# Elementi di Bioinformatica

Gianluca Della Vedova

Univ. Milano–Bicocca
http://gianluca.dellavedova.org

22 ottobre 2018

# Trie

## Trie

- Albero
- Query: parola $\in$ dizionario
- archi etichettati
- Percorso radice-foglia $=$ parola

## Dizionario

ABRACADABRA
ARRAY

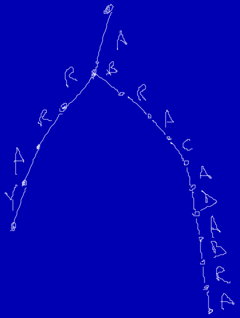
# Trie

## Trie

- Albero
- Query: parola $\in$ dizionario
- archi etichettati
- Percorso radice-foglia $=$ parola

## Dizionario

ABRACADABRA
ARRAY



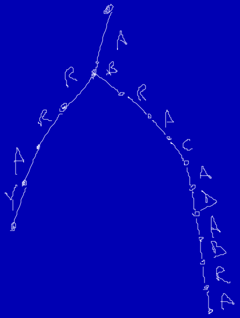
# Trie

## Trie

- Albero
- Query: parola $\in$ dizionario
- archi etichettati
- Percorso radice-foglia $=$ parola

## Dizionario

ABRACADABRA
ARRAY
ABRA



# Trie

## Terminatore

\$ non appartiene all'alfabeto

# Trie

## Terminatore

$ non appartiene all'alfabeto

## Dizionario

ABRACADABRA$
ARRAY$
ABRA$

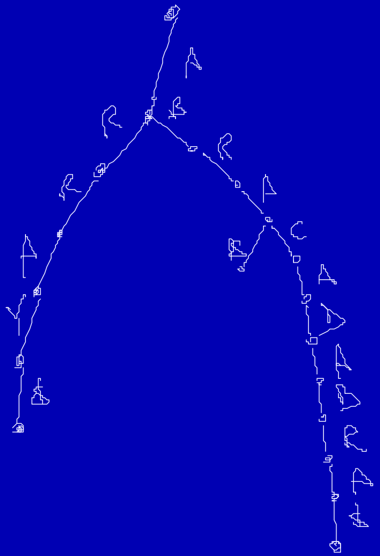
# Trie

## Terminatore

$ non appartiene all'alfabeto

## Dizionario

ABRACADABRA$
ARRAY$
ABRA$



# Suffix tree

## Definizione

- Trie compatto di tutti i suffissi di $T\$$

# Suffix tree

## Definizione

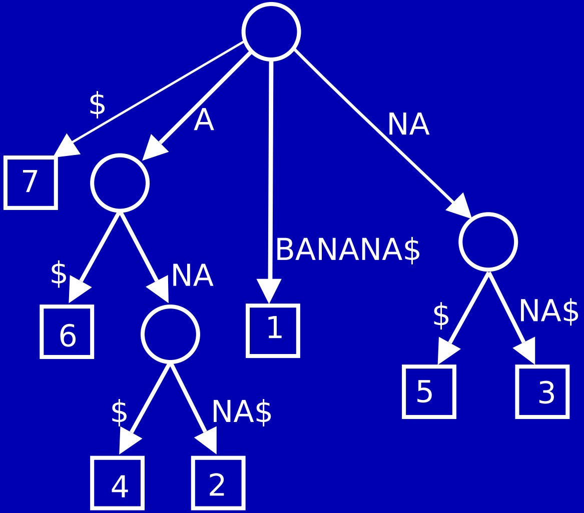
- Trie compatto di tutti i suffissi di $T\$$
- Le etichette degli archi uscenti da $x$ iniziano con simboli diversi

# Suffix tree

## Definizione

- Trie compatto di tutti i suffissi di $T\$$
- Le etichette degli archi uscenti da $x$ iniziano con simboli diversi
- suffissi $\iff$ percorso radice-foglia

# Suffix tree




## Definizione

- Trie compatto di tutti i suffissi di $T\$$
- Le etichette degli archi uscenti da $x$ iniziano con simboli diversi
- suffissi $\iff$ percorso radice-foglia

BANANA$

# Suffix tree 2: Definizione

- foglie etichettata con posizione inizio suffisso
- path-label($x$): concatenazione etichette
- string-depth($x$): lunghezza path-label($x$)
- Pattern matching = visita

## Problemi

- Spazio $O(n^2)$

# Suffix tree 2: Definizione

- foglie etichettata con posizione inizio suffisso
- path-label($x$): concatenazione etichette
- string-depth($x$): lunghezza path-label($x$)
- Pattern matching = visita

## Problemi

- Spazio $O(n^2)$
- Puntatori al testo (posizioni)

# Suffix tree 2: Definizione

- foglie etichettata con posizione inizio suffisso
- path-label($x$): concatenazione etichette
- string-depth($x$): lunghezza path-label($x$)
- Pattern matching = visita

## Problemi

- Spazio $O(n^2)$
- Puntatori al testo (posizioni)
- Spazio $20n$ bytes

# Suffix array

## Definizione

- Array dei suffissi in ordine lessicografico

# Suffix array

## Definizione

- Array dei suffissi in ordine lessicografico
- Posizioni iniziali del suffisso nell'array

# Suffix array

## Definizione

- Array dei suffissi in ordine lessicografico
- Posizioni iniziali del suffisso nell'array
- Spazio $4n$ bytes

# Suffix array

## Definizione

- Array dei suffissi in ordine lessicografico
- Posizioni iniziali del suffisso nell'array
- Spazio $4n$ bytes
- $Lcp[i]$: lunghezza prefisso comune $SA[i]$, $SA[i+1]$

