Trasformata di Burrows-Wheeler di automi a stati finiti

Letizia D'Achille
Prof. Roberto Grossi

sulla base delle notazioni e della teoria dell'articolo
"On Indexing and Compressing finite automata" [1]

24 febbraio 2021

Abstract

La compressione di automi a stati finiti è il primo passo per la costruzione di strutture dati che supportino operazioni quali il riconoscimento di pattern nel linguaggio regolare definito da tali automi. L'obiettivo è quindi realizzare un'efficiente compressione dei dati, e a tal fine può essere utile effettuare una trasformazione preliminare all'automa in modo che l'oggetto finale consenta ancora di eseguire le operazioni suddette.
Nella relazione verrà quindi presentata una possibile estensione dalle stringhe agli automi a stati finiti della trasformata di Burrows-Wheeler, che sfrutta la possibilità di costruire ordinamenti parziali con buone proprietà sull'insieme degli stati.
Sarà quindi mostrato come per gli automi a stati finiti deterministici sia possibile individuare in maniera costruttiva un ordine parziale che ci conduca ad una compressione ottimale. Verrà infine esibito un algoritmo completo per effettuare la trasformata su tale categoria di automi, e per eseguirne l'inversione.

1 Definizioni di base

Un automa a stati finiti non deterministico (NFA) è una quintupla (Q, E, Σ, s, F) dove Q è l'insieme degli stati, $E \subseteq Q \times Q \times \Sigma$ è la funzione di transizione[1], Σ è l'alfabeto, $s \in Q$ è lo stato iniziale, e $F \subseteq Q$ è l'insieme degli stati finali.
Un automa a stati finiti deterministico (DFA) è un NFA tale che ogni stato ha al massimo un arco uscente etichettato con un dato carattere.

L'insieme di tutte le stringhe accettate dall'automa A è detto linguaggio accettato, e sarà denotato con $\mathcal{L}(A)$.
Sia $Pref(\mathcal{L}(A))$ l'insieme dei prefissi, ovvero l'insieme di tutte le stringhe in Σ^* [2] che possono essere lette su A seguendo un qualche percorso che parte dallo stato iniziale s.
Inoltre, per ogni $\alpha \in \Sigma^*$, si denoti con I_α l'insieme degli stati che possono essere raggiunti da s seguendo un percorso i cui archi, quando concatenati, restituiscono α.

Si fanno le seguenti assunzioni non restrittive (ogni automa può essere modificato in modo da soddisfare questi requisiti lasciando inalterato il linguaggio accettato):

- Ogni carattere etichetta almeno un arco.
- Tutti gli archi entranti nello stesso stato hanno la stessa etichetta[3].
- Tutti gli stati sono raggiungibili dallo stato iniziale.
- Lo stato iniziale non ha archi entranti.
- Ogni stato è finale, oppure permette di raggiungere uno stato finale.
- Su Σ è fissato un ordine totale.

Si denota con $\lambda(u)$ l'etichetta, univocamente determinata, di tutti gli archi entranti nel nodo u. Per lo stato iniziale s si scrive $\lambda(s) = \# \notin \Sigma$ e si impone $\# < c$ per ogni $c \in \Sigma$.

Dato un ordine parziale $\leq$ su un insieme V, gli elementi u e v si dicono $\leq$-confrontabili se vale che $u \leq v$ oppure che $v \leq u$. Si scrive $u \| v$ quando u e v non sono $\leq$-confrontabili.
Se $V' \subseteq V$, si dice che $U \subseteq V'$ è un $\leq_{V'}$-intervallo se per ogni $u, v, z \in V'$ tali che $u < v < z$ e $u, z \in U$ si ha $v \in U$. In particolare, si dice che $U \subseteq V$ è un $\leq$-intervallo se è un $\leq_V$-intervallo.

Un sottoinsieme $Z \subseteq V$ è una $\leq$-catena se $(Z, \leq)$ è un ordine totale. Una partizione $\{V_i\}_{i=1}^m$ di V è una $\leq$-decomposizione in catene se V_i è una $\leq$-catena per ogni $i = 1, \ldots, m$.
La $\leq$-ampiezza di V è la dimensione della sua anticatena più grande, cioè il più grande sottoinsieme $A = \{u_1, \ldots, u_p\} \subseteq V$ tale che $u_i \| u_j$ per ogni $1 \leq i < j \leq p$.

[1] La funzione di transizione si trova solitamente definita come $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ dove $\delta(u, m) = A$ con A l'insieme degli stati raggiungibili dallo stato corrente $u \in Q$ in base al carattere letto $m \in \Sigma$.
Definita come $E \subseteq Q \times Q \times \Sigma$ è l'insieme delle terne (u, v, m) con (u, v) coppia di stati e m carattere.
[2] La stella di Kleene * è un'operazione definita su un alfabeto Σ e indica l'insieme di tutte le stringhe su Σ, cioè denotando con Σ^k l'insieme delle stringhe di lunghezza k con caratteri estratti da Σ, allora $\Sigma^* = \bigcup_k \Sigma^k$.
[3] Questa ipotesi non è restrittiva in quanto può essere forzata rimpiazzando ogni stato con $|\Sigma|$ copie di se stesso senza modificare il linguaggio accettato.

2 Ordinamenti parziali sugli stati

Si inizia definendo un ordine co-lessicografico degli stati di un automa a stati finiti estendendo quello che si ha per le stringhe[4].

Definizione 1. Sia $A = (Q, E, \Sigma, s, F)$ un NFA. Un ordine co-lessicografico di A è un ordine parziale $\leq$ su Q tale che soddisfi gli assiomi che seguono:

- *(Assioma 1)* Per ogni $u, v \in Q$, se $\lambda(u) < \lambda(v)$, allora $u < v$;
- *(Assioma 2)* Per tutti gli archi $(u', u), (v', v) \in E$, se $\lambda(u) = \lambda(v)$ e $u < v$, allora $u' \leq v'$.

Osservazione 1. Gli stati con nessun arco entrante si trovano prima nell'ordinamento di tutti gli altri stati per l'ipotesi $\# < c \quad \forall c \in \Sigma$.

Osservazione 2. Sia $A = (Q, E, \Sigma, s, F)$ un NFA e sia $\leq$ un ordine co-lessicografico di A. Siano $u, v \in Q$ tali che $u \neq v$ e $\lambda(u) = \lambda(v)$. Allora, $u \| v$ se vale almeno una delle seguenti:

- Esistono degli archi $(u', u), (v', v) \in E$ tali che $u' \| v'$;
- Esistono degli archi $(u', u), (v', v), (u'', u), (v'', v) \in E$ tali che $u' < v'$ e $v'' < u''$.

Infatti se per esempio fosse $u < v$, allora l'Assioma 2 implicherebbe che nel caso 1 dovrebbe valere $u' \leq v'$ e nel caso 2 dovrebbe valere $u'' \leq v''$ (non è possibile a causa dell'asimmetria di $\leq$).

La figura sottostante mostra un NFA che farà da esempio nella prima parte della relazione.

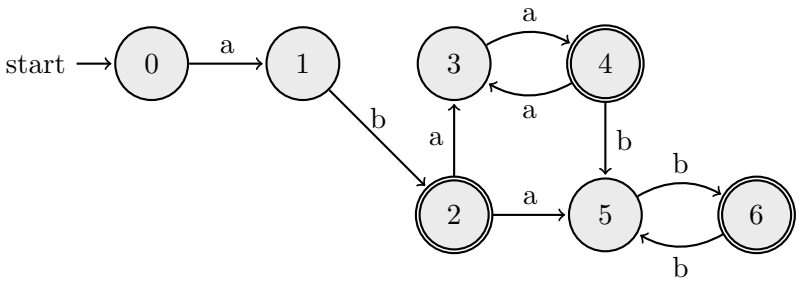


Figura 1: Esempio di NFA che riconosce il linguaggio regolare $\mathcal{L} = ab(aa)^*(bb)^*$.

