



# *Precondicionamiento y reconocimiento de patrones*

❖ Introducción	2
❖ Antecesoros en un árbol	9
❖ Evaluación repetida de un polinomio	12
❖ Reconocimiento de patrones	18
– Método directo	18
– Uso de firmas	21
– El algoritmo de Knuth, Morris y Pratt	26
– El algoritmo de Boyer y Moore	39



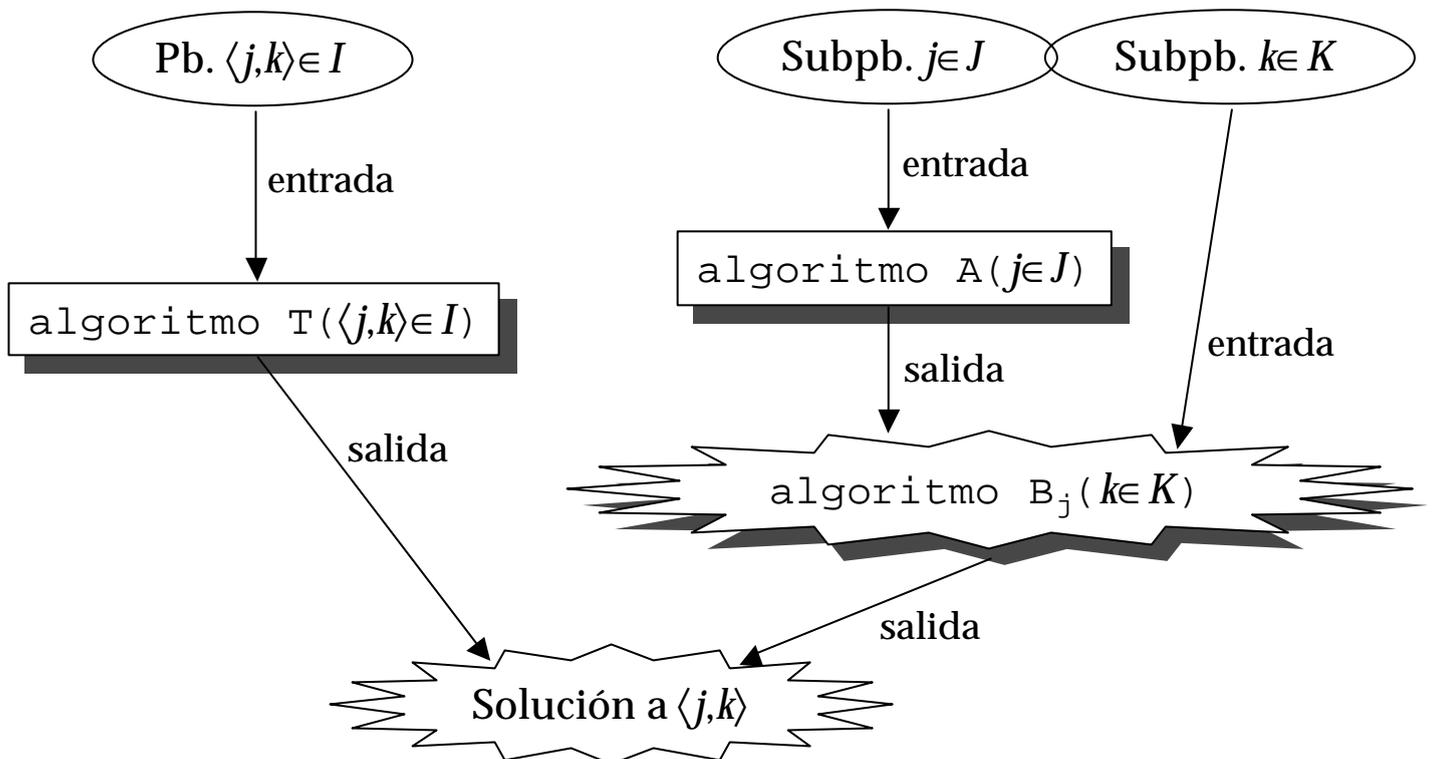
# *Precondicionamiento: Introducción*

- ❖ Si hay que **resolver muchos ejemplares de un mismo problema** merece la pena invertir un **esfuerzo inicial** para calcular resultados auxiliares que **acelerarán la solución de cada ejemplar.**
- ❖ Incluso si hay que resolver un sólo ejemplar del problema, esta técnica puede conducir a un algoritmo más eficiente.

# Precondicionamiento: Introducción

## ❖ De forma más precisa:

- Sea  $I = \{ \text{ejemplares de un problema dado} \}$
- Suponer que cada  $i \in I$  se puede descomponer en dos subproblemas  $j \in J$  y  $k \in K$  (es decir,  $I \subseteq J \times K$ ).
- Un algoritmo de precondicionamiento para  $I$  es un algoritmo  $A$  que tiene como entrada un elemento  $j$  de  $J$ , y como salida un nuevo algoritmo  $B_j$ , tal que si  $k \in K$  y  $\langle j, k \rangle \in I$  entonces  $B_j$  aplicado a  $k$  da la solución de  $\langle j, k \rangle$ .





# Precondicionamiento: Introducción

## ❖ Ejemplo:

- $J$  es un conjunto de gramáticas en forma normal de Backus para una familia de lenguajes de programación (Algol, Pascal, Simula, ...).
- $K$  es un conjunto de programas.

- Familia de problemas:

Saber si un programa dado es sintácticamente correcto en un lenguaje dado.