# Suffix array

## Definizione

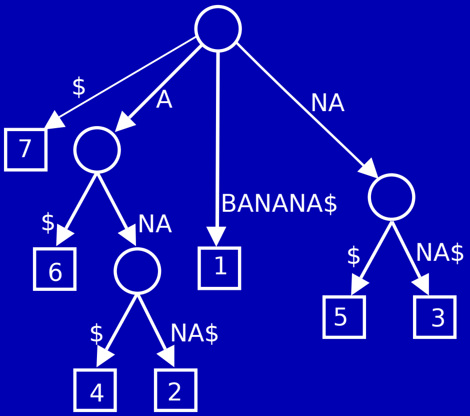
- Array dei suffissi in ordine lessicografico
- Posizioni iniziali del suffisso nell'array
- Spazio $4n$ bytes
- $Lcp[i]$: lunghezza prefisso comune $SA[i]$, $SA[i+1]$

## BANANA$

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $SA$ | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $Lcp$ | 0 | 1 | 3 | 0 | 0 | 2 | - |

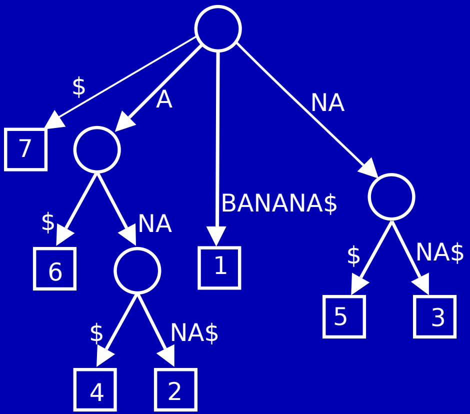
# Da Suffix tree a Suffix array

- Visita depth-first di $ST$



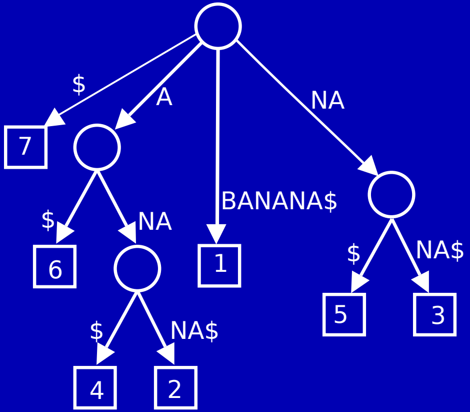
# Da Suffix tree a Suffix array

- Visita depth-first di $ST$
- archi uscenti di ogni nodo in ordine lessicografico



# Da Suffix tree a Suffix array

- Visita depth-first di $ST$
- archi uscenti di ogni nodo in ordine lessicografico
- $Lcp[i] =$ string-depth di $lca(i, i+1)$



# Da Suffix array a Suffix tree

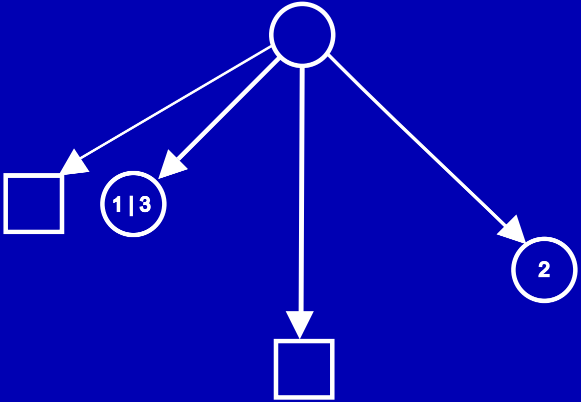
- $Lcp = 0$: partizione $SA$

## BANANA$

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $SA$ | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $Lcp$ | 0 | 1 | 3 | 0 | 0 | 2 | - |

# Da Suffix array a Suffix tree

- $Lcp = 0$: partizione $SA$
- corrispondono ai figli della radice




BANANA$

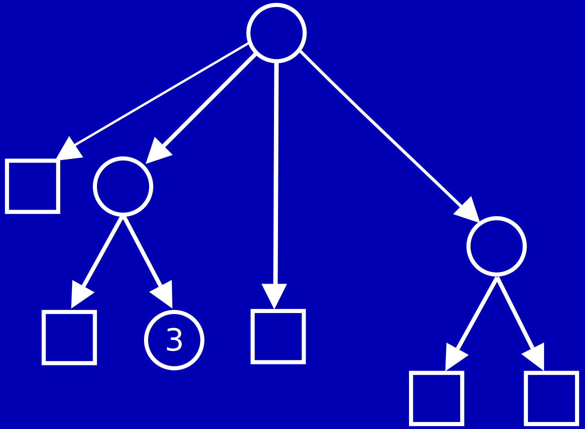
| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $SA$ | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $Lcp$ | 0 | 1 | 3 | 0 | 0 | 2 | - |

# Da Suffix array a Suffix tree

- $Lcp = 0$: partizione $SA$
- corrispondono ai figli della radice
- ricorsione prendendo i numeri minimi

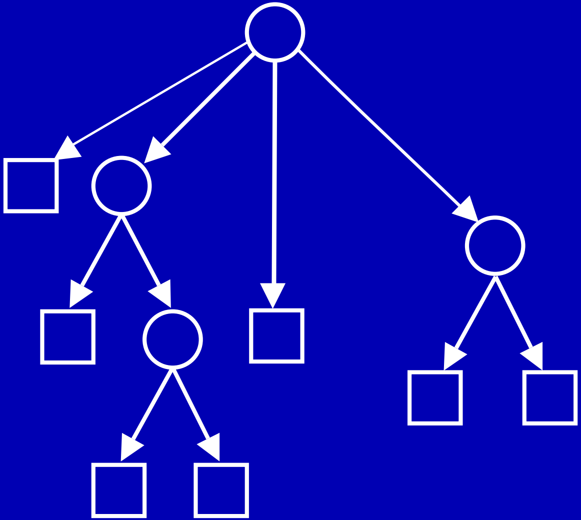
BANANA$

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $SA$ | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $Lcp$ | 0 | 1 | 3 | 0 | 0 | 2 | - |



# Da Suffix array a Suffix tree

- $Lcp = 0$: partizione $SA$
- corrispondono ai figli della radice
- ricorsione prendendo i numeri minimi



