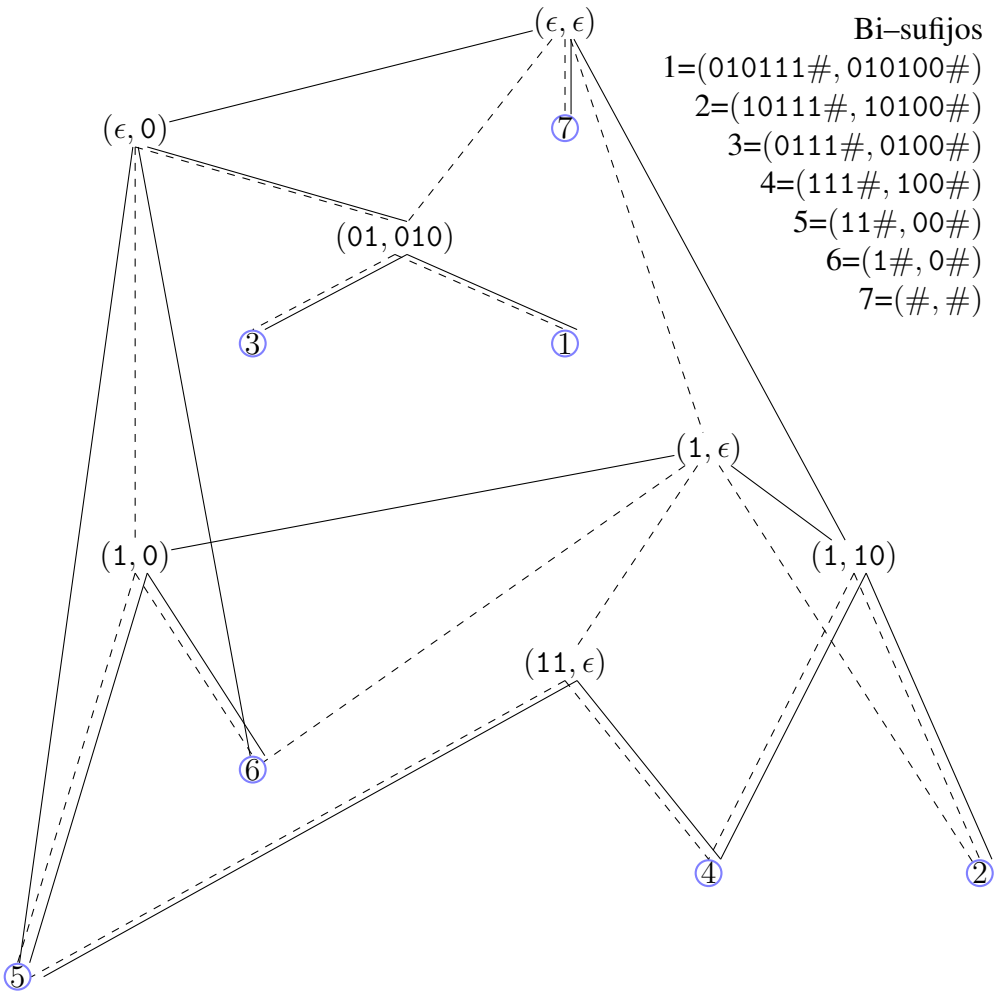


Figura 6.1: Grafo compacto de contexos bi–direccionales para Ejemplo 5.18. Se muestran todos los vértices del grafo. Los bi–sufijos se representan con el número de hoja y los bi–ramificables con su contexto. Las aristas de trazo continuo muestran las relaciones padre–hijo derecho, ordenadas para cada vértice de izquierda a derecha según las extensiones por la arista 0, # y 1 en caso de que las haya. Las aristas de trazo discontinuo muestran las relaciones padre–hijo izquierdo. Todas las aristas están orientadas desde arriba hacia abajo.

- $(\mathtt{0}, \epsilon)$ es bi–ramificable porque ocurren $(\mathtt{00}, \mathtt{0})$ (posiciones 3 y 4) y $(\mathtt{01}, \mathtt{1})$ (posición 10).
- $(\mathtt{0}, \mathtt{0})$ es bi–ramificable porque ocurren $(\mathtt{00}, \mathtt{00})$ (posición 3) y $(\mathtt{01}, \mathtt{01})$ (posición 8).
- $(\mathtt{1}, \mathtt{0})$ no ocurre.

Por lo tanto $(\mathtt{0}, \mathtt{0})$ es hijo derecho de (ϵ, ϵ) y de $(\mathtt{0}, \epsilon)$.

Un CBDCG se construye para obtener un conjunto óptimo definido por las hojas de un árbol óptimo, el cual se obtiene podando el grafo. Por eso, de cada vértice del grafo, además de la información de su estructura, debe mantenerse otra que permita determinar de qué manera se obtiene un conjunto óptimo relativo a él. Para cada contexto $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ esto se conoce con $\mathbf{split}(\mathbf{s}_\ell,\mathbf{s}_\mathbf{r})$, que indica si el conjunto óptimo se obtiene mediante la partición izquierda, la derecha, o con la 'no partición', o sea, si el vértice pertenece al conjunto óptimo. Para poder establecer este valor hay que conocer el del peso óptimo $\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, que a su vez depende del vector de conteos $\mathbf{c}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ vía el peso del vértice, $\mathtt{w}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. En la Sección 3.2 se presentó el Algoritmo 1 que encuentra el conjunto óptimo para el caso de contextos de longitud acotada. El correspondiente al caso actual se presentará en el Algoritmo 6 que usa información que se va estableciendo durante la construcción del grafo.

Con una analogía a los subárboles, definimos el *subgrafo enraizado* correspondiente a un vértice de un CBDCG como el conjunto de vértices que descienden del vértice considerado y las aristas que los unen. En adelante, siempre que se hable de subgrafo significa subgrafo enraizado. El subgrafo correspondiente a $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ se denota $G(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. El conjunto de todos los subgrafos es $\mathcal{G}$. Está implícito que son los subgrafos generados por la secuencia $\mathbf{x}$.

Entonces damos la especificación para cada subgrafo $G \in \mathcal{G}$.

- $\mathbf{root}(G) = (\mathbf{left}(G), \mathbf{right}(G)) \in \mathsf{R}$ es el contexto representativo que corresponde a la raíz del subgrafo.

  En lo que queda de esta especificación $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = (\mathbf{left}(G), \mathbf{right}(G))$.
- Si $\mathbf{root}(G) \in \mathcal{B}$, entonces $G^L, G^R \subset \mathcal{G}$ son los conjuntos de subgrafos cuyas raíces son los hijos izquierdos y derechos respectivamente de la raíz de

G. Cada conjunto tiene al menos dos elementos. Si $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$(resp. $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r} a)$) ocurre, entonces $G^L[a] = G(\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}))$ (resp. $G^R[a] = G(\mathsf{R}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r} a))$).

Si $\mathbf{root}(G) \in \mathcal{S}$ (conjunto de bi–sufijos), entonces $G^L = \emptyset$ y $G^R = \emptyset$.

- $\mathbf{c}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, $\mathtt{w}_{opt}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, $\mathbf{split}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ son el vector de conteos, el peso óptimo y el split respectivamente de la raíz de G. Se definen como en las Ecuaciones (4.5), (4.6) y (4.7) respectivamente.

6.2. Construcción del CBDCG

En la Sección 5.2 se describió cómo calcular los contextos representativos y también se obtuvo una cota superior de la cantidad de dichos contextos. Si sólo interesara obtener esos contextos el Algoritmo 3 sería suficiente. Pero como el objetivo es construir el CBDCG, además de obtener los contextos hay que establecer la estructura del grafo, esto es, las relaciones padre-hijo, y asignar los valores del vector de conteos, peso óptimo e indicador de split.

Sin embargo, del Algoritmo 3 también se puede deducir parte de la estructura del grafo. Para cada nodo $\mathbf{s}_\mathbf{r}$ de $T_{\mathbf{y}_\mathbf{r}}$ se obtuvo $\mathsf{R}_L(\mathbf{s}_\mathbf{r})$, el conjunto de nodos de $T_{\mathbf{y}_\ell}$ con los que forma contexto representativo. Además se conoce el conjunto $T^C_{\mathbf{y}_\ell}(\mathbf{s}_\mathbf{r})$ que contiene a $\mathsf{R}_L(\mathbf{s}_\mathbf{r})$ y a todos los nodos de $T_{\mathbf{y}_\ell}$ con los que los descendientes de $\mathbf{s}_\mathbf{r}$ en $T_{\mathbf{y}_\mathbf{r}}$ forman contexto representativo. Este conjunto $T^C_{\mathbf{y}_\ell}(\mathbf{s}_\mathbf{r})$ tiene estructura de árbol (Teorema 5.17), de la cual se pueden obtener las aristas padre-hijo izquierdo en el grafo que se quiere construir. En esta sección, extendiendo este algoritmo con el Algoritmo 4, se va a completar la construcción del grafo.