Es decir,

$$I = \{ \text{“¿es } k \in K \text{ un programa en el lenguaje de programación definido por la gramática } j \in J? \text{”} \}$$

- Algoritmo de precondicionamiento:

$A$  = generador de reconocedores:

aplicado sobre la gramática  $j \in J$   
genera un reconocedor  $B_j$ .

- Para resolver el problema original, basta con aplicar el reconocedor  $B_j$  sobre  $k$ .



# Precondicionamiento: Introducción

## ❖ Consideraciones sobre la eficiencia:

– Sean:

$a(j)$  = tiempo para producir  $B_j$  a partir de  $j$

$b_j(k)$  = tiempo para aplicar  $B_j$  a  $k$

$t(j,k)$  = tiempo para resolver  $\langle j,k \rangle$  directamente

– El precondicionamiento tiene sentido si:

$$\underline{b_j(k) \leq t(j,k) \leq a(j) + b_j(k)}$$

(obviamente, no tiene sentido si:  $b_j(k) > t(j,k)$  )



# Precondicionamiento: Introducción

## ❖ Utilidad:

- Si hay que resolver varios problemas  $\langle j, k_1 \rangle, \langle j, k_2 \rangle, \dots, \langle j, k_n \rangle$  con un mismo  $j$ :

$$a(j) + \sum_{i=1}^n b_j(k_i) \ll \sum_{i=1}^n t(j, k_i)$$

con tal que  $n$  sea suficientemente grande.

- Si hay que resolver un ejemplar pero muy rápidamente (para garantizar un tiempo de respuesta bajo en una aplicación en tiempo real), se calculan previamente  $|J|$  algoritmos precondicionados  $B_1, B_2, \dots, B_n$  y cuando se conoce el problema a resolver,  $\langle j, k \rangle$ , basta con aplicar  $B_j$  al subproblema  $k$ .



# Precondicionamiento: Introducción

## ❖ Ejemplos:

- Dados un conjunto de conjuntos de palabras clave:

$$J = \{ \{ \text{if, then, else, endif} \}, \\ \{ \text{si, entonces, sino, finsi} \}, \\ \{ \text{for, to, by} \}, \\ \{ \text{para, hasta, paso} \} \}$$

Y un conjunto de palabras clave:

$$K = \{ \text{si, begin, hasta, función} \}$$

Hay que resolver un gran número de problemas del tipo: “¿pertenece la palabra clave  $k \in K$  al conjunto  $j \in J$ ?”

Si se resuelve cada ejemplar directamente, el coste es  $t(j,k) \in \Theta(n_j)$ , con  $n_j = |j|$ .

Si previamente se ordenan todos los  $j \in J$ , con un coste  $a(j) \in \Theta(n_j \log n_j)$ , luego cada ejemplar tiene coste  $b_j(k) \in \Theta(\log n_j)$ .



# Precondicionamiento: Introducción

- Si hay que resolver el sistema de ecuaciones  $Ax = b$ , con  $A$  una matriz cuadrada no singular y  $b$  un vector columna para distintos valores de  $b$ , es rentable calcular de antemano la inversa de  $A$ .
- Si se tiene un conjunto de claves en el que hay que hacer muchas búsquedas y se conoce la probabilidad de tener que buscar cada clave, es rentable organizar las claves en un árbol binario de búsqueda óptimo (*Programación dinámica*, pág. 50), con un coste previo de  $\Theta(n^2)$ .



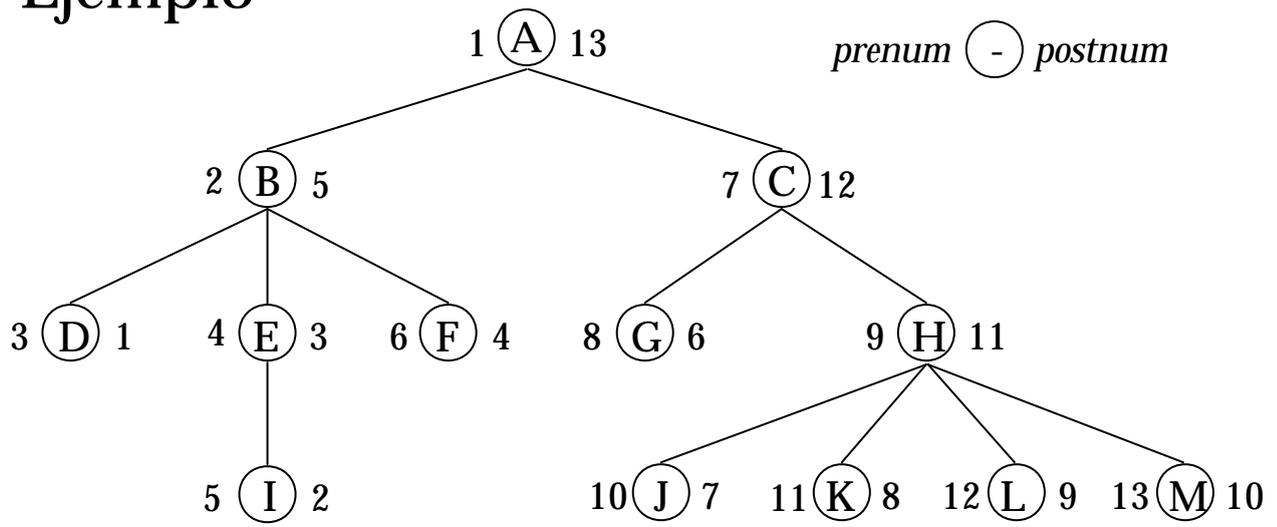
## *Antecesoros en un árbol*

- ❖ Se tiene el árbol  $j$  y hay que resolver muchas veces el problema “¿es el vértice  $u$  un antecesor del vértice  $v$  en el árbol  $j$ ?”
- ❖ Si el árbol  $j$  tiene  $n$  vértices, cualquier solución directa precisa en el peor caso un tiempo  $\Omega(n)$ .
- ❖ Se puede precondicionar el árbol en un tiempo  $\Theta(n)$  para poder resolver cada ejemplar en un tiempo  $\Theta(1)$ .



