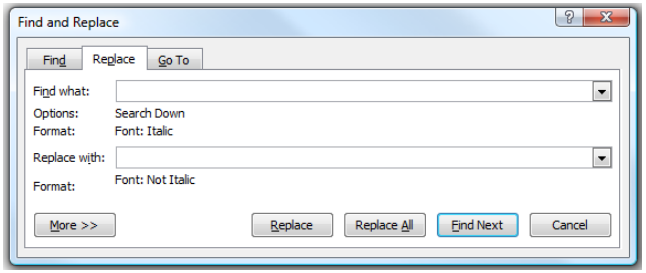
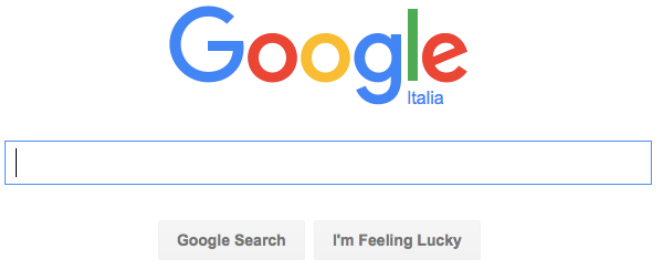
Text Processing

Campi Applicativi

- text editing



- web searching



- computational biology (e.g., *DNA analysis*)

- vision
- ...

Definizioni

Stringa P

$P \equiv P[0]P[1]\dots P[m-1]$ (lunghezza m)
$P[i]$ = singolo carattere $\in \Sigma$, Σ = alfabeto finito

Sottostringa

$P[i\dots j] \equiv P[i]P[i+1]\dots P[j]$, per qualche $0 \leq i, j < m$

- Si dice *propria* se $i > 0$ o $j < m-1$
- Si dice *vuota* se $i > j$

Prefisso di P

$P[0\dots i]$, per qualche $0 \leq i < m$

Suffisso di P

$P[i\dots m-1]$, per qualche $0 \leq i < m$

In Java: la classe `String` viene usata per stringhe non modificabili, mentre la classe `StringBuilder` viene usata per stringhe modificabili.

Pattern Matching Problem

Dati:

- *testo* T (stringa) di lunghezza n su Σ
- *pattern* P (stringa) di lunghezza $m \leq n$ su Σ

trovare, se esiste, l'indice $i \in [0, n-m]$ tale che $P = T[i \ldots i+m-1]$ o, equivalentemente, tale che $P[j] = T[i+j], \forall j : 0 \leq j < m$.

Problema Computazionale Π

$\mathcal{I} = \{(T, P) \ : \ T, P \text{ stringhe su } \Sigma \text{ con } |P| \leq |T|\}$

$\mathcal{S} = \{\text{interi } \geq -1\}$

$\Pi =$ insieme delle coppie $(T, P), i)$ con $(T, P) \in \mathcal{I}$ e $i \in \mathcal{S}$, tali che

- se P è sottostringa di T allora $i \geq 0$ e $P = T[i \ldots i+m-1]$
- altrimenti $i = -1$.

Algoritmo `findBrute` (approccio banale)