La construcción del CBDCG empieza por obtener las cadenas de formación de contextos y sus árboles de sufijos, $T_{\mathbf{y}_\ell}$ y $T_{\mathbf{y}_\mathbf{r}}$. Entonces se procede a un recorrido en post–orden de uno de ellos. Por razones de complejidad demostradas en el Teorema 5.20, y en las que se insistirá en la Sección 6.4, se elige aquel cuya altura sea menor. Sin pérdida de generalidad, la siguiente descripción supone que ese es el derecho, $T_{\mathbf{y}_\mathbf{r}}$. El objetivo es encontrar para cada $\mathbf{s}_\mathbf{r}$ en $\mathbf{V}(T_{\mathbf{y}_\mathbf{r}})$ el conjunto de nodos de $T_{\mathbf{y}_\ell}$ que forman un contexto representativo con $\mathbf{s}_\mathbf{r}$, o sea $\mathsf{R}_L(\mathbf{s}_\mathbf{r})$, establecer la estructura, esto es, las relaciones padre-hijo, y asignar los valores del vector de conteos, peso óptimo e indicador de split.

Para cada nodo $\mathbf{s}_\mathbf{r}$ de $T_{\mathbf{y}_\mathbf{r}}$, obtendremos un subgrafo, $g(\mathbf{s}_\mathbf{r})$, cuyos vértices están en correspondencia con el conjunto $T^C_{\mathbf{y}_\ell}(\mathbf{s}_\mathbf{r})$, o sea con los componentes izquierdos de los contextos representativos cuyos componentes derechos son las extensiones de $\mathbf{s}_\mathbf{r}$. Esto significa que es el subgrafo $G(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$, siendo $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = \mathsf{R}(\epsilon, \mathbf{s}_\mathbf{r})$.

En el paso inicial del recorrido en post–orden se relaciona cada hoja de $T_{\mathbf{y}_\mathbf{r}}$ con la hoja correspondiente de $T_{\mathbf{y}_\ell}$, de tal manera que se crea un vértice del CBDCG que corresponde a un bi–sufijo. El subgrafo que corresponde a la hoja de $T_{\mathbf{y}_\mathbf{r}}$ tiene

un único vértice. El resto de la información del vértice, esto es vector de conteos, peso óptimo y split se obtiene de manera trivial.

En el paso general, al procesar el nodo interno $\mathbf{s_r}$ de $T_{\mathbf{y_r}}$, el algoritmo usa $T_{\mathbf{y}_\ell}$ para obtener el conjunto $\mathsf{R}_L(\mathbf{s_r})$ formado por los contextos izquierdos que forman un contexto representativo con $\mathbf{s_r}$. Por cada elemento de $\mathsf{R}_L(\mathbf{s_r})$ se obtiene un vértice del `CBDCG` con $\mathbf{s_r}$ como componente derecho. Además se establece la estructura del grafo.

El recorrido de $T_{\mathbf{y_r}}$ se hace con la función recursiva **graph**, descrita en el Algoritmo 4.

En ese algoritmo, $\mathbf{s_r}$ es un nodo interno de $T_{\mathbf{y_r}}$ y recordamos que **children** devuelve el conjunto de hijos del nodo pasado como parámetro. La variable S es un conjunto de subgrafos.

Se puede ver que este algoritmo es una analogía del Algoritmo 3 en la que se devuelven grafos en lugar de árboles.

Algoritmo 4 $\mathbf{graph}(\mathbf{s_r} : \mathbf{B}(T_{\mathbf{y_r}}))$

Salida: $g(\mathbf{s_r}) : \mathcal{G}$
Var $S : \mathbf{set}(\mathcal{G})$
$S \leftarrow \emptyset$
Para cada $\mathbf{t_r} \in \mathbf{children}(T_{\mathbf{y_r}}, \mathbf{s_r})$

$$S \leftarrow S \bigcup \begin{cases} g(\mathbf{t_r}) & \text{si } \mathbf{t_r} \in \mathbf{L}(T_{\mathbf{y_r}}) \\ \mathbf{graph}(\mathbf{t_r}) & \text{si } \mathbf{t_r} \in \mathbf{B}(T_{\mathbf{y_r}}) \end{cases}$$

$g(\mathbf{s_r}) \leftarrow \mathbf{union}(S, \epsilon, \mathbf{s_r})$
devuelve $g(\mathbf{s_r})$
Postcondición: $\mathbf{root}(g(\mathbf{s_r})) = \mathsf{R}(\epsilon, \mathbf{s_r})$

La primera invocación de **graph** se hace desde el algoritmo principal (Algoritmo 7) pasándole ϵ como parámetro. Después del recorrido en post–orden, finaliza al procesar la raíz de $T_{\mathbf{y_r}}$ y la salida es el `CBDCG`.

Como se ve en el Algoritmo 4, $g(\mathbf{s_r})$ es la salida de la función **union**. Esta función es la más importante de la construcción y se describe en el Algoritmo 5. De acuerdo a la especificación, **root** es la raíz de un subgrafo, con **left** y **right** se representan los componentes izquierdo y derecho respectivamente de la raíz del subgrafo, el conjunto de grafos cuyas raíces son los hijos izquierdos (resp. derechos) de la raíz de un subgrafo G se denota G^L (resp. G^R), y $G^L[a] \in G^L$ tiene como raíz el hijo izquierdo de $\mathbf{root}(G)$ que extiende $\mathbf{left}(G)$ según la rama a. O sea $\mathbf{root}(G^L[a]) = \mathsf{R}(\mathbf{left}(G)a, \mathbf{right}(G))$. La función **lca** obtiene el ancestro común más profundo de un conjunto de nodos del árbol $T_{\mathbf{y}_\ell}$, que es el prefijo común más largo de los contextos (izquierdos) representados por esos nodos. Como se había adelantado, este algoritmo es una generalización del Algoritmo 2.

Algoritmo 5 $union(S : \mathbf{set}(\mathcal{G}), \mathbf{v}_\ell : \mathbf{B}(T_{\mathbf{y}_\ell}), \mathbf{s_r} : \mathbf{B}(T_{\mathbf{y_r}}))$

Salida: $U : \mathcal{G}$
Var $S' : \mathbf{set}(\mathcal{G})$
Precondición 1: $|S| \geq 2$
Precondición 2: Las raíces de los grafos de S forman el conjunto válido mínimo por derecha (Expresión 4.4) de $(\mathbf{v}_\ell, \mathbf{s_r})$:
$\{\mathbf{root}(G) : G \in S\} = \{\mathsf{R}(\mathbf{v}_\ell, \mathbf{s_r}b) : b \in \mathcal{A}, \mathcal{I}(\mathbf{v}_\ell, \mathbf{s_r}b) \neq \emptyset\}$

$\mathbf{s}_\ell = \mathbf{lca}\,\{\mathbf{left}(G) : G \in S\}$, $\mathbf{s}_\ell \in \mathbf{B}(T_{\mathbf{y}_\ell})$
$\mathbf{root}(U) \leftarrow (\mathbf{s}_\ell, \mathbf{s_r})$
$U^R \leftarrow S$
Para cada $\mathbf{t}_\ell \in \mathbf{children}(T_{\mathbf{y}_\ell}, \mathbf{s}_\ell)$
    Sea $a \in \mathcal{A}$ tal que $a = [\mathbf{t}_\ell]_{|\mathbf{s}_\ell|+1}$, o sea $\mathbf{s}_\ell a \preceq \mathbf{t}_\ell$
    $S' \leftarrow \emptyset$
    Para cada $G \in S$

$$S' \leftarrow S' \cup \begin{cases} G & \text{si } \mathbf{t}_\ell \preceq \mathbf{left}(G) \\ G^L[a] & \text{si } \mathbf{s}_\ell = \mathbf{left}(G) \text{ y } \mathbf{root}(G^L[a]) \text{ ocurre} \\ \emptyset & \text{en otro caso.} \end{cases}$$

    Si $S' \neq \emptyset$ **entonces**

$$U^L[a] \leftarrow \begin{cases} G' & \text{si } S' = \{G'\} \\ \text{union}(S', \mathbf{t}_\ell, \mathbf{s_r}) & \text{si } |S'| \geq 2. \end{cases}$$

{actualizar vector de conteos, peso óptimo, split de $\mathbf{root}(U)$}
devuelve U
Postcondición: $\mathbf{root}(\mathbf{U}) = \mathsf{R}(\mathbf{v}_\ell, \mathbf{s_r})$

