Introduccion a la teoria de la computacion: via palindromos

J. Andres Montoya

Universidad Industrial de Santander, Bucaramanga, Colombia,
juamonto@uis.edu.co

Abstract. En este escrito se estudia, de manera casi exhaustiva, la complejidad computacional de reconocer el lenguaje de los palindromos

El objetivo de este escrito es multiple: estudiaremos, con un cierto nivel de profundidad, la teoria de automatas y modelos uniformes de computacion; estudiaremos la nocion de *tiempo real*, y por ultimo revisaremos, de manera casi exhaustiva[1], los trabajos referentes a la complejidad computacional de reconocer palindromos sobre modelos secuenciales de computacion, haciendo enfasis en estudiar la recognizabiliad en tiempo real de este lenguaje.

Hemos elegido el problema de reconocer palindromos como una drosophila (*leit motiv*) que nos permita evaluar y estudiar conceptos, metodos y modelos propios de la teoria de la computacion y de la teoria de la complejidad computacional.

La complejidad computacional intenta clasificar los problemas algoritmicos de acuerdo a su dificultad intrinseca, esto es: de acuerdo a la cantidad de recursos computacionales que es necesario emplear para resolverlos. Al analizar un problema los practicantes de la teoria intentan establecer cotas superiores (*tal cantidad de recursos es suficiente*) y cotas inferiores (*tal cantidad de recursos es insuficiente*) que permitan estimar la complejidad intrinseca del problema, tal estimacion sera posible si las cotas son ajustadas, esto es: si la brecha existente entre estas cotas es negligible o, en el mejor de los casos, nula.

La historia de la complejidad computacional, en sus casi 40 años (en el año 2011 se cumplen 40 años de la publicacion del historico articulo de Cook [7] y la definicion de la clase NP), es la historia de una elusiva busqueda de cotas inferiores. Puede conjeturarse que la mayor parte de las cotas superiores obtenidas hasta el momento son ajustadas y que son nuestras debiles cotas inferiores las responsables de la gigantesca brecha existente entre las cotas obtenidas hasta el momento. Son muy pocos los problemas para los cuales la complejidad computacional puede brindar un clara estimacion-cuantificacion de su complejidad intrinseca, uno de estos pocos problemas es el reconocimento de palindromos que denotaremos con el simbolo Pal.

Sea L un problema de decision, diremos que L es *no trivial* si y solo si dada x una instancia de L, existen palabras y y z tales que xy es una instancia negativa

[1] Desde Poincare, el ultimo de los matematicos en saberlo todo, ya no es posible ser exhaustivo.

de L ($xy \notin L$) y xz es una instancia positiva de L ($xz \in L$). Intutivamente, un problema L es no trivial si al procesar una instancia x es necesario leer la palabra x en su totalidad, (cada uno de sus caracteres), antes de poder tomar una decision. Un problema L requiere, como minimo, tiempo real para ser resuelto sobre un modelo secuencial de computacion.

Dado L un problema no trivial y dado $\mathcal{M}$ un algoritmo que resuelve L diremos que $\mathcal{M}$ es un algoritmo de tiempo real (la nocion de tiempo real fue introducida hace ya 50 años por H. Yamada [54]) si y solo si al procesar una instancia x el algoritmo $\mathcal{M}$ emplea $|x|$ unidades de tiempo (realiza $|x|$ transiciones). Tiempo real es una cota inferior para todo problema no trivial (*for free*). Note que si fijamos un modelo uniforme y secuencial de computacion, digamos $\mathcal{C}$, y logramos probar la $\mathcal{C}$-computabilidad en tiempo real del problema Pal, habremos entonces logrado establecer un par de cotas ajustadas: la cota inferior (tiempo real) y la cota superior (tiempo real) cazan perfectamente.

El problema Pal es un modelo de juguete de la *stringologia* y la complejidad computacional: para este problema, a diferencia de para la mayoria de problemas algoritmicos no triviales, las cotas inferiores y superiores sobre (casi) cualquier modelo uniforme de computacion cazan perfectamente. Esta caracteristica del problema Pal nos impulsa a creer que este es un problema particularmente adecuado en terminos pedagogicos, dado que nos permite recorrer un amplio espectro de modelos de computacion exhibiendo de manera explicita el objetivo que quisieramos alcanzar al analizar cualquier problema sobre un modelo de computacion especifico: establecer cotas inferiores y superiores ajustadas.

Hemos hecho referencia en los parrafos anteriores a la nocion de modelo uniforme y secuencial de computacion. Los modelos no uniformes (clases de circuitos [39]) no corresponden a la nocion de algoritmo y por ello no son interesantes cuando lo que se pretende es analizar un problema concreto (son interesantes desde el punto de vista estructural, cuando se comparan clases no uniformes con clases uniformes). Adicionalnente el problema Pal es trivial desde el punto de vista uniforme ($Pal \in AC^0$: Pal puede ser reconocido usando una familia uniforme de circuitos de profundidad acotada [45]). En los paragrafos anteriores tambien hemos mencionado el termino *modelo no secuencial* para referirnos a los modelos de computacion paralela. En estos ultimos, una red de procesadores cuyo tamaño escala con el tamaño de los inputs, se encarga de procesar las intancias del problema. La habilidad que tienen estas redes de partir el input en pequeños fragmentos, y de analizar estos fragmentos de manera simultanea usando diferentes procesadores dentro de la red, permite que algunos problemas no triviales puedan ser resueltos en tiempo *sublineal*, e incluso *(poly)logaritmico*. Aunque puede ser muy interesante analizar la *complejidad paralela* de Pal y sus relativos no lo haremos en este escrito, algunos trabajos importantes en esta area son las referencias [31], [3]. Es necesario comentar en este punto, que si estudiaremos la complejidad de Pal desde el punto de vista de los *automatas celulares*, los cuales pese a ser considerados modelos de paralelismo masivo pueden ser entendidos como modelos secuenciales en los que cada procesador recibe un numero finito de bits por unidad de tiempo, las señales se transmiten de manera secuencial,

y no existen ni procesadores centrales ni dispositivos de entrada especiales a los que cada uno de los procesadores en el arreglo puede acceder en una unidad de tiempo a cualquiera de sus registros: un automata celular es una secuencia de procesadores (automatas regulares) y no una red de procesadores.

El reconocimiento de palindromos es un problema que, pese a su aparente sencillez, tiene importantes aplicaciones como por ejemplo en algoritmos de compresion de textos (por ejemplo en el algoritmo de Lempel-Ziv [33]) o en clasificacion de secuencias genomicas [4]. Entendemos que, en terminos de las posibles aplicaciones, la computacion en paralelo es de vital importancia, aun asi hemos escogido no estudiar este tema en este libro porque el tema central del libro es computaciones de tiempo real usando maquinas secuenciales, y metodos para probar cotas inferiores (resultados de imposibilidad) sobre este tipo de modelos. El reconocimiento de palindromos, lo repetimos, es nuestra drosophila y no el tema central de estas notas.

Organizacion del escrito.

El escrito esta organizado de la siguiente manera, en las primeras secciones estudiaremos cinco tipos diferentes de automatas: regulares, regulares no deterministicos, regulares de doble via, de pila y de pila no deterministicos. Estudiremos la computabilidad, y la computabilidad en tiempo real, del problema Pal sobre cada uno de estos modelos. A continuacion introduciremos el modelo de maquinas de Turing de una sola cinta, mostraremos que este modelo es incapaz de resolver Pal en tiempo lineal y que (Teorema de Hennie [26]) toda maquina de una sola cinta capaz de resolver Pal requiere tiempo cuadratico. En la siguiente seccion introduciremos el modelo de maquinas de Turing con una sola cinta bidimensional y demostraremos una version bidimensional del teorema de Hennie (Pal requiere tiempo $\Omega\left(\frac{n^2}{\log(n)}\right)$), adicionalmente exhibiremos una maquina de Turing bidimensional (ingeniosamente diseñada) que reconoce el lenguaje de los palindromos en tiempo $O\left(\frac{n^2}{\log(n)}\right)$, obteniendo con ello cotas ajustadas sobre este modelo de computacion. En este punto es importante comentar que existe una maquina de Turing unidimensinal que reconoce Pal en tiempo cuadratico, por lo que podemos afirmar que la cota inferior obtenida via el teorema de Hennie es ajustada. A continuacion introduciremos el modelo de maquinas probabilisticas de una sola cinta, estudiaremos el algoritmo de Pippenger que probabilisticamente reconoce palindromos en tiempo $O(n\log(n))$, comentaremos un teorema de Yao [55] quien prueba que toda maquina de Turing probabilistica de una sola cinta que reconozca palindromos requiere tiempo $\Omega(n\log(n))$, obteniendo con ello, una vez mas, un par de cotas ajustadas. Cerraremos la primera parte de este escrito (maquinas de una sola cinta) introduciendo tres modelos de automatas celulares: automatas celulares unidimensionales, automatas celulares unidimensionales y unidireccionales y arreglos iterados multidimensionales. En esta seccion probaremos, entre otras cosas, que Pal puede ser reconocido en tiempo real en cada uno de estos tres modelos.

En la segunda parte del escrito introduciremos el modelo de maquinas de Turing con varias cintas, Modelo en el cual la nocion de tiempo real es partic-

ularmente interesante y robusta, y probaremos el teorema de Galil-Slisenko: el lenguaje Pal puede ser reconocido en tiempo real usando una maquina de Turing con varias cintas.

Hemos incluido un capitulo adicional para estudiar algunos problemas algoritmicos, relacionados con palindromos, que no son problemas de reconocimiento: incluimos un problema de conteo, un problema de enumeracion y uno de optimizacion. Para cada uno de estos problemas exhibimos algoritmos de tiempo lineal que los resuelven, (los algoritmos ingenuos para enfrentar estos problemas son cuadraticos).

Finalmente hemos incluido un apendice en el que, de manera muy breve y veloz, introducimos (de manera intuitiva) algunos conceptos y metodos de la computacion en paralelo, que usamos luego para resolver en tiempo sublineal algunos problemas relacionados con palindromos.

Remark 1. Es importante aclarar en este punto, que hemos decidido incluir a los automatas de pila dentro de la clase de las maquinas de una sola cinta dado que la pila con todas sus limitaciones no es una cinta adicional, esto es: un automata de pila es una maquina con una sola cinta y una pila.

Sobre la importancia de los problemas impertinentes.

El lector puede pensar, al iniciar la lectura de este libro, que el lenguaje de los palindromos es un lenguaje trivial sin interes practico acerca del cual la teoria de la computacion tiene muy poco por decir. Una de los objetivos de este escrito es acabar con este prejuicio.

Una de las razones por las cuales hemos elegido el lenguaje de los palindromos es el rol singular que ha desempeñado este lenguaje en la teoria de lenguajes formales. El lenguaje de los palindromos ha sido estudiado desde casi todos los modelos de automatas y maquinas secuenciales porque su equilibrada combinacion de simpleza y complejidad le han permitido ser un elemento fundamental en la mayoria de las pruebas de separacion entre clases de automatas.

Cuando se quiere probar que un cierto tipo de automata es mas potente que un segundo tipo de maquinas, es necesario exhibir un lenguaje que pueda ser reconocido usando el primer tipo de dispositivos pero que no pueda serlo usando el segundo. Es necesario entonces contar con un candidato para tal fin. El lenguaje de los palindromos siempre sera un buen candidato. Esto es asi dado que es suficientemente simple como para que pueda ser reconocido usando un tipo especial de automata, y es suficientemente complejo como para que no pueda ser reconocido usando un segundo tipo de automatas ligeramente mas debiles que los del primer tipo. En realidad el lenguaje de los palindromos, o alguno de sus relativos, ha desempeñado un rol significativo en casi todas las pruebas de separacion del tipo antes discutido.

Esta caracteristica del lenguaje Pal nos permite, sin abandonar nuestro leit motiv, recorrer casi en su totalidad el amplio espectro de modelos de automatas introducidos en la literatura hasta el dia de hoy.

Finalmente es importante señalar que el lenguaje de los palindromos es importante en algunas aplicaciones, el juega un papel importante en los algoritmos

de compresion (Lempel-Ziv por ejemplo [33]), y la tarea consistente en reconocer palindromos presentes en cadenas largas es una tarea que ha mostrado ser importante en bioinformatica dado que, entre otras cosas, la presencia de palindromos largos en cadenas de DNA esta asociada al surgimiento de canceres en humanos [49].

Objetivos y alcances del texto. El texto fue diseñado pensando en un curso introductorio a la teoria de los lenguajes formales y modelos de computacion. Es posible escribir un tal texto desde multiples perspectivas, dependiendo la eleccion de una tal perspectiva, del area de expertisia de sus autores. Nosotros como practicantes de la teoria de la complejidad computacional hemos escrito un texto cesgado en esta direccion, es por ello que creemos que el texto puede ser usado idealmente como texto guia en un curso de teoria de la computacion que anteceda a cursos en Complejidad Computacional, Stringologia y Bioinformatica.

El libro es menos apropiado para ser usado en cursos de la teoria de la computacion que antecedan a cursos en teoria de lenguajes de programacion, parsing, compiladores, linguistica computacional etc.

Por su brevedad el libro puede ser cubierto en un curso de 16 semanas, creemos que cubrir la totalidad del libro puede brindar al estudiante un panorama amplio de las ciencias de la computacion.

Agradecimientos. Este libro esta dedicado a Hijodimitriou. Los autores quisieran agradecer a todos sus estudiantes quienes de una forma u otra hicieron posible la realizacion de este trabajo. Agradecimientos especiales a Joerg Flum, Anahi Gajardo, Moritz Mueller, DAAD, DFG y VIE-UIS.

Carolina Mejia, J. Andres Montoya
Bucarmanga, Agosto 2011

Maquinas de una cinta

Iniciaremos nuestro recorrido por el universo de las maquinas de una sola cinta introduciendo la nocion de *automata regular* (*automata de estado finito*).

Proviso. *A todo lo largo del libro, a menos que se especifique lo contrario, el simbolo* Σ *denotara el alfabeto* $\{0,1\}$. *De igual manera, a todo lo largo del libro (a menos que se especifique lo contrario) asumiremos que el alfabeto de la maquina, o lenguaje, bajo estudio es el alfabeto binario* $\{0,1\}$.

Notacion. *Usaremos el simbolo* ϵ *para denotar la palabra vacia.*

1 Palindromos automatas finitos y lenguajes regulares

Los automatas regulares pueden ser considerados como el modelo uniforme de computacion de menor poder. Un automata regular $\mathcal{M}$ es un quintupla $(Q, q_0, \Sigma, F, \delta)$, donde:

1. Q es un conjunto finito, el *conjunto de estados* del automata $\mathcal{M}$.
2. $q_0 \in Q$, es un elemento distinguido de Q que sera llamado el *estado inicial* del automata $\mathcal{M}$.
3. Σ es un conjunto finito, el *alfabeto* del automata.
4. $F \subseteq Q$ es el conjunto de *estado finales* o *estados de aceptacion* del automata $\mathcal{M}$.
5. δ, la *funcion de transicion*, es una funcion de $\Sigma \times Q$ en Q.

Dado $\mathcal{M}$ un automata regular supondremos que $\mathcal{M}$ posee una cinta semi-infinita constituida por una cantidad enumerable de celdas, supondremos ademas que la cinta se extiende de izquierda a derecha empezando en una celda inicial. Cada celda de $\mathcal{M}$ tiene la capacidad de albergar un unico caracter del conjunto $\Sigma \cup \{\square\}$, donde $\square$ es un caracter especial que denota espacio en blanco o, lo que es lo mismo, celda vacia. Supondremos que al comienzo de la computacion el input $w = w_1 w_2 ... w_n$ se encuentra ubicado en el extremo izquierdo de la cinta ocupando los primeros n caracteres. Tambien supondremos que al comienzo de la computacion la cabeza lectora de la maquina se encuentra ubicada sobre la primera celda leyendo el caracter w_1 y que el estado de la maquina es el *estado interno* inicial q_0. Al comenzar la computacion la cabeza lectora de la maquina se desplaza a la derecha de celda en celda leyendo cada caracter. La computacion termina cuando la maquina encuentra la primera celda ocupada por un caracter $\square$, esto es cuando la cabeza lectora llega al final del input. El caracter $\square$ sera usado para denotar el espacio en blanco y para indicar que la computacion debe terminar. Al terminar la computacion la maquina se encuentra en un estado interno $q \in Q$. Si $q \in F$, la maquina acepta el input, en caso contrario lo rechaza.

1.1 El lema de bombeo para lenguajes regulares

Empezaremos esta seccion introduciendo la definicion de lenguaje regular

Definition 1. *(lenguajes regulares)*

1. $L(\mathcal{M}) = \{w \in \Sigma^* : \mathcal{M} \text{ acepta } w\}$. *Dado* $\mathcal{M}$, *diremos que* $L(\mathcal{M})$ *es el lenguaje reconocido por el automata* $\mathcal{M}$.
2. *Dado* $L \subseteq \Sigma^*$, *diremos que* L *es un lenguaje regular si y solo si existe* $\mathcal{M}$, *un automata regular, tal que* $L = L(\mathcal{M})$.

Definition 2. *Dada* $\mathcal{M}$ *una maquina y dado* L *un lenguaje diremos que* $\mathcal{M}$ *reconoce* L *en tiempo real si y solo si el numero de transiciones realizadas por* $\mathcal{M}$ *al procesar un input* w *esta acotado por* $|w|$.

Note que todo automata regular $\mathcal{M}$ reconoce el lenguaje $L(\mathcal{M})$ en tiempo real.

Definition 3. *Dada* $w = w_1...w_n$ *una palabra, un factor de* w *es un fragmento* $w_i...w_j$ *con* $1 \leq i \leq j \leq n$. *A lo largo del libro usaremos el simbolo* $w[i...j]$ *para denotar el factor* $w_i...w_j$ *de* w.

¿como probar que un lenguaje dado no es regular? El lema de bombeo es una herramienta poderosa que puede ser usada para tal fin.

Theorem 1. *(lema de bombeo para lenguajes regulares)*
Dado L *un lenguaje regular, existe* $K \in \mathbb{N}$ *tal que si* $w \in L$ *y* $|w| \geq K$, *existen entonces* $u, v, s \in \Sigma^*$ *para las cuales se satisface lo siguiente*

1. $x = uvs$.
2. $|v| \gneqq 0$ *y* $|uv| \leq K$
3. *Para todo* $n \in \mathbb{N}$ *se tiene que* $uv^n s \in L$.

Proof. Sea $\mathcal{M} = (Q, q_0, \Sigma, F, \delta)$ un automata regular que reconoce el lenguaje L. Al automata $\mathcal{M}$ podemos asociarle un digrafo etiquetado $G[\mathcal{M}]$ el cual esta definido por

- $V(G[\mathcal{M}]) = Q$.
- Dados $q, p \in V(G[\mathcal{M}])$ y dado $a \in \Sigma$ existe una arista de q a p con etiqueta a si y solo si $\delta((a, q)) = p$.

Note que una computacion de $\mathcal{M}$ es simplemente un camino finito en $G[\mathcal{M}]$ con origen en q_0. Note tambien que todo camino finito con origen en q_0 es una computacion de $\mathcal{M}$, y que las computaciones aceptantes estan en correspondencia biyectiva con los caminos de este tipo que tienen su vertice final dentro del conjunto $F \subset V(G[\mathcal{M}])$.

Afirmamos que el K, en el enunciado del lema, es igual a $|Q|+1$. Dado que el grafo $G[\mathcal{M}]$ tiene tamaño $|Q|$ todo camino de longitud mayor o igual que $|Q|+1$ debe pasar al menos dos veces por un mismo vertice, digamos p. Podemos partir un tal camino en tres fragmentos a saber.

1. Fragmento inicial, que va de q_0 hasta la primera visita al vertice p.
2. Fragmento intermedio, el bucle que va de la primera visita a p hasta la segunda visita a p.
3. Fragmento final, desde la segunda visita al vertice p hasta el vertice final.

Dada $w \in L$, si $|w| \geq |Q|+1$ el camino determinado por w, al que denotaremos λ_w, puede partirse en tres fragmentos: el inicio, el bucle y el final. Estos tres fragmentos corresponden a factores de w, digamos u, v y s. Dado $n \in \mathbb{N}$, el camino determinado por la palabra $uv^n s$ tiene la siguiente estructura.

1. El inicio es identico al inicio del camino λ_w.
2. El fragmento intermedio corresponde a recorrer n veces el fragmento intermedio de λ_w (recuerde que este fragmento intermedio es un bucle y por lo tanto es posible recorrer este bucle tantas veces como sea necesario).
3. El fragmento final es identico al fragmento final de w (o mejor, del camino determinado por x)

Ahora, dado que $w \in L$ el vertice final de λ_w pertenece a F. Note que el vertice final del camino $\lambda_{uv^n s}$ es identico al vertice final del camino λ_w, por lo que la computacion de $\mathcal{M}$ con input $uv^n s$ es tambien una computacion aceptante y esto implica que $uv^n s \in L$. Note finalmente que es posible tomar u y v de tal forma que $|uv| \leq K$, esto es asi dado que todo fragmento inicial, del camino λ_w, que tenga una longitud mayor o igual que $|Q| + 1$ debe contener un bucle

A continuacion probaremos que Pal no es regular.

Corollary 1. *(Primera cota inferior para Pal)*
No existe un automata regular que pueda reconocer el lenguaje Pal.

Proof. Suponga que Pal es regular y sea K como en el enunciado del lema de bombeo. Considere $w = 1^{K+1}01^{K+1}$. El lema de bombeo afirma que existen palabras u, v, y w tales que $w = uvs$, $|uv| \leq K$ y tales que dado $i \geq 2$ la palabra $uv^i s \in Pal$. Note que la desigualdad $|uv| \leq K$ implica que v esta totalmente contenido en el factor izquierdo de w constituido por $K + 1$ unos. Tenemos entonces que $uv^i s = 1^{K+|v|i-|v|}01^K$ que no es un palindromo dado que $|v| \geq 1$ e $i \geq 2$

Hemos afirmado, del corolario anterior, que este es una cota inferior para Pal, esto es asi dado que el establece que los recursos computacionales que definen la clase de los automatas regulares no son suficientes para reconocer el lenguaje Pal, para ello se requiere algo mas, se necesita mayor poder de computo.

No es de extrañar que el lenguaje Pal no pueda ser reconocido por un automata regular. Note que en la definicion de automata regular hemos impuesto demasiadas limitaciones al poder de computo de este tipo de dispositivos, algunas de las mas prominentes limitaciones son:

– La cabeza lectora solo puede moverse unicamente en una direccion, esto es de izquierda a derecha.

- La maquina no tiene un dispositivo de memoria externo.
- La maquina no tiene funciones de escritura
- La maquina no puede marcar (o transformar) fragmentos o caracteres del input.

En lo que sigue estudiaremos modelos que podemos concebir como obtenidos a partir de nuestro modelo basico (la clase de los automatas regulares) eliminando una o varias de estas (y otras) limitaciones.

2 Automatas regulares no deterministicos

En esta seccion estudiaremos un segundo modelo de computacion, obtenido a partir del modelo basico, adicionando a este una habilidad: transiciones no deterministicas.

Un automata regular no deterministico es un automata $\mathcal{M} = (Q, q_0, \Sigma, F, \delta)$ para el cual la relacion de transicion δ no necesariamente es funcional, esto es: el automata $\mathcal{M}$ esta constituido por un conjunto de estados Q, un estado inicial q_0, un conjunto de estados aceptantes F y una relacion de transicion $\delta \subseteq \Sigma \times Q \times Q$.

Es discutible que el no-determinismo sea una habilidad computacional y que las maquinas no deterministicas puedan llegar a ser mas potentes que las deterministicas (para el caso de maquinas de Turing de tiempo polinomial la pregunta: ¿el no-determinismo incrementa el poder de computo? No es otra cosa que la famosa pregunta: ¿es P diferente de NP?). Existen al menos dos maneras de pensar el nodeterminismo:

- Una maquina no deterministica es una maquina capaz de realizar adivinanzas afortunadas.
- Una maquina no deterministica es una maquina capaz de realizar en paralelo un gran numero (una cantidad exponencial) de computaciones.

Es claro que desde cualquiera de las dos perpectivas anteriores el no-determinismo aparece como una habilidad adicional (un recurso computacional) que puede (y debe) incrementar el poder de nuestras maquinas.

Sea $\mathcal{M} = (Q, q_0, \Sigma, F, \delta)$ un automata no deterministico.

Definition 4. *Dado $w = w_1...w_n$ un input de $\mathcal{M}$, una computacion de $\mathcal{M}$ con input w es una secuencia $q_0 q_1 ... q_n$ de elementos de Q tal que para todo $i \leq n$ se tiene que $(w_i, q_{i-1}, q_i) \in \delta$.*

Una computacion es entonces una secuencia finita y ordenada de *configuraciones* que adicionalemente satisface la siguiente condicion: dos configuraciones consecutivas en esta secuencia estan conectadas por una transicion de la maquina.

La nocion de configuracion depende del modelo de computacion. Intuitivamente, una configuracion es una descripcion completa del estado de la computacion en un instante dado. En el caso de los automatas regulares es suficiente

especificar el estado (estado interno, elemento de Q) en el que se encuentra la maquina para describir completamente el estado de la computacion. Esto es asi dado que en el instante t de la computacion la cabeza lectora de la maquina se encuentra sobre la t-esima celda de su cinta (leyendo el t-esimo caracter) y el contenido de la cinta en este instante es identico al contenido de la cinta en el instante 0. Esto implica que el unico parametro que nosotros podriamos desconocer es precisamente el estado interno de la maquina, y es este parametro (y es precisamente lo desconocido) lo que la configuracion describe (y debe describir).

Definition 5. *Dado $\mathcal{M}$ un automata regular no deterministico y dado $w \in \Sigma^*$, diremos que $\mathcal{M}$ acepta w si y solo si existe una computacion de $\mathcal{M}$, con input w, que termina en un estado de aceptacion.*

En esta seccion probaremos que los automatas regulares no deterministicos y los automatas regulares deterministicos tienen exactamente el mismo poder de computo.

Theorem 2. *Todo automata regular no deterministico puede ser simulado por un automata regular.*

Proof. Sea L un lenguaje reconocible mediante un automata regular no deterministico y sea $\mathcal{M} = (Q, q_0, \Sigma, F, \delta)$ un automata no deterministico que reconoce a L. Considere el automata regular $\mathcal{M}^{\det} = \left(Q^{\det}, q_0^{\det}, \Sigma, F^{\det}, \delta^{\det}\right)$ definido por:

1. $Q^{\det} = \mathcal{P}(Q)$.
2. $q_0^{\det} = \{q_0\}$.
3. $F^{\det} = \{A \subset Q : A \cap F \neq \varnothing\}$.
4. La funcion $\delta^{\det}$ es definida por

$$\delta^{\det}((a, A)) = \{q \in Q : \exists p \in A((a, p, q) \in \delta = q)\}$$

Es facil verificar que $L(\mathcal{M}) = L\left(\mathcal{M}^{\det}\right)$.

El siguiente corolario es una consecuencia inmediata de los dos teoremas anteriores

Corollary 2. *(Segunda cota inferior para Pal)*
No existe un automata regular no deterministico que pueda reconocer el lenguaje Pal.

3 Automatas regulares de doble via: equivalencias

En esta seccion introduciremos una clase de maquinas aparentemente mas potentes que los automatas regulares, a esta clase de maquinas las llamaremos automatas regulares de doble via (*2way finite automata)*. Los automatas regulares de doble via son automatas regulares provistos de una cabeza lectora que

puede moverse en las dos direcciones. Es de esperar que esta habilidad adicional confiera mayor poder de computo a estas maquinas, y es natural suponer que este poder adicional sea suficiente para reconocer el lenguaje Pal. Intuitivamente, si la cabeza lectora de la maquina puede moverse de izquierda a derecha y de derecha a izquierda, puede entonces moverse varias veces a lo largo de la cinta en las dos direcciones y verificar (o, reciprocamente, verificar que no es el caso) que para todo $i \leq |w|$, el caracter i-esimo y el caracter $(|w| - i + 1)$-esimo coinciden. El problema es que la maquina posee una memoria finita (uniformemente acotada) y no puede marcar las celdas visitadas con anterioridad. El teorema principal de esta seccion afirma que los automatas regulares de doble via tienen exactamente el mismo poder de computo que los automatas regulares (y en consecuencia los automatas regulares de doble via son incapaces de reconocer el lenguaje Pal).

Definition 6. *Un automata regular de doble via es una quintupla*

$$\mathcal{M} = (Q, q_0, \Sigma, F, A, \delta)$$

tal que Q es un conjunto finito de estados, $q_0 \in Q$ es el estado inicial, $A \subseteq F \subset Q$ y δ es una funcion de $\Sigma \times Q$ en $Q \times \{\rightarrow, \leftarrow\}$.

Dado $a \in \Sigma$ y $q \in Q$, si la maquina se encuentra leyendo el caracter a en el estado q y $\delta(a, q) = (\rightarrow, p)$, entonces la cabeza de la maquina se desplaza una posicion a la derecha y el estado interno de la maquina cambia de q a p. Por el contrario si $\delta(a, q) = (\leftarrow, p)$, la cabeza de la maquina se mueve una posicion a la izquierda. Dado que la cabeza de la maquina podria llegar al extremo derecho de la cinta y devolverse a la izquierda (o podria no llegar nunca al extremo derecho), nada nos garantiza que la computacion de la maquina terminara en algun momento, es por ello que en la definicion de los automatas regulares de doble via hemos agregado el parametro F. La idea es que si el estado interno de la maquina pertenece a F la maquina para. En particular, si la maquina accede a un estado en A, la maquina para y acepta el input, pero si accede a un estado en $F - A$ para y rechaza el input.

3.1 El teorema de Myhill-Nerode

El teorema de Myhill-Nerode puede ser entendido como una caracterizacion algebraica de los lenguajes regulares. Caracterizaciones de este tipo siempre son utiles en Matematicas. Como lo veremos mas adelante, el teorema de Myhill-Nerode puede ser usado para probar algunos hechos importantes referentes a los lenguajes regulares.

Definition 7. *Dado $L \subset \Sigma^*$, podemos definir una relacion de equivalencia sobre Σ^* asociada a L. La relacion $R_L \subseteq \Sigma^* \times \Sigma^*$ se define por*

$$xR_Ly \text{ si y solo si } \forall w\,(xw \in L \Leftrightarrow yw \in L)$$

Theorem 3. *(Myhill-Nerode)*
L es regular si y solo si $\left|\frac{\Sigma^*}{R_L}\right|$ es finito.

Proof. Sea $L \subset \Sigma^*$ tal que la relacion R_L es de *indice finito,* (es decir el conjunto $\left|\frac{\Sigma^*}{R_L}\right|$ es finito). Sea $\mathcal{M} = (Q, q_0, \Sigma, F, \delta)$ el automata regular definido por:

- $Q = \frac{\Sigma^*}{R_L}$.
- $q_0 = [\epsilon]_{R_L}$.
- $F = \left\{[x]_{R_L} : x \in L\right\}$.
- La funcion $\delta : \Sigma \times Q \to Q$ esta definida por

$$\delta\left(\left(a, [x]_{R_L}\right)\right) = [xa]_{R_L}$$

Es claro que $\mathcal{M}$ es un automata regular, adicionalmente es facil probar que $L(\mathcal{M}) = L$.

Veamos ahora que si L es un lenguaje regular entonces la relacion R_L es de indice finito. Sea $\mathcal{M}$ un automata regular que reconoce L. Dada $x \in \Sigma^*$ podemos definir una funcion $\gamma_x^{\mathcal{M}} : Q \to Q$ de la siguiente manera:

$\gamma_x^{\mathcal{M}}(q)$ es igual al estado final al que accede $\mathcal{M}$, al procesar el input x, empezando en el estado q

Es facil probar que si $\gamma_x^{\mathcal{M}} = \gamma_y^{\mathcal{M}}$ entonces xR_Ly. Tenemos entonces que $\left|\frac{\Sigma^*}{R_L}\right| \leq |Q|^{|Q|}$.

Prueba de la equivalencia En esta subseccion probaremos que la clase de los lenguajes regulares coincide con la clase de los lenguajes aceptados por automatas regulares de doble via, esto es: probaremos que todo automata regular de doble via puede ser simulado por un automata regular.

Dado $\mathcal{M}$ un automata regular de doble via definimos una relacion de equivalencia $\equiv_{\mathcal{M}} \subseteq \Sigma^* \times \Sigma^*$ de la manera siguiente:

$$x \equiv_{\mathcal{M}} y \text{ si y solo si } \forall t \in \Sigma^* \left(\mathcal{M} \text{ acepta } xt \text{ si y solo si } \mathcal{M} \text{ acepta } yt\right)$$

Theorem 4. *(Rabin-Scott-Sheperdson)*
Dado $\mathcal{M}$, un automata regular de doble via, existe un automata regular $\overline{\mathcal{M}}$ tal que $L(\mathcal{M}) = L\left(\overline{\mathcal{M}}\right)$.

Proof. Lo que nosotros probaremos es que dado $\mathcal{M} = (Q, q_0, \Sigma, F, A, \delta)$ un automata regular de doble via, el lenguaje aceptado por $\mathcal{M}$, al que denotaremos con el simbolo $L(\mathcal{M})$, satisface las condiciones impuestas en el enunciado del teorema de Myhill-Nerode.

Dada $x \in \Sigma^*$, definimos una funcion $\gamma_x : Q \to Q \cup \{0\}$ de la siguiente manera:

- Si $q \neq q_0$, se define $\gamma_x(q)$ como el estado al que accede $\mathcal{M}$, si iniciamos la computacion con x como input, con la cabeza lectora ubicada sobre el caracter final de x. Si al imponer estas condiciones iniciales, el automata $\mathcal{M}$ para con la cabeza lectora ubicada en una posicion diferente al extremo izquierdo de x, o si $\mathcal{M}$ no para del todo se define $\gamma_x(q) = 0$.

– Si $q = q_0$, definimos de manera similar $\gamma_x(q)$, la unica diferencia es que en este caso asumimos que en el instante inicial la cabeza esta ubicada en el extremo izquierdo de x.

Afirmacion. Dadas $x, y \in \Sigma^*$, se tiene que $\gamma_x = \gamma_y$ implica $x \equiv_{L(\mathcal{M})} y$.

Note que si $n = |Q|$, la afirmacion anterior implica entonces que la relacion de equivalencia $\equiv_{L(\mathcal{M})}$esta constituida por a lo mas $(n+1)^n$ clases. En este punto podemos invocar el teorema de Myhill-Nerode para afirmar que $L(\mathcal{M})$ es regular.

Corollary 3. *(tercera cota inferior para Pal)*
Ningun automata regular de doble via puede reconocer el lenguaje Pal.

El corolario anterior indica que, para reconocer el lenguaje Pal, no es suficiente con adicionarle a nuestros automatas basicos la habilidad de mover su cabeza lectora en las dos direcciones. ¿Cuales son los recursos computacionales necesarios para reconocer el lenguaje Pal?

Remark 2. Hemos probado que si $\mathcal{M} = (Q, q_0, \Sigma, F, A, \delta)$ es un automata regular de doble via entonces $\mathcal{M}$ es incapaz de reconocer Pal, aun si la funcion *tiempo de computo de* $\mathcal{M}$, que denotaremos con el simbolo $t_{\mathcal{M}}$, es no acotada o no puede ser acotada linealmente. Esto implica, en particular, que los automatas regulares de doble via son incapaces de reconocer Pal en *tiempo real*. El mismo comentario puede hacerse respecto a los automatas regulares no deterministicos, si introdujeramos la nocion de *transicion en vacio* podriamos considerar automatas regulares no deterministicos cuyo tiempo de computo no puede ser acotado linealmente, como las transiciones en vacio pueden elimininarse [28] estos automatas no son mas poderosos que los automatas regulares y por lo tanto son incapaces de reconocer Pal, i.e. los automatas regulares no deterministicos no pueden reconocer Pal ni en tiempo real ni en algun otro regimen temporal.

4 Automatas de pila

En este capitulo introduciremos un segundo modelo de computacion: los automatas de pila. Los automatas de pila se diferencian de los automatas regulares en que estos poseen un dispositivo externo de memoria (la pila). Es de esperar que este dispositivo externo de memoria incremente el poder de computo. Es plausible entonces que los automatas de pila, a diferencia de los automatas regulares, sean capaces de reconocer el lenguaje Pal.

Un *automata de pila deterministico* es una septupla

$$\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$$

tal que

1. Q es un conjunto finito, el conjunto de estados de $\mathcal{M}$.
2. Σ es un conjunto finito, el alfabeto de entrada del automata $\mathcal{M}$.

3. Γ es un conjunto finito, el *alfabeto de la pila.*
4. $q_0 \in Q$ es el estado inicial de $\mathcal{M}$.
5. $F \subseteq Q$ es el conjunto de estados de aceptacion.
6. $\#$ es un elemento distinguido de Γ que se usa para indicar el inicio de la pila, es decir: si el automata $\mathcal{M}$ se encuentra leyendo el caracter $\#$, el automata sabe que la pila se encuentra vacia.
7. La funcion de transicion δ es una funcion de $\Sigma \times (\Gamma \times Q)$ en $\Gamma \times \{e, d\} \times (Q \cup \{\epsilon\})$, los simbolos w y d son simbolos especiales que usamos para indicar lo siguiente
   - Dados $a \in \Sigma$, $b \in \Gamma$ y $q \in Q$, si $\delta(a, b, q) = (c, e, p)$ la maquina cambia su estado interno de q a p y escribe c en la pila.
   - Dados $a \in \Sigma$, $b \in \Gamma$ y $q \in Q$, si $\delta(a, b, q) = (c, d, p)$ la maquina cambia su estado interno de q a p y si $b \neq \#$ la maquina borra este caracter de la pila, el cual es precisamente el ultimo caracter guardado en este registro. Si $b = \#$ la maquina para y rechaza el input.

Un automata de pila deterministico es como un automata regular al que se le ha adicionado un dispositivo externo de memoria: la pila. En cada transicion el automata de pila deterministico puede: o agregar un caracter a la derecha de la palabra contenida en la pila, o borrar el ultimo caracter de esta palabra. Lo que el automata escriba en la pila puede ser usado en la computacion, aunque en cada instante de la computacion el automata solo tenga acceso al ultimo caracter del contenido de la pila. La nocion de configuracion para automatas de pila deterministicos es un poco mas compleja que la nocion de configuracion para automatas regulares. Recuerde que una configuracion es una descripcion completa del estado de la maquina en un instante de la computacion. En consecuencia una tal descripcion debe declarar, como minimo, lo que del estado de la maquina es desconocido para el usuario. Dado que un automata de pila deterministico no puede: escribir caracteres, borrar caracteres o marcar caracteres en la cinta de entrada; el contenido de la cinta de entrada es invariante, (identico al contenido de esta cinta en el instante cero). Esto implica que no necesitamos incluir en nuestra nocion de configuracion informacion referente al contenido de la cinta de entrada. Por otro lado, dado que en cada transicion la cabeza lectora se mueve una posicion a la derecha, en el instante t la cabeza lectora ha de estar ubicada sobre la celda t-esima de la cinta de entrada. Tenemos entonces que lo unico que no conocemos apriori, respecto al estado de la maquina, es su estado interno y el contenido de la pila.

