

# Approfondimento sul Pattern Matching 2D

Liva Giovanni

Università di Udine

3 giugno 2014

## 1 Lavori Precedendi

- Exact vs Approximate
- Approximate Dictionary Matching
- Baker and Bird
- Zhu and Takaoka

## 2 2D con FA

## 3 2D con PDA

# Definizioni di Base

Dimensioni  $-i$  ( $m*m'$ ) Testo 2D;  $m$  Testo 1D

( $n*n'$ ) Pattern 2D  $-i$   $n$  Pattern 1D

$k$  errori ammessi

$D_H$  = hamming

$P$  = pattern

$T$  = testo

Il pattern matching esatto prevede di cercare l'occorrenza di una stringa all'interno di un testo. L'uso di automi per questo approccio è abbastanza naturale come si vede dal seguente algoritmo:

---

**Algorithm 1** Creazione FA :: Pattern - Matching Esatto

---

- 1:  $\delta(q_0, a) = \{q_0\}, \forall a \in A$
  - 2:  $\delta(q_0, p_1) = \{q_0, q_1\}$
  - 3: **for**  $i = 1$  to  $m - 1$  **do**
  - 4:      $\delta(q_i, p_{i+1}) = \{q_{i+1}\}$
  - 5: **end for**
- 

L'automa risultante ha  $m + 1$  stati. Ogni stato  $q_i$  indica che si è letto il prefisso del pattern fino all' $i$ -esimo carattere

# Exact vs Approximate

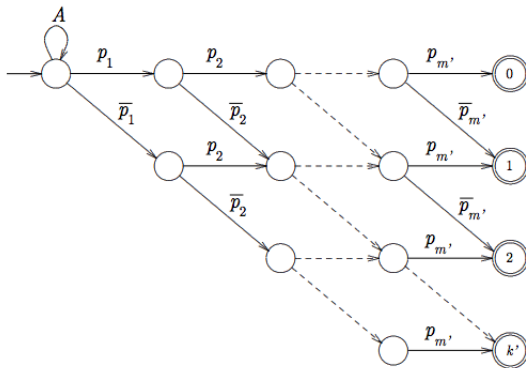
Il pattern matching approssimato si basa sulla distanza di Hamming che permette di quantificare il numero di errori ammessi. La costruzione di tale automa prevede l'uso di  $k + 1$  copie di automi per il Pattern Matching esatto,  $M_0, \dots, M_k$ .

L'idea è quella che ogni  $M_i$  rappresenta il pattern accettato con  $i$  errori. I vari  $M_i$  sono collegati con una transizione dallo stato  $q_j$  allo stato  $q_{j+1}$  che corrisponde all'azione di *sostituzione* nel calcolo della distanza di Hamming, etichettata con il simbolo  $\bar{p}_{j+1}$  corrispondente al carattere complementare in posizione  $j + 1$  in  $P$ .

# Exact vs Approximate

## Theorem

*L'automa per il Pattern Matching approssimato ha  $(k + 1)(m + 1 - k/2)$  stati*



# Approximate Dictionary Matching

## Definition

Sia  $\pi$  un dizionario di  $s$  pattern,  $\pi = \{p_1, \dots, p_s\}$

Sia  $m = \min\{|p_1|, \dots, |p_s|\}$

Sia  $k$  il numero di errori ammessi per ogni pattern,  $k < m$

L'automa  $\mathcal{A}$  per l'approximate dictionary matching riconosce il linguaggio

$$L(\mathcal{A}) = \bigcup_{i=1}^s \{uv \mid u, v \in A^*, D_H(p_i, v) \leq k, p_i \in \pi\}$$

---

## Algorithm 2 Creazione FA :: Approximate Dictionary Matching

---

- 1: **for**  $i = 1$  to  $s$  **do**
- 2:   Costruisci  $M_i$  con la tecnica per il pattern matching approssimato
- 3: **end for**
- 4: Costruisci lo stato iniziale  $q_0$
- 5: **for**  $i = 1$  to  $s$  **do**
- 6:   Aggiungi una transizione da  $q_0$  a  $q_0^i$  e da  $q_0$  a  $q_1^i$  etichettata come la transizione  $q_0^i \rightarrow q_1^i$
- 7: **end for**

## Definition

Sia  $\pi$  il dizionario ottenuto da PA,  $\pi = \{p_i | p_i \text{ è la } i\text{-esima colonna di PA}\}$

Sia  $\mathcal{A}$  l'automa ottenuto da PA con l'algoritmo di *Aho – Corasick*

Sia  $TA'$  il *textarray* ottenuto lanciando  $\mathcal{A}$  su ogni colonna di TA e salvando lo stato corrente della run dell'automa

