Titolo modulo: Approfondimenti su “String Matching” [P2_06]

Titolo unità didattica: algoritmo KMP di string matching [1-AT]

Algoritmi di string matching che riducono il numero dei confronti

Argomenti trattati:

- ✓ Il problema dello string matching
- ✓ Esempi di problemi di string matching
- ✓ Algoritmi di string matching
- ✓ Algoritmo KMP

Prerequisiti richiesti: algoritmo di ricerca diretta

String Matching (S.M.):

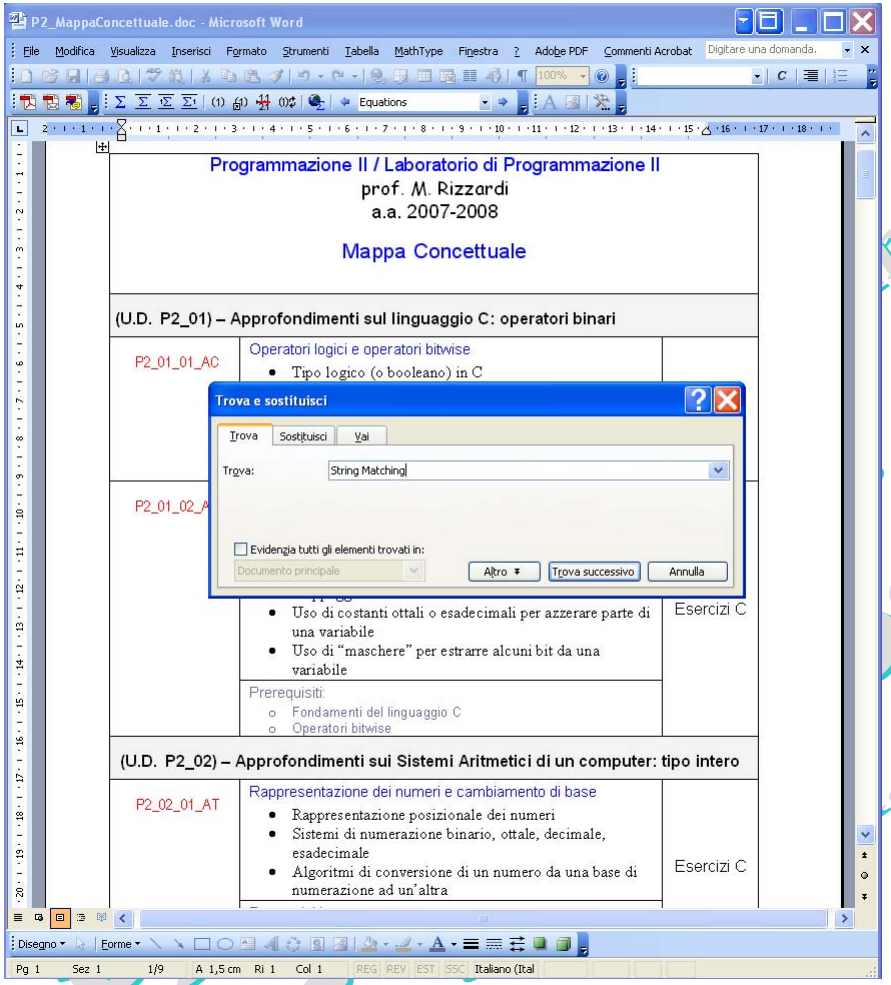
ricerca di un pattern P in un testo T

(è un particolare problema di ricerca)

Il problema dello String Matching

- **S.M. Decisionale**:
  il pattern P è presente nel testo T?
- **S.M. Quantitativo**:
  quante volte P si trova in T?
- **S.M. Enumerativo**:
  in quali posizioni di T occorre P?