**BANANA$**

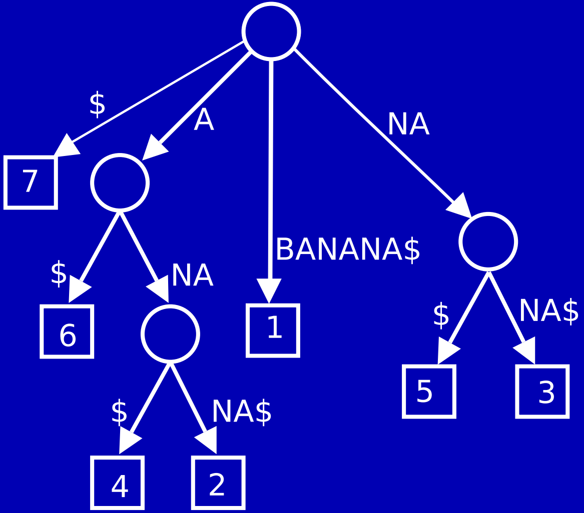
| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $SA$ | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $Lcp$ | 0 | 1 | 3 | 0 | 0 | 2 | - |

# Da Suffix array a Suffix tree

- $Lcp = 0$: partizione $SA$
- corrispondono ai figli della radice
- ricorsione prendendo i numeri minimi

**BANANA$**

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $SA$ | 7 | 6 | 4 | 2 | 1 | 5 | 3 |
| $Lcp$ | 0 | 1 | 3 | 0 | 0 | 2 | - |



# Sottostringa comune più lunga di due stringhe

## Due stringhe $s_1$ e $s_2$

- Suffix tree generalizzato = insieme di stringhe

# Sottostringa comune più lunga di due stringhe

## Due stringhe $s_1$ e $s_2$

- Suffix tree generalizzato $=$ insieme di stringhe
- $ST(s_1\$_1 s_2\$_2)$

# Sottostringa comune più lunga di due stringhe

## Due stringhe $s_1$ e $s_2$

- Suffix tree generalizzato = insieme di stringhe
- $ST(s_1\$_1 s_2\$_2)$
- Nodo $x$ con foglie di $s_1$ e $s_2$

# Sottostringa comune più lunga di due stringhe

## Due stringhe $s_1$ e $s_2$

- Suffix tree generalizzato = insieme di stringhe
- $ST(s_1\$_1 s_2\$_2)$
- Nodo $x$ con foglie di $s_1$ e $s_2$
- Sottostringa di $s_1$ e $s_2$

# Sottostringa comune più lunga di due stringhe

## Due stringhe $s_1$ e $s_2$

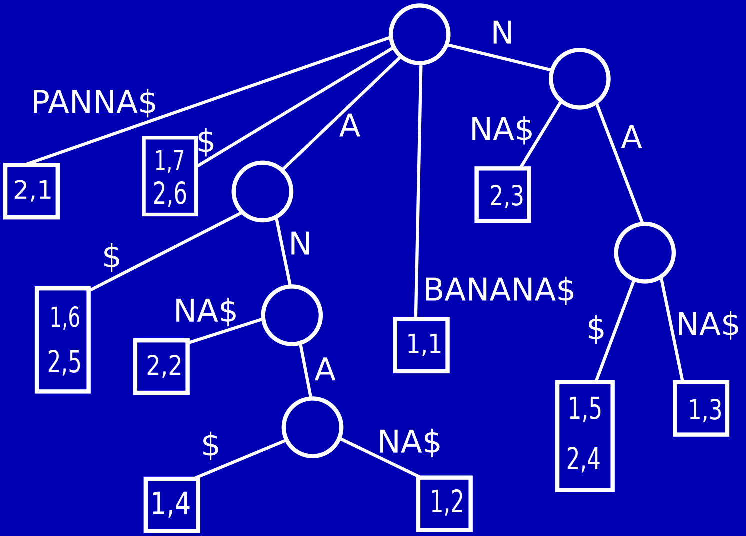
- Suffix tree generalizzato = insieme di stringhe
- $ST(s_1\$_1 s_2\$_2)$
- Nodo $x$ con foglie di $s_1$ e $s_2$
- Sottostringa di $s_1$ e $s_2$
- $ST(s_1\$s_2\$)$

# Sottostringa comune più lunga di due stringhe

## Due stringhe $s_1$ e $s_2$

- Suffix tree generalizzato = insieme di stringhe
- $ST(s_1\$_1 s_2\$_2)$
- Nodo $x$ con foglie di $s_1$ e $s_2$
- Sottostringa di $s_1$ e $s_2$
- $ST(s_1\$s_2\$)$
- Max string-depth

# Suffix tree generalizzato




$s_1$: BANANA$

$s_2$: PANNA$

# Pattern matching su suffix array

## Occorrenza $P$ in $T$

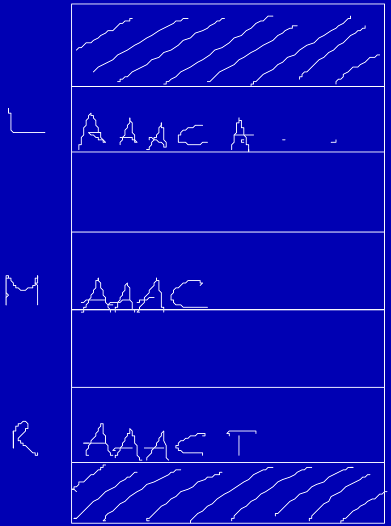
Suffissi di $T$ che iniziano con $P$

## Ricerca in $SA$

- Ricerca dicotomica
- Tempo $O(m \log n)$ – caso pessimo
- Controllare tutto $P$ ad ogni iterazione
- $log_2 n$ iterazioni

