Complementi di Algoritmi e Strutture Dati

Suffix Trees

Docente: *Nicolò Cesa-Bianchi* versione 21 settembre 2017

In generale, possiamo trovare tutte le occorrenze di un pattern y in un testo x in tempo $\mathcal{O}(|x|+|y|)$, dove $\mathcal{O}(|y|)$ è il tempo di preelaborazione del pattern e $\mathcal{O}(|x|)$ è il tempo di matching. In alcune situazioni, però, il testo x è fisso e l'algoritmo riceve una sequenza $y_1, \ldots, y_N$ di pattern su cui fare matching. In questo caso, invece di impiegare un tempo $\mathcal{O}(|x| + |y_i|)$ per ciascun pattern y_i, $i = 1, \ldots, N$, possiamo usare la tecnica del suffix tree e impiegare un tempo $\mathcal{O}(|x|)$ per la preelaborazione del testo più un tempo $\mathcal{O}(|y_i| + k_i)$ per il matching di ciascun pattern y_i, dove k_i è il numero di occorrenze di y_i in x.

Oltre a risolvere il problema di string matching, i suffix trees permettono di risolvere in modo efficiente una varietà di altri problemi. Per esempio:

1. trovare il più lungo prefisso del pattern che compare nel testo
2. trovare fra tutte le coppie di prefissi del testo quella col più lungo prefisso comune
3. trovare la più lunga sottostringa fra tutte quelle che compaiono almeno due volte nel testo.

Questi problemi trovano applicazioni in diversi settori: compressione dati, information retrieval, bioinformatica, elaborazione di testi, linguaggi di programmazione.

Sia Σ un alfabeto finito e ordinato e sia $\$ \notin \Sigma$ un carattere di terminazione. Data una stringa $x \in \Sigma^n$ e due interi $1 \le i \le j \le n$ indichiamo con $x[i \ldots j]$ la sottostringa che inizia in posizione i e termina in posizione j. I suffissi di una stringa $x\$$ con $x \in \Sigma^n$ sono

$$x[1 \ldots n]\$,\ x[2 \ldots n]\$,\ \ldots,\ x[n]\$,\ \$$$

Un albero di suffissi T per una stringa $x \in \Sigma^n$ è un albero diretto con radice e $n+1$ foglie numerate tale che:

1. ogni nodo interno ha almeno due figli (tranne quando l'albero ha soltanto due nodi: radice e una foglia)
2. ogni arco è etichettato da una sottostringa di $x\$$
3. le sottostringhe sugli archi che escono da un qualunque nodo iniziano con caratteri a due a due distinti
4. per ogni $i = 1, \ldots, n+1$ il suffisso $x\$$ che inizia al carattere i è ottenuto concatenando le sottostringhe che etichettano gli archi sul cammino dalla radice alla foglia i.

Si noti che ogni suffisso che è prefisso di un qualche altro suffisso non può stare su una foglia. La presenza del carattere \$ assicura che tali suffissi non esistono. Inoltre, dato che ogni nodo interno ha almeno due figli, ci possono essere al più n nodi interni. Quindi T ha al più $2n+1$ nodi e profondità al più n. Si veda la Figura 1 per un esempio.

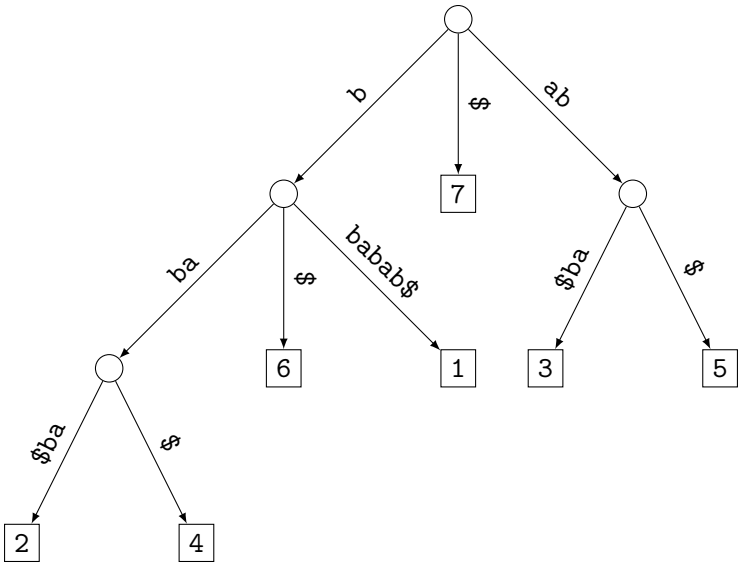

Figura 1: Il suffix tree per il testo `bbabab$`. Le etichette sugli archi vanno lette dall'alto in basso.