Uno dei possibili ordini co-lessicografici ammessi dall'automa considerato è riportato nella Figura 2.

[4] È l'ordine ottenuto riflettendo le stringhe, applicando l'ordine lessicografico, e riflettendole nuovamente.

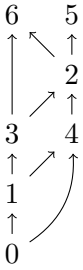


Figura 2: Diagramma di un possibile ordine co-lessicografico parziale degli stati dell'automa in Figura 1 ($u \to v \iff u \leq v$).

In questo esempio l'ampiezza dell'ordine parziale è 2, quindi l'ordine può essere partizionato in due $\leq$-catene per il seguente teorema [7]:

Teorema 1 (Teorema di Dilworth)**.** *In ogni insieme finito e parzialmente ordinato, l'anticatena più grande ha la stessa dimensione della più piccola $\leq$-decomposizione in catene.*

Dalla Figura 2 si rende evidente che una possibile $\leq$-decomposizione in catene (potrebbe non essere l'unica) è $\{\{0, 1, 3, 6\}, \{4, 2, 5\}\}$.

Si può facilmente osservare che ogni automa finito ammette un ordine co-lessicografico, poiché:

$$\leq \coloneqq \{(u, u) \in Q \times Q | u \in Q\} \cup \{(u, v) \in Q \times Q | \lambda(u) < \lambda(v)\}$$

soddisfa entrambi gli assiomi. Tuttavia in teoria quest'ordine potrebbe essere espanso con nuove coppie. Può essere infatti definito un criterio di massimalità che un ordine co-lessicografico dovrebbe avere al fine di essere utile per la compressione.

Definizione 2. Sia $A = (Q, E, \Sigma, s, F)$ un NFA.

1. Siano $\leq, \leq^*$ ordini co-lessicografici su Q. Si dice che $\leq^*$ è un raffinamento di $\leq$ se:
$$u \leq v \implies u \leq^* v \quad \forall u, v \in Q.$$
2. Un ordine co-lessicografico $\leq$ su Q è massimale se il suo unico raffinamento è $\leq$ stesso.

Osservazione 3. Ogni ordine co-lessicografico $\leq$ è raffinato da un ordine co-lessicografico massimale. In effetti, o $\leq$ è massimale o $\leq$ è raffinato da qualche altro ordine co-lessicografico, quindi si può costruire una catena non estendibile di ordini co-lessicografici diversi a due a due tali che ogni ordine co-lessicografico raffina il precedente. Dato che una tale catena deve essere finita, si ottiene un ordine co-lessicografico massimale che raffina $\leq$.

L'importanza della definizione di ordine co-lessicografico massimale si renderà evidente successivamente, quando si tratterà la sua unicità nei DFA. Quest'ultima sarà infatti una proprietà fondamentale per i risultati di complessità della trasformata di Burrows-Wheeler.

Per i risultati che seguono saranno considerati automi a stati finiti con un ordine co-lessicografico parziale sugli stati definito a priori.

Definizione 3. Un automa a stati finiti non deterministico (risp. deterministico) co-lessicografico (CNFA, risp. CDFA) è una sestupla $(Q, E, \Sigma, s, F, \leq)$ dove (Q, E, Σ, s, F) è un NFA (risp. DFA) e $\leq$ è un ordine co-lessicografico dell'automa.

Il lemma seguente mostra la relazione che intercorre tra l'ordinamento $\leq$ degli stati dell'automa definito tramite gli assiomi e l'ordine co-lessicografico delle stringhe che possono essere lette dallo stato iniziale.

Lemma 1. *Sia $A = (Q, E, \Sigma, s, F, \leq)$ un CNFA. Siano $u, v \in Q$ e $\alpha, \beta \in Pref(\mathcal{L}(A))$ tali che $u \in I_\alpha$, $v \in I_\beta$ e $\{u, v\} \not\subseteq I_\alpha \cap I_\beta$.*

- *Se $\alpha < \beta$, allora $u \| v$ oppure $u < v$;*
- *Se $u < v$, allora $\alpha < \beta$.*

Dimostrazione. Poiché $\{u, v\} \not\subseteq I_\alpha \cap I_\beta$, allora vale $u \in I_\alpha \setminus I_\beta$ e $v \in I_\beta \setminus I_\alpha$. Quindi $\alpha \neq \beta$ e $u \neq v$.
Per il primo punto si procede per induzione su $\min(|\alpha|, |\beta|)$.
Se $\min(|\alpha|, |\beta|) = 0$, allora $\alpha = \epsilon$, quindi $u = s$. Si conclude $u = s < v$ dall'Assioma 1.
Ora si assuma $\min(|\alpha|, |\beta|) \geq 1$. Questo implica $\alpha \neq \epsilon \neq \beta$ e $u \neq s \neq v$. Sia a l'ultima lettera di α e sia b l'ultima lettera di β; deve valere $a \leq b$, quindi $\lambda(u) < \lambda(v)$, il che implica $u < v$ per l'Assioma 1. Altrimenti, si può scrivere $\alpha = \alpha' e$ e $\beta = \beta' e$, con $e \in \Sigma$, $\alpha', \beta' \in \Sigma^*$ e $\alpha' < \beta'$. Siano $u', v' \in Q$ tali che $u' \in I_{\alpha'}$, $v' \in I_{\beta'}$, $u \in \delta(u', e)$, $v \in \delta(u', e)$. Allora $\{u', v'\} \not\subseteq I_{\alpha'} \cap I_{\beta'}$, altrimenti si avrebbe $\{u, v\} \subseteq I_\alpha \cap I_\beta$. Per ipotesi induttiva, si ha $u' \| v'$ oppure $u' < v'$. Quindi deve valere $u \| v$ o $u < v$, altrimenti varrebbe $v < u$, il che implica $v' \leq u'$ per l'Assioma 2.
Per il secondo punto, supponendo che $u < v$, si ha che $\alpha \neq \beta$. Se fosse $\beta < \alpha$, allora per la parte precedente varrebbe $v \| u$ oppure $v < u$, il che porta ad un assurdo. □

Una proprietà importante per la realizzazione di una compressione efficiente per gli automi è che l'insieme degli stati raggiunti da un percorso etichettato con una data stringa α formi un intervallo. Si mostra quindi che questa proprietà vale per l'ordine co-lessicografico parziale definito tramite gli assiomi.

Lemma 2. *Sia $A = (Q, E, \Sigma, s, F, \leq)$ un CNFA. Sia $\alpha \in \Sigma^*$, e sia U un $\leq$-intervallo di stati. Allora, l'insieme U' di tutti gli stati in Q che possono essere raggiunti da U percorrendo archi le cui etichette, quando concatenate, restituiscono α, è ancora un $\leq$-intervallo.*

Dimostrazione. Si proceda per induzione su $|\alpha|$. Se $|\alpha| = 0$, allora $\alpha = \epsilon$ e si conclude. Si supponga ora $|\alpha| \geq 1$. Si può scrivere $\alpha = \alpha' a$, con $\alpha' \in \Sigma^*, a \in \Sigma$. Siano $u, v, z \in Q$ tali che $u < v < z$ e $u, z \in U'$. Si deve provare che $v \in U'$. Per ipotesi induttiva, l'insieme U'' di tutti gli stati in Q che possono essere raggiunti da qualche stato in U percorrendo archi le cui etichette, quando concatenate, restituiscono α', è un $\leq$-intervallo. In particolare, esistono $u', z' \in U''$ tali che $(u', u), (z', z) \in E$. Poiché $u, z \in U'$, allora $\lambda(u) = a = \lambda(z)$. Questo implica che $\lambda(v) = a$, perché se per esempio fosse $\lambda(v) < a$, allora avremmo $\lambda(v) < \lambda(u)$, che per l'Assioma 1 dovrebbe implicare $v < u$, assurdo. Dato che $\lambda(v) = a$, allora v deve avere almeno un arco entrante $(v', v) \in E$. Per l'Assioma 2, si ha $u' \leq v' \leq z'$. Poiché $u', v' \in U''$ e U'' è un $\leq$-intervallo, allora $v' \in U''$, e quindi $v \in U'$. □

Corollario 1. *Sia $A = (Q, E, \Sigma, s, F, \leq)$ un CNFA. Sia $\alpha \in \Sigma^*$. Allora I_α è un $\leq$-intervallo.*

Dimostrazione. Nel Lemma 2 si prenda come $U = \{s\}$. Allora $U' = I_\alpha$. □

Il lemma seguente implica che un $\leq$-intervallo può essere codificato da al più p intervalli, ciascuno contenuto in una catena distinta, usando $O(p)$ bit.