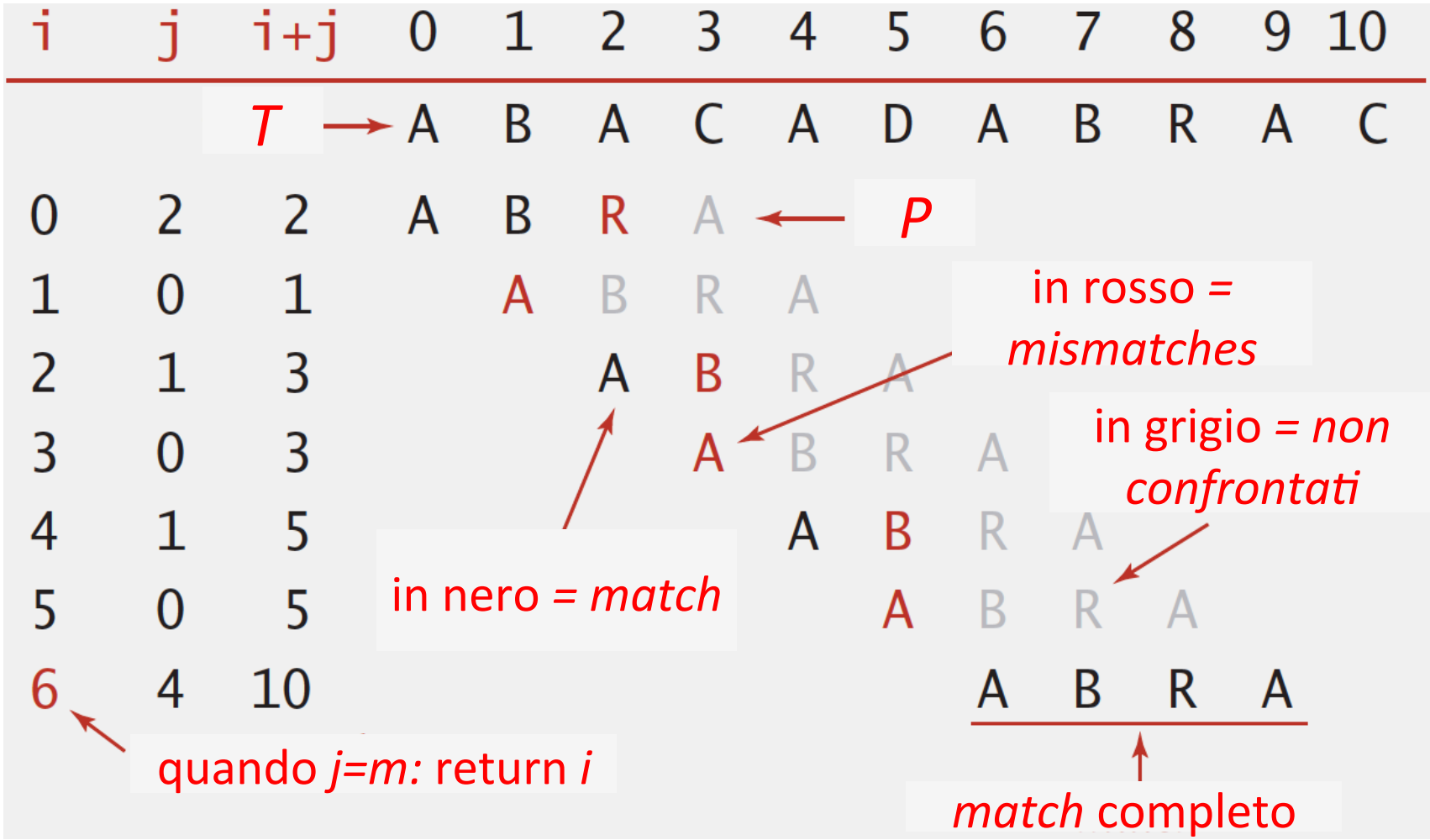
Algoritmo `findBrute(`T`,`P`)`
Input: T,P su Σ, $|T| = n$, $|P| = m$
Output: i t.c. $P = T[i \ldots i+m-1]$ se esiste, -1 altrimenti

for $i \leftarrow 0$ *to* $n - m$ **do**
    `/* verifica se` $P = T[i \ldots i+m-1]$ `*/`
    $j \leftarrow 0$;
    while $((j < m)$ AND $(T[i+j] = P[j]))$ **do** $j \leftarrow j+1$;
    if $j = m$ **then return** i;
return -1

Osservazione: la correttezza dell'algoritmo è banale.

Esempio

T = ABACADABRAC
P = ABRA



Complessità

Per quanto riguarda la complessità osserviamo che:

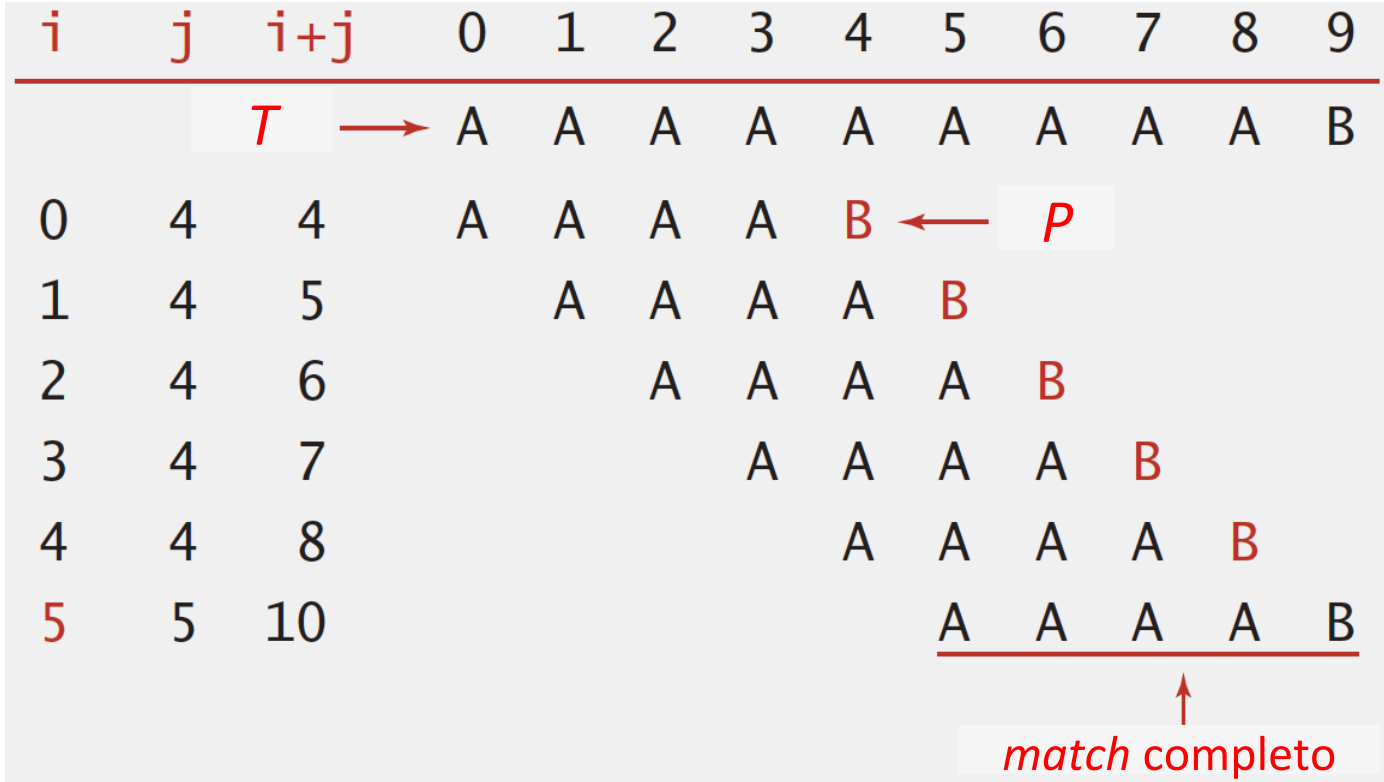
- $\leq n - m + 1$ iterazioni del for
- $\leq m$ iterazioni del while in ciascuna iterazione del for
- numero costante di operazioni in ciascuna iterazione del for (oltre al while) e in ciascuna iterazione del while.

