**Il pensiero computazionale**

Percorso Formativo per i Docenti della Scuola Secondaria di Secondo Grado – Università di Pisa

# Laboratorio #4 : Motori di Ricerca

(progettazione e coding)

**Prof. Paolo Ferragina e Davide Neri**

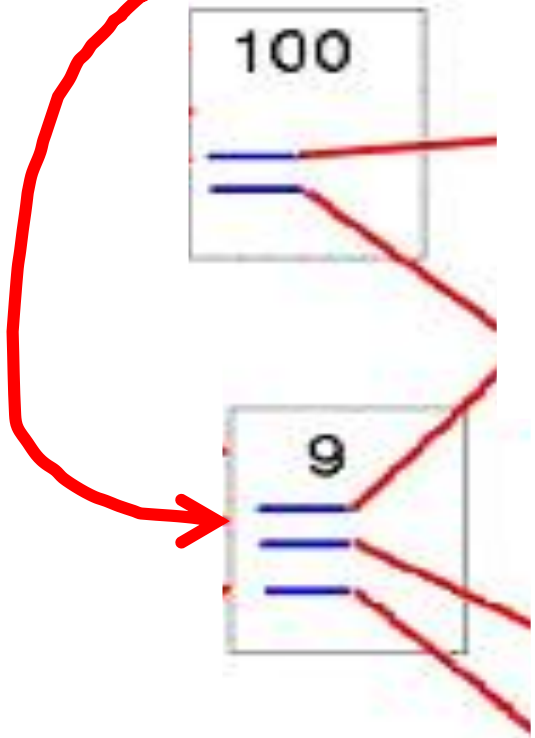
**11 marzo 2019, ore 14-18**

# Sommario

- PageRank e varianti
- Suffix Array
- Text mining

# PageRank e varianti

# Il (classico) PageRank

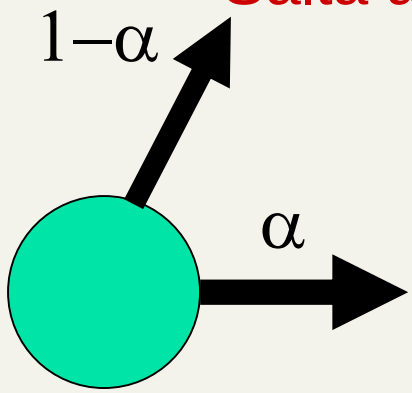



Se lasciamo propagare i valori, questi si "stabilizzano" ?

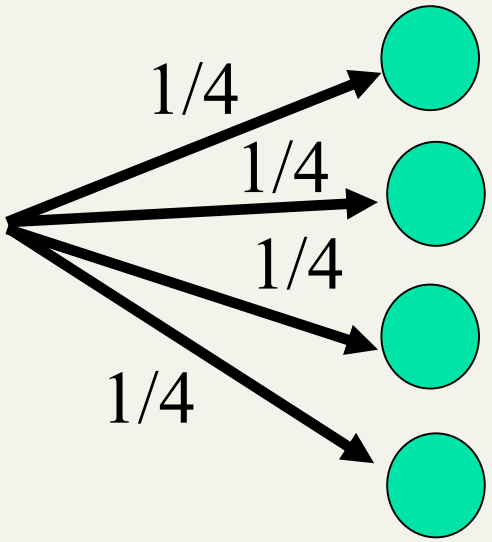
- Sistema lineare con miliardi di vincoli e variabili, efficienza?
- Varie interpretazioni: Cammini casuali, Catene di Markov

# PageRank, come Cammino Casuale sul grafo

Salta a caso su una pagina qualunque del Web




Salta a caso su uno dei vicini

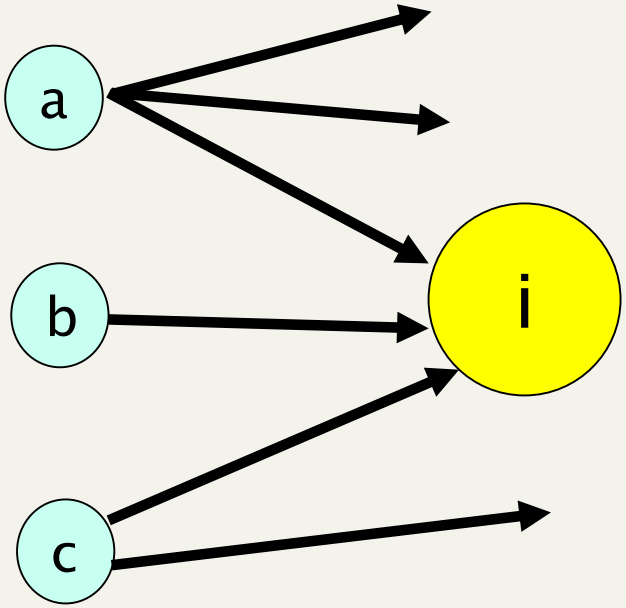



Il PageRank di un nodo è la «frequenza» con cui si visiterebbe quel nodo muovendosi per sempre

