Approfondimento sul Pattern Matching 2D

Liva Giovanni

Università di Udine

8 giugno 2014

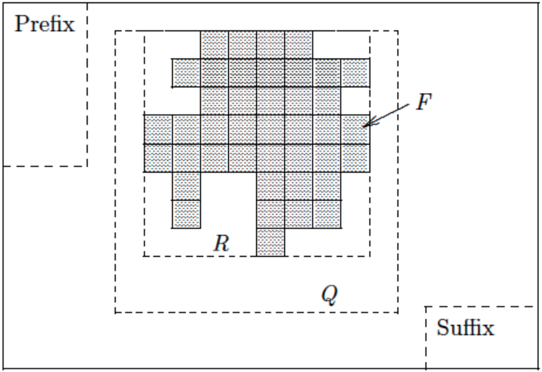
Argomenti

Definizioni di Base

Definition

- P viene usato per indicare un pattern unidimensionale di lunghezza m
- T viene usato per indicare un testo unidimensionale di lunghezza n
- PA viene usato per indicare un pattern bidimensionale di lunghezza $m \times m'$
- TA viene usato per indicare un testo bidimensionale di lunghezza $n \times n'$
- $D_H(x, y)$ è la funzione che calcola la distanza di hamming tra x ed y
- k viene usato per indicare il numero massimo di errorri ammessi

Definizioni di Base



Definition

- F è un fattore
- R è l'array di contenimento (minimo), così come Q

Exact vs Approximate

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

$\delta(q_0, a) = \{q_0\}, \forall a \in A$
for $i = 0$ to m **do**
  $\delta(q_i, p_i) = \{q_{i+1}\}$
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

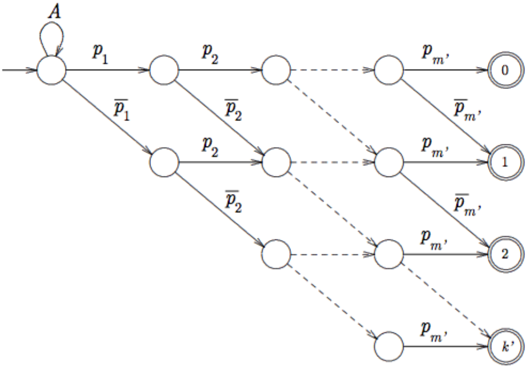
Exact vs Approximate

Il pattern matching approsimato 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 Matchin 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 $\overline{p}_{j+1}$ corrispondente al carattere complementare in posizione $j+1$ in P.

Exact vs Approximate

Theorem

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



Approximate Dictionary Matching

Definition

Sia π un dizionario di s pattern, $\pi = \{p_1, \ldots, p_s\}$
Sia $m = min\{|p_1|, \ldots, |p_s|\}$
Sia k il numero di errori ammessi per ogni pattern, $k < m$
L'automa $\mathcal{A}$ per l'approximate dictionary matching riconosce il linguaggio
$L(\mathcal{A}) = \bigcup_{i=1}^{s}\{uv|u, v \in A^*, D_H(p_i, v) \leq k, p_i \in \pi\}$

Algorithm 2 Creazione FA :: Approximate Dictionary Matching

```
for i = 1 to s do
    Costruisci M_i con la tecnica per il pattern matching approssimato
end for
Costruisci lo stato iniziale q_0
δ(q_o, a) = {q_0^i} ∀a ∈ A, ∀i ∈ {0, ..., s}
for i = 1 to s do
    Aggiungi una transizione da q_0 a q_1^i etichettata come la transizione q_0^i → q_1^i
end for
```

Baker and Bird

Definition

Sia π il dizionario ottenuto da PA, $\pi = \{p_i | p_i$ è la i-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

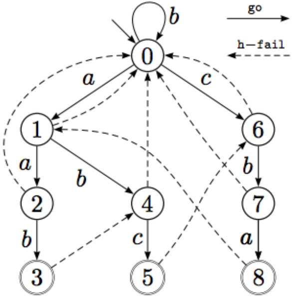
- Linearizzare TA' ottenendo T
- Linearizzare PA ottenendo P
- Usare KMP su P e T
- La complessità finale è $\mathcal{O}(mm' + nn')$