Definition 8. *Dado $\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$ un automata de pila deterministico, una configuracion de $\mathcal{M}$ es un par $(u, q) \in \Gamma^* \times Q$.*

Dado $w = w_1 ... w_n \in \Sigma^*$, la computacion de $\mathcal{M}$, con input w, es una secuencia ordenada y finita de configuraciones $(\epsilon, q_0), (u_1, p_1) ..., (u_n, p_n)$ que satisface lo siguiente:

1. *Si $\delta\left(w_i, (u_{i-1})_{|u_{i-1}|}, p_{i-1}\right) = (a, e, p_i)$, entonces $u_i = u_{i-1}a$.*

2. *Si* $\delta\left(w_i,(u_{i-1})_{|u_{i-1}|},p_{i-1}\right)=(a,d,p_i)$, *entonces*

$$u_i=(u_{i-1})_1\ldots(u_{i-1})_{|u_{i-1}|-1}$$

3. *No existen opciones diferentes a las tres listadas anteriormente.*

Definition 9. *Dado* $\mathcal{M}=(Q,\Gamma,q_0,F,\delta)$ *un automata de pila deterministico, dado* $w=w_1...w_n\in\Sigma^*$ *y dada*

$$(\epsilon,q_0),(u_1,p_1),...,(u_n,p_n)$$

la computacion de $\mathcal{M}$*, en el input* w*, diremos que* $\mathcal{M}$ *acepta* w *si y solo si* $p_n\in F$.

El teorema 5 a continuacion indica que el poder de computo de los automatas de pila no es suficiente para reconocer el lenguaje Pal. Esto quiere decir que no es suficiente con una unidad externa de memoria de acceso restringido (solo se puede leer el ultimo caracter, solo se puede escribir al final de la pila) para reconocer palindromos.

En adelante usaremos el simbolo $\mathcal{DCFL}$ para denotar la coleccion de los lenguajes libres de contexto deterministas, esto es: $\mathcal{DCFL}$ denota la coleccion de los lenguajes que pueden ser reconocidos usando un automata de pila deterministico.

Theorem 5. $Pal\notin\mathcal{DCFL}$.

Proof. (Esbozo de la prueba [2]) Supondremos que el lenguaje Pal es libre de contexto deterministico e intentaremos derivar una contradiccion.

Sea $\mathcal{M}=(Q,q_0,\Sigma,\Gamma,F,\#,\delta)$ un automata de pila deterministico que reconoce el lenguaje Pal. Dada $w\in\{0,1\}^*$ usaremos el simbolo $\delta_{\mathcal{M}}(w)$ para denotar la configuracion a la que accede $\mathcal{M}$ tras procesar el input w (esta es una definicion con sentido, dado que $\mathcal{M}$ es un automata de pila deterministico). Note que $\delta_{\mathcal{M}}(w)$ es un elemento de $Q\times\Gamma^{\leq n}$, y que para todo $n\geq 1$, la funcion $\delta_{\mathcal{M}}:\{0,1\}^n\to Q\times\Gamma^{\leq n}$ es inyectiva (si existieran $u,v\in\{0,1\}^n$ tal que $u\neq v$ y $\delta_{\mathcal{M}}(u)=\delta_{\mathcal{M}}(v)$ entonces $\mathcal{M}$ aceptaria el input $u\overline{v}$). Dado n, usaremos el simbolo $k(n)$ para denotar la cantidad $\max_{w\in\{0,1\}^n}\{|\pi_2(\delta_{\mathcal{M}}(w))|\}$. Tenemos entonces que

$$2^n\leq\frac{|\Gamma|^{k(n)+1}-1}{|\Gamma|-1}|Q|$$

[2] El esbozo presentado explica convenientemente el porque Pal no puede ser reconocido por un automata de pila deterministico. Debe ser claro que aun es necesario formalizar algunas de las afirmaciones hechas en la prueba. Una tal formalizacion, aunque posible, resulta tan dificil que los detalles tecnicos pueden oscurecer las ideas principales, es por ello que, siguiendo a Hopcroft-Ullman-Motwani [28] hemos preferido presentar solo la estructura basica de la prueba. Por otro lado es posible obtener este resultado como corolario del teorema 7, que estudiaremos en la ultima de las secciones que dedicaremos a estudiar los automatas de pila.

Y entonces

$$2^n \leq |\Gamma|^{k(n)+1} |Q|$$

Por lo tanto

$$k(n) + 1 \geq \frac{n - \log(|Q|)}{\log(|\Gamma|)}$$

Y si suponemos n suficientemente grande podemos afirmar que

$$k(n) \geq \frac{n}{2\log(|\Gamma|)}$$

Dado N suficientemente grande, y dado $n \geq N$, existe $w_n \in \{0,1\}^n$ tal que $|\pi_2(\delta_{\mathcal{M}}(w_n))| \geq \frac{n}{2\log(|\Gamma|)}$. La intuicion es que los modos de codificacion del automata $\mathcal{M}$ no permiten describir (codificar) la palabra w_n usando menos que $\frac{n}{2\log(|\Gamma|)}$ bits.

Dado $i \leq n$ definimos $u_{n,i}$ de la siguiente manera

$$(u_{n,i})_j = \begin{cases} (w_n)_j & \text{si } j \neq i \\ \left(\left((w_n)_j + 1\right) \bmod 2\right) & \text{si } j = i \end{cases}$$

esto es: $u_{n,i}$ se define como la palabra que solo difiere de w_n en la i-esima posicion. Es claro que:

1. $\mathcal{M}$ acepta las palabras $w_n\overline{w_n}$ y $u_{n,i}\overline{u_{n,i}}$ (para todo $i \leq n$)
2. $\mathcal{M}$ rechaza las palabras $w_n\overline{u_{n,i}}$ y $u_{n,i}\overline{w_n}$ (para todo $i \leq n$).

Lo anterior implica que al procesar el input $w_n\overline{u_{n,i}}$, el automata $\mathcal{M}$ debe usar el contenido de la pila para verificar que la mitad derecha del input es igual a $u_{n,i}$ y diferente de w_n (para que pueda rechazar el input $w_n\overline{u_{n,i}}$), y esto para todo $i \leq n$. Se tiene entonces que $\mathcal{M}$ tiene que guardar informacion, en el contenido de la pila, acerca de cada uno de los caracteres de w_n. La informacion referente al primer caracter del input no necesariamente esta confinada al primer caracter de la pila, pero lo que si debe ser cierto es que existen posiciones al inicio del input tales que toda la informacion referente a ellas, en la pila, ocurre al inicio de la pila. La intuicion es la siguiente: al procesar el input $w_n\overline{u_{n,i}}u_{n,i}\overline{w_n}$ el automata $\mathcal{M}$ debe borrar casi todo el contenido de la pila antes de terminar de leer la mitad izquierda, por lo que al iniciar la lectura de la mitad derecha no tiene informacion suficiente para decidir si esta segunda mitad es el reverso de la primera.

Note que un caracter de la pila permite describir a lo mas $\log(|\Gamma|)$ posiciones del input, por lo que un segmento de la pila de longitud K permitira describir a lo sumo $K\log(|\Gamma|)$ bits (posiciones del input). Lo anterior implica que el segmento final de la pila de longitud $\frac{n}{2\log(|\Gamma|)}$ describe a lo mas $\frac{n}{2}$ posiciones, y que en consecuencia existe un $j \lneqq n$ tal que, si se quiere usar el contenido de la pila para determinar w_j es necesario borrar al menos $\frac{n}{2\log(|\Gamma|)}$ caracteres de la pila (¡de N caracteres que se borren de la pila se podran recuperar a lo mas $\log(|Q|)$!). Sea j una tal posicion y analizemos la computacion de $\mathcal{M}$ en los inputs $w_n u_j$,

$w_n u_j \overline{u_{n,j} u_{n,k}}$ (con $k \geq n - \frac{n}{2\log(|\Gamma|)}$) y $w_n u_j \overline{u_{n,j} w_n}$. Para empezar es claro que en los primeros $\frac{n}{2\log(|\Gamma|)} + 1$ casos el automata $\mathcal{M}$ rechaza el imput y solo en el ultimo caso lo acepta. Por otro lado, al procesar el input $w_n u_j$ el automata $\mathcal{M}$ borra al menos $\frac{n}{2\log(|\Gamma|)}$ caracteres de los $k(n)$ caracteres escritos en la pila durante la lectura de w_n, en particular $\mathcal{M}$ borra toda la informacion referente a los ultimos $\frac{n}{2\log(|\Gamma|)}$ caracteres de w_n. La computacion continua cuando $\mathcal{M}$ procesa los inputs $w_n u_j \overline{u_{n,j} u_{n,k}}$ y $w_n u_j \overline{u_{n,j} w_n}$, en unos casos rechaza el input y solo en el ultimo caso lo acepta, ¿como puede $\mathcal{M}$ distinguir entre estos casos? Unicamente, usando la informacion que halla guardado en la pila acerca de los ultimos $\frac{n}{2\log(|\Gamma|)}$ caracteres de w_n, pero como ya lo hemos señalado, acerca de estos caracteres y en este instante de la computacion el automata $\mathcal{M}$ no tiene informacion guardada.

Corollary 4. *(cuarta cota inferior para Pal)*
Ningun automata de pila deterministico puede reconocer el lenguaje Pal.

Aunque los automatas de pila deterministicos son incapaces de reconocer Pal, es facil probar que los automatas de pila deterministicos son estrictamente mas potentes que los automatas regulares. Note primero que todo automata regular es un automata de pila deterministico que no usa la pila. Es facil verificar que el lenguaje $Pal^1 = \{1^n 01^n : n \in \mathbb{N}\}$ puede ser reconocido por un automata de pila deterministico, mientras que por otro lado es facil verificar (usando el lema de bombeo) que el lenguaje Pal^1 no es regular.

5 Automatas de pila no deterministicos y lenguajes libres de contexto

En esta seccion estudiaremos los *automatas de pila no deterministicos* (o simplemente automatas de pila). A diferencia de lo que sucede en el caso regular, el no-determinismo incrementa estrictamente el poder de computo de los automatas de pila: el lenguaje Pal, como veremos mas adelante, puede ser reconocido por un automata de pila no deterministico.

Un automata de pila es una septupla $\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$ tal que

1. Q es un conjunto finito, el conjunto de estados de $\mathcal{M}$.
2. Σ es un conjunto finito, el alfabeto de entrada del automata $\mathcal{M}$.
3. Γ es un conjunto finito, el *alfabeto de la pila,* tal que $\Sigma \cup \{\square\} \subseteq \Gamma$.
4. $q_0 \in Q$ es el estado inicial de $\mathcal{M}$.
5. $F \subseteq Q$ es el conjunto de estados de aceptacion.
6. $\#$ es el simbolo de Γ que indica el inicio de la pila
7. La relacion de transicion δ es un subconjunto de

$$((\Sigma \cup \{\square\}) \times \Gamma \times Q) \times (\{e, d\} \times \Gamma \times Q)$$

Note que en la definicion de la relacion δ estamos permitiendo que los automatas de pila no deterministicos tengan la capacidad de realizar *transiciones en vacio*, esto es: transiciones en las cuales la cabeza de la cinta de

entrada no se desplaza a la derecha. Una transicion en vacio es una tupla $(\square, a, q, x, b, p) \in \delta$. Si tal transicion ocurre, la maquina, (sin desplazar la cabeza lectora de la cinta de entrada), cambia de estado y transforma el contenido de la pila: si x es igual a e la maquina escribe b en la pila, si $x = d$ la maquina borra el ultimo caracter de la pila.

Las nociones de configuracion, computacion y aceptacion se definen de manera similar a como se definieron en el caso deterministico.

Proviso. *En adelante usaremos el termino automatas de pila para referirnos a los automatas de pila no deterministicos.*

Definition 10. *Dado $L \subset \Sigma^*$ diremos que L es libre de contexto si y solo si existe un automata de pila que reconoce L.*

Theorem 6. *(primera cota superior para Pal)*
Pal es libre de contexto.

Proof. Considere el automata de pila $\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$ definido por:

1. $Q = \{q_e, q_b, q_r, q_a\}$
2. $\Gamma = \Sigma$.
3. $q_0 = q_e$.
4. $F = \{q_a\}$
5. δ es la multifuncion de $\Sigma \times \Gamma \times Q$ en $\{e, d\} \times \Gamma \times Q$ (este automata no realiza transiciones en vacio y por lo tanto es un automata de tiempo real) definida por
   (a) Dados $a \in \Sigma$ y $x \in \Gamma$
   $$\delta(a, x, q_e) = \{(e, a, q_e), (e, a, q_b), (e, \square, q_b)\}$$
   (b) Dados $a \in \Sigma$ y $x \in \Gamma$
   $$\delta(a, x, q_b) = \{(d, x, q_b)\}$$
   (c) Sea $\#$ el caracter que marca el inicio de la pila y supongamos que $\mathcal{M}$ ha encontrado la primera celda vacia, se tiene que
   $$\delta(\square, \#, q_b) = \{(w, \square, q_a)\}$$
   (d) Suponga que $a \in \Sigma$
   $$\delta(a, \#, q_b) = \{(e, \square, q_r)\}$$
   (e) Suponga que $x \neq \#$, en este caso se tiene que
   $$\delta(\square, x, q_b) = \{(e, \square, q_r)\}$$
   (f) Si $a = \square$ o $x = \#$ se tiene que
   $$\delta(a, x, q_e) = \{(e, \square, q_r)\}$$

Es facil verificar que el automata $\mathcal{M}$ reconoce el lenguaje Pal

Que el lenguaje Pal sea libre de contexto no debe hacernos pensar que todo lenguaje es libre de contexto, recuerde que existen lenguajes no computables y que los lenguajes libres de contexto son computables. Tampoco debemos pensar que todo lenguje computable es libre de contexto. Sea $SQUARES$ el lenguaje $\{ww : w \in \Sigma^*\}$. Es facil probar que este lenguaje, estrechamente relacionado con Pal, no es libre de contexto (probar este hecho es una facil aplicacion de lema de Ogden, lema 1, que es una version fuerte del lema de bombeo para lenguajes libres de contexto).

5.1 Gramaticas libres de contexto

Los lenguajes libres de contexto pueden ser caracterizados de dos maneras diferentes: bien como la coleccion de los lenguajes que pueden ser reconocidos por automatas de pila [3]; o bien como la coleccion de los lenguajes que pueden ser generados por *gramaticas libres de contexto [28].*

Definition 11. *Una gramatica libre de contexto es un tupla* $\mathcal{G} = (T, N, S, R)$, *donde:*

- T *es un conjunto finito de caracteres: el conjunto de los simbolos terminales de* $\mathcal{G}$.
- N *es un conjunto finito de caracteres: el conjunto de los simbolos no terminales de* $\mathcal{G}$.
- $S \in N$ *es el simbolo inicial.*
- R *es un conjunto finito, el conjunto de reglas de produccion. Un elemento de* R *es una expresion de la forma* $A \rightarrow \alpha$, *donde* $A \in N$ *y* $\alpha \in (T \cup N)^*$.

Las reglas de produccion de una gramatica libre de contexto pueden ser entendidas como las reglas de inferencia de un sistema formal. Dada $\alpha \in (T \cup N)^*$ diremos que α es un teorema de $\mathcal{G}$ (α es $\mathcal{G}$-derivable) si y solo si existe una secuencia finita $\alpha_1, ..., \alpha_N \in (T \cup N)^*$ tal que:

1. $\alpha_1 = S$.
2. $\alpha_N = \alpha$.
3. Para todo $2 \leq i \leq N$ se tiene que α_i puede ser obtenida a partir de α_{i-1} aplicando alguna de las reglas de produccion del sistema al primer simbolo no terminal (a la izquierda) que ocurre en α_{i-1}.

Considere la gramatica $\mathcal{G}_1 = (T, N, S, R)$ definida por:

1. $T = \{0, 1\}$.
2. $N = \{S, \}$.

[3] Los lenguajes que pueden ser reconocidos usando automatas de pila deterministicos constituyen una subclase de la clase de los lenguajes libre de contexto, a saber: la clase de los lenguajes libres de contexto deterministas

3. $R = \{S \rightarrow \epsilon/0/1/0S0/1S1\}$.

Una $\mathcal{G}_1$-derivacion de la palabra 1001001 es la secuencia

$$S, 1S1, 10S01, 100S001, 1001001$$

Note que: el primer elemento de la secuencia es S (el unico axioma del sistema), el ultimo elemento de la secuencia es 1001001 (el teorema), y todos los elementos de la secuencia pueden ser obtenidos aplicando una de las reglas de produccion al elemento anterior:

1. $S \rightarrow_{S \rightarrow 1S1} 1S1$.
2. $1S1 \rightarrow_{S \rightarrow 0S0} 10S01$.
3. $10S01 \rightarrow_{S \rightarrow 0S0} 100S001$.
4. $100S001 \rightarrow_{S \rightarrow 1} 1001001$.

Dada $\mathcal{G}$ una gramatica, el conjunto de $\mathcal{G}$-teoremas es el conjunto

$$\{\alpha \in T^* : \alpha \text{ es } \mathcal{G}\text{-derivable}\}$$

Usaremos el simbolo $L(\mathcal{G})$ para denotar al conjunto de $\mathcal{G}$-teoremas, y dado $\alpha \in (T \cup N)^*$ usaremos el simbolo $\mathcal{G} \rightarrow \alpha$ para indicar que α es $\mathcal{G}$-derivable.

Definition 12. *Dada $\mathcal{G} = (T, N, S, R)$ una gramatica libre de contexto y dado $w \in T^*$ el simbolo $\#_{\mathcal{G}}(w)$ denotara el numero de $\mathcal{G}$-derivaciones de la palabra w.*

Exercise 1. Pruebe que $L(\mathcal{G}_1) = Pal$.

Definition 13. *Dado L un lenguaje libre de contexto, diremos que L es no ambiguo si y solo si existe una gramatica $\mathcal{G}$ tal que $L = L(\mathcal{G})$ y para todo $w \in L$ se tiene que $\#_{\mathcal{G}}(w) = 1$.*

Es facil probar que:

1. Todo lenguaje libre de contexto determinista es no ambiguo.
2. Pal es no ambiguo ($\mathcal{G}_1$ atestigua que Pal es no ambiguo).

5.2 Pal requiere $\Omega(n)$ pushdown-nodeterminismo

En esta seccion estudiaremos un teorema de Goldstine et al [19] quienes prueban que todo automata de pila que reconozca el lenguaje Pal debe emplear el maximo posible de no determinismo. Un automata de pila que reconozca Pal debe adivinar cual es la celda (o linea) media de su input, lo que implica adivinar $\Phi(\log(n))$ bits. En principio esta es la cantidad de nodeterminismo requerida para reconocer Pal. Contradiciendo esta intuicion probaremos que todo automata de pila que reconozca Pal debe emplear una cantidad lineal de no determinismo, el problema es que un automata de pila no puede adivinar la

posicion de la celda media al comienzo de la computacion y guardar esta informacion en algun lugar: el automata de pila no cuenta con una unidad de memoria que le permita guardar esta informacion. Un automata de pila que reconozca Pal debe ser capaz de realizar movimientos no deterministicos en cada instante de la computacion, no necesariamente al procesar cualquier input pero si al procesar los inputs que pertenecen a un conjunto infinito. C. Kintala, quien junto con P. Fischer inicio el estudio del rol que puede desempeñar el no determinismo en las computaciones de tiempo real [14], conjeturo en [29] que todo automata de pila que reconozca el lenguaje Pal requiere una cantidad lineal de no determinismo, (de acuerdo a una medida que el mismo introdujo en el articulo anteriormente citado). Una cantidad lineal de no determinismo es la maxima cantidad posible de no determinismo que, de acuerdo a la medida de Kintala, un automata de pila puede emplear.

Sea $\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$ un automata de pila. La relacion δ es una relacion que supondremos incluida en el conjunto

$$((\Sigma \cup \{\square\}) \times \Gamma \times Q) \times (\{w, d\} \times \Gamma^* \times Q)$$

Un elemento de δ, digamos (a, B, q, x, C, p), es llamado una transicion, y sera una transicion no deterministica si

$$|\{(y, R, t) : (a, B, q, y, R, t)\}| \geq 2$$

Remark 3. Las transiciones en vacio son un tipo muy especial de transiciones no deterministicas. En esta seccion queremos cuantificar el minimo numero de transiciones no deterministicas que debe realizar un automata de pila que reconozca el lenguaje Pal.Es por ello que hemos escogido distinguir las transiciones en vacio de las transiciones no deterministicas *autenticas, y* esta es la razon por la cual hemos decidido definir, en esta seccion, el concepto de transicion de la manera en que lo hemos definido, note que el quinto factor de

$$((\Sigma \cup \{\square\}) \times \Gamma \times Q) \times (\{w, d\} \times \Gamma^* \times Q)$$

es Γ^* y no Γ, esto es: hemos decidido cada secuencia (posiblemente larga) de transiciones en vacio como si fuera una unica transicion.

Una configuracion de $\mathcal{M}$ sera una tripla (q, X, w), donde q representa el estado de $\mathcal{M}$ en un instante de la computacion, X el contenido de la pila en ese instante y w el segmento de input que queda por leer. Una computacion es una secuencia de configuraciones, digamos $c_1...c_n$ tal que para todo $i \leq n$ la pareja (c_i, c_{i+1}) es compatible, es decir:

Si $c_i = (q, X_1...X_n, w_1...w_m)$ y $c_{i+1} = (p, Y_1...Y_r, u_1...u_k)$ se satisface lo siguiente

- Si $w_1...w_m = u_1...u_k$ existe una transicion en vacio de $\mathcal{M}$, digamos α, tal que $\alpha = (\epsilon, X_n, q, y, R, p)$ y tal que

$$Y_1...Y_r = \begin{cases} X_1...X_nR \text{ si } y = w \\ X_1...X_{n-1} \text{ si } y = d \end{cases}$$

– Si $w_1...w_m \neq u_1...u_k$, entonces $u_1...u_k = w\,[2...m]$ y existe una transicion de $\mathcal{M}$, digamos α, tal que $\alpha = (w_1, X_n, q, y, R, p)$ y tal que

$$Y_1...Y_r = \begin{cases} X_1...X_nR \text{ si } y = w \\ X_1...X_{n-1} \text{ si } y = d \end{cases}$$

Lo anterior define una relacion de compatibilidad (derivabilidad) entre configuraciones que denotaremos con el simbolo $\vdash_{\mathcal{M}}$.

Definition 14. *(La medida de Kintala)*

1. *Dado* $\alpha = (a, B, q, x, C, p)$ *un movimiento*

$$\kappa(\alpha) = |\{(y, R, t) : (a, B, q, y, R, t)\}|$$

2. *Dada* $c = (q, X, w)$ *una configuracion*

$$\kappa(c) = |\{d : (c, d) \in \ \vdash_{\mathcal{M}}\}|$$

3. *Dada* $\pi = c_1...c_n$ *una computacion*

$$\kappa(\pi) = \prod_{i \leq n} k(c_i)$$

4. *Dada* $w \in L(\mathcal{M})$ *se define* $\kappa_{\mathcal{M}}(w)$ *como*

$$\min\{\kappa(\pi) : \pi \text{ es una computacion de } \mathcal{M} \text{ que acepta } w\}$$

5. *Dado* $n \geq 1$ *se define*

$$\kappa_{\mathcal{M}}(n) = \max\{\kappa_{\mathcal{M}}(w) : w \in \Sigma^n\}$$

6. $\gamma_{\mathcal{M}}(n) = \log(\kappa_{\mathcal{M}}(n))$.

La funcion $\gamma_{\mathcal{M}}$ es la medida de no determinismo de Kintala asociada a $\mathcal{M}$. Note que para todo $\mathcal{M}$ se satisface $\gamma_{\mathcal{M}}(n) \in O(n)$, note tambien que si $\mathcal{M}$ es deterministico entonces $\gamma_{\mathcal{M}}$ es la funcion constante 1. En lo que sigue probaremos que si $\mathcal{M}$ es un automata de pila que reconoce Pal entonces $\gamma_{\mathcal{M}}(n) \in \Omega(n)$, de este teorema obtendremos como corolario que todo automata de pila que reconozca Pal es no deterministico (*i.e.* $Pal \notin \mathcal{DCFL}$), y mas aun que todo automata de pila que reconozca Pal usa la mayor cantidad posible de no determinismo. Antes de enunciar y probar el teorema necesitamos enunciar el lema de Ogden, el cual es una version fuerte del lema de bombeo para lenguajes libres de contexto.

Lemma 1. *(Lema de Ogden)*
Dado L *un lenguaje libre de contexto existe* p *suficientemente grande (la constante de Ogden del lenguaje* L*) tal que para toda* $w \in L$ *si* $|w| \geq p$ *y si se marcan al menos* p *caracteres de* w*, existen entonces* $v_1, ..., v_5$ *que satisfacen lo siguiente:*

1. $w = v_1...v_5$.
2. *O* v_1, v_2 *y* v_3 *contienen, cada una, al menos un caracter marcado, o* v_3, v_4 *y* v_5 *contienen, cada una, al menos un caracter marcado.*
3. $v_2 v_3 v_4$ *contiene a lo mas p caracteres marcados.*
4. *Para todo* $i \geq 0$ *la palabra* $v_1 v_2^i v_3 v_4^i v_5$ *pertenece a L.*

La prueba de este lema se deja como ejercicio, el lector interesado puede consultar la referencia [28]. El lema de Ogden es una herramienta poderosa para probar que ciertos lengujes no son libres de contexto. Es un excelente ejercicio para el lector usar el lema de Ogden para probar que el lenguaje

$$SQUARES = \{ww : w \in \Sigma^*\}$$

no es libre de contexto.

Es posible establecer un lema analogo para gramaticas libres de contexto que llamaremos *Lema de Ogden para gramaticas.*

Lemma 2. *(Lema de Ogden para gramaticas)*

Dada G una gramatica libre de contexto existe $p \in \mathbb{N}$*, que llamaremos la constante de Ogden de la gramatica G, tal que si* $w \in L(G)$ *es una palabra de longitud mayor o igual que p y si escojemos al menos p posiciones de w, existe* $X \in V$ *y derivaciones*

$$S \vdash_G \pi_1 X \pi_5 \vdash_G \pi_1 \pi_2 X \pi_4 \pi_5 \vdash_G \pi_1 \pi_2 \pi_3 \pi_4 \pi_5$$

tales que:

- $w = \pi_1 \pi_2 \pi_3 \pi_4 \pi_5$.
- *O* π_1, π_2 *y* π_3*, cada uno, contienen transiciones distinguidas.*
- *O* π_3, π_4 *y* π_5*, cada uno, contienen transiciones distinguidas.*
- *Para todo* $j \geq 0$ *se tiene que* $\pi_1 \pi_2^j \pi_3 \pi_4^j \pi_5 \in L(G)$.

Estamo listos para probar el resultado principal de esta seccion.

Theorem 7. *Si* $\mathcal{M}$ *es un automata de pila que reconoce Pal se tiene que* $\gamma_{\mathcal{M}}(n) \in \Omega(n)$.

Proof. Sea $\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$ un automata de pila que reconoce Pal y sea $G = (V, T, S, R)$ la gramatica libre de contexto definida por:

1. $V = \{[qZp] : q, p \in Q \text{ y } Z \in \Gamma\} \cup \{[qZ] : q \in Q \text{ y } Z \in \Gamma\}$
   Una variable $[qZp]$ representa las transiciones de $\mathcal{M}$ que inician en el estado q terminan en el estado p y consumen el caracter Z de la pila, las variables $[qZ]$ representan las transiciones que empiezan en q y consumen el simbolo Z.
2. $T = \{\mu : \mu \text{ es una transicion de } \mathcal{M}\}$.
3. $S = [q_{\#}]$.
4. Dada $\mu = (a, B, q, x, Z_1...Z_n, p)$ una transicion el conjunto R contiene reglas

- $[qZ] \to \varepsilon$ si $q \in F$.
- $[qZ] \to \mu$ si $q_1 \in F$.
- $[qZ] \to \mu\,[q_1 Z_1 q_2]\,[q_2 Z_2 q_3] \ldots [q_i Z_i q_{i+1}]\,[q_{i+1} Z_{i+1}]$ con $i \lneqq n$ y $q_1, ..., q_{i+1} \in Q$.
- $[qZ] \to \mu\,[q_1 Z_1 q_2]\,[q_2 Z_2 q_3] \ldots [q_i Z_i q_{i+1}]\,[q_{i+1} Z_{n+1}]$ con $q_1, ..., q_{i+1} \in Q$.

Note que cada una de las variables en G generan computaciones parciales de $\mathcal{M}$ que no disminuyen la altura de la pila por debajo de su altura inicial, excepto quizas en el ultimo movimiento. Es facil probar que $L(G)$ es igual al conjunto de computaciones aceptantes de $\mathcal{M}$.

Sea p la constante de Ogden asociada a la gramatica G y sea $k = (p\,|Q|\,|\Gamma|) + 1$. Dado $m \geq 1$ el simbolo v_m denota la palabra $(001^p)^{km+2}\,b$ y el simbolo w_m denota la palabra $v_m \overline{v_m}$. Sea π_m una computacion aceptante de w_m.

Afirmacion. $\gamma_{\mathcal{M}}(\pi_m) \geq m$.

Note que, en este punto, probar la afirmacion implica probar el teorema. En lo que sigue presentaremos una prueba de la afirmacion.

Escojamos uno de los 1^p-terminos de w_m y escojamos como caracteres distinguidos de π_m las p transiciones que consumen los p unos de este 1^p-termino. El lema de Ogden implica que existe $X \in V$ y derivaciones

$$S \vdash_G \pi_1 X \pi_5 \vdash_G \pi_1 \pi_2 X \pi_4 \pi_5 \vdash_G \pi_1 \pi_2 \pi_3 \pi_4 \pi_5$$

tales que:

- $\pi_m = \pi_1 \pi_2 \pi_3 \pi_4 \pi_5$.
- O π_1, π_2 y π_3, cada uno, contienen transiciones distinguidas.
- O π_3, π_4 y π_5, cada uno, contienen transiciones distinguidas.
- Para todo $j \geq 0$ se tiene que $\pi_1 \pi_2^j \pi_3 \pi_4^j \pi_5 \in L(G)$.

De la anterior tenemos que o π_2 o π_4 tienen como input un termino de la forma 1^i contenido en el 1^p-termino escogido. Por otro lado como para todo $j \geq 0$ la *computacion* π_j es aceptante, el input de π_2 es un termino 1^i contenido en v_m y el input de π_3 es un termino identico contenido en $\overline{v_m}$. Ahora, dado que $X \vdash_{\mathcal{M}} \pi_2 \pi_3 \pi_4$ y las variables de G generan computaciones parciales que no disminuyen la altura de la pila por debajo de su altura inicial, se tiene que en la computacion parcial $\pi_2 \pi_3 \pi_4$, en la que se procesa la subpalabra de w_m comprendida entre el inicio del termino leido por π_2 y el fin del termino leido por π_4, la altura de la pila nunca es inferior a la altura en el instante inicial.

Podemos factorizar a π como $\sigma_1 \sigma_2 ... \sigma_{km+2}$ de manera tal que si $i \geq 2$ la computacion parcial σ_i inicia en el punto del i-esimo 1^p-termino en el que la altura de la pila alcanza su menor valor. Sea t_i el numero de unos del i-esimo 1^p-termino que no han sido leidos cuando la computacion parcial σ_i inicia. Suponga que existe $1 \leq j \leq m$ tal que la computacion parcial $\sigma_{k(j-1)+2} ... \sigma_{kj+1}$ no contiene transiciones deterministicas. Note que existen a lo mas $p\,|Q|\,|\Gamma|$ triplas distintas de la forma (t_i, q_i, Z_i) y, por la manera en como hemos escogido k, se tiene que

$p\,|Q|\,|\Gamma| \lneqq k$. Existen entonces numeros r y s tales que

$$k\,(j-1)+2 \leq r \lneqq s \leq kj+1$$
y
$$(t_r, q_r, Z_r) = (t_s, q_s, Z_s)$$

Sea τ igual a $\sigma_r ... \sigma_s$. Durante la computacion τ la altura de la pila nunca es menor que su valor inicial, se tiene entonces que

$$\left(q, 1^p\,(001^p)^{s-r-1}\,001^{p-t}, Z\right) \vdash_\tau (q, \varepsilon, Zy)$$

para algun $y \in \Gamma^*$, y como no ocurren transiciones no deterministicas durante la computacion parcial τ, el automata $\mathcal{M}$ esta condenado a repetir la computacion parcial τ hasta consumir el input. Esto implica que π es un prefijo de $\sigma_1 ... \sigma_{r-1}\tau^l$ (con l suficientemente grande). Como π termina en un estado de aceptacion, τ debe pasar por un estado de aceptacion. Podemos entonces factorizar π como $\alpha\beta$, donde α termina en un punto ubicado entre la lectura del primer caracter procesado por σ_r y la lectura del ultimo caracter procesado por σ_s en el que el automata accede a un estado de aceptacion. Podemos usar el lema de Ogden nuevamente para probar que existen factorizaciones $\alpha = \alpha_1\alpha_2\alpha_3$ y $\beta = \beta_1\beta_2\beta_3$ tales que:

- α_2 procesa un termino de la forma 1^i (con $i \gneqq 0$) contenido en el primer 1^p-termino de v_m.
- β_2 procesa un termino de la forma 1^i contenido en el primer 1^p-termino de v_m.
- $\alpha_1\alpha_2^2\alpha_3\beta_1\beta_2^2\beta_3$ es una computacion aceptante de $\mathcal{M}$.

Ahora, como $\alpha_1\alpha_2\alpha_3$ es en si misma una computacion aceptante de $\mathcal{M}$ se tiene que $\alpha_1\alpha_2^2\alpha_3$ es una computacion aceptante de $\mathcal{M}$, pero esto es imposible dado que el primer 1^*-termino del input de $\alpha_1\alpha_2^2\alpha_3$ es 1^{p+i} mientras que todos los otros 1^*-terminos contenidos en el input (existen al menos dos de estos terminos) de $\alpha_1\alpha_2^2\alpha_3$ tienen la forma 1^p y $p \lneqq p+i$, esto es: el input de $\alpha_1\alpha_2^2\alpha_3$ no es un palindromo.

5.3 Automatas de pila de doble via.

Un automata de pila de doble via es un automata de pila en el cual la cabeza lectora de la cinta de entrada puede moverse en la dos direcciones o permanecer estatica. Ya hemos visto que en el caso de los automatas regulares la bidireccionalidad de la cabeza lectora no aporta poder de computo, es posible que esto mismo ocurra en el caso de los automatas de pila ¿aporta poder de computo a los automatas de pila la bidireccionalidad de la cabeza lectora de la cinta de entrada? El lector debe recordar que en el caso de los automatas regulares el nodeterminismo no aporta poder de computo, pero que en el caso de los automatas de pila el nodeterminismo si aporta poder de computo ¿que ocurre con la bidireccionalidad de la cabeza lectora?

Un *automata de pila deterministico y de doble via* es una septupla

$$\mathcal{M} = (Q, q_0, \Sigma, \Gamma, F, \#, \delta)$$

tal que

1. Q es un conjunto finito, el conjunto de estados de $\mathcal{M}$.
2. Σ es un conjunto finito, el alfabeto de entrada del automata $\mathcal{M}$.
3. Γ es un conjunto finito, el *alfabeto de la pila.*
4. $q_0 \in Q$ es el estado inicial de $\mathcal{M}$.
5. $F \subseteq Q$ es el conjunto de estados de aceptacion.
6. $\#$ es un elemento distinguido de Γ que se usa para indicar el inicio de la pila, es decir: si el automata $\mathcal{M}$ se encuentra leyendo el caracter $\#$, el automata sabe que la pila se encuentra vacia.
7. La funcion de transicion δ es una funcion de $\Sigma \times (\Gamma \times Q)$ en $\Gamma \times \{w, d, n\} \times (Q \cup \{\epsilon\}) \times \{\leftarrow, \Diamond, \rightarrow\}$, los simbolos e, d y n son simbolos especiales que usamos para indicar lo siguiente
   - Dados $a \in \Sigma$, $b \in \Gamma$ y $q \in Q$, si $\delta(a, b, q) = (c, e, p, r)$ la maquina cambia su estado interno de q a p, escribe c en la pila y desplaza su cabeza lectora de acuerdo a $r \in \{\leftarrow, \Diamond, \rightarrow\}$
   - Sean $a \in \Sigma$, $b \in \Gamma$ y $q \in Q$ y suponga que $\delta(a, b, q) = (c, d, p)$. Si $b \neq \#$ la maquina borra el ultimo caracter del contenido de la pila. Por otro lado si $b = \#$ la maquina para y rechaza el input.
   - Dados $a \in \Sigma$, $b \in \Gamma$ y $q \in Q$, si $\delta(a, b, q) = (c, n, p)$ la maquina cambia su estado interno de q a p y deja invariante el contenido de la pila.

Sea 2-$\mathcal{DCFL}$ el conjunto de todos los lenguajes que pueden ser reconocidos por un automata de pila deterministico y de doble via. Note que el tiempo de computo de un automata de pila de doble via no tiene porque estar acotado superiormente. Lo anterior nos permite introducir una segunda definicion.

Definition 15. 2-$\mathcal{LDCFL}$ *es el conjunto de todos los lenguajes que pueden ser reconocidos en tiempo lineal por un automata de pila deterministico y de doble via.*

Lemma 3. $Pal \in$ 2-$\mathcal{LDCFL}$.

Proof. Es facil diseñar un automata de pila deterministico y de doble via que reconozca Pal en tiempo lineal. A continuacion presentaremos una descripcion esquematica de un tal automata, al que denotaremos con el simbolo $\mathcal{M}$.

Sea w un input de $\mathcal{M}$, la computacion de $\mathcal{M}$ en el input w esta dividida en tres fases.

- En la fase 1 el automata $\mathcal{M}$ lee el input y lo copia en la pila.
- En la fase 2 el automata $\mathcal{M}$ desplaza la cabeza lectora de la cinta de entrada al extremo izquierdo de la misma.

– En la fase 3 el automata $\mathcal{M}$ mueve de manera simultanea la cabeza de la cinta de entrada (de izquierda a derecha) y la cabeza de la pila (desde el techo de la pila hasta la base de la misma). Durante esta etapa el automata compara los caracteres leidos en la cinta y en la pila.

Es claro que un tal automata es capaz de reconocer el lenguaje Pal y tambien debe ser claro que el tiempo de computo empleado por $\mathcal{M}$, al procesar el input w, esta acotado por $3\,|w| + 3$.

Corollary 5. $\mathcal{DCFL} \subset 2\text{-}\mathcal{LDCFL} \subseteq 2\text{-}\mathcal{DCFL}$.

El corolario anterior muestra que la bidireccionalidad de la cabeza lectora si aporta poder de computo: un lenguaje que no puede ser reconocido usando automatas de pila deterministicos, puede serlo usando automatas de pila deterministicos y de doble via. En este punto es natural preguntarse si la clase $2\text{-}\mathcal{LDCFL}$ esta contenida en la clase $\mathcal{CFL}$. Sorprendentemente la respuesta es no.