String matching (caso particolare di pattern matching)

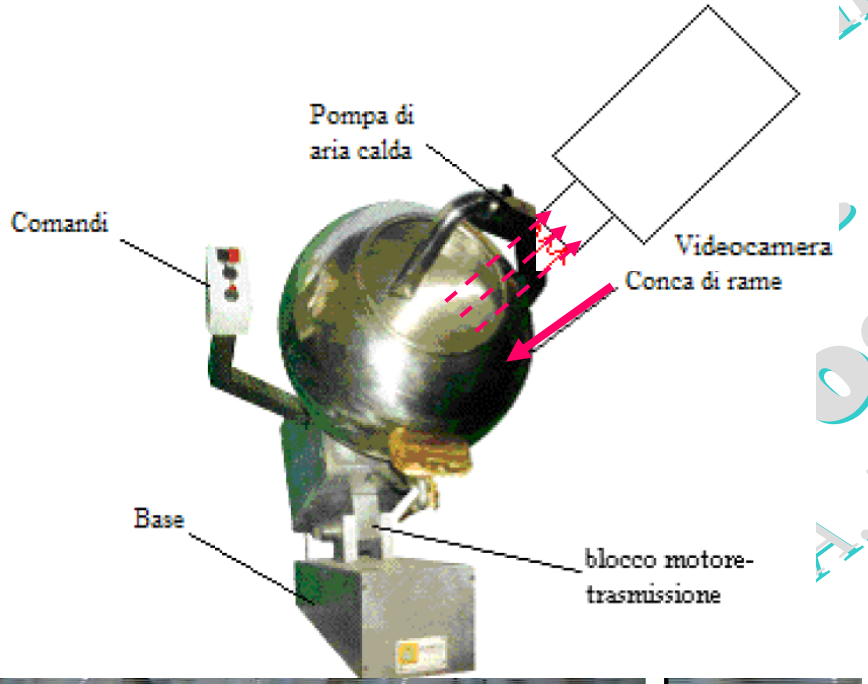


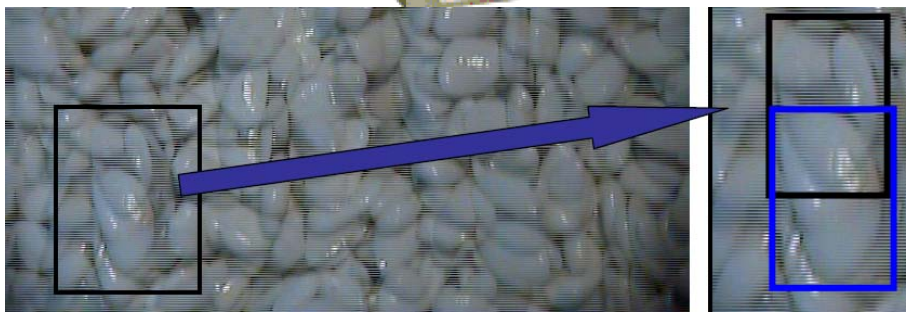
È un problema che si presenta spesso nei programmi ***editor*** o ***word processor***.

Gli stessi algoritmi di **string matching** sono anche usati per cercare particolari sottosequenze di DNA

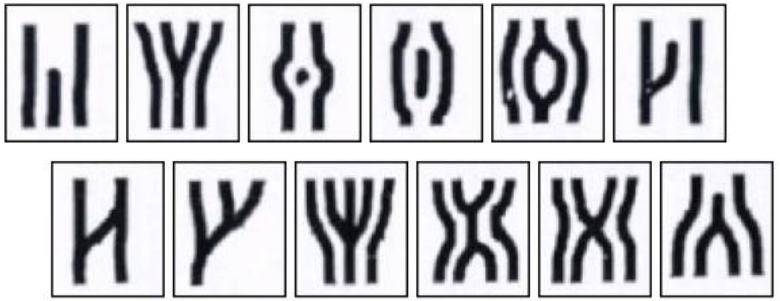
Altre applicazioni di Pattern/String matching

confettatura
=
rivestimento delle mandorle con uno strato zuccherato





riconoscimento di impronte digitali



minuzie

Algoritmi di string matching

- Naive (ricerca diretta);