Para cada nodo $\mathbf{s_r} \in \mathbf{B}(T_{\mathbf{y_r}})$ la llamada inicial de la función **union** invocada desde el Algoritmo 4 pasa como primer parámetro el conjunto de los subgrafos $g(\mathbf{t_r})$ para todos los hijos $\mathbf{t_r}$ de $\mathbf{s_r}$. La salida de la llamada inicial de **union** es el subgrafo $g(\mathbf{s_r})$ cuya raíz es $(\mathbf{s}_\ell, \mathbf{s_r}) = \mathsf{R}(\epsilon, \mathbf{s_r})$. Mientras se genera $g(\mathbf{s_r})$, **union** obtiene todos los vértices correspondientes a los contextos representativos cuyo componente derecho es $\mathbf{s_r}$. Cada llamada recursiva a **union** obtiene un vértice que es la raíz del subgrafo U que devuelve. Para eso en la línea 5 calcula $\mathbf{s}_\ell$, que es el `lca` de los componentes izquierdos de las raíces de los grafos del conjunto parámetro S. Estas raíces son los hijos derechos del vértice que se va a obtener, por lo que inmediatamente se puede establecer en la línea 7 las aristas padre–hijo derecho. Estas aristas están determinadas por la correspondencia establecida en la Precondición 2, es decir, si $\mathbf{root}(G) = \mathsf{R}(\mathbf{v}_\ell, \mathbf{s_r}b)$ entonces $U^R[b] = G$. Para establecer las relaciones padre–hijo izquierdo se aplica el bucle de la línea 8, que es el análogo del de la línea 4 del Algoritmo 2. Tal como en ese algoritmo, es posible que deban hacerse llamadas recursivas a **union**. Para cada hijo $\mathbf{t}_\ell$ de $\mathbf{s}_\ell$ en $T_{\mathbf{y}_\ell}$ se considera el símbolo a que es el primer símbolo de $\mathbf{u}$, donde $\mathbf{t}_\ell = \mathbf{s}_\ell\mathbf{u}$. Es decir el símbolo a identifica la rama por la cual desciende $\mathbf{t}_\ell$. En el bucle de la línea 11 se obtiene el subgrafo de cada G en S que puede contribuir a la formación de $U^L[a]$, el subgrafo izquierdo de $\mathbf{root}(U)$ según la rama a. Es el subgrafo cuyos vértices son extensiones de $(\mathbf{t}_\ell, \mathbf{right}(G))$. Puede no existir, ser el propio G, o ser $G^L[a]$. Se lo incluye en un conjunto S'. Si S' no es vacío entonces en la línea 14 se establece la relación padre–hijo izquierdo entre $\mathbf{root}(U)$ y $\mathbf{root}(U^L[a])$. El subgrafo $U^L[a]$ puede ser un subgrafo que ya existía si S' tiene un único elemento, o el resultado de una llamada recursiva si tiene más de uno. Finalmente, en la línea 15 se actualiza la información del vértice, esto es, vectores de conteos, peso óptimo y **split**, que serán utilizada para el podado del grafo como se verá en la Seccion 6.3. Para consideraciones de complejidad computacional notemos que el número total de llamadas a **union** es la cantidad de vértices internos del grafo.

Veamos que las precondiciones del Algoritmo 5 se cumplen. El algoritmo es invocado desde dos lugares, la llamada inicial desde el Algoritmo 4 y la recursiva desde la línea 14.

En la llamada hecha desde el Algoritmo 4 la Precondición 1 se cumple porque $\mathbf{s_r}$ es un nodo interno de $T_{\mathbf{y_r}}$ y por lo tanto S tiene más de un elemento. Para cada símbolo b tal que $(\epsilon, \mathbf{s_r}b)$ ocurre hay en $T_{\mathbf{y_r}}$ exactamente un $\mathbf{t_r}$ hijo de $\mathbf{s_r}$, y por lo tanto un grafo en S. La raíz de $g(\mathbf{t_r})$ es $\mathsf{R}(\epsilon, \mathbf{t_r})$ si $\mathbf{t_r}$ es una hoja porque es el bi–sufijo cuyo componente derecho es $\mathbf{t_r}$. Y si $\mathbf{t_r}$ no es hoja, por la postcondición del algoritmo, la raíz de $g(\mathbf{t_r})$ también es $\mathsf{R}(\epsilon, \mathbf{t_r})$. Notando que el valor de $\mathbf{v}_\ell$ en la llamada a **union** es ϵ, se concluye que se cumple la Precondición 2.

Para considerar las llamadas recursivas vamos a establecer dos lemas.

Lema 6.2. *Si* $(\mathbf{t}_\ell, \mathbf{t_r})$ *es extensión estricta de* $(\mathbf{s}_\ell, \mathbf{s_r})$ *y prefijo del representativo*

de $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$ *entonces ambos tienen el mismo representativo.*

$$\textit{Si } (\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \prec (\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) \preceq \mathsf{R}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) \textit{ entonces } \mathsf{R}(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r}) = \mathsf{R}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}).$$

Demostración. Por las relaciones de prefijo dadas en las hipótesis se sabe que $(\mathbf{t}_\ell, \mathbf{t}_\mathbf{r})$ pertenece a la misma clase que $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. Por lo tanto sus representativos son el mismo. □

Lema 6.3. *Si* $\mathsf{R}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = (\mathbf{s}_\ell, \mathbf{t}_\mathbf{r})$ *entonces* $\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) = \mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$.

Demostración. Si $\mathbf{s}_\mathbf{r} = \mathbf{t}_\mathbf{r}$ la proposición se cumple de manera trivial.

Supongamos, en otro caso, que $\mathbf{s}_\mathbf{r} b \preceq \mathbf{t}_\mathbf{r}$. La hipótesis implica que no ocurre $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r} d)$, con $d \neq b$, porque si ocurriera sería $\mathsf{R}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r}) = (\mathbf{s}_\ell, \mathbf{s}_\mathbf{r})$. Tenemos $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) \prec (\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r}) \preceq \mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$. Sea R' el representativo de $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$. No se puede cumplir $\mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r}) \prec R'$ porque el representativo de $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r})$ sería $\mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$ y no R'. Tampoco se puede cumplir $(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r}) \preceq R' \prec \mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$ porque el representativo de $(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$ sería R' y no $\mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$. Supongamos, para llegar a una contradicción, que se cumple $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r}) \preceq R' \prec (\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$ y por lo tanto $R' = (\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} v_r)$ para alguna cadena v_r, con $\mathbf{s}_\mathbf{r} v_r \prec \mathbf{t}_\mathbf{r}$. Recordando que $\mathbf{s}_\mathbf{r} b \preceq \mathbf{t}_\mathbf{r}$ y como R' es bi–ramificable tiene que existir $d \neq b$ tal que ocurra $(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} d)$ y por lo tanto también ocurre $(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r} d)$ en contradicción con lo que se había establecido al principio. La última suposición acerca de R' fue lo que llevó a esta contradicción. La única opción que queda es $R' = \mathsf{R}(\mathbf{s}_\ell a, \mathbf{t}_\mathbf{r})$ □

En las llamadas recursivas la Precondición 1 se cumple de manera explícita en la línea 14. Notemos que por la Precondición 2, se cumple que $\mathbf{v}_\ell \preceq \mathbf{left}(G)$ para todo $G \in S$, de donde $\mathbf{v}_\ell \preceq \mathbf{s}_\ell$, por lo que para todo $\mathbf{t}_\ell$ del bucle de la línea 8 se cumple $\mathbf{v}_\ell \prec \mathbf{t}_\ell$. Por lo tanto en el bucle de la línea 11 si $(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r} b)$ ocurre existe un único grafo G para el que se cumple

$$\mathbf{root}(G) = (\mathbf{left}(G), \mathbf{right}(G)) = \mathsf{R}(\mathbf{v}_\ell, \mathbf{s}_\mathbf{r} b) \preceq \mathsf{R}(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r} b). \tag{6.1}$$

Si en la línea 12 se da el primer caso, $\mathbf{t}_\ell \preceq \mathbf{left}(G)$, y como ya se sabía que $\mathbf{s}_\mathbf{r} b \preceq \mathbf{right}(G)$, tenemos $(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r} b) \preceq (\mathbf{left}(G), \mathbf{right}(G))$ de donde, junto con (6.1) y el Lema 6.2, tenemos $\mathbf{root}(G) = \mathsf{R}(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r} b)$. Y en este caso, G es el grafo que se incluye en S'.

Para el segundo caso de la línea 12, notemos que de $v_\ell \preceq \mathbf{s}_\ell$, por el Lema 6.2 y por la Precondición 2 se cumple $\mathsf{R}(\mathbf{s}_\ell, \mathbf{s}_\mathbf{r} b) = \mathbf{root}(G)$. Como en este caso $\mathbf{s}_\ell = \mathbf{left}(G)$ estamos en las hipótesis del Lema 6.3. Por lo tanto $\mathsf{R}(\mathbf{s}_\ell a, \mathbf{s}_\mathbf{r} b) = \mathsf{R}(\mathbf{left}(G) a, \mathbf{right}(G))$, que es $\mathbf{root}(G^L[a])$. Y $G^L[a]$ es el grafo que se incluye en S'.

Por lo tanto en S' se incluyen los grafos cuyas raíces son los representativos de $(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r} b)$ para todos los $(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r} b)$ que ocurren. Esas raíces son el conjunto válido mínimo por derecha de $(\mathbf{t}_\ell, \mathbf{s}_\mathbf{r})$ cuyos componentes son los parámetros de

union de la línea 14. Queda demostrado que se cumplen las precondiciones del Algoritmo 5.