La *pathstring* di un nodo è la concatenazione delle sottostringhe che etichettano gli archi sul cammino dalla radice al nodo. Quindi una pathstring è sempre il prefisso di un qualche suffisso di x.

Fatto 1 *Fissati un testo $x \in \Sigma^* \setminus \{\varepsilon\}$ con suffix tree T e un pattern $y \in \Sigma^*$.*

1. ***(Esistenza).*** *y, ya e yb (con $a, b \in \Sigma \cup \{\$\}$ e $a \neq b$) sono sottostringhe di $x\$$ se e solo se esiste un nodo interno di T che ha y come pathstring.*
2. ***(Completezza).*** *Una stringa $y \in \Sigma^*$ è una sottostringa di x se e solo se y è prefisso di una qualche pathstring di T.*

Dimostrazione. Per la parte 1: se ya e yb sono entrambe sottostringhe di $x\$$ e $a \neq b$ allora esistono due suffissi distinti yaw e ybz di $x\$$. Dato che le sottostringhe sugli archi che escono da un qualunque nodo iniziano con caratteri distinti, i cammini che portano alle due foglie corrispondenti ai suffissi devono essere coincidenti fino ad un nodo la cui pathstring è y, dopo il quale devono necessariamente divergere. Viceversa, se esiste un nodo interno di T che ha y come pathstring, allora y è una sottostringa di $x\$$ e le stringhe sugli archi uscenti (che sono almeno due) devono iniziare con caratteri distinti. Supponiamo che a e b siano due di questi caratteri. Allora anche ya e yb sono sottostringhe di $x\$$.

Per la parte 2: ovviamente, y è una sottostringa di x se e solo se y è prefisso di un qualche suffisso di x. Quindi una sottostringa y di x è prefisso della foglia di T che ha quel suffisso come pathstring. D'altra parte, ogni pathstring in T è prefisso di un qualche suffisso. Quindi se y è prefisso di una pathstring allora dev'essere una sottostringa di x. □

String matching con suffix trees. Dato un suffix tree T per un testo $x \in \Sigma^n$, per trovare tutti i match di un pattern $y \in \Sigma^m$ parto dalla radice e seguo il cammino che passa per nodi la cui

pathstring è un prefisso di y. Mi fermo quando non riesco più a trovare un cammino che soddisfa tale condizione, oppure quando arrivo ad un nodo con pathstring yw per un qualche $w \in \Sigma^*$. Nel primo caso non esistono match di y in x; nel secondo caso ogni foglia nel sottoalbero del nodo di arrivo corrisponde ad un match distinto di y in x. La posizione di ognuno di questi match è il numero della foglia nel sottoalbero. Queste foglie possono essere trovate mediante un attraversamento depth-first del sottoalbero.

La correttezza dell'algoritmo è una conseguenza della proprietà di completezza (parte 2 del Fatto 1). In particolare, se y ha più match in x, allora ci saranno più suffissi di x di cui y è prefisso. L'algoritmo si fermerà allora al nodo la cui pathstring ha lunghezza minima fra tutte quelle che hanno y come prefisso. Le pathstring delle foglie nel sottoalbero del nodo sono tutti suffissi di x che hanno y come prefisso, e quindi corrispondono a match validi.

Il tempo totale per il matching di un pattern $y \in \Sigma^m$ è $\mathcal{O}(m + k)$, dove k è il numero totale di match di y in x. Il termine m è il numero massimo di archi attraversati in T prima trovare un nodo con una pathstring di cui y è prefisso (o decidere che un tale nodo non esiste quando $k = 0$). Per trovare le posizioni dei $k > 0$ match l'algoritmo esegue depth first su un sottoalbero di al più $k+k-1$ nodi che corrispondono a $k+k-2 = 2(k-1)$ archi. In depth first, ogni arco è attraversato due volte, e quindi il numero totale di attraversamenti è al più $4(k-1) = \mathcal{O}(k)$.

Construzione di un suffix tree in tempo lineare. Nel seguito, identifichiamo ogni nodo con la sua pathstring. L'algorithmo di Chen e Seiferas lavora in tempo lineare leggendo il testo da destra a sinistra. Per ogni nodo z, l'algoritmo mantiene puntatori a:

- nodi figli
- nodo padre
- per ogni $a \in \Sigma$ un puntatore al nodo (se esiste) la cui pathstring è la più breve estensione di az (questi vengono chiamati nodi *a-shortcut*).

Se il testo è vuoto, cioè consiste soltanto del carattere \$, allora il suffix tree consiste soltanto di due nodi e può essere costruito in tempo costante. Per costruire il suffix tree di $az\$$ assumiamo induttivamente di aver già costruito il suffix tree per $z\$$ e di avere puntatori alla radice e alla foglia $z\$$ di quest'albero. Dimostriamo ora come costruire il suffix tree per $az\$$ a partire da quello per $z\$$. Per prima cosa dobbiamo aggiungere la foglia $az\$$ come figlia di un nuovo nodo o come figlia di un nodo già esistente. Il padre di $az\$$ sarà evidentemente il più lungo prefisso y di $az\$$ che è anche una sottostringa di $z\$$. Dobbiamo quindi trovare la posizione di y nell'albero per $z\$$.