- Linearizzare  $TA'$  ottenendo  $T$
- Linearizzare  $PA$  ottenendo  $P$
- Usare KMP su  $P$  e  $T$
- La complessità finale è  $\mathcal{O}(mm' + nn')$



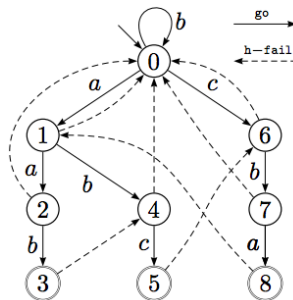
# Esempio

$$PA = \begin{bmatrix} a & c & a \\ b & b & a \\ c & a & b \end{bmatrix}, TA = \begin{bmatrix} b & b & a & b & b & a & b \\ a & a & c & a & c & b & a \\ b & b & b & a & c & a & c \\ a & c & a & b & b & a & b \\ c & a & a & c & a & b & a \\ b & b & b & b & a & c & c \\ a & c & c & a & b & a & b \end{bmatrix}, |PA| = (3 \times 3), |TA| = (7 \times 7).$$

# Esempio

$$PA = \begin{array}{|c|c|c|} \hline a & c & a \\ \hline b & b & a \\ \hline c & a & b \\ \hline \end{array}$$

5 8 3



# Esempio

$TA =$

$b$	$b$	$a$	$b$	$b$	$a$	$b$
$a$	$a$	$c$	$a$	$c$	$b$	$a$
$b$	$b$	$b$	$a$	$c$	$a$	$c$
$a$	$c$	$a$	$b$	$b$	$a$	$b$
$c$	$a$	$a$	$c$	$a$	$b$	$a$
$b$	$b$	$b$	$b$	$a$	$c$	$c$
$a$	$c$	$c$	$a$	$b$	$a$	$b$

$TA' =$

0	0	1	0	0	1	0
1	1	6	1	6	4	1
4	4	7	2	6	1	6
1	5	8	3	7	2	7
6	1	1	5	8	3	8
7	4	4	7	2	5	6
8	5	5	8	3	1	7

# Esempio

	a	c	a		
	b	b	a	c	a
	c	a	b	b	a
		a	c	a	b
		b	b	a	
		c	a	b	

0	0	1	0	0	1	0
1	1	6	1	6	4	1
4	4	7	2	6	1	6
1	<b>5</b>	<b>8</b>	<b>3</b>	7	2	7
6	1	1	<b>5</b>	<b>8</b>	<b>3</b>	8
7	4	4	7	2	5	6
8	5	<b>5</b>	<b>8</b>	<b>3</b>	1	7

Estende l'idea di Karp e Rabin alle due dimensioni usando la tecnica delle *impronte*.

I passi dell'algoritmo sono:

- Genera  $P'$  come array di impronte di  $P$  usando la funzione di hash per colonne
- Genera  $T'$  nello stesso modo di  $P'$  (Guardando solo  $m'$  caratteri per colonna)
- Lancia KMP su  $P'$  e  $T'$  e ogni volta che trova una occorrenza esegue il controllo sulle matrici
- Passa alla riga successiva aggiornando  $T'$

Più il pattern ha la componente  $m'$  grande, migliore è l'efficienza di questo algoritmo che, nel caso pessimo è  $\mathcal{O}(n'n + n'mm')$

## Definition

Sia  $\pi$  il dizionario ottenuto da PA,  $\pi = \{p_i | p_i \text{ è la } i\text{-esima colonna di PA}\}$

Sia  $R$  la linearizzazione di PA

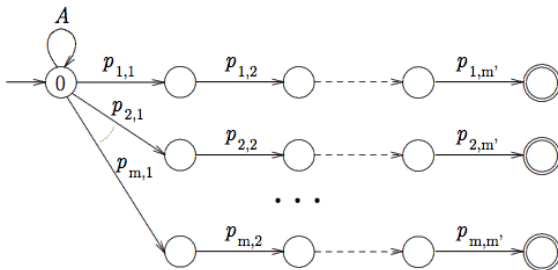
Sia  $M(\pi) = (Q, A, \delta, I, F)$  l'automa costruito con la tecnica per l'approximate dictionary matching

- Costruiamo  $M(\pi)$  con  $k'$  errori
- Costruiamo  $TA'$  ottenuto dai valori della run di  $M(\pi)$  su TA
- Costruiamo  $R$  come l'insieme degli stati finali di  $M(\pi)$  che rappresentano il match esatto di PA
- Costruiamo  $M'$  che cerca  $R$  con  $k$  errori (limitiamo la somma degli errori del punto 2)
- Lanciamo  $M'$  su  $TA'$  in modo che rilevi tutte le occorrenze di  $R$

## Caso Esatto :: Costruzione $M(\pi)$

$M(\pi)$  riconosce il linguaggio  $L = A * E(\pi)$  con  $E(\pi) = \{P \in A^{m'}; P \in \pi\}$

La costruzione di tale automa consiste in un *Trie* con un self loop sullo stato iniziale etichettato con  $A$  e  $m$  transizioni verso gli automi per i singoli pattern.