Esempio

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

Esempio

$$PA = \begin{array}{ccc} \boxed{a} & \boxed{c} & \boxed{a} \\ \boxed{b} & \boxed{b} & \boxed{a} \\ \boxed{c} & \boxed{a} & \boxed{b} \\ 5 & 8 & 3 \end{array}$$



Esempio

$$TA=\begin{array}{|c|c|c|c|c|c|c|}\hline b&b&a&b&b&a&b\\ a&a&c&a&c&b&a\\ b&b&b&a&c&a&c\\ a&c&a&b&b&a&b\\ c&a&a&c&a&b&a\\ b&b&b&b&a&c&c\\ a&c&c&a&b&a&b\\ \hline\end{array}\qquad TA'=\begin{array}{|c|c|c|c|c|c|c|}\hline 0&0&1&0&0&1&0\\ 1&1&6&1&6&4&1\\ 4&4&7&2&6&1&6\\ 1&5&8&3&7&2&7\\ 6&1&1&5&8&3&8\\ 7&4&4&7&2&5&6\\ 8&5&5&8&3&1&7\\ \hline\end{array}$$

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	**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

Zhu and Takaoka

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'

Nel caso pessimo la complessità è $\mathcal{O}(n'n + n'mm')$

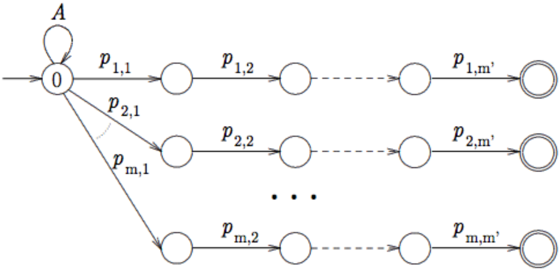
2D con Finite Automata

Definition

Sia π il dizionario ottenuto da PA, $\pi = \{p_i | p_i$ è la i-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 1)
- Lanciamo M' su TA' in modo che rilevi tutte le occorrenze di R

$M(\pi)$ 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.



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

Theorem

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

Dimostrazione.

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

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

Caso Esatto :: Costruzione TA'

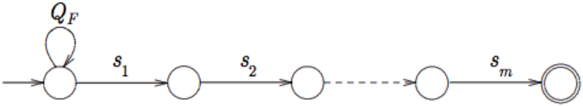
Grazie al teorema precedente possiamo costruire TA' usando la seguente formula:

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

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].

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 $E(R) = \{R \mid R \in F^m\}$, $F = \{s1, \ldots, s_{|\pi|}\}$, $|F| = |\pi|$.



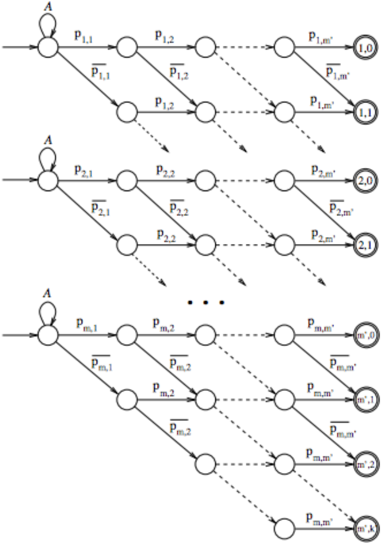
- Tutti gli automi presentati sono non deterministici
- Bisogna utilizzare una simulazione per non dover determinizzare gli automi
- Opera in tempo lineare: $\mathcal{O}(mm' + nn')$
  - $\mathcal{O}(mm')$ per $M(\pi)$
  - $\mathcal{O}(nn')$ per TA'
  - $\mathcal{O}(m)$ per R ed M'
  - $\mathcal{O}(nn')$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

Caso Approssimato

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/righa.
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