Lemma 3. *Sia $(V, \leq)$ un ordine parziale, e sia U un $\leq$-intervallo. Sia $\{V_i\}_{i=1}^p$ una $\leq$-decomposizione di catene di V. Allora U è unione disgiunta di p insiemi $U_1, \ldots, U_p$ (anche vuoti), dove U_i è un $\leq_{V_i}$-intervallo, per $i = 1, \ldots, p$.*

Dimostrazione. Si definisca $U_i := U \cap V_i$. Allora, U è l'unione disgiunta di tutti gli U_i poiché $\{V_i\}_{i=1}^p$ è una partizione. Perciò si deve solamente provare che U_i è un $\leq_{V_i}$-intervallo. Siano $u, v, z \in V_i$ tali che $u < v < z$ e $u, z \in U_i$. In particolare $u, z \in U$, quindi $v \in U$ (poiché U è un $\leq$-intervallo) e si conclude $v \in U_i$. □

È possibile quindi riformulare il Lemma 2 come segue.

Lemma 4. *Sia $A = (Q, E, \Sigma, s, F, \leq)$ un CNFA e sia $\{Q_i\}_{i=1}^p$ una $\leq$-decomposizione in catene di Q. Sia $\alpha \in \Sigma^*$, e sia U un $\leq$-intervallo di stati. Allora, l'insieme U' di tutti gli stati in Q che possono essere raggiunti da U percorrendo archi le cui etichette, quando concatenate, restituiscono α, è unione disgiunta di p insiemi $U'_1, \ldots, U'_p$ (anche vuoti), dove U'_i è un $\leq_{Q_i}$-intervallo, per $i = 1, \ldots, p$.*

Il Lemma 4 è alla base del risultato di compressione della sezione che segue.

3 Trasformata di Burrows-Wheeler

La trasformata di Burrows-Wheeler (BWT) [6] è un algoritmo nato per agire sulle stringhe riordinandone i caratteri in accordo con l'ordine lessicografico delle rotazioni del testo.
È possibile generalizzare tale trasformata agli automi a stati finiti come segue.

Definizione 4 (BWT di un NFA)**.** Sia $A = (Q, E, \Sigma, s, F)$ un NFA. Sia $\leq$ un ordine co-lessicografico di A e sia $\mathcal{Q} \coloneqq \{Q_i\}_{i=1}^{p}$ una $\leq$-decomposizione in catene di Q, con $s \in Q_1$ senza perdere generalità. Sia $\pi(v)$, con $v \in Q$, l'unico intero tale che $v \in Q_{\pi(v)}$. Si consideri l'ordine $v_1, \ldots, v_n$ di Q tale che per ogni $1 \leq i < j \leq n$ vale che $\pi(v_i) < \pi(v_j) \vee (\pi(v_i) = \pi(v_j) \wedge v_i < v_j)$.
La BWT di $(A, \leq, \mathcal{Q})$ è la tripletta di sequenze BWT = (OUT, IN, FINAL), ognuna di lunghezza n, tale che, per ogni $i = 1, \ldots, n$:

- OUT$[i]$ è la lista delle coppie $(\pi(u), c)$, per ogni arco $(v_i, u, c) \in E$ uscente da v_i.
- IN$[i]$ è la lista degli interi $\pi(w)$, per ogni arco $(w, v_i, c) \in E$ entrante in v_i.
- FINAL$[i]$= 1 se $v_i \in F$, 0 altrimenti.

Per rendere più evidente come viene attuata la trasformata a partire da un automa a stati finiti con un ordine co-lessicografico e una decomposizione in catene fissati, si consideri l'esempio di NFA in Figura 1 con l'ordine co-lessicografico riportato in Figura 2 e la decomposizione in catene $Q_1 = \{0, 1, 3, 6\},\ Q_2 = \{4, 2, 5\}$.
Si prenda la sequenza di stati che si ottiene ordinando gli elementi delle due catene e poi concatenandole rispetto agli indici: 0,1,3,6,4,2,5. Si ottiene così l'ordine degli stati descritto nella definizione.
Si costruisca la matrice di adiacenza del grafo usando questo ordine. Le diverse catene sono evidenziate con diversi colori. Ora si partizioni la matrice di adiacenza in blocchi disegnando una linea orizzontale ogni volta che inizia una nuova catena e una linea verticale ogni volta che inizia una nuova catena oppure l'etichetta associata allo stato considerato cambia. In questo modo si formano dei blocchi, il risultato è mostrato nella figura sottostante con blocchi distinti evidenziati da sfumature di grigio distinte.

	#	a		b	a	b	
	0	1	3	6	4	2	5
0		a					
1						b	
3					a		
6							b
4			a				b
2			a				b
5				b			

Figura 3: Matrice di adiacenza del grafo mostrato in Figura 1, con blocchi evidenziati.

A questo punto per ogni arco si deve solo specificare la sua etichetta e le due catene agli estremi. La Figura 4 mostra questa costruzione. Le due liste OUT e IN sono facilmente ottenibili come segue: in verticale vengono raccolte le prime componenti di ogni tripletta (in verde) e in orizzontale le altre due componenti (in rosso).

	IN	[]	[1]	[2,2]	[2]	[1]	[1]	[1,2,2]
OUT		0	1	3	6	4	2	5
$[(1,a)]$	0		(1,1,a)					
$[(2,b)]$	1						(1,2,b)	
$[(2,a)]$	3					(1,2,a)		
$[(2,b)]$	6							(1,2,b)
$[(1,a),(2,b)]$	4			(2,1,a)				(2,2,b)
$[(1,a),(2,b)]$	2			(2,1,a)				(2,2,b)
$[(1,b)]$	5				(2,1,b)			

Figura 4: Visualizzazione in due dimensioni della BWT dell'automa.

La BWT definita sugli automi generalizza l'approccio descritto per le stringhe, per il quale vale $p = 1$. Infatti, in questo caso, $\pi(u)$ e $\pi(w)$ sono sempre uguali a 1 e le liste OUT$[i]$ e IN$[i]$ hanno lunghezza 1. Rimuovendo $\pi(u)$ e $\pi(w)$, rimane solo un'etichetta per stato e si ottiene che la sequenza OUT coincide con la classica BWT.

Sulla base della definizione appena data si ottiene un risultato di importanza fondamentale per la complessità in spazio, la cui dimostrazione mostra anche costruttivamente la reversibilità della trasformazione.

Definizione 5. Sia $A = (Q, E, \Sigma, s, F)$ un NFA. Si dice che A è p-ordinabile se esiste un ordine co-lessicografico $\leq$ di A tale che Q ammetta una $\leq$-decomposizione in catene $\{Q_i\}_{i=1}^{p}$.