Sorta di «misura di centralità» di un nodo nel grafo

# PageRank, come Sistema Lineare

$$r(i) = \alpha \cdot \sum_{j \in B(i)} \frac{r(j)}{\#out(j)} + (1-\alpha) \cdot \frac{1}{N}$$



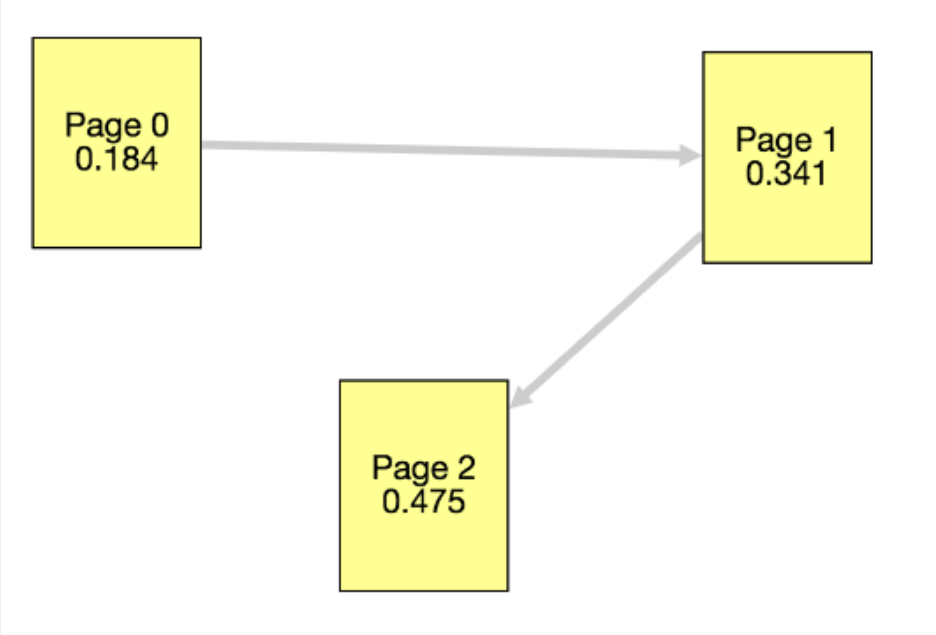

$\alpha = 0.85$

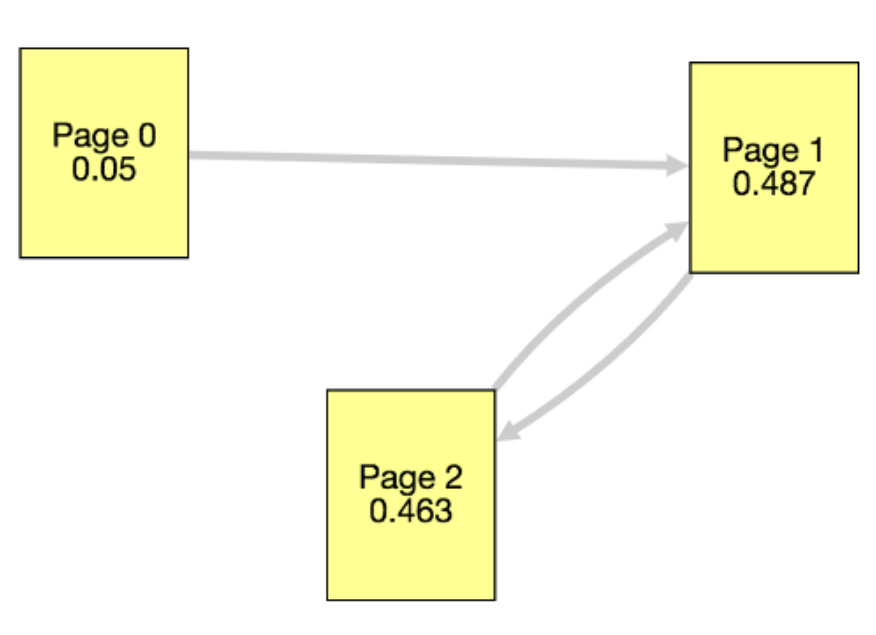
**N** = # nodi grafo

**r(i)** = $\alpha$ (r(a) / 3 +
r(b) / 1 +
r(c) / 2 )
+ (1 - $\alpha$) N