Lemma 4. $SQUARES \in 2\text{-}\mathcal{LDCFL}$.

Proof. Es facil diseñar un automata de pila deterministico y de doble via que reconozca $SQUARES$ en tiempo lineal. A continuacion presentaremos una descripcion esquematica de un tal automata, al que denotaremos con el simbolo $\mathcal{M}$.

Sea w un input de $\mathcal{M}$, la computacion de $\mathcal{M}$ en el input w esta dividida en 4 fases.

– En la fase 1 el automata $\mathcal{M}$ determina si $|w|$ es un numero par, si $|w|$ es par el automata pasa a la fase 2, en caso contrario el automata rechaza el input. Para calcular la paridad de $|w|$ el automata simplemente lee a w alternando entre dos estados q_0 y q_1, si al terminar la lectura de w el automata se encuentra en el estado q_0 se tiene que $|w|$ es par, en caso contrario $|w|$ es impar. Durante esta fase el automata copia el input en la pila.
– En la fase 2 desplazamos las cabezas, la de la cinta de lectura de derecha a izquierda, la de la pila desde el techo hacia la base de la pila, la cabeza de la cinta se mueva dos veces mas rapido que la cabeza de la pila. Si w es igual a $w_1...w_{2n}$, al terminar esta fase el contenido de la pila es $w_1...w_n$.
– En la fase 3 la cabeza de la cinta de entrada se desplaza hasta la celda que contiene el ultimo caracter de w.
– En la fase 4 las dos cabezas de desplazan de manera simultanea comparando los caracteres leidos.

Es claro que un tal automata es capaz de reconocer el lenguaje $SQUARES$ y tambien debe ser claro que el tiempo de computo empleado por $\mathcal{M}$, al procesar el input w, esta acotado por $4\,|w| + 4$.

Tenemos entonces que la bidireccionalidad de la cabeza le aporta tanto poder de computo a los automatas de pila, que este poder puede llegar a superar el

poder de computo aportado por el nodeterminismo. La afirmacion anterior seria completamente cierta si pudieramos probar la contenencia $\mathcal{CFL} \subseteq 2\text{-}\mathcal{LDCFL}$. Hasta donde sabemos este sigue siendo, despues de 37 años [38], un problema abierto en teoria de automatas. Un caso particular del *problema abierto* antes mencionado es el siguiente problema (presumiblemente abierto) relacionado con palindromos.

Problema. Pruebe o refute lo siguiente

$$Pal^* \in 2\text{-}\mathcal{LDCFL}$$

Recuerde que Pal^* es el lenguaje libre de contexto definido por

$$\{w : \exists n \exists w_1 ... \exists w_n \, (w = w_1 ... w_n \ \& \ w_1, ..., w_n \in Pal)\}$$

Cerraremos esta breve seccion comentado dos hechos importantes relacionados con los automatas de pila deterministicos y de doble via.

1. Los automatas de pila deterministicos y de doble via son incapaces de reconocer el lenguaje Pal en tiempo real. Note que todo automata de pila deterministico de doble via y de tiempo real es un automata de pila deterministico, esto es: si $2 - \mathcal{RDCFL}$ es la coleccion de todos los lenguajes que pueden ser reconocidos en tiempo real por un automata de pila deterministico y de doble via, se tiene que

$$2 - \mathcal{RDCFL} = \mathcal{DCFL}$$

2. Un importante resultado de Cook [8] afirma que todo automata de pila deterministico y de doble via puede ser simulado en tiempo lineal por una maquina de Turing, esto es: si $\mathcal{LIN}$ es la coleccion de todos los problemas que pueden ser resueltos en tiempo lineal, se tiene que

$$2 - \mathcal{DCFL} \subseteq \mathcal{LIN}$$

6 Maquinas de Turing

Para empezar introduciremos el modelo de maquinas de Turing de una sola cinta. Podemos pensar en una maquina de Turing de una sola cinta (para abreviar, a este tipo de maquinas las llamaremos simplemente maquinas deTuring) como en un automata regular dotado con una cabeza que puede moverse en las dos direcciones y que ademas puede escribir y borrar. Dado que una maquina de Turing puede escribir en la cinta de entrada, una tal maquina puede usar su cinta de entrada como un dispositivo externo de memoria. Las maquinas de Turing son maquinas que combinan las habilidades de los automatas de pila y de los automatas de doble via, es de esperar que esta combinacion de habilidades de como resultado un tipo de automata mas poderoso.

Definition 16. *Una maquina de Turing es una sextupla*

$$\mathcal{M} = (Q, \Gamma, q_0, F, \delta, A)$$

tal que:

1. *Q es un conjunto finito, el conjunto de estados de la maquina.*
2. *Γ es un conjunto finito, el alfabeto de la maquina que satisface la ecuacion*

$$\Sigma \cup \{\square\} \subseteq \Gamma$$

3. *$q_0 \in Q$ es el estado inicial de la maquina.*
4. *$F \subset Q$ es el conjunto de estados finales de la maquina.*
5. *La funcion de transicion δ es una funcion de $\Gamma \times Q$ en $\Gamma \times Q \times \{\leftarrow, \Diamond, \rightarrow\}$.*
6. *$A \subset F \subset Q$ y si el estado interno de la maquina pertenece a A la maquina para y acepta el input.*

Dados $a, b \in \Gamma$; $q, p \in Q$ y $h \in \{\leftarrow, \Diamond, \rightarrow\}$, si $\delta(a, q) = (b, p, h)$, cada vez que la maquina se encuentre en el estado q leyendo el caracter a, la maquina borrara a, escribira a cambio el caracter b (reemplazara a por b), pasara del estado q al estado p y se movera a la derecha, a la izquierda o permanecera en su lugar dependiendo de quien sea h ($\rightarrow$ indica movimiento a la derecha, $\leftarrow$ indica movimiento a la izquierda y $\Diamond$ indica que la cabeza de la maquina permanece sobre la misma celda, i.e. no se mueve).

Example 1. Recuerde que $SQUARES$ no es libre de contexto. Por otro lado es muy facil diseñar una maquina de Turing que reconozca el lenguaje $SQUARES$.

Del ejemplo anterior tenemos que existen lenguajes que no son libres de contexto pero que si pueden ser reconocidos usando maquinas de Turing.

Definition 17. *Diremos que $L \subseteq \Sigma^*$ es Turing-computable si y solo si existe una maquina de Turing que reconoce L.*

A continuacion definiremos la nocion de configuracion para maquinas de Turing, la nocion de configuracion nos permitira, a su vez, definir la nocion de computacion para este tipo de maquinas.

Definition 18. *Una configuracion de $\mathcal{M} = (Q, \Gamma, q_0, F, \delta, A)$ es una tripla (w, q, i) tal que*

1. *$w \in \Gamma^*$ (es el contenido de la cinta)*
2. *$q \in Q$ (es el estado interno de la maquina).*
3. *$i \leq |w|$ (es la posicion de la cabeza lectora).*

Una maquina de Turing no esta obligada a parar al procesar cada uno de sus posibles inputs. Sea $\mathcal{M} = (Q, \Gamma, q_0, F, \delta, A)$ una maquina de Turing y sea w un input de $\mathcal{M}$. Supondremos que $\mathcal{M}$ para al procesar el input w.

Notacion. *Dada w una palabra y dado $i \leq |w|$ el simbolo $(w)_i$ denotara el i-esimo caracter de w.*

Definition 19. *La computacion de $\mathcal{M}$, en el input w, es una secuencia*

$$(w_1, q_1, i_1) ... (w_m, q_m, i_m)$$

de configuraciones tal que:

1. $w_1 = w$, $q_1 = q_0$, $i_1 = 1$ *y* $q_m \in F$.
2. *Para todo* $j \leq m$ *se tiene que* $|i_j - i_{j-1}| \leq 1$.
3. *Para todo* $j \leq m$ *se tiene que*

$$\delta\left(\left(w_{j-1}\right)_{i_{j-1}}, q_{j-1}, i_{j-1}\right) = \left(\left(w_j\right)_{i_{j-1}}, q_j, x_j\right)$$

donde $x \in \{\leftarrow, \Diamond, \rightarrow\}$ *y se tiene que*

$$i_j = \begin{cases} i_{j-1} - 1 \text{ si } x = \leftarrow \\ i_{j-1} \text{ si } x = \Diamond \\ i_{j-1} + 1 \text{ en otro caso} \end{cases}$$

4. *Para todo* $j \leq m$ *se tiene que si* $k \neq i_{j-1}$ *entonces* $\left(w_{j-1}\right)_k = \left(w_j\right)_k$.

Dado w un input de $\mathcal{M}$ y dada $(w_1, q_1, i_1) ... (w_m, q_m, i_m)$ la computacion de $\mathcal{M}$ en el simbolo w, diremos que $\mathcal{M}$ *acepta* w si y solo si $q_m \in A$. Usaremos el simbolo $L(\mathcal{M})$ para denotar el lenguaje aceptado por $\mathcal{M}$, esto es:

$$L(\mathcal{M}) = \{w : \mathcal{M} \text{ acepta } w\}$$

Definition 20. *Dada $\mathcal{M}$, dado w, un input de $\mathcal{M}$, y dada*

$$(w_1, q_1, i_1) ... (w_m, q_m, i_m)$$

la computacion de $\mathcal{M}$ en el input w, el tiempo de computo de $\mathcal{M}$ en w es igual a m. Usaremos el simbolo $t_{\mathcal{M}}(w)$ para denotar esta cantidad. Dado $n \geq 1$ definimos el tiempo de computo de $\mathcal{M}$ en los inputs de tamaño n como

$$t_{\mathcal{M}}(n) = \max_{w:|w|=n} \{t_{\mathcal{M}}(w)\}$$

6.1 Complejidad computacional y reconocimiento de palindromos

La complejidad computacional (o la teoria de la complejidad computacional) intenta dar respuestas al siguiente tipo de pregunta: ¿dado L un lenguaje computable, cuales son los recursos computacionales requeridos para resolver el problema L? La complejidad computacional parte del hecho de que todo problema tiene un grado de dificultad intrinseco y que por lo tanto toda solucion computacional de un problema algoritmico requiere tener acceso a una cantidad minima de recursos computacionales. El concepto de recurso computacional es un concepto informal y multifacetico. Recursos computacionales pueden ser: tiempo de computo, espacio de memoria, habilidades del hardware, recursos aleatorios (bits aleatorios) etc. A lo largo de este primer capitulo del libro hemos estado analizando, en cierto sentido, la complejidad computacional (esto es: la dificultad intrinseca) del problema Pal. Hemos establecido cotas inferiores tales como:

1. No es posible resolver Pal usando automatas regulares (una cabeza lectora unidireccional).
2. No es posible resolver Pal usando automatas de pila deterministicos (cabeza lectora unidireccional + dispositivo externo tipo pila).
3. No es posible resolver Pal usando automatas de doble via (una cabeza bidireccional).

Reciprocamente, hemos establecido cotas superiores tales como:
Es posible resolver Pal usando un automata de pila no deterministico (cabeza lectora unidireccional + dispositivo externo tipo pila + no-determinismo).

Este tipo de resultados, cotas inferiores y cotas superiores, es el tipo de resultados que la complejidad computacional intenta establecer cada vez que se analiza un problema computable. Una cota superior es simplemente un algoritmo que resuelve el problema. Dado L un problema y dado $\mathcal{M}$ un algoritmo, si $\mathcal{M}$ resuelve L entonces los recursos usados por $\mathcal{M}$ son **suficientes** para resolver L. Establecer una cota superior consiste precisamente en probar que cierta cantidad de recursos (usados inteligentemente), son suficientes para resolver L. Por otro lado establecer cotas inferiores (tal cantidad de recursos es insuficiente, por lo tanto algo mas es **necesario**), presupone un analisis teorico tanto del problema como de las potencialidades de los recursos en cuestion. Algunas cotas inferiores puden obtenerse por dos caminos diferentes

- Como corolarios de resultados generales concernientes al modelo de computacion (o conjunto de modelos con acceso a cierta cantidad de recursos) tales como: el lema de bombeo para automatas regulares, la simulabilidad mediante automatas regulares de los automatas regulares no deterministicos y de los automatas de doble via.
- Via argumentos combinatorios ad hoc, tales como el argumento de secuencias de cruce que usaremos en este capitulo para probar que Pal requiere tiempo cuadratico cuando se emplean maquinas de Turing de una cinta (teorema 8).

6.2 Una cota inferior para Pal: el teorema de Hennie

En esta seccion probaremos una cota inferior para Pal. Probaremos que toda maquina de Turing de una sola cinta capaz de reconocer el lenguaje Pal tiene un tiempo de computo por lo menos cuadratico. La prueba es un argumento combinatorio elemental que se basa en la nocion de *secuencia de cruce,* nocion que ha sido utilizada con exito para obtener otras cotas inferiores (vea [6], [55]).

Theorem 8. *(Teorema de Hennie, quinta cota inferior para Pal)*
Pal requiere tiempo $\Omega\left(n^2\right)$ *sobre maquinas de Turing de una sola cinta*

Proof. Sea $\mathcal{M}$ una maquina de Turing de una sola cinta que reconoce el lenguaje Pal. Considere el subconjunto Φ de Pal definido por

$$\left\{w1^{2|x|}\overline{w} : x \in \Sigma^*\right\}$$

Dado $w1^{2|x|}\overline{w} \in \Sigma^*$ y dado $i \leq 4|w|$, definimos $C_i^{\mathcal{M}}(w)$, la i-esima secuencia de cruce *(crossing sequence)* asociada a w y a $\mathcal{M}$, de la siguiente manera: $C_i^{\mathcal{M}}(w)$ es la secuencia ordenada de estados a los que accede $\mathcal{M}$ cada vez que, procesando el input $w1^{2|x|}\overline{w}$, la cabeza lectora accede a la celda i-esima Dado $w \in \Sigma^*$, definimos $C^{\mathcal{M}}(w)$ como

$$\left\{C_i^{\mathcal{M}}(w) : |w| \leq i \leq 3|w|\right\}$$

La observacion fundamental es que dados $w, u \in \Sigma^n$, si $w \neq u$ entonces $C^{\mathcal{M}}(w) \cap C^{\mathcal{M}}(u) = \varnothing$. Suponga que existen $i, j \in \{n, n+1, ..., 3n\}$ tal que $C_i^{\mathcal{M}}(w) = C_j^{\mathcal{M}}(u)$, sea z el prefijo de longitud i de la palabra $w1^{2|x|}\overline{w}$ y sea v el sufijo de longitud $4n - j$ de la palabra $u1^{2|x|}\overline{u}$. Es facil convencerse de los siguientes hechos

1. $zv \notin Pal$
2. $\mathcal{M}$ acepta la palabra zw

Lo anterior implica que $\mathcal{M}$ no puede reconocer el lenguaje Pal, contradiciendo con ello la hipotesis inicial.

Dado $w \in \Sigma^*$ el simbolo t_w denotara la cantidad $\min_{i\in\{|w|,...,3|w|\}}\left\{\left|C_i^{\mathcal{M}}(w)\right|\right\}$ y dado $n \in \mathbb{N}$ el simbolo t_n denotara la cantidad $\max_{w\in\Sigma^n}\{t_w\}$.

Afirmacion. $t_n \in \Omega(n)$.

Note primero que la afirmacion implica que el tiempo de computo de $\mathcal{M}$ es $\Omega\left(n^2\right)$. Para terminar la prueba es suficiente entonces probar la afirmacion.

Note que el numero de $\mathcal{M}$-secuencias de cruce de longitud menor o igual que t esta acotado superiormente por

$$\sum_{i=0}^{t} |Q_{\mathcal{M}}|^i = \frac{|Q_{\mathcal{M}}|^{t+1} - 1}{|Q_{\mathcal{M}}| - 1}$$

Recuerde que dado $n \in \mathbb{N}$ y dados $w, u \in \Sigma^n$ se tiene que $C^{\mathcal{M}}(w) \cap C^{\mathcal{M}}(u) = \varnothing$. Se tiene entonces que

$$\frac{|Q_{\mathcal{M}}|^{t_n+1} - 1}{|Q_{\mathcal{M}}| - 1} \geq 2^n$$

Esta ultima desigualdad implica que $t_n \in \Omega(n)$, dado que $Q_{\mathcal{M}}$ es una cantidad constante que no depende de n.

Es muy facil verificar que existe una maquina de Turing de una sola cinta capaz de reconocer el lenguaje Pal en tiempo $O\left(n^2\right)$.

Lemma 5. *(segunda cota superior para Pal)*

Existe una maquina de Turing de una sola cinta que reconoce Pal en tiempo $O\left(n^2\right)$.

Remark 4. El teorema de Hennnie implica que toda maquina de Turing de una sola cinta que reconozca Pal debe realizar, al procesar un input de longitud n, una cantidad $\Omega(n)$ de zig zags de amplitud $\Omega(n)$, esto es: toda maquina de Turing de una sola cinta es esencialmente la maquina ingenua que recorre la cinta $\frac{n}{2}$ veces, verificando en la i-esima travesia que el caracter i-esimo y el $(n-i)$-esimo coinciden. Este resultado no es del todo sorprendente, toda maquina de Turing de una sola cinta debe realizar un numero no acotado de zig zags de gran amplitud (lo zig zags de amplitud acotada son inocuos, pueden ser eliminados agregando un numero suficientemente grande de estados) para poder reconocer un lenguaje no regular. Hennie probo en [27] que si $\mathcal{M}$ es una maquina de una sola cinta cuyo tiempo de computo $T_{\mathcal{M}}$ satisface $T_{\mathcal{M}}(n) \in O(n)$, entonces $L(\mathcal{M})$, el lenguaje reconocido por $\mathcal{M}$, es regular. Trachtenbrot extendio en [50] este resultado de Hennie probando que si $\mathcal{M}$ es una maquina de Turing de una sola cinta que satisface $T_{\mathcal{M}}(n) \in o(n\log(n))$ (i.e. $\lim_{n\to\infty}\left(\frac{T_{\mathcal{M}}(n)}{n\log(n)}\right) = 0$), entonces $L(\mathcal{M})$ es un lenguaje regular: el reconocimiento de lenguajes no regulares implica realizar una cantidad al menos logaritmica de zig zags de amplitud no acotada.

Tenemos entonces una cota inferior y una cota superior referentes al tiempo de computo requerido por una maquina de una sola cinta para reconocer el lenguaje Pal, y tenemos adicionalmente un hecho infrecuente en ciencias de la computacion: la cota inferior y la superior coinciden, o lo que es lo mismo las cotas son *ajustadas* (*tight*).

7 Maquinas multidimensionales

En esta seccion estudiaremos un modelo de computacion que extiende el modelo de maquina de Turing.

Definition 21. *Una maquina de Turing bidimensional (para mayor brevedad, simplemente maquina bidimensional) es una sextupla* $(Q, \Gamma, q_0, F, \delta, A)$ *tal que:*

1. Q *es un conjunto finito, el conjunto de estados de la maquina.*
2. Γ *es un conjunto finito, el alfabeto de la maquina que satisface* $\Sigma \cup \{\square\} \subseteq \Gamma$.
3. $q_0 \in Q$ *es el estado inicial de la maquina.*
4. $F \subset Q$ *es el conjunto de estados finales de la maquina.*
5. *La funcion de transicion* δ *es una funcion de* $\Gamma \times Q$ *en* $\Gamma \times Q \times \{\leftarrow, \Diamond, \rightarrow, \uparrow, \downarrow\}$.
6. $F \subset A \subset Q$ *y si el estado interno de la maquina pertenece a* A *la maquina para.*

Las maquinas bidimensionales son maquinas de Turing que cuentan con una unica cinta bidimensional, la cual puede ser vista como un semiplano (el semiplano nororiental) particionado en celdas. Adicionalmente la maquina cuenta con una cabeza lectora capaz de moverse a lo *largo y ancho* de la cinta. Es por ello que en la definicion de δ, el tercer factor del rango de δ, que es el conjunto $\{\leftarrow, \Diamond, \rightarrow, \uparrow, \downarrow\}$, incluye cinco parametros: movimientos a derecha, a izquierda, hacia arriba, hacia abajo y permanecer sobre la celda que se acaba de leer.

Dada $\mathcal{M}$ una maquina bidimensional y dado x un input de M supondremos que al iniciar el computo el input aparece escrito en la primera fila y con el primer caracter ocupando el extremo izquierdo de la primera fila. Podemos pensar en la cinta de $\mathcal{M}$ como en el reticulo $\{(i,j) : i,j \in \mathbb{N}\}$. La primera fila de la cinta es el conjunto $\{(0,j) : j \in \mathbb{N}\}$. El *origen* de la cinta es la celda $(0,0)$, ubicada en el extremo suroccidental de la misma.

Es posible que una cinta bidimensional agregue algun poder de computo a la maquina, es posible que no. Si la tesis de Church es cierta, una cinta bidimensional no nos va a permitir decidir un lenguaje que no sea Turing-computable, pero lo que si puede pasar es que la bidimensionalidad de la cinta nos permita decidir algunos lenguajes computables de manera mas eficiente. Esta ultima hipotesis podriamos intentar revisarla a la luz de lo que sucede con el lenguje Pal. Antes de esto enunciaremos un teorema que afirma que toda maquina bidimensional puede ser efectivamente simulada por una maquina unidimensional, y adicionalmente que el tiempo de computo de la maquina unidimensional esta acotado por un polinomio en el tiempo de computo de la maquina bidimensional.

Theorem 9. *Suponga que $\mathcal{M}$ es una maquina bidimensional que decide L. Existe una maquina unidimensional $\mathcal{M}_1$ que decide el lenguaje L y tal que para todo $n \in \mathbb{N}$ se satisface la desigualdad*

$$T_{\mathcal{M}_1}(n) \leq 2\left(\left(T_{\mathcal{M}}(n)^2 + n + 4\right)\right)^2$$

El lector interesado en la prueba del teorema puede consultar la referencia [39].

7.1 Una cota inferior

En esta seccion probaremos que toda maquina de Turing bidimensional que acepte Pal tiene un tiempo de computo $\Omega\left(\frac{n^2}{\log(n)}\right)$. El argumento es una adaptacion del argumento de secuencias de cruce usado en el caso unidimensional. Dada $\mathcal{M}$ una maquina bidimensional asumimos que el input esta ubicado en la primera fila, escrito de izquierda a derecha y con su primer caracter ubicado en el origen. Lo primero que debemos hacer para adaptar el argumento es adaptar la nocion de secuencia de cruce al caso bidimensional. Suponga que $\mathcal{M}$ es una maquina de Turing bidimensional que acepta Pal, suponga que x es un input de $\mathcal{M}$ de longitud n y suponga que $i \leq n$, la secuencia de cruce $C_i^{\mathcal{M}}(w)$ es una secuencia de parejas ordenadas, donde cada pareja esta constituida por un estado de $\mathcal{M}$ y por un numero natural. Cada pareja en la secuencia $C_i^{\mathcal{M}}(w)$ contiene informacion acerca del estado en que estaba la maquina al cruzar la linea que separa las columnas i-esima e $i+1$-esima, ademas de la columna usada al realizar este cruce (la altura a la cual se realizo este cruce). Adicionalmente pedimos que exista una correspondencia entre la secuencia $C_i^{\mathcal{M}}(w)$ y la secuencia ordenada de ocasiones en que la cabeza lectora cruzo tal linea.

En lo que sigue fijaremos una maquina bidimensional que acepta Pal. Los dos lemas a continuacion son la version bidimensional de los dos hechos claves usados en la prueba del teorema de Hennie (8).

Lemma 6. *Suponga que $\mathcal{M}$ acepta uv y xy, suponga ademas que $|x| = i$, $|u| = j$ y que $C_i^{\mathcal{M}}(xy) = C_j^{\mathcal{M}}(uv)$ Tenemos entonces que $\mathcal{M}$ acepta la palabra xv.*

Dada $w \in \Sigma^*$ el simbolo w^* denotara el palindromo $w0^{|w|}\overline{w}$. Dado $m \geq 1$, el simbolo L_m denotara el conjunto $\{w^* : |w| = m\}$.

Lemma 7. *Dadas $w \neq u$ dos palabras de longitud m, y dados $i, j \in \{m, ..., 2m\}$ se tiene que $C_i^{\mathcal{M}}(w^*) \neq C_j^{\mathcal{M}}(u^*)$.*

De los lemas anteriores es facil obtener la prueba del siguiente teorema

Theorem 10. *(octava cota inferior para Pal)*
Si $\mathcal{M}$ es una maquina bidimensional que acepta Pal, el tiempo de computo de $\mathcal{M}$ es $\Omega\left(\frac{n^2}{\log(n)}\right)$.

Proof. El numero de posibles secuencias de cruce de longitud l y que corresponden a computaciones de $\mathcal{M}$ de duracion acotada por n^2 esta acotado superiormente por $\left(|Q|\, n^2\right)^l$, donde $|Q|$ es el numero de estados de la maquina $\mathcal{M}$. Dada $w \in \Sigma^m$ definimos

$$t_w = \min_{m+1 \leq i \leq 2m} \left(\left|C_i^{\mathcal{M}}(w^*)\right|\right)$$

donde el simbolo $\left|C_i^{\mathcal{M}}(w^*)\right|$ denota la longitud de la secuencia $C_i^{\mathcal{M}}(w^*)$. Podemos definir una funcion ψ de L_m en el conjunto de las posibles secuencias de cruce de la siguiente manera:

Dada $w \in \Sigma^m$ escogemos i_w tal que $t_w = \left|C_{i_w}^{\mathcal{M}}(w^*)\right|$ y definimos $\psi(w)$ como $C_{i_x}^{\mathcal{M}}(w^*)$.

Note que la funcion ψ es inyectiva. Por lo tanto el conjunto L_m (que tiene exactamente 2^m elementos) determina 2^m secuencias de cruce distintas dos a dos, (las secuencias $(\psi(w))_{x \in \Sigma^m}$).

Suponga que l es una cota en la longitud de las secuencias $(\psi(x))_{x \in \Sigma^m}$. Tenemos que $\left(|Q|\, n^2\right)^l \geq 2^m$; y esto implica que

$$l \geq \log_{|Q|n^2} 2^m = \frac{m}{2\log(n) + \log(|Q|)}$$

De ello se tiene que existe $w_m \in L_m$ tal que w_m determina m secuencias de cruce de longitud $\Omega\left(\frac{m}{\log(m)}\right)$ y esto claramente implica que el tiempo de computo de $\mathcal{M}$ es $\Omega\left(\frac{n^2}{\log(n)}\right)$, dado que el tiempo de computo de $\mathcal{M}$ en el input w_m es mayor o igual que la suma de las longitudes de las secuencias de cruce determinadas por w_m, y esta cantidad es al menos $\frac{m^2}{\log(m)}$ (recuerde que $|w_m| = 3m$)

Remark 5. El argumento usado en la prueba anterior puede ser facilmente adaptado al caso k-dimensional (con $k \geq 3$) para obtener la siguiente cota inferior: si $\mathcal{M}$ es una maquina de Turing k-dimensional que reconoce el lenguaje Pal, se tiene que $t_{\mathcal{M}}(n) \in \Omega\left(\frac{n^2}{\log^{k-1}(n)}\right)$.

7.2 Una cota superior

En esta seccion probaremos que existe una maquina de Turing bidimensional que reconoce Pal en tiempo $O\left(\frac{n^2}{\log(n)}\right)$. Note que esta cota superior, junto con la cota inferior de la seccion anterior, implica que una maquina de Turing bidimensional que resuelva Pal de manera optima (optima en terminos del tiempo de computo), tiene un tiempo de computo $O\left(\frac{n^2}{\log(n)}\right)$.

A continuacion presentaremos una descripcion detallada del algoritmo bidimensional de Biedl et al [6] que permite reconocer el lenguaje Pal en tiempo $O\left(\frac{n^2}{\log(n)}\right)$.

Asumiremos que la cinta de la maquina tiene un marcador en el origen. Adicionalemente asumiremos que el input esta escrito en la primera fila empezando en la celda justo a la derecha del origen.

Algoritmo bidimensional Biedl et al

Sea w una instancia de Pal de longitud n.

1. (*fase de inicializacion*)
   - Calcule $\lceil \log(n) \rceil$ y escriba el resultado en unario en la segunda fila, utilice ceros para tal fin. (Es posible calcular $\log(n)$ en tiempo $O(n \log(n))$ de la siguiente manera: recorra el input marcando posiciones alternativamente, es decir una si una no una si..... Repita el proceso hasta que todas las posiciones esten marcadas. El numero de iteraciones es igual a $\lceil \log(n) \rceil + 1$)
   - Calcule $\log(\log(n))$ y escriba el resultado en unario en la tercera fila, utilice ceros para tal fin.
   - Calcule $y = \log(n) - \log(\log(n))$. Borre los contenidos de la segunda y tercera fila. Escriba y en unario en la segunda fila, utilice ceros para tal fin.
2. (*Comparacion de bloques*)
   - Empezando con la segunda fila, que contiene la palabra 0^y, copie repetidamente el contenido de la fila actual en la fila que queda justo arriba y al contenido de esta nueva fila sumele 1. Pare cuando escriba en alguna fila la palabra 1^y. Al finalizar este proceso la maquina ha escrito todos los numeros entre 0 y $2^y - 1$, en orden ascendente en las filas $2, 3, ..., 2^y + 1$.
   - Lea w_1, el primer caracter de w, recorra la primera columna entre las filas 2 y $2^y + 1$. En cada celda verifique si el caracter es igual a w_1, si es el caso reemplaze el contenido de la celda por el simbolo $\uparrow$. En caso contrario reemplaze el contenido de la celda por el simbolo $\downarrow$. Repita este proceso para todas las columnas entre la primera y la y-esima.
   - Lea los contenidos de las filas 2 a $2^y + 1$. Si una de estas filas contiene un simbolo $\downarrow$, reemplaze el contenido de esta fila por $\#^y$. Al finalizar el proceso una unica fila contiene la palabra $\uparrow^y$ mientras que todas las demas contienen la palabra $\#^y$. Note que con esto hemos marcado la fila que correponde al contenido del primer bloque.

– Repita el proceso con el bloque del extremo derecho, pero en esta ocasion, lea el contenido del bloque de derecha a izquierda y escriba de derecha a izquierda. Verifique que la fila marcada para el primer y ultimo bloque coinciden. Si no es el caso, pare y rechaze a w. En caso contrario continue. Repita el proceso con el segundo bloque a la izquierda y el segundo bloque a la derecha. Continue despues con los bloques subsiguientes hasta que el par de bloques que usted esta trabajando se intersecten. Cuando esto suceda, verifique que el fragmento restante (el fragmento obtenido al suprimir de w los bloques ya analizados) es un palindromo. Note que este fragmento tiene una longitud acotada por $2y \leq 2\log(n)$. Para verificar que el bloque restante es un palindromo podemos usar el algoritmo estandard para maquinas unidimensionales, lo cual requiere $O\left(\log^2(n)\right)$ unidades de tiempo.

Theorem 11. *(sexta cota superior para Pal)*

El lenguaje Pal puede ser reconocido en tiempo $O\left(\frac{n^2}{\log(n)}\right)$ *usando una maquina bidimensional de una sola cinta.*

Proof. Para probar el teorema es suficiente probar que el tiempo de computo del algoritmo bidimensional de Biedl et al es $O\left(\frac{n^2}{\log(n)}\right)$. Dado $i \in \{1,2\}$ usaremos el simbolo $t_i(n)$ para denotar el tiempo de ejecucion del paso i en inputs de tamaño n. Es claro que $t_1(n) \in O(n\log(n))$. Es suficiente entonces probar que $t_2(n) \in O\left(\frac{n^2}{\log(n)}\right)$.

Sea w un input de tamaño n. En la segunda fase debemos comparar un total de $\frac{n}{2(\log(n)-\log(\log(n)))}$ parejas de factores de w, cada uno de longitud $\log(n) - \log(\log(n))$. Veamos entonces cuanto nos cuesta realizar cada una de estas comparaciones.

Sea $y = \log(n) - \log(\log(n))$ y sea (u,v) la pareja de factores que necesitamos comparar.

Empezamos trabajando con el factor u, y lo primero que hacemos es generar una tabla de altura 2^y y ancho y, que contenga (en orden ascendente) los numeros $0, 1, ..., 2^y - 1$. La tabla la generamos fila por fila. La primera fila, que contiene la palabra 0^y, la generamos en tiempo $O(y)$, generar la fila $i+1$ a partir de la fila i nos cuesta tiempo $O(y)$. Tenemos entonces que generar toda la tabla nos cuesta $O(y2^y)$. Una vez generada la tabla debemos encontrar la fila cuyo contenido es igual a u, esto podemos hacerlo leyendo la tabla columna por columna, el tiempo de computo requerido para este trabajo es $O(y2^y)$.

Debemos realizar el mismo trabajo con el factor v, invirtiendo el sentido de lectura.

Construir las tablas asociadas a estos dos factores e identificar las filas (alturas) que les corresponden nos toma $O(y2^y)$ unidades de tiempo. Lo unico que queda por verificar es que las dos filas calculadas esten ubicadas a la misma altura. Es claro que esto podemos hacerlo en tiempo $O(n)$.

De todo lo anterior tenemos que $t_2(n) \in O\left(\frac{n}{y}y2^y\right)$, note que $O\left(\frac{n}{y}y2^y\right)$ es igual a

$$O\left(\frac{n}{y}y2^y\right) = O\left(n2^y\right) = O\left(n\left(2^{\log(n)-\log\log(n)}\right)\right) = O\left(n\left(\frac{n}{\log(n)}\right)\right)$$

Tenemos entonces que el tiempo de computo del algoritmo de Biedl et al es $O\left(\frac{n^2}{\log(n)}\right)$

Remark 6. ¿De que manera explota la bidimensionalidad de la cinta el algoritmo de Biedl et al? Sea w una palabra de longitud n, sea u su prefijo de longitud z y sea v su sufijo de longitud z. Comparar los bloques u y v usando una cinta unidimensional implica recorrer la cinta $\Omega(z)$ veces. El algoritmo de Biedl et al le asigna a cada bloque de w una altura (un caracter de un alfabeto infinito) que lo caracteriza, comparar dos bloques se reduce entonces a comparar dos alturas lo cual puede hacerse recorriendo la cinta una unica vez. Un algoritmo que optimiza el reconocimiento de palindromos es un algoritmo que minimiza los recorridos a lo largo de las cinta de entrada. Biedl et al logran reducir y recorridos a un unico recorrido, pero para que tal reduccion sea de utilidad es necesario que el calculo de las alturas (que asignammos a los bloques) tome poco tiempo, es alli donde la eleccion del y es crucial.

Problema. Sea $k \geq 3$, el teorema anterior implica que es posible reconocer Pal en tiempo $O\left(\frac{n^2}{\log(n)}\right)$ usando una maquina k-dimensional. ¿Podemos hacerlo mejor? La mejor cota inferior para el reconocimiento de palindromos sobre este tipo de maquinas es $\Omega\left(\frac{n^2}{\log^{k-1}(n)}\right)$ ¿podemos reconocer Pal en tiempo $O\left(\frac{n^2}{\log^{k-1}(n)}\right)$ usando maquinas k-dimensionales? ¿podemos explotar la k-dimensionalidad de la maquina?

8 Maquinas probabilisticas

Una maquina de Turing probabilistica es una maquina de Turing de una sola cinta equipada con un generador de bits aleatorios.

Definition 22. *Una maquina de Turing probabilistica es una sextupla*

$$\mathcal{M} = (Q, q_0, A, F, \Gamma, , \delta)$$

tal que

1. $q_0 \in Q$ *es el estado inicial de la maquina.*
2. Γ *es el alfabeto de la maquina, del cual supondremos que* $\Sigma \cup \{\square\} \subseteq \Gamma$.
3. $A \subseteq F \subset Q$, *donde* A *es el conjunto de los estados de aceptacion y* F *es el conjunto de los estados finales.*
4. δ *es una funcion de* $Q \times \Gamma \times \{0,1\}$ *en* $Q \times \Gamma \times \{\leftarrow, \Diamond, \rightarrow\}$.

Dada $\mathcal{M}$ una maquina de Turing probabilistica supondremos que $\mathcal{M}$ cuenta con dos cintas, la cinta de entrada y una cinta de solo lectura en la que se escribe la secuencia de bits aleatorios generada por la maquina en el instante cero. Dado x un input de la maquina supondremos que en el instante cero la segunda cinta se llena magicamente con una secuencia aleatoria de longitud infinita. Al iniciar la computacion la cabeza de la segunda cinta se desplaza una celda a la derecha por unidad de tiempo. Durante la computacion la maquina trabaja deteministicamente como si procesara dos fragmentos, de un unico input, escritos en cintas diferentes.

Una maquina de Turing probabilistica es un tipo especial de maquina no determinstica, esto implica que dado un input x, la maquina puede realizar muchos computos distintos en el input x (estos computos dependen (estan determinados) por la secuencia aleatoria generada en el instante cero de la computacion).

Sea x un input de $\mathcal{M}$. Cada secuencia $\sigma \in \{0,1\}^*$ genera una unica computacion de $\mathcal{M}$ en el input x, a esta computacion la llamaremos la σ-computacion de $\mathcal{M}$ en x.

Definition 23. *Sea $\mathcal{M}$ una maquina probabilistica.*

1. *Dado $v \in \Sigma^*$ se define*

$$\beta(v) = \Pr_{\sigma}[\textit{la } \sigma\textit{-computacion de } \mathcal{M} \textit{ con input } v \textit{ es aceptante}]$$

2. *Dado $0 \lneqq \varepsilon \lneqq 1$, diremos que $\mathcal{M}$ reconoce L con error ε si y solo si para todo $v \in \Sigma^*$ se tiene que*
   - *Si $v \in L$ entonces $\beta(v) = 1$.*
   - *Si $v \notin L$ entonces $\beta(v) \leq \varepsilon$.*

8.1 Una cota superior: el algoritmo de Pippenger

En esta seccion presentaremos un algoritmo debido a Pippenger [40] que reconoce Pal en tiempo $O(n \log(n))$. Existe un interesante algoritmo alternativo debido a Freivalds [15] que reconoce Pal y cuyo tiempo de computo es $O\left(n \log^2(n)\right)$

Los algoritmos probabilisticos tienen una larga historia dentro de la teoria de la computacion, ellos presentan desventajas respecto a los algoritmos deterministas dado que pueden cometer errores, pero presentan a su vez una gran ventaja: suelen ser mas eficientes. Existe una larga lista de algoritmos probabilisticos que son mas eficientes que todos sus competidores deterministicos. Una pequeñisima lista parcial es la siguiente:

1. El algoritmo de Schwartz-Zippel (consulte la referencia [39]) para determinar si un polinomio en varias variables, sobre un campo finito dado, es no nulo.
2. El algoritmo de Freivalds para multiplicar matrices [15].
3. El algoritmo de Pippenger para reconocer Pal.
4. Los algoritmos de Strassen-Solovay y de Rabin-Scott para reconocer primos [39].

Es importante anotar que todo algoritmo probabilistico puede ser convertido en un algoritmo clasico, aunque ello puede implicar un alto costo: el tiempo de computo del algoritmo clasico podria ser exponencial en el tiempo de computo del algoritmo probabilistico.

Empezemos recordando algunos hechos fundamentales, y ampliamente conocidos, de la teoria de la probabilidad y la teoria de numeros elementales.