Teorema 2. *La BWT di un NFA p-ordinabile $A = (Q, E, \Sigma, s, F)$ può essere memorizzata come una rappresentazione invertibile usando $|E|(\lceil \log \sigma \rceil + 2\lceil \log p \rceil + 2) + |Q|$ bit.*

Dimostrazione. Usando due vettori di bit di lunghezza $|E|$, si contrassegna con un bit l'ultimo elemento in ogni lista OUT$[i]$ e IN$[i]$ al fine di codificare le loro lunghezze. I componenti rimanenti prendono $\lceil \log \sigma \rceil + 2\lceil \log p \rceil$ bit per arco. Il vettore di bit FINAL occupa $|Q|$ bit.
Si mostra ora come invertire la rappresentazione. Innanzitutto, si assegni la numerazione $v_i = i$ agli stati. Usando OUT e IN, si possono ricostruire le funzioni π (numero della catena di ogni stato) e λ (etichette entranti di ogni stato): si scansioni OUT e si conti quanti archi entrano in ogni catena; poi, si combini questa informazione con i gradi entranti dei nodi (sequenza IN) per ricostruire la funzione π. Questo è possibile in quanto gli archi entranti in IN sono ordinati per catena crescente. In modo simile, una volta ricostruita π, si scansioni OUT e si rilevi il numero di archi entranti in ogni catena etichettati c, per ogni $c \in \Sigma$, e si combini questa informazione con il numero di archi entranti in ciascun nodo per ricostruire la funzione λ. Questo è possibile in quanto gli archi entranti in IN che entrano nella stessa catena sono ordinati per lettera crescente. Questo restituisce la partizione a blocchi della matrice di adiacenza mostrata

in Figura 3. A questo punto, si usi la seguente proprietà dei blocchi: coppie di archi con la stessa etichetta che lasciano una catena e entrano in un'altra preservano l'ordine co-lessicografico dei loro stati estremi. Si denoti con $\mathrm{IN}_{k,c}$ la sottosequenza in IN corrispondente agli stati u con $\pi(u) = k$ e $\lambda(u) = c$. Per $i = 1, \ldots, n$, si estraggano le coppie da OUT$[i]$. Per ogni tale coppia (k, c), si estragga l'elemento più a sinistra uguale a $\pi(v_i)$ da $\mathrm{IN}_{k,c}$ che non sia già stato considerato. Sia j il suo numero di colonna. Infine, si inserisca un arco etichettato c con estremi (i, j). □

Si conclude la sezione con un teorema che non verrà dimostrato in questa relazione, ma che mostra come la trasformata definita sugli automi sia utile ai fini di una compressione ottimale che riesca comunque a supportare le operazioni di ricerca di pattern.

Teorema 3. *La BWT di un NFA p-ordinabile $A = (Q, E, \Sigma, s, F)$ può essere codificata con una struttura dati di $|E|(\lceil \log \sigma \rceil + 2\lceil \log p \rceil + 2) \cdot (1 + o(1)) + 2|Q| \cdot (1 + o(1))$ bit che, data una stringa $\alpha \in \Sigma^m$, supporta le seguenti operazioni in $O(m \cdot p^2 \cdot \log(p \cdot \sigma))$ tempo:*

(i) Contare il numero di stati raggiunti tramite un percorso etichettato α.

(ii) Ritornare identificativi unici per gli stati raggiunti tramite un percorso etichettato α.

(iii) Decidere se $\alpha \in \mathcal{L}(A)$.

Dato un qualsiasi I_α e $c \in \Sigma$, la struttura può anche computare $I_{\alpha \cdot c}$ in $O(p^2 \cdot \log(p \cdot \sigma))$ tempo.

4 Unicità dell'ordine co-lessicografico massimale nei DFA

In generale, un NFA ammette diversi ordini co-lessicografici massimali: si consideri, come semplice esempio, una sorgente s connessa con la stessa etichetta a n stati non adiacenti a due a due. Si vuole invece provare che i DFA ammettono un unico ordine co-lessicografico massimale.
Inizialmente si presentano delle proprietà che permettono di identificare un ordine co-lessicografico in un automa a stati finiti.

Lemma 5. *Sia $A = (Q, E, \Sigma, s, F)$ un NFA, e sia $\leq$ una relazione riflessiva e antisimmetrica su V che soddisfa le seguenti proprietà:*

- *Per ogni $u, v \in Q$, se $\lambda(u) < \lambda(v)$, allora $u < v$;*
- *Per tutti gli archi (u', u), $(v', v) \in E$ tali che $\lambda(u) = \lambda(v)$, se $u < v$, allora $u' \leq v'$.*

Sia $\leq^*$ la chiusura transitiva di $\leq$, e si assuma che $\leq^*$ sia antisimmetrica. Allora, $\leq^*$ è un ordine co-lessicografico di A.

Dimostrazione. Innanzitutto, $\leq^*$ è riflessiva in quanto $\leq$ è riflessiva, quindi $\leq^*$ è un ordine parziale. Inoltre, $\leq^*$ soddisfa l'Assioma 1, poiché se $u, v \in Q$ sono tali che $\lambda(u) < \lambda(v)$, allora $u < v$ e quindi $u \leq^* v$. Quindi si deve solo mostrare che è soddisfatto l'Assioma 2. Si considerino due archi $(u', u), (v', v) \in E$ tali che $\lambda(u) = \lambda(v)$ e $u \leq^* v$; si deve provare che $u' \leq^* v'$. Dato che $\leq^*$ è la chiusura transitiva di $\leq$, esistono stati $z_1, \ldots, z_r$ $(r \geq 0)$ tali che $u < z_1, z_1 < z_2, \ldots, z_r < v$, e in particolare $u <^* z_1 <^* z_2 <^* \cdots <^* z_r <^* v$. Poiché $\lambda(u) = \lambda(v)$, allora $\lambda(u) = \lambda(z_1) = \cdots = \lambda(z_r) = \lambda(v)$. Infatti, se per qualche j fosse per esempio $\lambda(z_j) > \lambda(u) = \lambda(v)$, allora dovrebbe valere $v < z_j$ e quindi $v <^* z_j$, il che contraddice $z_j <^* v$. In particolare, dato che u e v hanno archi entranti, allora anche tutti gli z_i hanno archi entranti $(z_i', z_i) \in E$, per $i = 1, \ldots, k$. La seconda assunzione implica che $u' \leq z_1', z_1' \leq z_2', \ldots, z_k' \leq v'$, quindi $u' <^* z_1' <^* z_2' <^* \cdots <^* z_r' <^* v'$ e si conclude $u' \leq^* z'$. □

Determinare un ordine co-lessicografico di minima ampiezza può essere più semplice se ogni ordine co-lessicografico parziale deve essere la restrizione di qualche ordine totale sull'insieme degli stati. Viene quindi introdotta la definizione seguente.

Definizione 6. Sia $A = (Q, E, \Sigma, s, F)$ un NFA. Si dice che un ordine totale $\leq_\#$ su Q è un ordine fondamentale di A se per ogni ordine co-lessicografico $\leq$ di A:

$$u < v \implies u \leq_\# v \quad \forall u, v \in Q.$$

In generale, un NFA non ammette un ordine fondamentale: si consideri di nuovo l'esempio di una sorgente s connessa con la stessa etichetta a n stati non adiacenti a due a due. Per i DFA si ha tuttavia il risultato che segue.