Il PageRank «legato» agli autovettori della matrice del sistema lineare

# Test sul web

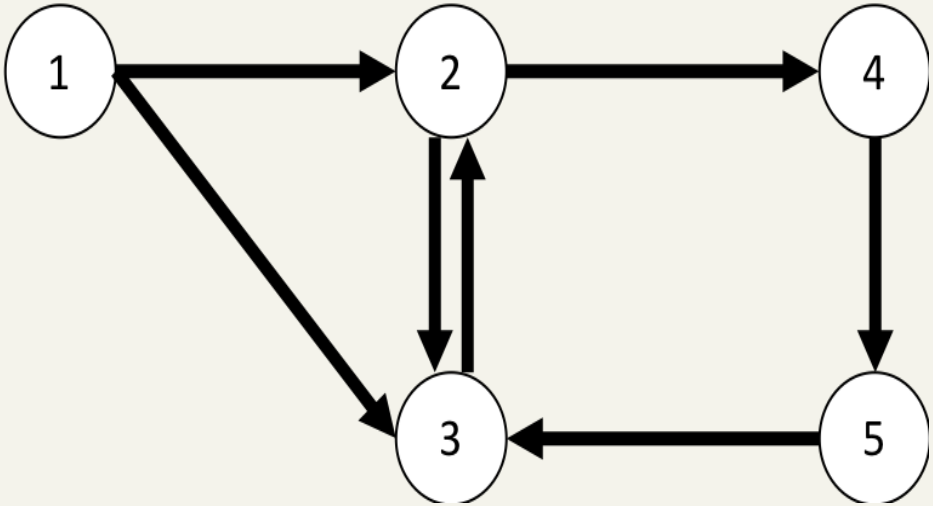



http://bit.ly/2iwHH3e

http://faculty.chemeketa.edu/ascholer/cs160/WebApps/PageRank/

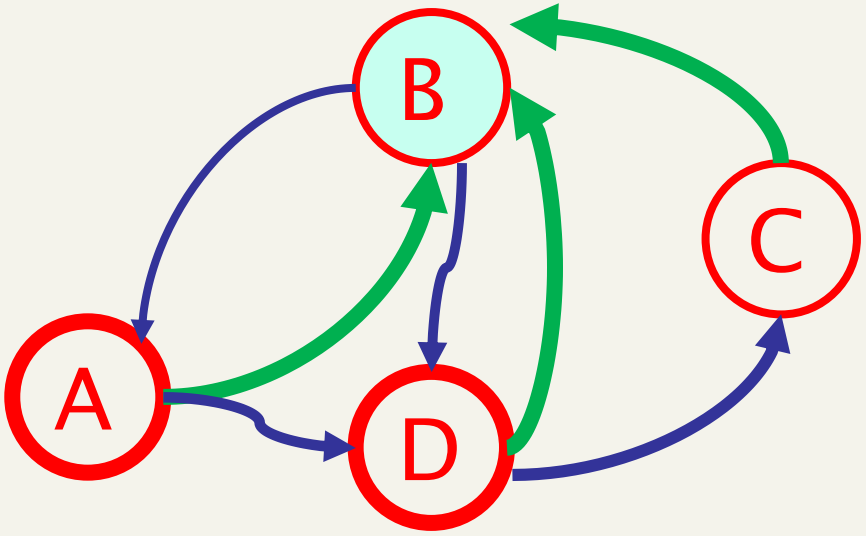
# Un esercizio




- Eseguire due passi di PageRank, assumendo teleportation step uniforme, alpha = ½ e vettore iniziale di probabilità uniforme.

