2.4 Algoritmos de empate de cadenas

Los programas de edición de texto con frecuencia necesitan encontrar todas las apariciones de un patrón en el texto. Por lo general, el texto es un documento que se está editando y el patrón buscado es una palabra particular proporcionada por el usuario. Existen algoritmos eficientes para este problema, llamados "Coincidencia de cadenas" (string matching) puede ayudar mucho a la capacidad de respuesta del programa de edición de texto. Entre sus muchas otras aplicaciones, los algoritmos de coincidencia de cadenas buscan patrones en secuencias de ADN. Los motores de búsqueda en Internet también los usan para encontrar páginas web relevantes para consultas.

Formalización

El problema de coincidencia entre cadenas puede ser tratado de la siguiente forma:

Suponga que se tiene un texto y es una matriz $T[1, \ldots., n]$ de longitud *n* y que el patrón es una matriz $P[1, \ldots., m]$ de longitud m y $m < n$. Suponga además que los elementos de *P* y *T* son caracteres extraídos de un alfabeto finito $\sum$. Por ejemplo, podemos tener $\sum = \{0, 1\}$ o $\sum = \{a, b, c, \ldots, z\}$.

Las matrices de caracteres *P* y *T* a menudo se denominan cadenas de caracteres. Si observamos la figura 2.4.1, se puede decir que el patrón *P* ocurre con el desplazamiento *s* en el texto *T* (o, de manera equivalente, que el patrón *P* ocurre a partir de la posición *s* + 1 en el texto *T*) si $0 \leq s \leq n - m$ y $T[s + 1 \ldots s + m] = P[1 \ldots m]$ para $1 \leq j \leq m$ (es decir, si $T[s + j] = P[j]$, para$1 \leq j \leq m$). Si *P* ocurre con el desplazamiento *s* en *T*, entonces llamamos a *s* un desplazamiento válido; de otra forma, llamamos a *s* un turno no válido. El problema de coincidencia de cadenas es el de encontrar todos los cambios válidos con los que ocurre un patrón *P* dado en un texto *T*.

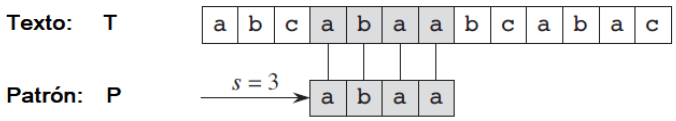


Figura 2.4.1 Coincidencia entre cadenas (String Matching)

En la tabla 2.4.1 se muestran los algoritmos para resolver este problema con su complejidad

Tabla 2.4.1 Algoritmos para String Matching.

Algoritmo	Tiempo de pre procesamiento	Tiempo de coincidencia
Ingenuo	0	$O\big((n - m + 1)m\big)$
Rabin-Karp	$\Theta(m)$	$O\big((n - m + 1)m\big)$
Autómata finito	$O(m\|\Sigma\|)$	$\Theta(n)$
Knut-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

Notación y terminología

Se denotará como Σ^* (se lee: sigma estrella) y es el conjunto de todas las cadena finitas formadas utilizando los caracteres del alfabeto Σ. Solo se consideran cadenas de longitud finita. La cadena de longitud cero es una cadena vacía denotada por ε y también pertenece a Σ^*. La longitud de una cadena *x* se denota por $|x|$. La concatenación de dos cadenas *x* e *y* será *xy*, y tiene longitud $|x| + |y|$ y consiste de los caracteres de *x* seguidos de los caracteres de *y*.

Digamos que una cadena *w* es un prefijo de una cadena *x*, y lo llamaremos así $w \sqsubset x$, si $w = x$ y para alguna cadena $y \in \Sigma^*$. Note que si $w \sqsubset x$ entonces $|w| \leq |x|$. De igual forma se puede decir que la cadena *w* es un sufijo de la cadena *x*, decimos que $w \sqsupset x$ si $x = yw$ para algún $y \in \Sigma^*$. Así que con un prefijo $w \sqsupset x$ implica $|w| \leq |x|$. Por ejemplo, tenemos $ab \sqsubset abcca$ y $cca \sqsupset abcca$. La cadena vacía ε es ambas un sufijo y un prefijo de cualquier cadena. Para alguna cadena *x* e *y*, para cualquier carácter *a*, podemos tener $x \sqsupset y$ si y solo si $xa \sqsupset ya$. También deberemos notar que $\sqsubset$ y $\sqsupset$ son relaciones transitivas.

Lema 2.4.1 de sufijo superpuesto

Suponga que *x*, *y*, *z* son cadenas tal que $x \sqsupset z$ e $y \sqsupset z$.

Si $|x| \leq |y|$ entonces $x \sqsupset y$.
Si $|x| \geq |y|$ entonces $y \sqsupset x$.
Si $|x| = |y|$ entonces $y = x$.

Demostración

Vea la figura 2.4.2 para que se dé una idea gráfica de esto. Denotemos un *k* carácter prefijo $P[1 \ldots k]$ de un patrón $P[1 \ldots m]$ por P_k. Así $P_0 = \varepsilon$ y $P_m = P = P[1 \ldots m]$. De igual forma si llamamos un *k* carácter prefijo del texto *T* por T_k. Empleando está notación, podemos indicar el problema de coincidencia de cadenas como el de encontrar todos los cambios en el rango $0 \leq s \leq n - m$ tal que $P \sqsupset T_{s+m}$.