## Theorem

*L'automata ha al più uno stato finale attivo dopo aver letto un carattere d'input.*

## Dimostrazione.

Siccome  $\pi$  è un insieme, non esistono duplicati di una stringa. Visto che ogni stringa all'interno di  $\pi$  ha la stessa lunghezza, dopo ogni step dell'automata solo, al più, uno stato finale può essere attivo.  $\square$

Questo teorema vale grazie a come viene eseguita la simulazione di  $M(\pi)$ .



Grazie al teorema precedente possiamo costruire  $TA'$  usando la seguente formula:

$$\forall i, j \ 1 \leq i \leq n, 1 \leq j \leq n' \quad TA'[i, j] = \begin{cases} q & \text{se } q \in F \\ 0 & \text{se } q \notin F \end{cases}$$

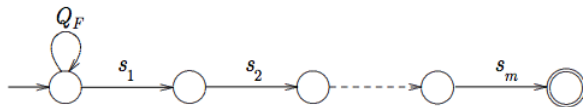
## Caso Esatto :: Costruzione $R$

La stringa  $R$  è ottenuta linearizzando PA grazie all'automa  $M(\pi)$  costruito in precedenza.

L' $i$ -esimo carattere di  $R$  è ottenuto leggendo lo stato finale di  $M(\pi)$  su input  $PA[i]$ .

## Caso Esatto :: Costruzione $M'$

Costruiamo l'automa  $M'$  con il solito algoritmo sull'alfabeto  $F \cup \{0\}$  che riconosca il linguaggio  $L(M') = (F \cup \{0\})^* E(R)$  dove  $E(R) = \{R \mid R \in F^m\}$ ,  $F = \{s_1, \dots, s_{|\pi|}\}$ ,  $|F| = |\pi|$ .



- Tutti gli automi presentati sono non deterministici
- Bisogna utilizzare una simulazione per non dover determinizzare gli automi
- Opera in tempo lineare:  $\mathcal{O}(mm' + nn')$ 
  - $\mathcal{O}(mm')$  per  $M(\pi)$
  - $\mathcal{O}(nn')$  per  $TA'$
  - $\mathcal{O}(m)$  per  $R$  ed  $M'$
  - $\mathcal{O}(nn')$  per il pattern matching

Dobbiamo modificare  $M(\pi)$  in modo che accetti il linguaggio

$L(M) = A * H_k(\pi)$  dove:

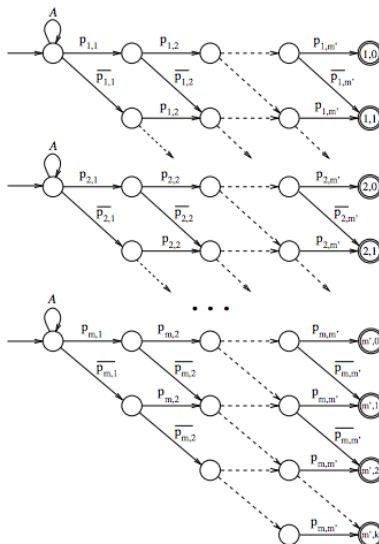
$$H_k(\pi) = \{X \mid X \in A^*, D_H(X, P) \leq \min(k, m' - 1) \wedge P \in \pi\}$$

Dobbiamo usare il minimo perchè  $k$  potrebbe essere più grande della dimensione di una colonna/riga.

Bisogna costruire l'automa per l'Approximate Dictionary Matching con l'algoritmo visto in precedenza con una modifica, gli stati finali sono etichettati con copie  $(s, x)$  dove:

- $s$  = l'indice del pattern riconosciuto
- $x$  = il numero di errori con il quale viene riconosciuto

# I singoli automi del ADM



## Definition

$$k' = \min(k, m' - 1)$$

## Theorem

*L'automa per il ha  $|\pi|(k' + 1)(m + 1 - k'/2)$  stati di cui  $|\pi|(k' + 1)$  finali*

## Theorem

*L'automa potrebbe avere più di uno stato finale attivo dopo aver letto un carattere d'input.*

Dobbiamo estendere  $TA'$  ad un array tridimensionale in quanto più pattern allo stesso istante potrebbero venire riconosciuti.