# Antecesoros en un árbol

## ❖ Ejemplo

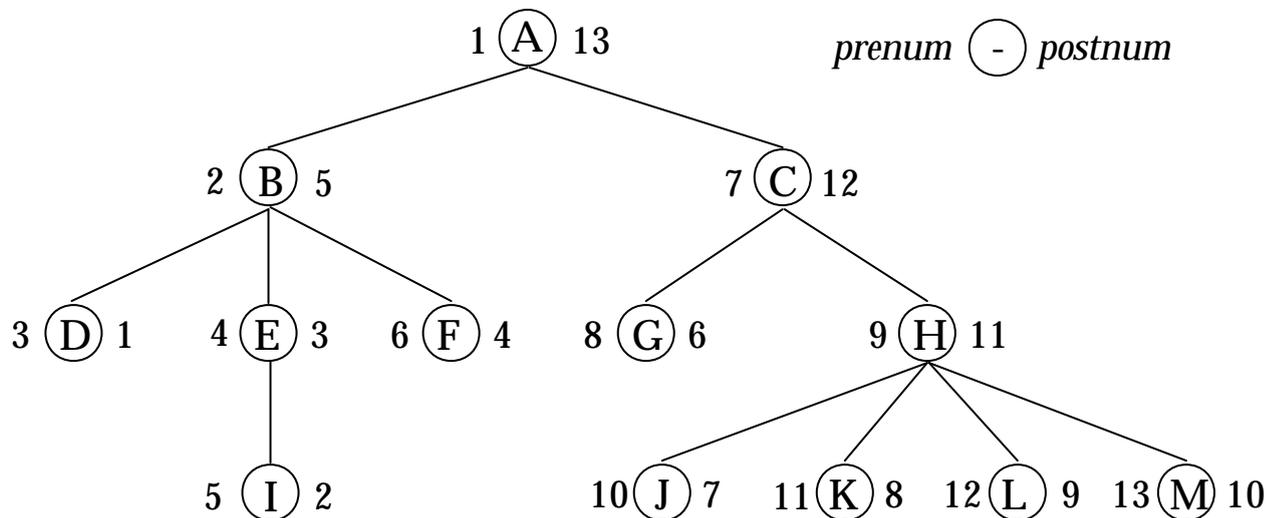


– Se recorre primero en pre-orden numerando los vértices y luego en post-orden, numerándolos también.

- ◆  $prenum[u] = n^o$  correspondiente al recorrido en pre-orden
- ◆  $postnum[u] = n^o$  correspondiente al recorrido en post-orden



# Antecesoros en un árbol



– Se tiene:

$$prenum[u] \leq prenum[v] \Leftrightarrow$$

$$\Leftrightarrow (u \text{ es antecesor de } v) \vee (u \text{ está a la izquierda de } v \text{ en el árbol})$$

$$postnum[u] \geq postnum[v] \Leftrightarrow$$

$$\Leftrightarrow (u \text{ es antecesor de } v) \vee (u \text{ está a la derecha de } v \text{ en el árbol})$$

– Por tanto:

$$(prenum[u] \leq prenum[v]) \wedge (postnum[u] \geq postnum[v]) \Leftrightarrow$$

$$\Leftrightarrow u \text{ es antecesor de } v$$

# Evaluación repetida de un polinomio

- ❖ Sea  $J$  el conjunto de todos los polinomios de una variable entera y coeficientes enteros.
- ❖ Los problemas considerados son los de evaluar un elemento dado de  $J$  para valores de  $K$  (números enteros).
- ❖ Caso particular:
  - polinomios unitarios (el coeficiente de la mayor potencia de  $x$  es 1),
  - de grado  $n = 2^k - 1$ , para un entero  $k \geq 0$ .

# Evaluación repetida de un polinomio

## ❖ Ejemplo:

$$p(x) = x^7 - 5x^6 + 4x^5 - 13x^4 + 3x^3 - 10x^2 + 5x - 17$$

– Método directo (e ingenuo): 12 multiplicaciones.

– Método de Horner: precondicionar  $p$  en la forma

$$p(x) = \left( \left( \left( \left( \left( \left( (x - 5)x + 4 \right) x - 13 \right) x + 3 \right) x - 10 \right) x + 5 \right) x - 17 \right)$$

6 multiplicaciones

– Mejor aún:

$$p(x) = (x^4 + 2) \left( (x^2 + 3)(x - 5) + (x + 2) \right) + \left( (x^2 - 4)x + (x + 9) \right)$$

5 multiplicaciones

# Evaluación repetida de un polinomio

❖ En general:

- Si  $p(x)$  es unitario de grado  $2^k-1$ , se puede expresar como:

$$p(x) = (x^{2^{k-1}} + a)q(x) + r(x),$$

con  $a$  una constante y  $q(x)$  y  $r(x)$  dos polinomios unitarios de grado  $2^{k-1}-1$ .

A continuación, se aplica recursivamente el mismo procedimiento a  $q(x)$  y  $r(x)$ .