Caso Approssimato

Definition

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

Theorem

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

$$\forall i, j, s \ 1 \leq i \leq n, 1 \leq j \leq n', 1 \leq s \leq |\pi|$$

$$TA'[i,j,s] = \begin{cases} x & \text{se } \exists q \text{ attivo, } q \in F, q = (s,x) \\ \oslash & \text{altrimenti} \end{cases}$$

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 $(M_1, \ldots, M_k)$, 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

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

Caso Approssimato :: Costruzione di M'

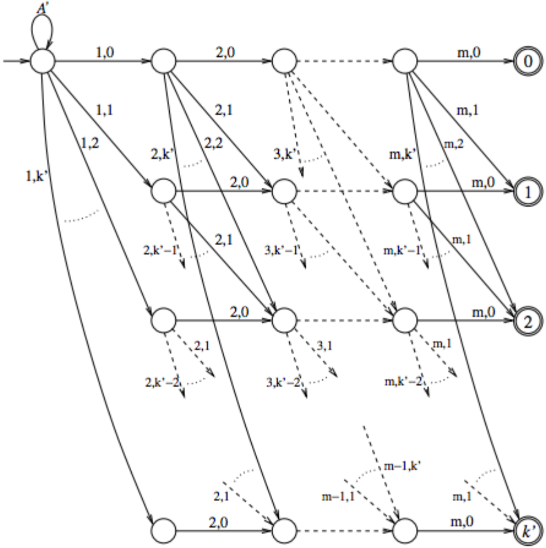
Ogni stato $q_{i,j}$ ha una transizione verso lo stato $q_{k,j+1}(\forall k. i \leq k \leq m')$ etichettata con (p, z) dove:

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

Per calcolare z si utilizza la seguente funzione:

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

Caso Approssimato :: Costruzione di M'



Simulazione

Gli automi costruiti in questi passaggi sono non deterministici, la simulazione prevede l'uso della *programmazione dinamica*.
Si introduce un array $E[0, \ldots, m-1]$ 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 di M' su TA'

```
for y = m' to n' do
    ∀q.E[q] inattivo
    for x = 1 to n do
        E[0] attivo, E[0] = 0
        foreach q attivo in E do
            err = TA'[x, y, R[q + 1]]
            if err = ⊘ then
                err = m'
            end if
            err = err + E[q]
            if err ≤ k then
                if q + 1 = m then
                    "Trovata Occorrenza"
                else
                    E[q + 1] attivo, E[q + 1] = err
                end if
            else
                E[q + 1] disattivato al prossimo step
            end if
        end for
    end for
end for
```

Conclusioni

- $M(\pi)$ 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 $\mathcal{O}(mm' \cdot nn' + m + mnn' - mmn') = \mathcal{O}(mm' \cdot nn')$

2D con PushDown Automata

Matrice Estesa

Definition

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

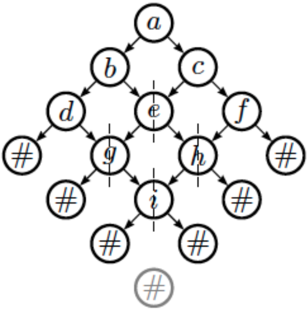
Sia $M \in A^{(n \times n')}$, definiamo la matrice estesa $M' \in A_\#^{(n+1 \times n'+1)}$ come M con una nuova riga e colonna riempite con il carattere $\#$

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

Da matrice a DAG

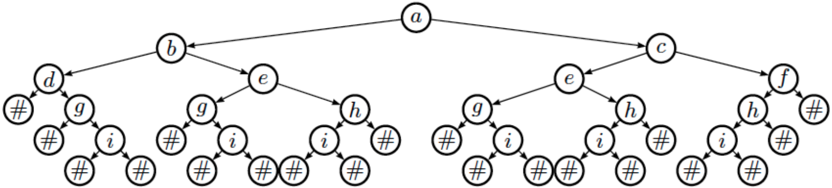
Dalla matrice M' estesa possiamo creare un *DAG* seguento le seguenti regole:

- 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 #



Da Matrice ad Alberi

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

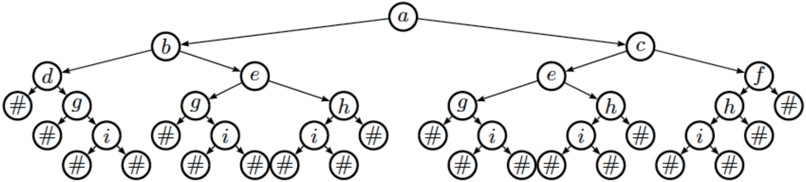
$$tree(M,x,y) = \begin{cases} (M[x,y], tree(M,x,y+1), tree(M,x+1,y)) & \text{Se } M[x,y] \neq \# \\ (M[x,y], null, null) & \text{Altrimenti} \end{cases}$$

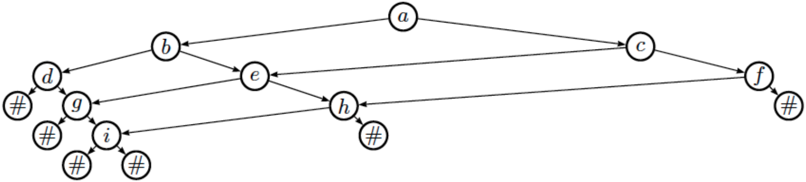
Definition

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.

Index-Tree :: Da Alberi a DAG

Notiamo che sottoalberi sono ripetuti più volte, possiamo allora eseguire una facile ottimizzazione.





Da Matrice ad Alberi

Theorem

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

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 $\mathcal{O}(2^{depth(t)})$

Dimostrazione.

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

Algoritmo Generico

- Estendiamo il testo 2D e il pattern 2D
- **Costruiamo l'index del testo 2D come un PDA**
- Costruiamo l'albero per il pattern 2D
- **Linearizziamo il pattern calcolando *pref* dell'albero**
- **Cerchiamo il *pref* del pattern con il PDA costruito al passo precedente**

Algorithm 4 Costruzione del PDA $\mathcal{M} = (Q, A_{\#}, G, \delta, q_{1,1}, S, \emptyset)$

```
δ = ∅, Q = ∅, G = {S}
G = G ∪ {S_{x+1,y}} ∀x, y 1 ≤ x < n ∧ 1 ≤ y < n′
crea_nodi()
crea_transizioni_base()
crea_transizioni_sharp()
crea_transizioni_bottom()
crea_transizioni_right()
crea_transizioni_suffix()
crea_transizioni_prefix()
```

Algoritmo per la costruzione del PDA

Algorithm 5 crea_nodi()

1: $Q = Q \cup \{Q_{x,y}\}\ \forall x, y\ 1 \leq x \leq n+1, 1 \leq y \leq n'+1$ e $y < n' \vee x < n$

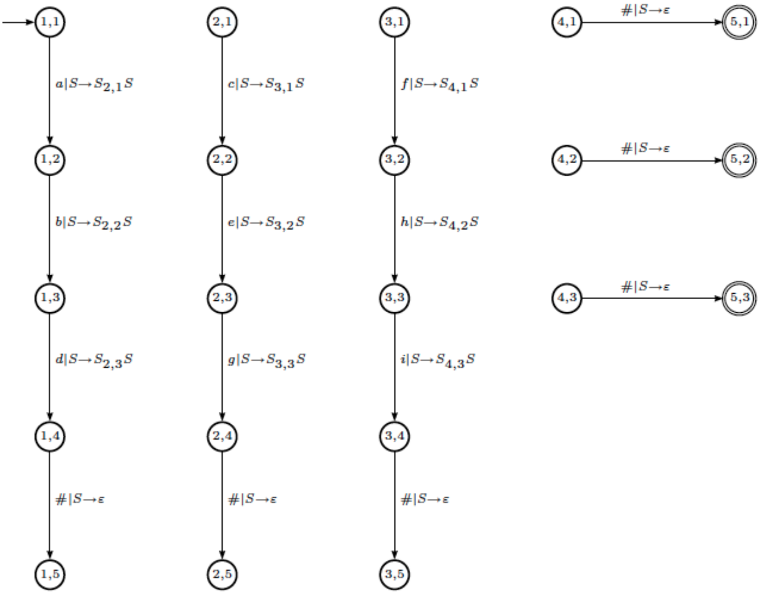
Algorithm 6 crea_transizioni_base()