Theorem 12. *(desigualdad de Markov)*
Si X es una variable aletoria positiva y $E[X]$ es su valor esperado, se tiene que para todo $a \in \mathbb{R}^+$

$$\Pr[X \geq a] \leq \frac{E[X]}{a}$$

Lemma 8. *Existe N tal que dado $n \geq N$ y dado $\epsilon \in (0,1]$, si se escogen al azar $\frac{2\log(n)}{\epsilon}$ numeros en el intervalo $\{1,...,n\}$, la probabilidad de que al menos uno de ellos sea primo es mayor o igual a $1-\epsilon$.*

Proof. El teorema de los numeros primos asegura que si $\pi(n)$ es igual a

$$|\{i \leq n : i \text{ es primo}\}|$$

entonces $\lim_{n\to\infty}\left(\frac{\pi(n)}{\frac{n}{\log(n)}}\right) = 1$. Esto implica la existencia de un N tal que para todo $n \geq N$

$$\frac{1}{2\log(n)} \leq \Pr_{i\leq n}[i \text{ es primo}] \leq \frac{2}{\log(n)}$$

Fijamos $n \geq N$ y consideramos $(\mathbb{N}^+, \mu_n)$ el espacio de probabilidad cuyo universo es el conjunto de los naturales no nulos y cuya funcion de distribucion es la funcion μ_n definida por

$$\mu(m) = \left(1 - \frac{\pi(n)}{n}\right)^{m-1}\frac{\pi(n)}{n}$$

Sea $X : \mathbb{N}^+ \to \mathbb{R}^+$ la variable aleatoria definida por: $X(m) = m$ para todo $m \geq 1$. El valor esperado de X corresponde a la cantidad esperada de numeros que deberiamos elegir del intervalo $\{1,...,n\}$ antes de elegir el primer primo. Se tiene que $E[X] = \frac{n}{\pi(n)} \leq 2\log(n)$ Dado $\epsilon \in (0,1]$ se tiene que

$$\Pr_{n_1,...,n_{\frac{2\log(n)}{\epsilon}} \in \{1,...,n\}}[\text{Para todo } i \leq 2\log(n), \text{ el numero } n_i \text{ es compuesto}] \geq$$

$$\Pr\left[X \geq \frac{E[X]}{\epsilon}\right] \leq \epsilon$$

Tenemos entonces que si $n \geq N$ y $\epsilon \in (0,1]$, la probabilidad de escoger al menos un numero primo, al escoger $\frac{2\log(n)}{\epsilon}$ numeros del intervalo $\{1,...,n\}$, es mayor o igual que $1-\epsilon$.

Sean x, y dos numeros en el intervalo $\{1, ..., 2^n\}$. ¿Cual es la probabilidad de que al escoger un numero primo en el intervalo $\left\{1, ..., n^4\right\}$, se escoja un primo p que satisfaga $x \bmod p \neq y \bmod p$?

Lemma 9. *Sea $A = \{p_1, ..., p_m\}$ un conjunto de numeros primos, y sean x, y numeros naturales tales que:*

- *$x \neq y$.*
- *Para todo $p \in A$ se tiene que $x \bmod p = y \bmod p$.*

Se tiene entonces que $|x - y| \geq \prod_{i \leq m} p_i$.

Proof. Suponga que para todo $p \in A$ se satisface la ecuacion $x \bmod p = y \bmod p$, se tiene entonces

$$x \bmod \prod_{i \leq m} p_i = y \bmod \prod_{i \leq m} p_i$$

Pero esto implica quesi $x \neq y$ sesatisface la desigualdad $|x - y| \geq \prod_{i \leq m} p_i$.

Pretendemos usar el lema anterior para probar que dados x y y dos numeros diferentes en el intervalo $\{1, ..., 2^n\}$, el conjunto de primos en el intervalo $\left\{1, ..., n^4\right\}$ para los cuales x y y son congruentes, es un conjunto pequeño. A continuacion enunciaremos un lema, cuya prueba el lector podra encontrar en la referencia [23].

Dado n, definimos $P(n)$ como el conjunto $\{i \leq n : i \text{ es primo}\}$

Lemma 10. *Dado $n \geq 150$, se tiene que $\prod_{a \in P(n^2)} a \geq n! 2^n$.*

Lemma 11. *Sea A_n un subconjunto de $P\left(n^4\right)$ de tamaño $\frac{\pi\left(n^4\right)}{2}$, se tiene que $\prod_{a \in A_n} a \gneqq 2^n$*

Proof. Sea B el conjunto de los primeros $\frac{\pi\left(n^4\right)}{2}$ primos. Dado A_n como en la hipotesis del lema se satisface la desigualdad $\prod_{a \in A_n} a \geq \prod_{a \in B} a$. Por lo tanto es suficiente probar que $\prod_{a \in B} a \geq 2^n$. Como $P\left(n^2\right) \subset B$ se tiene lo siguiente

$$\prod_{a \in A_n} a \geq \prod_{a \in B} a \geq \prod_{p \in P(n^2)} a \geq n! 2^n \gneqq 2^n$$

Corollary 6. *Dados $x, y \in \{1, ..., 2^n\}$ y dado*

$$A_{xy} = \left\{p \in P\left(n^4\right) : x \bmod p = y \bmod p\right\}$$

se tiene que si $x \neq y$, se satisface entonces la desigualdad $|A_{xy}| \leq \frac{\pi\left(n^4\right)}{2}$.

Proof. Suponga que $|A_{xy}| \gneqq \frac{\pi(n^4)}{2}$, de los lemas anteriores tenemos que

$$|x-y| \geq \prod_{p \in A_{xy}} p \gneqq 2^n.$$

Pero esto claramente es imposible dado que $x, y \in \{1, ..., 2^n\}$

Lemma 12. *Dado $n \geq N$ y dados $x, y \in \{1, ..., 2^n\}$, se tiene que*

$$\Pr_{p \in P(n^4)} [x \bmod p \neq y \bmod p] \geq \frac{1}{2}$$

Proof. Del lema anterior tenemos que $|A_{xy}| \leq \frac{\pi(n^4)}{2}$, y esto implica que

$$\frac{1}{2} \leq \Pr_{p \in P(n^4)} [p \notin A_{xy}] = \Pr_{p \in P(n^4)} [x \bmod p \neq y \bmod p]$$

Estamos listos para presentar el algoritmo de Pippenger. Sea $\mathcal{P}$ el algoritmo definido a continuacion.

Algoritmo de Pippenger

Con input $x = x_1...x_n$, donde $n \geq N$, el algoritmo $\mathcal{P}$ trabaja de la siguiente manera

1. Calcule $\overline{x}$ y $\log(|x|)$.
2. Sea $X_1X_2....$ el contenido de la cinta aleatoria, mientras $k \leq 64 \log(m)$ y Z sea un numero compuesto haga lo siguiente.
   - $k \leftarrow k+1$.
   - $Z \leftarrow X_{k4\log(|w|)+1}...X_{(k+1)4\log(|w|)}$. (Note que esto es equivalente a generar un numero aleatorio en el intervalo $\left\{0, ..., |w|^4\right\}$.
   - Use un algoritmo deterministico de tiempo polinomial para decidir si Z es un numero primo.
3. Dado $Z = X_{k4\log(|w|)+1}...X_{(k+1)4\log(|w|)}$ el output del paso dos calcule $A = x \bmod Z$ y $B = \overline{x} \bmod Z$.
4. Decida si A y B son iguales, en caso de serlo imprima x es un palindromo, en caso contrario imprima x no es un palindromo.

De lo hecho en los paragrafos anteriores podemos obtener una prueba del siguiente teorema.

Theorem 13. *(Teorema de Pippenger)*

1. *Si $x \in Pal$, la maquina $\mathcal{P}$ acepta x con probabilidad 1.*
2. *Si $x \notin Pal$, la maquina $\mathcal{P}$ rechaza x con probabilidad al menos $\frac{7}{16}$.*

Proof. El item 1 es trivial, probaremos el item 2. Sea x el input de $\mathcal{P}$ y suponga que x no es un palindromo. Defina

$$A_w = \{p \leq 4\log(w) : p \text{ es primo y } w \bmod p \neq \overline{w} \bmod p\}$$

La probabilidad de que $\mathcal{P}$ rechaze a x esta acotada inferiormente por la probabilidad de que en el paso 2 se genere un elemento de A_w. En el paso 2 se realizan hasta $64\log(m)$ experimentos, por lo que la probabilidad de generar, en este paso, un numero primo en el intervalo $\{2, ..., 4\log(|w|)\}$esta acotada inferiormente por $1 - \frac{1}{32}$. Por otro lado la probabilidad de que el primo generado en el paso 2 (si se genera un primo en este paso) pertenezca al conjunto A_w esta acotada inferiormente por $\frac{1}{2}$. De lo anterior tenemos que

$$\Pr[\mathcal{P} \text{ rechaze } w] \geq \frac{31}{64} \geq \frac{7}{16}$$

Hemos demostrado que el algoritmo de Pippenger es un algoritmo aleatorio que reconoce Pal con una probabilidad de error que esta acotada inferiormente por $\frac{9}{16}$. Veamos ahora que este algoritmo puede ser implementado por una maquina probabilistica de una sola cinta cuyo tiempo de computo es $O(n\log(n))$.

Lemma 13. *Existe una maquina de Turing de una sola cinta, llamemosla $\mathcal{M}$, que dado $m \geq 1$, dado $x \in \{1, ..., 2^m\}$ y dado $y \in \{1, ..., m^4\}$ calcula $x \bmod y$ en tiempo $O(m\log(m))$.*

Proof. Supondremos que al comienzo de la computacion la cinta de la maquina contiene el input $y\blacksquare x$. Supondremos ademas que el alfabeto de la maquina es un alfabeto enriquecido que permite multiplicar la capacidad de cada celda por una constante adecuada. Podemos imaginar que cada celda tiene un numero k de filas, la primera de ellas ocupada por el caracter del input que corresponde a la celda, las demas vacias al comienzo de la computacion. Usaremos que multiplicar por dos, dividir por dos, sumar y restar numeros de longitud r puede hacerse en tiempo $O(r)$ usando r celdas de trabajo. Sea $x = y_1...y_m$, podemos escribir a x como $z_1...z_{\frac{m}{4\log(m)}}$ donde cada z_i es un bloque de longitud $4\log(m)$ (excepto el primero posiblemente mas corto). Identificar el bloque $z_{\frac{m}{4\log(m)}}$ puede hacerse en tiempo $O\left(\log^2(m)\right)$, que es lo que cuesta escribir y debajo de $z_{\frac{m}{\log(m)}}$, usando el espacio adicional disponible en las celdas. Identificar el siguiente bloque a la izquierda puede hacerse de la misma manera. Note que $x = \sum_{i=0}^{\frac{m}{\log(m)}-1} 2^{(i-1)\log(m)} z_{\frac{m}{\log(m)}-i}$, por lo que $x \bmod y$ es igual a

$$\left(\sum_{i=0}^{\frac{m}{\log(m)}-1} \left(\left(\left(2^{i-1} \bmod y\right)\left(z_{\frac{m}{\log(m)}-i} \bmod y\right)\right)\right) \bmod y\right) \bmod y \qquad [S1]$$

Dado z tal que $|z| = 4\log(m)$ podemos calcular $z \bmod y$ en tiempo $O(\log(m))$ usando $\log(m)$ celdas de trabajo de la siguiente manera:

1. Decidimos si z es menor que y, en este caso $z \bmod y$ es igual a z. En caso contrario pasamos al item 2.
2. Note primero que dado z tal que $|z| \leq 4\log(m)$ podemos calcular en tiempo $O(\log(m))$, usando busqueda binaria, el minimo k tal que ky es mayor que z. Calculamos k y calculamos $z-(k-1)y$.

Una vez calculado $z_{\frac{m}{4\log(m)}} \bmod y$ escribimos y justo debajo del siguiente bloque a la izquierda, calculamos en tiempo $O\left(\log(m)^2\right)$ el termino $2^{\log(m)} \bmod y$ y guardamos este valor.debajo de y.(tenemos $k \gneqq 3$ lineas disponibles en cada bloque). Finalmente calculamos $\left(\left(2^{i-1} \bmod y\right)\left(z_{\frac{m}{\log(m)}-i} \bmod y\right)\right) \bmod y$ usando las $4\log(m)$ celdas disponibles. Al finalizar este proceso habremos calculado en tiempo $O\left(\log^2(m)\right)$ el segundo sumando de la sumatoria $S1$. Si continuamos trabajando de esta manera podemos calcular todos los sumandos de $S1$ en tiempo $O(m\log(m))$, al final del proceso el sumando i-esimo estara escrito en las $4\log(m)$ celdas del bloque i-esimo y la cabeza lecto-escritora estara al comienzo de la cinta en el extremo izquierdo. Solo nos falta calcular la suma (modulo $\log(m)$) de los bloques. Desplazamos el primer bloque debajo del segundo y sumamos, esto nos toma tiempo $O\left(\log(m)^2\right)$, continuamos de esta manera: desplazando el resultado obtenido en el bloque a la izquierda y sumando, esto toma tiempo $O(m\log(m))$. Al final del proceso la cantidad buscada, (el termino $x \bmod y$), estara escrito en el ultimo bloque a la izquierda. El tiempo de computo de este algoritmo es $O(m\log(m))$.

Theorem 14. *(septima cota superior para Pal)*
Existe una maquina de una sola cinta que implementa el algoritmo de Pippenger y cuyo tiempo de computo es $O(n\log(n))$.

Proof. Simplemente debemos verificar que los 4 pasos del algoritmo antes descrito pueden ser implementados por maquinas de una sola cinta con un tiempo de computo adecuado. Sea $x = x_1...x_n$ un input de $\mathcal{P}$.

1. Es claro que podemos calcular $\overline{x}$ y $\log(n)$ en tiempo $O(n\log(n))$
2. Podemos suponer que al final del paso 1 hemos marcado todas las celdas contenidas en el intervalo de longitud $4\log(n)$ ubicado justo a la derecha del input, este intervalo puede ser usado como patron para extraer de la secuencia de bits aleatorios los numeros de longitud $4\log(n)$ que se generan en el paso 2 del algoritmo. Generar un numero de longitud $4\log(n)$, usando la cinta aleatoria y el patron, nos cuesta $O(\log(n)+r)$, donde r es la longitud del segmento de la cinta aleatoria que ya ha sido empleado. Note que $r \leq 256\log(n)$, por lo que generar cada uno de los numeros del paso 2 nos cuesta $O\left(\log^2(n)\right)$. Dado Z tal que $|Z| \leq 4\log(n)$, chequear la primalidad de Z nos cuesta $O\left(\log^R(n)\right)$ unidades de tiempo, donde R es una constante adecuada mayor o igual que 2. En el peor de los casos debemos generar $64\log(n)$ numeros en el paso 2, generar y procesar cada uno de estos numeros nos

cuesta $O\left(\log^R(n)\right)$ unidades de tiempo, por lo que el tiempo de computo empleado en ejecutar el paso 2 esta acotado por $O\left(\log^{R+1}(n)\right)$.

3. Dado Z el output del paso 2, el paso 3 se reduce a calcular $A = x \bmod Z$ y $B = \overline{x} \bmod Z$ lo cual puede hacerse en tiempo $O(n \log(n))$.
4. Decidir si A y B son iguales puede hacerse en tiempo $O\left(\log^2(n)\right)$ dado que $|A|, |B| \leq 4 \log(n)$.

Remark 7. Hemos diseñado un algoritmo que reconoce Pal con error acotado por $\frac{9}{16}$, usando tecnicas estandard de amplificacion de probabilidades (consulte referencia [39]) es posible, para todo $\epsilon \gneqq 0$ derivar un algoritmo que reconozca Pal con error acotado por ϵ.

Remark 8. (*novena cota inferior para* Pal) Es posible establecer una cota inferior para el reconocimiento de palindromos mediante maquinas probabilisticas, que como ya es costumbre con el lenguaje Pal, coincide con la cota superior ya establecida. Yao pruebe que reconocer el lenguaje Pal usando maquinas probabilisticas requiere tiempo $\Omega(n \log(n))$ [55], la prueba de Yao se basa en un argumento tipo secuencias de cruce, adaptado al contexto de las maquinas probabilisticas.

9 Automatas finitos probabilisticos

Las maquinas de Turing probabilisticas son maquinas de Turing con un criterio de aceptacion probabilistico, de manera analoga los automatas finitos probabilisticos son automatas finitos (de una o dos vias) con un criterio de aceptacion probabilistico. En esta seccion estudiaremos dos tipos de automatas finitos probabilisticos y probaremos que estos automatas son incapaces de reconocer el lenguaje Pal.

Definition 24. *Un automata regular probabilistico es una quintupla*

$$\mathcal{M} = (Q, q_0, A, \delta, \omega)$$

tal que:

1. *La cuadrupla* $\mathcal{N} = (Q, q_0, A, \delta)$ *es un automata regular no deterministico.*
2. ω *es una funcion de* δ *en* $\mathbb{R}^+$ *que satisface la siguiente condicion: dado* $(q, a) \in Q \times \Sigma$ *se tiene que*

$$\sum_{p:(q,a,p)\in\delta} (q, a, p) = 1$$

El lector debe notar que la computacion de un automata probabilistico en un input w es una *cadena de Markov* $K(w)$ [43] cuyo conjunto de estados es el conjunto

$$S = \{(q, l) : q \in Q \text{ y } l \in \{1, ..., n\}\}$$

y cuya matriz de transicion $M(w) = [t_{v,u}]_{v,u \in S}$ es la matriz

$$t_{v,w} = \begin{cases} w(p, w_{l+1}, q) & \text{si } v = (p,l) \text{ y } u = (q, l+1) \\ 0 \text{ en otro caso} \end{cases}$$

Podemos identificar esta cadena de Markov con el digrafo etiquetado $G(\mathcal{N}, w) = (S, E_w, \varpi)$ definido por:

1. Dados $(p,l), (q,r) \in S$ existe una arista de (p,l) a (q,r) si y solo si $r = l+1$ y $\omega(p, w_{l+1}, q) \gneqq 0$.
2. Dada $((p,l),(q,l+1)) \in E_w$ se tiene que $\varpi((p,l),(q,l+1)) = \omega(p, w_{l+1}, q)$.

Dado $\gamma = ((q_0,0),(q_1,1)) ... ((q_{m-1}, m-1),(q_m, m))$

Definition 25. *Sea $\gamma = ((q_0,0),(q_1,1)) ... ((q_{m-1}, m-1),(q_m, m))$ un camino en $G(\mathcal{N})$, la probabilidad del camino γ, que denotaremos con el simbolo $w(\gamma)$, es igual a $\prod_{i \leq n-1} w(q_i, w_{i+1}, q_{i+1})$.*

Definition 26. *Dada $w \in \Sigma^n$ la probabilidad de aceptacion de w, que denotaremos $p_{\mathcal{M}}(w)$, es igual $\sum_{\gamma} w(\gamma)$, donde la suma es una sumatoria sobre el conjunto de los caminos de longitud n que inician en q_0 y terminan en A.*

Definition 27. *Diremos que L es aceptado por $\mathcal{M}$ con error ε si y solo si ocurre lo siguiente:*

- *Si $x \in L$ se tiene que $p_{\mathcal{M}}(w) = 1$.*
- *Si $x \notin L$ se tiene $p_{\mathcal{M}}(w) \leq \varepsilon$.*

Theorem 15. *(decima cota inferior para Pal)*
Dado $0 \leq \varepsilon \lneqq 1$ no existe un automata regular que reconozca el lenguaje Pal con error ε.

Proof. Suponga que $\mathcal{M}$ es un automata regular probabilistico que acepta el lenguaje Pal, note que $\mathcal{M}$ es una maquina de Turing probabilistica de una cinta y de tiempo real, esto implica que Pal puede ser reconocido por una maquina probabilistica en tiempo $o(n \log(n))$ (i.e. existe una maquina probabilistica, digamos $\mathcal{N}$, que reconoce Pal y tal que $\lim_{n \to \infty} \left(\frac{T_{\mathcal{N}}(n)}{n \log(n)} \right) = 0$), contradiciendo el teorema de Yao.

En lo que sigue estudiaremos el modelo de automatas regulares probabilisticos de doble via. Probaremos que estos automatas tampoco pueden reconocer el lenguaje Pal, probarlo sera un poco mas dificil, no podemos afirmar que este hecho es un corolario del teorema de Yao dado que los automatas de doble via pueden tener un tiempo de computo que no esta acotado por $O(n \log(n))$.

Definition 28. *Un automata finito de doble via probabilistico (un $2pfa$ para abreviar) es un tupla*

$$\mathcal{M} = (Q, -, q_0, q_{acc}, q_{rej}, \delta, \varpi)$$

tal que:

1. *Q es un conjunto finito, el conjunto de estados de la maquina.*
2. *$\Gamma = \Sigma \cup \{L, D\}$ es el alfabeto de la maquina.*
3. *$q_0, q_{acc}, q_{rej} \in Q$ son estados distinguidos, q_0 es el estado inicial, q_{acc} y q_{rej} son estados de parada (si el automata accede a uno de estos dos estados la computacion termina), q_{acc} es el estado de aceptacion y q_{rej} el de rechazo.*
4. *$\delta \subseteq \Gamma \times (Q - \{q_{acc}, q_{rej}\}) \times Q \times \{\leftarrow, \Diamond, \rightarrow\}$ es una relacion de transicion no deterministica.*
5. *ϖ es una funcion de peso que asigna probabilidades a cada una de las cuadruplas en δ, la funcion ϖ satisface las siguientes dos condiciones adicionales:*
   - *Para todo $(a, q, p, i) \in \delta$ se satisface que $\varpi((a, q, p, i)) \in \left\{0, \frac{1}{2}\right\}$.*
   - *Para todo $(a, q) \in \Gamma \times (Q - \{q_{acc}, q_{rej}\})$ se tiene que*

$$\sum_{(a,q,p,i)\in\delta} \varpi((a, q, p, i)) = 1.$$

Dado $\mathcal{M}$ un $2pfa$ y dada $w \in \Sigma^*$ un input de $\mathcal{M}$, la computacion de $\mathcal{M}$ en w inicia con el automata en el estado q_0 y la cabeza lectora ubicada sobre la celda 0 que contiene el simbolo L, (el simbolo L es un simbolo especial que indica el inicio del input, mientras que el simbolo D es otro simbolo especial que se usa para indicar el final del input). La dinamica de la cabeza lectora y la dinamica de los estados internos de $\mathcal{M}$ son probabilisticas y estan controladas por la relacion δ. Dada w podemos asignar a w una probabilidad de aceptacion, que denotaremos con el simbolo $p_{\mathcal{M}}(w)$, y que corresponde a la pobabilidad de que el automata pare en el estado q_{acc} al procesar el input x, esta probabilidad es calculada sobre las elecciones aleatorias de la maquina.

Definition 29. *Dado L y dado un $2pfa$ $\mathcal{M}$ diremos que $\mathcal{M}$ acepta L con error ε si y solo si*

- *Si $w \in L$ se tiene que $p_{\mathcal{M}}(w) = 1$.*
- *Si $w \notin L$ se tiene que $p_{\mathcal{M}}(w) \leq \varepsilon$*

Probaremos que no existe un $2pfa$ capaz de reconocer el lenguaje Pal. Para ello consideraremos una nocion un poco mas general de aceptacion, la nocion de *separacion*. Sean $A, B \subset \Sigma^*$ tales que $A \cap B = \varnothing$ y sea $0 \lneqq \varepsilon \lneqq \frac{1}{2}$ diremos que $\mathcal{M}$ separa A de B con error ε si y solo si ocurre lo siguiente:

- Si $w \in A$ se tiene que $p_{\mathcal{M}}(w) \geq 1 - \varepsilon$.
- Si $w \in B$ se tiene que $p_{\mathcal{M}}(w) \leq \varepsilon$.

Remark 9. Dado L un lenguaje y dado ε, si no existe un $2pfa$ capaz de separar L y *co-L* con error ε, no existe entonces un $2pfa$ que reconozca L con error ε

Dado w un input de $\mathcal{M}$ una *condicion inicial* es un par $\sigma = (q, i)$, donde $q \in Q$ e $i \in \{L, D\}$. El significado de una condicion inicial (q, i) es el siguiente:

La computacion de M, en el input w, inicia con el automata en el estado interno q y con la cabeza ubicada en la celda 0 (en el caso en que $i = L$, si $i = D$ la cabeza esta ubicada, al inicio de la computacion, en la celda $|w| + 1$)

Una *condicion final* τ puede ser o un par $\tau = (p, i)$ con $p \in Q$ e $i \in \{L, D\}$ o una expresion del tipo *Acepta, Rechaza, Bucle*. El significado de un par $\tau = (p, i)$ es el siguiente:

l*a maquina accede al estado p con la cabeza ubicada en la celda 0 (si $i = L$, o con la cabeza ubicada en la celda $|x| + 1$ si $i = D$).*

El significado de la condicion final *Bucle* es el siguiente:

la maquina entra en un bucle infinito.

El significado de las otras dos condiciones finales es el obvio.

La computacion de un $2pfa$ en un input w puede ser vista como una cadena de Markov $K(w)$ cuyo conjunto de estados S es el conjunto

$$\{(q, l) : q \in Q \text{ y } l \in \{0, ..., |w| + 1\}\}$$

y cuya matrix de transicion $M(w) = \left[t^K_{st}\right]_{s,t \in S}$ es la matrix definida por:

$$t^K_{st} = \begin{cases} \varpi(p, w_l, q, i) \text{ si } s = (p, l), t = (q, l + 1) \text{ e } i = \rightarrow \\ \varpi(p, w_l, q, i) \text{ si } s = (p, l), t = (q, l - 1) \text{ e } i = \leftarrow \\ \varpi(p, w_l, q, i) \text{ si } s = (p, l), t = (q, l) \text{ e } i = \Diamond \\ 0 \text{ en otro caso} \end{cases}$$

Esta cadena de Markov es una *cadena de Markov absorbente* (i.e. dado $s \in S$ existe $t \in S$ tal que t es accesible desde s y $t^K_{tt} = 1$, los estados t que satisfacen la ecuacion $t^K_{tt} = 1$ son los *estados absorbentes* de la cadena de Markov). Para todo $i \in \{0, ..., |w| + 1\}$ los estados (q_{acc}, i) y (q_{rej}, i) son absorbentes.

Dada K una cadena de Markov absorbente con conjunto de estados $\{1, ..., n\}$ y dado s un estado absorbente, el simbolo $a(K, s)$ denota la *probabilidad de absorcion* en s que es igual a la probabilidad de que la cadena termine en el estado absorbente s si inicia en el estado 1.

Dados a, b y $\beta \gneqq 0$ diremos que a, b son *β-cercanos* si y solo si o $a = b = 0$ o $\beta^{-1} \lneqq \frac{a}{b} \lneqq \beta$. Sean K y K^* dos cadenas de Markov cuyo conjunto de estados es S, si para todo par $s, t \in S$ se tiene t^K_{st} y $t^{K^*}_{st}$ son β-cercanos diremos que K y K^* son β-cercanas.

Lemma 14. *Sean K y K^* dos cadenas de Markov β-cercanas con conjunto de estados $S = \{1, ..., n\}$, si $s \in S$ es un estado absorbente en las dos cadenas, entonces $a(K, s)$ y $a(K^*, s)$ son β^{2n}-cercanos.*

Una prueba de este lema puede ser consultada en la referencia [11].

Sea $\mathcal{M}$ un $2pfa$, sea w un input de $\mathcal{M}$, sea σ una condicion inicial y sea τ una condicion final, el simbolo $p_{\mathcal{M}}(w, \sigma, \tau)$ denote la probabilidad de que el

automata $\mathcal{M}$ acceda a la condicion final τ al procesar el input w cuando la computacion inicia en la condicion inicial σ.

Theorem 16. *(undecima cota inferior para Pal)*

Sean $A, B \subset \Sigma^*$ tales que $A \cap B = \varnothing$. Suponga que existe un conjunto infinito $I \subseteq \mathbb{Z}$ tal que a cada $m \in I$ podemos asignarle un conjunto $W_m \subseteq \Sigma^{\leq m}$ de manera tal que la secuencia $\{W_m\}_{m \in I}$ satisface:

1. *Para todo $k \geq 1$ existe m_k tal que si $m \geq m_k$ y $m \in I$ entonces $|W_m| \geq m^k$.*
2. *Para todo $m \in I$ y para todo par $w, u \in W_m$ con $w \neq u$ existen palabras s, t tales que: o $(swt \in A$ y $sut \in B)$ o $(swt \in B$ y $sut \in A)$.*

Se tiene entonces que para todo $\varepsilon \lneqq \frac{1}{2}$ no puede existir un $2pfa$ que separe A y B con error ε.

Proof. Sea $\beta \lneqq \frac{1}{2}$ y suponga que existe un $2pfa$, llamemoslo $\mathcal{M}$, que separa los conjuntos A y B con error β. Sea c el numero de estados de $\mathcal{M}$ y sea $\mathcal{C}_{\mathcal{M}}$ el conjunto

$$\{(\sigma, \tau) : \sigma \text{es una condicion inicial y } \tau \text{ es una condicion final}\}$$

Note que $d = |\mathcal{C}_{\mathcal{M}}| = 4c^2 + 6c$. Sea $m \geq 1$, sea $w \in W_m$ y sea $p(x) = (p_{\mathcal{M}}(w, \sigma, \tau))_{(\sigma,\tau) \in \mathcal{C}_{\mathcal{M}}}$.

Afirmacion. *Si $p(w)_{(\sigma,\tau)} \gneqq 0$ entonces $p(w)_{(\sigma,\tau)} \gneqq 2^{-cm}$.*

(*prueba de la afirmacion*) Consideremos el caso especial en que τ es una condicion de la forma $(p, |w| + 1)$, si $p(w)_{(\sigma,\tau)} \neq 0$ existe un camino en $K(w)$ de probabilidad no nula, como $K(w)$ tiene a lo mas cm estados no absorbentes y las probabilidades de transicion no nulas son mayores o iguales que $\frac{1}{2}$ existe un camino de σ en $(p, |w| + 1)$ cuya probabilidad es mayor o igual que 2^{-cm}. Tenemos entonces que $p(w)_{(\sigma,\tau)} \gneqq 2^{-cm}$. Los otros casos son similares.

Fije $m \in I$ y divida a W_m en clases de equivalecia de acuerdo a la relacion

$$w \equiv u \text{ si y solo si los vectores } p(w) \text{ y } p(u) \text{ tienen el mismo soporte}$$

donde el soporte de un vector $p(w)$ es el conjunto

$$\left\{(\sigma, \tau) : p(w)_{(\sigma,\tau)} \neq 0\right\}$$

que denotaremos con el simbolo $Sop(w)$. Sea E_m la clase de equivalencia de mayor tamaño, se tiene que $|E_m| \geq \frac{|W_m|}{2^d}$. Sea d^* el tamaño del soporte de los elementos en E_m. Dado $w \in E_m$ usamos el simbolo $P(w)$ para denotar la restriccion de $p(w)$ a los indices que pertenecen a $Sop(w)$. Note que el vector $P(w)$ pertenece a $[2^{-cm}, 1]^{d^*}$. Si usamos el simbolo $\log(P(w))$ para denotar el vector $(\log(P(w)_i))_{i \in Sop(w)}$ se tiene que $\log(P(w)) \in [-cm, 0]^{d^*}$. Divida el intervalo $[-cm, 0]$ en subintervalos disyuntos de longitud μ de manera que

$[-cm, 0]^{d^*}$ es dividido en $\left(\frac{cm}{\mu}\right)^{d^*}$ cajas de tamaño μ^{d^*}. Como $|W_m|$ crece mas rapido que todo polinomio y d^* es constante, existe m_μ tal que para todo $m \geq m_\mu$ se satisface la desigualdad

$$\left(\frac{cm}{\mu}\right)^{d^*} \lneqq \frac{|W_m|}{2^d} \leq |E_m|$$

Fije μ y suponga que $m \geq m_\mu$, existen $w, u \in E_m$ tales que $\log(P(w))$ y $\log(P(u))$ pertenecen a la misma caja, esto implica que si $i \in Sop(w)$ entonces $p(w)_i$ y $p(u)_i$ son 2^μ-cercanos. Sean s y t como en el enunciado del teorema y suponga que $swt \in A$ y $sut \in B$, sean $R(w)$ y $R(u)$ las cadenas de Markov definidas por:

Dado $v \in \{w, u\}$ la cadena de Markov $R(v)$ es una cadena de Markov con conjunto de estados

$$S(v) = \{(q, j) : q \in Q \text{ y } j \in \{1, 2, 3, 4\}\} \cup \{I, Acc, R, B\}$$

El significado de tales estados esta dado por:

1. $(q, 1)$ significa que $\mathcal{M}$ esta en el estado q leyendo el final de s.
2. $(q, 2)$ significa que $\mathcal{M}$ esta en el estado q leyendo el inicio de v.
3. $(q, 3)$ significa que $\mathcal{M}$ esta en el estado q leyendo el final de v.
4. $(q, 4)$ significa que $\mathcal{M}$ esta en el estado q leyendo el inicio de t.
5. I significa que $\mathcal{M}$ esta en el estado q_0 leyendo el caracter L.
6. Acc significa que $\mathcal{M}$ esta en el estado q_{acc}.
7. R significa que $\mathcal{M}$ esta en el estado q_{rej}.
8. L significa que $\mathcal{M}$ entro en un bucle infinito.

Las probabilidades de transicion de $R(v)$ corresponden a las probabilidades $p(v, \sigma, \tau)$ con la interpretacion natural. Los estados Acc, R y L son los estados absorbentes. Sea $l = 2(4c + 4)$ y sea $a(v)$ la probabilidad de que $\mathcal{M}$ acepte la palabra svt, note que $a(v) = a(R(v), Acc)$. Como $swt \in A$ se tiene que $a(w) \geq 1 - \varepsilon$ y como $R(w)$ y $R(u)$ son 2^μ cercanos se tiene que $\frac{a(u)}{a(w)} \geq 2^{-\mu l}$ y $a(u) \geq (1 - \varepsilon) 2^{-\mu l}$. Dado que $\varepsilon \lneqq \frac{1}{2}$ y l es constante podemos elegir m suficientemente grande y μ suficientemente pequeño para que $a(u) \gneqq \frac{1}{2}$. Tenemos entonces que $a(u) = p_{\mathcal{M}}(sut) \gneqq \frac{1}{2}$ y en consecuencia podemos afirmar que $\mathcal{M}$ no separa A de B con error ε.

Corollary 7. *Para todo $\varepsilon \lneqq \frac{1}{2}$ no existe un $2pfa$ que reconozca Pal con error $\varepsilon \lneqq \frac{1}{2}$.*

Proof. Sea $A = Pal$, $B = co\text{-}Pal$, $I = \mathbb{Z}$ y par todo $m \in I$ sea $W_m = \Sigma^m$. Dado $m \geq 1$ y dados $w, u \in W_m$ podemos tomar $s = \epsilon$ y $t = \overline{w}$, es claro que $swt \in Pal$ y $sut \in co\text{-}Pal$.

Remark 10. Dwork y Stockmeyer probaron en [11] un resultado un poco mas general: no existe un protocolo interactivo que use bits aleatorios de conocimieto publico y cuyo verificador sea un $2pfa$ capaz de reconocer el lenguaje Pal. Por otro lado Dwork y Stockmeyer probaron que si existe un protocolo interactivo que usa bits privados y cuyo verificador es un $2pfa$ que usa bits privados capaz de reconocer Pal. Desafortunadamente estudiar estos modelos esta mas alla del alcance de estas notas.

10 Automatas celulares

Los automatas celulares son los modelos mas simples de computacion en paralelo. En esta seccion estudiaremos algunos modelos de automatas celulares. Empezaremos estudiando el modelo de arreglos iterados (*Iterative arrays).*

Definition 30. *Un arreglo iterado n-dimensional es una cudrupla* $\mathcal{N} = (Q, q_0, \Sigma, A, \delta)$ *tal que*

1. Q *es un conjunto finito (el conjunto de estados de* $\mathcal{N}$*).*
2. $\Sigma \subset Q$ *es el alfabeto de la maquina.*
3. $q_0 \in Q$ *es el estado inicial de la maquina.*
4. $A \subseteq Q$ *es el conjunto de estados de aceptacion.*
5. $\delta : Q \times (Q)^{3^n - 1} \to Q$ *es la funcion de transicion de la maquina.*

Dado $u \in \mathbb{Z}^n$ la vecindad de u es el conjunto

$$N_n(u) = \left\{ v \in \mathbb{Z}^n : \max_{i \leq n} \{|u_i - v_i| \leq 1\} \right\}$$

Esta nocion de vecindad define una topologia o, lo que es lo mismo, una estructura de grafo sobre $\mathbb{Z}^n$. Note que $|N_n(u)| = 3^n$. Supondremos que cada uno de los conjuntos $N_n(u)$ esta dotado de un orden para el cual el primer elemento es u. A este grafo lo llamaremos el reticulo cubico de dimension n. Un arreglo iterado n-dimensional esta constituido por el reticulo cubico $\mathbb{Z}^n$, y por un automata finito $\mathcal{N}$. Sobre cada vertice de $\mathbb{Z}^n$ tenemos ubicada una copia de $\mathcal{N}$. En el instante cero todos los vertices de $\mathbb{Z}^n$ (celdas) se encuentran en el estado q_0, (donde q_0 es el estado inicial de $\mathcal{N}$). En el instante i la celda cero recibe el caracter w_i y pasa al estado $\delta\left(w_i, (q_{j,i-1})_{j \in \{2,...,3^n\}}\right)$, donde δ es la funcion de transicion de $\mathcal{N}$ y dado $j \in \{2, ..., 3^n\}$ el estado $q_{j,i-1}$ es el estado, en el instante $i-1$, del vecino j-esimo de cero. Dado $u \neq 0$, el estado de u en el instante i es igual a $\delta\left(q_{1,i-1}, (q_{j,i-1})_{j \in \{2,...,3^n\}}\right)$.

Definition 31. *Dado* $\mathcal{N} = (Q, q_0, \Sigma, A, \delta)$ *un arreglo n-dimensional, el lenguaje reconocido por* $\mathcal{N}$ *es el conjunto de aquellas palabras w para las cuales el estado de la celda cero en el instante* $|w|$*, llamemoslo q, satisface la condicion* $\pi_1(q) \in A$.

Note que nuestra definicion de aceptacion implica que todo lenguaje aceptado por un arreglo es aceptado en tiempo real.

Definition 32. *Dados L y T dos lenguajes sobre el alfabeto Σ el lenguaje $L \cdot T$ es el lenguaje definido por*

$$L \cdot T = \{w \in \Sigma^* : \exists u \in L \exists v \in T \, (w = uv)\}$$

Lo que haremos a continuacion es mostrar que el lenguaje $\Sigma^* \cdot Pal$ no puede ser reconocido por un arreglo iterado. Por otro lado es facil verificar que el lenguaje Pal si puede ser reconocido por un arreglo unidimensional, el lector interesado puede consultar la referencia [9] o estudiar la prueba del teorema 19 la cual puede ser facilmente modificada para obtener un arreglo iterado que reconozca el lenguaje Pal.