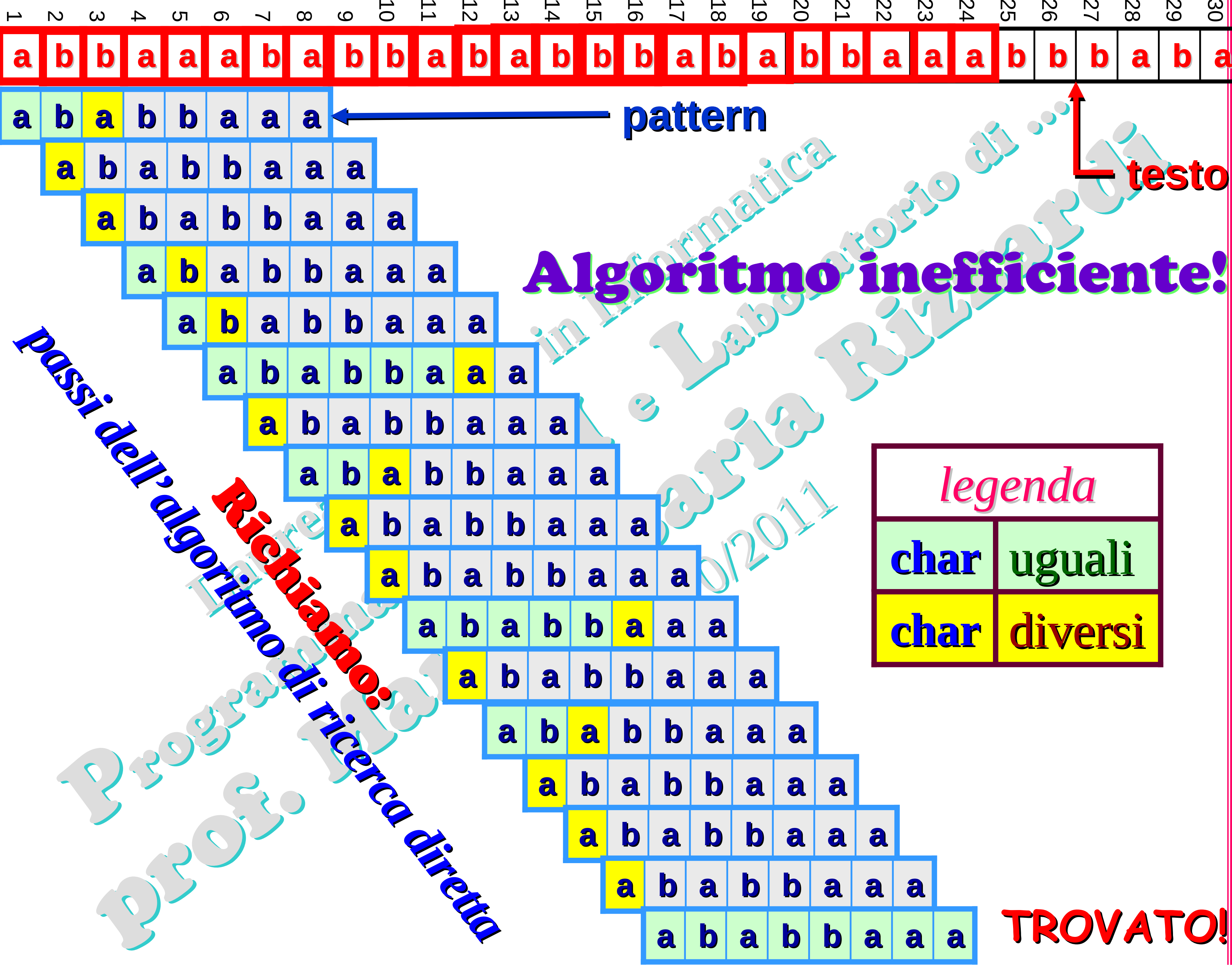
già visto!

- Smart (automa a stata finiti);
- Rabin-Karp;
- Knuth-Morris-Pratt;
- Boyer-Moore.

algoritmi “veloci”

Algoritmo inefficiente!

Richiamo: *passi dell'algoritmo di ricerca diretta*



legenda	
char	uguali
char	diversi

Algoritmo Naive (ricerca diretta): valutazione

- Complessità computazionale nel caso peggiore: $O(MN)$ dove N è la lunghezza del testo e M quella del pattern.
- Complessità computazionale nel caso medio: $O(N)$ se la distribuzione dei caratteri del testo e del pattern è uniforme e l'alfabeto contiene molti caratteri.