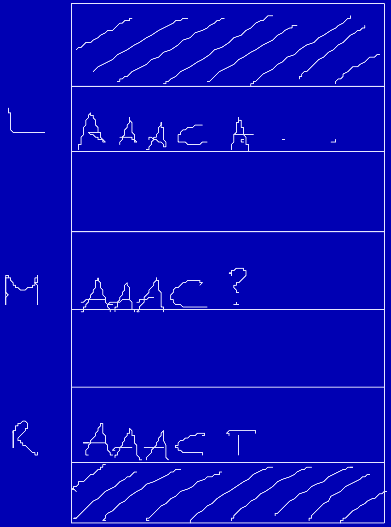
# Accelerante 1

- Intervallo $SA(L,R)$ di $SA$
- Elemento mediano $M$
- Tutti i suffissi in $SA(L,R)$ iniziano con uno stesso prefisso lungo $Lcp(SA[L],SA[R])$
- Non confrontare con i primi $Lcp(SA[L],SA[R])$ caratteri



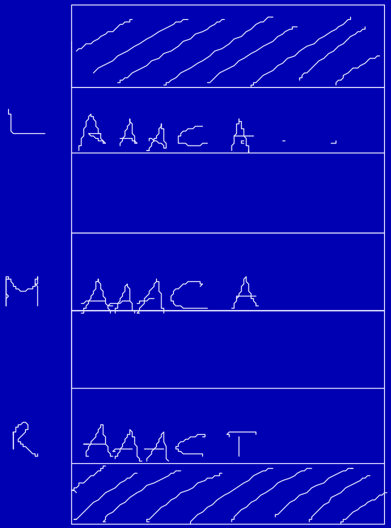
# Accelerante 1

- Intervallo $SA(L, R)$ di $SA$
- Elemento mediano $M$
- Tutti i suffissi in $SA(L, R)$ iniziano con uno stesso prefisso lungo $Lcp(SA[L], SA[R])$
- Non confrontare con i primi $Lcp(SA[L], SA[R])$ caratteri



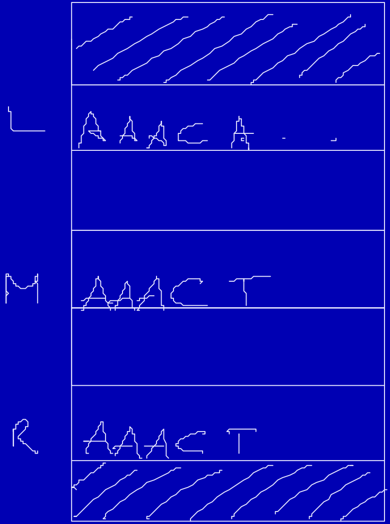
# Accelerante 1

- Intervallo $SA(L, R)$ di $SA$
- Elemento mediano $M$
- Tutti i suffissi in $SA(L, R)$ iniziano con uno stesso prefisso lungo $Lcp(SA[L], SA[R])$
- Non confrontare con i primi $Lcp(SA[L], SA[R])$ caratteri



# Accelerante 1

- Intervallo $SA(L,R)$ di $SA$
- Elemento mediano $M$
- Tutti i suffissi in $SA(L,R)$ iniziano con uno stesso prefisso lungo $Lcp(SA[L],SA[R])$
- Non confrontare con i primi $Lcp(SA[L],SA[R])$ caratteri



# Accelerante 1

- Intervallo $SA(L,R)$ di $SA$
- Elemento mediano $M$
- Tutti i suffissi in $SA(L,R)$ iniziano con uno stesso prefisso lungo $Lcp(SA[L],SA[R])$
- Non confrontare con i primi $Lcp(SA[L],SA[R])$ caratteri

# Accelerante 2

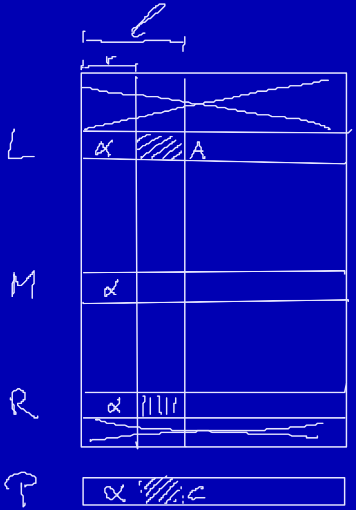
**$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$**

1. Caso 1: $l > r$

# Accelerante 2

**$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$**

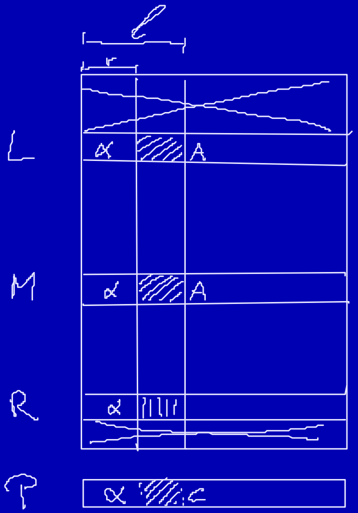
1. Caso 1: $l > r$



# Accelerante 2

$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$

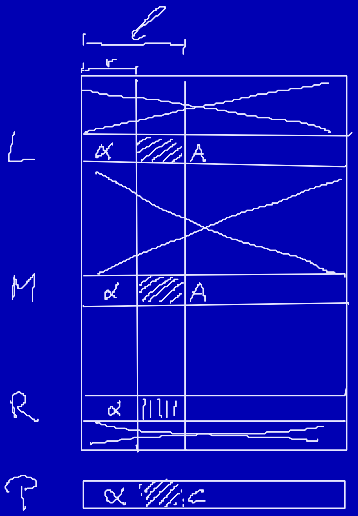
1. Caso 1: $l > r$
   - $Lcp(L, M) > l$



# Accelerante 2

$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$

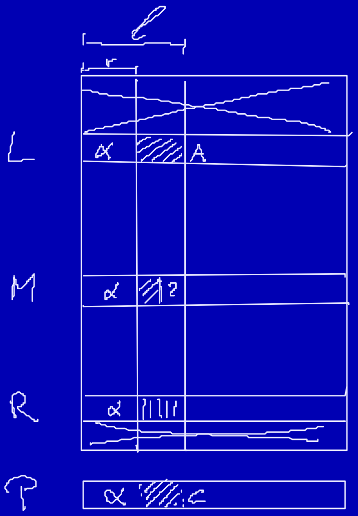
1. Caso 1: $l > r$
   - $Lcp(L, M) > l \Rightarrow L \leftarrow M$



