

# Introduzione

Esistono diverse informazioni interessanti che riguardano i fattori di un testo. Dato un alfabeto  $\Sigma$ , un testo  $T$  e una parola  $x$ , consideriamo quattro problemi che ricorrono spesso nell'analisi del testo:

## 1) Problema di appartenenza all'indice

Comunque preso  $x \in \Sigma$ , trovare se  $x$  appartiene al testo ed eventualmente trovare il più lungo prefisso di  $x$  che appartiene ai fattori del testo.

## 2) Problema della posizione

Sia  $x$  appartenente ai fattori di un testo, vogliamo trovare la prima posizione della parola nel testo.

## 3) Problema delle occorrenze

Sia  $x$  appartenente ai fattori di un testo, vogliamo trovare il numero delle occorrenze di  $x$  nel testo.

## 4) Problema della lista delle posizioni

Sia  $x$  appartenente ai fattori di un testo, vogliamo trovare la lista delle posizioni delle occorrenze di  $x$ .

### Nota: tempi d'esecuzione delle query

Crochemore dà una definizione di "abstract data type", quindi senza tempi ma alcuni autori richiedono, ad una struttura ad indici, dei tempi di risposta alle query parecchio inferiori alla taglia del database.

Distinzione tra preprocessing sul pattern e preprocessing sul testo.

Strutture e algoritmi che effettuano il preprocessing sul pattern.

RICHIAMO: Automa dei fattori del pattern

RICHIAMO: Automa dei fattori del testo che risponde alla query esistenziale

RICHIAMO: Dimostrare che tale automa ha taglia lineare rispetto al testo

# Preprocessing sul testo

## Suffix trie

Il Suffix trie è una struttura ad albero deterministico che rappresenta tutti i suffissi di un testo in cui ogni arco è etichettato con una sola lettera.

Note:

- Le foglie indicano un suffisso.
- Ogni sottostringa del testo può essere trovata, partendo dalla radice, attraversando un percorso.
- Ogni nodo interno rappresenta una sottostringa che appare più di una volta nel testo.

## Labels dei nodi

Le labels da assegnare ad ogni nodo si ottengono, mediante la concatenazione dei caratteri incontrati, percorrendo il cammino che dalla radice porta al nodo stesso.

## Suffix Link

Dato un nodo  $v$  etichettato  $xa$  (con  $a$  pattern eventualmente nullo) e un nodo  $w$  etichettato con  $a$ , l'arco  $(v,w)$  è chiamato suffix link.

## Suffix tree

È costituito dalla coppia albero-testo in cui l'albero si ottiene eliminando nel suffix trie i nodi che hanno un solo figlio e i relativi archi uscenti. Successivamente, i nodi rimasti isolati si riconnettono all'antenato più vicino mediante un arco. Gli archi sono etichettati da due interi che puntano al testo (vedi edge-label compression a pag. 104).

La coppia di numeri che etichettano l'arco individuano univocamente una sottostringa del testo. Esistono diversi criteri di assegnazione, che ad esempio possono essere: inizio-lunghezza, fine-lunghezza o inizio-fine. La scelta del criterio dipende dalle finalità prefissate.

## Utilizzo del terminatore \$

Se non si utilizza un terminatore di fine stringa, esisteranno dei suffissi (*net trie*) non rappresentati dalle foglie ma da nodi interni. Questo accade quando un suffisso è prefisso di un altro suffisso. Per risolvere il problema basta aggiungere un terminatore di fine stringa, convenzionalmente si utilizza il carattere "\$".

## Suffix link nel tree

Dimostriamo che il suffix link è ben definito anche nel suffix tree. Occorre provare che se  $xv \in \text{Tree}$  allora esiste  $v \in \text{Tree}$ .

**Caso 1)**  $xv$  è foglia:

Nota: per alcuni autori il suffix link non è definito per le foglie.

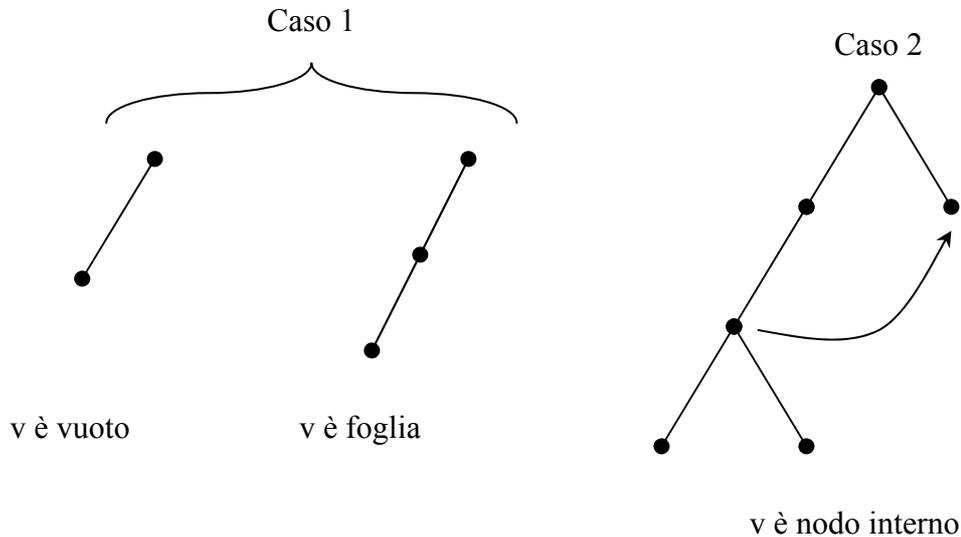
Se  $xv$  è foglia  $\rightarrow$  l'ultima lettera di  $xv$  è \$. Allora si può verificare uno dei seguenti casi:

- 1)  $v$  è la radice ( $v$  è vuoto, in questo caso  $x = \$$ )
- 2)  $v$  è foglia (ha \$ come ultima lettera)

In entrambi i casi  $v \in \text{Tree}$ .