$\Rightarrow t(n, m) \in O((n - m + 1)m)$

È *tight* (Esercizio C-12.14 [GTG14])?

Si consideri la seguente istanza

- $T = \mathrm{AA}\ldots\ldots\mathrm{AB}$
- $P = \mathrm{AA}\ldots\mathrm{AB}$



esattamente $m-1$ iterazioni del while in ciascuna iterazione del for

$$\Rightarrow t(n,m) \in \Omega\left((n-m+1)\,m\right) \Rightarrow t(n,m) \in \Theta\left((n-m+1)\,m\right)$$

Osservazioni

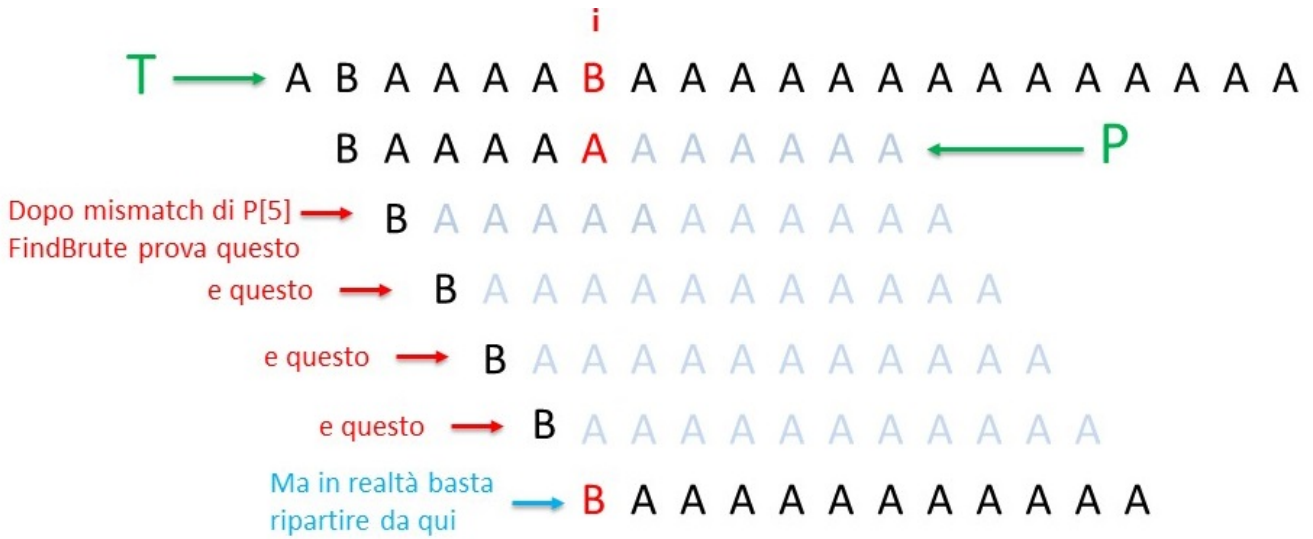
- se $m \leq cn$ con $c \in (0,1)$ costante $\Rightarrow t(n,m) \in \Theta\left(nm\right)$
- se $m = cn$ con $c \in (0,1)$ costante $\Rightarrow t(n,m) \in \Theta\left(n^2\right)$

Esercizio

Si consideri un testo T e un pattern P, con $|T| = n$, $|P| = m \leq n$ e si supponga che $P[0] \neq P[i], \forall 1 \leq i \leq m-1$.

1. Modificare `findBrute` per risolvere efficientemente le istanze del problema di pattern matching che soddisfano queste ipotesi. L'algoritmo modificato deve contenere un solo ciclo e deve avere complessità $O(n)$.
2. Analizzare la complessità dell'algoritmo
3. Identificare un opportuno invariante che serva per provare la correttezza del ciclo.