# Accelerante 2

$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$

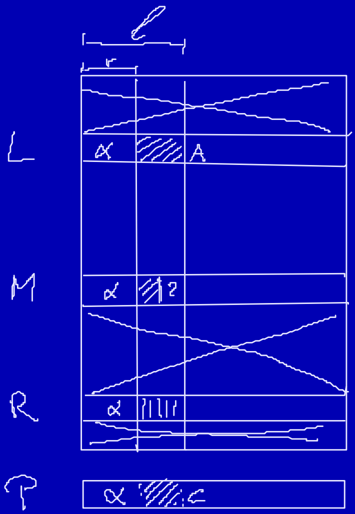
1. Caso 1: $l > r$
   - $Lcp(L, M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L, M) < l$



# Accelerante 2

**$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$**

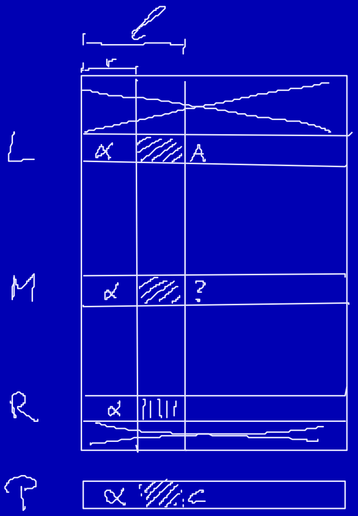
1. Caso 1: $l > r$
   - $Lcp(L, M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L, M) < l \Rightarrow$
     $R \leftarrow M, r \leftarrow Lcp(M, L)$



# Accelerante 2

### $l$: $lcp(L, P)$; $r$: $Lcp(R, P)$

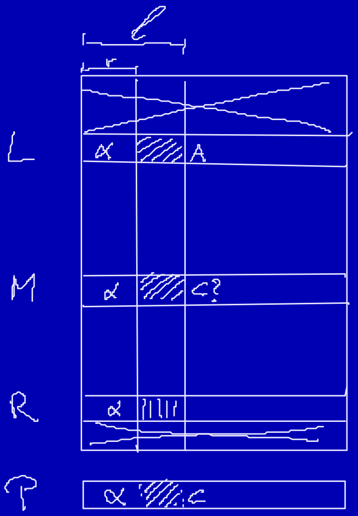
1. Caso 1: $l > r$
   - $Lcp(L, M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L, M) < l \Rightarrow$
     $R \leftarrow M, r \leftarrow Lcp(M, L)$
   - $Lcp(L, M) = l$



# Accelerante 2

**$l$: $lcp(L, P)$; $r$: $Lcp(R, P)$**

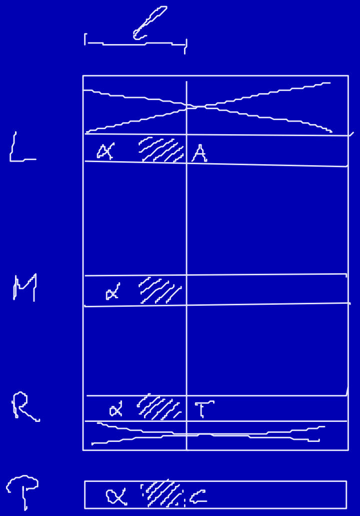
1. Caso 1: $l > r$
   - $Lcp(L, M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L, M) < l \Rightarrow$
     $R \leftarrow M, r \leftarrow Lcp(M, L)$
   - $Lcp(L, M) = l \Rightarrow$ confronto $P[l + 1 :]$, $M[l + 1 :]$



# Accelerante 2

## $l$: $lcp(L, P)$; $r$: $Lcp(R, P)$

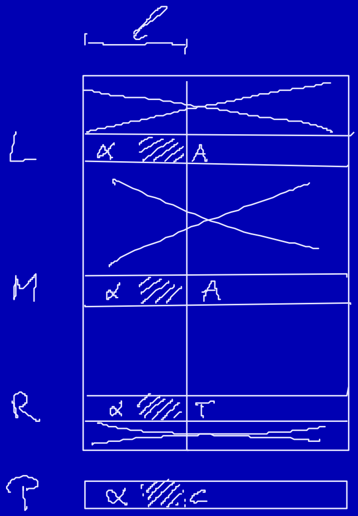
1. Caso 1: $l > r$
   - $Lcp(L, M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L, M) < l \Rightarrow$ $R \leftarrow M, r \leftarrow Lcp(M, L)$
   - $Lcp(L, M) = l \Rightarrow$ confronto $P[l + 1 :]$, $M[l + 1 :]$
2. Caso 2: $l = r$



# Accelerante 2

**$l$: $lcp(L,P)$; $r$: $Lcp(R,P)$**

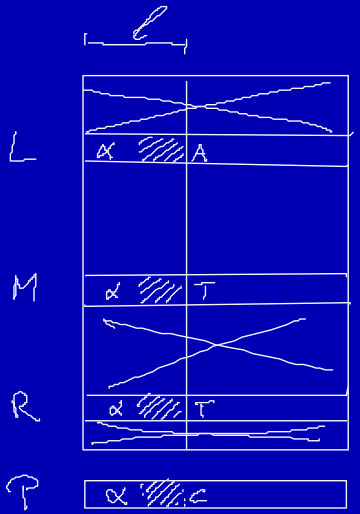
1. Caso 1: $l > r$
   - $Lcp(L,M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L,M) < l \Rightarrow$
     $R \leftarrow M, r \leftarrow Lcp(M,L)$
   - $Lcp(L,M) = l \Rightarrow$ confronto $P[l+1:]$, $M[l+1:]$
2. Caso 2: $l = r$
   - $Lcp(L,M) > l$