Al final,  $p(x)$  queda en función de polinomios de la forma

$$x^{2^i} + c$$

# Evaluación repetida de un polinomio

❖ En el ejemplo:

$$p(x) = x^7 - 5x^6 + 4x^5 - 13x^4 + 3x^3 - 10x^2 + 5x - 17$$

Se expresa como:

$$p(x) = (x^4 + a)(x^3 + q_2x^2 + q_1x + q_0) + (x^3 + r_2x^2 + r_1x + r_0)$$

Igualando los coeficientes de  $x^6$ ,  $x^5$ , ..., 1, se obtiene

$$p(x) = (x^4 + 2)(x^3 - 5x^2 + 4x - 13) + (x^3 - 3x + 9)$$

De la misma forma, se obtienen:

$$x^3 - 5x^2 + 4x - 13 = (x^2 + 3)(x - 5) + (x + 2)$$

$$x^3 - 3x + 9 = (x^2 - 4)x + (x + 9)$$

Para llegar a:

$$p(x) = (x^4 + 2)((x^2 + 3)(x - 5) + (x + 2)) + ((x^2 - 4)x + (x + 9))$$

# Evaluación repetida de un polinomio

## ❖ Análisis del método:

$M(k)$  = nº de multiplicaciones necesarias para evaluar  $p(x)$ , de grado  $2^k-1$

$M'(k) = M(k) - k + 1$ , nº de multiplicaciones si no se cuentan las del cálculo de  $x^2, x^4, \dots$

– Ecuación de recurrencia:

$$M'(k) = \begin{cases} 0 & \text{si } k = 0 \text{ ó } k = 1 \\ 2M'(k-1) + 1 & \text{si } k \geq 2 \end{cases}$$

Por tanto  $M'(k) = 2^{k-1}-1$ , para  $k=1$  y  $M(k) = 2^{k-1}+k-2$

– Es decir, el número de multiplicaciones necesarias para evaluar un polinomio de grado  $n$  “precondicionado” con este método es

$$(n-3)/2 + \log(n+1)$$

# Evaluación repetida de un polinomio

## ❖ Comentarios finales:

- El método se puede generalizar a polinomios no unitarios de cualquier grado.
- El método suele comportarse bien pero **no** da necesariamente una solución óptima.



# Reconocimiento de patrones: Método directo

- ❖ Problema muy frecuente en el diseño de sistemas de tratamiento de textos o en aplicaciones de biología molecular (búsqueda de patrones en moléculas de ADN):

- Dados una **cadena madre** de  $n$  caracteres

$$S = 's_1 s_2 \dots s_n'$$

- y un **patrón** de  $m$  caracteres ( $n \geq m$ )

$$P = 'p_1 p_2 \dots p_m'$$

se quiere saber si  $P$  es una subcadena de  $S$  y, en caso afirmativo, en qué posición de  $S$  se encuentra.

- Instrucción **crítica** para medir la eficiencia de las soluciones:

número de comparaciones entre pares de caracteres



# Reconocimiento de patrones: Método directo

## ❖ Método directo:

```
función subcadena(S,P:cadena; n,m:natural)
    devuelve natural
{Devuelve r si la primera aparición de P en S
 empieza en la posición r (i.e. es el entero más
 pequeño tal que  $S_{r+i-1}=P_i$  para  $i=1,2,\dots,m$ ), y
 devuelve 0 si P no es subcadena de S}
variables ok:booleano; i,j:natural
principio
    ok:=falso; i:=0;
    mq not ok and  $i \leq n-m$  hacer
        ok:=verdad; j:=1;
        mq ok and  $j \leq m$  hacer
            si P[j]≠S[i+j]
                entonces ok:=falso
            sino j:=j+1
        fsi
    fmq;
    i:=i+1;
fmq;
si ok
    entonces devuelve i
    sino devuelve 0
fsi
fin
```



# Reconocimiento de patrones: Método directo

## ❖ Análisis del método directo:

- Intenta encontrar el patrón  $P$  en cada posición de  $S$ .
- Peor caso:
  - ◆ Hace  $m$  comparaciones en cada posición para comprobar si ha encontrado una aparición de  $P$ .

E.g.:  $S = \text{'aaaaaab'}$      $P = \text{'aab'}$

- ◆ El número total de comparaciones es

$$\Omega(m(n-m))$$

es decir,  $\Omega(mn)$  si  $n$  es sustancialmente mayor que  $m$ .



# Reconocimiento de patrones: Uso de firmas

- ❖ Caso particular en que la cadena madre  $S$  se descompone en subcadenas  $S_1 \dots S_k$  y que el patrón  $P$  si está incluido en  $S$  lo está íntegramente en alguna de las  $S_i$ .
  - Por ejemplo,  $S$  es un fichero de texto compuesto por líneas.
  - **Objetivo:** encontrar una función booleana  $T(P, S_i)$  que se pueda evaluar rápidamente para realizar una **comprobación previa**.
    - ◆ Si  $T(P, S_i)$  es falso,  $P$  **no puede ser** subcadena de  $S_i$ , luego no hay que perder más tiempo con ella.
    - ◆ Si  $T(P, S_i)$  es verdad,  $P$  **puede ser** subcadena de  $S_i$ , luego hay que verificarlo (por ejemplo con el método directo).