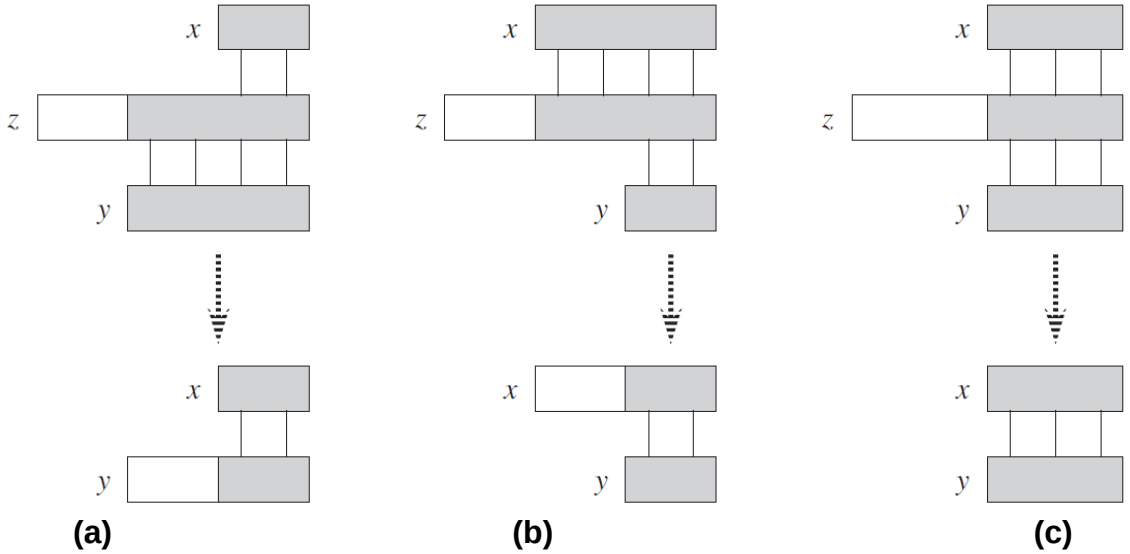


Figura 2.4.2 (**a**) $|x| \leq |y|$, (**b**) $|x| \geq |y|$, (**c**) $|x| = |y|$

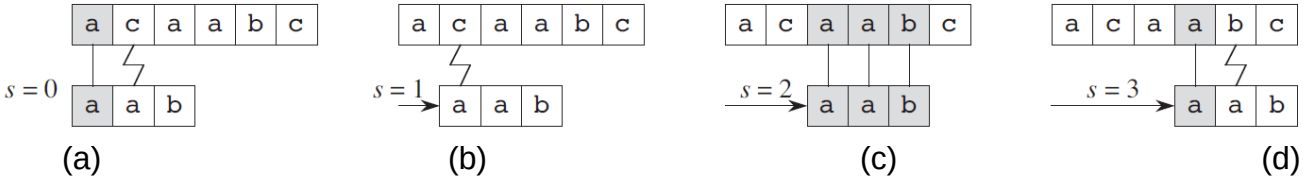
2.4.1 Algoritmo ingenuo

En seudocódigo se permite comparar dos cadenas de la misma longitud para ver si son iguales como una operación primitiva. Si las cadenas son comparadas de izquierda a derecha y la comparación se detiene cuando se descubre que son iguales, se asume que el tiempo que toma para esta prueba es una función linear del número de caracteres que tienen que compararse. Para precisar, la prueba $x = = y$ se asume que toma un tiempo$\Theta(t + 1)$, donde *t* es la longitud de la cadena *z* tal que $z \sqsubset x$ y $z \sqsubset y$.

El algoritmo ingenuo encuentra todos los cambios válidos utilizando un ciclo con la condición de verificación $P[1 \ldots m] = T[s + 1 \ldots s + m]$ por cada uno de los $n - m + 1$ posibles valores de *s*. El procedimiento de igualación de cadenas de este algoritmo se muestra a continuación:

```
funcion encontrarCadenaAlgoritmoIngenuo(T, P)
        n = len(T)
        m = len(P)
        for s = 0 to n-m
                if P[1 ... m] == T[s + 1 ... s + m]
                        imprimir "Ocurrencia de patrón con cambio: " s
```

La figura 2.4.3 muestra cómo se lleva a cabo este algoritmo



(a) (b) (c) (d)

Figura 2.4.3 Igualación de cadenas por algoritmo ingenuo paso a paso.

Como puede ver este algoritmo no está optimizado para este problema

2.4.2 Algoritmo con autómata finito

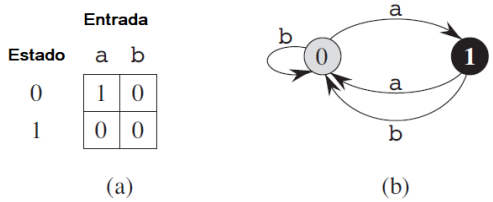
Muchos algoritmos de coincidencia de cadenas crean un autómata finito, una máquina simple para procesamiento de información que escanea la cadena de texto *T* para todas las apariciones del patrón *P*. Estos los autómatas de coincidencia de cadenas son muy eficientes, examinan cada carácter de texto exactamente una vez, esto toma un tiempo constante por carácter de texto. El tiempo de coincidencia utilizado, después pre procesar el patrón para construir el autómata, por lo tanto, es $\Theta(m)$. El tiempo sin embargo, para construir el autómata puede ser grande si Σ es grande.

Comenzamos con la definición de un autómata finito. Luego se examina un autómata especial de coincidencia de cadenas y se muestra cómo usarlo para encontrar ocurrencias de un patrón en un texto. Finalmente, se muestra cómo construir la coincidencia de cadenas autómata para un patrón de entrada dado.

Autómata finito

Un autómata finito M, como el que se muestra en la figura 2.4.2.1, es una quíntupla (5-tupla) $(Q, q_0, A, \Sigma, \delta)$ donde

- Q es un conjunto finito de estados
- $q_0 \in Q$ Es el estado inicial
- $A \subseteq Q$ es un conjunto distinguido de estados de aceptación
- Σ es un alfabeto de entrada
- δ es una función desde *Q* x Σ en *Q*, denominada función de transición de *M*.



(a) (b)

Figura 2.4.2.1 Autómata de estados finitos (a) Representación tabular, (b) Representación gráfica

Para el autómata de la figura anterior tenemos $(Q, q_0, A, \Sigma, \delta)$ donde:

- Q = {0,1} Los estados
- $q_0 = 1$ Es el estado inicial
- $A \subseteq Q$ es un conjunto distinguido de estados de aceptación
- $\Sigma = \{\mathrm{a}, \mathrm{b}\}$ el alfabeto de entrada
- $\delta(0, a) = 1, \delta(0, b) = 0, \delta(1, a) = 0\ \delta(1, b) = 0,$ está representada por la tabla (a).