Lemma 6. *Sia $A = (Q, E, \Sigma, s, F)$ un DFA. Allora A ammette un ordine fondamentale $\leq_\#$. In particolare, per ogni $u, v \in Q$, se $\lambda(u) < \lambda(v)$, allora $u \leq_\# v$.*

Dimostrazione. Sia $Q = \{u_1, \ldots, u_n\}$. Per ogni $i = 1, \ldots, n$ sia $\alpha_i \in Pref(\mathcal{L}(A))$ tale che $q_i \in I_{\alpha_i}$. Intuitivamente, gli α_i possono essere determinati costruendo un albero di copertura orientato di A con radice s. Poiché A è un DFA, allora $\alpha_1, \ldots, \alpha_n$ sono distinti a due a due. Senza perdere generalità, si assume $\alpha_1 < \alpha_2 < \cdots < \alpha_n$. Il Lemma 1 implica che, per ogni ordine co-lessicografico di A, se u_i e u_j, con $i < j$, sono $\leq$-confrontabili, allora deve essere $u_i < u_j$. Di conseguenza, se $\leq_\#$ è l'ordine totale su Q tale che $u_1 \leq_\# u_2 \leq_\# \cdots \leq_\# u_n$, allora $\leq_\#$ è un ordine fondamentale su A. Dato che ogni automa ammette un ordine co-lessicografico, l'enunciato finale segue dall'Assioma 1. □

Osservazione 4. In generale, $\leq_\#$ non è unico, si veda la Figura 5. Nell'esempio si ha infatti che $1 \in I_a$ e $2 \in I_{aa}$, quindi si può porre $1 <_\# 2$. D'altra parte, $1 \in I_{aaa}$ e $2 \in I_{aa}$, quindi si può anche porre $2 <_\# 1$.
Tuttavia, si ha unicità nel senso che segue: se esiste un ordine co-lessicografico $\leq$ per il quale u e v sono $\leq$-confrontabili, allora l'ordine reciproco di u e v in $\leq_\#$ è determinato in maniera univoca. Questo è coerente con la Figura 5: per ogni ordine co-lessicografico $\leq$, gli stati q_1 e q_2 non possono essere $\leq$-confrontabili per l'Assioma 2.

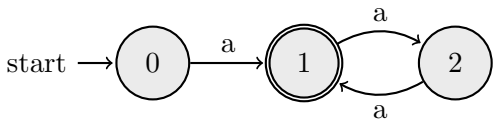


Figura 5: Esempio di DFA che ammette più ordini fondamentali distinti.

Nei DFA si può però comunque dimostrare l'unicità dell'ordine co-lessicografico massimale.

Teorema 4. *Sia $A = (Q, E, \Sigma, s, F)$ un DFA. Allora A ammette un unico ordine co-lessicografico massimale.*

Dimostrazione. Per il Lemma 6, A ammette un ordine fondamentale $\leq_\#$. Si considerino due ordini co-lessicografici massimali $\leq_1$, $\leq_2$ di A; si vuole provare che $\leq_1$ e $\leq_2$ sono uguali. Sia $\leq_3$ l'unione di $\leq_1$ e $\leq_2$ (ovvero, $u \leq_3 v$ se e solo se $(u \leq_1 v) \vee (u \leq_2 v)$) e sia $\leq_4$ la chiusura transitiva di $\leq_3$. Allora $\leq_3$ è riflessivo (perché per esempio $\leq_1$ è riflessivo); inoltre, $\leq_3$ e $\leq_4$ sono antisimmetrici (perché sono restrizioni dell'ordine fondamentale $\leq_\#$). Si noti che $\leq_3$ soddisfa le ipotesi del Lemma 5, perché:

- Se $\lambda(u) < \lambda(v)$, allora per esempio $u \leq_1 v$ (dato che che $\leq_1$ è un ordine co-lessicografico) e quindi $u \leq_3 v$;
- Se $(u', u), (v', v) \in E$ sono tali che $\lambda(u) = \lambda(v)$ e $u \leq_3 v$, allora si ha $(u \leq_1 v) \vee (u \leq_2 v)$, il che implica $(u' \leq_1 v') \vee (u' \leq_2 v')$ (dato che $\leq_1$ e $\leq_2$ sono ordini co-lessicografici) e quindi $u' \leq_3 v'$.

Per il Lemma 5 si può quindi concludere che $\leq_4$ è un ordine co-lessicografico. Comunque, $\leq_4$ è un raffinamento sia di $\leq_1$, sia di $\leq_2$, che sono massimali, quindi $\leq_4$ deve essere uguale sia a $\leq_1$, sia a $\leq_2$. Segue la tesi. □

Infine si presenta una caratterizzazione non dimostrata dell'ordine co-lessicografico massimale nei DFA, che sarà usata nel Teorema 7 per dimostrare la complessità polinomiale del problema della costruzione dell'ordine co-lessicografico massimale di un DFA.

Teorema 5. *Sia $A = (Q, E, \Sigma, s, F)$ un DFA, e sia $\leq_\#$ un ordine fondamentale di A. Sia $\leq$ la restrizione riflessiva di $\leq_\#$ tale che per tutti gli stati $u, v \in Q$ con $u \leq_\# v$ vale $u \| v$ se e solo se per qualche $r \geq 1$ esistono stati $u_0, u_1, \ldots, u_r$ e $v_0, v_1, \ldots, v_r$ con le proprietà seguenti:*

- *$u_r = u$ e $v_r = v$;*
- *$(u_k, u_{k+1}), (v_k, v_{k+1}) \in E$ per $k = 0, 1, \ldots, r-1$;*
- *$\lambda(u_k) = \lambda(v_k)$ per $k = 1, 2, \ldots, r$ (in particolare, $\lambda(u) = \lambda(v)$);*
- *$u_k <_\# v_k$ per $k = 1, 2, \ldots, r$;*
- *$v_0 <_\# u_0$.*

Allora, $\leq$ è un ordine co-lessicografico massimale di A. Inoltre, $\leq$ è l'unico ordine co-lessicografico massimale di A.

5 Risultati di complessità

Dai risultati ottenuti nelle sezioni precedenti si evidenzia come sia necessario minimizzare la quantità p, da cui dipende la complessità in spazio. A tal fine si introduce un'utile definizione.

Definizione 7. L'ampiezza co-lessicografica $\overline{p}$ di A è il minimo intero p per il quale A è p-ordinabile.

Il problema è quindi trovare (o limitare) $\overline{p}$ e determinare una corrispondente decomposizione in catene degli stati dell'automa.
Il primo risultato utile a tal fine, che non verrà dimostrato nella relazione, è il seguente.

Teorema 6. *Si definisca la versione decisionale del problema dell'ordinamento come segue: dato un NFA A e un intero p, determinare se A è p-ordinabile. Allora il problema dell'ordinamento è NP-hard.*

Nel lemma che segue risulta evidente come la difficoltà del problema dell'ordinamento risieda nel di trovare l'ordine co-lessicografico di ampiezza minore, in quanto, fissato un ordine co-lessicografico $\leq$, si può trovare la minima $\leq$-decomposizione in catene in tempo polinomiale.

Lemma 7. *Sia $(V, \leq)$ un ordine parziale, con $|V| = n$. La più piccola $\leq$-decomposizione in catene di V può essere trovata in $O(n^{5/2})$ tempo.*

Dimostrazione. Ford e Fulkerson [4] hanno fornito una riduzione dal problema della minima decomposizione in catene al problema dell'abbinamento massimo[5] sul grafo bipartito (V', V'', E), dove $V' = V'' = V$ e $(v', v'') \in E \subseteq V' \times V''$ se e solo se $v' \leq v''$. Completando l'abbinamento con le coppie $(v, v) \in V' \times V''$ per ogni $v \in V$, le componenti connesse risultanti sono le catene suddette. La complessità della procedura quindi si riduce a quella usata per trovare un abbinamento massimo, che può essere risolta con l'algoritmo di Hopcroft e Karp [3] in $O(|E| \cdot \sqrt{n}) \in O(n^{5/2})$ tempo. □

Tramite i risultati dimostrati precedentemente, è possibile trarre delle conclusioni più apprezzabili per gli automi a stati finiti deterministici.

Osservazione 5. Si supponga $\leq^*$ sia un raffinamento di $\leq$, e siano w^* e w le loro rispettive ampiezze. Allora deve valere $w^* \leq w$ poiché ogni $\leq$-decomposizione in catene è anche una $\leq^*$-decomposizione in catene. Questo implica che se $\overline{p}$ è l'ampiezza co-lessicografica di $A = (Q, E, \Sigma, s, F)$, allora esiste un massimo ordine co-lessicografico su Q la cui ampiezza è $\overline{p}$.