1: $\delta(q_{x,y}, TA[x,y], S) = \delta(q_{x,y}, TA[x,y], S) \cup \{(q_{x,y+1}, S_{x+1,y}S)\}$
2: $\forall x, y\ 1 \leq x < n, 1 \leq y < n'$

Algorithm 7 crea_transizioni_sharp()

1: $\delta(Q_{n,y}, \#, S) = \delta(q_{n,y}, \#, S) \cup \{(q_{n+1,y}, \epsilon)\}\ \forall y 1 \leq y < n'$
2: $\delta(Q_{x,n'}, \#, S) = \delta(q_{x,n'}, \#, S) \cup \{(q_{x,n'+1}, \epsilon)\}\ \forall x 1 \leq y < n$

crea_nodi + crea_transizioni_base + crea_transizioni_sharp

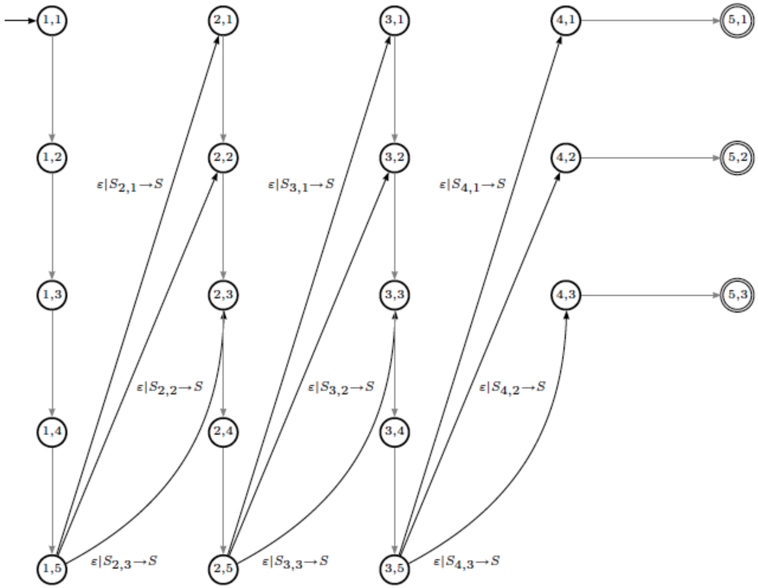


Algoritmo per la costruzione del PDA

Algorithm 8 crea_transizioni_bottom()

1: $\delta(q_{x,n'+1}, \epsilon, S_{x+1,y}) = \delta(q_{x,n'+1}, \epsilon, S_{x+1,y}) \cup \{(q_{x+1,y}, S)\}$
2: $\forall x, y \; 1 \leq x < n, 1 \leq y < n'$