Entonces, el resultado de $union$ es el subgrafo constituido por todos los contextos representativos que son extensión de $(\mathbf{v}_\ell, \mathbf{s_r})$. Como en el Algoritmo 4 esta función se invoca con $\mathbf{v}_\ell = \epsilon$ se concluye el siguiente teorema:

Teorema 6.4. *El algoritmo* $\mathbf{graph}(\mathbf{s_r})$ *construye correctamente la estructura del subgrafo que contiene todos los vértices cuyo componente derecho es una extensión de* $\mathbf{s_r}$.

Decir que lo construye correctamente significa que hay una correspondencia uno a uno entre los vértices del subgrafo y los contextos representativos cuya componente derecha es extensión de $\mathbf{s_r}$, y que los hijos de los vértices internos son los representativos de las extensiones mínimas que ocurren.

6.3. Podado

El podado del grafo se hace al mismo tiempo que se lo construye cuando en la función **union** se asigna el valor del indicador de **split** al vértice creado. Por lo tanto está implícitamente definido el árbol óptimo que tiene raíz en ese vértice.

En un paso final se obtiene formalmente el árbol óptimo que corresponde a $\mathbf{x}$. Se hace recursivamente, comenzando en la raíz usando la función **prune** descrita en el Algoritmo 6.

Algoritmo 6 $\mathbf{prune}\ (G : \mathcal{G})$

Salida: $T_{opt} : \mathcal{T}_{opt}(\mathbf{root}(G))$

Sea $\mathbf{split} = \mathbf{split}(\mathbf{root}(G))$

$$T_{opt} = \{\mathbf{root}(G)\} \cup \begin{cases} \emptyset & \text{si } \mathbf{split} = \text{`}N' \\ \bigcup_{G' \in G^L} \mathbf{prune}(G') & \text{si } \mathbf{split} = \text{`}L' \\ \bigcup_{G' \in G^R} \mathbf{prune}(G') & \text{si } \mathbf{split} = \text{`}R' \end{cases}$$

devuelve T_{opt}

Para ver que el valor de **split** se asigna correctamente describimos ahora la línea 15 del final del Algoritmo 5. Por el Teorema 6.4 se sabe que para cada vértice del CBDCG cada hijo es asignado correctamente de acuerdo a la definición de CBDCG. Y los conjuntos de hijos así definidos son los conjuntos en que se basan las Ecuaciones (4.5), (4.6) y (4.7). En el Algoritmo 5 se asignan los valores

a la raíz de U usando estas ecuaciones que sabemos que son correctas. Por lo tanto $\mathbf{split}(U)$ establece la partición correcta.

Queda por ver que la estructura devuelta es un árbol. Lo es de manera trivial si $\mathbf{split} = `N'$ porque tiene un único nodo. En otro caso a un conjunto de árboles se le agrega un nodo, $\mathbf{root}$(G), y para cada árbol del conjunto se agrega una rama desde $\mathbf{root}$(G) hacia la raíz del árbol, que por ser raíz no tenía ramas entrantes. Entonces tenemos una estructura conexa con un nodo que no tiene ramas entrantes y cualquier otro nodo tiene exactamente una rama entrante. Por lo tanto esa estructura es un árbol.

Con el Algoritmo 7 se resume todo el procedimiento. La entrada es una secuencia de n elementos que son m-uplas de símbolos de $\mathcal{A}$. En la aplicación para dos pistas, m es 2. En la eliminación de ruido m es 1. En esa descripción $\mathbf{contexts}$ genera las dos cadenas de formación de contextos, y $\mathbf{stree}$ es una función que construye el árbol de sufijos de una secuencia.

Algoritmo 7 $\mathbf{optimal-tree}\ (\mathbf{x}) : [\mathcal{A}^m]^n$

Salida: $T_{opt} : \mathcal{T}_{opt}(\epsilon, \epsilon)$
$(\mathbf{y}_\ell, \mathbf{y_r}) \leftarrow \mathbf{contexts}(\mathbf{x})$
$(T_{\mathbf{y}_\ell}, T_{\mathbf{y_r}}) \leftarrow (\mathbf{stree}(\mathbf{y}_\ell), \mathbf{stree}(\mathbf{y_r}))$
Para $i = 1$ to n
    $g(\mathbf{s}_{\mathbf{r}i}) = (\mathbf{s}_{\ell i}, \mathbf{s}_{\mathbf{r}i})$, donde $(\mathbf{s}_{\ell i}, \mathbf{s}_{\mathbf{r}i}) \in \mathcal{S} \subset \mathbf{L}(T_{\mathbf{y}_\ell}) \times \mathbf{L}(T_{\mathbf{y_r}})$
$G \leftarrow \mathbf{graph}(\epsilon)$
$T_{opt} \leftarrow \mathbf{prune}(G)$
devuelve T_{opt}

La correctitud del algoritmo es enunciada en el siguiente teorema.

Teorema 6.5 (Correctitud)**.** *Las hojas del árbol T_{opt} generado por el Algoritmo 7 constituyen un conjunto óptimo de contextos bi–direccionales para* $\mathbf{x}$.

Demostración. Teniendo en cuenta que, según el Teorema 6.4, $\mathbf{graph}(\epsilon)$ construye correctamente el CBDCG, incluyendo split, como se vio al describir el podado, obtenemos un árbol con peso mínimo. Las hojas de este árbol constituyen el conjunto óptimo de contextos. □

6.4. Complejidad del Algoritmo

En la Sección 5.2.2 se obtuvo una cota de la cantidad de contextos representativos. Vamos a ver que esa también es una cota de la complejidad del Algoritmo 7.

Teorema 6.6 (Complejidad)**.** *La complejidad en el peor caso del algoritmo de obtención de un árbol óptimo correspondiente a una secuencia de longitud* n *es*

$$O(n \cdot \min\{h(T_{\mathbf{y}_\ell}), h(T_{\mathbf{y_r}})\}).$$

Demostración. La línea 2 obtiene las cadenas de formación de contextos recorriendo $\mathbf{x}$ con tiempo constante por símbolo. Para la línea 3, hay varias construcciones de árboles de sufijos en tiempo lineal [10], [11], [12], [13].

El subgrafo $g(\mathbf{s_r})$ que corresponde a cada hoja de $T_{\mathbf{y_r}}$ tiene un único vértice y se construye en tiempo constante. Entonces, el bucle de la línea 4 es $O(n)$.

La línea 7, el podado, también implica tiempo lineal. Devuelve un árbol que a lo sumo tiene n hojas por lo que su cantidad de vértices es $O(n)$. El valor **split** ya había sido calculado al construir el `CBDCG` lo cual permite que: (a) procesar cada nodo del árbol requiera tiempo $O(1)$, y (b) se procesen sólo los nodos del árbol devuelto sin recorrer los que son descartados. Por lo tanto se concluye que el tiempo de este algoritmo es $O(n)$.

Por lo tanto, la complejidad del algoritmo propuesto está dominada por la construcción del `CBDCG`, en la línea 6.

El `CBDCG` es construido mediante una recorrida en post–orden de $T_{\mathbf{y_r}}$ en el Algoritmo 4. Este algoritmo se invoca una vez por cada vértice y todo lo que se hace en él es $O(1)$, excepto la llamada a **union**. Entonces cada nodo $\mathbf{s_r}$ de $T_{\mathbf{y_r}}$ es procesado por el Algoritmo 5. Una característica clave de este algoritmo es que, como el `lca` puede calcularse en tiempo constante (Apéndice A), la ejecución es $O(1)$ excepto por las llamadas recursivas. Por lo tanto, la complejidad de la construcción del `CBDCG` es igual a la cantidad de llamadas a la función **union**. Para un contexto $\mathbf{s_r}$ dado, las llamadas recursivas generan todos los vértices del grafo cuyas componentes izquierdas forman un contexto representativo con $\mathbf{s_r}$. Por lo tanto para cada $\mathbf{s_r}$ todas esas componentes izquierdas son diferentes. Entonces la complejidad de procesar $\mathbf{s_r}$ es igual a la cantidad de vértices en $\mathsf{R}_L(\mathbf{s_r})$. Entonces el presente teorema queda demostrado por el Teorema 5.20. □

El peor caso se da cuando la altura del árbol es $n-1$, produciendo un algoritmo cuya complejidad es $O(n^2)$

6.5. Datos experimentales

6.5.1. Consideraciones prácticas

La implementación del árbol de sufijos se hizo mediante árboles binarios con semántica *primer–hijo siguiente–hermano*. Se usa la versión *suffix–links* y *hojas abiertas* [12]. El nodo representa un contexto, el cual se representa mediante la

posición de inicio en la cadena y la longitud (o con las posiciones de inicio y fin). Si el nodo es una hoja alcanza con la posición de inicio porque se sabe que termina al final de la cadena. En el caso de las hojas tampoco hacen falta el puntero al primer hijo ni el suffix–link. Teniendo en cuenta esto se mantienen separadas las estructuras en un arreglo para los nodos internos, `interns`, y otro para las hojas, `leaves`. Se sabe que hay n hojas y a lo sumo n nodos internos. Los punteros, primer–hijo, siguiente–hermano, suffix–link, son posiciones en los arreglos. Para saber si un puntero es a un nodo que está en `interns` o en `leaves` se utiliza uno de los bits de la dirección. Vamos a suponer que una posición se puede representar con 4 bytes, lo que permite procesar cadenas de hasta 2^{31} símbolos (en lugar de 2^{32} ya que se necesita el bit mencionado; más adelante se explica que esta cota se va a limitar a 2^{30}). Por lo tanto para un nodo interno hacen falta 20 bytes correspondientes a 5 posiciones: inicio y fin, primer–hijo, siguiente–hermano y suffix–link. Para las hojas hacen falta 8 bytes correspondientes a 2 posiciones: inicio y siguiente–hermano. Entonces el árbol de sufijos se puede implementar con $28n$ bytes.