En el estado de inicio (1), con entrada *a* o *b* pasa al estado (0), en el estado (0) con entrada *b* se queda aquí y en el estado (0) con entrada *a* regresa al estado (1). Este autómata acepta las cadenas que terminan en un número impar de *a*. Más precisamente, acepta una cadena *x* si y solo si $x = yz$, donde $y = \varepsilon$ "o *y* termina con *a b*", y $z = a^k$, donde *k* es impar. Por ejemplo, en la entrada abaaa, incluido el estado de inicio, este autómata entra en la secuencia de estados (0, 1, 0, 1, 0, 1), por lo que acepta esta entrada. Para la entrada abbaa, ingresa la secuencia de estados (0, 1, 0, 0, 1, 0), por lo que rechaza esta entrada.

El autómata finito comienza en el estado q_0 y lee los caracteres de su cadena de entrada, uno a la vez. Si el autómata está en estado *q* y lee el carácter de entrada *a*, se mueve (hace una transición) del estado *q* al estado $\delta(q, a)$. Siempre que su estado actual *q* sea miembro de *A*, la máquina *M* ha aceptado la cadena leída hasta ahora. Una entrada que No es aceptada es por lo tanto rechazada.

Un autómata finito *M* induce una función ϕ, llamada función de estado final, de Σ^* a Q tal que $\phi(w)$ está en el estado *M* y termina después de escanear la cadena *w*. Por lo tanto, *M* acepta una cadena *w* si y solo si $\phi(w) \in A$. Definimos la función recursivamente, utilizando la función de transición:

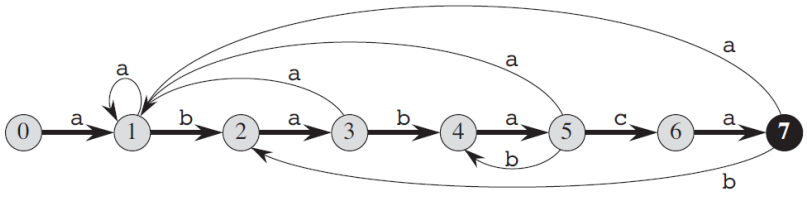
$$\phi(\varepsilon) = q_0,$$
$$\phi(\varepsilon wa) = \delta(\phi(w), a)\ para\ w \in \Sigma^*, a \in \Sigma$$

Autómata de igualación de cadenas

Para un patrón *P* dado, construimos un autómata de coincidencia de cadenas en un paso de pre procesamiento antes de usarlo para buscar la cadena de texto. La figura 2.4.2.2 ilustra cómo construir el autómata para el patrón $P = \mathrm{ababaca}$. Suponga que *P* es una cadena de patrón fijo dada; por brevedad, no se indicará la dependencia de *P* en la notación.

Para especificar el autómata de coincidencia de cadenas correspondiente a un patrón dado $P[1 \ldots m]$, primero definimos una función σ auxiliar, llamada función de sufijo correspondiente a *P*. La función σ mapea Σ^* $a\ \{0,1, n, \ldots, n\}$ tal que $\sigma(x)$ es la longitud del prefijo más largo de *P* que también es un sufijo de x:

$$\sigma(x) = \max\ \{k : P_k \sqsupset x\} \qquad (3)$$



Estado	Entrada a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

	i	—	1	2	3	4	5	6	7	8	9	10	11
	$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
Estado	$\phi(T_i)$	0	1	2	3	4	5	4	5	6	**7**	2	3

Figura 2.4.2.2 Autómata de igualación de cadenas

De la figura anterior, el gráfico arriba es el diagrama de transición entre estados, este acepta patrones de cadenas como ababaca, el estado 0 es el estado inicial y el estado 7 (en negro) es un estado de aceptación. *A* es dirige del estado *i* al estado *j* etiquetado *a* y representa $\delta(i, a) = j$. A la izquierda está la tabla de transición entre estados para una entrada dada. Abajo a la derecha se tiene la operación del autómata para el texto $T = abababacaba$. Debajo de cada carácter del texto *T*[*i*] aparece el estado $\phi(T_i)$ tal que el autómata está procesando para el prefijo T_i. El autómata encuentra una ocurrencia para el patrón y termina en la posición 9.

El sufijo de la función σ está bien definido para la cadena vacía $P_0 = \varepsilon$ que también es sufijo de cualquier cadena. Cuando se define un autómata de igualación de cadenas que corresponda a un patrón dado $P[1 \ldots m]$ como sigue a continuación:

- Se incluye el estado *Q* que es $\{0, 1, 2, \ldots, m\}$. Se establece el estado inicial q_0 como estado 0, y el estado *m* es el estado de aceptación.
- La función de transición δ está definida por la siguiente ecuación, para cualquier estado y carácter:

$$\delta(q, a) = \sigma(P_q a) \qquad (4)$$

Definimos $\delta(q, a) = \sigma(P_q a)$ porque queremos hacer seguimiento del prefijo más grande del patrón *P* que ha igualado en el texto *T*. Considere la más reciente lectura de caracteres en T. En orden para una subcadena *T* (decimos que el final de la subcadena termina en $T[i]$) para igualar algún prefijo P_j de *P*, este prefijo P_j será un sufijo de T_i. Suponga que $q = \phi(T_i)$, tal que después de leer T_i, el autómata está en el estado q, podemos diseñar la función de transición δ tal que el número de estado q, le diga la longitud del prefijo más grande de P que iguale un sufijo de T_i. Esto es, en el estado q, $P_q \sqsupset T_i$ y $q = \sigma(T_i)$, (cualquier $q = m$ para todos lo *m* caracteres de *P* que igualen un sufijo de T_i y así hasta encontrar la igualdad). Así desde $\phi(T_i)$ y $\sigma(T_i)$ ambos igual a *q*, decimos que el autómata mantiene el siguiente invariante:

$$\phi(T_i) = \sigma(T_i) \qquad (5)$$

Si el autómata está en el estado *q* y lee el siguiente carácter $T[i + 1] = a$, entonces queremos llevarlo al estado correspondiente al prefijo más grande de P tal que sea un sufijo de $T_i a$, y ese estado es

$\sigma(T_i a)$. Porque P_q es el prefijo más grande de *P* tal que es un sufijo de T_i, el prefijo más grande de *P* tal que es un sufijo de $T_i a$ no solo es $\sigma(T_i a)$, pero si $\sigma(P_q a)$. Por lo tanto, cuando el autómata está en el estado *q*, queremos que la función de transición en el carácter *a* para llevar el autómata a estado $\sigma(P_q a)$.