crea_nodi + crea_transizioni_bottom

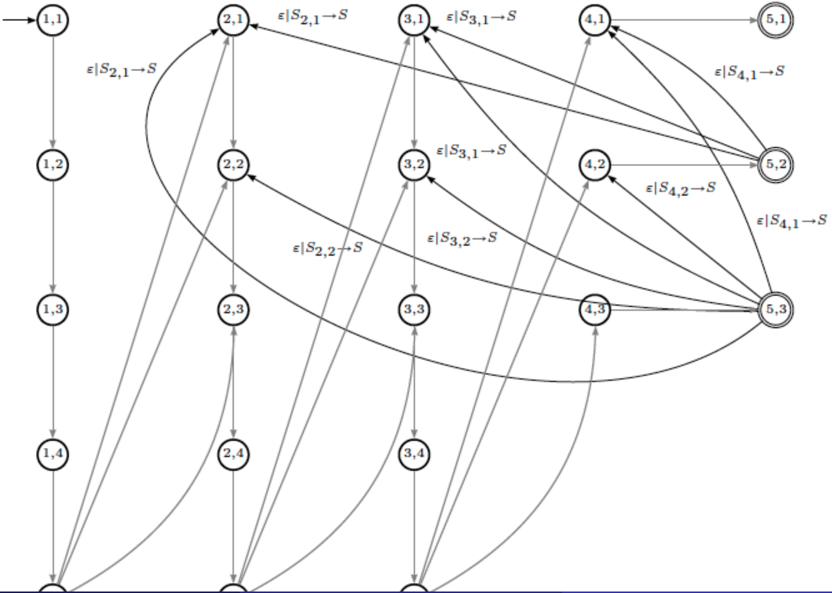


Algoritmo per la costruzione del PDA

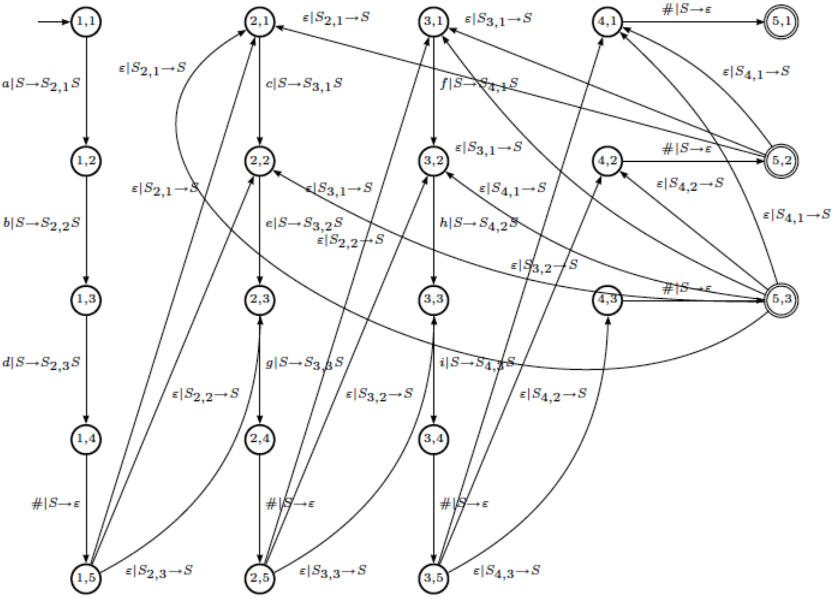
Algorithm 9 crea_transizioni_right()

1: $\delta(q_{n+1,y}, \epsilon, S_{x,y-1}) = \delta(q_{n+1,y}, \epsilon, S_{x,y-1}) \cup \{(q_{x,y-1}, S)\}$
2: $\forall x, y \ 2 \leq x \leq n, 2 \leq y < n'$

crea_nodi + crea_transizioni_right



nodi + basic + sharp + bottom + right



Algoritmo per la costruzione del PDA

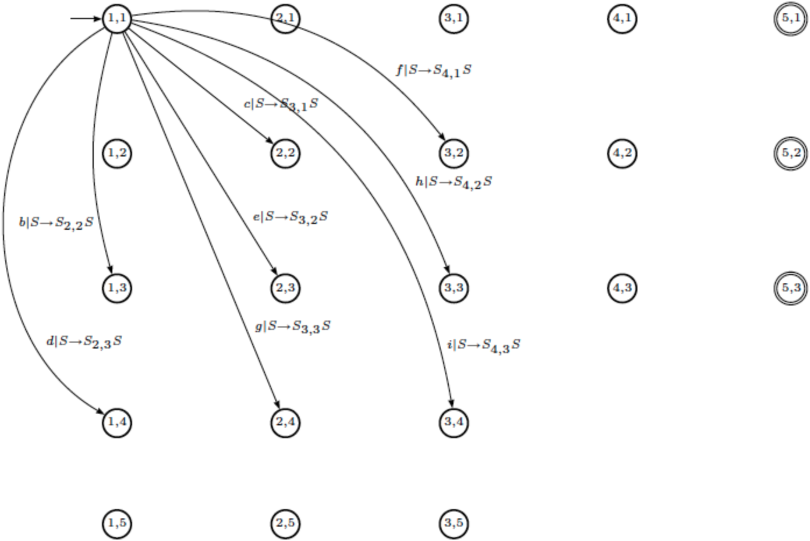
Algorithm 10 crea_transizioni_suffix()