# Reconocimiento de patrones: Uso de firmas

- ❖ “Firma” = herramienta para diseñar una función booleana  $T(P, S_i)$  de comprobación previa.
- ❖ Ejemplo de firma:
  - Hipótesis:
    - ◆ Sea  $\{‘A’, ‘B’, \dots, ‘X’, ‘Y’, ‘Z’, \textit{otro}\}$  el juego de caracteres utilizado en  $S$  y  $P$  (“*otro*” agrupa a todos los caracteres no alfabéticos).
    - ◆ Supongamos que estamos trabajando con un computador de 32 bits de longitud de palabra.
  - Construcción de una firma:
    - ◆ Definir  $val(‘A’) = 0, val(‘B’) = 1, \dots, val(‘Z’) = 25, val(\textit{otro}) = 26$ .
    - ◆ Si  $c_1$  y  $c_2$  son dos caracteres, definir  $B(c_1, c_2) = (27 \times val(c_1) + val(c_2)) \bmod 32$
    - ◆ Definir la firma de una cadena  $C = c_1 c_2 \dots c_r$  como un valor de 32 bits donde los bits  $B(c_1, c_2), B(c_2, c_3), \dots, B(c_{r-1}, c_r)$  están a 1 y el resto a 0.

# Reconocimiento de patrones: Uso de firmas

## ❖ Ejemplo:

$C = \text{'RECONOCIMIENTO'}$

$$B(\text{'R'}, \text{'E'}) = (27 \times 17 + 4) \bmod 32 = 15$$

$$B(\text{'E'}, \text{'C'}) = (27 \times 4 + 2) \bmod 32 = 14$$

$$B(\text{'C'}, \text{'O'}) = (27 \times 2 + 14) \bmod 32 = 4$$

$$B(\text{'O'}, \text{'N'}) = (27 \times 14 + 13) \bmod 32 = 7$$

$$B(\text{'N'}, \text{'O'}) = (27 \times 13 + 14) \bmod 32 = 13$$

$$B(\text{'O'}, \text{'C'}) = (27 \times 14 + 2) \bmod 32 = 28$$

$$B(\text{'C'}, \text{'I'}) = (27 \times 2 + 8) \bmod 32 = 30$$

$$B(\text{'I'}, \text{'M'}) = (27 \times 8 + 12) \bmod 32 = 4$$

$$B(\text{'M'}, \text{'I'}) = (27 \times 12 + 8) \bmod 32 = 12$$

$$B(\text{'I'}, \text{'E'}) = (27 \times 8 + 4) \bmod 32 = 28$$

$$B(\text{'E'}, \text{'N'}) = (27 \times 4 + 13) \bmod 32 = 25$$

$$B(\text{'N'}, \text{'T'}) = (27 \times 13 + 19) \bmod 32 = 18$$

$$B(\text{'T'}, \text{'O'}) = (27 \times 19 + 14) \bmod 32 = 15$$

$\text{firma}(C) = 0000\ 1001\ 0000\ 1111\ 0010\ 0000\ 0100\ 1010$

(numerando de 0 a 31, de izquierda a derecha)



# Reconocimiento de patrones: Uso de firmas

❖ Uso de la firma anterior para diseñar la función booleana  $T(P, S_i)$  de comprobación previa:

- Se calcula la firma de  $S_i$  y de  $P$ .
- Si  $S_i$  contiene el patrón  $P$ , entonces todos los bits que están a 1 en la firma de  $P$  están también a 1 en la firma de  $S_i$ . Es decir:

$$T(P, S_i) \equiv ((\text{firma}(P) \text{ and } \text{firma}(S_i)) = \text{firma}(P))$$

donde la operación “and” representa la conjunción bit a bit de las dos palabras.



# Reconocimiento de patrones: Uso de firmas

## ❖ Comentarios:

- Es un ejemplo de preconditionamiento.
- El cálculo de firmas requiere un tiempo:
  - ♦  $O(n)$  para  $S$
  - ♦  $O(m)$  para  $P$
- El cálculo de  $T(P, S_j)$  es muy rápido.
- En la práctica, el beneficio de este método depende de una buena elección del procedimiento de cálculo de firmas.
- Si la cadena madre es muy larga y no está subdividida en subcadenas, el método no es bueno porque su firma tiende a tener todo 1's y por tanto  $T(P, S)$  es verdad con una muy alta probabilidad.
- Si el juego de caracteres es mayor (por ejemplo, 128), se puede definir  $B$  en la forma:

$$B(c_1, c_2) = (128 \times \text{val}(c_1) + \text{val}(c_2)) \bmod 32$$

Aunque en este caso conviene aumentar el  $n^{\circ}$  32, es decir, el  $n^{\circ}$  de bits de la firma.

- También puede hacerse que la función  $B$  tenga como parámetros tres caracteres consecutivos.





# Reconocimiento de patrones: El algoritmo KMP

- Notar que hay muchas comparaciones redundantes: **el método directo compara el mismo subpatrón en el mismo lugar de la cadena madre varias veces.**
  - ◆ Por ejemplo, verificamos dos veces que la subcadena 'xyxy' está en la posición 11 de S (en las líneas 6 y 11).
  - ◆ En la aplicación de buscar una palabra en un fichero de texto, el método directo no es muy malo porque las discrepancias ocurren enseguida y, por tanto, los retrocesos son pequeños.
  - ◆ Sin embargo, en otras aplicaciones (por ejemplo, biología molecular) el alfabeto es muy pequeño y hay muchas repeticiones, por tanto los retrocesos son mayores.