### Definition

Sia  $\emptyset \notin Q$  un simbolo nuovo che non appare negli stati e nelle label.  
 $\emptyset$  viene usato per indicare che nessun pattern è trovato in una data posizione

$$\forall i, j, s \ 1 \leq i \leq n, 1 \leq j \leq n', 1 \leq s \leq |\pi|$$
$$TA'[i, j, s] = \begin{cases} x & \text{se } q \text{ attivo, } q \in F, q = (s, x) \\ \emptyset & \text{altrimenti} \end{cases}$$



# Caso Approssimato :: Costruzione $R$

Per costruire  $R$  a partire da  $PA$  vorremmo usare lo stesso algoritmo visto per il caso esatto.

Nel caso approssimato  $M(\pi)$  ha più stati finali per ogni pattern  $p \in \pi$ , bisogna quindi apportare i seguenti aggiustamenti:

- Per ogni sottoautoma  $M_i$  di  $M(\pi)$  viene selezionato (arbitrariamente) uno stato finale
- Lo stato finale scelto è della forma  $(s,x)$  e per costruire  $R$  guardiamo solo alla componente  $s$

# Caso Approssimato :: Costruzione di $M'$

L'idea è quella di costruire  $k$  copie dell'automa che riconosce  $R$  in maniera esatta  $(M_1, \dots, M_k)$ , dove ogni coppia è numerata e il suo indice permette di contare quanti errori sono stati compiuti.

## Definition

Sia  $M' = (Q', A', \delta', q'_{0,0}, F')$  dove

- $A' = F \cup \{\emptyset\}$
- $\delta' : Q' \times (s, A') \rightarrow \mathcal{P}(Q')$
- $q_{i,j}$  stato  $j$ -esimo dell'automa  $i$ -esimo

## Caso Approssimato :: Costruzione di $M'$

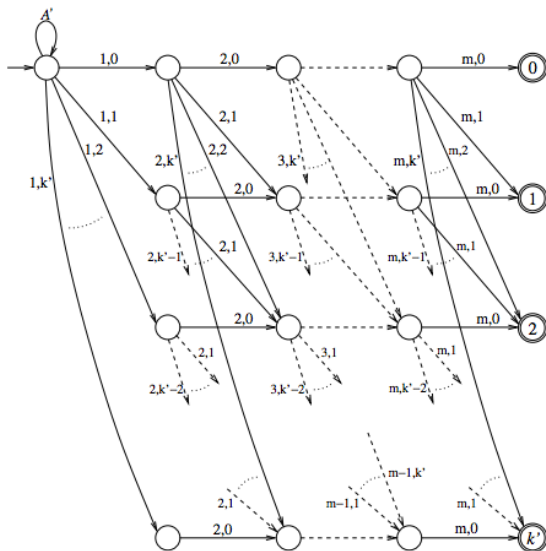
Ogni stato  $q_{i,j}$  ha una transizione verso lo stato  $q_{k,j+1}(\forall k.i \leq k \leq m')$  etichettata con  $(p, z)$  dove:

- $p$  indica l' $p$ -esimo carattere di  $R$  riconosciuto
- $z$  indica il numero di errori che si commette eseguendo tale transizione

Per calcolare  $x$  si utilizza la seguente funzione:

$$|(s, x) - (t, y)| = \begin{cases} |x - y| & s = t \\ m' & ((s, t) \vee (t, y)) = \emptyset \\ m' & s \neq t \end{cases}$$

# Caso Approssimato :: Costruzione di $M'$



Gli automi costruiti in questi passaggi sono non deterministici, la simulazione prevede l'uso della *programmazione dinamica*.

Si introduce un array  $E[0, \dots, m - 1]$  che tiene conto del numero minimo di errori trovati.

Se uno stato attivo non compie nessuna transizione al passo successivo, questa computazione viene terminata.