Una vez terminada la construcción cada árbol se adapta para su función específica. El árbol derecho se mantiene por niveles para recorrerlo desde abajo hacia arriba. Esta estructura requiere menos espacio. El árbol izquierdo se modifica para soportar consultas de los `lca`, lo cual se describe en el Apéndice A. Se necesita mantener arreglos cuyo tamaño, que llamamos p, es el doble de la cantidad de nodos. Entonces, como p puede llegar a ser $4n$, si se limita el largo de las cadenas a 2^{30} también las posiciones de estos arreglos se pueden representar con 4 bytes. Además de las estructuras específicas para calcular el `lca`, se mantienen otros dos arreglos. En `lengths` se mantiene la longitud del contexto representado por cada nodo. Como esa longitud es a lo sumo n y la cantidad de nodos está acotada superiormente por $2n$ el espacio ocupado por `lengths` es $8n$. En `lexord_leaf` se especifica cuál es la posición relativa en orden lexicográfico de los contextos de cada una de las n hojas en el conjunto de los contextos de los hasta $2n$ nodos. Por lo tanto requiere $4n$ bytes.

El grafo también se construye con un arreglo para los bi–sufijos, `bileaf` y otro para los bi–ramificables, `bibranch`. Pero como no se sabe hasta terminar de construirlo la cantidad de bi–ramificables este último arreglo se debe poder redimensionar. Cada bi–sufijo se representa con una posición por lo que `bileaf` se implementa con $4n$ bytes. Para los bi–ramificables se mantienen 3 listas mediante nodos enlazados, una para hijos izquierdos, otra para hijos derechos y otra para el conteo de ocurrencias de los símbolos. La longitud de cada una de estas listas puede llegar a ser $|\mathcal{A}|$. Los nodos ocupan 5 bytes por lo que cada lista puede ocupar $5\,|\mathcal{A}|$. Además se mantiene la longitud de cada una de esas listas y otro byte con *banderas*.

En esta descripción se asumió que las posiciones se representan con 4 bytes

y los símbolos del alfabeto con 1. Para una implementación más ajustada, en el Cuadro 6.1, se muestra el espacio en cantidad de bits de las diferentes estructuras. La longitud de algunos arreglos es p que es $2(|nodos|-1)$ (y por lo tanto $(p+2)/2$ es la cantidad de nodos), donde $|nodos|$ es la cantidad de nodos del árbol izquierdo. Su máximo valor posible es $4n$ pero su valor real se conoce en el momento de crear los arreglos. Uno de esos arreglos se lo considera en bloques de longitud $l = (1/2)(\log p)$. h es la altura del árbol derecho, y se conoce en el momento de crear el arreglo `depth`.

`interns`	$5n \lceil \log n \rceil$
`leaves`	$2n \lceil \log n \rceil$
`nodes`	$p \lceil \log \frac{p+2}{2} \rceil$
`depth`	$p \lceil \log h \rceil$
`branch`	$\frac{p+2}{2} \lceil \log \lvert\mathcal{A}\rvert \rceil$
`firstpos`	$\frac{p+2}{2} \lceil \log p \rceil$
`length`	$\frac{p+2}{2} \lceil \log n \rceil$
`lexord_leaf`	$n \lceil \log \frac{p+2}{2} \rceil$
`rpow2`	$(p/l) \log(p/l) \lceil \log p \rceil$
`canonics`	$2^{l-1}\frac{1}{2}(l-1)(l-2) \lceil \log l \rceil$
`b2c`	$(p/l) \lceil l-1 \rceil$
`bileaf`	$n \lceil \log n \rceil$
`bibranch`	$\lvert\mathcal{B}\rvert (1 + 3 \lceil \log \lvert\mathcal{A}\rvert \rceil + 3(\lceil \log \lvert\mathcal{A}\rvert \rceil + \lceil \log n \rceil) \lvert\mathcal{A}\rvert)$

Cuadro 6.1: Espacio en bits de las estructuras

6.5.2. Resultados experimentales

Este algoritmo fue implementado y aplicado en el marco de la aplicación eliminación de ruido al texto "Don Quijote" de Miguel de Cervantes. El alfabeto consistió en 96 símbolos (salto de línea y los correspondientes a los códigos ASCII desde el 32 hasta el 126). Las estadísticas del `CBDCG` generado se presentan en el Cuadro 6.2. La primera fila da la longitud de las secuencias. Cada secuencia contiene a la anterior. La segunda fila da la altura del árbol derecho. La tercera da la cantidad de contextos bi–ramificables, o sea, la cantidad de vértices internos del grafo. Este número es proporcional a la complejidad del algoritmo. En la cuarta fila se puede ver que $n \cdot h(T_{\mathbf{yr}})$ es una cota superior de la cantidad de bi–ramificables, tal como se propuso en el Teorema 6.6. Al crecer n la relación entre las dos magnitudes no crece, sino que incluso disminuye levemente.

En el Listado 6.1 se muestra el resultado de la corrida de prueba con la secuencia de longitud 640000. En esta implementación no se tuvo en cuenta cuál de

$\mathbf{n}$ ($\times$ 1000)	10	20	40	80	160	320	640
$\mathbf{h}(\mathbf{T_{y_r}})$	14	14	15	18	22	24	28
$\lvert\mathcal{B}\rvert$	16489	34074	73513	161879	353018	759116	1627152
$\lvert\mathcal{B}\rvert / (n \cdot h(T_{\mathbf{y_r}}))$	0.12	0.12	0.12	0.11	0.10	0.10	0.09

Cuadro 6.2: Magnitud de CBDCG's

los dos árboles era el de menor altura sino que de manera predeterminada se usó el árbol derecho para la recorrida y el izquierdo para el cálculo de los `lca`. En el segmento *Construcción de la estructura del grafo* del listado se muestra para cada nivel del árbol derecho, desde el más profundo hasta la raíz, la cantidad de hojas y nodos internos del árbol y la cantidad de bi–ramificables que tienen a los nodos de ese nivel como componentes derechos. La cantidad de bi–sufijos es igual a la cantidad de hojas. En el segmento *Perfil del conjunto óptimo* se muestra la cantidad de hojas del árbol óptimo que corresponden a cada nivel del árbol derecho, desde la raíz hasta el más profundo.

Esta corrida fue realizada en una máquina con arquitectura x86-64 con 4 CPUs de 3093 MHz, con Caché L1d 32K, L1i 32K, L2 256K, L3 3072K, con 4 GB de memoria. El sistema operativo fue Fedora 25 (Workstation Edition), con núcleo Linux 4.13.16-100.fc25.x86_64.