Hay dos casos a considerar:

- En el primer caso, $a = P[q+1]$, de modo que el carácter *a* sigue coincidiendo con el patrón; en este caso, porque $\delta(q, a) = q + 1$, en la transición continúa a lo largo de la "columna vertebral" del autómata (los bordes pesados en figura 2.4.2.2).
- En el segundo caso, $a \neq P[q+1]$, en donde no continúa coincidiendo el patrón. Aquí, debemos encontrar un prefijo más pequeño de *P* que también sea un sufijo de T_i. Debido a que el paso de preprocesamiento coincide del patrón contra sí mismo al crear el autómata de coincidencia de cadenas, la función de transición identifica rápidamente el más largo prefijo contra el más pequeño de *P*.

Veamos un ejemplo. El autómata de coincidencia de cadenas de la figura 2.4.2.2 tiene $\delta(5, c) = 6$, que ilustra el primer caso, en el que el partido continúa. Para ilustrar el segundo caso, observe que el autómata de la figura 2.4.2.2 tiene $\delta(5, b) = 4$. Hacemos esta transición porque si el autómata lee a *b* en el estado $q = 5$, entonces $P_q b = ababab$, y el prefijo más grande de *P* que también es sufijo de ababab es$P_4 = abab$.

Para aclarar el funcionamiento de un autómata de coincidencia de cadenas, ahora damos un simple, algoritmo eficiente para simular el comportamiento de dicho autómata (representado por su función de transición δ) al encontrar ocurrencias de un patrón *P* de longitud *m* en un entrada de texto $T[1 \ldots n]$. En cuanto a cualquier autómata de coincidencia de cadenas para un patrón de longitud *m*, el conjunto de estados *Q* es $\{0, 1, 2, \ldots., m\}$, el estado inicial es 0, y el único estado de aceptación es el estado m.

Algoritmo

```
funcion igualarCadenaConAutomataFinito( T, δ, m )
    n = len(T)
    q = 0
    for i = 1 to n
        q = δ(q, T[i])
        if q == m
            imprimir "El patron ocurrio en el cambio " i − m
```

Desde la estructura del ciclo simple de igualarCadenaConAutomataFinito(), podemos fácilmente ver que su tiempo de coincidencia en una cadena de texto de longitud *n* es $\Theta(n)$. Esta coincidencia el tiempo, sin embargo, no incluye el tiempo de pre procesamiento requerido para calcular la función de transición δ. Para probar que el procedimiento igualarCadenaConAutomataFinito() funciona correctamente, considere cómo funciona el autómata con un texto de entrada $T[1 \ldots n]$. Probaremos que el autómata está en estado $\sigma(T_i)$ después de escanear el carácter $T[i]$. Desde $\sigma(T_i) = m$ si y solo si $P \sqsupset T_i$, la máquina está en el estado de aceptación *m* si y solo si tiene solo escaneado el patrón *P*. Para probar este resultado, utilizamos los siguientes dos Lemas sobre la función sufijo σ.

Lema 2.4.2 (Desigualdad de función de sufijo)

Para cualquier cadena *x* y carácter *a*, tenemos $\sigma(xa) \leq \sigma(x) + 1$.

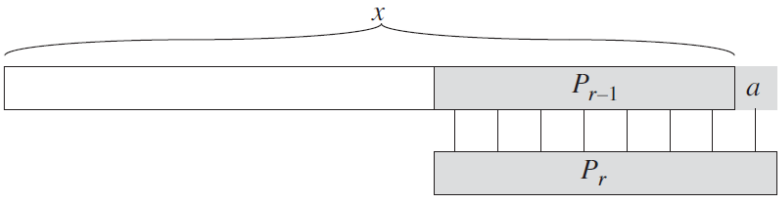


Figura 2.4.2.3 Una ilustración para la prueba de Lemma 2.4.2. La figura muestra que $r \leq \sigma(x) + 1$, donde $r = \sigma(xa)$.

Prueba:

Refiriéndose a la figura 2.4.2.3, sea $r = \sigma(xa)$. Si $r = 0$, entonces la conclusión $\sigma(xa) = r \leq \sigma(x) + 1$ se satisface trivialmente, por la no negatividad de $\sigma(x)$. Ahora supongamos que $r > 0$.

Entonces, $P_r \sqsupset xa$, por la definición de σ. Por lo tanto, $P_{r-1} \sqsupset x$, dejando caer la *a* desde el final de P_r y desde el final de xa. Por lo tanto, $r - 1 \leq \sigma(x)$, dado que $\sigma(x)$ es la *k* más grande tal que $P_k \sqsupset x$, y por lo tanto $\sigma(x) = r \leq \sigma(x) + 1$.

Lema 2.4.3 (lema de recursión con función de sufijo)

Para cualquier cadena *x* y carácter *a*, si $q = \sigma(x)$, entonces $\sigma(xa) = \sigma(P_q a)$.

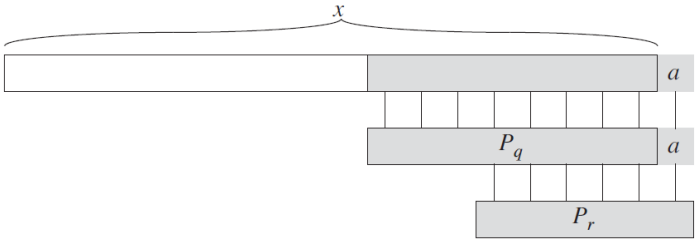


Figura 2.4.2.4 Una ilustración de la prueba del Lema 2.4.3. La figura muestra que $r = \sigma(P_q a)$, donde $q = \sigma(x)$ y $r = \sigma(xa)$.