[5] Dato un grafo, un abbinamento è un insieme di archi tale che presi due elementi distinti non hanno vertici in comune. Un abbinamento massimo è un abbinamento che contiene il massimo numero possibile di archi.

Da questa osservazione e dall'unicità dell'ordine co-lessicografico massimale dimostrata nel Teorema 4, si può affermare che tale ordine ha ampiezza uguale all'ampiezza co-lessicografica $\overline{p}$ dell'automa.
L'ordine co-lessicografico massimale può essere trovato in tempo polinomiale, come è mostrato nel teorema che segue. Dunque il problema dell'ordinamento nei DFA è polinomiale nel numero dei nodi.

Teorema 7. *Sia $A = (Q, E, \Sigma, s, F)$ un DFA. Si può trovare l'unico ordine co-lessicografico massimale di A in $O(|E|^2)$ tempo.*

Dimostrazione. Per il Lemma 6, A ammette un ordine fondamentale. Seguendo [2, Teorema 4], in $O(|E|)$ tempo si può costruire un ordine fondamentale $\leq_\#$ tramite l'ordinamento dei prefissi di un albero di copertura orientato di A con sorgente s. Si consideri il grafo $G = (V, F)$, dove $V = \{(u, v) | (\lambda(u) = \lambda(v)) \wedge (u \leq_\# v)\}$ e $F = \{((u', v'), (u, v)) \in V \times V | (u', u), (v', v) \in E\}$. Inizialmente, per ogni coppia di archi $(u', u), (v', v) \in E$ tali che $\lambda(u) = \lambda(v)$, $u \leq_\# v$ e $v' \leq_\# u'$, si contrassegna il nodo (u, v) di G. Questo processo richiede $O(|E|^2)$ tempo. Dunque si contrassegnano tutti i nodi raggiungibili su G dai nodi contrassegnati. Questo può essere fatto con una semplice visita DFS di G, inizializzando la pila con tutti i nodi contrassegnati. Anche questo processo richiede $O(|E|^2)$ tempo. Per il Teorema 5, se si rimuove da $\leq_\#$ l'insieme di tutte le coppie contrassegnate di V si ottiene l'ordine co-lessicografico massimale di A. □

Corollario 2. *Sia $A = (Q, E, \Sigma, s, F)$ un DFA. Si può trovare l'unico ordine co-lessicografico massimale di A e la corrispondente più piccola decomposizione in catene $\{Q_i\}_{i=1}^{\overline{p}}$ dove $\overline{p}$ è l'ampiezza co-lessicografica di A, in $O(|E|^2 + |Q|^{5/2})$ tempo.*

Nei DFA è quindi possibile minimizzare la quantità p, e quindi effettuare una compressione ottimale, in un tempo polinomiale.

6 Algoritmo

Nella presente sezione si esibisce l'algoritmo sviluppato sulla base dei risultati teorici esposti.
La procedura si divide in tre parti principali:

1. Dato un DFA, trovare l'unico ordine co-lessicografico massimale dei suoi stati;
2. Dati un NFA e un ordine co-lessicografico parziale dei suoi stati, calcolarne la BWT;
3. Data una BWT, ricostruire l'NFA corrispondente.

6.1 Parte 1 - Ordine co-lessicografico massimale

Si prenda come esempio il DFA mostrato nella Figura 6.

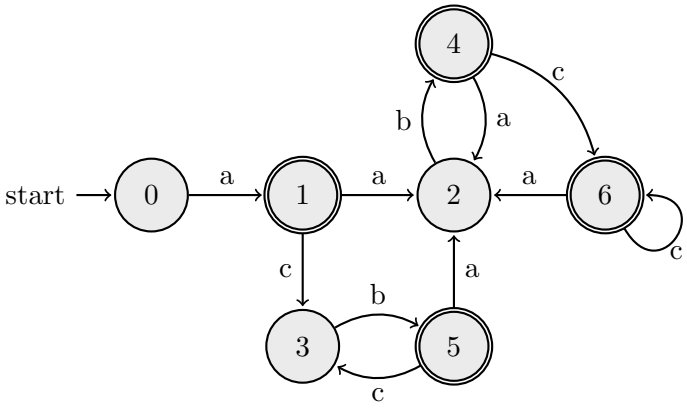
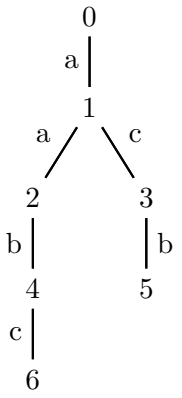


Figura 6: Esempio di DFA che riconosce il linguaggio regolare $\mathcal{L} = a(cb)^*(ab(c)^*)^*$.

La procedura segue quanto descritto nel Teorema 7.

Inizialmente si costruisce un ordine fondamentale del DFA ordinando i prefissi di un albero di copertura orientato dell'automa con sorgente s (si veda Appendice A).
Si è implementata la costruzione dell'albero di copertura con una visita DFS, sono stati poi determinati i prefissi tramite una pre-visita sull'albero, tale lista di prefissi è stata quindi ordinata. L'ordinamento è stato effettuato usando l'algoritmo combinato merge-insertion sort [5].



$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 5 \longrightarrow 3 \longrightarrow 6$$

Figura 7: Sinistra: Albero di copertura dell'automa in Figura 6. Destra: Ordine fondamentale ottenuto eseguendo l'algoritmo sull'automa in Figura 6.

Si ricava quindi l'unico ordine co-lessicografico massimale come nella dimostrazione del Teorema 7, costruendo il grafo G, ed effettuando due successive contrassegnazioni dei nodi. La prima si effettua esaminando tutti gli stati di G, la seconda tramite una visita DFS inizializzando la pila con i nodi contrassegnati al primo passo.

Si rimuovono infine dall'ordine fondamentale tutte le coppie appena contrassegnate in G, e per il Teorema 5 si ottiene l'ordine co-lessicografico massimale.

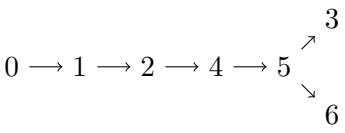


Figura 8: Ordine massimale ottenuto eseguendo l'algoritmo sull'automa in Figura 6.

6.2 Parte 2 - BWT

Si continua ad usare l'esempio di DFA in Figura 6, ma è comunque possibile eseguire la procedura su un NFA generico a partire da un ordine co-lessicografico parziale dei suoi stati.

Per calcolare la BWT dell'automa si deve prima costruire la più piccola decomposizione in catene rispetto all'ordine co-lessicografico dato in input. A tal fine si segue la procedura descritta nel Lemma 7.
Si costruisce quindi il grafo bipartito, e su di esso si risolve il problema dell'abbinamento massimo con l'algoritmo di Hopcroft e Karp [3] (si veda Appendice B).

Algoritmo di Hopcroft e Karp Dato in input un grafo bipartito $G(V \cup V', E)$ si vuole dare in output l'abbinamento massimo $M \subseteq E$.
L'algoritmo è il seguente:

1. $M = \emptyset$;
2. Si ripetano i seguenti passi:
   2.1. Si trovi $P = \{P_1, P_2, \ldots, P_k\}$ insieme massimale dei cammini aumentanti[6] di lunghezza minima tali che siano a vertici disgiunti;
   2.2. Si aggiungano ad ogni passo i cammini trovati all'abbinamento corrente: $M = M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$;
3. Si termini quando non ci sono più cammini con tali proprietà, ovvero $P = \emptyset$.

Si completa quindi l'abbinamento seguendo ancora il Lemma 7, e si individuano le componenti connesse. Quest'ultimo passo è stato realizzato scegliendo una delle due parti del grafo bipartito ed effettuando una visita DFS inizializzando lo stack con tutti i nodi che si trovano nella parte scelta e che hanno un solo arco entrante. Questa procedura è conclusiva per la seguente osservazione.