Vantaggi

- Semplice da implementare.
- È ragionevole usarlo per la prima occorrenza del pattern quando questa abbia una buona probabilità di trovarsi all'inizio del testo o quando il pattern sia lungo al più 3-4 caratteri.
- Il caso peggiore si verifica maggiormente con stringhe particolarmente ripetitive.

Svantaggi

- Inefficiente su sequenze lunghe del pattern (sequenze del DNA) in quanto l'algoritmo confronta ogni volta anche sottostringhe matched spostando il pattern sul testo soltanto di una posizione.

String matching: Algoritmo KMP (Knuth-Morris-Pratt)

Idea dell'algoritmo KMP:

quando il confronto fra il carattere del testo e del pattern <u>fallisce</u> (mismatch) <u>dopo un certo numero di successi</u> (match), invece che arretrare il puntatore sul pattern e sul testo, ignorando l'esito dei confronti già eseguiti, <u>si possono sfruttare le conoscenze sul pattern evitando confronti il cui esito è già noto</u>.

match — mismatch

...	a	b	a	b	b	a	b	a	...
	a	b	a	b	b	a	a	a	

→ Si possono saltare dei passaggi

È richiesta la costruzione di una **tabella preliminare** a partire dai caratteri di pattern.

<u>Algoritmo di ricerca più veloce</u> in due fasi:

(1) elaborazione del pattern;

(2) scansione "veloce" del testo

algoritmo KMP: elaborazione del pattern

definizione di **prefisso** e **suffisso** di una stringa

STRINGA = ”Buon giorno!”

”**Buon**” ”Buon giorno!” ”Buon ”**giorno!**”

”Buon”= $pref_4$(STRINGA)
prefisso di lunghezza 4 di STRINGA

”giorno!”= $suff_7$(STRINGA)
suffisso di lunghezza 7 di STRINGA

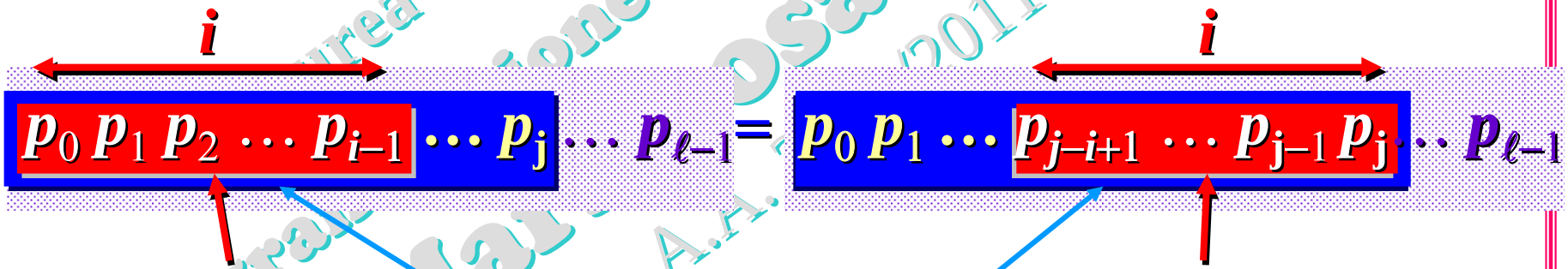
Si può calcolare il prefisso ed il suffisso anche di una sottostringa

Siano p_k i caratteri del **pattern** (di lunghezza totale ℓ):

$$\textbf{pattern} = \text{“}p_0\, p_1\, p_2 \cdots p_{\ell-1}\text{”}$$

$\forall$**j** sia **sub$_j$[pattern]** la sottostringa di **pattern** di lunghezza **j+1**:

$$\textbf{sub}_j\textbf{[pattern]} = \text{“}p_0\, p_1\, p_2 \cdots p_j\text{”} = \textbf{pattern}[0..j]$$



prefisso di lunghezza *i*
pref$_i$(sub$_j$[pattern])

suffisso di lunghezza *i*
suff$_i$(sub$_j$[pattern])

sub$_j$[pattern] = sottostringa di lunghezza **j+1** di **pattern**

Esempio: **prefisso** e **suffisso** di una sottostringa

STRINGA = ”Buon giorno!”

sub_6[STRINGA] = ”Buon gi”

”Buon” ”Buon gi” ”Buon gi” ”gi”