Dado $L \subset \Sigma^*$ y dado $k \geq 1$ definimos una relacion de equivalencia $\equiv_{L,k}$ sobre el conjunto Σ^* de la siguiente manera

$v \equiv_{L,k} u$ si y solo si para todo $\alpha \in \Sigma^{\leq k}$ se tiene que $(v\alpha \in L \Longleftrightarrow u\alpha \in L)$

Theorem 17. *(Myhill-Nerode para arreglos)*
Dado $\mathcal{N} = (Q, q_0, \Sigma, A, \delta)$ un arreglo n-dimensional y dado $k \geq 1$ se tiene que la relacion $\equiv_{L(\mathcal{N}),k}$ contiene a lo mas $|Q|^{(2k+1)^n}$ clases de equivalencia.

Proof. Suponga que existe k tal que $\equiv_{L(\mathcal{N}),k}$ contiene al menos $p = |Q|^{(2k+1)^n} + 1$ clases de equivalencia diferentes. Sean $C_1, ..., C_p$ clases de equivalencia diferentes dos a dos y sean $w_1, ..., w_p$ representantes de estas clases. Sea $N_n\left(\overrightarrow{0}, k\right)$ la vecindad de $\overrightarrow{0}$ de radio k. Dado u un input de $\mathcal{N}$, la k-configuracion determinada por u en el instante i es la funcion $q_{i,w} : N_n\left(\overrightarrow{0}, k\right) \to Q$. Note que si existen dos palabras u y v tales que la k-configuracion determinada por u en el instante $|u|$ coincide con la k-configuracion determinada por v en el instante $|v|$, entonces para todo $\alpha \in \Sigma^{\leq k}$ se tiene que

$$u\alpha \in L(\mathcal{N}) \text{ si y solo si } v\alpha \in L(\mathcal{N})$$

Note ahora que el tamaño del conjunto de k-configuraciones esta acotado por $|Q|^{(2k+1)^n}$. Existen entonces $i, j \leq p$ tales que $i \neq j$ y las palabras w_i y w_j determinan las mismas k-configuraciones. Tenemos entonces que $w_i \equiv_{L(\mathcal{N}),k} w_j$. Note que esto ultimo es una contradiccion dado que w_i y w_j son representantes de $\equiv_{L(\mathcal{N}),k}$-clases diferentes.

Theorem 18. *$\Sigma^* \cdot Pal$ no puede ser reconocido en tiempo real por un arreglo iterado.*

Proof. Suponga que $\mathcal{N} = (Q, q_0, \Sigma, A, \delta)$ es un arreglo iterado n-dimensional que reconoce $\Sigma^* \cdot Pal$. Escoja k tal que:

- k es impar.
- $2^{2^{\binom{\frac{k-3}{2}}{}}} \gneqq |Q|^{(2k+1)^n}$.

Sea $f : \Sigma^* \to \Sigma^*$ el homomorfismo definido por la funcion

$$f(0) = 01 \text{ y } f(1) = 11$$

Sea X igual a

$$\{1f(w)00 : w \in \Sigma^m\}$$

Dado $Y \subseteq X$ definimos una palabra w_Y de la siguiente manera:

1. Si $Y = \varnothing$ entonces $w_Y = \epsilon$.
2. Si $Y = \{w_1, ..., w_p\}$, definimos w_Y de manera inductiva
   - $\psi_{Y,0} = \epsilon$.
   - $\psi_{Y,i+1} = (w_{i+1})^R \left(\psi_{Y,i}\right)^R \psi_{Y,i}$, para todo $i \leq p-1$.
   - $w_Y = \psi_{Y,p}$

Es facil verificar lo siguiente

1. $w_Y w_p \in \Sigma^* \cdot Pal$.
2. Para todo $Y \subseteq X$ y para todo $u \in X - Y$ se tiene que $w_A u \notin \Sigma^* \cdot Pal$.
3. Dados $Y, Z \subset X$ dos subcojuntos de X diferentes, w_Y y w_Z no son $\equiv_{L(\mathcal{N}),k}$ equivalentes.

El ultimo hecho en la lista anterior implica que la relacion $\equiv_{L(\mathcal{N}),k}$ contiene al menos $2^{2^{\binom{\frac{k-3}{2}}{}}}$ clases de equivalencia, pero esto es una contradiccion dado que $2^{2^{\binom{\frac{k-3}{2}}{}}} \gneqq |Q|^{(2k+1)^n}$.

A continuacion introduciremos el modelo de automatas celulares unidimensionales de una sola via (1-OCA). Un automata celular unidireccional es un automata celular en el que la informacion solo puede fluir en un sentido (en nuestra definicion de izquierda a derecha, aunque podria ser tambien de derecha a izquierda)

Definition 33. *Un automata celular unidireccional (y unidimensional) es una quintupla $\mathcal{M} = (Q, \Sigma, \#, A, \delta)$ tal que*

1. *Q es un conjunto finito (el conjunto de estados de $\mathcal{M}$).*
2. *$\Sigma \subset Q$ es el alfabeto de la maquina.*
3. *$\# \in (Q - \Sigma)$ es un simbolo especial que se usa para determinar (marcar) el inicio y el fin del input.*
4. *$A \subseteq Q$ es el conjunto de estados de aceptacion.*
5. *$\delta : (Q \cup \{\#\}) \times Q \to Q$ es la funcion de transicion de la maquina.*

Un automata celular $\mathcal{M}$ (unidimensional y unidireccional) cuenta con una cinta semi-infinita, la celda cero de la cinta (el extremo izquierdo) siempre esta ocupado por el caracter # (siempre esta en el estado #). Dado $w = w_1...w_n$ un input de $\mathcal{M}$ y dado $i \leq n$, al inicio de la computacion de $\mathcal{M}$ en el imput w la celda i se encuentra en el estado w_i y la celda $n+1$ se encuentra en el estado #. Cada celda es una cabeza lecto-escritora que puede leer de manera simultanea el caracter (estado) en el que se encuentran ella y su sucesor. La funcion de transicion permite actualizar el estado de cada una de las celdas de manera simultanea. La diferencias basicas entre arreglos iterados unidimensionales y 1-OCA's son esencialmente dos:

1. En los 1-OCA's el input se encuentra escrito en la cinta antes de iniciar la computacion mientras que en los arreglos el input es introducido caracter por caracter una vez iniciada la computacion.
2. En los 1-OCA's el flujo de informacion es unidireccional mientras que en los arreglos es bidireccional

¿Podemos reconocer Pal en tiempo real usando automatas celulares unidireccionales? La respuesta es si, y la prueba de este hecho la obtendremos como una facil consecuencia de un hecho mucho mas general que discutiremos a continuacion.

Definition 34. *Una gramatica libre de contexto $\mathcal{G} = (T, N, S, P)$ es lineal si y solo si todas las reglas de produccion presentes en P son la forma $X \to a$ o $X \to Ya$ o $X \to aY$, donde $X, Y \in N$ y $a \in T$.*

Definition 35. *Un lenguaje libre de contexto L es lineal si y solo si existe una gramatica lineal $\mathcal{G}$ tal que $L = L(\mathcal{G})$.*

Example 2. El lenguaje Pal es lineal, sea $\mathcal{G} = (\{0,1\}, N, S, P)$ la gramatica definida por:

- $N = \{S, X_0, X_1\}$.
- P es el conjunto

$$\{S \to \epsilon/0/1/X_1 1/X_0 0; X_0 \to 0S; X_1 \to 1S\}$$

Remark 11. Lo anterior muestra que la clase $\mathcal{LCFL}$ (de los lenguajes lineales) no esta contenida en $\mathcal{DCFL}$, no deberia sorprender a nadie que el testigo de este hecho sea el lenguaje Pal.

Theorem 19. *Si L es un lenguaje lineal L, puede ser reconocido por un automata celular unidireccional en tiempo real.*

Proof. Sea $\mathcal{G} = (T, N, S, P)$ una gramatica lineal y sea $w = w_1...w_n \in \Sigma^*$. Dado $X \in N$, si $X \to^* a_i...a_j$ existe $Y \in N$ tal que o ($Y \to^* a_i...a_{j-1}$ y $X \to Ya_j \in P$) o ($Y \to^* a_{i+1}...a_j$ y $X \to a_1 Y \in P$), esta observacion es la base para diseñar un automata celular unidireccional que reconozca $L(\mathcal{G})$.

Sean $1 \leq i \leq j \leq n$, definimos un conjunto $N(i,j) \subseteq N$ de la siguiente manera:

– Para todo $i \leq n$ se define

$$N(i,i) = \{X \in N : X \to w_i \in P\}$$

– $N(i,j) = N_1(i,j) \cup N_2(i,j)$.
– $N_1(i,j) = \{X \in N : X \to Ya_j \in P \text{ y } Y \in N(i,j-1)\}$.
– $N_2(i,j) = \{X \in N : X \to a_iY \in P \text{ y } Y \in N(i+1,j)\}$.

Note que $w \in L(\mathcal{G})$ si y solo si $S \in N(1,n)$. Sea $\mathcal{M} = (Q,T)$ el automata unidireccional definido por:

1. $Q = \left(\mathcal{P}(N) \times \Sigma^2\right) \cup \Sigma$.
2. El operador T esta definido por:
   Al procesar el input w, el automata $\mathcal{M}$ hace lo siguiente

– En el instante 1 y para todo $i \leq n$ la i-esima celda accede al estado

$$(N(i,i),(w_i,w_i))$$

– Dado $k \geq 2$ y para todo $i \leq n$ suponemos que el estado de la celda i-esima justo antes de la k-esima iteracion es

$$(N(i,i+k-2),(w_i,w_{i+k-2}))$$

entonces si $i+k-1 \leq n$ en el instante k la celda i-esima calcula

$$(N(i,i+k-1),(w_i,w_{i+k-1}))$$

usando su estado y el estado de su vecino a la derecha que es igual a

$$(N(i+1,i+k-1),(w_{i+1},w_{i+k-1}))$$

si $i+k-1 \gneqq n$ la celda i-esima no cambia su estado.

Note que para el instante n la celda 1 ha calculado $N(1,n)$. Sea $q(1,n)$ el estado de la celda 1 en el instante n. Tenemos entonces que $\mathcal{M}$ acepta w si y solo si S pertenece a $\pi_1(q(1,n))$ si y solo si $w \in L(\mathcal{G})$.

Corollary 8. *(novena cota superior para Pal)*

1. *Pal puede ser reconocido en tiempo real usando automatas celulares unidireccionales.*
2. *Pal puede ser reconocido en tiempo real usando arreglos iterados unidimensionales.*

Definition 36. *Un lenguaje L es 2-lineal si y solo si existe un lenguaje lineal K tal que $L = K \cdot K$.*

Todo lenguaje lineal puede ser reconocido en tiempo real por un automata celular unidireccional, ¿es esto cierto de los lenguajes 2-lineales? La respuesta es no, existen lenguajes 2-lineales que no pueden ser reconocidos por automatas celulares unidireccionales [32]. Sea $Pal^{(2)}$ el lenguaje $Pal \cdot Pal$, ¿$Pal^{(2)}$ puede ser reconocido en tiempo real por un automata celular unidireccional?

Cerraremos esta seccion introduciendo (en aras de la completez) la definicion de automata celular unidimesional y bidireccional (1-CA para abreviar).

Definition 37. *Un automata celular es una quintupla $\mathcal{M} = (Q, \Sigma, \#, A, \delta)$ tal que*

1. *Q es un conjunto finito (el conjunto de estados de $\mathcal{M}$).*
2. *$\Sigma \subset Q$ es el alfabeto de la maquina.*
3. *$\# \in (Q - \Sigma)$ es un simbolo especial que se usa para determinar (marcar) el inicio y el fin del input.*
4. *$A \subseteq Q$ es el conjunto de estados de aceptacion.*
5. *$\delta : (Q \cup \{\#\}) \times Q \times (Q \cup \{\#\}) \to Q$ es la funcion de transicion de la maquina.*

Todo 1-OCA puede ser visto como un 1-CA que no usa la bidireccionalidad del flujo de informacion. Lo anterior implica que Pal puede ser reconocido en tiempo real por un 1-CA.

10.1 La ubicacion de Pal en la jerarquia de Chomsky

Sea $\mathcal{REG}$ la coleccion de los lenguajes regulares, $\mathcal{DCFL}$ la coleccion de los lenguajes libres de contexto deterministicos, $\mathcal{UCFL}$ la coleccion de los lenguajes libres de contexto no ambiguos, $\mathcal{LCFL}$ la coleccion de los lenguajes lineales y $\mathcal{CFL}$ la coleccion de los lenguajes libres de contexto. Tenemos que:

1. $\mathcal{REG} \subsetneqq \mathcal{DCFL}$ dado que $Pal^* \in \mathcal{DFCL} - \mathcal{REG}$ y $\mathcal{REG} \subseteq \mathcal{DCFL}$
2. $\mathcal{REG} \subsetneqq \mathcal{LCFL}$ dado que $Pal^* \in \mathcal{LFCL} - \mathcal{REG}$ y $\mathcal{REG} \subseteq \mathcal{LCFL}$
3. $\mathcal{DCFL} \subsetneqq \mathcal{UCFL}$ dado que $Pal \in \mathcal{UCFL} - \mathcal{DCFL}$ y $\mathcal{DCFL} \subseteq \mathcal{UCFL}$
4. $\mathcal{LCFL} \nsubseteqq \mathcal{DCFL}$ dado que $Pal \in \mathcal{LCFL} - \mathcal{DCFL}$.
5. $\mathcal{DCFL}, \mathcal{UCFL} \nsubseteqq \mathcal{LCFL}$ dado que el lenguaje Pal^*Pal^* es determinista (y entonces no ambiguo) pero no es lineal.
6. Existen lenguajes que son a la vez lineales y ambiguos (i.e. $\mathcal{LCFL} \nsubseteqq \mathcal{UCFL}$). Sea L el lenguaje regular definido por

$$L = \left\{ w \in \{0,1\}^* : |w|_1 \equiv 1 \,(\mathrm{mod}\, 3) \;\text{ y }\; |w|_0 \equiv 2 \,(3) \right\}$$

donde $|w|_i$ denota el numero de ocurrencias del caracter i en la palabra w. El lenguaje $L \cdot Pal$ es lineal y es ambiguo.

Es interesante notar que en todas las separaciones discutidas en el paragrafo anterior el lenguaje Pal (y sus relativos) ha jugado un papel determinante.

Las separaciones $1 - 4$ de la lista anterior han sido verificadas y discutidas en profundidad en los capitulos anteriores, las separaciones $5, 6$ y 7 pueden ser verificadas usando la caracterizacion de lenguajes lineales que introducimos a continuacion.

Definition 38. *Sea $\mathcal{M}$ un automata de pila y sea w un input de $\mathcal{M}$. Si en el instante t de una computacion de $\mathcal{M}$ en el input w, el automata $\mathcal{M}$ escribe en la pila, pero en el instante $t+1$ borra un caracter de la pila decimos que $\mathcal{M}$ ha realizado un giro en esta computacion. Si en el instante t el autmata borra en la pila y en el instante $t+1$ escribe en la pila decimos tambien que el automata $\mathcal{M}$ ha realizado un giro en esta computacion.*

Definition 39. *Un automata de pila $\mathcal{M}$ es un automata de un giro (one-turn pushdown automata) si y solo si para todo w input de $\mathcal{M}$ y para toda computacion de $\mathcal{M}$ en w, el automata $\mathcal{M}$ realiza a lo mas un giro.*

Theorem 20. *L es lineal si y solo si existe un automata de un giro que reconoce L.*

Sea $\mathcal{CSL}$ la coleccion de *lenguajes sensitivos al contexto* (lenguajes que pueden ser reconocidos mediante automatas linealmente acotados, esto es: usando maquinas de Turing de una sola cinta y con la restriccion adicional de que las celdas que pueden ser usadas durante la computacion son las celdas ocupadas por el input al inicio de la la computacion) y sea $\mathcal{TC}$ la coleccion de los lenguajes Turing computables. La jerarquia de Chomsky es la jerarquia

$$\mathcal{REG} \subsetneq \mathcal{DCFL} \subsetneq \mathcal{UCFL} \subsetneq \mathcal{CFL} \subsetneq \mathcal{CSL} \subsetneq \mathcal{TC}$$

De todo el trabajo realizado hasta el momento es posible dar la ubicacion exacta de Pal en esta jerarquia:

$$Pal \in \mathcal{UCFL} \cap \mathcal{LCFL} \cap (\mathcal{DCFL})^c$$

¿Hemos logrado caracterizar la complejidad intrinseca del problema Pal? No realmente, una tal caracterizacion requiere exhibir una clase de maquinas tal que Pal es el problema mas dificil de entre todos los problemas que pueden ser resueltos por este tipo de maquinas, esto es: tal que Pal es completo para la clase de complejidad definida por este tipo de maquinas.

El lector desprevenido podra pensar que no existen problemas abiertos referentes a las propiedades algoritmicas del lenguaje Pal. El lector experimentado sospechara que este no es el caso. Considere la siguientes preguntas: El lenguaje Pal parece capturar lo especifico de los lenguajes libres de contexto no deterministas, no ambiguos y lineales. ¿Es posible precisar la (muy vaga) afirmacion anterior? La intuicion indica que Pal es el lenguaje mas dificil entre aquellos que pueden ser reconocidos usando automatas no ambiguos de un solo giro ¿es Pal completo para esta clase de lenguajes? ¿Bajo que tipo de reduccion? Note que $Pal \in L$ (i.e. Pal puede ser resuelto usando espacio logaritmico). Lo anterior implica que las reducciones que debemos considerar no son reducciones de espacio logaritmico. ¿Que tipo de reduccion, mas debil que espacio logaritmico, podria ser util? En la literatura se han estudiado reducciones no uniformes de profundidad acotada (*constant depth reductions* [45]), estas reducciones no parecen ser de utilidad a la hora de estudiar la complejidad de Pal dado que desde el

punto de vista no uniforme Pal es trivial: el lenguaje Pal puede ser reconocido por una famila de circuitos de profundidad 2, i.e. $Pal \in AC^0$ [45].

Problema. ¿Es Pal completo para $\langle LIN \rangle^{sub(L)}$, la clausura de LIN (la coleccion de lenguajes lineales) bajo reducciones de espacio sublogaritmico?

11 Automatas cuanticos

Las maquinas cuanticas son maquinas de escala microcospica que intentan explotar la riqueza de los estados cuanticos (superposicion de estados). En esta seccion nos limitaremos a estudiar el modelo de automatas finitos de doble via cuanticos (*quantum two-way automata*) los cuales, a diferencia de sus analogos clasicos, pueden reconocer el lenguaje Pal. Es importante señalar que los automatas finitos de una sola via cuanticos solo pueden reconocer lenguajes regulares por lo que son incapaces de reconocer el lenguaje Pal [2].

Se conjetura que los computadores cuanticos son exponencialmente mas potentes que los computadores clasicos, esto es: se conjetura que existen problemas, los cuales pueden ser resueltos en tiempo polinomial con maquinas cuanticas pero requieren tiempo exponencial sobre maquinas clasicas [4]. Una evidencia a favor de esta conjetura es el famoso algoritmo cuantico de Shor [44], el cual permite factorizar enteros y calcular logaritmos discretos en tiempo polinomial (¡permite romper los protocolos criptograficos mas conocidos!), recuerde que a la fecha todos los algoritmos clasicos conocidos que resuelven estos problemas son de tiempo exponencial. Se sabe que las maquinas cuanticas pueden ser estrictamente mas poderosas que las maquina clasicas, Grover [21] descubrio un algoritmo cuantico que encuentra un item dado en una lista no ordenada de tamaño n y que realiza $O(\sqrt{n})$ queries, es facil probar que una maquina clasica que resuelva el mismo problema debe realizar $\Omega(n)$ queries. Recuerde que el no determinismo incrementa el poder de computo de los automatas de pila, pero no lo hace en el caso de los automatas regulares, ¿puede *lo cuantico* incrementar el poder de computo al nivel de los automatas regulares? Si y no, ya hemos mencionado que los automatas cuanticos de una via solo pueden reconocer lenguajes regulares. Por otro lado, en esta seccion, estudiaremos el modelo de automatas de doble via cuanticos y probaremos que estos si pueden reconocer el lenguaje Pal.

Sea Q un conjunto finito, una superposicion de Q-estados es un vector unitario $v \in \mathcal{K}^{|Q|}$, i.e. un vector $|Q|$-dimensional que satisface la ecuacion $\|v\| = 1$ (donde $\mathcal{K}$ es un campo adecuado que puede ser o $\mathbb{R}$ o $\mathbb{C}$ dependiendo de la aplicacion en mente). Podemos identificar cada $q \in Q$ con un vector de la base canonica de $\mathcal{K}^{|Q|}$, al vector asignado al elemento q lo denotaremos con el simbolo v_q, tenemos entonces que una superposicion es un vector $\sum_{i \leq |Q|} \alpha_q v_q$

[4] La tesis fuerte de Church afirma que toda maquina computadora puede ser simulada en tiempo polinomial por una maquina de Turing. Las maquinas cuanticas podrian ser el primer contraejemplo a la tesis fuerte de Church. Realizaciones concretas de maquinas cuanticas (por ejemplo de la maquina de Shor) podrian ser construidas en el futuro cercano, no es este el caso con las maquinas no deterministicas, las cuales son un modelo de computacion esencialmente teorico.

tal que $\sum_{q\leq|Q|}|\alpha_q|^2=1$. Si el conjunto Q es el conjunto de estados de una maquina cuantica, una superposicion de q estados es una superposicion de los estados posibles de la maquina, en el universo cuantico un objeto puede estar simultaneamente en varios estados, en cada uno con diferentes amplitudes (no probabilidades, las amplitudes pueden ser negativas o pueden ser complejas), la unica condicion es que la suma de las normas de estas amplitudes sea igual a 1.

La evolucion de un sistema cuantico esta gobernada por un, o por un conjunto de, operador(es) unitario(s). Recuerde que un operador unitario sobre $\mathcal{K}^{|Q|}$ es un endomorfismo lineal que preserva la norma. Es posible caracterizar los operadores unitarios como los endomorfismos que satisfacen la ecuacion

$$U^{-1}=U^t$$

donde el superindice t denota la operacion de transposicion matricial.

En el universo cuantico existen estados, operadores unitarios y mediciones. Una medicion ortogonal es un conjunto $\{P_i\}_{i\leq n}$ de operadores lineales que satisface las condiciones:

1. $P_i=P_i^t$.
2. $P_i^2=P_i$.
3. $P_iP_j=0$, si $i\neq j$.
4. $\sum P_i=I$.

Un conjunto $\{P_i\}_{i\leq n}$ es una medicion ortogonal sobre el espacio $\mathcal{K}^{|Q|}$ si y solo si existe una descomposicion ortogonal de $\mathcal{K}^{|Q|}$, digamos $V_1\oplus...\oplus V_n$ tal que para todo $i\leq n$ se tiene que P_i es la proyeccion ortogonal de $\mathcal{K}^{|Q|}$ en V_i (medir es poyectar desde un espacio complejo, que no podemos visualizar, en un espacio de baja complejidad).

Dada v una supersposicion podemos aplicar a v un operador unitario U y obtener una nueva superposicion Uv o podemos aplicarle una medicion ortogonal $\{P_i\}_{i\leq n}$, el principio de incertidumbre de Heisenberg indica que una superposicion cuantica no puede ser observada directamente, tan solo puede ser medida, y al ser medida se transforma en una nueva superposicion que no podemos predecir directamente. Al aplicar una medicion $\{P_i\}_{i\leq n}$ a una superposicion , ocurre lo siguiente:

1. Para cada $i\in\{1,...,n\}$ el resultado de la medicion es i con probabilidad $\|P_iv\|^2$.
2. Si el resultado de la medicion es i, la superposicion se transforma en $\frac{P_iv}{\|P_iv\|}$.

Definition 40. *Un automata cuantico de doble via es una tupla*

$$\mathcal{M}=(Q,S,q_0,s_0,\Theta,\delta,S_{acc},S_{rej})$$

tal que:

1. *Q es un conjunto finito, el conjunto de estados cuanticos de $\mathcal{M}$.*

2. *S es un conjunto finito, el conjunto de estados clasicos de $\mathcal{M}$.*
3. *$q_0 \in Q$ es el estado cuantico inicial.*
4. *$s_0 \in S$ es el estado clasico inicial.*
5. *$S_{acc} \cup S_{rej} \subset S$ es el conjunto de estados de parada. Si la maquina accede a un estado de parada la computacion se detiene. S_{acc} es el conjunto de estados de aceptacion y S_{rej} es el conjunto de estados de rechazo.*
6. *Θ determina la evolucion de los estados de la maquina. En un instante de la computacion el estado cuantico de la maquina es una superposicion $\sum \alpha_q v_q$. Dada $(s,\sigma) \in (S - (S_{acc} \cup S_{rej})) \times \Sigma$ una configuracion clasica de la maquina $\Theta(s,\sigma)$ es o un operador unitario o una medicion ortogonal que se aplica, en ambos casos, al estado cuantico $\sum \alpha_q v_q$.*
7. *δ determina la evolucion de los estados clasicos y el movimiento de la cabeza. Si $\Theta(s,\sigma)$ es un operador unitario, entonces $\delta(s,\sigma)$ es un elemento de $S \times \{\leftarrow, \Diamond, \rightarrow\}$, si $\Theta(s,\sigma)$ es una medicion entonces $\delta(s,\sigma)$ es una funcion del conjunto de resultados posibles de la medicion en $S \times \{\leftarrow, \Diamond, \rightarrow\}$ (note que δ, al depender de las mediciones, es una funcion de transicion probabilista).*

Dado $w = w_1 ... w_n$ un input de $\mathcal{M}$, la computacion de la maquina procede de la siguiente manera. En el instante cero la cinta de la maquina contiene la palabra $Lw_1...w_nD$ que ocupa las celdas $\{0, ..., n+1\}$, los simbolos L y D son simbolos especiales que indican el inicio y el fin del input. En este mismo instante, el instante cero, la cabeza de la maquina esta en la celda cero, el estado cuantico es v_{q_0} y el estado clasico es s_0. La cabeza de la maquina no puede pasar a la izquierda de la celda ocupada por el simbolo L y no puede pasar a la derecha de la celda ocupada por el simbolo D. En cada transicion se transforma primero el estado cuantico de acuerdo a Θ, y dependiendo del resultado de esta transformacion la funcion δ determina la evolucion del estado clasico y de la cabeza lectora.

Dado w, el simbolo $\Pr_{\mathcal{M}}(w)$ denota la probabilidad de que la maquina acceda a un estado de aceptacion. Es posible que un automata de doble via no pare al procesar un input, diremos que $\mathcal{M}$ es un automata correcto si y solo si para todo w la probabilidad de que $\mathcal{M}$ pare, al procesar el input w, es igual a 1. Dado un automata correcto de doble via cuantico, digamos $\mathcal{M}$, diremos que $\mathcal{M}$ acepta el lenguaje L con error ε si y solo si ocurre lo siguiente

- $w \in L$ implica que $\Pr_{\mathcal{M}}(w) = 1$.
- $w \notin L$ implica que $\Pr_{\mathcal{M}}(w) \leq \varepsilon$.

En lo que sigue probaremos que dado $\varepsilon \gneqq 0$ existe un automata cuantico de doble via $\mathcal{M}$ (y correcto) que reconoce el lenguaje Pal con error ε. El conjunto de estados cuanticos del automata $\mathcal{M}$ es el conjunto $\{q_0, q_1, q_2\}$, la funcion Θ asociada a $\mathcal{M}$ depende de los operadores unitarios

$$U_0 = \begin{bmatrix} \frac{4}{5} & \frac{3}{5} & 0 \\ -\frac{3}{5} & \frac{4}{5} & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ y } U_1 = \begin{bmatrix} \frac{4}{5} & 0 & \frac{3}{5} \\ 0 & 1 & 0 \\ -\frac{3}{5} & 0 & \frac{4}{5} \end{bmatrix}$$

Y la medicion ortogonal es el conjunto $\{\pi_0, \pi_1\}$, donde π_0 es la proyeccion ortogonal sobre el espacio $\langle v_{q_0} \rangle$ y π_1 es la proyeccion ortogonal sobre el espacio $\langle v_{q_1}, v_{q_2} \rangle$.

Sean X_0 y X_1 las matrices $5U_0$ y $5U_1$ (respectivamente), sea $f : \mathbb{Z}^3 \to \mathbb{Z}$ la funcion

$$f(u) = 4u_1 + 3u_2 + 3u_3$$

y sea $K \subset \mathbb{Z}^3$ el conjunto

$$\left\{ u \in \mathbb{Z}^3 : u_1 \bmod 5 \neq 0,\ f(u) \bmod 5 \neq 0 \text{ y } u_2 u_3 \bmod 5 = 0 \right\}$$

El lema a continuacion es un excelente ejercicio de algebra lineal cuya prueba dejaremos al lector (el lector interesado puede consultar la referencia [2]).

Lemma 15. *Sea u un elemento de $\mathbb{Z}^3$, se tiene lo siguiente*

1. *Si $u \in K$ entonces $X_0 u$ y $X_1 u$ pertenecen a K.*
2. *Si existen $v, w \in \mathbb{Z}^3$ tales que $u = X_0 v = X_1 w$ entonces u no pertenece a K.*
3. *Si $u = Y_1^{-1} ... Y_n^{-1} Z_n ... Z_1 e_1$ y para todo $i \leq n$ se tiene que $Y_i, Z_i \in \{X_0, X_1\}$, entonces*

$$u_2^2 + u_3^2 = \begin{cases} 0 \text{ si para todo } i \leq n \text{ se tiene que } Y_i = Z_i \\ \gneqq 25^{-n} \text{ en caso contrario} \end{cases}$$

Theorem 21. *(decima cota superior para Pal)*

Dado $\varepsilon \gneqq 0$ existe un automata cuantico de doble via cuantico y correcto, llamemoslo $\mathcal{M}$, que reconoce el lenguaje Pal con error ε.

Proof. El automata $\mathcal{M}$ esta definido de la siguiente manera.

1. La computacion de $\mathcal{M}$ en el input $w = w_1 ... w_n$ inicia con el automata en el estado cuantico $v_{q_0} = e_1 = (1, 0, 0)$, el automata recorre la cinta de izquierda a derecha hasta encontrar el simbolo D. En esta etapa, al leer un simbolo x_i, el automata $\mathcal{M}$ aplica al estado cuantico actual el operador $U_{k(i)}$ definido por

$$U_{k(i)} = \begin{cases} U_0 \text{ si } w_i = 0 \\ U_1 \text{ en caso contrario} \end{cases}$$

   Al encontrar el simbolo D la cabeza del automata se desplaza al extremo izquierdo de la cinta de entrada.
2. En la segunda etapa, el automata recorre nuevamente la cinta de izquierda a derecha, al leer el caracter w_i aplica al estado cuantico actual el operador $U_{k(i)}^{-1}$ definido por

$$U_{k(i)}^{-1} = \begin{cases} U_0^{-1} \text{ si } w_i = 0 \\ U_1^{-1} \text{ en caso contrario} \end{cases}$$

   Note que al finalizar la segunda etapa el estado cuantico q es igual a

$$\begin{aligned} & U_{k(n)}^{-1} ... U_{k(1)}^{-1} U_{k(n)} ... U_{k(1)} e_1 \\ = & \ 25^{-n} X_{k(n)}^{-1} ... X_{k(1)}^{-1} X_{k(n)} ... X_{k(1)} e_1 \end{aligned}$$

y note que para todo $i \leq n$ se tiene $U_{k(i)} = U_{k(n-i)}$ si y solo si x es un palindromo, entonces de acuerdo al lema 15 se tiene que

$$\left\|\pi_1(q)\right\|^2 = \begin{cases} 0 \text{ si } w \text{ es un palindromo} \\ \gneqq 25^{-2n} \text{ en caso contrario} \end{cases}$$

3. Lo anterior sugiere que es un buen momento para aplicar una medicion, al aplicar una medicion al estado q se obtiene lo siguiente:
   - Si w es un palindromo el resultado de la medicion es 0 con probabilidad 1.
   - Si w no es un palindromo, el resultado de la medicion es igual a 0 con probabilidad $\left\|\pi_1(q)\right\|^2$ o igual a 1 con probabilidad $\left\|\pi_2(q)\right\|^2 \gneqq 25^{-2n}$. En este punto, al terminar la primera ronda, el automata $\mathcal{M}$ ya distingue entre palindromos y no palindromos, a los palindromos los *acepta* con probabilidad 1 y a los no palindromos los rechaza con probabilidad mayor que 25^{-2n}, probabilidad que desafortunadamente es negligible para n grande (la robabilidad de rechazo depende de la longitud del input), ¿que puede hacerse? La unica solucion a la vista es amplificar esta probabilidad de rechazo.

   Si el resultado de la medicion es 1, el automata para y rechaza el input. Si el resultado de la medicion es 0, el estado cuantico, despues de la medicion, se transforma en $\frac{\pi_0(q)}{\|\pi_0(q)\|} = e_1$. Si el estado clasico de la maquina era igual a p, el estado clasico se transforma despues de la medicion en $(p,0)$ (suponemos que $S = A \cup (A \times \{0,1\})$, donde A es un conjunto finito adecuado).
4. La cabeza de la maquina se desplaza de derecha a izquierda hasta encontrar el caracter L, durante el proceso realiza lo siguiente:
   Sea $k \geq \max\{\log(25), -\log(\varepsilon)\}$, al leer cada uno de los caracteres la maquina simula la generacion de k bits aleatorios: sea $v = \frac{1}{\sqrt{2}}e_1 + \frac{1}{\sqrt{4}}(e_2 + e_3)$, la maquina emplea la supersposicion v como un estado cuantico auxiliar y realiza k mediciones sobre v, sea u_i el resultado de la i-esima medicion, note que

   $$\Pr[u_i = 0] = \frac{1}{2} \text{ y } \Pr[u_i = 1] = \frac{1}{2}$$

   Si al realizar la simulacion se obtiene al menos un 0, el estado clasico de la maquina pasa a ser $(p,1)$, en caso contrario el estado clasico se mantiene igual a $(p,0)$. Si al leer el caracter L el estado clasico es igual a $(p,0)$ la maquina acepta el input, en caso contrario repite los pasos 1 a 4.

Suponga que w no es un palindromo, el estado cuantico de la maquina al final de una ronda es igual a $(p,0)$ (para algun $p \in A$) con probabilidad menor o igual que $2^{-k(n)} \lneqq \varepsilon$ por lo que podemos asegurar que un no palindromo sera aceptado con probabilidad menor que ε, esto es: la maquina reconoce Pal con error ε. Por otro lado, es imposible que la maquina rechaze un palindromo, y es facil probar que si w es un palindromo la probabilidad de que la maquina pare y acepte w es igual a 1, esto es: el automata $\mathcal{M}$ es correcto.

Theorem 22. *(doudecima cota inferior para Pal)*
No existe un automata cuantico de doble via que reconozca el lenguaje Pal en tiempo real.

Proof. Suponga que existe un automata cuantico de doble via que reconoce el lenguaje Pal en tiempo real, sea $\mathcal{M}$ un tal automata. Como Pal es un lenguaje no trivial, $\mathcal{M}$ debe leer la totalidad del input que se encuentre procesando. Ahora, dado que $\mathcal{M}$ es un automata de tiempo real, $\mathcal{M}$ es un automata de una sola via. Tenemos entonces que Pal es regular, (todo lenguaje reconocido por automata cuantico de una sola via es regular [2]), pero esto contradice el corolario 1.

Remark 12. Es importante notar que no es posible acotar el tiempo de computo del automata $\mathcal{M}$, lo mas que podemos decir es que $\mathcal{M}$ es un automata correcto (para con probabilidad 1). ¿Existe un automata cuantico de doble via que reconozca Pal, que siempre pare y cuyo tiempo de computo este acotado por una funcion recursiva? ¿Existe uno de tiempo lineal?

12 Ejercicios capitulo 1

1. Dado n el lenguaje Pal_n es el lenguaje finito definido por

$$Pal_n = \{w \in Pal : |w| = n\}$$

   pruebe que todo automata regular que reconozca Pal requiere cuando menos n estados, use esto para derivar una prueba alternativa de la no regularidad de Pal.
2. Pruebe el lema de Ogden para lenguajes libres de contexto (consulte la referencia [28]).
3. Use el lema de Ogden para probar que el lenguaje $\left\{ww : w \in \{0,1\}^*\right\}$ no es libre de contexto.
4. Diseñe una maquina de Turing de una sola cinta que reconozca Pal en tiempo $O\left(n^2\right)$.
5. Pruebe que dada $\mathcal{M}$ una maquina bidimensional existe una maquina unidimensional equivalente (i.e. que reconoce el mismo lenguaje), llamemosla $\mathcal{N}$, tal que $T_{\mathcal{N}}(n) \in O\left(\mathcal{T}_{\mathcal{M}}(n)^4\right)$.
6. Pruebe todas las afirmaciones, lemas y teoremas que se dejaron sin probar en este capitulo y que usted considere no evidentes.
7. Pruebe que un lenguaje libre de contexto es lineal si y solo si puede ser reconocido por un automata de pila de un solo giro.
8. Un automata de cola (queque automata) es un automata con una unidad externa de memoria llamada cola, en los automatas de cola la cabeza de la unidad externa de memoria esta ubicada, a lo largo de toda la computacion, al inicio y no al final (como enlos automatas de pila) mientras que la informacion que entra en la cola se ubicara el final de la misma (como en los automatas de pila). Investigue la nocion de automata de cola (consulte referencia [28], consulte wiki). Diseñe un automata de cola que reconozca

Pal. Muestre que los automatas de cola son universales, esto es: pueden reconocer cualquier lenguaje Turing computable. Los automatas de cola no son automatas de tiempo real, ¿cual es el tiempo de computo requerido por un automata de cola para reconocer Pal?

9. Pruebe que si $\mathcal{M}$ es un automata de cola que reconoce Pal, entonces $t_{\mathcal{M}}(n) \in \Omega\left(\frac{n^{\frac{4}{3}}}{\log(n)}\right)$, (ayuda: consulte la referencia [35]).
10. Pruebe que toda maquina de Turing puede ser simulada por un automata con dos pilas (ayuda: muestre que una cola puede ser simulada por dos pilas).

Maquinas multicinta

En esta seccion introduciremos el modelo de maquinas de Turing multicinta. Dado $k \geq 2$ una maquina de Turing con k cintas es una maquina de Turing que ademas de contar con la cinta de entrada, en la cual se escribe el input antes de iniciar el computo, cuenta con $k-1$ cintas adicionales las cuales pueden ser usadas como espacio de trabajo. Adicionalmente la maquina cuenta con k-cabezas lecto-escritoras (una por cada cinta) que se mueven de manera independiente.