Soluzione: intuizione



Osservazione. Le ipotesi su P implicano che se per un certo allineamento di P sotto T si trova il primo mismatch confrontando $P[j]$ e $T[i]$ ($j = 5$ e $i = 6$ nell'esempio), il prossimo confronto utile è tra $P[0]$ e $T[i]$

Soluzione: algoritmo

Algoritmo `FBPlus(`T`,` P`)`
Input: T,P su Σ, $|T| = n$, $|P| = m$: $P[0] \neq P[i]$ per $i \neq 0$
Output: [il solito]

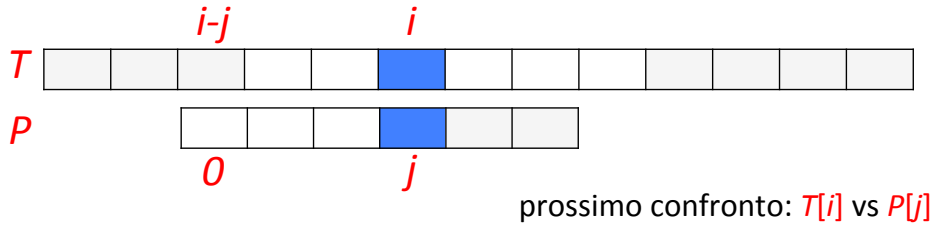
```
i ← 0; j ← 0;
while i < n do
    if P[j] = T[i] then
        i ← i + 1; j ← j + 1;
        if j = m then return i − j;
    else
        if j > 0 then j ← 0;
        else i ← i + 1;
return −1
```

Soluzione: analisi

Complessità:

- proporzionale al numero di iterazioni del while
- i aumenta di 1 almeno ogni 2 iterazioni e non decresce mai $\Rightarrow O(n)$
- tempo è $\Omega(n)$: $P = T[n-m \ldots n-1] \Rightarrow \Theta(n)$

Invariante per il while:



- $P \neq T[\ell \ldots \ell + m - 1], \forall 0 \leq \ell < i - j$
- $P[0 \ldots j-1] = T[i-j \ldots i-1]$.

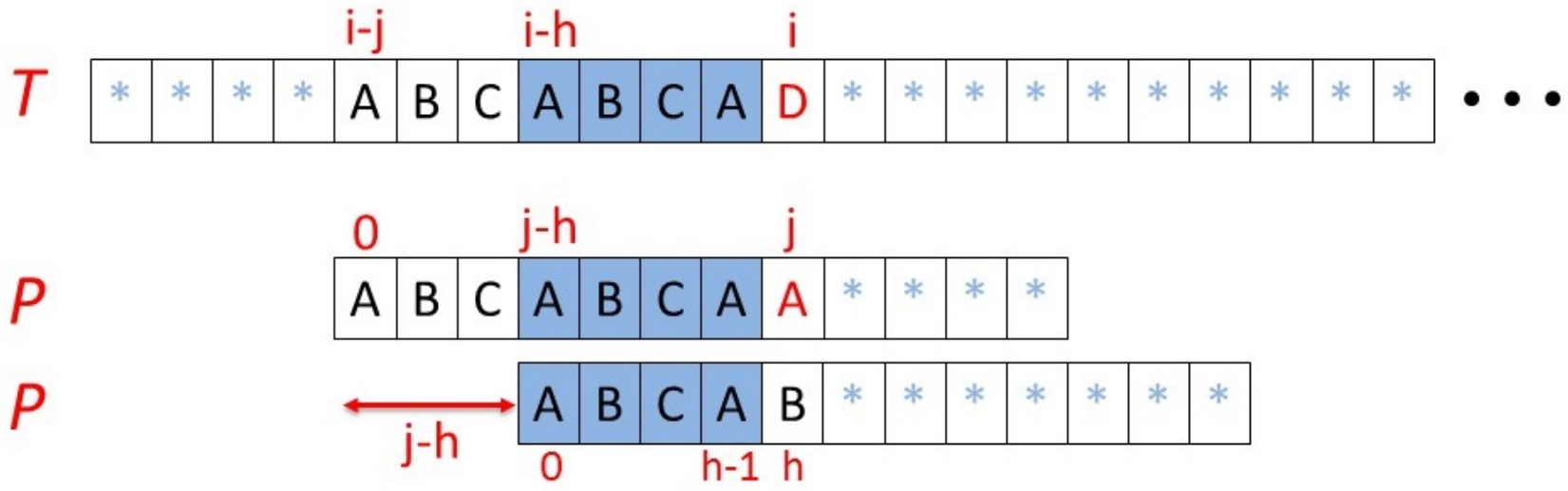
Algoritmo `findKMP` (di Knuth-Morris-Pratt)

Idea

- generalizza in modo intelligente il caso particolare dell'esercizio precedente
- simile all'approccio banale
- se la verifica di una posizione iniziale per P in T fallisce, salta posizioni successive che possono essere *sicuramente escluse* e sfrutta l'informazione acquisita rimanendo fermo su T

Idea principale

Supponiamo che $P[j] \neq T[i]$ e $T[i-j+s] = P[s]$ per $0 \leq s < j$:



Sia h il *massimo* valore per cui $P[0 \ldots h-1] = P[j-h \ldots j-1]$

$\Longrightarrow$ posso shiftare P verso destra di $j-h$ posizioni evitando di considerare tutti gli allineamenti in cui $P[0]$ è sotto $T[r]$ con $i-j < r < i-h$