”Buon”= $pref_4$(sub_6[STRINGA])
prefisso di lunghezza 4
della sottostringa [6] di STRINGA

”_gi”= $suff_3$(sub_6[STRINGA])
suffisso di lunghezza 3
della sottostringa [6] di STRINGA

$p_0\, p_1\, p_2 \cdots p_{i-1} \cdots p_j \cdots p_{\ell-1}$

sottostringa di lunghezza j+1 di pattern

definizione di funzione insuccesso

$p_0\, p_1\, p_2 \cdots p_{i-1} \cdots p_j \cdots p_{\ell-1}$ (i)

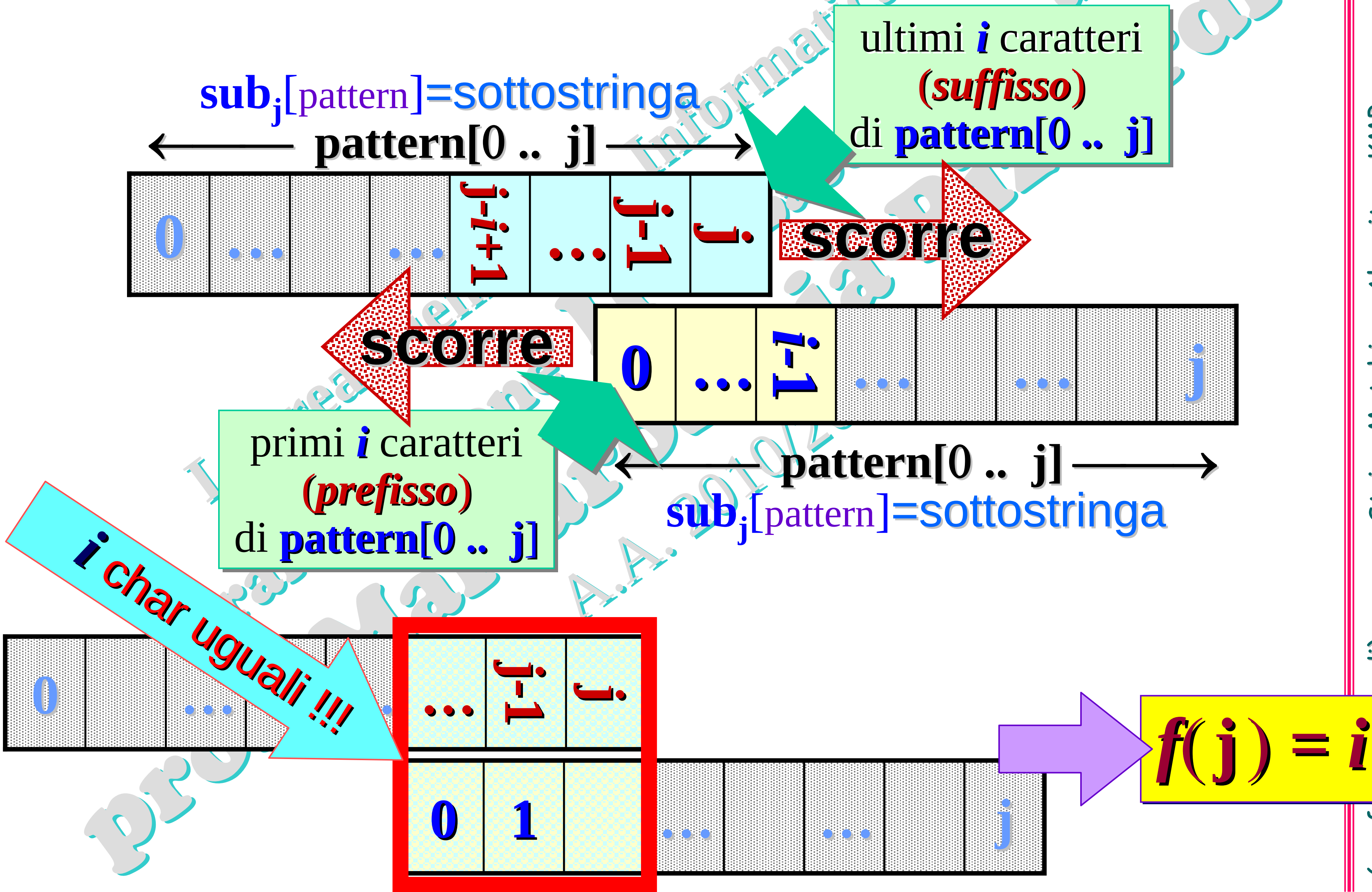
$\mathbf{pref_i(sub_j[pattern])}$

$p_0 \cdots p_{j-i+1} \cdots p_{j-1}\, p_j \cdots p_{\ell-1}$ (i)