Definition 41. *Una maquina de Turing con k cintas es una sextupla*

$$(Q, \Gamma, q_0, F, \delta, A)$$

tal que:

1. *Q es un conjunto finito, el conjunto de estados de la maquina.*
2. *Γ es un conjunto finito, el alfabeto de la maquina que satisface $\Sigma \cup \{\square\} \subseteq \Gamma$.*
3. *$q_0 \in Q$ es el estado inicial de la maquina.*
4. *$F \subset Q$ es el conjunto de estados finales de la maquina.*
5. *La funcion de transicion δ es una funcion de $(\Gamma)^k \times Q$ en $(\Gamma)^k \times Q \times \{\leftarrow, \Diamond, \rightarrow\}^k$.*
6. *$F \subset A \subset Q$ y si el estado interno de la maquina pertenece a A la maquina para.*

Es posible establecer los siguientes resultados

- Dada $\mathcal{M}$ una maquina con k-cintas, existe $\mathcal{M}^*$ una maquina de una sola cinta que simula $\mathcal{M}$. Adicionalmente se tiene que para todo $n \in \mathbb{N}$ se satisface la desigualdad $T_{\mathcal{M}^*}(n) \leq k\left(T_{\mathcal{M}}(n) + n + 2k\right)^2$ (esto es: $T_{\mathcal{M}}(n) \in O\left(\left(T_{\mathcal{M}^*}(n)\right)^2\right)$)
- El lenguaje Pal puede ser decidido en tiempo lineal (¡incluso en tiempo real!) por una maquina multicinta.

Tenemos entonces que la adicion de hardware ($k-1$ cintas adicionales) incrementa el poder de computo pero de una manera modesta. Lo incrementa porque permite decidir algunos lenguajes computables de manera mas eficiente. El incremento en poder de computo es modesto dado que las nuevas maquinas son capaces de decidir exactamente los mismo lenguajes y ademas porque el incremento en eficiencia (aceleramiento, *speed up*) es de caracter polinomial, y mas especificamente cuadratico.

Theorem 23. *(undecima cota superior para Pal)*
El lenguaje Pal puede ser decidido en tiempo lineal por una maquina multicinta.

Proof. Sea M la maquina de dos cintas definida por:

- Al comienzo de la computacion, el input w aparece escrito en el extremo izquierdo de la cinta 1, la cinta 2 se encuentra vacia, las dos cabezas lecto-escritoras de la maquina se encuentran ubicadas en el extremo izquierdo de cada cinta.
- Supongamos que el input w tiene longitud n. En la primera etapa del computo las dos cabezas se desplazan n posiciones a la derecha. Mientras la cabeza de la cinta de trabajo se mueve sobre su cinta va escribiendo los caracteres de w (que la cabezade la cinta de entrada esta leyendo), esto es: en la primera etapa simplemente se copia el input en la cinta de trabajo.
- En la segunda etapa del computo la cabezade la cinta de entrada se desplaza al extremo izquierdo de su cinta.
- En la tercera etapa las dos cabezas se mueven de manera simultanea, la cabeza de la cinta de entrada se mueve de izquierda a derecha leyendo el input, la cabeza de trabajo se mueve de derecha a izquierda. Las dos cabezas van leyendo los contenidos de sus cintas. La cabeza de trabajo lee a w de derecha a izquierda, mientras que la cabeza de la cinta de entrada lee a w de izquierda a derecha. La maquina compara paso a paso los caracteres leidos por cada una de las cabezas. Si en algun instante de esta etapa las cabezas se encuentran leyendo caracteres distintos la maquina para y rechaza el input. Si todos los caracteres coinciden la maquina acepta el input.

Es facil convencerse de que la maquina M acepta el lenguaje Pal. Adicionalmente es facil convencerse de que el tiempo de computo de M es $3n$.

13 Reconocimiento de palindromos en tiempo real usando maquinas multi-cinta.

En esta seccion estudiaremos el teorema de Galil-Slisenko [16] y [46] el cual afirma que Pal puede ser reconocido en tiempo real usando maquinas multicinta.

Un problema algoritmico es un problema *online*, si al procesar una cualquiera de sus instancias estamos obligados a procesar cada uno de sus caracteres una vez estos son recibidos (leidos), y no podemos aplazar la tarea de procesarlos (usar la informacion que nos proporcionan), note que esto ultimo siempre es posible si la cabeza lectora de la cinta de entrada es bidireccional: ignoramos el caracter, seguimos adelante, nos devolvemos y usando informacion aposteriori (caracteres a la derecha) lo procesamos. Tetris, el problema consistente en apilar de la mejor manera las piezas que van apareciendo en pantalla, es un problema online: debemos acomodar cada una de las piezas (caracteres) cuando estas aparecen en pantalla, no podemos seguir adelante, devolvernos y reacomodarlas.

La nocion de problema online sugiere la nocion de maquina online.

Definition 42. *Dada $\mathcal{M}$ una maquina de Turing con una cinta de entrada de solo lectura y $k-1$ cintas de trabajo, diremos que $\mathcal{M}$ es una maquina online si y solo si la cabeza lectora de la cinta de entrada solo puede moverse de izquierda a derecha, si por el contrario la cabeza lectora de la cinta de trabajo puede moverse en las dos direcciones diremos que la maquina es offline.*

Remark 13. No todo problema es online (no todo problema puede ser resuelto online). considere por ejemplo la funcion $F : \{0,1\}^* \to \mathbb{N}$ definida por

$$F(w)_i = 1 \text{ si y solo si } i \text{ es le centro de un palindromo incial de } w$$

Es claro que determinar el i-esimo bit de $F(w)$, (i.e. el bit $F(w)_i$) implica leer el segmento $w[1...2i+1]$ de w.

La nocion informal de tiempo real que hemos mencionado hasta el momento corresponde a lo siguiente: un problema L puede ser resuelto en tiempo real si y solo si existe un automata que resuelve L y tal que para todo $w \in \Sigma^*$ se satisface la desigualdad

$$t_{\mathcal{M}}(w) \leq |w|$$

La nocion de tiempo real que se usa en la literatura es un poco mas flexible, (pero esencialmente equivalente).

Definition 43. *(Tiempo real)*

1. *Dada $\mathcal{M}$ una maquina online, dada $w \in \Sigma^*$ y dado $a \in \Sigma$, el simbolo $\Delta t_{\mathcal{M}}(w,a)$ denota el numero de transiciones que realiza la maquina $\mathcal{M}$ justo despues de leer el ultimo caracter de w y antes de leer el caracter a.*
2. *Una maquina online $\mathcal{M}$ es una maquina de tiempo real si y solo si existe c tal que para todo $w \in \Sigma^*$ y para todo $a \in \Sigma$ se tiene que $\Delta t_{\mathcal{M}}(w,a) \leq c$.*
3. *Un problema L es resoluble en tiempo real si y solo si existe una maquina de tiempo real que lo resuelve.*

La definicion de maquina online implica que para todo $w \in \Sigma^*$ se satisface la desigualdad $t_{\mathcal{M}}(w) \leq c|w|$, la cual es menos exigente que la desigualdad asociada a la nocion intuitiva de tiempo real. El teorema del *aceleramiento lineal* (*linear speed up*, consulte la referencia [39]) implica que toda maquina online $\mathcal{M}$ de tiempo real puede ser transformada en una maquina $\mathcal{N}$ que satisfaga la desigualdad

$$t_{\mathcal{N}}(w) \leq |w|$$

esto es: la nocion intuitiva de tiempo real es equivalente a la nocion tecnica de tiempo real que acabamos de introducir. Todo problema que pueda ser resuelto en tiempo real, de acuerdo a la segunda definicion, puede ser resuelto en tiempo real de acuerdo a la primera definicion (supuestamente mas exigente).

Sea $f : \Sigma^* \to \{0,1\}^*$ una funcion computable, diremos que f es una funcion online si y solo si satisface:

1. Para todo $w \in \Sigma^*$ se tiene que $|f(w)| = |w|$.
2. Para todos $u, w \in \Sigma^*$ si $u \sqsubset w$ se tiene que $f(u) \sqsubset f(w)$ o, lo que es lo mismo, para todos $u, w \in \Sigma^*$ se tiene que $f(uw) = f(u) f(w)$.

Las dos condiciones anteriores implican que si $\mathcal{M}$ es una maquina online que calcula f, la maquina imprimira el caracter i-esimo del output justo despues de leer el i-esimo caracter del input.

Sea $\chi_{pal} : \{0,1\}^* \to \{0,1\}^*$ la funcion definida por

para todo $i \leq |w|$ se tiene que $\left(\chi_{pal}(w)\right)_i = 1$ si y solo si $w_1...w_i \in Pal$

Galil probo en [16] que la funcion χ_{pal} puede ser calculada en tiempo real, este resultado y parte de su prueba es el tema de esta seccion.

13.1 Algoritmos predecibles

Sea $f : \Sigma^* \to \{0,1\}^*$ una funcion online y sea $k_f : \{0,1\}^* \to \mathbb{N}$ la funcion definida por

$$k_f(x) = \min\left\{|z| : \exists a \in \Sigma\left(f(xza) = f(x)\,0^{|z|}1\right)\right\}$$

Definition 44. *Dado $\mathcal{M}$ un algoritmo que calcula la funcion f decimos que $\mathcal{M}$ satisface la condicion de predictibilidad si y solo si existe una constante c tal que:*

1. *$\Delta t_{\mathcal{M}}(w,a) \geq c$ implica que $f(wa) = f(w)\,0$.*
2. *$\Delta k_f(w,a) \geq \frac{\Delta t_{\mathcal{M}}(w,a)}{c} - 2$.*

Galil prueba en [17] que todo algoritmo predecible puede ser transformado en un algoritmo de tiempo real. Haremos un fuerte uso de este resultado al probar que χ_{pal} puede ser calculada en tiempo real. Es importante comentar que el trabajo desarrollado por Galil en [16] es esencialmente una simplificacion del trabajo previamente realizado por Slisenko [46] quien probo, a lo largo de 172 densas paginas, que χ_{pal} puede ser calculada en tiempo real, la prueba de Slisenko es dificil y directa (Slisenko exhibe un algoritmo de tiempo real que calcula χ_{pal}), la prueba de Galil es indirecta y mucho mas simple (Galil exhibe un algoritmo predecible que calcula χ_{pal}).

En lo que sigue estudiaremos la prueba del siguiente teorema.

Theorem 24. *Existe un algoritmo predecible que calcula la funcion χ_{pal}.*

De este teorema podemos obtener como corolario la doudecima cota superior para Pal.

Corollary 9. *(doudecima cota superior para Pal)*
Pal puede ser reconocido en tiempo real usando maquinas multicinta.

14 El algoritmo predecible de Galil-Slisenko

Intentaremos mostrar, en esta seccion, que existe una maquina de Turing multicinta de tiempo real que reconoce el lenguaje Pal. No exhibiremos una tal maquina, exhibiremos una maquina de una sola cinta, llamemosla $\mathcal{M}$, con multiples cabezas asociadas a esta unica cinta y tal que:

1. Dado $w = w_1...w_n \in \{0,1\}^*$ y dado $w^* = w_1 c w_2 c....c w_n$ (con $c \notin \{0,1\}$), si $y = \epsilon_1...\epsilon_{2n-1} \in \{0,1\}^*$ es el output de $\mathcal{M}$ en el input w^* entonces dado $i = 2m-1 \leq 2n-1$ se tiene que $\epsilon_i = 1$ si y solo si $w\,[1...m] \in Pal$.
2. $\mathcal{M}$ es una maquina predecible.

Lo primero que debemos comentar es que la existencia de una tal maquina es suficiente para garantizar la existencia de una maquina multicinta de tiempo real que calcule la funcion χ_{pal}. Esto es asi porque:

1. Fischer, Meyer y Rosenberg probaron que una maquina de una sola cinta y k cabezas puede ser simulada en tiempo real por una maquina multicinta standard con $11k-9$ cintas [12]. Posteriormente Leong y Seiferas probaron que $4k-3$ cabezas son suficientes [34].
2. El *blow up* en el tiempo de computo generado por transformar el input w en la palabra w^*, que es dos veces mas larga, puede ser eliminado usando las tecnicas de *linear speed up* ya mencionadas.
3. El teorema de Galil afirma que predictibilidad y tiempo real son conceptos equivalentes.

Sea $w = w_1...w_n \in \{0,1\}^*$ y sea $w^* = u_1...u_{2n-1}$, suponga que $i \lneqq j \lneqq k$ satisface lo siguiente:

1. $j - i = k - j$.
2. La palabra $u\,[i...k]$ es un palindromo.
3. La palabra $u\,[i-1...k+1]$ no es un palindromo.

Suponga ademas que para todo $l \leq j-1$ ya se ha determinado si la posicion l es o no es el centro de un palindromo inicial (esto es: el centro de un segmento inicial de w de longitud impar y que pertenece a Pal). El item 3 implica que k no es el centro de un palindromo inicial, los items 2 y 3 implican que $i-1$ y $k+1$ son posiciones impares. Por otro lado, todo segmento inicial de longitud par no es un palindromo (el caracter inicial pertenece a $\{0,1\}$, mientras que el caracter final es igual a $c \notin \{0,1\}$). Esto es: todos los palindromos iniciales de w^* tienen longitud impar y estan en correspondencia biunivoca con los palindromos iniciales de w (la biyeccion consiste en borrar las c's en los palindormos inciales de w^*).

Suponga ahora que $u = u_{k+1}u_k u_{k-1}...u_{l+1}u_l$ es el palindromo inicial mas largo en la palabra $u_{k+1}u_k u_{k-1}...u_i u_{i-1}$. Dado que $k+1$ e $i-1$ son impares, todo palindromo inicial en $u_{k+1}u_k u_{k-1}...u_i u_{i-1}$ es de longitud impar y por lo tanto tiene centro. Sea $r = \frac{k+1+l}{2}$ el centro de u, note que $j \lneqq r \leq k+1$ ($r = k+1$ si y solo si el unico palindromo inicial de $u_{k+1}u_k u_{k-1}...u_{l+1}u_l$ es u_{k+1} cuyo centro es la posicion $k+1$).

Lemma 16. *Para todo $l \in \{j+1,...,r-1\}$ se tiene que l no puede ser el centro de un palindromo incial de w^*.*

Proof. Suponga que existe $l \in \{j+1, ..., r-1\}$ tal que l es la posicion centro de un palindormo incial de w^*, en este caso la palabra $u_{k+1}u_k...u_{2l-k}u_{2l-k-1}$ seria un palindromo inicial de $u_{k+1}u_ku_{k-1}...u_iu_{i-1}$ mas largo que u, lo cual es una contradiccion.

El lema anterior es el nucleo combinatorio del algoritmo de Galil-Slisenko, el cual pasamos a describir a continuacion.

Una version ingenua del algoritmo Sea w la palabra que necesitamos clasificar, bien sea como palindromo o bien como no palindromo. Sea w^* el input de $\mathcal{M}$. La maquina $\mathcal{M}$ cuenta, para empezar, con al menos tres cabezas que llamaremos izquierda, central y derecha. Sea t un instante durante la computacion de $\mathcal{M}$ en el input w^*, en el instante t las cabezas de $\mathcal{M}$ ocupan posiciones $L \leqq C \leqq R$, satisfaciendose lo siguiente:

1. $C - L = R - C$.
2. La palabra $u[L...R]$ es un palindromo.

La cabeza lectora de $\mathcal{M}$ es la cabeza derecha, cuando esta cabeza ocupa la posicion R, la maquina $\mathcal{M}$ ha calculado los caracteres $y_1, ..., y_R$ del output. En el instante $t+1$ la cabeza izquierda se desplaza una posicion a la izquierda, hasta $L-1$, mientras que la cabeza derecha se desplaza una posicion a la derecha, hasta $R+1$. La maquina $\mathcal{M}$ verifica si los caracteres leidos por estas dos cabezas son iguales, en caso de serlo $\mathcal{M}$ imprime

$$y_{R+1} = \begin{cases} 1 \text{ si } L-1=1, \text{ i.e. la cabeza izquierda ocupa la celda inicial} \\ 0, \text{ en otro caso} \end{cases}$$

Remark 14. Note que si $L-1=1$ la palabra $u_1...u_{R+1}$ es un palindromo inicial.

Remark 15. Para que la verificacion exigida en el paragrafo anterior ($L-1=1$, i.e. la cabeza L ocupa la celda inical) pueda realizarse supondremos que la celda inicial tiene una marca especial.

Supongamos ahora que los caracteres leidos por L y R difieren, tenemos entonces que la posicion ocupada por C no puede ser el centro de un palindromo incial. En este caso la maquina $\mathcal{M}$ simula el algoritmo de Fischer y Paterson [13] en el input $u_{R+1}u_R...u_Lu_{L-1}$. Existe $K \geqq 0$ tal que la simulacion del algoritmo FP (usaremos el simbolo FP para denotar el algoritmo de Fischer y Paterson) por parte de la maquina $\mathcal{M}$, en el input $u_{R+1}u_R...u_Lu_{L-1}$, consta de a lo mas $K(R-L)$ transiciones. Tras a lo mas $K(R-L)$ transiciones $\mathcal{M}$ habra calculado u, el palindromo inicial mas largo contenido en $u_{R+1}u_R...u_Lu_{L-1}$, asi como el centro de u que llamaremos D. Resulta que D es el punto mas cercano a C, por la derecha, que podria ser el centro de un palindromo inicial. La maquina $\mathcal{M}$ deja la cabeza lectora (derecha) en la posicion $R+1$, mueve la cabeza central a la posicion D y mueve a la cabeza izquierda hasta la celda $2D-R-1$.

El procedimiento descrito en los paragrafos anteriores es el nucleo del algoritmo de Galil-Slisenko. Es facil convencerse de que el algoritmo es correcto,

esto es: dado $w^* = u_1...u_{2n-1}$, dado $y_1...y_{2n-1}$ el output de $\mathcal{M}$ en w^* y dado $i \leq 2n-1$, se tiene que $y_i = 1$ si y solo si $u\,[1...i]$ es un palindromo.

Queda pendiente verificar que la maquina $\mathcal{M}$ es predecible. Consideremos tres casos:

1. (match) $u_{L-1} = u_{R+1}$ y $L - 1 \gneqq 1$. En este caso la maquina imprime 0 y su cabeza lectora (la cabeza R) avanza una posicion a la derecha. Note que en este caso $\Delta t\,(u_1...u_{R+1}, u_{R+2}) = 1$.
2. (missmatch) $u_{i-1} \neq u_{k+1}$. En este caso la maquina simula la computacion de FP en el input $u_{R+1}u_R...u_Lu_{L-1}$, encuentra el siguiente centro tentativo y desplaza las cabezas central e izquierda, entonces imprime 0 y las cabezas derecha e izquierda se desplazan una posicion. Note que en este caso $\Delta t\,(u_1...u_{R+1}, u_{R+2}) = O\,(R-L)$.
3. (palindromo) $u_{L-1} = u_{R+1}$ y $L - 1 = 1$. Este, al igual que el caso anterior, puede ser un caso problematico (i.e. un caso en el que $\Delta t\,(u_1...u_{R+1}, u_{R+2})$ no esta acotado por una constante). Podemos considerar este caso como un caso especial del anterior, recuerde que nosotros necesitamos marcar la celda inicial, esto podemos hacerlo usando el alfabeto $\Gamma = \Sigma \cup (\Sigma \times \{0\})$ y definiendo w^* como $(w_1, 0)\,cw_2cw_3c...cw_n$. Note que, de esta manera, siempre que la cabeza L alcanze la celda 1 ocurrira un missmatch, un missmatch especial que la maquina podra reconocer gracias a la componente 0 del caracter que esta leyendo a la izquierda. En este caso, la maquina imprime 1 (esto es lo especial) y realiza el mismo trabajo que en el caso anterior.

La version ingenua del algoritmo de Galil-Slisenko, aqui expuesta, no es un algoritmo predecible. El problema es que en el caso missmatch debemos simular el algoritmo FP, y es posible que el tiempo empleado en esta simulacion no pueda ser acotado linealmente con respecto a la distancia existente entre la posicion R, de la cabeza derecha, y el extremo final del siguiente palindromo inicial (es posible que no se satisfaga la condicion 2 en la definicion de algoritmo predecible). Note por otro lado que las cabezas izquierda y central deben ser desplazadas una distancia menor o igual que la existente entre la cabeza derecha y el extremo final del siguiente palindromo inicial. Tenemos entonces que, el unico problema que presenta la version ingenua del algoritmo de Galil es la necesidad de simular FP en cada missmatch. ¿Que podemos hacer? Quizas no simular FP en cada missmatch e intentar usar la informacion obtenida en las anteriores simulaciones.

Una version mas cuidadosa: rompiendo cadenas. El algoritmo de Galil-Slisenko se basa fuertemente en la nocion de *cadena* (*reticulo cohesivo*) introducida por Slisenko [46]. Las cadenas son los objetos combinatorios que permiten guardar informacion referente a las simulaciones de FP ejecutadas en etapas anteriores de la computacion.

Definition 45. *Dada $w_1...w_n$ una palabra, una semicadena sobre w es una secuencia $H = i_1, ..., i_k$ tal que:*

1. $1 \lneqq i_1 \lneqq i_2 \lneqq ... \lneqq i_k \lneqq n$.

2. *Para todos* $l, r \leq k-1$ *se tiene que* $i_{l+1} - i_l = i_{r+1} - i_r$, *(usaremos el simbolo* $h(H)$ *para denotar la cantidad* $i_2 - i_1$*).*
3. *Para todo* $i \leq k$ *la palabra* $w[i_k - h(H) ... i_k + h(H)]$ *es un palindromo con centro en* i_k*.*

Dada w una palabra y dado $S(w)$ el conjunto de todas las semicadenas sobre w, podemos definir un orden natural sobre $S(w)$ de la siguiente manera:

Dadas $H, K \in S(w)$ diremos que K *refina* a H, (o que H es menor que K), si y solo si $H \subset K$.

Definition 46. *Dada* $CH \in S(w)$ *diremos que* CH *es una cadena sobre* w *si y solo si* CH *es una semicadena maximal en el orden antes definido.*

Dada w una palabra y dado $i \leq |w|$ el simbolo $R_w(i)$ denota la cantidad

$$\max_{k \leq \min(n-i, i)} \{w[i-k...i+k] \text{ es un palindromo}\}$$

Dada $CH = i_1, ..., i_k$, una cadena sobre w, el simbolo CH^+ denota la posicion $i_k + R_w(i_k)$.

Suponga ahora que al procesar el input

$$(w_1, 0)\, cw_2 cw_3 c...cw_n = u_1 ... u_{2n+1}$$

un missmatch ha ocurrido y que el centro del palindromo mas largo incluido en $u_{R+1}u_R...u_C...u_Lu_{L-1}$, llamemoslo D, satisface la desigualdad $D - C \leq \frac{R-C}{4}$.

Lemma 17. *Existe una cadena* CH *sobre* w *tal que:*

1. C *y* D *son nodos adyacentes de la cadena.*
2. *Existen al menos tres nodos de* CH *a cada lado de* C*.*
3. $R \leq CH^+$*.*

Proof. Tenemos que $C \leq D \leq \frac{R-C}{4}$, esto implica que dado L^* el extremo izquierdo del palindromo con centro en D se satisface la desigualdad $L^* \leq L + \frac{C-L}{2}$, y en consecuencia la palabra $u[C...D + (D-C)]$ es un palindromo. Si reflejamos respecto a C, que es el centro de un palindromo de radio $R - C$, obtenemos que las palabras

$$u[C - 2(D-C) ... D + (D-C)] \text{ y } u[C - 2(D-C) ... C]$$

son palindromos. Si reflejamos respecto a D, que es el centro de un palindromo de radio mayor o igual que $3(D-C)$, obtenemos que las palabras de radio $D-C$ y con centro en $C-2(D-C)$, $C-(D-C)$, C, D o $D+(D-C)$ son palindromos identicos. Si reflejamos una vez mas respecto a C obtenemos que las palabras de radio $D - C$ y centros en $C \pm i(D-C)$, con $i \in \{-3, ..., 3\}$, son palindromos identicos. Es facil probar que $CH^+ \geq R$ (siga reflejando de manera alternada en torno a C y D). Tenemos entonces que estos siete centros son nodos de una semicadena que se extiende hasta L por la izquierda y hasta R por la derecha.

Suponga que esta semicadena no es una cadena, en este caso existiria un nodo K de CH tal que $C \lneqq K \lneqq D$ y entonces K seria el centro de un palindromo inicial mas largo que aquel centrado en D (contradiccion). Podemos entonces concluir que los siete nodos $C \pm i\,(D-C)$, con $i \in \{-3,...,3\}$, son nodos sucesivos de una cadena.

Volvamos ahora al algoritmo, suponga que un missmatch ocurre al mover la cabeza izquierda a la posicion $L-1$ y mover la cabeza derecha a la posicion $R+1$. Distinguimos dos casos:

- (*caso 1: cadena*) El centro D, del palindromo mas largo en $u_{L-1}...u_{R+1}$ satisface la desigualdad $D - C \leq \frac{R-C}{4}$.
- (*caso 2: no cadena*) D satisface la desigualdad $D - C \gneqq \frac{R-C}{4}$.

Note que en el caso 2 el extremo derecho, llamemoslo R^*, del siguiente palindromo inicial contenido en w^*, satisface la desigualdad $R^* - R \gneqq 2\left(\frac{R-C}{4}\right)$. Esto implica que

$$\Delta k\,(u_1...u_R, u_{R+1}) \geq 2\left(\frac{R-C}{4}\right)$$

Podemos suponer (aplicando las tecnicas del linear speed up) que el numero de transiciones realizadas por el algoritmo antes de leer el siguiente caracter esta acotado por $2\,(R-C)$. Esto implica que

$$\Delta k\,(u_1...u_R, u_{R+1}) \geq \frac{\Delta t\,(u_1...u_R, u_{R+1})}{4} - 2$$

y por lo tanto, al menos en el caso 2, la condicion de predictibilidad es satisfecha. Tenemos entonces que el caso problematico es el caso 1, dado que en este caso la unica desigualdad que podemos establecer para $\Delta k\,(u_1...u_R, u_{R+1})$ es

$$\Delta k\,(u_1...u_R, u_{R+1}) \geq 2\,(D-C)$$

y $D-C$ podria ser, en este caso, igual a 2; mientras que si nos vemos obligados a simular FP, la unica desigualdad que podemos establecer para $\Delta t\,(u_1...u_R, u_{R+1})$ es

$$\Delta t\,(u_1...u_R, u_{R+1}) \leq 2\,(R-C)$$

y es imposible establecer una cota superior significativa para la cantidad $(R-C)$. Debemos entonces evitar simular FP en esta etapa de la simulacion, pero si simulamos a FP ¿como podemos determinar la posicion a la cual debemos desplazar la cabeza lectora? Dada CH una cadena sobre w, usaremos el simbolo $h\,(CH)$ para denotar la *brecha* de CH que no es otra cosa que la distancia entre los nodos adyacentes de CH. Note que si $CH = i_1, ..., i_k, C, D, i_{k+3}, ..., i_p$ es la cadena de menor brecha entre todas las cadenas que pasan por C y que contienen al menos tres nodos a cada lado de C, entonces D es el centro del palindromo inicial mas largo en $u_{R+1}u_R...u_C...u_{L-1}$. La idea que surge de manera inmediata, (para destrabar nuestro algoritmo), es tener el cuidado de llegar a todo missmatch del tipo 1 con una cadena precomputada, porque en este caso

lo unico que tendriamos que hacer antes de leer el siguiente bit es desplazar la cabeza central a D y la cabeza izquierda a $L + 2(D - C)$, lo cual nos tomaria $2(D - C)$ unidades de tiempo, satisfaciendose entonces que

$$\begin{aligned} \Delta k(u_1...u_R, u_{R+1}) &\geq 2(D - C) \\ &\geq \frac{\Delta t(u_1...u_R, u_{R+1})}{4} - 2 \end{aligned}$$

con lo que nuestro algoritmo estaria en condiciones de satisfacer la condicion de predictibilidad en todos los casos posibles. ¿Como llegar bien preparado a los missmatch de tipo 1? Note que al caer en un missmatch del tipo 2 tenemos suficiente tiempo para simular el algoritmo FP en la palabra $u_{R+1}u_R...u_C...u_{L-1}$.

Definition 47. *Un palindromo doble de brecha k es una palabra $v_1...v_{4k+1}$ tal que $v[1...2k+1]$ y $v[2k+1...4k+1]$ son palindromos identicos.*

Note que si $v = v_1...v_{4k+1}$ es un palindromo doble entonces $k+1, 2k+1, 3k+1$ es una semicadena sobre v.

Remark 16. Dado $v = v_1...v_{4k+1}$ un palindromo doble, diremos que $k+1, 2k+1, 3k+1$ son los tres nodos de v

Lo anterior implica que si

$$CH = i_{-k}, i_{-k+1}, ..., i_{-1}, C, D, i_2, ..., i_p$$

es la cadena de brecha mas corta que pasa por C y que contiene al menos tres nodos a cada lado de C, entonces D, i_2, i_3 son los tres nodos del palindromo doble mas corto que empieza en C.

Lemma 18. *Existe un algoritmo que calcula en tiempo $O(R - C)$, usando el algoritmo FP y cuatro cabezas adicionales, el palindromo doble mas corto con inicio en C.*

Proof. El algoritmo procede por etapas, dado $i = 1, 2, ..$ la etapa i-esima consiste en:

1. Marcamos las celdas en el intervalo $C, ..., C + 2^{i+2}$. Suponemos que en la etapa $i - 1$ las celdas en el intervalo $C, ..., C + 2^{i+1}$ han sido marcadas, para marcar las 2^{i+1} celdas adicionales usamos dos cabezas S y M, la cabeza S se mueve entre C y la ultima celda marcada, mientras que M se mueve a la derecha de la ultima celda marcada en la etapa anterior. Si no podemos marcar todas las 2^{i+1} celdas que deberiamos marcar en esta etapa, (porque la cabeza M alcanza a la cabeza lectora R sin que la cabeza S haya llegado hasta la ultima celda marcada en la etapa anterior), la maquina accede a un estado que indica que esta etapa es final.
2. Sea X_i la palabra marcada en la etapa i, el algoritmo simula el procedimiento FP para calcular los palindromos iniciales de X_i.

3. Una vez han sido marcados los palindromos iniciales de X_i la maquina decide, usando dos cabezas adicionales, si existe un $k \geq 2$ tal que X_i contiene palindromos iniciales de longitudes $2k+1$ y $4k+1$. En caso de existir un tal k, el palindromo inicial que termina en la posicion $C+4k$ es un palindromo doble. Y si k es el menor entero para el cual existen palindromos iniciales de longitudes $2k+1$ y $4k+1$, entonces el palindromo inicial de X_i que termina en la posicion $C+4k$ es el menor palindromo doble que empieza en C. En este caso el algoritmo marca la posicion $C+4k$ y para, en caso contrario pasa a la etapa $i+1$, si la etapa i era final, el algoritmo imprime un mensaje de error.

El palindromo doble mas corto que empieza en C es el germen de la cadena que buscamos calcular, una vez calculado este germen, es facil extender la cadena a derecha e izquierda [16] (mas adelante discutiremos, con un cierto nivel de detalle, el *procedimiento de extension de cadenas* empleado por Galil)

Una *pre-configuracion* del algoritmo esta dada por tres posiciones L, R y C tales que $R-C = C-L \gneqq 0$. Una pre-configuracion es positiva, respecto a $w^* = u_1...u_{2n+1}$, si y solo si $u[L...R]$ es un palindromo; y es negativa en caso contrario. La computacion de nuestro algoritmo en el input w^* procede de pre-configuracion en pre-configuracion. Esta secuencia de pre-configuraciones puede ser factorizada en secuencias de la forma

$$(L_1, C, R_2), ..., (L_k, C, R_k), (L_{k+1}, D, R_k)$$

de modo tal que para todo $i \leq k-1$ se tiene que $L_{i+1} = L_i - 1$ y $R_{i+1} = R_i + 1$, ademas cada una de las pre-configuraciones en esta secuencia es positiva, excepto (L_k, C, R_k) que es negativa. Si $D - C \leq \frac{R_k - C}{4}$ diremos que el factor es de tipo 1 y si $D - C \gneqq \frac{R_k - C}{4}$ diremos que es del tipo 2. Tenemos entonces que la computacion de nuestro algoritmo en el input w^* puede ser vista como una secuencia de factores.

Una configuracion es una tupla $(L, C, R, i_1, i_2, i_3, \epsilon)$ tal que:

1. $R - C = C - L \gneqq 0$.
2. Si $\epsilon = 1$ se tiene que $i_3 - i_2 = i_2 - i_1 \gneqq 0$; $i_1 \gneqq C$; $i_3 \leq R$, la palabra $u[C...2i_3 - i_2]$ es el palindromo doble mas corto con inicio en C contenido en $u[C...R]$ y tal que la semicadena CH definida por este palindromo doble se extiende hasta L a la izquierda y hasta R a la derecha (i.e $CH^+ \geq R$ y $CH^- \leq L$).
3. Si $\epsilon = 0$ no existe un palindromo doble con inicio en C que este contenido en $u[C...L]$ y que pueda ser extendido hasta L y C.

Dado $(L_1, C, R_2), ..., (L_k, C, R_k), (L_{k+1}, D, R_k)$ un factor de la computacion de nuestro algoritmo en el input w^*, este factor puede ser visto como una secuencia

$$\left(L_1, C, R_1, i_1^1, i_2^1, i_3^1, \epsilon_1\right) ... \left(L_{k+1}, D, R_k, i_1^1, i_2^1, i_3^1, \epsilon_{k+1}\right)$$

de configuraciones. Si el factor es de tipo 1 entonces $\epsilon_1 = ... = \epsilon_k = 1$. Si el factor es de tipo 2 existe algun $i \leq k$ tal que $\epsilon_i = 0$. Lo importante entonces

es que podemos llegar preparados, con una cadena precomputada, a todo missmatch de tipo 1. No siempre podremos llegar preparados, i.e. con una cadena precomputada, hasta un missmatch de tipo 2, pero en estos casos tendremos suficiente tiempo de calcular el germen de una cadena centrada en D, que es el centro de la nueva pre-configuracion. Lo unico que queda por resolver es lo siguiente: sea $(L, C, R, i_1, i_2, i_3, 1)$ la configuracion en el instante t ¿como decidir, eficientemente, si dicha cadena puede o no ser extendida hasta $R+1$ y $L-1$? Galil utiliza para ello el siguiente *procedimiento de extension de cadenas*:

El procedimiento emplea tres cabezas adicionales, llamemoslas D_1, D_2 y D_3. La cabeza D_1 se mueve de izquierda a derecha y de derecha a izquierda entre los nodos (convenientemente marcados) C e i_1, mientras que la cabeza D_2 va avanzando a la derecha, empezando en la posicion C, comparando los caracteres leidos por ella y los leidos por D_1. Note que la palabra $u[C...i_1]$ es un semiperiodo de la palabra $u[C...R]$, es decir: dado que $\epsilon = 1$, la cadena se extiende hasta R y L, y entonces las palabras $u[C...R]$ y $u[L...R]$ son iguales a $v_1 v_2 ... v_k x$ y $x^R v_k^R ... v_1^R$ respectivamente, donde

$$v_i = \begin{cases} u[C...i_1] & \text{si } i \text{ es impar} \\ u[i_1 + 1...C - 1] & \text{si } i \text{ es par} \end{cases}$$

Adicionalmente se tiene que x es o un segmento inicial de $u[C...i_1]$, si k es par; o un segmento incial de $u[i_1 + 1...C - 1]$, si k es impar. Las cabezas D_1 y D_2 se encargan entonces de extender la cadena a la derecha, podemos usar simultaneamente (y de manera similar) las cabezas D_1 y D_3 para extender la cadena a la izquierda.

Continuando con la descripcion del algoritmo tenemos que al desplazar las cabezas izquierda y derecha a las posiciones $L-1$ y $R+1$, la maquina simplemente verifica que $u_{L-1} = u_{D \pm 1} = u_{R+1}$ ($+$ si k es par, $-$ si k es impar), si este es el caso la nueva configuracion es igual a $(L-1, C, R+1, i_1, i_2, i_3, 1)$, en caso contrario la nueva configuracion es igual a $(L-1, C, R+1, i_1, i_2, i_3, 0)$, y la maquina avanza hacia un missmatch de tipo 2 por lo que puede olvidar la cadena, esperar a llegar a este missmatch y simular entonces FP para calcular el palindromo mas largo contenido en $u[L^*...R^*]$ llamemoslo P, y calcular entonces el palindromo doble mas pequeño con inicio en el centro de P.

Lo anterior es una descripcion esquematica (aunque completa) del algoritmo de Galil-Slisenko, el lector interesado en detalles adicionales puede consultar las referencias [46], [16] y [47].

Remark 17. La maquina de Galil usa 6 cabezas L, C, R, D_1, D_2 y D_3 ademas de las 6 cabezas empleadas para simular el algoritmo FP (Slisenko [47] exhibe una maquina de Turing de 6 cabezas que capaz de simular el algoritmo FP en tiempo lineal) y las 4 cabezas adicionales empleadas para calcular palindromos dobles. Convertir un algoritmo predecible en un algoritmo de tiempo-real implica introducir una cabeza adicional, lo que nos da un total de 17 cabezas, el teorema de Leong-Seiferas antes citado nos permite convertir esta maquina de Galil en una maquina con 65 cintas. ¿Cual es el minimo numero de cintas que deben usarse para reconocer Pal en tiempo real?

El teorma de Galil nos permite probar la recognizabilidad en tiempo real de muchos otros lenguajes. Considere el lenguaje Pal^* : ¿es posible reconocer en tiempo real el lenguaje Pal^*? La respuesta es SI, la recognizabilidad en tiempo real del lenguaje Pal^* es un corolario del teorema de Galil y del lema de Pratt que enunciamos a continuacion.

Lemma 19. *(lema de Pratt)*
Sea w un elemento de Σ^*. Si $w = uv$ y $u, w \in Pal^*$ se tiene que $v \in Pal^*$.

Para una prueba el lector puede consultar la referencia [30].

Theorem 25. *El lenguaje Pal^* puede ser reconocido en tiempo real.*

Proof. Podemos pensar en el algoritmo de Galil como en un algoritmo que con input $w_1...w_n$ calcula la palabra $x_1...x_n$ definida por

$$x_i = \begin{cases} 1 \text{ si } w_1...w_i \in Pal \\ 0, \text{ en caso contrario} \end{cases}$$

Sea $\mathcal{M}$ el algoritmo definido por:
Con input $w_1...w_n$ el algoritmo $\mathcal{M}$ empieza simulando al algoritmo de Galil en el input $w_1...w_n$. Cada vez que el algoritmo de Galil imprime un 1, el algoritmo $\mathcal{M}$ interrumpe la simulacion e inicia una nueva simulacion del algoritmo de Galil en el sufijo de $w_1...w_n$ constituido por los caracteres que no han sido leidos hasta el momento.
La validez del algoritmo $\mathcal{M}$ es consecuencia del lema de Pratt, que el algoritmo $\mathcal{M}$ sea un algoritmo de tiempo real es consecuencia del teorema de Galil.