# **Personalized** Pagerank

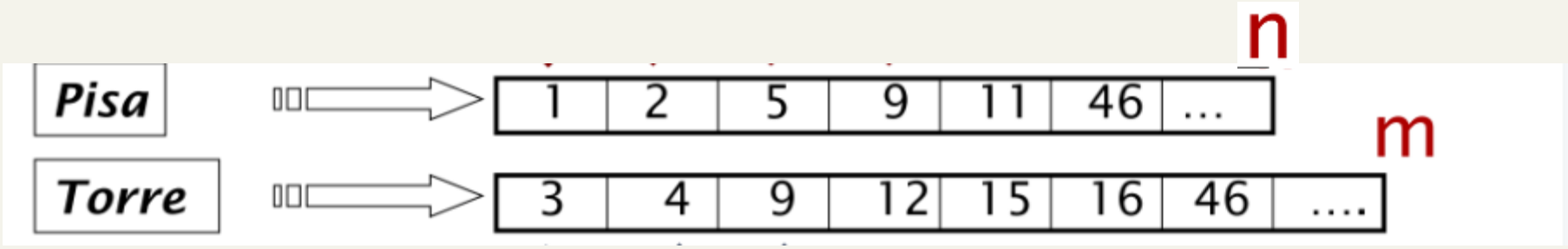
Il teleportation step non viene eseguito saltando **uniformemente** a qualunque nodo del grafo ma a un sottoinsieme di **nodi preferiti**




Teleport step al nodo **B**

# Suffix Array

# Il cuore dei motori di ricerca



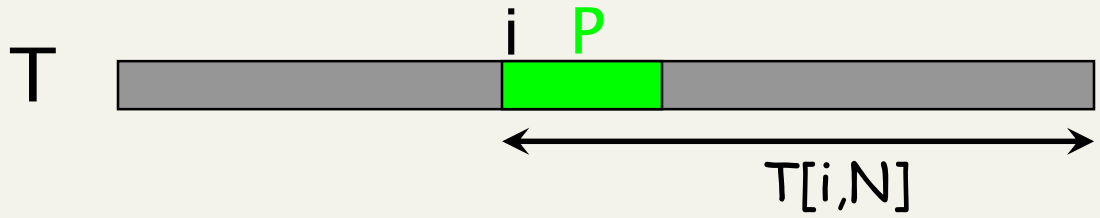

Occorre definire un **concetto di parola** (token) su cui è possibile fare le ricerche.

Come cercare **sottostringhe** di composizione e lunghezza arbitraria ?

# Alcuni fatti di base

Pattern P occorre in posizione i del testo T

iff P è prefisso dell' i-mo suffisso di T (ie. T[i,N])