Listado 6.1: Corrida de prueba

```
Alojando memoria para alfabeto y matrices.
 Memoria: 149728; Memoria total: 149728
Hojas: 640000
Alojando memoria para construir árboles.
 Memoria: 17920028; Memoria total: 18069756

Construyendo árbol izquierdo.
 Tiempo: 0.20; Tiempo total: 0.20
 Nodos internos: 308744
Altura: 26
Convirtiendo el árbol para cálculos de LCA.
 Tiempo: 0.10; Tiempo total: 0.33
 Memoria: 37775918; Memoria total: 55845674

Construyendo árbol derecho.
 Tiempo: 0.21; Tiempo total: 0.54
 Nodos internos: 309629
Convirtiendo el árbol para construcción del grafo.
 Tiempo: 0.16; Tiempo total: 0.70
```

```
 Memoria: 11395896; Memoria total: 67241570
 Altura = 28

Eliminando mapeo entre hojas.
 Memoria: 2560004; Memoria total: 64681566
Eliminando árbol de sufijos.
 Memoria: 17920028; Memoria total: 46761538

Iniciando estructuras para el grafo.
 Memoria: 86827200; Memoria total: 133588738

Construcción de la estructura del grafo.
Hojas   Inter.  Biramificables
2       0       0
5       1       1
18      3       4
31      8       11
42      14      17
66      22      31
94      31      61
151     43      86
213     67      132
283     98      204
467     142     301
657     229     453
1029    337     671
1734    539     1021
3442    897     1668
7247    1745    3023
14865   3659    5934
29288   7508    11850
51522   14683   23177
80400   25627   42915
105144  39977   72257
117526  52188   108175
109262  58356   144258
76199   53336   173032
34014   35118   188553
5999    13139   195258
297     1795    203016
3       66      213464
0       1       237579
```

```
Cantidad de biramificables: 1627152
Cantidad de aristas: 10561333
 Memoria: 52806665; Memoria total: 186395403
 Tiempo: 1.26; Tiempo total: 1.96

Eliminando memoria auxiliar.
 Memoria: 5002176; Memoria total: 181393227

Ajustando el tamaño de la estructura de biramificables.
 Memoria: 0; Memoria total: 181393227

Estableciendo la posición de las hojas.
 Tiempo: 0.01; Tiempo total: 1.97

Eliminando estructura para LCA.
 Memoria: 37775918; Memoria total: 143617309
 Tiempo: 0.01; Tiempo total: 1.97
Calculando vectores de conteo.
 Memoria: 950704; Memoria total: 144568013
.............................
 Tiempo: 0.69; Tiempo total: 2.66
Cantidad de entradas en los vectores de
conteos de los nodos internos: 3508635
 Memoria: 17543175; Memoria total: 162111188
Eliminando memoria auxiliar:
 Memoria: 950704; Memoria total: 161160484
Calculando peso de los nodos.
 Memoria: 21207472; Memoria total: 182367956
.............................
 Tiempo: 5.39; Tiempo total: 8.05
Eliminando estructuras auxiliares.
 Memoria: 14055088; Memoria total: 168312868

Perfil del conjunto óptimo.
        Nivel    Cantidad
        0       0
        1       14
        2       943
        3       7888
        4       21344
        5       27829
        6       27639
        7       24222
```

```
        8       18542
        9       12401
        10      7285
        11      3850
        12      1928
        13      918
        14      458
        15      278
        16      168
        17      139
        18      107
        19      99
        20      63
        21      48
        22      42
        23      25
        24      20
        25      22
        26      6
        27      4
Tamaño del conjunto óptimo: 156282

Eliminando árbol de construcción.
 Memoria: 11395896; Memoria total: 156916972

Podado y eliminación de ruido.
 Memoria: 960; Memoria total: 156917932
 Tiempo: 0.08; Tiempo total: 8.13
```

Capítulo 7

Conclusión

En esta tesis se desarrolló un algoritmo para construir de manera eficiente conjuntos óptimos de contextos cuya longitud es no acotada. Se generaliza lo anteriormente presentado para contextos de longitud acotada [5], que tiene aplicación en compresión y eliminación de ruido. El algoritmo construye una estructura, el grafo compacto de contextos bi–direccionales que es una generalización de los árboles compactos de sufijos, definido en el Capítulo 6. Es un grafo acíclico dirigido y enraizado, en el que cada vértice que no es sumidero tiene dos conjuntos de aristas salientes. Cada vértice del grafo corresponde a una clase de contextos, más concretamente a un contexto representativo de la clase. Partir en clases de equivalencia el conjunto de contextos que ocurren y tratar con un contexto representativo de cada clase es lo que hace que el algoritmo sea eficiente. Se pasa de una cantidad de orden cúbico de contextos que ocurren a una cantidad cuadrática de contextos representativos.

Todos los contextos de una misma clase comparten propiedades esenciales para los problemas tratados: vector de conteos y como consecuencia el peso. A su vez, el vector de conteos está determinado por las posiciones en las que ocurre el contexto. Todos los contextos de una clase ocurren en las mismas posiciones. En el Capítulo 2 se reformula la notación habitual definiendo el conjunto de índices de un contexto, esto es, las posiciones de la secuencia en las que el contexto ocurre. Se ve que encontrar un conjunto válido de contextos (de los que los conjuntos óptimos buscados son casos particulares) implica una partición de los índices en conjuntos de índices.

En el Capítulo 4 se demuestra que todos los contextos de una clase de contextos tienen una extensión común más corta, que es un contexto de la clase. Esto no es trivial ya que, a diferencia del caso uni–direccional, las clases no tienen un prefijo común más largo. Esos contextos, extensiones comunes más cortas de sus respectivas clases, son los contextos representativos que se corresponden con los vértices del grafo.

Los contextos representativos (vistos como vértices del grafo) son una generalización de los nodos de los árboles de sufijos compactos [10, 11, 12, 13]. Son un subconjunto del producto cartesiano de los nodos de dos de estos árboles: los correspondientes a las cadenas de formación de contextos definidas en el Capítulo 3. Para poder encontrar los contextos representativos de manera eficiente en el Capítulo 5 se definen dos operaciones entre subárboles compactos de un árbol base, concepto también definido ahí. Una de ellas, `glca`, es una generalización de la obtención de los ancestros comunes más profundos de dos nodos de un árbol (`lca` [17, 18]). Es una generalización porque encuentra los `lca` de todos los nodos de un conjunto de árboles. Se describe un algoritmo que los encuentra de manera eficiente. La otra operación, `union`, es la unión de los nodos de subárboles y el `glca` de ellos.

En el Capítulo 6 se describe la construcción del grafo. Se recorre desde las hojas hacia la raíz (recorrido bottom–up) el árbol de sufijos de una de las cadenas de formación de contextos y para cada nodo del árbol recorrido se encuentra, mediante la operación `glca` en el árbol de sufijos correspondiente a la otra cadena de formación de contextos, los contextos con los que forma un contexto representativo. La operación `union` permite obtener la estructura del grafo. Se demuestra que la complejidad del algoritmo es $O(n \cdot h)$, siendo n la longitud de la secuencia a procesar y h el mínimo de las alturas de los árboles de sufijos de las dos cadenas de formación de contextos. En el peor caso esa altura es n, lo que demuestra que la construcción del grafo es en el peor caso $O(n^2)$.

El algoritmo se implementó y con experimentos se comprobó esta cota.

El posterior podado del grafo permite obtener en los nodos que quedan sin ramas salientes el conjunto óptimo de contextos.

Problemas abiertos

El Teorema 6.6 establece que la complejidad de la construcción del grafo está acotada superiormente por el producto de n y la menor de las alturas de los árboles. En el peor caso la altura de un árbol es n, lo cual conduce a un algoritmo $O(n^2)$. Esto ocurre, por ejemplo, si la secuencia consiste en n veces el mismo símbolo. Sin embargo cabe preguntarse qué tan inusual es este caso. Se ha demostrado que la altura promedio de los árboles de sufijos es $O(\log n)$ para algunos modelos de fuentes [19] (informalmente, cuando hay una débil dependencia entre un símbolo y los que ocurrieron d o más posiciones antes, para un cierto entero d). En particular para la aplicacion eliminación de ruido, como los dos árboles no son independientes, se puede estudiar si en algún modelo al menos uno de los dos árboles es casi seguramente de baja altura.

La cota superior $n \times h(T_{\mathbf{y_r}})$ de cantidad de contextos representativos encontrada en el Lema 5.19 es para el peor caso, en el que al procesar los nodos de cada

nivel de $T_{\mathbf{y_r}}$ es necesario involucrar cada hoja en el cálculo de algún `lca`. En el otro extremo, en el mejor caso el procesamiento de cada nodo $T_{\mathbf{y_r}}$ consistiría en un único cálculo de `lca`. Esto ocurre si los subárboles de $T_{\mathbf{y}_\ell}$ asociados a los hijos del nodo de $T_{\mathbf{y_r}}$ son disjuntos. Entonces la pregunta es qué tan ajustada es la cota $n \times h(T_{\mathbf{y_r}})$. En la cuarta fila del Cuadro 6.2 se observa que al crecer n decrece la relación entre la cantidad de contextos representativos y $n \times h(T_{\mathbf{y_r}})$. Podría haber modelos de fuentes para los cuales, con esta observación y con la anterior el algoritmo sea $o(n \log n)$ en promedio.

El uso de la estructura definida en esta tesis a los problemas de modelado por contextos presenta inconvenientes para la eliminación de ruido porque en esta aplicación no se conoce la cadena original, limpia. Como consecuencia del grafo se obtiene el conjunto de contextos que minimizan la estimación de la pérdida, y no la pérdida verdadera, por lo que no necesariamente se logra maximizar la eliminación de ruido. Se ha demostrado [20] que una longitud de los contextos mayor a $\log n/(2 \log |\mathcal{A}|)$ compromete la calidad de la estimación, y se ha propuesto en consecuencia limitar a ese valor el largo de los contextos, en concordancia con sugerencias anteriores [21]. Resultados de un comienzo de experimentación con esta aplicacion confirman estas conclusiones teóricas. Se pueden hacer experimentos más rigurosos para confirmar o no esta hipótesis. Además, esta limitación teórica en la longitud de los contextos hace pensar en el uso de esta estructura, truncada de manera adecuada, no con el objetivo de mejorar la calidad de los resultados sino con el de mejorar la eficiencia ya que para este propósito se puede aprovechar la compactibilidad dada por los contextos representativos. Hace falta determinar cómo se puede construir eficientemente el grafo a partir de árboles de sufijos con longitudes truncadas ya que deja de existir una correspondencia uno a uno entre las hojas de ambos árboles.