15 Maquinas de Turing no deterministicas

Una maquina de Turing no deterministica es una maquina de Turing cuyo relacion de transicion no es funcional. Las maquinas de Turing no deterministicas, como las maquinas no deterministicas estudiadas en capitulos anteriores, son capaces de realizar adivinanzas afortunadas. Es importante anotar que las maquinas no deterministicas son un modelo de computacion esencialmente teorico, (con esto queremos decir que no es un modelo suceptible de ser implementado en la practica de manera eficiente). Toda maquina de Turing no deterministica puede ser simulada por una maquina de Turing deterministica pero las simulaciones conocidas tienen un costo exponencial: dada $\mathcal{M}$ una maquina de Turing no deterministica cuyo tiempo de computo es igual a la funcion $T_{\mathcal{M}}$, existe una maquina de Turing deterministica $\mathcal{N}$ tal que $L(\mathcal{M}) = L(\mathcal{N})$ y $T_{\mathcal{N}} \in \Omega\left(2^{T_{\mathcal{M}}}\right)$.

La pregunta: ¿podemos simular toda maquina de Turing no deterministica con un sobrecosto a lo mas polinomial? Es el famoso problema: ¿es P igual a NP? La hipotesis de la gran mayoria de los cientificos de la computacion es que P es diferente de NP, (i.e. la mayoria de los investigadores creen que las maquinas de Turing no deterministicas son un contraejemplo (debil) a la *tesis*

fuerte de Church). Es de esperar que el poder de realizar adivinanzas afortunadas incremente la eficiencia de nuestras maquinas.

Intentaremos ilustrar el poder del no-determinismo, (al menos en el diseño de algoritmos), apelando nuevamente a nuestro *leit-motiv*, el lenguaje Pal.

Diseñar una maquina con muchas cintas que reconozca Pal en tiempo real es posible (Galil), pero esta muy lejos de ser facil. Reconocer Pal en tiempo real usando una maquina deterministica con solo dos cintas parece no ser posible

Conjecture 1. No existe una maquina de tiempo real con solo dos cintas que reconozca el lenguaje Pal.

Por otro lado resulta muy facil diseñar una maquina de Turing no deterministica provista unicamente con dos cintas y que reconozca Pal en tiempo real. Note que la dificultad para reconocer Pal en tiempo real radica en que para reconocer el caracter central de un input es necesario recorrer el input hasta el ultimo caracter y a continuacion realizar algunos computos adicionales. Por otro lado si nuestra maquina tiene la capacidad de realizar adivinanzas afortunadas, podemos pedirle que durante la lectura del input, adivine cual es el caracter central y use esta informacion (calculada de manera no deterministica) para decidir en tiempo real si el input dado es o no un palindromo.

Sea $\Sigma = \{0, 1\}$ y sea $\mathcal{M}_1 = (Q, q_0, \Gamma, F, \delta)$ la maquina de Turing no deterministica, con alfabeto de entrada Σ, definida por:

1. $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
2. $\Gamma = \Sigma$.
3. $F = \{q_4\}$.
4. δ es la relacion de transicion definida por:
   Al iniciar la computacion la maquina se encuentra en el estado q_0, al leer el primer caracter la maquina pasa al estado q_1 y escribe este caracter en la segunda cinta. El estado q_1 indica que la maquina aun no ha pasado por el caracter central, cuando la maquina esta en el estado q_1 sus dos cabezas se mueven a la derecha en cada transicion y en cada transicion la maquina copia el ultimo caracter leido en la segunda cinta. La maquina puede pasar no deterministicamente del estado q_1 al estado q_2 o al estado q_3. Pasar al estado q_2 indica que la maquina cree que el input es una palabra de longitud par, y que su cabeza lectora acaba de leer el ultimo caracter de la primera mitad, al pasar al estado q_2 la maquina sigue desplazando a la derecha su cabeza lectora (la cabeza correspondiente a la primera cinta, esto es a la cinta de entrada) pero empieza a desplazar su segunda cabeza de derecha a izquierda. Estando en el estado q_2, cada vez que la maquina lee un caracter en la cinta de entrada lo compara con el caracter que esta leyendo en la segunda cinta, si coinciden la maquina borra el caracter de la segunda cinta desplaza la cabeza correspondiente a la izquierda, y desplaza la cabeza lectora a la derecha. Si por el contrario los caracteres no coinciden la maquina pasa al estado q_5 para la computacion y rechaza el input. Si estando en el estado q_2 la maquina llega al extremo derecho del input, hace lo siguiente: si la segunda cinta esta vacia, pasa a q_4 y acepta, en caso contrario pasa a q_5 y rechaza. El estado

q_3 corresponde a que la maquina cree que el input tiene longitud impar y que en ese instante de la computacion su cabeza lectora se encuentra justo encima del caracter central, la manera en que la computacion continua una vez la maquina ha accedido al estado q_3 es similar al caso en que la maquina accede al estado q_2.

La anterior es una descripcion suficientemente detallada de una maquina de Turing no determininistica (la maquina $\mathcal{M}_1$). Es facil verificar que $\mathcal{M}_1$ reconoce el lenguaje Pal y que su tiempo de computo al procesar inputs de tamaño n es exactamente n, (i.e. la maquina $\mathcal{M}_1$ es una maquina de tiempo real). Note por otro lado que la maquina $\mathcal{M}_1$ no es otra cosa que el automata de pila no deterministico que usamos en el capitulo 3 para reconocer palindromos en tiempo real (los automatas de pila no deterministicos son maquinas de Turing de dos cintas).

Algunos lectores podran pensar que la conjetura **??** es trivialmente falsa, dado que si es posible reconocer Pal en tiempo real con un cierto numero de cintas, es entonces posible reconocer Pal con dos cintas. El lector no debe despreciar el poder de computo que una o varias cintas adicionales pueden aportar. Sea $i \geq 1$ y defina $\mathcal{RT}(i)$ como la coleccion de todos los lenguajes que pueden ser reconocidos en tiempo real usando una maquina de Turing con i cintas. Sea ahora $\Sigma = \{0, 1, a, b, \alpha, \beta\}$ y sean L_1 y L_2 los lenguajes definidos por

$$L_1 = \left\{uv\alpha\overline{u} : u \in \{0,1\}^* \text{ y } v \in \{a,b\}^*\right\}$$
$$L_2 = \left\{uv\beta\overline{v} : u \in \{0,1\}^* \text{ y } v \in \{a,b\}^*\right\}$$

Rabin [41] prueba que el lenguaje $L = L_1 \cup L_2$ pertenece a $\mathcal{RT}(3) - \mathcal{RT}(2)$ ¡tres cintas son mas poderosas que dos!

Dado $n \geq 1$ podemos definir un lenguaje L_n de la siguiente manera

$$L_n = \left\{u_1 c u_2 c ... u_n c v : u_1, ..., u_n, v \in \{0,1\}^* \ \& \ \exists i \leq n \left(u_i = \overline{v}\right)\right\}$$

Hartmanis y Stearns [25] probaron que $L_n \in \mathcal{RT}(n+1) - \mathcal{RT}(n)$, esto es: cada cinta adicional puede aportar poder de computo. Note por otro lado que cada uno de los lenguajes L_n puede ser reconocido en tiempo real con dos cintas y no determinismo: dos cintas y no determinismo pueden ser mas poderosas que n cintas.

La situacion puede ser incluso aun mas dramatica, considere el lenguaje

$$\Omega = \left\{u_1 c u_2 c ... u_n c v : n \geq 1 \ \& \ u_1, ..., u_n, v \in \{0,1\}^* \ \& \ \exists i \leq n \left(u_i = \overline{v}\right)\right\}$$

Hartmanis y Stearns [25] probaron que Ω no puede ser reconocido en tiempo real con un numero finito de cintas, pero es claro que Ω puede ser reconocido en tiempo real no-deterministico usando dos cintas, esto es: existen lenguajes que no pueden ser resueltos en tiempo real usando un numero finito de cintas pero que pueden ser nodeterministicamente resueltos en tiempo real con solo dos

cintas. Becvar [5], en un survey ya clasico, se preguntaba si es posible reconocer Ω en tiempo real usando una maquina con una sola cinta y varias cabezas, hoy sabemos que esto no es posible (es una consecuencia del teorema de Leong-Seiferas [34]): dos cintas y no determinismo pueden ser mas poderosas que un numero ilimitado de cabezas[5].

Es importante señalar que dado $n \geq 1$ es suficiente usar n pilas para reconocer L_n en tiempo real: $n+1$ pilas pueden ser mas poderosas que n cintas. Por otro lado es posible diseñar una maquina de Turing no deterministica con una cinta de solo lectura y una pila que reconozca Ω : una pila y no determinismo pueden ser mas poderosos que un numero finito de cintas. Lo anterior implica que

Corollary 10. *Existe un lenguaje libre de contexto que no puede ser reconocido en tiempo real con un numero finito de pilas.*

Proof. Ω es libre de contexto dado que Ω puede ser reconocido en tiempo real usando no determinismo y una pila, i.e. con un automata de pila no deterministico.

Lemma 20. *Todo lenguaje libre de contexto puede ser reconocido en tiempo real usando una maquina nodeterministica con dos cintas.*

Proof. Todo automata de pila no deterministico es una maquina de Turing no deterministica con dos cintas: la cinta de entrada y la pila. Por otro lado todo automata de pila no deterministico puede ser transformado en un automata de pila no deterministico de tiempo real: eliminando las transiciones en vacio.

Problem 1. (*enunciados nuevos para problemas viejos*) De la prueba del teorema de Galil es posible establecer que $Pal \in \mathcal{RT}(65)$. ¿Cual es la posicion exacta del lenguaje Pal en la jerarquia $\mathcal{RT}$? O, lo que es lo mismo, determine el unico valor de i para el cual $Pal \in \mathcal{RT}(i+1) - \mathcal{RT}(i)$.

16 Ejercicios capitulo 2

1. Pruebe que dada $\mathcal{M}$ una maquina con k-cintas, existe $\mathcal{M}^*$ una maquina de una sola cinta que simula $\mathcal{M}$ y tal que $T_{\mathcal{M}^*}(n) \in O\left(T_{\mathcal{M}}(n)^2\right)$.
2. Pruebe el teorema de predictibilidad de Galil (consulte la referencia [17]).
3. Complete los detalles en la prueba del teorema de Galil-Slisenko.
4. Pruebe el lema de Pratt.

[5] Es posible definir una jeraquia similar a la jerarquia $\mathcal{RT}$ antes definida, usando como parametro el numero de cabezas. Sea $\mathcal{C}$ tal jerarquia, Aanderaa [1] prueba que esta jerararquia es estricta, el teorema de Seiferas implica que $\mathcal{RT} = \mathcal{C}$.

Problemas relacionados y aplicaciones

La teoria de la computabilidad y la teoria de la complejidad han focalizado sus analisis en un tipo especial de problemas: los problemas de decision. Existen otros tipos de problemas como por ejemplo los problemas de conteo, optimizacion y enumeracion.

En este capitulo estudiaremos problemas, relacionados con palindromos, de cada uno de tipos mencionados anteriormente.

17 Enumeracion de palindromos: el algoritmo de Gusfield

En esta seccion estudiaremos un problema de listado (o enumeracion) relacionado con palindromos. Considere el problema:

Problem 2. (List-Pal : listado de palindromos)

- *Input: w, donde w es una palabra.*
- *Problema: liste todos los subpalindromos de w.*

Sea $w = 1^n$, note que el conjunto de palindromos contenidos en w es el conjunto

$$\left\{1^k : k \leq n\right\}$$

Una lista exhaustiva de los elementos de este conjunto requiere $\Omega\left(n^2\right)$ bits. Si lo que queremos es exhibir un algoritmo de tiempo lineal que resuelva el problema *List-Pal* debemos definir una manera suscinta de presentar la lista de los palindromos contenidos en una palabra dada. Sea $w = w_1...w_n$ una palabra y sea $i \leq n$, un subpalidromo de longitud impar contenido en w y centrado en i es un factor de w de la forma

$$w_{i-r}...\mathbf{w}_i...w_{i+r}$$

Un subpalidromo de longitud par contenido en w y centrado en i es un factor de w de la forma

$$w_{i-r}...\mathbf{w}_{i-1}\mid \mathbf{w}_i...w_{i+r-1}$$

Todo subpalindromo de w es un palindromo centrado en alguna posicion $i \leq n$. Un palindromo maximal en w es un subpalidromo $w\left[i...k\right]$ tal que o $i = 1$ o $k = n$ o el factor $w\left[i-1...k+1\right]$ no es un palindromo. Si u es un subpalindromo no maximal centrado en i existe un palindromo maximal centrado en i que lo contiene. Por otro lado si $w\left[i...k\right]$ es un palindromo maximal centrado en s, para todo $r \leq \frac{k-i+1}{2}$ el factor $w\left[i+r...k-r\right]$ es un subpalindromo centrado en s. Todo lo anterior indica que una manera suscinta de describir el conjunto de

los palindromos contenidos en w es mediante los vectores $OPAL_w$ y $EPAL_w$ definidos por:

$$OPAL_w[i] = r_i \text{ si y solo si } w[i - r_i ... i + r_i] \text{ es el palindromo maximal de longitud impar centrado en } i$$

y

$$EPAL_w[i] = r_i \text{ si y solo si } w[i - r_i ... i + r_i - 1] \text{ es el palindromo maximal de longitud par centrado en } i$$

Sea $w = w_1 w_2 ... w_n$ una palabra y definamos

$$w^{(2)} = w_1 w_1 w_2 w_2 ... w_n w_n$$

Note que los subpalindromos de w estan en correspondencia biunivoca con los subpalindromos de longitud par de la palabra $w^{(2)}$. Lo anterior nos permite reducir el problema de listar todos los subpalindromos maximales de una palabra dada al problema consistente en listar todos los subpalindromos maximales de longitud par. Fijemos un alfabeto Σ y considere el problema

Problem 3. (List-MaxPal : listado de palindromos maximales)

- *Input:* w, *donde* $w \in \Sigma^*$.
- *Problema: calcule el vector* $EPAL_w$.

En lo que sigue estudiaremos un algoritmo de tiempo lineal, debido a D. Gusfield [22], que resuelve el problema *List-MaxPal*. El algoritmo se basa fuertemente en la tecnica de arboles de sufijos (Suffix trees)

17.1 Un metodo: Suffix trees

El metodo de arboles de sufijos es un metodo general que puede ser usado, y ha sido usado [22], para diseñar algoritmos de tiempo lineal para una amplia variedad de problemas en stingologia. Fijemos (como es costumbre) un alfabeto finito Σ y sea w un elemento de Σ^*. Es posibe asignarle a w uno o mas arboles de sufijos, los arboles de sufijos de w, exhiben propiedades estructurales de w que no son evidentes en lsu presentacion estandard como una cadena lineal de simbolos. Los arboles de sufijos fueron introducidos por Weiner [52] con el nombre de arboles de posicion (*position trees*).

Definition 48. *Dada* $w \in \Sigma^*$ *un arbol de sufijos para* w *es un arbol dirigido y con raiz, digamos* (T, r), *tal que:*

- (T, r) *tiene exactamente* $|w|$ *hojas etiquetadas con los numeros de* 1 *a* $|w|$.
- *Todo nodo interior diferente de la raiz tiene al menos dos hijos.*
- *Cada arista de* (T, r) *tiene por etiqueta un elemento de* Σ^*.

- *Dos aristas con el mismo origen tiene etiquetas cuyos caracteres iniciales son diferentes.*
- *Para todo* $i \in \{1, ..., |w|\}$ *se tiene que* γ_i *es igual* $w[i...|w|]$, *donde* γ_i *es la palabra obtenida al concatenar las etiquetas de las aristas que ocurren en el camino que va de* r *a* i.

Dada $w \in \Sigma^*$ usaremos el simbolo $\mathcal{ST}(w)$ para denotar un arbol de sufijos de w. Dados $1 \leq i \lneqq j \leq |w|$ usaremos el simbolo $w[i...j]$ para denotar el factor $w_i...w_j$.

Definition 49. *Dada* $w \in \Sigma^*$ *dado* $\mathcal{ST}(w)$ *un arbol de sufijos de* w *y dado* v *un nodo de* $\mathcal{ST}(w)$, *la profundidad de* v, *que denotaremos con el simbolo* $d_{\mathcal{ST}(w)}(v)$, *es igual a* $|\gamma_v|$, *donde* γ_v *es la la palabra obtenida al concatenar las etiquetas de las aristas que ocurren en el camino que va de* r *a* v.

Remark 18. Dada $w \in \Sigma^*$ no podemos afirmar apriori que exista un arbol de sufijos para w. Si existen dos sufijos u, v de w tales que u es un prefijo propio de v, entonces w no posee un arbol de sufijos. Si por el contrario no existen tales sufijos la palabra w si posee arboles de sufijos.

Podemos resolver el problema de que no toda palabra tenga un arbol de sufijos ecribiendo al final de cada palabra un marcador de fin de cadena, i.e. dada $w \in \Sigma^*$ podemos escribir a w como $w\#$, donde $\# \notin \Sigma$, de esta manera garantizamos que todas las palabras (que nos interesan) tengan un arbol de sufijos.

Existen diferentes algoritmos de tiempo lineal para calcular arboles de sufijos. Uno de ellos es el algoritmo de Ukkonen [51], el cual dada $w \in \Sigma^*$ permite calcular online y en tiempo lineal un arbol de sufijos para w.

Definition 50. *Dadas* u *y* v *dos elementos de* Σ^* *de longitud* n, *un arbol generalizado de sufijos para el par* (u, v) *es un arbol de sufijos para la palabra* $u\$v\#$ *donde* $\$, \# \notin \Sigma$.

Dadas $u, v \in \Sigma^*$, usaremos el simbolo $\mathcal{GST}(u, v)$ para denotar un arbol generalizado de sufijos para el par (u, v). Dadas $u, v \in \Sigma^*$ dos palabras de longitud n podemos usar el algoritmo de Ukkonen para calcular online en tiempo $O(n)$ un arbol generalizado de sufijos para el par (u, v).

El problema del ancestro comun mas cercano. Dado (T, r) un arbol con raiz y dado v un nodo de T, un *ancestro* de v es cualquier nodo de T que ocurra en el unico camino que va de r a v.

Definition 51. *Dados* u, v *dos nodos del arbol* (T, r) *el ancestro comun mas cercano es el ancestro comun a* v *y* w *que esta mas alejado de la raiz.*

El problema que definiremos a continuacion es de gran importancia practica en la teoria de bases de datos y en algoritmos de analisis de textos.

Problem 4. (LCA, Lowest common ancestor).

- *Input:* (W, u, v), *donde* W es *un arbol con raiz* y u, v *son nodos de* W.
- *Problema: calcule el ancestro comun mas cercano de* v *y* w.

Harel y Tarjan [24] muestran que existe un algoritmo $\mathcal{M}$ el cual, con input W, calcula en tiempo lineal un nuevo codigo del mismo arbol, llamemoslo W^*, de manera tal que toda LCA-consulta del tipo ¿$(W, *, *) \in LCA$? pueda ser resuelta en tiempo constante (constante independiente del arbol W y de su tamaño).

El algoritmo de Harel y Tarjan es un ingrediente esencial del algoritmo de Gusfield que estudiaremos mas adelante, antes de presentar el algoritmo de Gusfield debemos estudiar un problema (y un algoritmo) adicional.

El problema de la extension comun mas larga.

Definition 52. *Sean* u *y* v *dos palabras de longitud* n, *dados* $1 \leq i, j \leq n$ *se define* $lcp_{u,v}(i,j)$ *de la siguiente manera*

$$lcp_{u,v}(i,j) = \max_k \{u[i...i+k-1] = v[i...i+k-1]\}$$

Considere el problema definido a continuacion

Problem 5. (LCE, calculo de la extension comun mas larga)

- *Input:* (w, u, i, j), *donde* $w, u \in \Sigma^*$, $i \leq |w|$ *y* $j \leq |u|$.
- *Problema: calcule* $lcp_{u,v}(i,j)$.

Lo que estudiaremos en esta seccion es un algoritmo que con input (w, u), donde w y u son palabras de longitud n, procesa en tiempo lineal este input de manera tal que toda LCE-consulta pueda ser resuelta en tiempo constante (constante independiente de w, u y del tamaño de w). Sea $\mathcal{M}$ el algoritmo definido a continuacion:

con input (u, v) el algoritmo $\mathcal{M}$ hace lo siguiente.

1. Usando el algoritmo de Ukkonen $\mathcal{M}$ calcula en tiempo lineal $\mathcal{GST}(u, v)$.
2. Usando el algoritmo de Harel-Tarjan, el algoritmo $\mathcal{M}$ preprocesa el arbol $\mathcal{GST}(u, v)$ de manera tal que toda LCA-consulta sobre $\mathcal{GST}(u, v)$ pueda ser resulta en tiempo lineal, durante esta etapa de preprocesamiento de $\mathcal{GST}(u, v)$ el algoritmo $\mathcal{M}$ calcula $d(v)$ para todo v nodo de $\mathcal{GST}(u, v)$.

Dadas $u, v \in \Sigma^n$ y dados $i, j \leq n$ existen hojas k_i y r_j en $\mathcal{GST}(u, v)$ tales que k_i esta asociada al sufijo $u[i...n]$ de u y r_j esta asociada el sufijo $v[j...n]$ de v.

Lemma 21. $lcp_{u,v}(i,j)$ *es igual a* $d(v)$, *donde* v *es el ancestro comun mas cercano a* k_i *y* r_j.

Proof. Sea $s_0 = r, s_1, s_2, ..., s_p, k_i = s_{p+1}$ el camino que va de r a la hoja k_i y sea $r_0 = r, t_1, t_2, ..., t_q, r_j = r_{q+1}$ el camino que va de r a la hoja r_j. Supongamos que $u[i...n] = \epsilon_1...\epsilon_{p+1}$, $v[j...n] = \delta_1...\delta_{q+1}$ y que ϵ_l y δ_m son las etiquetas de las aristas $s_{l-1} \rightarrow s_l$ y $t_{m-1} \rightarrow t_m$ (respectivamente), supongamos ademas que $s_f = t_f$ es el ancestro comun mas cercano de las hojas k_i y r_j. Es claro que $\epsilon_1...\epsilon_f = \delta_1...\delta_f$ es un prefijo comun de $u[i...n] = \epsilon_1...\epsilon_{p+1}$ y $v[j...n] = \delta_1...\delta_{q+1}$, note que $d(s_f) = |\epsilon_1...\epsilon_f|$, nos falta verificar que $\epsilon_1...\epsilon_f$ es el prefijo comun mas largo, de la definicion de arbol de sufijos tenemos que $(\epsilon_{f+1})_1 \neq (\delta_{f+1})_1$, la inecuacion anterior implica que $\epsilon_1...\epsilon_f$ si es el prefijo comun mas largo de $u[i...n]$ y $v[j...n]$, y tambien implica que $lcp_{u,v}(i,j)$ es igual a $d(s_f)$.

Corollary 11. *Existe un algoritmo de tiempo lineal que con input (u,v) calcula una estructura $\mathcal{T}_{uv}$ con la ayuda de la cual toda LCE-consulta de la forma ¿$lcp_{u,v}(i,j) = *$? puede ser resuelta en tiempo constante.*

El algoritmo de Gusfield En esta seccion presentaremos el algoritmo de Gusfield. Sea Σ un alfabeto fijo y sea $w \in \Sigma^*$.

Remark 19. Note que si en la definicion 52 tomamos $u = w$ y $v = \overline{w}$ se tiene que

$$lcp_{w,\overline{w}}(i,j) = \max_k \left\{ \begin{array}{c} w[i...i+k-1] = \overline{w}[j...j+k-1] \\ = w[n-(j+k-1)+1...n-j+1] \end{array} \right\}$$

Definition 53. *Dado $i \leq n$ un palindromo de longitud impar centrado en i es un factor simetrico (i.e. el factor es un palindromo) de w de la forma $w_{i-k}...\mathbf{w}_i...w_{i+k}$. Analogamente, un palindromo de longitud par centrado en i es un factor simetrico de w de la forma $w_{i-k}...\mathbf{w}_i|\ \mathbf{w}_{i+1}...w_{i+k-1}$.*

Lemma 22. *Sea $n = |w|$ y sea $1 \leq k \leq n$, note que:*

1. *El palindromo maximal de longitud impar centrado en k tiene longitud $2l-1$ con $l = lcp_{w,\overline{w}}(k, n-k+1)$.*
2. *El palindromo maximal de longitud par centrado en k tiene longitud $2l$ con $l = lcp_{w,\overline{w}}(k, n-k+2)$.*

Proof. Probaremos el item 1, la prueba del item 2 es analoga y por ello se omite. Sea $w_{k-s}...\mathbf{w}_k...w_{k+s}$ el palindromo maximal de longitud impar centrado en k, note que si $r \gneqq s$ se tiene que w

$$w[k...k+1+r-1] \neq w[k...k-1-r+1]$$

Se tiene entonces que

$$1 + s = \max_r \{w[k...k+r-1] = w[k...k-r+1]\}$$

De lo anterior tenemos que

$$2s + 1 = 2 \max_r \{w[k...k+r-1] = \overline{w}[n-k+1...n-k+1+(r-1)]\} - 1$$

Corollary 12. *Dada $w \in \Sigma^*$ y dado $i \leq |w|$ el radio del palindromo mas largo centrado en i (si existe) es igual a $lcp_{w,\overline{w}}(i, n-i+2)$.*

Corollary 13. *Dada $w \in \Sigma^*$ el vector $EPAL_w$ puede ser calculado en tiempo $O(|w|)$.*

Proof. La prueba es el algoritmo de Gusfield. Sea $\mathcal{G}$ el algoritmo definido por:
Con input w el algoritmo $\mathcal{G}$ hace lo siguiente:

1. Calcula $\overline{w}$.
2. Calcula $\mathcal{GST}(w, \overline{w})$.
3. Preprocesa el arbol $\mathcal{GST}(w, \overline{w})$ de manera tal que toda LCA-consulta referente a $\mathcal{GST}(w, \overline{w})$ pueda ser resuelta en tiempo constante (cosntante independiente de w y de su tamaño).
4. Dado $2 \leq i \leq |w|$ el algoritmo $\mathcal{G}$ calcula en tiempo constante

$$d\left(LCA_{\mathcal{GST}(w,\overline{w})}(i, |w| - i + 2)\right)$$

Se sigue de todo lo anterior que el algoritmo $\mathcal{G}$ es un algoritmo de tiempo lineal que calcula de manera correcta el vector $EPAL_w$.

18 Conteo de palindromos: combinatoria de palabras

Los problemas de conteo constutiyen una clase importante de problemas. Una robusta teoria de la complejidad de problemas de conteo ha sido desarrollada desde finalesde los años 70 [39], [53]. En este capitulo estudiaremos un problema de conteo relacionado con palindromos.

La *combinatoria sobre (de) palabras* es una teoria matematica que ha experimentado un amplio desarrollo en los ultimos 30 años, una referencia basica en el area son los trabajos del grupo Lothaire [36]. En combinatoria sobre palabras se han estudiado diferentes nociones de complejidad (para cadenas finitas) algunas de ellas relacionadas con palindromos.

Note que toda palabra w contiene a lo mas $|w| + 1$ palindromos distintos (incluyendo el palindromo vacio), note tambien que para todo $n \geq 1$ la palabra 0^n contiene $n + 1$ palindromos distintos.

Definition 54. *Una palabra finita w se dice rica (compleja) [18] si y solo si el numero de palindromos distintos que ocurren como factores de w es igual a $|w| + 1$.*

Considere el siguiente problema.

Problem 6. ($RICH$: Reconocimento de palabras ricas)

- *Input: w, donde w es una palabra finita en un alfabeto fijo Σ.*
- *problema: decida si w es rica.*

¿Es posible resolver el problema $RICH$ en tiempo lineal? Una posible manera de resolver el poblema $RICH$ en tiempo lineal consiste en resolver, en tiempo lineal, el problema de conteo definido a continuacion:

Problem 7. ($\#DIST$: Conteo de palindromos distintos)

- *Input: w, donde w es una palabra finita en un alfabeto fijo Σ.*
- *problema: calcule el numero de palindromos distintos contenidos en w.*

En esta seccion estudiaremos un algoritmo de tiempo lineal que resuelve el problema $\#DIST$, el algoritmo que estudiaremos fue descubierto recientemente por Groult, Prieur y Richomme [20].

En lo que sigue fijaremos un alfabeto finito Σ. Dado $w = w_1...w_n$ un elemento de Σ^*, un sufijo palindromico es un factor $w[i...n]$ tal que $w[i...n] = w[n...i]$. Si $w[i...n]$ es el mas largo de todos los sufijos palindromicos de w, diremos que $w[i...n]$ es el LPS de w. Diremos que un sufijo palindromico $w[i...n]$ es *uniocurrente* en w si y solo si no existe algun otro factor de w, llamemoslo x, tal que $x = w[i...n]$. El algoritmo de Groult et al se basa en la siguiente observacion clave.

Lemma 23. *El numero de palindromos distintos contenidos en w es igual al numero de prefijos de w cuyo LPS es uniocurrente.*

Proof. Sea u un palindromo contenido en w, y sea $w[i...j]$ la primera ocurrencia de u en w. Note que $w[i...j]$ es un sufijo palindromico del prefijo $w[1...j]$. Por ser $w[i...j]$ la primera ocurrencia de u en w, el sufijo $w[i...j]$ es uniocurrente en $w[1...j]$.

Suponga que $w[i...j]$ no es el LPS de $w[1...j]$, en este caso existe $k \lneqq i$ tal que $w[k...j]$ es un palindromo. Como $w[i...j]$ es un sufijo de $w[k...j]$, su reverso es un prefijo de $w[k...j]$, pero como $u = w[i...j]$ es un palindromo, u es un prefijo de $w[k...j]$. Tenemos entonces que $w[k...k+j-i]$ es igual a u. Por otro lado se tiene que $k + (j - i) \lneqq j$, esto es: el sufijo u ocurre al menos dos veces en $w[1...j]$. De lo anterior se tiene que es posible inyectar el conjunto de palindromos distintos que ocurren en w, en el conjunto de prefijos de w cuyo LPS es uniocurrente. Por otro lado es claro que esta inyeccion puede ser invertida, por lo que podemos afirmar que existe una biyeccion entre el conjunto de palindromos distintos contenidos en w y el conjunto de prefijos que contienen un sufijo palindromico unicocurrente.

Dada $w \in \Sigma^*$ y dado $u = w[i...j]$ un factor simetrico de w diremos que u es un *subpalindromo maximal* si y solo si o u es un sufijo o u es un prefijo o $w[i-1...j+1]$ no es un palindromo.

Definition 55. *Dado $i \leq |w|$ usaremos el simbolo $LMP_w[i]$ para denotar la longitud del subpalindromo maximal mas largo que termina en la posicion i (si tal subpalindromo maximal existe, en caso de no existir definimos $LMP_w[i]$ como -1).*

Definition 56. *Dado $i \leq |w|$ usaremos el simbolo $LPS_w[i]$ para denotar la longitud del sufijo palindromico mas largo del factor $w[1...i]$.*

Lemma 24. *Si $|w| = n$ se tiene que*

1. *$LPS_w[n] = LMP_w[n]$.*
2. *Para todo $i \lneqq n$ se tiene que*

$$LPS_w[i] = \max\{LMP_w[i], LPS_w[i+1] - 2\}$$

Proof. El item 1 es obvio, probaremos el item 2. Es claro que $LPS_w[i] \geq LMP_w[i]$, suponga que $k = LPS_w[i] \gneqq LMP_w[i]$, esto implica que $w[i-k+1...i]$ es el sufijo palindromico mas largo del factor $w[1...i]$ y esto implica tambien que $w[i-k...i+1]$ es un palindromo. Tenemos entonces que $w[i-k...i+1]$ es un sufijo palindromico de $w[1...i+1]$ de longitud $LPS_w[i]+2$.

Para terminar es suficiente probar que este es el sufijo palindromico mas largo del factor $w[1...i+1]$. Suponga que este no es el caso, sea $r \lneqq i-k$ tal que $w[r...i+1]$ es un palindromo. En este caso se tiene que $w[r+1...i]$ es un sufijo palindromico del factor $w[1...i]$ pero esto no es posible dado que $r+1 \lneqq i-k+1$.

Dada $w \in \Sigma^*$ usaremos el simbolo LPS_w para denotar el vector $(LPS_w[i])_{i \leq |w|}$ y el simbolo LMP_w para denotar el vector $(LMP_w[i])_{i \leq |w|}$. Del lema anterior es facil obtener el siguiente corolario.

Corollary 14. *Es posible calcular el vector LPS_w en tiempo $O(|w|)$ si se conoce el vector LMP_w.*

Ahora veremos que es posible calcular el vector LMP_w en tiempo lineal.

Lemma 25. *$LMP[i]$ es el elemento mas grande de la union de los conjuntos A_i, B_i y C_i definidos por:*

- $A_i = \{-1\}$.
- $B_i = \{2lcp_{w,\overline{w}}(k, n-k+1) - 1 : i = k + lcp_{w,\overline{w}}(k, n-k+1) - 1\}$.
- $C_i = \{2lcp_{w,\overline{w}}(k, n-k+2) : i = k + lcp_{w,\overline{w}}(k, n-k+2)\}$.

Proof. Si no existe un palindromo maximal que termine en la posicion i los conjuntos B_i y C_i resultan ser vacios y en este caso

$$LMP[i] = -1 = \max_z \{z : z \in A_i \cup B_i \cup C_i\}$$

Suponga que existen palindromos maximales que terminan en la posicion i y suponga ademas que $w[k...i]$ es el mas largo de estos palindromos. Podemos suponer que este palindromo es de longitud impar, sean entonces r y s dos enteros positivos tales que $k = r - s$ e $i = r + s$. Note que la posicion r es el centro de este palindromo, note tambien que

$$i = r + s = r + lcp_{w,\overline{w}}(r, n-r+1) - 1$$

dado que $w[k...i]$ es el palindromo maximal centrado en r y su radio es igual a $s+1$. Tenemos entonces que

$$LMP[i] = 2(s+1) - 1 \leq \max_z \{z : z \in A_i \cup B_i \cup C_i\}$$

Es facil verificar que

$$LMP[i] \geq \max_z \{z : z \in A_i \cup B_i \cup C_i\}$$

Sea MAX_w el vector definido por: dado $i \leq n$, la entrada $MAX_w[i]$ es igual al radio del palindromo maximal centrado en i

De lo anterior tenemos que si pudieramos calcular en tiempo lineal el vector MAX_w, podriamos entonces calcular en tiempo lineal el vector LMP_w.

El algoritmo de Gusfield [22], que estudiamos en la seccion anterior, nos permite calcular en tiempo lineal el vector MAX_w. Tenemos entonces que es posible calcular en tiempo lineal los vectores LMP_w, LSP_w y MAX_w. Calcular en tiempo lineal el vector LSP_w nos permite calcular en tiempo lineal el LPS de cada prefijo de w. Nos falta entonces determinar cuales de estos sufijos palindromicos son uniocurrentes. Dado $i \leq n$ definimos $LPF_w[i]$ de la siguiente manera

$$LPF_w[i] = \max_k \{w[i...i+k-1] \text{ es un factor de } w[1...i+k-2]\}$$

Crochemore e Ilie exhiben en [10] un algoritmo de tiempo lineal que con input w calcula el vector LPF_w.

Lemma 26. *Dado $i \leq n$ se tiene que $w[i - LSP[i] + 1...i]$ es uniocurrente en $w[1...i]$ si y solo si*

$$LPF_w[i - LPS[i] + 1] \lneqq LPS[i]$$

Proof. $w[i - LSP[i] + 1...i]$ no es uniocurrente en $w[1...i]$ si y solo si existe $l \lneqq i - LPS[i] + 1$ tal que

$$w[l...l + LPS[i] - 1] = w[i - LSP[i] + 1...i]$$

si y solo si $w[i - LSP[i] + 1...i]$ es un factor de $w[1...i-1]$ si y solo si

$$LPF_w[i - LPS[i] + 1] \geq LPS[i]$$

Corollary 15. *(Groult, Prieur, Richomme)*

1. *El numero de palindromos distintos contenidos en w es igual a*

$$|\{i \leq n : LPF_w[i - LPS[i] + 1] \lneqq LPS[i]\}|$$

   y esta cantidad puede ser calculada en tiempo lineal.
2. *El lenguaje $RICH$ puede ser reconocido en tiempo lineal.*

19 El algoritmo de Fischer-Paterson: un problema de optimizacion.

Considere el siguiente problema de optimizacion

Problem 8. (LONGEST: calculo del palindromo inicial mas largo)

- *Input: w, donde w es un elemento de Σ^*.*
- *Problema: calcule el palindromo inicial mas largo contenido en w.*

En capitulos anteriores nos hemos referido al algoritmo FP (algoritmo de Fischer-Paterson) como a un algoritmo que permite resolver el problema $LONGEST$ en tiempo lineal. El problema $LONGEST$ es un tipico problema de optimizacion: calcule una solucion que maximize una cierta funcion de costo (en nuestro caso la funcion de costo es la funcion de longitud).

En esta seccion estudiaremos el algoritmo FP. Fijemos $\Sigma = \{0,1\}$ y sea $F : \Sigma^* \to \Sigma^*$ la funcion definida por

$$\text{si } F(w_1...w_n) = x_1...x_n \text{ se tiene que } x_i = 1 \text{ si y solo si } w[1...i] \in Pal$$

Calcular la funcion $F(w)$ es equivalente a listar todos los palindromos iniciales contenidos en w. Si podemos listar en tiempo lineal todos los palindromos iniciales contenidos en w, podemos entonces calcular (en tiempo lineal) el palindromo inicial mas largo contenido en w.

Sea $G : \Sigma^* \to \Sigma^*$ la funcion definida por

$$\text{si } G(w_1...w_n) = x_1...x_n \text{ se tiene que } x_i = 1 \text{ si y solo si } w[i...n] \in Pal$$

Note que calcular G es equivalente a listar todos los sufijos palindromicos de w, note tambien que calcular G en tiempo lineal permite calcular F en tiempo lineal y permite resolver (en tiempo lineal) el problema $LONGEST$. Lo que estudiaremos en este capitulo es un algoritmo de tiempo lineal que calcula la funcion G.

Remark 20. Podemos usar el algoritmo de Galil-Slisenko para calcular la funcion G (y resolver el problema $LONGEST$) en tiempo real. Esto podria hacernos pensar que no vale la pena estudiar un algoritmo de tiempo lienal que calcule F. La razon por la cual si es interesante es que la maquina de Galil-Slisenko requiere, como uno de sus ingredientes fundamentales, un algoritmo de tiempo lineal que calcule F, esto es: para construir la maquina de Galil-Slisenko necesitamos contar con un algoritmo de tiempo lineal que calcule F (y que, obviamente, sea independiente del algoritmo de Galil-Slisenko).

Sea $w \in \Sigma^*$ tal que $|w| = n$, dado $i \leq n$ definimos $P_w(i)$ como

$$P_w(i) = \max_t \left\{ \bigwedge_{1 \leq j \leq t} w_{i-j+1} = w_{t-j+1} \text{ y } t \lneqq i \right\}$$

Dado $w \in \Sigma^*$ y dado $i \leq |w|$ se define $P_w^{(0)}(i) = i$; dado $k \geq 1$ se define $P_w^{(k)}(i) = P\left(P_w^{(k)}(i)\right)$. Note que:

1. $P_w(i)$ es el maximo $t \lneqq i$ tal que el prefijo de w de longitud t coincide con el factor de w de longitud t que termina en la posicion i.
2. Si $P_w^{(k)}(i) = l$, entonces l es el k-esimo numero, en orden descendente, tal que el prefijo de w de longitud l coincide con el factor de w de longitud l que termina en la posicion i.