# 4 Reconocimiento de patrones: El algoritmo KMP

## ❖ Otro ejemplo:

$S = \text{'yyyyyyyyyyyyyyx'}$

$P = \text{'yyyyyx'}$

- Con el método directo, las cinco 'y' del patrón se comparan con la cadena madre, se encuentra la discrepancia de la 'x', se "desplaza" el patrón un lugar a la derecha y se hacen cuatro comparaciones redundantes:

¡las cuatro primeras 'y' ya sabemos que están ahí!

Se desplaza el patrón un lugar más a la derecha y ¡de nuevo cuatro comparaciones redundantes!

Etcétera.



# Reconocimiento de patrones: El algoritmo KMP

## ❖ Un ejemplo más:

$S = \text{'xyyyxyxyxyyy'}$

$P = \text{'xyyyy'}$

- Buscamos la ocurrencia de una 'x' seguida de cinco 'y'.

Si el número de 'y' no es suficiente, no hay necesidad de volver atrás y desplazar una posición a la derecha.

Las cuatro 'y' encontradas no valen para nada y hay que buscar una nueva 'x'.





# Reconocimiento de patrones: El algoritmo KMP

- Notar que la discusión anterior es **independiente de la cadena madre  $S$** .

Conocemos los últimos caracteres de  $S$  porque han coincidido con los anteriores del patrón  $P$ .

- Sigamos con el ejemplo...

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	x	y	x	x	y	x	y	x	y	y	x	y	x	y	x	y	y	x	y	x	y	x	x	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6:						x	y	x	y	y	x	y	x	y	x									
7:						x																		
8:							x	y	x															
9:								x																
10:									x															
11:										x	y	x	y	y										
12:											x													
13:												x	y	x	y	y	x	y	x	y	x	x	x	

- La discrepancia en la línea 6 es en  $p_{11}$ .
- Ahora podemos ahorrarnos 15 comparaciones...



# Reconocimiento de patrones: El algoritmo KMP

- La discrepancia fue entre  $p_{11}$  y  $s_{16}$ .
- Consideremos el subpatrón  $p_1p_2\dots p_{10}$ .
  - ◆ Sabemos que  $p_1p_2\dots p_{10} = s_6s_7\dots s_{15}$ .
  - ◆ Queremos saber cuántas posiciones hay que desplazar  $P$  hacia la derecha hasta que vuelva a existir la posibilidad de que coincida con una subcadena de  $S$ .
  - ◆ Nos fijamos en el **máximo sufijo de  $p_1p_2\dots p_{10}$  que coincide con un prefijo de  $P$** .

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	x	y	x	x	y	x	y	x	y	y	x	y	x	y	x	y	y	x	y	x	y	x	x
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6:						x	y	x	y	y	x	y	x	y	x	x							

- ◆ En este caso, el sufijo es de longitud 3: 'xyx'.
- ◆ Luego se puede continuar comparando  $s_{16}$  con  $p_4$ .

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	x	y	x	x	y	x	y	x	y	y	x	y	x	y	x	y	y	x	y	x	y	x	x
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6:						x	y	x	y	y	x	y	x	y	x	x							
13:													x	y	x	y	y	x	y	x	y	x	x



# Reconocimiento de patrones: El algoritmo KMP

- ❖ Notar, de nuevo, que **toda la información necesaria para saber cuánto hay que desplazar a la derecha el patrón está incluida en el propio patrón.**
- ❖ Se puede “preprocesar” (o precondicionar) el patrón para acelerar el método directo.
- ❖ La idea es la siguiente:
  - La cadena madre  $S$  siempre se recorre hacia la derecha (no hay retrocesos), aunque un mismo carácter de  $S$  puede compararse con varios del patrón  $P$  (cuando haya discrepancias).
  - Cuando haya una discrepancia se consultará una **tabla** para saber cuánto hay que retroceder en el patrón o, dicho de otra forma, cuántos desplazamientos del patrón hacia la derecha pueden hacerse.
  - En la tabla hay un entero por cada carácter de  $P$ , e indica cuántos desplazamientos hacia la derecha deben hacerse cuando ese carácter discrepe con uno de la cadena madre.



# Reconocimiento de patrones: El algoritmo KMP

## ❖ Definición precisa de la tabla:

- Para cada  $p_i$  de  $P$ , hay que calcular el sufijo más largo  $p_{i-j}p_{i-j+1}\dots p_{i-1}$  que es igual al prefijo de  $P$ :  
$$\text{sig}(i) = \max \{ j \mid 0 < j < i-1, p_{i-j}p_{i-j+1}\dots p_{i-1} = p_1p_2\dots p_j \}$$
  
0 si no existe tal  $j$
- Así, si  $\text{sig}(i) = j$ , el carácter discrepante de  $S$  con  $p_i$  puede pasar a compararse directamente con  $p_{j+1}$  (sabemos que los  $j$  caracteres más recientes de  $S$  coinciden con los  $j$  primeros de  $P$ ).
- Por convenio,  $\text{sig}(1) = -1$ , para distinguir ese caso especial.
- Además, es obvio que siempre  $\text{sig}(2) = 0$  (no existe ningún  $j$  tal que  $0 < j < 2-1$ ).