# Accelerante 2

**$l$: $lcp(L,P)$; $r$: $Lcp(R,P)$**

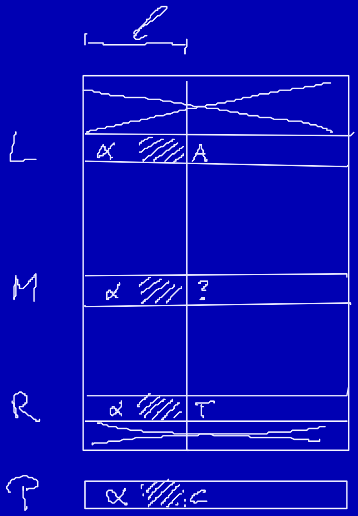
1. Caso 1: $l > r$
   - $Lcp(L,M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L,M) < l \Rightarrow$ $R \leftarrow M, r \leftarrow Lcp(M,L)$
   - $Lcp(L,M) = l \Rightarrow$ confronto $P[l+1:]$, $M[l+1:]$
2. Caso 2: $l = r$
   - $Lcp(L,M) > l$
   - $Lcp(M,R) > l$



# Accelerante 2

## $l$: $lcp(L,P)$; $r$: $Lcp(R,P)$

1. Caso 1: $l > r$
   - $Lcp(L,M) > l \Rightarrow L \leftarrow M$
   - $Lcp(L,M) < l \Rightarrow$
     $R \leftarrow M, r \leftarrow Lcp(M,L)$
   - $Lcp(L,M) = l \Rightarrow$ confronto $P[l+1:]$, $M[l+1:]$
2. Caso 2: $l = r$
   - $Lcp(L,M) > l$
   - $Lcp(M,R) > l$
   - $Lcp(L,M) = Lcp(M,R) = l$



# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L, R) = (1, n)$

# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L,R) = (1,n)$
- Iterazione 2: $(L,R) = (1,n/2)$ oppure $(n/2,n)$

# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L, R) = (1, n)$
- Iterazione 2: $(L, R) = (1, n/2)$ oppure $(n/2, n)$
- Iterazione $k$: $L = h\frac{n}{2^{k-1}}$, $R = (h+1)\frac{n}{2^{k-1}}$

# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L,R) = (1,n)$
- Iterazione 2: $(L,R) = (1,n/2)$ oppure $(n/2,n)$
- Iterazione $k$: $L = h\frac{n}{2^{k-1}}$, $R = (h+1)\frac{n}{2^{k-1}}$
- Iterazione $\lceil \log_2 n \rceil$: $R = L+1$, $Lcp(h,h+1)$

# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L,R) = (1,n)$
- Iterazione 2: $(L,R) = (1,n/2)$ oppure $(n/2,n)$
- Iterazione $k$: $L = h\frac{n}{2^{k-1}}$, $R = (h+1)\frac{n}{2^{k-1}}$
- Iterazione $\lceil \log_2 n \rceil$: $R = L+1$, $Lcp(h,h+1)$
- Iterazione $\lceil \log_2 n \rceil - 1$: aggrego i risultati dell'iterazione $\lceil \log_2 n \rceil$

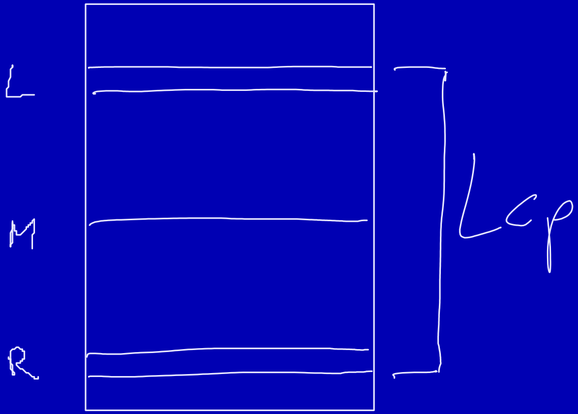
# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L,R)=(1,n)$
- Iterazione 2: $(L,R)=(1,n/2)$ oppure $(n/2,n)$
- Iterazione $k$: $L=h\frac{n}{2^{k-1}}$, $R=(h+1)\frac{n}{2^{k-1}}$
- Iterazione $\lceil\log_2 n\rceil$: $R=L+1$, $Lcp(h,h+1)$
- Iterazione $\lceil\log_2 n\rceil-1$: aggrego i risultati dell'iterazione $\lceil\log_2 n\rceil$
- Iterazione $k$: $Lcp(h\frac{n}{2^{k-1}},(h+1)\frac{n}{2^{k-1}})$

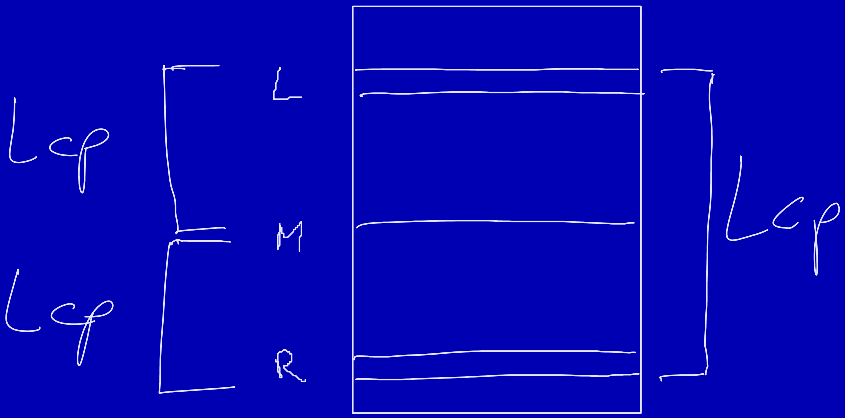
# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

- Iterazione 1: $(L, R) = (1, n)$
- Iterazione 2: $(L, R) = (1, n/2)$ oppure $(n/2, n)$
- Iterazione $k$: $L = h\frac{n}{2^{k-1}}$, $R = (h+1)\frac{n}{2^{k-1}}$
- Iterazione $\lceil \log_2 n \rceil$: $R = L + 1$, $Lcp(h, h+1)$
- Iterazione $\lceil \log_2 n \rceil - 1$: aggrego i risultati dell'iterazione $\lceil \log_2 n \rceil$
- Iterazione $k$: $Lcp(h\frac{n}{2^{k-1}}, (h+1)\frac{n}{2^{k-1}})$
- $t = \frac{n}{2^k}$, $Lcp(2ht+1, (2h+2)t)) = \min\{ Lcp(2ht+1, (2h+1)t),$ $Lcp((2h+1)t+1, (2h+2)t), Lcp((2h+1)t, (2h+1)t+1)\}$