[6]Un cammino aumentante è un cammino tale che i suoi estremi non siano incidenti con nessun arco nell'abbinamento M corrente, e i suoi archi siano alternatamente in M e in M^C

Osservazione 6. Ricordando che i nodi del grafo bipartito sono divisi in due parti V e V', vale che ogni componente connessa del grafo ottenuto deve contenere un nodo in V con un solo arco entrante e un nodo in V' con la stessa proprietà. Infatti tutti gli archi hanno uno dei due estremi in V e l'altro in V'. Per come è stato costruito il grafo si può fissare una delle due parti, ad esempio V, in modo che per ogni arco (u, v) del grafo con $u \in V$, $v \in V'$ e $u \neq v$ valga che $u < v$ rispetto all'ordine co-lessicografico fissato. Se per una componente connessa non esistesse un nodo in V con un solo arco entrante allora si potrebbe costruire una catena di disuguaglianze strette $u < u_1 < \cdots < u_n < u$ per certi $u, u_1, \ldots, u_n$, ovvero $u < u$, assurdo. Lo stesso vale per V'.

Le componenti connesse che sono state individuate sono le catene cercate.
Nell'esempio in Figura 6 la decomposizione in catene che si ottiene eseguendo l'algoritmo è la seguente: $Q_1 = \{6\}$, $Q_2 = \{0, 1, 2, 4, 5, 3\}$.

A questo punto si costruisce la matrice di adiacenza dell'automa con righe e colonne ordinate secondo le catene, e con le singole catene ordinate rispetto all'ordine parziale. Infine si generano i vettori OUT, IN e FINAL della BWT, esaminando la matrice di adiacenza (si veda Appendice C).

	IN	[1,2]	[]	[2]	[1,2,2,2]	[2]	[2]	[2,2]
OUT		6	0	1	2	4	5	3
$[(1,c),(2,a)]$	6	(1,1,c)			(1,2,a)			
$[(2,a)]$	0			(2,2,a)				
$[(2,a),(2,c)]$	1				(2,2,a)			(2,2,c)
$[(2,b)]$	2					(2,2,b)		
$[(1,c),(2,a)]$	4	(2,1,c)			(2,2,a)			
$[(2,a),(2,c)]$	5				(2,2,a)			(2,2,c)
$[(2,b)]$	3						(2,2,b)	

Figura 9: Visualizzazione in due dimensioni della BWT dell'automa in Figura 6

$$
\begin{aligned}
OUT = [\ & [(1,c),(2,a)], \\
& [(2,a)], \\
& [(2,a),(2,c)], \\
& [(2,b)], \\
& [(1,c),(2,a)], \\
& [(2,a),(2,c)], \\
& [(2,b)]\ ]
\end{aligned}
\qquad
\begin{aligned}
IN = [\ & [1,2], \\
& [\,], \\
& [2], \\
& [1,2,2,2], \\
& [2], \\
& [2], \\
& [2,2]\ ]
\end{aligned}
\qquad
\begin{aligned}
FINAL = [\ & 1, \\
& 0, \\
& 1, \\
& 0, \\
& 1, \\
& 1, \\
& 0\ ]
\end{aligned}
$$

Figura 10: BWT dell'automa in Figura 6

6.3 Parte 3 - Inversione della BWT

Si noti che questo passo dell'algoritmo può essere applicato indifferentemente su trasformate di DFA o NFA. Sarà usata come esempio la trasformata calcolata in Figura 10. È possibile verificare la correttezza dell'algoritmo controllando se l'automa che sarà ottenuto sia uguale all'automa originale in Figura 6.

Per questo passo si procede come nella dimostrazione del Teorema 2 (si veda Appendice D).

In primo luogo si ricostruisce la decomposizione in catene. A tal fine si scansiona il vettore OUT e si contano quanti archi entrano in ogni catena. Poi, si combina questa informazione con il numero di archi entranti nei nodi, che può essere ricavato dalla lista IN. Si può quindi ottenere la decomposizione in quanto gli archi entranti in ciascun nodo sono ordinati per catena crescente nella lista IN.

In secondo luogo si scansiona il vettore OUT e si conta il numero di archi per ciascuna etichetta che entrano in ogni catena. Si combina questa informazione con il numero di archi entranti nei nodi, già calcolato al passo precedente. Si ricava quindi l'etichetta (univoca) entrante in ciascuno stato in quanto gli archi entranti nella lista IN che si immettono nella stessa catena sono ordinati per etichetta crescente.

A questo punto si devono solamente combinare queste due informazioni come descritto nel Teorema 2 per ottenere l'automa in partenza.
Gli stati finali sono poi facilmente identificati dalla lista FINAL, e per individuare la sorgente è sufficiente controllare quale degli stati ha come etichetta #.

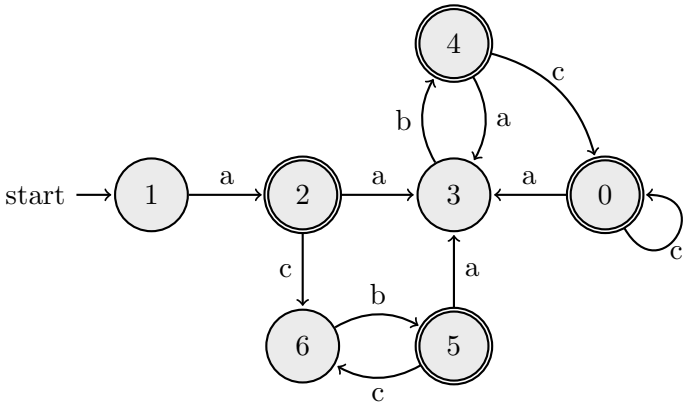


Figura 11: Automa ottenuto applicando l'algoritmo alla BWT in Figura 10

Si osserva che l'automa ottenuto nell'esempio è lo stesso della Figura 6 con un riordinamento dei nodi, a conferma dei risultati teorici riportati.

A Algoritmo - Calcolo ordine fondamentale

```
//Calcolo albero di copertura orientato

void spanning_tree(vector<edge>* automa, vector<edge>* tree){
    bool app[n];

    for (int i=0; i<n; i++){
        app[i] = false;
    }

    stack<int> st;
    st.push(s);
    app[s] = true;

    do{
        int u = st.top();
        st.pop();
        for (edge e: automa[u]){
            int v = e.v;
            if (!app[v]){
                app[v] = true;
                edge en;
                en.v = v;
                en.label = e.label;
                tree[u].push_back(en);
                st.push(v);
            }
        }
    }while(!st.empty());
}

//Calcolo dei prefissi e quindi ordinamento tramite merge-insertion sort

void previsit(int node, vector<edge>* tree, string* labels, string* prefixes){

    if (!tree[node].empty()){
        for (edge e: tree[node]){
            prefixes[e.v] = labels[e.v] + prefixes[node];
            previsit(e.v, tree, labels, prefixes);
        }
    }

    return;
}

void prefix_sorting(vector<edge>* tree, string* labels, int* underlying_order){
    string prefixes[n];
    prefixes[s]="";

    previsit(s, tree, labels, prefixes);

    merge_insertion_sort_prefix(prefixes, 0, n - 1, underlying_order);
}
```

```
53 //Funzione principale
54
55 void find_underlying_order(vector<edge>* automa, int* underlying_order, string*
       labels){
56     vector<edge> tree[n];
57
58     spanning_tree(automa, tree);
59
60     for(int i=0; i<n; i++){
61         underlying_order[i] = i;
62     }
63
64     prefix_sorting(tree, labels, underlying_order);
65 }
```