## ❖ En el ejemplo anterior:

$i =$	1	2	3	4	5	6	7	8	9	10	11
$P =$	$x$	$y$	$x$	$y$	$y$	$x$	$y$	$x$	$y$	$x$	$x$
$\text{sig} =$	-1	0	0	1	2	0	1	2	3	4	3



# Reconocimiento de patrones: El algoritmo KMP

```
función KMP(S,P:cadena; n,m:natural)
    devuelve natural
    {Idéntica especificación a 'subcadena'.}
variables i,j,pos:natural
principio
    j:=1; i:=1;
    pos:=0;
    mq pos=0 and i≤n hacer
        si P[j]=S[i]
            entonces
                j:=j+1;
                i:=i+1
            sino
                j:=sig[j]+1;
                si j=0 entonces
                    j:=1;
                    i:=i+1
                fsi
            fsi;
        si j=m+1 entonces pos:=i-m fsi
    fmq;
    devuelve pos
fin
```



# Reconocimiento de patrones: El algoritmo KMP

## ❖ Falta el cálculo de la tabla *sig*:

- Lo haremos por inducción.
- $sig(2) = 0$
- Suponer que *sig* ya está calculada para  $1, 2, \dots, i-1$ .
- ◆ Como mucho,  $sig(i)$  puede ser  $sig(i-1)+1$

Esto ocurre si  $p_{i-1} = p_{sig(i-1)+1}$ .

$i =$	1	2	3	4	5	6	7	8	9	10	11
$P =$	<u>x</u>	<u>y</u>	<u>x</u>	<u>y</u>	<u>y</u>	<u>x</u>	<u>y</u>	<u>x</u>	<u>y</u>	x	x
$sig =$	-1	0	0	1	2	0	1	2	3	<u>4</u>	

- ◆ Si  $p_{i-1} \neq p_{sig(i-1)+1}$ :

$i =$	1	2	3	4	5	6	7	8	9	10	11
$P =$	<u>x</u>	<u>y</u>	<u>x</u>	<u>y</u>	<u>y</u>	<u>x</u>	<u>y</u>	<u>x</u>	<u>y</u>	<u>x</u>	x
$sig =$	-1	0	0	1	<u>2</u>	0	1	2	3	4	?

Es como una búsqueda de un patrón...

$i =$	1	2	3	4	5	6	7	8	9	10	11
$P =$	x	y	x	y	y	x	y	x	y	<u>x</u>	x
$p_{1 \dots p_{sig(i-1)+1}} =$							x	y	x	y	y
$p_{1 \dots p_{sig(sig(i-1)+1)+1}} =$								x	y	<u>x</u>	
											$\Rightarrow sig(11) = 3$

# 4 Reconocimiento de patrones: El algoritmo KMP

```
tipo vect = vector[1..m] de entero
```

```
algoritmo calculaSig(ent P:cadena;  
                    ent m:natural;  
                    sal sig:vect)  
{Cálculo de la tabla 'sig' para el  
 patrón 'P'.}  
variables i,j:natural  
principio  
  sig[1]:=-1;  
  sig[2]:=0;  
  para i:=3 hasta m hacer  
    j:=sig[i-1]+1;  
    mq j>0 and entonces P[i-1]≠P[j] hacer  
      j:=sig[j]+1  
    fmq;  
    sig[i]:=j  
  fpara  
fin
```



# Reconocimiento de patrones: El algoritmo KMP

## ❖ Análisis de la función KMP:

- La cadena madre  $S$  se recorre sólo una vez, aunque un carácter  $s_i$  de  $S$  puede que haya que compararlo con varios de  $P$ .
- ¿Cuántas veces se puede llegar a comparar un carácter  $s_i$  de  $S$  con otros de  $P$ ?

- ◆ Supongamos que la primera comparación de  $s_i$  se hace con  $p_k$ . Por tanto, en particular, se ha avanzado  $k$  veces en  $P$  hasta hacer esa comparación y sin retroceder ninguna vez.
- ◆ Si  $s_i \neq p_k$ , se retrocede en  $P$  (usando la tabla *sig*). Sólo se puede retroceder un máximo de  $k$  veces.

→ Si se suma el coste de los retrocesos al de los movimientos de avance, únicamente se dobla el coste de los avances.

Pero el número de avances en  $P$  coincide con el número de avances en  $S$  y es  $n$ .

Luego el número de comparaciones es  $O(n)$ .

- ## ❖ Con similares argumentos se demuestra que el coste del cálculo de la tabla *sig* es $O(m)$ , luego el coste total es $O(n)$ .

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

R.S. Boyer y J.S. Moore:

“A fast string searching algorithm”,

*Communications of the ACM*, 20(10), pp. 762-772, 1977.

- ❖ Como el algoritmo KMP, el algoritmo BM puede encontrar todas las apariciones de un patrón  $P$  (de longitud  $m$ ) en una cadena madre  $S$  (de longitud  $n$ ) en un tiempo  $O(n)$  en el caso peor.
- ❖ KMP examina cada carácter de  $S$  al menos una vez, luego realiza un mínimo de  $n$  comparaciones.
- ❖ BM es **sublineal**: no examina necesariamente todos los caracteres de  $S$  y el  $n^0$  de comp. es, a menudo, inferior a  $n$ .
- ❖ Además, BM tiende a ser más eficiente cuando  $m$  crece.
- ❖ En el mejor caso, BM encuentra todas las apariciones de  $P$  en  $S$  en un tiempo  $O(m+n/m)$ .

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

## ❖ Descripción:

- Como en KMP, desplazamos  $P$  sobre  $S$  de izquierda a derecha examinando los caracteres enfrentados.
- Pero ahora la verificación de los caracteres de  $P$  se hace de derecha a izquierda después de cada desplazamiento del patrón.
- Se utilizan dos reglas para decidir el desplazamiento que hay que hacer después de una discrepancia:

**Regla A.** Después de un desplazamiento, se compara  $p_m$  con un carácter  $c$  de  $S$  y son distintos:

- ◆ Si  $c$  aparece más a la izquierda en el patrón, se desplaza éste para alinear la última aparición de  $c$  en el patrón con el carácter  $c$  de  $S$ .
- ◆ Si  $c$  no está en  $P$ , se coloca éste inmediatamente detrás de la aparición de  $c$  en  $S$ .