$\delta(q_{1,1}, TA[x,y], S) = \delta(q_{1,1}, TA[x,y], S) \cup \{(q_{x,y+1}, S_{x+1,y}S)\}$
$\forall x, y \; 1 \leq x < n, 1 \leq y < n'$ e $x > 1 \vee y > 1$

Algorithm 11 crea_transizioni_prefix()

$\delta(q_{x,y}, \#, S) = \delta(q_{x,y}, \#, S) \cup \{(q_{n+1,y}, \epsilon)\}$
$\forall x, y \; 1 \leq x < n, 1 \leq y < n'$ e $x \neq 1 \vee y \neq 1$

nodi + suffix


1,1
2,1
3,1
4,1
5,1
1,2
2,2
3,2
4,2
5,2
1,3
2,3
3,3
4,3
5,3
1,4
2,4
3,4
1,5
2,5
3,5
$f|S \to S_{4,1}S$
$c|S \to S_{3,1}S$
$h|S \to S_{4,2}S$
$b|S \to S_{2,2}S$
$e|S \to S_{3,2}S$
$g|S \to S_{3,3}S$
$d|S \to S_{2,3}S$
$i|S \to S_{4,3}S$

nodi + prefix


#|S→ε
#|S→ε
1,1
2,1
3,1
4,1
5,1
#|S→ε
#|S→ε
#|S→ε
1,2
2,2
3,2
4,2
5,2
#|S→ε
#|S→ε
#|S→ε
1,3
2,3
3,3
4,3
5,3
1,4
2,4
3,4
1,5
2,5
3,5

Dimostrazione correttezza

Theorem

Sia TA un testo 2D esteso. Il PDA $\mathcal{M}$ costruito da TA accetta tutti i $pref(F)$ con F un fattore di TA. Quindi $\mathcal{M}$ accetta ogni $pref(TA[i..k, j..l]')$ con $1 \leq i \leq k \leq n, 1 \leq j \leq l \leq n'$

Dimostriamo per induzione sull'altezza dell'albero t costruito con $tree(F, 1, 1)$:

- F ha un solo elemento, $TA[i, j]$. t ha altezza 1 e $pref(t) = TA[i, j]\#\#$.
  Per mezzo delle transizioni suffix: $(q_{i,j+1}, S_{i+1,j}S) \in \delta(q_{1,1}, TA[i, j], S)$
  Quindi il PDA evolve nel seguente modo:
  $(q_{1,1}, TA[i, j]\#\#, S) \vdash^{+}_{\mathcal{M}} (q_{n+1,j}, \epsilon, \epsilon)$

Dimostrazione correttezza

- Assumiamo valga per $F_b = TA[i..k, j+1..l]$ e $F_r = TA[i+1..k, j..l]$. Sia $t_b = tree(F_b)$ e $t_r = tree(F_r)$ con $height(t_b) \leq m \wedge height(t_r) \leq m$, $m \geq 1$. Dimostriamo che vale anche per $t = TA[i,j]pref(t_b)pref(t_r)$, $height(t) \geq m+1$
  $(q_{1,1}, TA[i,j]pref(t_b)pref(t_r), S) \vdash_{\mathcal{M}}$ %Leggiamo il primo carattere
  $(q_{i,j+1}, pref(t_b)pref(t_r), S_{i+1,j}S) \vdash^{+}_{\mathcal{M}}$ %Per ip. Induttiva
  $(q_{n+1,j+1}, pref(t_r), S_{i+1,j}S) \vdash_{\mathcal{M}}$ %Transizioni Right
  $(q_{i+1,j}, pref(t_r), S) \vdash^{+}_{\mathcal{M}}$ %Per ip. Induttiva
  $(q_{n+1,j}, \epsilon, \epsilon)$.

Fine