Lemma 2 *Sia $y = av$ il più lungo prefisso di $az\$$ che è anche una sottostringa di $z\$$. Allora nel suffix tree per $z\$$ ci dev'essere un nodo v.*

Dimostrazione. Dimostriamo che esistono $b, c \in \Sigma \cup \{\$\}$ con $b \neq c$ tali che vb e vc sono entrambe sottostringhe di $z\$$. Per la proprietà di esistenza (parte 1 del Fatto 1) ne deriva che esiste un nodo con pathstring v.

Cominciamo coll'osservare che y non può terminare con \$ in quanto y è sia un prefisso di $az\$$ che una sottostringa di $z\$$. Sia quindi b il carattere che segue y in $az\$$. Ma allora $yb = avb$ è pure un prefisso di $az\$$, da cui ne segue che vb è una sottostringa di $z\$$. D'altra parte, dato che y è il più lungo prefisso di $az\$$ che è anche una sottostringa di $z\$$, yb è un prefisso di $az\$$ che non è una

sottostringa di $z\$$. Ora, dato che y è una sottostringa di $z\$$ che non contiene \$, ci dev'essere una carattere $c \in \Sigma \cup \{\$\}$ tale che yc è pure una sottostringa di $z\$$, da cui $b \neq c$. Inoltre, dato che $y = av$, anche vc è sottostringa di $z\$$. In conclusione, esistono b e c con $b \neq c$ tali che vb e vc sono entrambi sottostringhe di $z\$$. □

Per cercare il nodo y sotto le ipotesi del Lemma 2, notiamo che tale lemma garantisce l'esistenza di un nodo v. Inoltre, tale nodo deve trovarsi sul cammino dalla radice alla foglia $z\$$ dato che v è prefisso di $z\$$. Infine, risalendo il cammino dalla foglia $z\$$, il nodo v sarà il primo con un puntatore ad un nodo a-shortcut. Infatti, av è una sottostringa di $z\$$ e quindi ci sarà un nodo la cui pathstring è la più breve estensione di av; se ci fosse un nodo che estende v e possiede un nodo a-shortcut, allora av non sarebbe il più lungo prefisso di $az\$$ che è anche sottostringa di $z\$$.

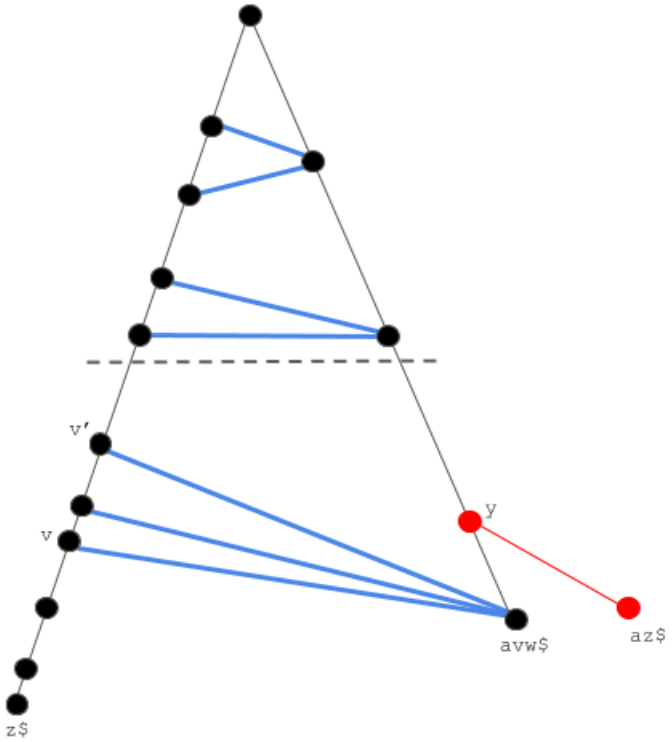

Figura 2: Aggiornamento del suffix tree per $z\$$ con l'aggiunta del nodo y e della foglia $az\$$ (entrambi in rosso). Gli archi blu indicano i nodi a destra che sono a-shortcut di nodi a sinistra. I nodi a sinistra a monte della linea tratteggiata sono in numero non inferiore ai nodi a sinistra a monte della linea tratteggiata (si veda la dimostrazione del Teorema 3).