**Regla B.** Si un cierto  $n^0$  de caracteres al final de  $P$  se corresponde con caracteres de  $S$ , aprovechamos este conocimiento parcial de  $S$  (como en KMP) para desplazar  $P$  a una nueva posición compatible con la información que poseemos.

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

## ❖ Ejemplo:

$S =$  'se espera cielo nublado para mañana'  
 $P =$  'lado'



Se detecta la primera discrepancia en la primera comparación. Se aplica la Regla A:

$S =$  'se espera cielo nublado para mañana'  
 $P =$  'lado'



De nuevo una discrepancia. Regla A:

$S =$  'se espera cielo nublado para mañana'  
 $P =$  'lado'



Lo mismo. Regla A:

$S =$  'se espera cielo nublado para mañana'  
 $P =$  'lado'

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

$S =$  'se espera cielo nublado para mañana'

$P =$  'lado'



Lo mismo. Regla A:

$S =$  'se espera cielo nublado para mañana'

$P =$  'lado'



Discrepancia. Regla A. Ahora el carácter enfrentado a  $p_m$  aparece en  $P$ . Se desplaza  $P$  para alinear ambas apariciones.

$S =$  'se espera cielo nublado para mañana'

$P =$  'lado'



Se hace la verificación (de derecha a izquierda) y se encuentra el patrón.

- Sólo se ha usado la Regla A.
- Se han hecho sólo 9 comparaciones.

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

## ❖ Otro ejemplo:

$S = \text{'babcbabcbabcaabcbabcbacabc'}$

$P = \text{'abcabcbacab'}$



- Se hacen 4 comparaciones hasta encontrar la 1ª diferencia.
- Sabemos que, a partir de esa posición,  $S$  contiene los caracteres  $\text{'xcab'}$  con  $x \neq a$ .
- Regla B: Desplazando  $P$  5 posiciones hacia la derecha se mantiene esa información (el subrayado indica los caracteres alineados).

$S = \text{'babcbabcbabcaabcbabcbacabc'}$

$P = \text{'abcabcbacab'}$



- Regla A:

$S = \text{'babcbabcbabcaabcbabcbacabc'}$

$P = \text{'abcabcbacab'}$



- A diferencia del algoritmo KMP, el BM puede hacer comparaciones redundantes (correspondientes a los caracteres subrayados).
- De nuevo, la Regla B (para alinear  $\text{'cab'}$ ):

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

$S = \text{'babcbabcabcaabcabcbacabc'}$

$P = \text{'abcabcacab'}$



– Regla A:

$S = \text{'babcbabcabcaabcabcbacabc'}$

$P = \text{'abcabcbacab'}$



– Otra vez, Regla A:

$S = \text{'babcbabcabcaabcabcbacabc'}$

$P = \text{'abcabcbacab'}$



– Se han hecho 21 comparaciones para encontrar  $P$ ,  
2 de ellas redundantes.

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

## ❖ Precondicionamiento o preproceso necesario para implementar BM:

- Se necesita dos vectores:

$d1[\{\text{juego de caracteres}\}]$

$d2[1..m-1]$

el 1º para implementar la Regla A y el 2º la B.

- El cálculo de  $d1$  es fácil:

```
para todo  $c \in \{\text{juego caract.}\}$  hacer  
  si  $c \notin P[1..m]$   
    entonces  $d1[c] := m$   
    sino  $d1[c] := m - \max\{i \mid P[i] = c\}$   
  fsi  
fpara
```

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

– El cálculo de  $d_2$  es más complicado (no veremos los detalles).

– La interpretación de  $d_2$  es:

Después de una discrepancia en la posición  $i$  del patrón, recomenzamos la comprobación en la posición  $m$  del patrón  $d_2[i]$  caracteres más a la derecha en  $S$ .

– Veamos un ejemplo:

$S = \text{'?????xe????????'}$  con  $x \neq t$

$P = \text{'ostente'}$



Como  $x \neq t$  no se puede alinear la  $e$  de  $S$  con la otra  $e$  de  $P$ , luego hay que desplazar  $P$  completamente, es decir,  $d_2[6] := 8$ :

$S = \text{'?????xe????????'}$  con  $x \neq t$

$P = \text{'ostente'}$

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

- De igual forma:

$S = \text{'????xte??????'}$  con  $x \neq n$

$P = \text{'ostente'}$



En este caso, es claro que  $d2[5] := 5$ :

$S = \text{'????xte??????'}$  con  $x \neq n$

$P = \text{'ostente'}$



Etcétera:  $d2 := [13, 12, 11, 10, 5, 8]$

# 4 Reconocimiento de patrones: El algoritmo de Boyer y Moore

- Hay veces que según lo dicho hasta ahora hay que aplicar la Regla B, sin embargo es más eficiente aplicar la Regla A:

$S = \text{'??virte??????'}$

$P = \text{'ostente'}$



Si se aplica la Regla B ( $d_2[5]=5$ ):

$S = \text{'??virte??????'}$

$P = \text{'ostente'}$



Sin embargo, como 'r' no aparece en  $P$ , se puede desplazar directamente en ( $d_1[u]=7$ ):

$S = \text{'??virte??????'}$

$P = \text{'ostente'}$



❖ Detalles: ¡consultar bibliografía!