Failure Function

Definizione

Failure function f per un pattern P:

- $f(0) = 0$
- per $j > 0$: $f(j)$= lunghezza del più lungo prefisso di P che è suffisso di $P[1 \ldots j]$

N.B. $f(j) \leq j$

Definizione equivalente, più matematica

Definizione

$$f(j) = \begin{cases} 0 & j = 0 \\ \max\{h : 0 \leq h \leq j \wedge P[0 \ldots h-1] = P[j-h+1 \ldots j]\} & j > 0 \end{cases}$$

Esempio

	0	1	2	3	4	5
$P =$	A	B	A	C	A	B

$f(0) = 0$

$f(1) = 0$

A	B	
A		NO

$f(2) = 1$

A	B	A	
A	B		NO
A			SI

$f(3) = 0$

A	B	A	C	
A	B	A		NO
A	B			NO
A				NO

$f(4) = 1$

A	B	A	C	A	
A	B	A	C		NO
A	B	A			NO
A	B				NO
A					SI

$f(5) = 2$

A	B	A	C	A	B	
A	B	A	C	A		NO
A	B	A	C			NO
A	B	A				NO
A	B					SI

N.B. La failure function $f(i)$ è definita per tutti i valori $0 \leq i < m$, tuttavia $f(m-1)$ non verrà utilizzata da `findKMP`

Altro Esempio

	0	1	2	3	4
$P =$	A	A	A	A	B

$f(0) = 0$
$f(1) = 1$
$f(2) = 2$
$f(3) = 3$
$f(4) = 0$

Algoritmo `findKMP(`T`,`P`)`
Input: T,P su Σ, $|T| = n$, $|P| = m$
Output: i t.c. $P = T[i \ldots i + m - 1]$ se esiste, -1 altrimenti

```
f ← computeFailKMP(P);
i ← 0; j ← 0;
while i < n do
    if P[j] = T[i] then                          /* caso 1 */
        i ← i + 1; j ← j + 1;
        if j = m then return (i − m);
    else
        if j > 0 then j ← f(j − 1);              /* caso 2 */
        else i ← i + 1;                          /* caso 3 */
return −1
```

Complessità

- Escludendo il calcolo di f, la complessità risulta essere proporzionale al numero di iterazioni del while.
- Definiamo $\ell(i,j) = i - j$ e osserviamo che:

  1. "$0 \leq i \leq n$" $\wedge$ "$j \geq 0$" $\wedge$ "i non decresce mai"
  2. "$\ell(i,j)$ non decresce mai" $\wedge$ "$\ell(i,j)$ parte da 0"
  3. 1) e 2) $\Rightarrow 0 \leq \ell(i,j) \leq n$
  4. in ciascuna iterazione: i o $\ell(i,j)$ (o entrambi) crescono di 1

  Si ha che 1), 3), 4) $\Rightarrow$ num. iterazioni $\leq 2n$

$\Rightarrow t_{\texttt{KMP}} \in O\left((\text{calcolo di } f) + n\right)$

Esempio

	0	1	2	3	4	5
P =	A	B	A	C	A	B

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T =	A	B	A	C	A	C	B	A	B	A	B	A	C	A	B

	0	1	2	3	4	5
P =	A	B	A	C	A	B

i	j	CASO	
5	5	2	(dopo 5 casi 1)
5	1	2	
5	0	3	
6	0	3	
10	3	2	(dopo 3 casi 1)
10	1	FINE	(dopo 5 casi 1)

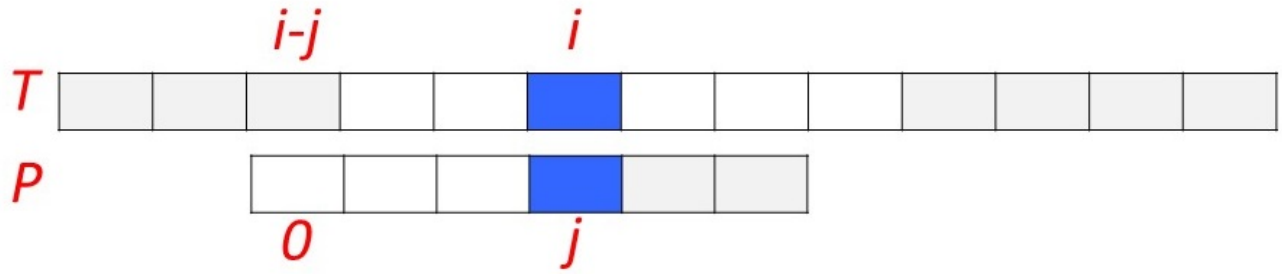
confronto e match

confronto e mismatch

Correttezza

Assumiamo computeFailKMP corretta

Determiniamo un invariante che valga alla fine di una generica iterazione del while (prossimo confronto $T[i]$ vs $P[j]$)



L'invariante è il seguente

- $P \neq T[\ell \ldots \ell + m - 1], \forall \ell : 0 \leq \ell < i - j$
- $P[0 \ldots j - 1] = T[i - j \ldots i - 1]$