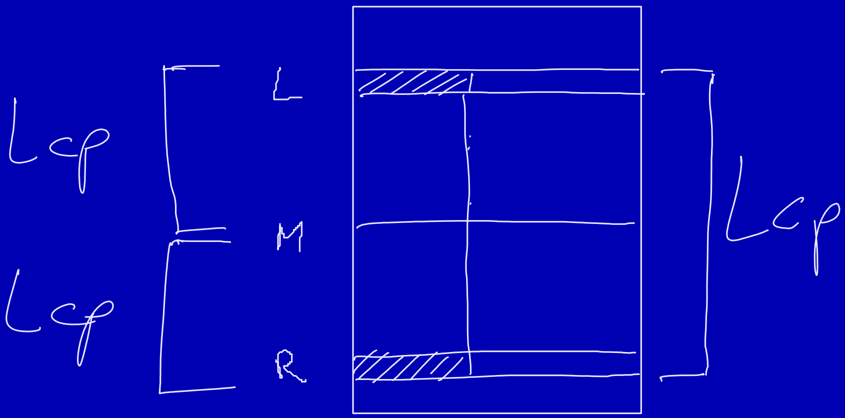
# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$



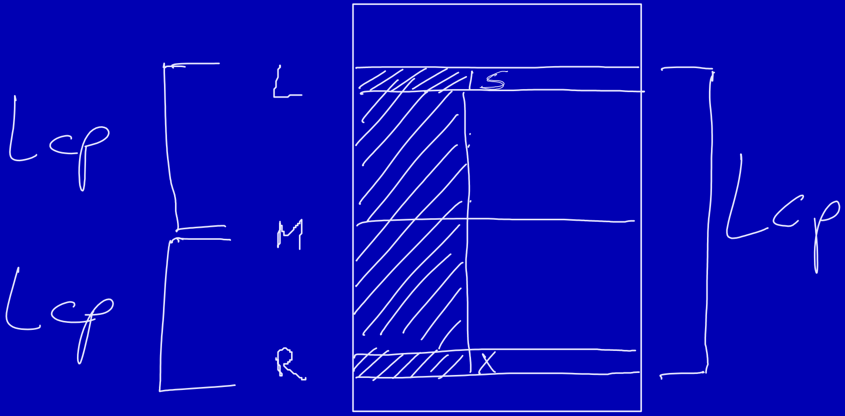
# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$



# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

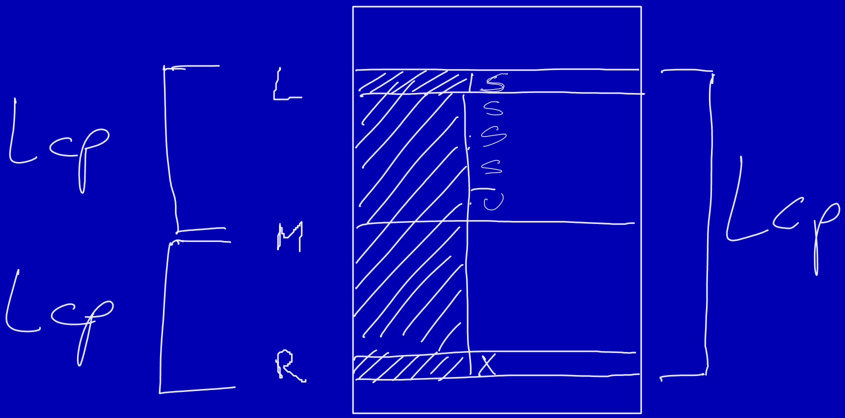


# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$



# Accelerante 3: calcolo $Lcp$ in tempo $O(n)$

Passaggio da $s$ a $z$ deve esistere



# Acceleranti 2: Osservazione

- Tempo per trovare un'occorrenza

# Acceleranti 2: Osservazione

- Tempo per trovare un'occorrenza
- Tempo per trovare tutte le occorrenze?

# Acceleranti 2: Osservazione

- Tempo per trovare un'occorrenza
- Tempo per trovare tutte le occorrenze?
- $O(n+m+k)$, per $k$ occorrenze

# Costruzione suffix array: nuovo alfabeto

- Alfabeto $\Sigma$ con $\sigma$ simboli, testo $T$ lungo $n$
- Aggrego triple di caratteri
- Alfabeto $\Sigma^3$ con $\sigma^3$ simboli, testo lungo $n/3$
- $T_1 = (T[1], T[2], T[3]) \cdots (T[3i+1], T[3i+2], T[3i+3]) \cdots$
  $T_2 = (T[2], T[3], T[4]) \cdots (T[3i+2], T[3i+3], T[3i+4]) \cdots$
  $T_0 = (T[3], T[4], T[5]) \cdots (T[3i], T[3i+1], T[3i+2]) \cdots$
- suffissi$(T) \Leftrightarrow \bigcup_{i=0,1,2}$ suffissi$(T_i)$

# Costruzione suffix array: ricorsione

1. Ricorsione su $T_0T_1$
2. suffissi$(T_0T_1) \iff$ suffissi$(T_0)$, suffissi$(T_1)$
3. suffissi$(T_0T_1) \iff$ suffissi$(T_2)$
4. $T_2[i:] \approx T[3i+2:]$
5. $T[3i+2:] = T[3i+2]T[3i+3:] = T[3i+2]T_0[i+1:]$
6. suffissi$(T_0)$ ordinati
7. Radix sort
8. Fusione suffissi$(T_0T_1)$, suffissi$(T_2)$