Le occurrenze di P in T = Tutti i suffissi di T con P prefisso

P = si

T = mississippi
mississippi

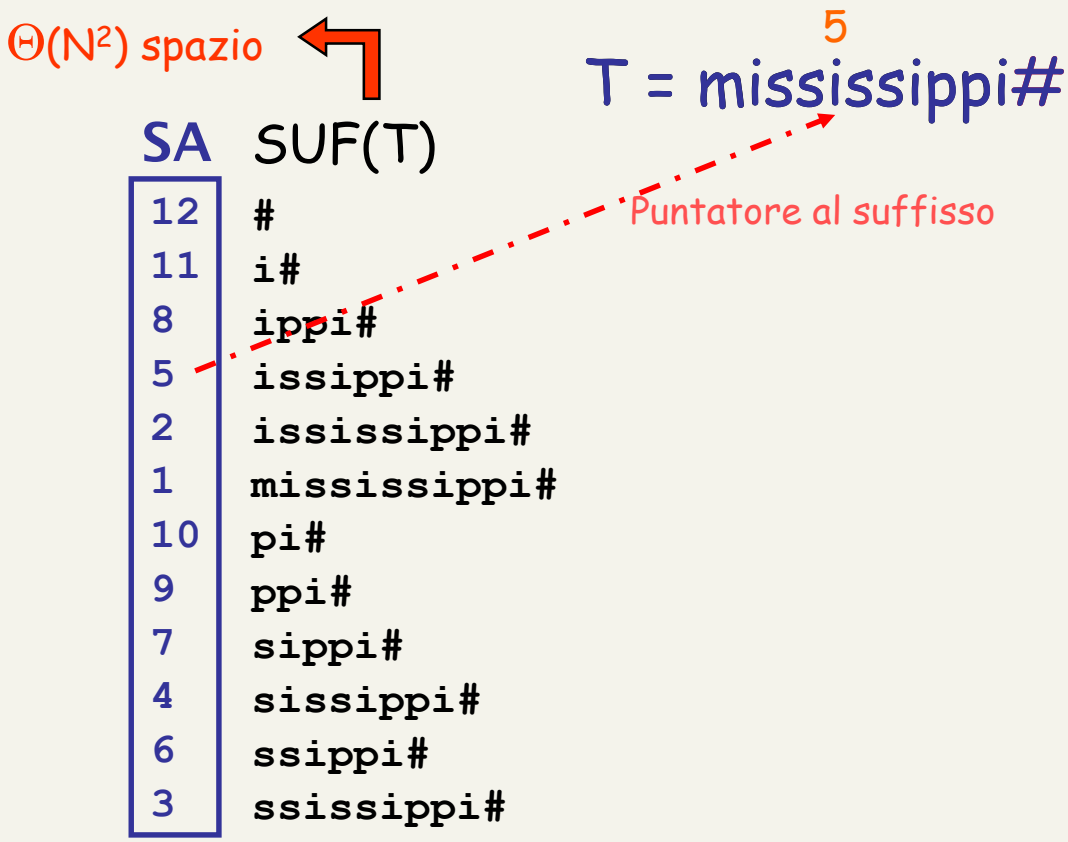
4,7

Riduzione

SUF(T) = Insieme ordinato dei suffissi

Da ricerca per sottostringa
A ricerca per prefisso

# Suffix Array

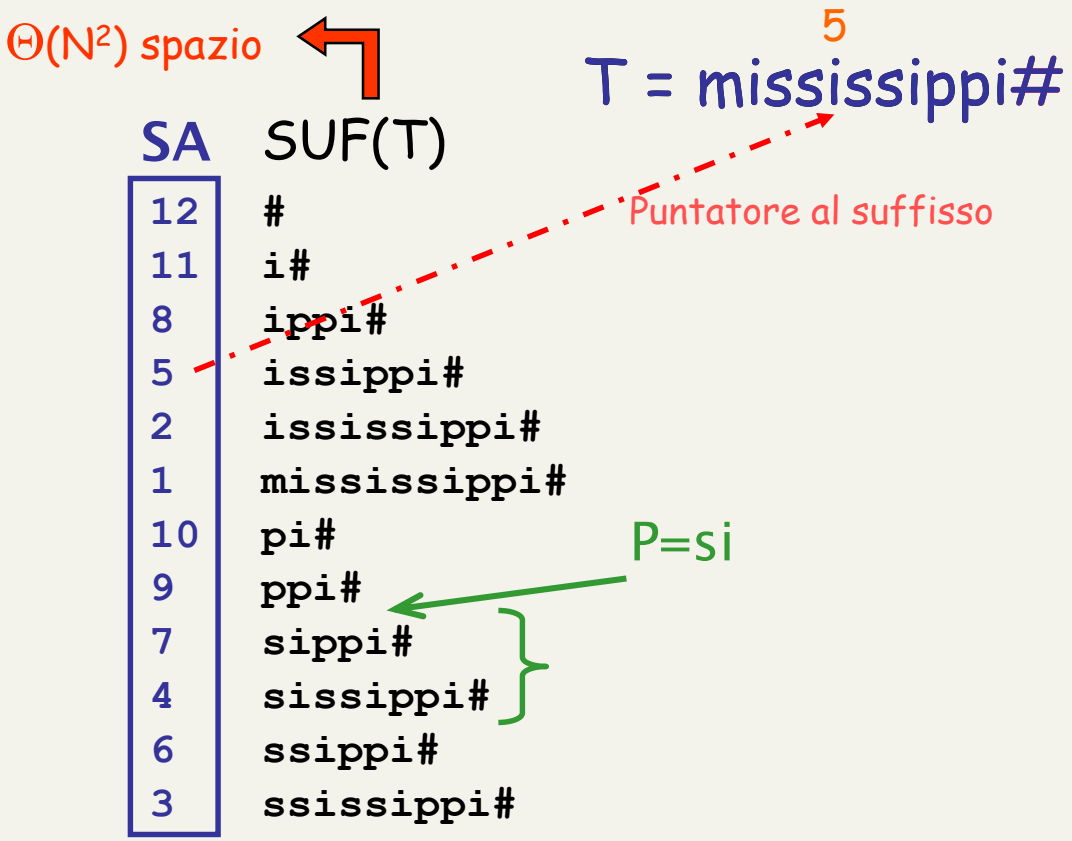



| SA | SUF(T) |
|---|---|
| 12 | # |
| 11 | i# |
| 8 | ippi# |
| 5 | issippi# |
| 2 | ississippi# |
| 1 | mississippi# |
| 10 | pi# |
| 9 | ppi# |
| 7 | sippi# |
| 4 | sissippi# |
| 6 | ssippi# |
| 3 | ssissippi# |