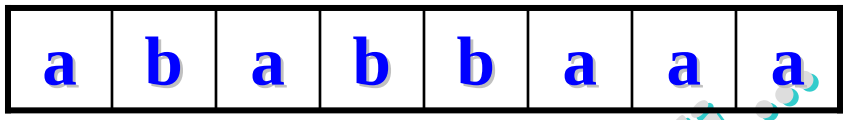
$\mathbf{suff_i(sub_j[pattern])}$

$$f(j) = \begin{cases} \text{valore massimo di } i\ (0 < i \leq j), \text{ se } \exists i, \text{ tale che} \\ \mathbf{pref_i(sub_j[pattern])} = \underbrace{"p_0 p_1 p_2 \cdots p_{i-1}"}_{\text{prefisso di lunghezza } i} = \mathbf{suff_i(sub_j[pattern])} \\ \qquad \underbrace{"p_{j-(i-1)} \cdots p_{j-2} p_{j-1} p_j"}_{\text{suffisso di lunghezza } i} \\ 0 \ \text{altrimenti} \end{cases} \qquad \forall\, j \equiv 0, 1, \ldots \ell - 1$$

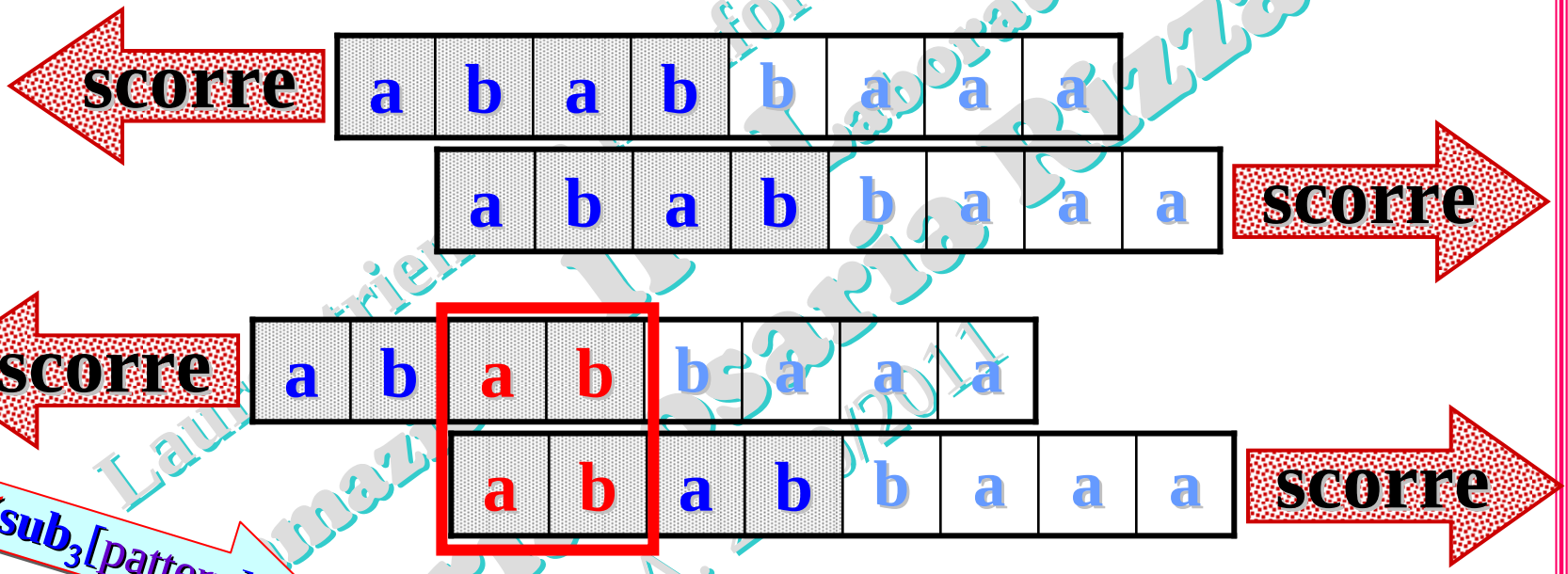
Cerca nella sottostringa del pattern di lunghezza j+1 eventuali prefissi e suffissi uguali: sceglie quelli di lunghezza massima.