# Costruzione suffix array: fusione

Confronto suffisso di $T_0$ e $T_2$

1. $T_0[i:] <=> T_2[j:]$
2. $T[3i:] <=> T[3j+2:]$
3. $T[3i]T[3i+1:] <=> T[3j+2]T[3j+3:]$
4. $T[3i]T_1[i:] <=> T[3j+2]T_0[j+1:]$

# Costruzione suffix array: fusione

Confronto suffisso di $T_1$ e $T_2$

1. $T_1[i:] <=> T_2[j:]$
2. $T[3i+1:] <=> T[3j+2:]$
3. $T[3i+1]T[3i+2:] <=> T[3j+2]T[3j+3:]$
4. $T[3i+1]T[3i+2]T[3i+3:] <=> T[3j+2]T[3j+3]T[3j+4:]$
5. $T[3i+1]T[3i+2]T_0[i+1:] <=> T[3j+2]T[3j+3]T_1[j+1:]$

# KS


1. Juha Kärkkäinen, Peter Sanders and Stefan Burkhardt. Linear work suffix array construction. J. ACM, 53 (6), 2006, pp. 918-936.
2. Difference cover (DC) 3
3. Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking In Proc. 14th Symposium on Combinatorial Pattern Matching (CPM '03), LNCS 2676, Springer, 2003, pp. 55-69. http://www.stefan-burkhardt.net/CODE/cpm_03.tar.gz
4. Yuta Mori. SAIS https://sites.google.com/site/yuta256/

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato
2. Vettore $C_x[1:k]$ per ogni nodo $x$

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato
2. Vettore $C_x[1 : k]$ per ogni nodo $x$
3. $C_x[i]$: sottoalbero con radice $x$ ha una foglia di $s_i$

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato
2. Vettore $C_x[1:k]$ per ogni nodo $x$
3. $C_x[i]$: sottoalbero con radice $x$ ha una foglia di $s_i$
4. $C_x = \bigvee C$ sui figli di $C$

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato
2. Vettore $C_x[1:k]$ per ogni nodo $x$
3. $C_x[i]$: sottoalbero con radice $x$ ha una foglia di $s_i$
4. $C_x = \bigvee C$ sui figli di $C$
5. Nodo $z$, $C_z =$ tutti 1

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato
2. Vettore $C_x[1:k]$ per ogni nodo $x$
3. $C_x[i]$: sottoalbero con radice $x$ ha una foglia di $s_i$
4. $C_x = \bigvee C$ sui figli di $C$
5. Nodo $z$, $C_z =$ tutti 1
6. Tempo $O(kn)$

# Sottostringa comune più lunga

## $k$ stringhe $\{s_1, \ldots, s_k\}$

1. Suffix tree generalizzato
2. Vettore $C_x[1 : k]$ per ogni nodo $x$
3. $C_x[i]$: sottoalbero con radice $x$ ha una foglia di $s_i$
4. $C_x = \bigvee C$ sui figli di $C$
5. Nodo $z$, $C_z =$ tutti 1
6. Tempo $O(kn)$
7. $n$: summa lunghezze $|s_1| + \cdots + |s_k|$

# Lowest common ancestor (lca)

## Dati albero $T$ e 2 foglie $x$, $y$

- $z$ è antenato comune di $x$, $y$ se $z$ è antenato di entrambi $x$ e $y$
- $z$ è lca di $x$, $y$ se:
  1. $z$ è antenato comune di $x$ e $y$
  2. nessun discendente di $z$ è antenato comune di $x$ e $y$

## Proprietà

- Preprocessing di $T$ in tempo $O(n)$
- Calcolo lca$(x, y)$ in tempo $O(1)$
- Algoritmo complesso, ma pratico

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia
3. $N_x[i] =$ somma dei figli

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia
3. $N_x[i] =$ somma dei figli
4. $D_x[i]$: numero di consecutive di foglie di $s_i$, ordinate secondo visita depth-first, discendenti di $x$

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia
3. $N_x[i] =$ somma dei figli
4. $D_x[i]$: numero di consecutive di foglie di $s_i$, ordinate secondo visita depth-first, discendenti di $x$
5. $N_x[i] = 0 \Rightarrow D_x[i] = 0$

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia
3. $N_x[i] =$ somma dei figli
4. $D_x[i]$: numero di consecutive di foglie di $s_i$, ordinate secondo visita depth-first, discendenti di $x$
5. $N_x[i] = 0 \Rightarrow D_x[i] = 0$
6. $N_x[i] = 1 \Rightarrow D_x[i] = 0$

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia
3. $N_x[i] =$ somma dei figli
4. $D_x[i]$: numero di consecutive di foglie di $s_i$, ordinate secondo visita depth-first, discendenti di $x$
5. $N_x[i] = 0 \Rightarrow D_x[i] = 0$
6. $N_x[i] = 1 \Rightarrow D_x[i] = 0$
7. $N_x[i] \geq 1 \Rightarrow D_x[i] = N_x[i] - 1$

# Sottostringa comune più lunga di $k$ stringhe

## Arricchimento $ST$

1. $N_x[i]$: numero foglie di $s_i$ discendenti di $x$
2. $N_x[i] = 0$ o $1$ per ogni foglia
3. $N_x[i] =$ somma dei figli
4. $D_x[i]$: numero di consecutive di foglie di $s_i$, ordinate secondo visita depth-first, discendenti di $x$
5. $N_x[i] = 0 \Rightarrow D_x[i] = 0$
6. $N_x[i] = 1 \Rightarrow D_x[i] = 0$
7. $N_x[i] \geq 1 \Rightarrow D_x[i] = N_x[i] - 1$
8. $N_x[i] - D_x[i] = C_x[i]$

# Sottostringa comune più lunga di $k$ stringhe

## Gestione $ST$

- Visita depth-first di $ST$
- $L_i$: lista ordinata delle foglie di $s_i$
- Per ogni coppia $x, y$ consecutiva in $L_i$
  1. $z \leftarrow lca(x, y)$
  2. $D_z[i] =$
  3. Aggiorna $C_z$

# Licenza d'uso