Prueba:

De la definición de σ, tenemos $P_q \sqsupset x$. Como muestra la figura 2.4.2.4, también tienen $P_q a \sqsupset xa$. Si dejamos $r = \sigma(xa)$, entonces $P_r \sqsupset xa$ y, según el Lema 2.4.2, $r \leq q + 1$. Por lo tanto, tenemos $|P_r| = r \leq q + 1 = [P_q a]$. Desde $P_q a \sqsupset xa$, $P_r \sqsupset xa$, y $|P_r| = r \leq q + 1 = |P_q a|$, El Lema 2.4.1 implica que $P_r \sqsupset P_q a$. Por lo tanto, $r \leq \sigma(P_q a)$, es decir, $\sigma(xa) \leq \sigma(P_q a)$. Pero también tenemos $\sigma(P_q a) \leq \sigma(xa)$, ya que $P_q a \sqsupset xa$. Por lo tanto, $\sigma(xa) = \sigma(P_q a)$.

Ahora estamos listos para demostrar nuestro teorema principal que caracteriza el comportamiento de un autómata de coincidencia de cadenas para un texto de entrada dado. Como se señaló anteriormente, este teorema muestra que el autómata simplemente realiza un seguimiento, en cada paso, del más grande prefijo del patrón que es un sufijo de lo que se ha leído hasta ahora. En otras palabras, el autómata mantiene el invariante.

Teorema 4.1

Si ϕ es la función de estado final de un autómata de coincidencia de cadenas para un patrón dado *P* y $T[1 \ldots n]$ es un texto de entrada para el autómata, luego entonces

$$\phi(T_i) = \sigma(T_i)$$

Para $i = 0, 1, 2, \ldots, n$.

Prueba

La prueba es por inducción en *i*. Para $i = 0$, el teorema es trivialmente verdadero, desde $T_0 = \varepsilon$. Por lo tanto, $\phi(T_0) = 0 = \sigma(T_0)$.

Ahora, asuma que $\phi(T_i) = \sigma(T_i)$ y probaremos que $\phi(T_{i+1}) = \sigma(T_{i+1})$. Sea *q* denote $\phi(T_i)$, y sea *a* que denote $T[i+1]$. Entonces,

$\phi(T_{i+1}) = \phi(T_i a)$	(por las definiciones de T_{i+1} y *a*)
$\phi(T_{i+1}) = \delta(\phi(T_i),\ a)$	(por la definición de ϕ)
$\phi(T_{i+1}) = \delta(q, a)$	(por la definición de *q*)
$\phi(T_{i+1}) = \sigma(P_q a)$	(por la definición (4.1) de δ)
$\phi(T_{i+1}) = \sigma(T_i a)$	(por Lema 2.4.3 e inducción)
$\phi(T_{i+1}) = \sigma(T_{i+1})$	(por la definición de T_{i+1}).

Según el teorema 4.1, si la máquina ingresa al estado *q* en la línea 4, entonces *q* es el más grande valor tal que $P_q \sqsupset T_i$. Por lo tanto, tenemos $q = m$ en la línea 5 si y solo si la máquina acaba de escanear una aparición del patrón *P*. Concluimos que igualarCadenaConAutomataFinito() funciona correctamente.

Calculando la función de transición δ

El siguiente procedimiento calcula la función de transición ı a partir de un patrón dado $P[1 \ldots m]$.

Algoritmo

```
funcion calcularFuncionDeTransicion(P, Σ)
    m = len(P)
    for q = 0 to m
        for each caracter a ∈ Σ
            k = min(m + 1, q + 2)
                repetir
                    k = k + 1
                hasta P_k ⊐ p_q a
                δ(q, a) = k
    return δ
```

Este procedimiento calcula $\delta(q, a)$ de forma directa según su definición en la ecuación (4). Los ciclos anidados que comienzan en las líneas 2 y 3 consideran todos los estados *q* y todos los caracteres *a*, y las líneas 4–8 establecen $\delta(q, a)$ a ser el mayor *k* tal que$P_k \sqsupset P_q a$. El código comienza con el mayor valor concebible de *k*, que es $min(m, q + 1)$. Luego disminuye *k* hasta $P_k \sqsupset P_q a$, que eventualmente debe ocurrir, ya que $P_0 = \varepsilon$ es un sufijo de cada cadena

El tiempo de ejecución de calcularFuncionDeTransicion() es $O(m^3|\Sigma|)$, porque los ciclos externos contribuyen con un factor de $m|\Sigma|$, el ciclo interno de repetición puede ejecutarse a lo sumo $m+1$ veces, y la prueba $P_k \sqsupset P_q a$ en la línea 7 puede requerir comparar hasta m caracteres. Existen procedimientos mucho más rápidos; que utilizan algunos cálculos ingeniosos sobre la información contenida en el patrón *P*, se puede mejorar el tiempo requerido para calcular δ de *P* a $O(m|\Sigma|)$. Con este procedimiento mejorado para calcular δ, podemos encontrar todas las apariciones de un patrón de longitud *m* en un texto de longitud *n* sobre un alfabeto Σ con $O(m|\Sigma|)$ tiempo de preprocesamiento $\Theta(n)$ tiempo de coincidencia.