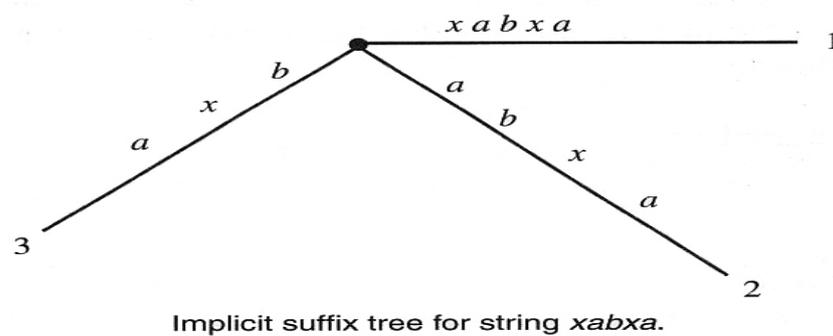
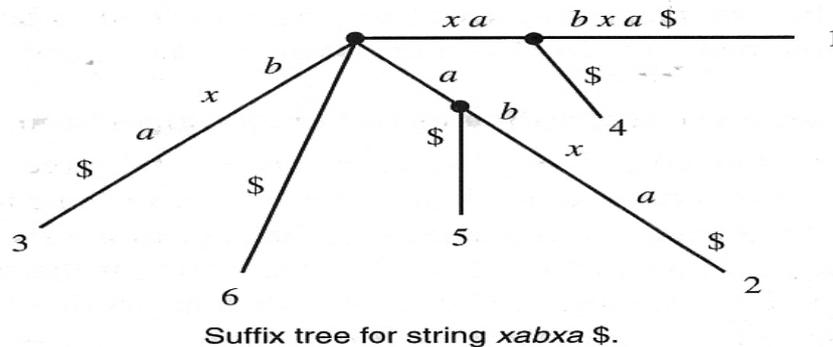
**Caso 2)**  $xv$  è un nodo interno:

Per definizione, esistono  $a, b \in \Sigma$  con  $a \neq b$  tali che  $xva$  e  $xvb$  sono fattori di  $w \rightarrow v \in \text{Tree}$ , quindi il suffix link esiste ed è ben definito.



### Implicit suffix tree

Data una stringa "S", l'implicit suffix tree è ottenuto dal suffix tree che rappresenta la stringa terminata "\$", eliminando il terminatore, rimuovendo gli archi rimasti senza label e i nodi rimasti con meno di due figli.



### Generalized suffix tree per un set di stringhe

Occorre estendere il concetto di suffix tree ad un set di stringhe  $[S_1, S_2, \dots, S_n]$  al fine di rappresentarne i suffissi.

### Metodo intuitivo o naive:

- 1) Si utilizza un diverso marcatore di fine per ogni parola  $\alpha, \beta, \gamma, \delta, \dots$
- 2) Si concatenano insieme le stringhe
- 3) Si costruisce il suffix tree di  $W=S_1\alpha S_2\beta\dots S_n\gamma$

Ad ogni foglia si associa un doppio indice che identifica la parola e la posizione di partenza nella parola stessa.

Note:

- Poiché i markers sono diversi, le etichette dei percorsi dalla radice a un qualunque nodo interno deve essere una sottostringa di una delle stringhe di partenza.
- Questo metodo richiede tempo proporzionale alla somma delle lunghezze di ogni stringa concatenata.
- Il difetto del metodo è che l'albero rappresenta sottostringhe che attraversano più di una stringa di partenza. Si può comunque porre rimedio percorrendo tutti i percorsi possibili, a partire dalla radice ed eliminando i caratteri successivi al primo marcatore incontrato.

### Metodo principale:

Date due parole  $S_1$  e  $S_2$ :

- 1) Si costruisce il suffix tree per  $S_1$
- 2) Sul suffix tree, appena costruito, si aggiungono caratteri di  $S_2$  rispettando il seguente criterio:
  - Fino a quando i caratteri di  $S_2$  matchiano quelli di  $S_1$  non occorre alcuna azione
  - Quando si verifica un "mismatch", si aggiunge una biforcazione e alla fine dell'arco, si inserirà la doppia label per  $S_2$

Possiamo estendere il ragionamento per un set di stringhe  $[S_1, S_2, \dots, S_n]$ .

Esempio:

$S_1 =$  x a b x a \$  
1 2 3 4 5 6

$S_2 =$  b a b x b a \$  
1 2 3 4 5 6 7