Esempio: pattern



$sub_3[pattern]=pattern[0..3]$



$f(3) = 2$

Esempio: pattern = “a b c a b c a c a b”

“$p_0\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8\ p_9$”

j	f(j)	costruzione tabella
0	0	a b c a b c a c a b a b c a b c a c a b
1	0	a b c a b c a c a b a b c a b c a c a b
2	0	a b c a b c a c a b a b c a b c a c a b
3	1	a b c a b c a c a b a b c a b c a c a b a
4	2	a b c a b c a c a b a b c a b c a c a b a
5	3	a b c a b c a c a b a b c a b c a c a b
6	4	a b c a b c a c a b a b c a b c a c a b
7	0	a b c a b c a c a b a b c a b c a c a b
8	1	a b c a b c a c a b a b c a b c a c a b

legenda

sottostringa

prefisso

suffisso

Esempio: trovare il pattern | a | b | a | b | b | a | a | a | nel seguente testo

a	b	b	a	a	a	b	a	b	b	a	b	a	b	b	b	a	b	a	b	b	a	a	a	b	b	b	a	b	a

pattern = | a | b | a | b | b | a | a | a |

$p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7$

j	f(j)	costruzione tabella							
0	0	a	b	a	b	b	a	a	a
1	0	a	b	a	b	b	a	a	a
2	1	a	b	a	b	b	a	a	a
3	2	a	b	a	b	b	a	a	a
4	0	a	b	a	b	b	a	a	a
5	1	a	b	a	b	b	a	a	a
6	1	a	b	a	b	b	a	a	a
		p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7

j	f(j)
0	0
1	0
2	1
3	2
4	0
5	1
6	1

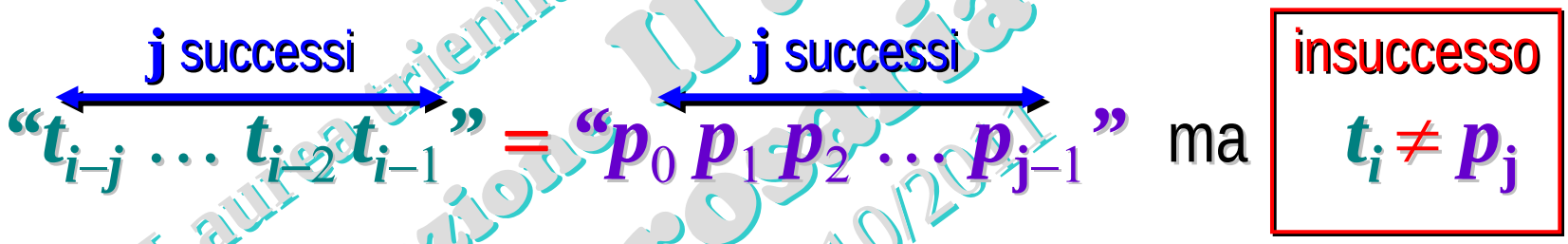
legenda

sottostringa

prefisso suffisso

algoritmo KMP: continuazione confronti

Se nel confronto dei caratteri t_k del **testo** con quelli p_h del **pattern** si verifica un "insuccesso" (mismatch) dopo **j** "successi" (match), cioè