2.4.3 Algoritmo de Knuth-Morris-Pratt

Ahora presentamos un algoritmo de coincidencia de cadenas de tiempo lineal debido a sus autores Knuth, Morris y Pratt. Este algoritmo evita calcular la función de transición δ por completo, y su el tiempo de coincidencia es $\Theta(n)$ usando solo una función auxiliar π, que calcula previamente del patrón en un tiempo $\Theta(m)$ y almacena en una matriz $\pi[1 \ldots m]$. La matriz π nos permite calcular la función de transición δ eficientemente (en un sentido amortizado) "al vuelo" según sea necesario. Hablando libremente, para cualquier estado $q = 0, 1, 2, \ldots, m$ y cualquier carácter $a \in \Sigma$, el valor $\pi[q]$ contiene la información que necesitamos para calcular $\delta(q, a)$ pero eso no depende de *a*. Dado que la matriz tiene solo *m* entradas, mientras que tenemos $\Theta(m|\Sigma|)$ entradas, ahorramos un factor de $|\Sigma|$ en el tiempo de pre procesamiento calculando π en lugar de δ.

La función de prefijo para un patrón

La función de prefijo para un patrón encapsula el conocimiento sobre cómo el patrón partido contra turnos de sí mismo. Se puede aprovechar esta información para evitar probar los cambios inútiles del algoritmo ingenuo de coincidencia de patrones y evitar pre calcular la función de transición completa δ para un autómata de coincidencia de cadenas. Considere la operación del algoritmo ingenuo combinador de cuerdas. La figura 2.4.2.5 en su parte superior muestra un turno particular de una plantilla que contiene el patrón $P = \text{ababaca}$ contra un texto *T*. Para este ejemplo, $q = 5$ de los caracteres que coinciden correctamente, pero el sexto carácter del patrón no coincide con el carácter de texto correspondiente. La información de que q caracteres coincidan con éxito determina los correspondientes caracteres de texto. Sabiendo que estos caracteres de texto q nos permite determinar de inmediato que ciertos turnos no son válidos. En el ejemplo de la figura, el cambio $s+1$ es necesariamente inválido, ya que el primer carácter de patrón (en este ejemplo) estaría alineado con un texto el carácter que sabemos que no coincide con el primer carácter del patrón, pero sí coincide el segundo carácter de patrón (mostrado en la parte media de la figura). El cambio $s' = s+2$ que se muestra en la parte (media) de la figura, sin embargo, alinea los primeros tres caracteres del patrón con tres caracteres de texto que necesariamente debe coincidir. En general, es útil saber la respuesta a lo siguiente pregunta:

Dado que los caracteres del patrón $P[1 \ldots q]$ coinciden con los caracteres de texto $T[s+1 \ldots s+q]$, ¿Cuál es el menor cambio $s' > s$ tal que para algunos $k < q$,

$$P[1 \ldots k] = T[s'+1 \ldots s'+k] \qquad (6)$$

donde $s'+k = s+q$?

En otras palabras, sabiendo que $P_q \sqsupset T_{s+q}$, queremos el prefijo apropiado más largo P_k de P_q que también es un sufijo deT_{s+q}. (Dado que $s' + k = s + q$, si se nos da un *s* y un *q*, entonces encontrar el turno más pequeño s' equivale a encontrar el prefijo más largo longitud *k*). Agregando la diferencia $q - k$ en las longitudes de los prefijos de *P* a los cambios de *s* para llegar a nuestro nuevo cambio s', de modo que $s' = s + (q - k)$. En el mejor de los casos, $k = 0$, entonces $s' = s + q$, e inmediatamente descartamos los cambios $s + 1$, $s + 2$, ... $s + q - 1$. En cualquier caso, en el nuevo turno s' no necesitamos comparar los primeros *k* caracteres de *P* con los caracteres correspondientes de *T*, ya que la ecuación (6) garantiza el emparejamiento.

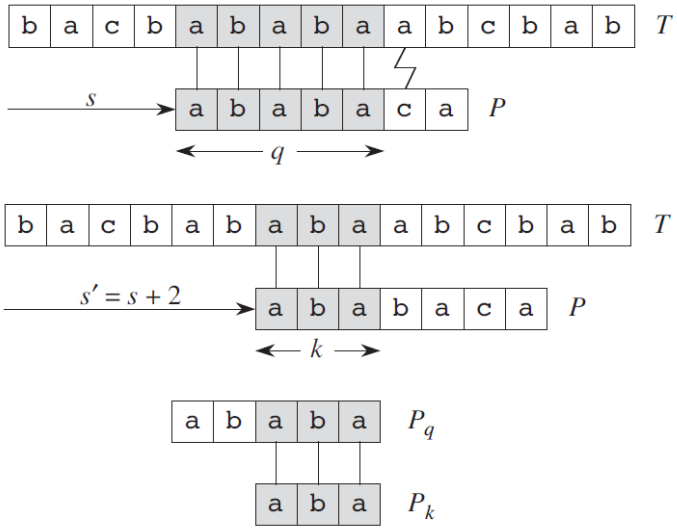


Figura 2.4.2.5 La función prefijo π. Arriba el patrón $P = ababaca$ alineado con un texto T tal que $q = 5$ caracteres de emparejamiento. Empleando el conocimiento de igualar 5 caracteres, se puede deducir que el cambio $s + 1$ no es válido. Abajo podemos pre calcular por comparación empleando la información del mismo patrón.

Podemos pre calcular la información necesaria comparando el patrón con en sí mismo, como lo demuestra la parte baja de la figura 2.4.2.5. Como $T[s' + 1 \ldots s' + k]$ es una parte de la parte conocida del texto, es un sufijo de la cadena P_q. Por lo tanto, podemos interpretar ecuación (6) como pedir la mayor $k < q$ tal que $P_k \sqsupset P_q$. Entonces, lo nuevo cambio $s' = s + (q - k)$ es el siguiente cambio potencialmente válido. Nos resultará conveniente almacenar, para cada valor de *q*, el número *k* de caracteres coincidentes en el nuevo turno s', en lugar de almacenar, digamos, $s' - s$.

Formalizamos la información que calculamos de la siguiente forma:

- Dado un patrón $P[1 \ldots m]$, la función de prefijo para el patrón *P* es la función $\pi: \{1,2,3, \ldots m\} \longrightarrow \{0, 1, 2, \ldots, m - 1\}$ tal que
  - $\pi[q] = max\{k: k < q \; y \; p_k \sqsupset P_q\}$
- Es decir, $\pi[q]$ es la longitud del prefijo más largo de *P* que es un sufijo apropiado de P_q. La figura 2.4.2.6 (la izquierda) proporciona la función de prefijo completo para el patrón ababaca.