B Algoritmo - Hopcroft e Karp

```
bool BFS_hk(vector<int>* G, int* pair, int* dist){
    queue<int> Q;

    for (int i=0; i<n; i++){
        if (pair[i]==2*n){
            dist[i]=0;
            Q.push(i);
        }else{
            dist[i]=MAXN;
        }
    }

    dist[2*n]=MAXN;

    while(!Q.empty()){
        int i=Q.front();
        Q.pop();
        if(dist[i]<dist[2*n]){
            for(int j: G[i]){
                if(dist[pair[j]] == MAXN){
                    dist[pair[j]] = dist[i] + 1;
                    Q.push(pair[j]);
                }
            }
        }
    }

    return dist[2*n]!=MAXN;
}

bool DFS_hk(vector<int>* G, int* pair, int* dist, int i){

    if (i!=2*n){
        for(int j: G[i]){
            if(dist[pair[j]] == dist[i] + 1){
                if(DFS_hk(G, pair, dist, pair[j]) == true){
                    pair[j] = i;
                    pair[i] = j;
                    return true;
                }
            }
        }
        dist[i] = MAXN;
        return false;
    }

    return true;
}
```

```
void maximum_matching_hk(vector<int>* G, int* pair){
    int dist[2*n+1];

    for (int i=0; i<2*n; i++){
        pair[i] = 2*n;
    }
    while(BFS_hk(G, pair, dist)==true){
        for (int i=0; i<n; i++){
            if (pair[i] == 2*n){
                DFS_hk(G, pair, dist, i);
            }
        }
    }
}
```

C Algoritmo - BWT

```
bwt calculate_bwt(vector<edge>* automa, string* labels){
    int underlying_order[n];
    vector<inequality> maximal_order;
    vector<vector<int> > chain_decomposition;
    int order[n];
    vector<vector<string> > adjacency_matrix;
    bwt bwt_vec;
    bwt_vec.build();

    find_underlying_order(automa, underlying_order, labels);

    maximal_order = find_maximal_order(automa, underlying_order, labels);

    chain_decomposition = find_chain_decomposition(automa, maximal_order, order);

    adjacency_matrix = find_adjacency_matrix(automa, order);

    int index1=-1, c1=0;

    for (vector<int> row1: chain_decomposition){
        c1++;
        for (int i: row1){
            index1++;
            for (int u: fs){
                if(u == i){
                    bwt_vec.final[index1] = true;
                }
            }
            int index2=-1, c2=0;
            for (vector<int> row2: chain_decomposition){
                c2++;
                for (int j: row2){
                    UNUSED(j);
                    index2++;
                    if (adjacency_matrix[index1][index2]!=""){
                        bwt_vec.out_1[index1].push_back(c2);
                        bwt_vec.out_2[index1].push_back(adjacency_matrix[index1][
    index2]);
                        bwt_vec.in[index2].push_back(c1);
                    }
                }
            }
        }
    }

    return bwt_vec;
}
```

D Algoritmo - Inversione della BWT

```
void invert_bwt(bwt bwt_vec, vector<edge>* automa){

    //Ricostruzione funzione pi
    int n_chain = 0;
    int in_degree[n];

    for(int i=0; i<n; i++){
        for(int j: bwt_vec.out_1[i]){
            if (j > n_chain){
                n_chain = j;
            }
        }
    }

    int n_edges_in_chain[n_chain+1];
    int chain_decomposition[n];

    for(int i=0; i<n_chain+1; i++){
        n_edges_in_chain[i]=0;
    }
    for(int i=0; i<n; i++){
        for(int j: bwt_vec.out_1[i]){
            n_edges_in_chain[j]++;
        }
    }
    for(int i=0; i<n; i++){
        in_degree[i]=0;
        for(int j: bwt_vec.in[i]){
            UNUSED(j);
            in_degree[i]++;
        }
    }

    int j = 0;

    for(int i=1; i<n_chain+1; i++){
        int c = 0;
        while(c < n_edges_in_chain[i]){
            c = c + in_degree[j];
            chain_decomposition[j]=i;
            j++;
        }
    }

    //Ricostruzione funzione lambda
    int n_labels = 0;
    vector<string> poss_labels;
    vector<int> n_edges_per_label_in_chain[n_chain+1];
```

```
    for(int i=0; i<n; i++){
        for(string j: bwt_vec.out_2[i]){
            int flag=0;
            for(string u: poss_labels){
                if(u==j){
                    flag = 1;
                    break;
                }
            }
            if (flag==0){
                n_labels++;
                poss_labels.push_back(j);
            }
        }
    }
    for(string u: poss_labels){
        for (int k = 1; k < n_chain+1; k++){
            n_edges_per_label_in_chain[k].push_back(0);
        }
    }
    for(int i=0; i<n; i++){
        int u=0;
        for(string j: bwt_vec.out_2[i]){
            int k=0;
            for(string u: poss_labels){
                if(u==j){
                    break;
                }else{
                    k++;
                }
            }
            n_edges_per_label_in_chain[bwt_vec.out_1[i][u]][k]++;
            u++;
        }
    }

    poss_labels = merge_insertion_sort_invert(poss_labels, 0, n_labels-1,
    n_edges_per_label_in_chain, n_chain);

    string labels[n];

    int node = 0;
    for (int j=1; j<n_chain+1; j++){
        int i=0;
        for (int k: n_edges_per_label_in_chain[j]){
            int c=0;
            while(c < k){
                if(in_degree[node]==0){
                    labels[node]="";
                }else{
                    c = c + in_degree[node];
                    labels[node]=poss_labels[i];
                }
                node++;
            }
            i++;
        }
    }

    vector<int> in_app[n];
```

```
108     for(int i=0; i<n; i++){
109         for(int j: bwt_vec.in[i]){
110             in_app[i].push_back(j);
111         }
112     }
113 
114     //Costruzione automa
115     bool flag;
116     m = 0;
117 
118     for(int i=0; i<n; i++){
119         int k=-1;
120         for(int j: bwt_vec.out_1[i]){
121             k++;
122             flag = false;
123             //coppia (k,c)=(j,bwt_vec.out_2[i][k])
124             //pi(vi)=chain_decomposition[i]
125             for(int u=0; u<n; u++){
126                 if(chain_decomposition[u]==j && labels[u]==bwt_vec.out_2[i][k]){
127                 int c=0;
128                     for(int v: in_app[u]){
129                         if(chain_decomposition[i]==v){
130                             //(i,j)=(i,u)
131                             edge e;
132                             e.v=u;
133                             e.label=labels[u];
134                             automa[i].push_back(e);
135                             m++;
136                             in_app[u][c]=0;
137                             flag = true;
138                             break;
139                         }
140                         c++;
141                     }
142                 }
143                 if(flag){
144                     break;
145                 }
146             }
147         }
148     }
149 
150     //Individuazione sorgente
151     for(int i=0; i<n; i++){
152         if(labels[i]==""){
153             s = i;
154             break;
155         }
156     }
157 
158     //Individuazione stati finali
159     fs.clear();
160     nfs = 0;
161 
162     for(int i=0; i<n; i++){
163         if(bwt_vec.final[i]){
164             fs.push_back(i);
165             nfs++;
166         }
167     }
168 }
```

Riferimenti bibliografici

[1] Nicola Cotumaccio, Nicola Prezza. *On Indexing and Compressing Finite Automata.* Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), 2021.

[2] Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, Nicola Prezza. *Regular languages meet prefix sorting.* Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, pp. 911–930.

[3] John E. Hopcroft, Richard M. Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM Journal on Computing (2), pp. 225–231, 1973.

[4] L.R. Ford, D.R. Fulkerson. *Flows in networks.* Princeton University Press, 1962.

[5] Lester R. Ford Jr., Selmer M. Johnson (1959). *A Tournament Problem.* The American Mathematical Monthly, 66:5, 387-389, DOI: 10.1080/00029890.1959.11989306.

[6] Michael Burrows, David J. Wheeler. *A block-sorting lossless data compression algorithm.* 1994.

[7] Robert P. Dilworth. *A decomposition theorem for partially ordered sets.* In Classic Papers in Combinatorics, pages 139–144. Springer, 2009.