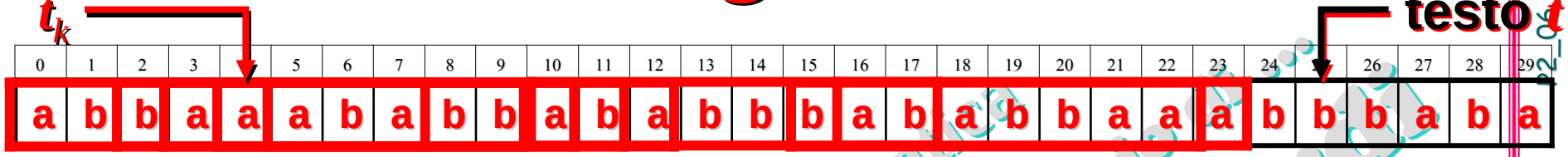


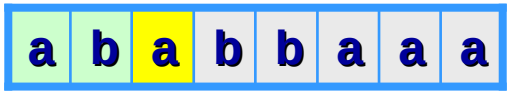
allora la ricerca può ricominciare confrontando

next step

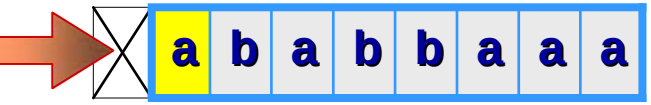
- t_i con $p_{f(j-1)}$ se $j \neq 0$
- t_{i+1} con p_0 se $j = 0$

Passi dell'algoritmo KMP



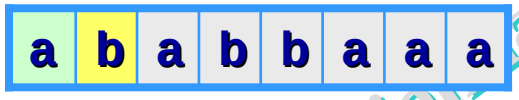


$j=2 \Rightarrow t_2$:?: $p_{f(j-1)}=p_0$ essendo $f(1)=0$

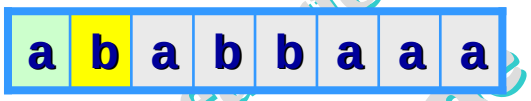


Algoritmo efficiente!

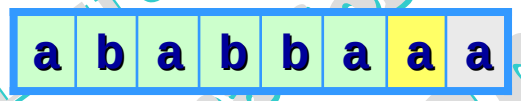
j	f(j)
0	0
1	0
2	1
3	2
4	0
5	1
6	1



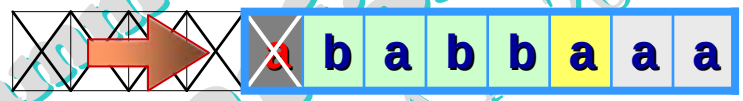
$j=1 \Rightarrow t_4$:?: $p_{f(j-1)}=p_0$ essendo $f(0)=0$



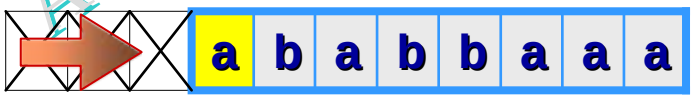
$j=1 \Rightarrow t_5$:?: $p_{f(j-1)}=p_0$ essendo $f(0)=0$

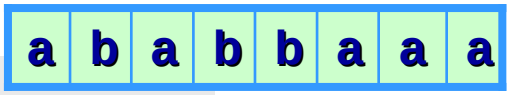


$j=6 \Rightarrow t_{11}$:?: $p_{f(j-1)}=p_1$ essendo $f(5)=1$



$j=5 \Rightarrow t_{15}$:?: $p_{f(j-1)}=p_0$ $f(4)=0$





TROVATO!

"$t_{k-j} \ldots t_{k-2}\, t_{k-1}$"="$p_0\, p_1\, p_2 \ldots p_{j-1}$" ma $t_k \neq p_j$

next step $\begin{cases} t_k & \text{con} \quad p_{f(j-1)} \quad \text{se} \quad j \neq 0 \\ t_{k+1} & \text{con} \quad p_0 \quad \text{se} \quad j = 0 \end{cases}$

Algoritmo kMP (Knuth-Morris-Pratt): valutazione

- Complessità computazionale nel caso peggiore: $O(MN)$.
- Complessità computazionale nel caso medio: $O(M+N)$.

Vantaggi

- Indipendenza dall'alfabeto e dalla distribuzione dei suoi caratteri.
- Velocità di esecuzione della ricerca.

Svantaggi

- Degrada all'algoritmo naive nel caso peggiore.

Esercizio

1 Determinare il numero totale di occorrenze di un pattern in un testo, usando per la ricerca l'***algoritmo KMP***.

[liv. 3]