Prova dell'invariante

- è vero all'inizio: per vacuità
- da una iterazione alla successiva: mantenuto
  - casi 1 e 3: ovvi
  - caso 2: grazie alla failure function
- alla fine dell'ultima iterazione
  - $i \leq n$ e $j = m \Rightarrow$ return i
  - $i = n$ e $j < m \Rightarrow$ return -1

  In entrambi i casi, l'invariante assicura la correttezza

Aggiungere i dettagli dei passaggi per esercizio

Esercizio

Si consideri un testo T di n caratteri e un pattern P di $m \leq n$ caratteri dove

$$\begin{aligned} T &= AA\ldots\ldots A \\ P &= AA\ldots AB \end{aligned}$$

1. Determinare la failure function di P
2. Descrivere l'esecuzione di `findKMP` per $n = 9$ e $m = 5$
3. Per n, m generici, determinare il numero di confronti che coinvolgono ciascun $T[i]$.

Soluzione

1. Determinare la failure function di P

Esempio: m=5

	0	1	2	3	4
P =	A	A	A	A	B

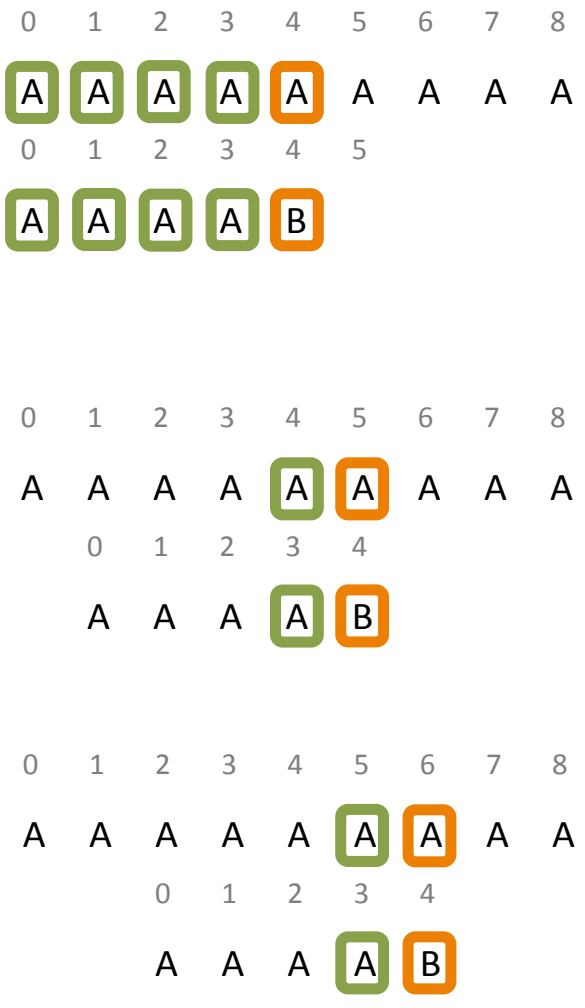
$f(0) = 0$; $f(1) = 1$; $f(2) = 2$; $f(3) = 3$; $f(4) = 0$

In generale:

- $f(0) = 0$
- Per $1 \leq j \leq m-2$: $f(j) = j$. Infatti
  - $P[1 \ldots j] = AA \ldots A$ (j volte A)
  - $P[0 \ldots j-1] = AA \ldots A$ (j volte A)
- $f(m-1) = 0$ (nessun prefisso proprio di P finisce con B)

Soluzione (continua)

2. Descrivere l'esecuzione di `findKMP` per $n = 9$ e $m = 5$



i	j
0	0
1	1
2	2
3	3
4	4
4	3
5	4
5	3
6	4
⋮	

confronto e match

confronto e mismatch

Soluzione (continua)

3. Per n, m generici, determinare il numero di confronti che coinvolgono ciascun $T[i]$.

i	j
0	0
$\vdots$	$\vdots$
$m-1$	$m-1$
$m-1$	$m-2$
m	$m-1$
m	$m-2$
$m+1$	$m-1$
$m+1$	$m-2$
$m+2$	$m-1$
$m+2$	$m-2$
$m+3$	$m-1$
$\vdots$	$\vdots$

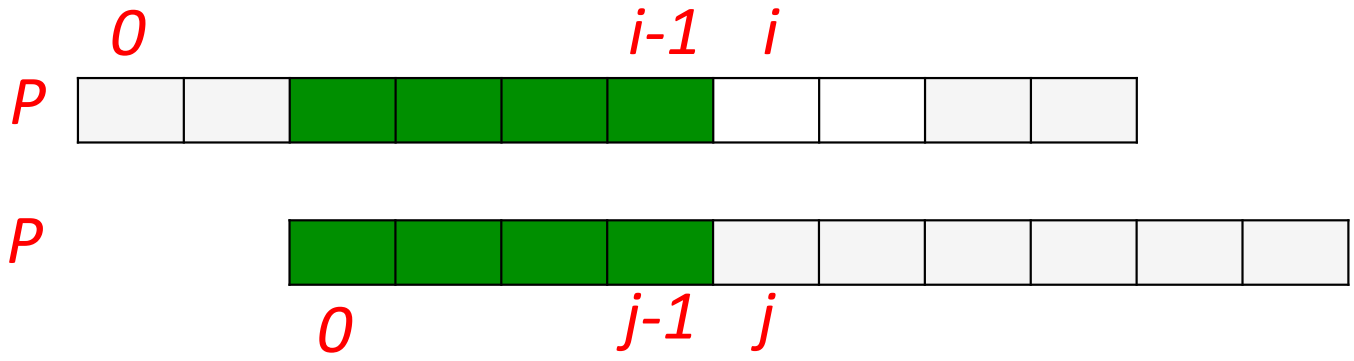
- $\forall i : 0 \leq i < m-1$ l'algoritmo fa 1 solo confronto con $T[i]$
- $\forall i \geq m-1$ l'algoritmo fa 2 soli confronti con $T[i]$