Bibliografía

[1] Jorma J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29(5):656–664, Sep 1983.

[2] Marcelo J. Weinberger, Jorma J. Rissanen, and Meir Feder. A universal finite memory source. *IEEE Transactions on Information Theory*, 41(3):643–652, May 1995.

[3] Frans Willems, Yuri Shtarkov, and Tjalling Tjalkens. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.

[4] Álvaro Martín, Gadiel Seroussi, and Marcelo Weinberger. Linear time universal coding and time reversal of tree sources via FSM closure. *IEEE Transactions on Information Theory*, 50(7):1442–1468, 2004.

[5] Erik Ordentlich, Marcelo J. Weinberger, and Tsachy Weissman. Efficient pruning of bi-directional context trees with applications to universal denoising and compression. *Information Theory Workshop, 2004. IEEE*, 0:94–98, 24-29 Oct. 2004.

[6] Erik Ordentlich, Marcelo J. Weinberger, and Cheng Chang. On multi-directional context sets. *IEEE Transactions on Information Theory*, 57(10):6827–6836, 2011.

[7] Erik Ordentlich, Marcelo J. Weinberger, and Tsachy Weissman. Multi-directional context sets with applications to universal denoising and compression. *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, 0:1270–1274, 4-9 Sept. 2005.

[8] Alfredo Viola. Efficient algorithms for universal denoising. In Lars Arge, Robert Sedgewick, and Raimund Seidel, editors, *Data Structures*, number 08081 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008.

[9] Fernando Fernández, Alfredo Viola, and Marcelo J. Weinberger. Efficient algorithms for constructing optimal bi-directional context sets. *Data Compression Conference*, 0:179–188, 2010.

[10] Peter Weiner. Linear pattern matching algorithms. *FOCS 14th Annual Symposium on Switching and Automata Theory*, 0:1–11, 1973.

[11] Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

[12] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[13] Robert Giegerich and Stefan Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.

[14] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 2006.

[15] Ragnar Nohre. *Some Topics in Descriptive Complexity*. Linköping studies in science and technology: Dissertations. Linköping Univ., Department of Electrical Engineering, 1994.

[16] Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. Context weighting for general finite-context sources. *IEEE Transactions on Information Theory*, 42(5):1514–1520, 1996.

[17] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. On finding lowest common ancestors in trees. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 253–265, New York, NY, USA, 1973. ACM.

[18] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, London, UK, 2000. Springer-Verlag.

[19] Wojciech Szpankowski. A generalized suffix tree and its (un)expected asymptotic behaviors. *SIAM J. Computing*, 22:1176–1198, 1996.

[20] Erik Ordentlich, Krishnamurthy Viswanathan, and Marcelo J. Weinberger. Twice-universal denoising. *IEEE Transactions on Information Theory*, 59(1):526–545, Jan 2013.

[21] Tsachy Weissman, Erik Ordentlich, Gadiel Seroussi, Sergio Verdu, and Marcelo J. Weinberger. Universal discrete denoising: known channel. *IEEE Transactions on Information Theory*, 51(1):5–28, Jan. 2005.

[22] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[23] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

Apéndice A

Cálculo del `lca`

En este apéndice se expone cómo encontrar el `lca` en tiempo constante.

Definición A.1. El Problema LCA consiste en, dados dos nodos v y w de un árbol enraizado T, encontrar el nodo u de T que es el ancestro común más profundo (lowest common ancestor) de v y w.

Si T es un árbol de contextos resolver este problema permite encontrar el prefijo común más largo de los contextos representados por v y w. El Poblema LCA fue planteado en 1973 [17]. La forma en que aquí se resuelve está basada en obras posteriores [18, 22, 23].

Para poder encontrar el `lca` de dos nodos en tiempo $O(1)$ se realiza un pre–procesamiento del árbol. Suponemos que el árbol tiene n hojas y por lo tanto $O(n)$ nodos. El resultado del preprocesamiento podría consistir en una tabla que mantenga el `lca` de cada par de nodos. El inconveniente de este enfoque es que tal tabla requeriría espacio y tiempo de construcción $O(n^2)$. Aquí se va a describir cómo reducir esa cota a $O(n)$.

Para el propósito de esta tesis se resuelve un problema extendido que consiste en encontrar el `lca` de un conjunto de nodos y el símbolo que identifica la rama por la cual cada uno de ellos desciende de ese `lca`

$$\texttt{lca\_branchs : SET(node)} \rightarrow \texttt{node} \times \texttt{SET}(\mathcal{A}) \tag{A.1}$$

en donde está implícita la estructura del árbol, y las dos listas tienen el mismo tamaño, que es mayor o igual a 2 y menor o igual al tamaño del alfabeto $\mathcal{A}$.

El Problema LCA se transforma en el Problema RRMQ (restricted range minimum query).

Definición A.2. El Problema RMQ consiste en, dadas las posiciones i y j de un arreglo A de números, encontrar la posición del elemento más pequeño en el sub–arreglo $A[i \ldots j]$.

Si el mínimo se presenta en más de una posición aquí elegimos la última de ellas.

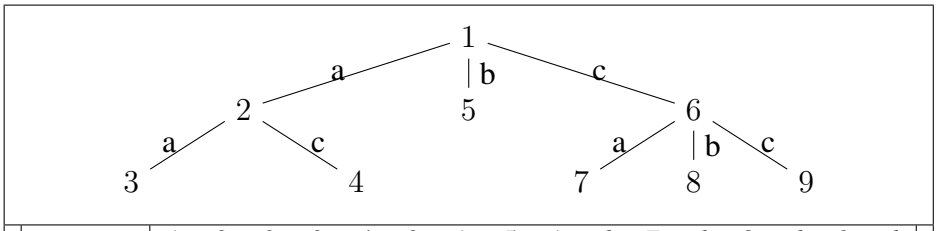
El Problema RRMQ es una versión del Problema RMQ en el que el arreglo debe cumplir una restricción por la cual el valor de cada posición debe ser ± 1 el valor de la posición anterior.

Para transformar el problema LCA en el problema RRMQ se hace una recorrida euleriana del árbol (depth first). Cada vez que se visita un nodo, tanto en el sentido padre–hijo como en el sentido hijo–padre, se agrega una entrada en el arreglo cuyo valor es la profundidad del nodo. Dos posiciones consecutivas del arreglo corresponden a los extremos de una rama y por lo tanto cumplen la restricción requerida por el problema. Llamemos `depth` al arreglo. La primera y la última posición corresponderían a la raíz del árbol. La última no hace falta incluirla. En la Figura A.1 los nodos del árbol están numerados de acuerdo a la recorrida euleriana.

Para cada nodo hay una posición en `depth` por cada rama que incide en el nodo. Por cada hoja hay una posición y por cada nodo interno distinto de la raíz hay una por cada hijo y otra más para volver al padre. Para la raíz hay una por cada hijo. El tamaño de `depth` es $2 \cdot (|nodos| - 1)$. Para ver esto hay que tener en cuenta que la cantidad de ramas es $|nodos| - 1$ y que cada rama se recorre dos veces, una en cada sentido.

Para encontrar el `lca` de dos nodos del árbol es necesario poder encontrar en `depth` una posición correspondiente a cada uno de ellos. Esto significa que hay que mantener una estructura que, dado un nodo del árbol encuentre una posición que le corresponda, por ejemplo la primera, en `depth`. Llamemos `firstpos` a esta estructura que es un arreglo con una posición por cada nodo. El `lca` buscado tiene en el rango definido por las dos posiciones (incluidas) al menos una posición. Todas las posiciones del rango corresponden a descendientes del nodo buscado, por lo que todas las que tengan el valor mínimo corresponden al `lca`. Hay que encontrar una de ellas, por ejemplo la última, y hay que poder determinar a qué nodo corresponde. O sea hace falta un mapeo de $[1 \dots p] \rightarrow [1 \dots |nodos|]$, donde p es la cantidad de posiciones de `depth`. Esto se resuelve con un arreglo del mismo tamaño que `depth`, al que llamamos `nodes`. En la figura, el rango de `depth` definido por los nodos 4 y 9 va desde la posición 5 hasta la 15 y en ese rango hay dos posiciones con valor 0 que corresponden como se ve en `nodes` al nodo 1, que es el `lca`.