El pseudocódigo siguiente proporciona el algoritmo de coincidencia Knuth-Morris-Pratt denominado igualarCadenaConKMP() que llama a la función o procedimiento auxiliar calcularFuncionPrefijo() para calcular la función prefijo.

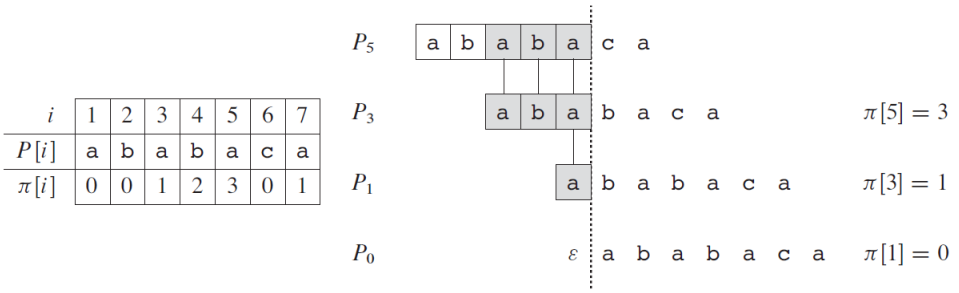


Figura 2.4.2.6 Una ilustración de igualación de cadena para el patrón $P = ababaca\ y\ q = 5$. A la izquierda la función π para el patrón dado. A la derecha, los cambios de la plantilla para el patrón *P* dado.

```
funcion igualarCadenaConKMP(T, P)
    n = len(T)
    m = len(P)
    π = calcularFuncionPrefijo(P)
    q = 0                                   // numero de caracteres a igualar
    for i = 1 to n                          // escanea el texto de izquierda a derecha
        while q > 0 y P[q + 1] ≠ T[i]
            q = π[q]                        // siguiente carácter que no iguala
        if P[q + 1] == T[i]
            q = q + 1                       // siguiente carácter que iguala
        if q == m                           // si todos son iguales
            imprimir "Ocurrencia de patron en: " i − m
            q = π[q]                        // va a la siguiente busqueda

funcion calcularFuncionPrefijo(P)
    m = len(P)
    let π[1 ... m]          // es un arreglo nuevo
    π[1] = 0
    k = 0
    for q = 2 to n
        while k > 0 and P[k + 1] ≠ P[q]
            k = π[k]
        if P[k + 1] == P[q]
            k = k + 1
        π[q] = k
    return π
```

Estos dos procedimientos tienen mucho en común, porque ambos coinciden en una cadena contra el patrón *P*: igualarCadenaConKMP() coincide con el texto *T* contra *P* y calcularFuncionPrefijo() coincide con *P* contra sí mismo.

Comenzamos con un análisis de los tiempos de ejecución de estos procedimientos. La prueba de estos procedimientos correctos es más complicada.

2.4.4 Algoritmo Rabin-Karp

Los autores Rabin y Karp propusieron un algoritmo de igualación de cadenas que trabaja mejor en la práctica y se ha generalizado para resolver otros problemas como la igualación de patrones en dos

dimensiones. Este algoritmo emplea un tiempo de pre procesamiento de $\Theta(m)$ y que en el peor de los casos el tiempo de ejecución será de $\Theta((n-m+1)m)$. Basados en algunas aseveraciones, este es un caso en donde se puede mejorar en un porcentaje el tiempo de ejecución.

Este algoritmo emplea nociones de teoría de números como la equivalencia del módulo de dos números a un tercero. Asuma que $\Sigma = \{0,1,2,\ldots\ldots,9\}$, dígitos decimales cuya notación base-d donde $d = |\Sigma|$. Ahora vea que una cadena de *k* caracteres consecutivos lo que representa una cadena de longitud de numero decimal *k*. La cadena de caracteres 31415 corresponde al número decimal 31,415. Porque la interpretación de caracteres como entrada pueden ser símbolos gráficos y dígitos.

Dado un patrón $P[1 \ldots m]$, denotemos p como su correspondiente valor decimal. De igual forma dado un texto $T[1 \ldots n]$, dado t_s denota el valor decimal de longitud m una subcadena $T[s+1 \ldots s+m]$, para $s = 0,1,2,\ldots, n-m$. Ciertamente $t_s = p$ si y solo si $T[s+1 \ldots s+m] = P[1 \ldots m]$, esto es, *s* es un cambio válido si y solo si $t_s = p$. Si calculamos *p* en un tiempo $\Theta(m) + \Theta(n-m+1) = \Theta(m)$ por comparar p con cada uno de los t_s valores (por el momento, no importa la posibilidad de que p y los valores t_s sean números muy grandes). Podemos calcular p en un tiempo $\Theta(m)$ utilizando la regla de Horner:

$$p = P[m] + 10\big(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\ldots)\big)$$

Similarmente, podemos calcular t_0 desde $T[1 \ldots m]$ en un tiempo $\Theta(m)$. Para calcular los demás valores $t_1, t_2, t_3, \ldots\ldots t_{n-m}$ en un tiempo $\Theta(n-m)$, observe que se puede calcular t_{s+1} desde t_s en un tiempo constante de la forma

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1] \qquad (1)$$

La sustracción de $10^{m-1}T[s+1]$ remueve los dígitos de orden alto de t_s, multiplicando el resultado por 10 cambia el numero izquierdo por un digito de posición, y sumando $T[s+m+1]$ salta apropiadamente al digito más bajo. Poe ejemplo si $m = 5$ y $t_s = 31415$, si deseamos remover el digito más alto $T[s+1] = 3$, saltamos a un nuevo dígito de orden más bajo, suponga $T[s+5+1] = 2$, se obtiene

$$t_{s+1} = 10(31415 - 10000 \cdot 3) + 2$$
$$t_{s+1} = 14152$$

Si pre calculamos la constante 10^{m-1} (podemos lograr un tiempo $O\big(lg(m)\big)$), así que también podemos calcular p en un tiempo $\Theta(m)$ y podemos calcular todos los $t_0, t_1, t_2, \ldots\ldots t_{n-m}$ en un tiempo $\Theta(n-m+1)$. Así mismo podemos encontrar todas la ocurrencias del patrón $P[1 \ldots m]$ en el texto $T[1 \ldots n]$ con $\Theta(m)$ tiempo de pre procesamiento y $\Theta(n-m+1)$ tiempo de empate o igualación.