Seguendo il puntatore al nodo a-shortcut possono succedere due cose. Se il nodo è $av = y$ non devo fare nulla. Se il nodo è avw (la più breve estensione di av che corrisponde a un nodo), allora creo un nuovo nodo y fra avw e il suo genitore. In entrambi i casi creo la foglia $az\$$ come figlia di y. Si noti che y avrà gli stessi puntatori ai nodi di shortcut di avw (non dimostrato). Invece, i nodi che avranno un nuovo puntatore a y come nodo a-shortcut saranno quelli sul cammino che da v risale all'ultimo nodo v' che non ha il genitore di y come nodo a-shortcut. I nodi che avranno un nuovo puntatore a $az\$$ come nodo a-shortcut saranno tutti i nodi sul cammino che da $z\$$ risale all'ultimo

nodo u senza un puntatore ad un nodo a-shortcut (si noti che u è figlio di v) —si veda la Figura 2 per un'ilustrazione.

La creazione di questi puntatori richiede di risalire il cammino da $z\$$ a v'. Quindi il tempo per aggiornare il suffix tree per $z\$$ è proporzionale alla lunghezza di questo cammino.

Nel seguito, useremo $\pi(v)$ per indicare il cammino dalla radice del suffix tree al nodo v e $\big|\pi(v)\big|$ per indicare il numero di archi in questo cammino.

Teorema 3 *Il suffix tree per un testo $x \in \Sigma^*$ può essere costruito in tempo $\mathcal{O}(|x|)$.*

Dimostrazione. Usiamo un'analisi ammortizzata per stabilire come il tempo per aggiornare l'albero da $z\$$ a $az\$$ possa essere messo in relazione con la riduzione in altezza dell'albero. Cioè, quanto più risaliamo da $z\$$ a v', quanto meno dovremo risalire da $az\$$ nel passo successivo.

Dimostriamo ora che per ogni nodo av'' in $\pi(y)$ c'è un nodo v'' in $\pi(v')$. Per prima cosa, fissiamo un nodo av'' in $\pi(y)$. Allora la parte 1 del Fatto 1 implica che $av''b\$$ e $av''c\$$ sono prefissi di $az\$$ con $b \neq c$. Quindi $v''b\$$ e $v''c\$$ sono prefissi di $z\$$ e perciò, sempre per la parte 1 del Fatto 1, anche v'' è un nodo in $\pi(z\$)$. Si noti che, per definizione di nodo shortcut, av'' è il nodo a-shortcut per v''. Dato che v' è il primo nodo sul cammino $\pi(v)$ per il quale y è il nodo a-shortcut e dato che av'' viene prima di y, v'' deve venire prima di v' sul cammino $\pi(v)$. Da questo fatto possiamo quindi dedurre che $\big|\pi(y)\big| \leq \big|\pi(v')\big|$.

Sia $s_i = x[i, \ldots, n]\$$ l'i-esimo suffisso di $x\$$ per $i = 1, \ldots, n+1$. Sia τ_{i+1} il tempo che l'algoritmo impiega per aggiornare l'albero T_{i+1} per s_{i+1} ottenendo l'albero T_i per s_i. Sia v'_{i+1} il nodo v' sull'albero T_{i+1} e sia y_i il nodo y sull'albero T_i. Sappiamo quindi che $\big|\pi(y_i)\big| \leq \big|\pi(v'_{i+1})\big|$ per ogni $i = 1, \ldots, n$. Sappiamo inoltre che

$$\tau_{i+1} = \mathcal{O}\Big(\big|\pi(s_{i+1})\big| - \big|\pi(v'_{i+1})\big|\Big)$$

Infine, $\big|\pi(s_i)\big| = \big|\pi(y_i)\big| + 1$ in quanto il nodo s_i è creato come figlio di y_i. Si noti quindi che

$$\big|\pi(s_{i+1})\big| - \big|\pi(v'_{i+1})\big| \leq \big|\pi(s_{i+1})\big| - \big|\pi(y_i)\big| \leq \big|\pi(s_{i+1})\big| - \big|\pi(s_i)\big| + 1$$

Quindi

$$\begin{aligned}\sum_{i=1}^{n} \Big(\big|\pi(s_{i+1})\big| - \big|\pi(v'_{i+1})\big|\Big) &\leq \sum_{i=1}^{n} \Big(\big|\pi(s_{i+1})\big| - \big|\pi(s_i)\big| + 1\Big) \\ &= \big|\pi(s_{n+1})\big| - \big|\pi(s_1)\big| + n \leq n\end{aligned}$$

dato che $\big|\pi(s_{n+1})\big| = 1 \leq \big|\pi(s_1)\big|$. Da ciò otteniamo

$$\sum_{i=1}^{n} \tau_{i+1} = \sum_{i=1}^{n} \mathcal{O}\Big(\big|\pi(s_{i+1})\big| - \big|\pi(v'_{i+1})\big|\Big) = \mathcal{O}(n)$$

che conclude la dismostrazione. □