Claim. Para todo $j \leq n$ se tiene que $P_w(j+1) - 1 \leq P_w(j) \lneqq j$.

Proof. La desigualdad de la derecha es consecuencia de la definicion, probaremos la desigualdad de la izquierda. Suponga que $P_w(j+1) = r$, se tiene que

$$w[1...r] = w[j+2-r...j+1]$$

Lo cual implica que

$$w[1...r-1] = w[j+2-r...j]$$
$$y$$
$$P_w(j) \geq r-1 \geq P_w(j+1) - 1$$

Sea $X_w = \overline{w}cw$, donde $c \notin \{0,1\}$.

Lemma 27. *$w[i...n]$ es un sufijo palindromico de w si y solo si existe k tal que $n - i = P_{X_w}^{(k)}(2n+1)$.*

Proof. Note primero que si u es un sufijo de X_w cuya longitud satisface la desigualdad

$$n+1 \leq |u| \lneqq 2n+1$$

no existe un prefijo v tal que $u = v$, esto es asi porque en todo sufijo u de X_w, que satisfaga la desigualdad anterior, ocurre el caracter c exactamente una vez y a la derecha de esta ocurrencia de c ocurren n caracteres, por otro lado en todo prefijo de X_w de longitud menor que $2n+1$ en el que ocurra c, ocurren a lo mas $n-1$ caracteres a la derecha de c.

De lo anterior tenemos que para todo $k \geq 1$ se tiene que $P_{X_w}^{(k)}(2n+1) \leq n$. Sea $i \leq n$ tal que para algun $k \geq 1$ se satisface la ecuacion $P_{X_w}^{(k)}(2n+1) = i$, de la definicion de P se tiene que

$$\overline{w}[1...i] = w[n-i+1...n]$$

donde $x_1...x_n = \overline{w}$. La ecuacion anterior es equivalente a la ecuacion

$$w_n...w_{n-i+1} = w[n-i+1...n]$$

y esta ultima ecuacion implica que el sufijo $w[n-i+1...n]$ es un palindromo, i.e. un sufijo palindromico.

Por otro lado, si suponemos que $w[n-i+1...n]$ es un sufijo palindromico, se tiene que

$$\overline{w}[1...i] = w[n-i+1...n]$$

lo cual implica que para algun k, el numero i es el k-esimo numero en orden descendente para el cual se cumple que el prefijo de X_w de longitud i coincide con el sufijo de X_w de longitud i, i.e. el numero i es igual a $P_{X_w}^{(k)}(2n+1)$.

Considere el problema PRE definido a continuacion

Problem 9. (Calculo de la funcion P)

- *Input:* w, *donde* $w \in \Sigma^*$.
- *Problema: calcule el vector* $\left[P^{(k)}(|w|)\right]_{k\leq|w|}$.

El lema anterior nos dice que para poder calcular la funcion G en tiempo $O(n)$ es suficiente poder resolver el problema PRE en tiempo $O(n)$.

El algoritmo de Fischer-Paterson, que denotamos con el simbolo FP, es un algoritmo de tiempo lineal que con input w calcula el vector $\left[P^{(k)}(|w|)\right]_{k\leq|w|}$. Defina

- $\Delta_w(i) = 1 + P_w(i) - P_w(i+1)$.
- $p_w(i,k) = P_w^{(k)}(i)$.
- $s_w(i,k) = P_w^{(k)}(i) - P_w^{(k+1)}(i)$.
- $d_w(k,i) = P_w(i) - P_w^{(k)}(i)$.

El algoritmo FP emplea variables p, s y d y parametros i, k. El algoritmo FP es el algoritmo a continuacion:

Algoritmo de Fischer-Paterson

- **Input**: w.
- **Procedimento**:
  1. *Inicializacion*: $\Delta(0) \leftarrow 0$, $p \leftarrow 0$, $s \leftarrow 0$, $d \leftarrow 0$. Vaya al paso $(1,1)$.
  2. *Paso* $(i+1,k)$.
     - Si $w_{p+1} = w_{i+1}$ haga $\Delta(i) \leftarrow d$, $p \leftarrow p+1$, $s \leftarrow s + \Delta(p)$, $d \leftarrow 0$. Vaya al paso $(i+2,1)$.
     - Si $p = 0$ haga $\Delta(i) \leftarrow d$, $p \leftarrow p$, $s \leftarrow s$, $d \leftarrow 0$.
     - En otro caso haga $p \leftarrow p - s$, $s \leftarrow s - \sum_{p-s}^{p-1} \Delta(j)$, $d \leftarrow d + s$. Vaya al paso $(i+1,k+1)$.

No es dificil convencerse de que el algoritmo anterior permite calcular el vector $\left[P^{(k)}(|w|)\right]_{k\leq|w|}$, ligeras modificaciones del algoritmo permiten calcular en tiempo lineal las funciones F y G y el palindromo inicial mas largo contenido en una palabra dada.

20 Ejercicios capitulo 3

1. Estudie el algoritmo de Ukkonen [51].
2. Pruebe que toda palabra de longitud n contiene a lo mas $n+1$ palindromos distintos.
3. Pruebe que el algoritmo de Fischer-Paterson es un algoritmo de tiempo lineal que puede ser implementado por una maquina de Turing con a lo mas seis cabezas (ayuda: consulte la referencia [47]).

Conclusiones y problemas abiertos

A lo largo de este escrito hemos estudiado la computabilidad en tiempo real del lenguaje Pal sobre diferentes tipos de automatas. Algunos de estos automatas, los regulares y los de pila deterministicos, solo pueden realizar computaciones en tiempo real, por lo que no cabe preguntarse si tales automatas pueden reconocer Pal con un tiempo de computo que domina a tiempo real. Los demas automatas, incluyendo los regulares no deterministicos, pueden realizar coputaciones de mayor duracion o bien porque sus cabezas son bidireccionales o bien porque pueden realizar transiciones en vacio (que no hemos mencionado explicitamente porque, entre otras cosas, las transiciones en vacio pueden ser eliminadas y pueden ser entendidas como una forma de no determinismo). La tabla a continacion sumariza parte del trabajo realizado en este escrito.

	real	$O(n)$	$O(n \log(n))$	$o(n^2)$	$O(n^2)$	computable
reg.	no	no	no	no	no	no
reg. no det	no	no	no	no	no	no
reg. doble via	no	no	no	no	no	no
pila det.	no	no	no	no	no	no
pila no det.	si	si	si	si	si	si
pila det 2-*way*	no	si	si	si	si	si
de cola	no	no	no	?[6]	si	si
una cinta	no	no	no	no	si	si
una cinta prob.	no	no	si	si	si	si
aut. reg. prob.	no	no	no	no	no	no
$2pfa$	no	no	no	no	no	no
1-OCA	si	si	si	si	si	si
1-IA	si	si	si	si	si	si
1-CA	si	si	si	si	si	si
quantum 1-way	no	no	no	no	no	no
quantum 2-way	no	?	?	?	?	si
multicinta	si	si	si	si	si	si

La tabla anterior caracteriza la complejidad de Pal respecto a los 17 modelos de computacion considerados (excepto quizas respecto al modelo de automatas cuanticos de doble via $2qfa$). Por otro lado la tabla oculta, en cierta medida, que en todos estos modelos las cotas inferiores y las superiores son ajustadas (excepto en el modelo $2qfa$ y en el modelo de automatas de cola). ¿Queda algo por decir respecto a la dificultad intrinseca de Pal? El lector desprevenido pudo pensar, al iniciar la lectura de este escrito, que no es mucho lo que la complejidad computacional puede decir en torno a un problema, en principio trivial, como Pal. Esperamos que a estas alturas tal prejuicio se haya modificado

[6] vea problema 6

radicalmente. Pero, ¿queda algo por decir? No existe un tema completamente agotado en matematicas, y este es el caso con la complejidad computacional de Pal. A lo largo del escrito hemos enunciado algunos problemas abiertos referentes a Pal (o sus relativos). En este punto quisieramos listar los, a nuestro juicio, problemas abiertos mas interesantes. El primero de ellos consiste en llenar las casillas vacias de la tabla anterior, o mas precisamente:

Problema 1. ¿Cual es el tiempo de computo requerido para reconocer Pal con un automata de doble via cuantico?

Problema 2. Es sabido que reconocer Pal usando automatas de cola requiere un tiempo de computo $\Omega\left(\frac{n^{\frac{4}{3}}}{\log(n)}\right)$ (consulte la referencia [35]). Es facil diseñar un automata de cola de tiempo cuadratico que reconozca Pal, ¿es posible diseñar un automata de cola de tiempo subcuadratico que reconozca Pal?

Problema 3 ¿Cuantas cintas se requieren para reconocer Pal en tiempo real? Sabemos que existe k tal que Pal puede ser reconocido en tiempo real con k cintas, ¿cual es el minimo k? Para empezar seria interesante probar que dos cintas no son suficientes.

Problema 4. ¿El lenguaje Pal^2 puede ser reconocido en tiempo real usando un 1-OCA?

Problema 5. ¿El lenguaje Pal^* puede ser reconocido en tiempo lineal usando un automata de de pila deterministico y de doble via? O mas generalmente ¿la clase $\mathcal{CFL}$ esta contenida en la clase 2-$\mathcal{LDCFL}$?

Problema 6. Sea $RICH$ el conjunto

$$\left\{w \in \{0,1\}^* : w \text{ es rica}\right\}$$

Problem 10. Sabemos que el conjunto $RICH$ puede ser decidido en tiempo lineal, ¿es posible reconocer $RICH$ en tiempo real?

Problema 7. Cuando se trabaja con palabras comprimidas debe evitarse diseñar algoritmos que primero descompriman y luego pasen a trabajar con la palabra descomprimida. Esto es asi dado que en muchos casos la longitud de la palabra descomprimida es exponencial con respecto a la palabra comprimida que se recibe como input, por lo que este tipo de algoritmos (primero descompresion) resultan ser de tiempo exponencial. Diseñar algoritmos para problemas textuales sobre palabras comprimidas (respecto a diferentes esquemas de compresion) es una agenda de investigacion importante (los archivos comprimidos son cada vez mas ubicuos e importantes) y de reciente desarrollo, un buen *survey* del estado de la teoria a finales de la decada de los 90 es el articulo [42].

Considere el problema Eq definido por

- *Input:* (w,u), *donde* w *y* u *son los codigos de Lempel-Ziv (consulte la referencia [33]) de las palabras* x, y.
- *Problema: decida si* x *y* y *son iguales.*

Sea $Eq(n)$ el tiempo de computo necesario para resolver cualquier instancia (w,u) de Eq que satisfaga la desigualdad $|w|, |u| \leq n$. Sea $comp\,(List\text{-}Pal)$ el problema.

- *Input: w, donde w es el codigo de Lempel-Ziv de la palabra x.*
- *Problema: liste todos los palindromos incluidos en x.*

En [42] W. Rytter presenta el siguiente resultado (cota superior):

Existe un algoritmo de tiempo $O\left(|w|\log\left(|w|\right)\log^2\left(|x|\right)Eq\left(|w|\log|w|\right)\right)$ que con input x (un texto) lista todos los palindromos incluidos en x.

¿Es ajustada esta cota superior? ¿Que cotas pueden establecerse para los otros problemas, relacionados con palindromos, que estudiamos en capitulos anteriores? ¿Que cotas pueden establecerse para estos problemas cuando se considera el codigo de Lempel-Ziv-Welch [42]?

21 Ejercicios capitulo 4

1. Investigue las nocion de automata de pila probabilistico, investigue acerca de la recognizabilidad del lenguaje Pal sobre este tipo de automatas.

A manera de apendice: computacion en paralelo

Finalmente, en aras de la completez, hemos decidido incluir un breve apendice referente a la computacion en paralelo. Lo hemos hacho pensando en que este texto pueda ser usado como una introduccion a los conceptos y tecnicas fundamentales de la stringologia.

Es importante anotar que no definiremos un modelo de computacion en paralelo, nuestra aproximacion sera puramente intuitiva: un algoritmo paralelo es un algoritmo que puede ser ejecutado usando una red de procesadores interconectados cuyo tamaño escala con el tamaño del input.

21.1 El poder de la computacion en paralelo: un ejemplo

Podemos ilustrar el poder de la computacion en paralelo considerando el siguiente ejemplo.

Problem 11. (calculo de sumatorias)

- *Input:* $(n, a_1, ..., a_{2^n})$, *donde* n *es un numero natural y* $a_1, ..., a_{2^n}$ *son* 2^n *numeros naturales*
- *Problema: Calcule* $\sum_{i=1}^{2^n} a_i$.

Es claro que un procesador secuencial, o lo que es lo mismo un algoritmo secuencial, debe realizar al menos $\frac{2^n}{2}$ sumas antes de llegar a la respuesta y que por lo tanto un procesador secuencial requiere al menos 2^{n-1} unidades de tiempo para procesar el input $(n, a_1, ..., a_{2^n})$. Suponga ahora que tenemos 2^{n-1} procesadores conectados a un procesador central en el cual se ha almacenado el input $(n, a_1, ..., a_{2^n})$. Podemos usar esta red de computadores para calcular la sumatoria $\sum_{i=1}^{2^n} a_i$ en n unidades de tiempo. El algoritmo paralelo que podriamos usar para realizar un tal calculo es el siguiente:

Sea $(n, a_1, ..., a_{2^n})$ el input de nuestro algoritmo y sea $(\mathcal{M}_1, ..., \mathcal{M}_{2^{n-1}}, \mathcal{K})$ la red de computadores que pretendemos usar

1. *Para todo* $i \leq n$ *haga lo siguiente*
   - *Para todo* $j \leq 2^{i-1}$ *el procesador* $\mathcal{M}_j$ *calcula* $a^i_{2j-1} + a^i_{2j}$, *donde los* a^i_k *se definen de la siguiente manera:*
     - *Para todo* $k \leq 2^n$ *se tiene que* $a^1_k = a_k$.
     - *Si* $j \geq 2$ *entonces para todo* $k \leq 2^{n-j+1}$ *se tiene que* $a^j_k = a^{j-1}_{2k-1} + a^{j-1}_{2k-1}$.
2. *Calcule e imprima* $a^n_1 + a^n_2$.

Es claro que este algoritmo paralelo requiere a lo mas $n+1$ unidades de tiempo para procesar el input $(n, a_1, ..., a_{2^n})$: el algoritmo consta de $n+1$ iteraciones, y cada iteracion se reduce a sumar en paralelo parejas de numeros naturales, (lo cual asumimos se puede realizar en una unidad de tiempo). Note que el algoritmo

paralelo permite procesar inputs de tamaño 2^n en tiempo $n+1$, mientras que el mejor algoritmo secuencial requiere tiempo $\Omega\left(2^n\right)$ para procesar el mismo tipo de inputs (y esto para todo $n \geq 2$). Sea $t_{par}\left(n\right)$ el tiempo de computo del algoritmo paralelo al procesar inputs de tamaño 2^n y sea $t_{\text{sec}}\left(n\right)$ el tiempo de computo del algoritmo secuencial en el mismo tipo de inputs, es claro que

$$t_{\text{sec}}\left(n\right) = 2 \cdot 2^{t_{par}(n)} - 1$$

O equivalentemente

$$t_{par}\left(n\right) = \log\left(t_{\text{sec}}\left(n\right) + 1\right) - 1$$

Esta ultima ecuacion indica que el problema de calcular sumatorias admite un aceleramiento o mejora (*speed up*) de tipo exponencial, cuando se consideran algoritmos paralelos.

Remark 21. Note que dada $\left(n, a_1, ..., a_{2^n}\right)$, una instancia del problema de calcular sumatorias, su tamaño (en el sentido algoritmico) es al menos 2^n. El numero de procesadores requeridos por el algoritmo paralelo para procesar un input de tamaño 2^n es menor o igual que 2^{n-1} y por lo tanto polinomial en el tamaño del input. Cuando se consideran algoritmos paralelos es importante cuantificar y acotar el numero de procesadores requeridos por el algoritmo. Si el algoritmo requiere un numero constante de procesadores (un numero que no depende del input) el algoritmo es esencialmente un algoritmo secuencial. Si el numero de procesadores requerido por el algoritmo crece exponencialmente respecto al tamaño de los inputs, el algoritmo es inpractico y no realista (*unfeasible*). Es un principio fundamental en teoria de la computacion que los recursos computacionales requeridos por un algoritmo que quepa considerar util (*feasible*) no pueden crecer de manera superpolinomial con respecto al tamaño de los inputs. Dado que el numero de procesadores es un recurso computacional, es la convencion estandard en la teoria de algoritmos paralelos que este numero no puede crecer de manera superpolinomial con respecto al tamaño de los inputs.

Dado L un problema, el *tiempo secuencial* de L es igual al tiempo de computo del mas eficiente de los algoritmos secuenciales que resuelven L (i.e. el tiempo de computo de la mas eficiente de las maquinas de Turing que resuelven L). Dada $f : \mathbb{N} \to \mathbb{N}$, la funcion f es el tiempo secuencial de L si y solo si existe una maquina de Turing $\mathcal{M}$ que resuelve L en tiempo f y tal que para toda maquina de Turing $\mathcal{N}$ se tiene lo siguiente: si $\mathcal{N}$ resuelve L entonces $f \in O\left(t_{\mathcal{N}}\right)$. Usaremos el simbolo $\text{sec}\left(L\right)$ para denotar el tiempo secuencial de L.

Dado $\mathcal{A}$ un algoritmo paralelo, el numero de procesadores usados por $\mathcal{A}$ es la funcion $\#_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ definida por:

$$\#_{\mathcal{A}}\left(n\right) := \max_{w \in \Sigma^n}\left(\#_{\mathcal{A}}\left(w\right)\right)$$

donde $\#_{\mathcal{A}}\left(w\right)$ es el numero de procesadores usados por el algoritmo $\mathcal{A}$ al procesar el input w.

Theorem 26. *Para todo problema L y para todo algoritmo paralelo $\mathcal{A}$ que resuelve L se tiene que*

$$\sec(L) \in O(\#_{\mathcal{A}} t_{\mathcal{A}})$$

Proof. Presentaremos un esbozo de la prueba. Dado L, la cantidad $\sec(L)(n)$ corresponde al numero minimo de operaciones elementales que es necesario realizar al procesar instancias de L de tamaño n. Un algoritmo paralelo por ingenioso que sea no nos permitira realizar un numero significativamente menor de operaciones, note que $\#_{\mathcal{A}} t_{\mathcal{A}}(n)$ es mayor o igual que el numero de operaciones elementales realizadas por $\mathcal{A}$ al procesar instancias de L de tamaño n. De lo anterior tenemos que $\sec(L) \in O(\#_{\mathcal{A}} t_{\mathcal{A}})$.

Suponga sin embargo que existe un algoritmo paralelo $\mathcal{A}$ tal que $\sec(L) \notin O(\#_{\mathcal{A}} t_{\mathcal{A}})$. Podemos convertir a $\mathcal{A}$ en un algoritmo secuencial $\mathcal{M}$ (cada uno de los procesadores usados por $\mathcal{A}$ es simulado por el unico procesador de $\mathcal{M}$) tal que $\#_{\mathcal{A}} t_{\mathcal{A}} = t_{\mathcal{M}}$. Por lo tanto $\sec(L) \notin O(t_{\mathcal{M}})$, lo cual es imposible

Corollary 16. *Si $\mathcal{A}$ es un algoritmo paralelo que resuelve el problema Pal, se tiene que*

1. $\#_{\mathcal{A}}(n) \geq \frac{n}{t_{\mathcal{A}}}$.
2. $t_{\mathcal{A}}(n) \geq \frac{n}{\#_{\mathcal{A}}}$.

En lo que queda del apendice estudiaremos el siguiente teorema

Theorem 27. *Existe un algoritmo paralelo $\mathcal{M}$ que resuelve el problema List-MaxPal en tiempo $O(\log(n))$ y usa n procesadores.*

Remark 22. Podemos afirmar que el algoritmo $\mathcal{M}$, en el enuciado del teorema anterior, no es optimo dado que existe $C \geq 1$ tal que

$$\#_{\mathcal{M}}(n)\, t_{\mathcal{M}}(n) \geq Cn\log(n) \notin O(n) = O(\sec(n))$$

Construccion y analisis del algoritmo Sea $w = w_1 w_2 ... w_n$ una instancia de $List\text{-}MaxPal$, estamos interesados en listar todos los subpalindromos maximales de longitud par contenidos en w. Dado $i \leq n$ el simbolo R_i denotara la cantidad $EPAL_w[i]$ que es el radio del mayor subpalindromo par centrado en i.

Lo que nosotros probaremos es que existe un algoritmo $\mathcal{M}$ que resuelve el problema $List\text{-}MaxPal$ en tiempo $O(\log(n))$ y que usa n procesadores. El algoritmo $\mathcal{M}$ es precisamente el algoritmo en el enunciado del teorema 27.

Antes de presentar y analizar el algoritmo $\mathcal{M}$ necesitamos introducir un concepto adicional y probar algunos lemas auxiliares.

Definition 57. *Dada $w = w_1 ... w_n$ diremos que w tiene un periodo de longitud j si y solo si para todo $i \leq n - j$ se tiene que $w_i = w_{i+j}$.*

Lemma 28. *Dada $w = w_1 ... w_n$ una palabra, si w tiene periodos de longitud i y j, con $i + j \leq n$, se tiene entonces que w tiene un periodo de longitud* $\gcd(i, j)$.

Una prueba del lema puede ser encontrada en la referencia [37].

Lemma 29. *Suponga que $w = w_1...w_n$ contiene dos palindromos pares cuyos radios son mayores que $r-1$ y cuyos centros i y j satisfacen*

- $i \lneqq j$.
- $j - i \leq r$.

Entonces, la subpalabra $w[i-r...j+r-1]$ tiene un periodo de longitud $2(j-i)$.

Proof. Sea $1 \leq k \leq r$

$$w_{i-k} = w_{i+k-1} = w_{j-(j-i)+k-1} = w_{j+(j-i)-k} = w_{i+2(j-i)-k}$$

y

$$w_{j+k-1} = w_{j-k} = w_{i+(j-i)-k} = w_{i-(j-i)+k-1} = w_{j-2(j-i)+k-1}$$

Lemma 30. *Suponga que $w = w_1...w_n$ contiene un palindromo par de radio mayor que $r-1$ y centrado en k. Asuma tambien que $w[e_1...e_2]$ es la subpalabra maximal que contiene a $w[k-r...k+r-1]$ y tiene periodo de longitud $2r$. Entonces, para todo $c = k + lr$ (con $e_1 \leq c \leq e_2$ y $l \in \mathbb{Z}$) se tiene que:*

- *Si $c - e_1 \neq e_2 - c + 1$, entonces $R_c = \min(c - e_1, e_2 - c + 1)$.*
- *Si $c - e_1 = e_2 - c + 1$, entonces $R_c \geq c - e_1$ y la igualdad se tiene si y solo si $w_{e_1-1} \neq w_{e_2+1}$.*

Proof. Por la periodicidad de $w[e_1...e_2]$ se tiene que $w_i = w_j$ si $e_1 \leq i, j \leq e_2$ e $i \equiv j \bmod (2r)$. Tenemos entonces que $w_i = w_j$ si $e_1 \leq i, j \leq e_2$ e $i + j \equiv 2k - 1 \bmod (2r)$, dado que existe un palindromo par de radio r y centrado en k.

Considere el palindromo par centrado en $c = k+lr$ ($l \in \mathbb{Z}$) tal que $e_1 \leq c \leq e_2$. Dado que $(c-i) + (c+i-1) \equiv 2k - 1 \bmod (2r)$ tenemos que $w_{c-i} = w_{c+i-1}$ para todo $i \leq \min(c - e_1, e_2 - c)$.

Si $c - e_1 \lneqq e_2 - c + 1$ entonces $w_{c-(c-e_1+1)} \neq w_{c+(c-e_2+1)-1}$, dado que $w_{e_1-1} \neq w_{e_1+2r-1}$ y $w_{2c-e_1} = w_{e_1+2r-1}$. De lo anterior se tiene que si $c - e_1 \lneqq e_2 - c + 1$ entonces el radio es exactamente $c - e_1$. Si suponemos $c - e_1 \gneqq e_2 - c + 1$ podemos usar un argumento analogo para probar que el radio es exactamente $e_2 - c + 1$.

Finalmente si $c - e_1 = e_2 - c + 1$ entonces el radio es mayor que $c - e_1$ si y solo si $w_{e_1-1} = w_{e_2+1}$

Theorem 28. *Existe un algoritmo paralelo $\mathcal{M}$ que con input $w = w_1...w_n$ calcula el vector $EPAL_w$ en tiempo $O(\log(n))$ empleando $O(n)$ procesadores.*

Proof. Sea $w = w_1...w_n$ un input de $\mathcal{M}$. El computo de $\mathcal{M}$ en el input w esta constituido por $\log(n) - 1$ iteraciones diferentes. Dado $0 \leq i \leq \log(n) - 2$, en la i-esima iteracion el input w es particionado en $\frac{n}{2^i}$ bloques consecutivos de longitud 2^i. En la i-esima iteracion podemos usar $\frac{n}{2^i} \leq n$ procesadores para trabajar simultaneamente en cada bloque. .

Supondremos que para todo $i \leq \log(n)-2$, en la i-esima iteracion el algoritmo $\mathcal{M}$ calcula un vector $V_i = \left(R_i^{(x_j)}\right)_{j \leq \frac{n}{2^i}}$ con $R_i^{(x_j)} = (e_k)_{(j-1)2^i+1 \leq k \leq j2^i}$ y para todo $(j-1)\,2^i + 1 \leq k \leq j2^i$

$$e_k = \begin{cases} R_k \text{ si } R_k \leq 2^{i+2} - k \\ ? \text{ en caso contrario} \end{cases}$$

Suponga que la etapa i-esima del computo del algoritmo $\mathcal{M}$, con input $w = w_1...w_n$, ha sido efectuada. El nucleo del algoritmo consiste en calcular el vector V_{i+1} a partir del vector V_i. Note primero que

1. La etapa 1 puede ser realizada en tiempo constante usando a lo mas n procesadores.
2. Dado que el algoritmo $\mathcal{M}$ consta de $\log(n) - 1$ iteraciones (etapas), para que el tiempo de computo del algoritmo sea $O(\log(n))$, cada etapa debe ejecutarse en tiempo constante.
3. Usando n procesadores es posible calcular en tiempo constante todos los radios de tamaño menor o igual que 4.
4. Al final de la etapa i-esima, todos los radios de longitud menor que 2^{i+2} han sido determinados, pero los radios mayores o iguales que 2^{i+2} no han sido determinados. En la etapa i usamos un procesador por bloque de manera tal que el procesador asignado al bloque j ($j \leq \frac{n}{2^i}$) calcule los radios asociados a las posiciones en el bloque j. Para ello el procesador asignado al bloque j trabaja con este y con los cuatro bloques a su izquierda y los cuatro bloques a su derecha (si existen menos que cuatro bloques a derecha o izquierda se toma el maximo numero posible de bloques). De esta manera el procesador asociado al bloque j podra calcular los radios que sean menores que 2^{i+2}. Algunos radios mayores que $2^{i+2} - 1$ podrian ser calculados, pero en la etapa $i-$esima el procesador asignado los ignora, aplazando su calculo hasta la siguiente(s) iteracion.
5. Del item 4 se tiene que al final de la etapa $\log(n)-2$ todos los radios han sido determinados (y el problema $List\text{-}MaxPal$ ha sido resuelto para la instancia w) dado que todo palindromo contenido en w tiene una longitud acotada por n.

Que los items de lista anterior sean efectivamente realizables es evidente en cada uno de los casos, excepto quizas en el item 4, el cual requiere un analisis cuidadoso, este analisis hace parte de lo que queda de la prueba. Note que un punto clave del algoritmo consiste en mantener, a lo largo de las $\log(n) - 1$ iteraciones, el siguiente invariante asociado al item 4:

Al final de la i-esima iteracion todos los radios de longitud menor que 2^{i+2} *han sido determinados.*

Dado x uno de los $\frac{n}{2^i}$ bloques de longitud 2^i, al final de la i-esima etapa existen posiciones en el bloque para las cuales los radios ya han sido determinados y por otro lado existen posiciones, digamos $c_1 \lneqq c_2 \lneqq ... \lneqq c_l$ para las cuales los radios aun no han sido determinados.

Afirmacion. Si $l \geq 3$ las posiciones $c_1, c_2, ..., c_l$ constiyuyen una progresion aritmetica, i.e. existe c tal que para todo $i \leq 2, ..., l-1$ se tiene lo siguiente: $c_{i+1} - c_i = c = c_i - c_{i-1}$.

Note que toda progresion aritmetica puede ser determinada usando tan solo tres numeros naturales: el inicio, el salto y la longitud de la secuencia, esto es: dada $c_1 \lneqq c_2 \lneqq ... \lneqq c_l$ una progresion aritmetica, esposible definir esta progresion usando una tripla (in, s, l), donde (in, s, l) es igual a $(c_1, c_2 - c_1, l)$. Note tambien que dados dos bloques contiguos que contienen posiciones indeterminadas definidas por las triplas (in_1, s_1, l_1) y (in_2, s_2, l_2), es posible calcular en tiempo constante una tripla (in_3, s_3, l_3) que defina la sucesion de posiciones indeterminadas en la union de estos dos bloques.

En lo que sigue mostraremos como, dados x_1 y x_2 dos bloques consecutivos de longitud 2^i, el algoritmo $\mathcal{M}$ calcula $R_{i+1}^{(x_1x_2)}$ a partir de $R_i^{(x_1)}$ y $R_i^{(x_2)}$.

1. Si x_1x_2 contiene una unica posicion indeterminada, el algoritmo $\mathcal{M}$ verifica si el radio asociado es mayor o igual que 2^{i+3}, si este es el caso la posicion permanece indeterminada, si este no es el caso $\mathcal{M}$ calcula el radio asociado a esta posicion.
2. Si x_1x_2 contiene una progresion aritmetica no trivial de posiciones indeterminadas, definida por la tripla (in, s, l), el algoritmo $\mathcal{M}$ hace lo siguiente: Sea $w[i...j]$ la palabra maximal que contiene a $w[c_1 - c...c_l + c - 1]$ y que es periodica con periodo de longitud $2c$. Por el lema 29 el radio asociado a cada posicion c_k es igual a $\min(c_k - i, j - c_k + 1)$, excepto para a lo mas uno de los c_k, el cual debe satisfacer la ecuacion $c_k - i = j - c_k + 1$. En este ultimo caso el algoritmo $\mathcal{M}$ verifica si $w\left[c_1 - 2^{i+3}...c_l + 2^{i+3} - 1\right]$ es una palabra periodica con periodo de longitud $2c$. Si la verificacion es negativa, el algoritmo $\mathcal{M}$ calcula al menos uno de los dos elementos del conjunto $\{i, j\}$ y entonces determina si el radio asociado a c_k es mayor o igual que 2^{i+3}. Si el radio es menor que 2^{i+3} el algoritmo calcula este radio.

Para terminar la prueba del teorema es suficiente (y facil) convencerse de que los computos realizados en los item 1 y 2 de la lista anterior pueden ser realizados en tiempo constante usando los lemas 29 y 30.

Remark 23. El algoritmo $\mathcal{M}$ del teorema anterior fue tomado de la referencia [3], en este mismo trabajo los autores presentan un algoritmo $\mathcal{N}$ que resuelve el problema *List-MaxPal* en tiempo $O(\log(\log(n)))$ usando $\frac{n\log(n)}{\log(\log(n))}$ procesadores. Un interesante ejercicio para el lector consiste en estudiar la referencia [3] y mas especificamente el algoritmo $\mathcal{N}$ que alli se presenta. Por otro lado existen trabajos, pertenecientes al area de reconocimiento de patrones, en los cuales se considera un problema mas general que *List-MaxPal*, a saber: dada w una palabra, liste todos los palindromos aproximados contenidos en w. La nocion de palindromo aproximado puede variar dependiendo de los autores y de las aplicaciones en mente, la idea intuitiva es que un palindromo aproximado x es una palabra obtenida a partir de un palindromo y, reemplazando algunos caracteres de y e insertando algunos caracteres adicionales (Intuitivamente un palindromo

aproximado es una palabra que se obtiene al transmitir un palindromo a travez de un canal imperfecto), el lector interesado puede consultar la referencia [31].

22 Ejercicios apendice

1. Pruebe el lema 28.

References

[1] S. Aanderaa. On k-tape vs (k-1)-tape real time computation. SIAM-AMS proceedings, 7 :75-96, 1974.

[2] A. Ambainis, J. Watrous. Two-way finite automata with quantum and classical state. Theor. Comput. Sci. 287(1): 299-311, 2002.

[3] A. Apostolico, D Breslauer, Z. Galil. Parallel Detection of all the Palindromes in a Sring. Theoretical Computer Science 141(1&2):163-173, 1995.

[4] R. Backofen, P. Clote. Computational molecular biology. John Wiley & Sons, NY 2000.

[5] J. Becvar. Real-time and complexity problems in automata theory. Kybernetika, 1(6): 475-498, 1965.

[6] T. Biedl, J. Buss, E. Demaine, M. Demaine, M. Hajiaghayi, M. Vinar. Palindrome Recognition Using a Multidimensional Tape. Theoretical Computer Science 302(1-3): 475-480 (2003).

[7] S. Cook. The complexity of theorem-proving procedures. STOC proceedings: 151-158, 1971.

[8] S. Cook. Linear time simulation of deterministic two-way pushdown automata. Information Processing 71: 75-80, 1972.

[9] S. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Transactions on Computing. C-18: 349-365, 1969.

[10] M. Crochemore, L. Ilie. Computing Longest Previous Factor in Linear Time and Applications. Information Processing Letters 106(2): 75-80, 2008.

[11] C. Dwork, L. Stockmeyer. Finite state verifiers I: the power of interaction. Journal of the ACM, 39(4): 800-828.

[12] M. Fischer, A. Meyer, A. Rosenberg. Real-time Simulations of Multihead Units. Journal of ACM 19:590-607, 1972.

[13] M. Fischer, M. Paterson. String Matching and other Products. Proceedings Symposioum SIAM ACM on Complexity of Computing: 113-126, 1974.

[14] M. Fischer, C. Kintala. Real time computations with restricted nondeterminism. Math. Systems theory 12: 219-231, 1979.

[15] R. Freivalds. Fast computation by probabilistic Turing machines. Theory of Algorithms and Programs, Latvian State University 2:201-205, 1975.

[16] Z. Galil. Palindrome Recognition in Real Time. Journal of Computers and Systems Sciences 16(2):140-157, 1978.

[17] Z. Galil. String matching in Real Time. Journal of the ACM 28(1):134-149, 1981.

[18] A. Glen, J. Justin, S. Widmer, L. Zamboni. Palindromic Richness. European Journal of Combinatorics 30: 510-531, 2009.

[19] J. Goldstine, H. Leung, D. Wotschke. Measuring nondeterminism in pushdown automata. Journal of computer and system sciences 71: 440-466, 2005.

[20] R. Groult, I. Prieur, G. Richomme. Counting distinct palindromes in a word in linear time. Information Processing Letters 110: 908-912, 2010.

[21] L. Grover. A fast quantum mechanical algorithm for database search. Proceedings of STOC: 212-219, 1996.

[22] D. Gusfield. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York NY, USA, 1997.

[23] G. Hardy, E Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, Oxford UK, 1960.

[24] D. Harel, R. Tarjan. Fast algorithms for finding nearest common ancestors. SIAM Journal on Computing 13(2): 388-355, 1984.

[25] J. Hartmanis, R. Stearns. Computational omplexity of recursive functions. Proceedings of 5^{th} symposium on switching theory: 82-90, 1964.
[26] F. Hennie. Crossing Sequences and Off-line Turing Machines. Proceedings FOCS :168-172, 1965.
[27] F. Hennie. One tape offline Turing machine computations. Information and Control, 8(6): 553-578, 965.
[28] J. Hopcroft, J. Ullman, R. Motwani. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1989.
[29] C. Kintala. Refining nondeteminism in context-free languages. Math Systems Theory 12: 1-8, 1978.
[30] D. Knuth, E. Morris, V. Pratt. Fast pattern matching in strings. Tech. report CS 440, Stanford University Computer Science, 1974.
[31] R. Kolpakov, G. Kucherov. Searching for Gapped Palindromes. Proceedings CPM:18-30, 2008.
[32] M. Kutrib. Cellular Automata and Language Theory. En R. Meyer (editor); *Encyclopedia of Complexity and System Theory.* Springer Verlag, Heidelberg, pages 800-823, 2009.
[33] A. Lempel, J. Ziv. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3): 337-343, 1977.
[34] B. Leong, J. Seiferas. New real-time simulations of multi-head tape units. Proceedings 9^{th} STOC: 239-248.
[35] M. Li, L. Longpre, P. Vitanyi. The power of queue. SIAM journal on computing, 21(4): 697-712, 1992.
[36] M. Lothaire. Applied Combinatorics on Words. Cambridge University Press, New York, NY, USA, 2005.
[37] R. Lyndon, M. Schutzenberger. The equation $a^m = b^n c^p$ in a free group. Michigan Mathematical Journal, 9:289-298, 1962.
[38] G. Manacher. A new linear time online algorithm for finding the smallest initial palindrome of a string. Journal of the ACM 22(3): 346-351, 1975.
[39] C. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.
[40] N. Pippenger. Private Communication.
[41] M. Rabin. Real-time computation. Israel Journal of mathematics, 1:203-211, 1963.
[42] W. Rytter. Algorithms on compressed strings and arrays. Proceedings of SOFSEM Lecture Notes in Computer Science 1725: 48-65, 1999.
[43] E. Seneta. Non-negative matrices and Markov chains. Springer-Verlag, NY, 1981.
[44] P. Shor: Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5): 1484-1509, 1997.
[45] M. Sipser. *Introduction to the Theory of Computation.* Course Technology, 2005.
[46] A. Slisenko. Recognition of Palindromes by Multihead Turing Machines. Proceedings of the Steklov Institute of Mathematics No. 129: 30-202, 1973.
[47] A. Slisenko. A simplified proof of the real-time recognazability of palindormes on Turing machines. Zapiski Nauchnykh 68: 123-139, 1977.
[48] I. Sudborough. Time and Tape Bounded Auxiliary Pushdown Automata. LNCS No. 53:493-503, 1977.
[49] H. Tanaka. Large DNA palindromes as a common form of structural chromosome aberration in human cancers. Human Cell 19(1): 17-23, 2006.
[50] B. Trachtenbrot. Turing computations with logarithmic delay. Algebra i Logik, 3(4): 33-48, 1964.
[51] E. Ukkonen. Online construction of suffix trees. Algorithmica 14(3): 249-260, 1995.
[52] P. Weiner. Linear pattern matching algorithm. Proceedings 14^{th} IEEE Symposioum on Switching and Automata Theory: 1-11, 1973.

[53] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing,* 8(3):410-421, 1979.

[54] H. Yamada. Real time computations and recursive functions non real time computable. IRE transactions on electronic computers EC-11: 753-760, 1962.

[55] A. Yao. A Lower Bound for Palindrome Recognition by Probabilistic Turing Machines. Technical report #77-647, Standford University, 1977.