Hasta ahora, hemos pasado por alto un problema intencionalmente: *p* y t_s pueden ser demasiado grandes para trabajar convenientemente. Si *P* contiene *m* caracteres, entonces no podemos asumir razonablemente que cada operación aritmética en *p* (que tiene *m* dígitos de longitud) tomará un "tiempo constante". Afortunadamente, se puede resolver este problema fácilmente, como se muestra en la Figura 2.4.4 en ella se calcula *p* y los valores de t_s para el módulo adecuado de **q**. Si elegimos el módulo *q* como primo de tal manera que 10*q* solo cabe dentro de una palabra de computadora, entonces podemos realizar todos los cálculos necesarios con precisión simple aritmética. En general, con alfabeto d $\{1,2,3,\ldots,d\}$ elegimos *q* tal que *dq* cabe dentro de una palabra de computadora y se ajuste a la ecuación de recurrencia (1) para trabajar con módulo q de tal forma que:

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1])\ mod\ q \qquad (2)$$

Donde $h \equiv d^{m-1}$, (mod q) es el valor del digito “1” en la posición de orden más alto de un digito m de la ventana de texto.

La solución de trabajar con el modulo q no es perfecta, sin embargo $t_s \equiv p \ (mod\ q)$ no implica que $t_s = p$. De otra forma, si $t_s \neq p \ (mod\ q)$ entonces definitivamente $t_s \neq p$, así que el cambio en *s* no es válido. Así que podemos usar la prueba $t_s \equiv p \ (mod\ q)$ como prueba heurística rápida para descartar turnos no válidos en *s*. Cualquier cambio en s para qué $t_s \equiv p \ (mod\ q)$ debe probarse más para ver si *s* es realmente válido o solo tenemos un golpe espurio. Esta prueba adicional verifica explícitamente la condición $P[1 \ldots m] = T[s+1 \ldots s+m]$. Si *q* es lo suficientemente grande, entonces esperamos que los golpes (espurios) ocurran con poca frecuencia como para que el costo de la verificación adicional sea bajo.

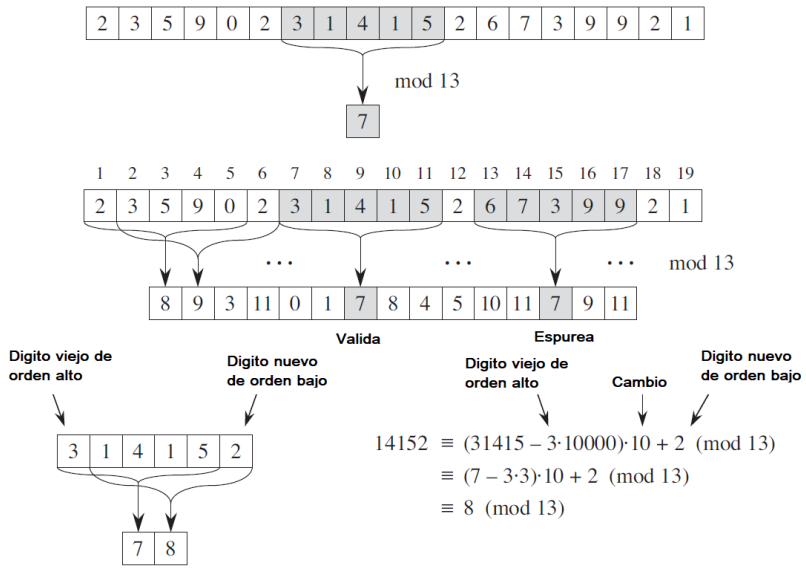


Figura 2.4.4 Algoritmo Rabin-Karp

El siguiente procedimiento hace que estas ideas sean precisas. Las entradas al procedimiento. son el texto *T*, el patrón *P*, la raíz *d* a usar (que generalmente se toma como |Σ|), y el primer *q* para usar.

Algoritmo

función ejecutarAlgoritmoRabinKarp(*T*, *P*, *d*, *q*)

n = len(*T*)
m = len(*P*)
$h = d^{m-1} mod\ q$
p = 0
$t_0 = 0$
for i=1 **to** m // **preprocesamiento**
$p = (dp + P[i]) \bmod q$
$t_0 = (dt_0 + T[i]) \bmod q$
for *s* = 0 **to** *n* – *m*
if $p == t_s$
if $P[1 \ldots m] == T[s+1 \ldots s+m]$
imprimir “Ocurrencia de patrón con cambio: ” *s*
if $s < n - m$
$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

TAREA:

Esta tarea solo servirá a aquellas personas que quieran subir calificación del primer departamental (calificado por el Dr. Flavio), cada inciso vale un punto sobre su calificación final, el último inciso solo cuenta si se programa el inciso 2 y el 3. Todo deberá ser programado en lenguaje Java y orientado a objetos.

1. Programar emparejamiento de cadenas empleando el algoritmo ingenuo.
2. Programar emparejamiento de cadenas empleando el algoritmo de autómata finito.
3. Programar emparejamiento de cadenas empleando el algoritmo Knuth, Morris y Pratt.
4. Programar emparejamiento de cadenas empleando el algoritmo Rabin-Karp.
5. Buscar el código de las clases string de C++, String de Java y verificar que algoritmo emplean para el emparejamiento de cadenas. Buscar el algoritmo en todas las funciones que tiene cada clase y reportar donde está añadiendo el código del método explicando el porque.

Hay que entregar el código fuente programado y reportar el punto 5 con sus conclusiones.