## Algorithm 3 Simulazione

```
1: for  $y = m'$  to  $n'$  do
2:    $\forall q.E[q]$  inattivo
3:   for  $x = 1$  to  $n$  do
4:      $E[0]$  attivo,  $E[0] = 0$ 
5:     foreach  $q$  attivo in  $E$  do
6:        $err = TA'[x, y, R[q + 1]]$ 
7:       if  $err = \emptyset$  then
8:          $err = m'$ 
9:       end if
10:       $err = err + E[q]$ 
11:      if  $err \leq k$  then
12:        if  $q + 1 = m$  then
13:          "Trovata Occorrenza"
14:        else
15:           $E[q + 1]$  attivo,  $E[q + 1] = err$ 
16:        end if
17:      else
18:         $E[q + 1]$  disattivato al prossimo step
19:      end if
20:    end for
21:  end for
22: end for
```

- $M(\pi)$  è fatto da  $m$  sub-automi la cui simulazione su  $TA$  richiede  $m \cdot \mathcal{O}(m'nn')$
- $R$  è costruito in  $\mathcal{O}(m)$
- La simulazione di  $M'$  richiede  $\mathcal{O}(mnn' - mm'n)$

La complessità totale dell'algoritmo per trovare PA in TA con  $k$  errori chiede  $\mathcal{O}(mm' \cdot nn' + m + mnn' - mm'n) = \mathcal{O}(mm' \cdot nn')$

## Definition

Definiamo  $A_{\#} = A \cup \{\#\}$  con  $\#$  che non appare in  $A$

Sia  $M \in A^{(n \times n')}$ , definiamo la matrice estesa  $M' \in A_{\#}^{(n+1 \times n'+1)}$  come  $M$  con una nuova riga e colonna riempite con il carattere  $\#$

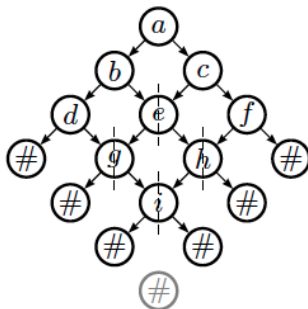
$$M = \begin{array}{|c|c|c|} \hline a & c & f \\ \hline b & e & h \\ \hline d & g & i \\ \hline \end{array} \rightarrow M' = \begin{array}{|c|c|c|c|} \hline a & c & f & \# \\ \hline b & e & h & \# \\ \hline d & g & i & \# \\ \hline \# & \# & \# & \# \\ \hline \end{array}$$



# Da matrice a DAG

Dalla matrice  $M'$  estesa possiamo creare un *DAG* seguendo le seguenti regole:

- La radice è in  $M'[1, 1] = M[1, 1]$
- Ogni nodo interno è formato dai valori di  $M$
- I caratteri  $\#$  sono tutte foglie
- Se un nodo  $M'[i, j]$  è interno, allora ha due figli:  $M'[i + 1, j]$  e  $M'[i, j + 1]$

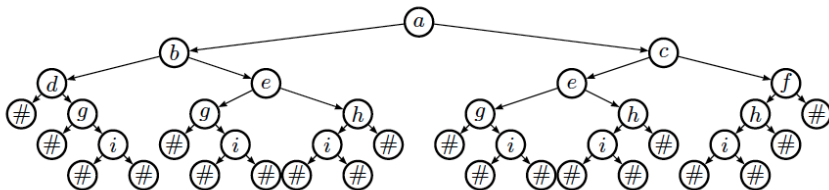


# Da DAG ad Albero

La radice dell'albero è la radice del DAG.

In ricorsione si vedono i figli di un nodo e li si copiano nell'albero.

Terminiamo quando arriviamo al nodo #



Possiamo costruire direttamente l'albero a partire dalla matrice.  
La struttura di DAG è mantenuta implicitamente.

$$tree(M, x, y) = \begin{cases} (M[x, y], tree(M, x, y + 1), tree(M, x + 1, y)) & \text{Se } M[x, y] \neq \# \\ (M[x, y], null, null) & \text{Altrimenti} \end{cases}$$

## Definition

Gli elementi della matrice che vengono visti più di una volta sono detti *splitting nodes*.

$pref(t)$  è la visita in preorder con la stampa delle label dei nodi.

## Theorem

*Sia  $t$  ottenuto da  $\text{tree}(M', 1, 1)$ . La profondità massima di un nodo in  $t$ ,  $\text{depth}(t)$  è  $\text{depth}(t) = n + n' - 1$ .*

## Dimostrazione.

Il percorso più lungo in  $t$  è quello che percorre  $n - 1$  nodi che non sono di *splitting* e  $n$  *splitting* nodes. □

## Theorem

*La costruzione dell'albero ha complessità  $\mathcal{O}(2^{\text{depth}(t)})$*

## Dimostrazione.

In un albero binario completo il numero di nodi è  $2^{\text{depth}(t)-1} + 2^{\text{depth}(t)} = \mathcal{O}(2^{\text{depth}(t)})$ . □