Falta determinar cuál es la rama por la cual cada nodo desciende del `lca` encontrado. Se mantiene una estructura que asocie cada nodo distinto de la raíz con la rama por la cual desciende de su padre. Es un arreglo que llamamos `branch`. En `nodes` y `depth`, para cada nodo interno hay una posición para cada rama que desciende de él. La posición siguiente indica en `nodes` cuál es el hijo que desciende por esa rama. Con `branch` se obtiene la rama por la cual cada



nodes	1	2	3	2	4	2	1	5	1	6	7	6	8	6	9	6
depth	0	1	2	1	2	1	0	1	0	1	2	1	2	1	2	1

branch		a	a	c	b	c	a	b	c
firstpos	1	2	3	5	8	10	11	13	15

Figura A.1: Transformación del problema LCA en el problema RRMQ

nodo desciende de su padre. Si el nodo u es ancestro del nodo v, el `lca` de ambos es u, y aparece primero en `nodes`. En el rango determinado por ambos nodos las posiciones que en `depth` tienen valor mínimo corresponden a u. La última de estas posiciones (que, tal como se anticipó es la que se obtiene mediante `rrmq`) corresponde al inicio de la rama que es la primera en el camino que conecta ambos nodos. El otro extremo de la rama corresponde al nodo que está en la siguiente posición en `nodes`. Este nodo es el hijo de u que es ancestro de v. Al conocer ese nodo, se obtiene en `branch` la rama buscada. Por ejemplo en el rango definido por los nodos 1 y 8 hay 3 posiciones en `depth` con valor 0. La posición siguiente a la última corresponde en `nodes` al nodo 6, que como se ve en `branch` desciende por la rama c.

Para resolver la Función (A.1) sea N la lista de nodos. Supongamos que es un arreglo cuyas entradas son enteros que identifican nodos del árbol. En orden lexicográfico el primero y el último de esos nodos son $a = \text{mín}_i \{\texttt{firstpos}[N[i]]\}$ y $b = \text{máx}_i \{\texttt{firstpos}[N[i]]\}$.

Por lo tanto, recordando el Lema 4.22, el `lca` de los nodos de N es

$$lca = \texttt{nodes}[\texttt{rrmq}(a, b)]$$

y las ramas por las que descienden son

$$B[i] = \texttt{branch}[\texttt{nodes}[1 + \texttt{rrmq}(f, \texttt{firstpos}[N[i]])]]$$

siendo $f = \texttt{firstpos}[lca]$.

Por lo tanto hay que hacer una llamada a `rrmq` por cada elemento de la lista de entrada. Como el tamaño del alfabeto es acotado la complejidad de esta operacion es $O(1)$ si también lo es el cálculo de `rrmq`.

RRMQ Al arreglo `depth` de tamaño p se lo parte en $m = p/l$ bloques de longitud l. Se verá que conviene que la magnitud de l sea $O(\log p)$.

En tiempo $O(p)$ se obtiene la posición del mínimo de cada bloque. Se construye una tabla que devuelva la posición del mínimo entre dos bloques si la distancia entre ellos es potencia de 2. Es decir, en esa tabla está la posición en `depth` en donde está el mínimo entre el bloque i y los bloques $i+1, i+2, i+4, \ldots$. El espacio que ocupa esta tabla es $O(m \cdot \log m)$. Se la puede construir mediante programación dinámica también en tiempo $O(m \cdot \log m)$. La llamamos `rpow2` y tiene una fila por cada potencia de 2 y una columna por cada bloque. Si la longitud de los bloques es $l = O(\log p)$, entonces el espacio ocupado por la tabla y el tiempo necesario para construirla es $O(p)$.

Supongamos que se quiere encontrar la posición del mínimo entre los bloques i y j incluidos. Sea $k = \lfloor \log(j-i) \rfloor$. De `rpow2` se conoce la posición del mínimo entre los bloques i e $i + 2^k$, y la posición del mínimo entre los bloques $j - 2^k$ y j. El menor de los valores en `depth` de las dos posiciones determina cual de los dos es la posición del mínimo en todo el rango.

En el Problema RRMQ se quiere encontrar la posición del mínimo entre las posiciones que en `depth` están en x e y, donde suponemos $x < y$. El bloque, que está entre 0 y $m-1$, y la posición dentro del bloque, que está entre 0 y $l-1$, de cada uno de los parámetros es:

$$\begin{aligned} bx &= \lfloor (x-1)/l \rfloor, \quad px = (x-1) \bmod l, \\ by &= \lfloor (y-1)/l \rfloor, \quad py = (y-1) \bmod l \end{aligned} \tag{A.2}$$

donde se resta 1 a las posiciones x e y porque están en el rango $[1 \ldots p]$.

Cuando dos posiciones pertenezcan al mismo bloque se debe poder resolver el problema en el interior de cada bloque. Para esto para cada bloque se mantiene una tabla de tamaño $\frac{l(l-1)}{2}$, o sea $O(l^2)$. La tabla que corresponde al bloque b es `bloque`$[b]$, con $0 \leq b \leq m-1$. En `bloque`$[b][i][j]$ con $0 \leq i < j \leq l-1$, se encuentra la posición en la que está el mínimo entre i y j en b.

Entonces si x e y pertenecen al mismo bloque el mínimo está en la posición de `depth`

$$1 + bx \cdot l + \texttt{bloque}[bx][px][py]. \tag{A.3}$$

Si en cambio x e y pertenecen a bloques diferentes el problema se descompone en encontrar las posiciones de tres mínimos y el resultado es la posición en la que está el menor de ellos. Se debe encontrar las posiciones en las que se encuentra el mínimo

- entre x y el fin de su bloque;

- entre el bloque siguiente al de x y el bloque anterior al de y, ambos incluidos;
- entre y y el inicio de su bloque.

El segundo mínimo se obtiene usando la tabla `rpow2`. Se calcula

$$k = \lfloor \log((by - 1) - (bx + 1)) \rfloor ,$$

y el mínimo en `depth` está en

$$\begin{array}{l} \texttt{rpow2}[k][bx + 1] \\ \text{o en} \\ \texttt{rpow2}[k][by - 1 - 2^k]. \end{array} \tag{A.4}$$

El primero y el tercero son mínimos interiores a los bloques y corresponden respectivamente a las posiciones en `depth`

$$\begin{array}{l} 1 + bx \cdot l + \texttt{bloque}[bx][px][l - 1] \\ 1 + by \cdot l + \texttt{bloque}[by][0][py]. \end{array} \tag{A.5}$$

Por lo tanto el mínimo está o bien en la posición dada en la expresión (A.3) si x e y están en el mismo bloque, o bien, si están en bloques distintos, en la posición en la que está el mínimo entre las posiciones dadas por las ecuaciones (A.4) y (A.5).

Como sólo se realiza una cantidad constante de comparaciones y búsquedas en tablas el Problema RRMQ, y por lo tanto también el Problema LCA, se resuelve en tiempo $O(1)$.

Tablas canónicas El inconveniente se presenta al intentar cumplir con el objetivo de espacio requerido y tiempo de preprocesamiento $O(n)$ porque en principio debería haber m tablas, una para cada bloque.

Pero si al valor de cada posición de un bloque se resta el valor de la posición inicial se obtiene otro bloque que empieza con 0. El mínimo entre dos posiciones en este bloque está en la misma posición que el mínimo entre las mismas posiciones en el bloque original. Debido a la restricción que debe cumplir el arreglo original por la cual el valor de cada posición es ± 1 el valor de la posición anterior, sólo hay 2^{l-1} posibles bloques que comienzan con 0, llamémosles *canónicos*. Entonces, en lugar de $m = p/l$ sólo hacen falta 2^{l-1} tablas de tamaño $O(l^2)$. Si $l = \frac{\log p}{2}$, el espacio que ocupan las $\frac{1}{2}\sqrt{p}$ tablas es $O(\sqrt{p} \log^2 p)$, que es $o(p)$. Al conjunto de estas tablas le llamamos `canonics`.

Por ejemplo si $l = 3$ hay 4 tablas canónicas: $[0, -1, -2]$, $[0, -1, 0]$, $[0, 1, 0]$ y $[0, 1, 2]$. La tabla `depth` de la Figura A.1 queda partida en bloques de rangos $[3i + 1 \ldots 3(i + 1)]$, con $i \geq 0$. La tabla canónica $[0, 1, 0]$ representaría en `depth` al bloque $[0, 1, 0]$ que está entre las posiciones 7 y 9, pero también a los dos bloques $[1, 2, 1]$ uno de los cuales está entre las posiciones 4 y 6, y el otro entre las posiciones 10 y 12.

Como se ve, estas tablas no dependen de la secuencia a procesar, sino sólo del largo de los bloques. Esto implica que si se pretende procesar varias secuencias de igual longitud estas tablas sólo deben calcularse una vez.

Además es necesario un mapeo que a cada bloque le asigne su tabla canónica. Se resuelve con un arreglo de tamaño m al que llamamos `b2c`. El mapeo de cada bloque a su tabla canónica se hace en tiempo $O(l)$, por lo que todo el mapeo se completa en tiempo $O(p)$.

En las ecuaciones (A.3) y (A.5) se debe sustituir las ocurrencias de `bloque`$[\cdot]$ por `canonics`$[$`b2c`$[\cdot]]$.

Con la construcción de `canonics` y `b2c` queda terminado el preprocesamiento que se hace en tiempo y espacio $O(p)$. Recordando que para esta tesis $p = O(n)$, siendo n el tamaño de la secuencia procesada (igual a la cantidad de hojas del árbol) se cumple el objetivo planteado.