N.B.: arrivato al mismatch sull'ultimo carattere di P, non deve riprendere i confronti da $P[0]$ ma sfrutta il fatto che $P[0 \ldots m-2]$ andava bene e $f(m-2) = m-2$.

Calcolo della Failure Function per un pattern P

Idea

Ho calcolato $f(i-1), f(i-2), \ldots, f(1), f(0)$. Calcolo di $f(i)$:

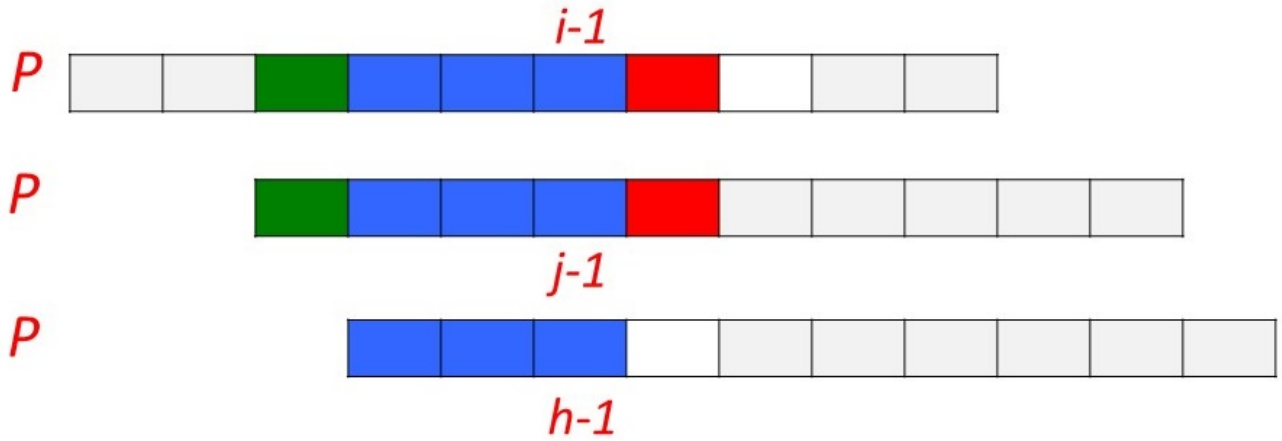


Supponiamo che $f(i-1) = j$, e quindi $f(i) \leq j+1$. Abbiamo allora 3 casi (analoghi a quelli di `findKMP`):

- **Caso 1:** $P[i] = P[j]$. Allora, ovviamente $f(i) = j+1$

Calcolo della Failure Function per un pattern P

- **Caso 2:** $P[i] \neq P[j]$ e $j > 0$. Sia $h = f(j-1)$.



  per definizione di f le sottostringhe in blu, che hanno lunghezza h, sono uguali e quindi il prossimo valore candidato per $f(i)$ è $h+1$. Imposto quindi j a h (ovvero a $f(j-1)$) e reitero il procedimento

- **Caso 3:** $P[i] \neq P[j]$ e $j = 0$. IN questo caso, semplicemente imposto $f(i)$ a 0 e passo al calcolo di $f(i+1)$.

Algoritmo `computeFailKMP(`P`)`
Input: pattern P, $|P| = m$
Output: failure function f for P

$i \leftarrow 1$; $j \leftarrow 0$; $f(0) \leftarrow 0$;
while $i < m$ **do**
  if $P[j] = P[i]$ **then** `/* caso 1 */`
    $f(i) \leftarrow j + 1$;
    $i \leftarrow i + 1$;
    $j \leftarrow j + 1$
  else
    if $j > 0$ **then** $j \leftarrow f(j - 1)$; `/* caso 2 */`
    else `/* caso 3 */`
      $f(i) \leftarrow 0$;
      $i \leftarrow i + 1$
return f

Esempio

$P = AAABACD$

	0	1	2	3	4	5	6
P =	A	A	A	B	A	C	D
		A					
		A	A				
		A	A	A			
			A	A			
				A			
					A		
					A	A	
						A	
							A

i	j	$f(i)$	CASO
1	0	1	1
2	1	2	1
3	2	-	2
3	1	-	2
3	0	0	3
4	0	1	1
5	1	-	2
5	0	0	3
6	0	0	3

Complessità

- Proporzionale al numero di iterazioni del while
- Definiamo $\ell(i,j) = i - j$ e osserviamo che

  1. "$1 \leq i \leq m$" $\wedge$ "$j \geq 0$" $\wedge$ "i non decresce mai"
  2. $\ell(i,j)$ non decresce mai e parte da 1
  3. 1), 2) $\Rightarrow 1 \leq \ell(i,j) \leq m$
  4. in ciascuna iterazione uno dei due valori i o $\ell(i,j)$ (o entrambi) crescono di 1

  Si ha che 1), 2), 4) $\Rightarrow$ num. iterazioni $\leq 2m$

$\Rightarrow$ complessità $\in O(m)$

N.B. Il ciclo non può durare meno di $m - 1$ iterazioni
$\Rightarrow$ complessità $\in \Theta(m)$

Ripresa: complessità di `findKMP`

$t_{\texttt{findKMP}}(n, m) \in O(n + m) = O(n)$ (dato che nel problema computazionale: $m \leq n$)

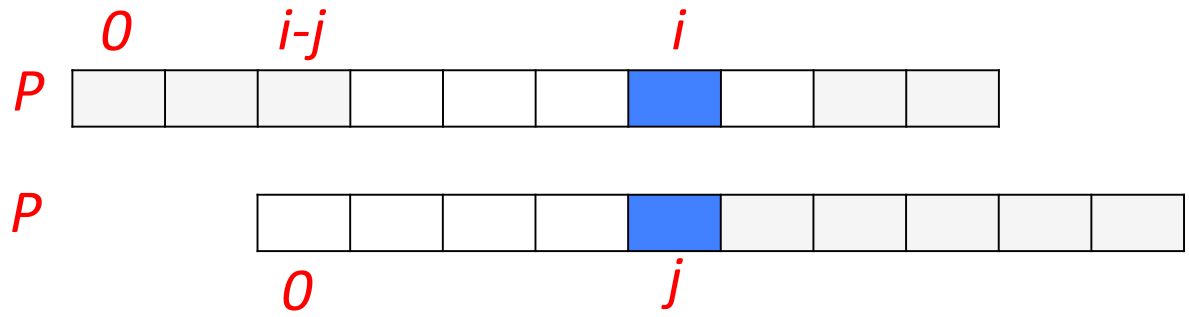
$t_{\texttt{findKMP}}(n, m) \in \Omega(n)$?

Sì!

Se P non è sottostringa di T il ciclo while termina con $i = n$ $\Rightarrow \Theta(n)$ iterazioni.

Correttezza di `computeFailKMP`

Determiniamo un invariante che vale alla fine di una generica iterazione del while (prossimo confronto $P[i]$ vs $P[j]$)



L'invariante è il seguente

- $f(0), f(1), \ldots, f(i-1)$: calcolati correttamente
- $f(i) \leq j + 1$
- $P[0 \ldots j-1] = P[i-j \ldots i-1]$

Esercizio

Sfruttando l'invariante specificato sopra, dimostrare la correttezza di `computeFailKMP`

Esercizi

Esercizio (C-12.19 [GTG14])

Dati un testo T e un pattern P, con $|T| = n$, $|P| = m$, si vogliono trovare inizio e lunghezza del più lungo prefisso di P in T in tempo $O(n + m)$.

1. Modificare l'algoritmo `findKMP` per risolvere il problema.
2. Far vedere che la complessità dell'algoritmo è $\Theta(n + m)$

Esercizio

Dati un testo T e un pattern P, con $|T| = n$, $|P| = m \leq n$, si vogliono trovare *tutte* le occorrenze di P in T.

1. Modificare l'algoritmo `findKMP` per risolvere il problema.
2. Analizzare la complessità dell'algoritmo.

Esercizi

Esercizio

Dati un testo T e un pattern P, con $|T| = n$, $|P| = m \leq n$, si vuole determinare se P è *sottostringa circolare* di T, cioè se:

- P è sottostringa ("normale") di T; oppure
- P si può ottenere concatenando un suffisso di T a un prefisso di T.

(Ad esempio, DBA è sottostringa circolare di ABCDDB.)
Progettare un algoritmo che risolva il problema in tempo $O(n)$.

Riepilogo

- Definizioni: stringa, sottostringa, prefisso, suffisso
- Pattern matching come problema computazionale
- Algoritmo `findBrute`
  - complessità
  - correttezza
- Algoritmo `findKMP`
  - failure function
  - complessità
  - correttezza
- Algoritmo `computeFailKMP`
  - complessità
  - correttezza

Esempio domande prima parte

- Determinare la failure function per il pattern $P = \text{STATISTA}$.
- Mostrare un'istanza (T, P) su cui l'algoritmo `findBrute` esegue $\Theta((n - m + 1)m)$ operazioni.
- Analizzare la complessità di `findKMP` supponendo che il calcolo della failure function sia fatto con complessità lineare.
- Si consideri l'algoritmo `findKMP` e si supponga che in una certa iterazione del while si trovi $T[i] \neq P[j]$ con $j > 0$. Dire quale confronto farà l'algoritmo nella successiva iterazione, spiegando perché ciò è corretto.

Errata

Cambiamenti rispetto alla prima versione dei lucidi:

- Lucido 34: modificata la seconda domanda del secondo esercizio