# Suffix Array

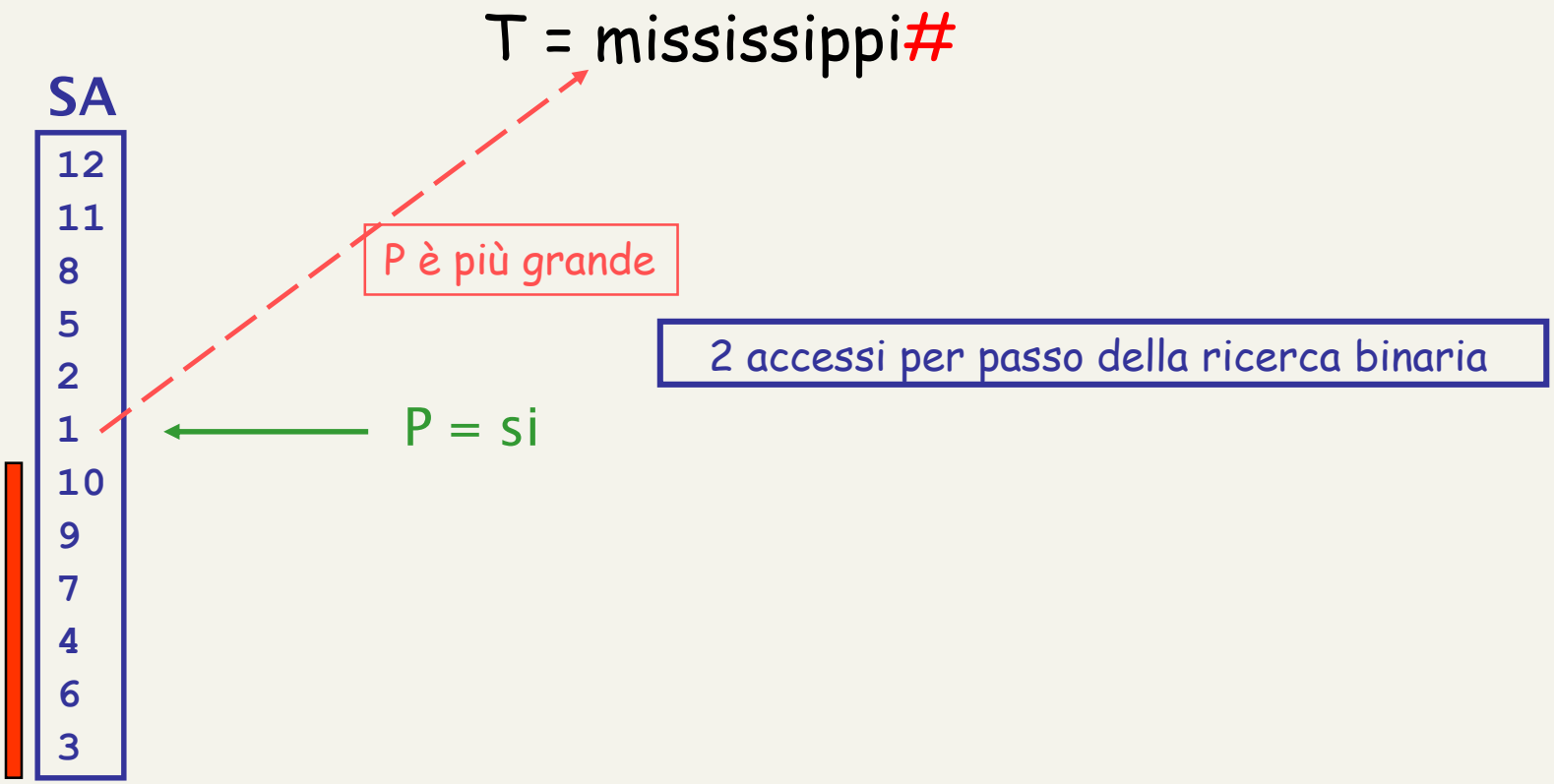
Prop 1. Tutti i suffissi con prefisso P sono contigui in SUF(T)

Prop 2. La posizione è quella lessicografica di P in SUF(T).



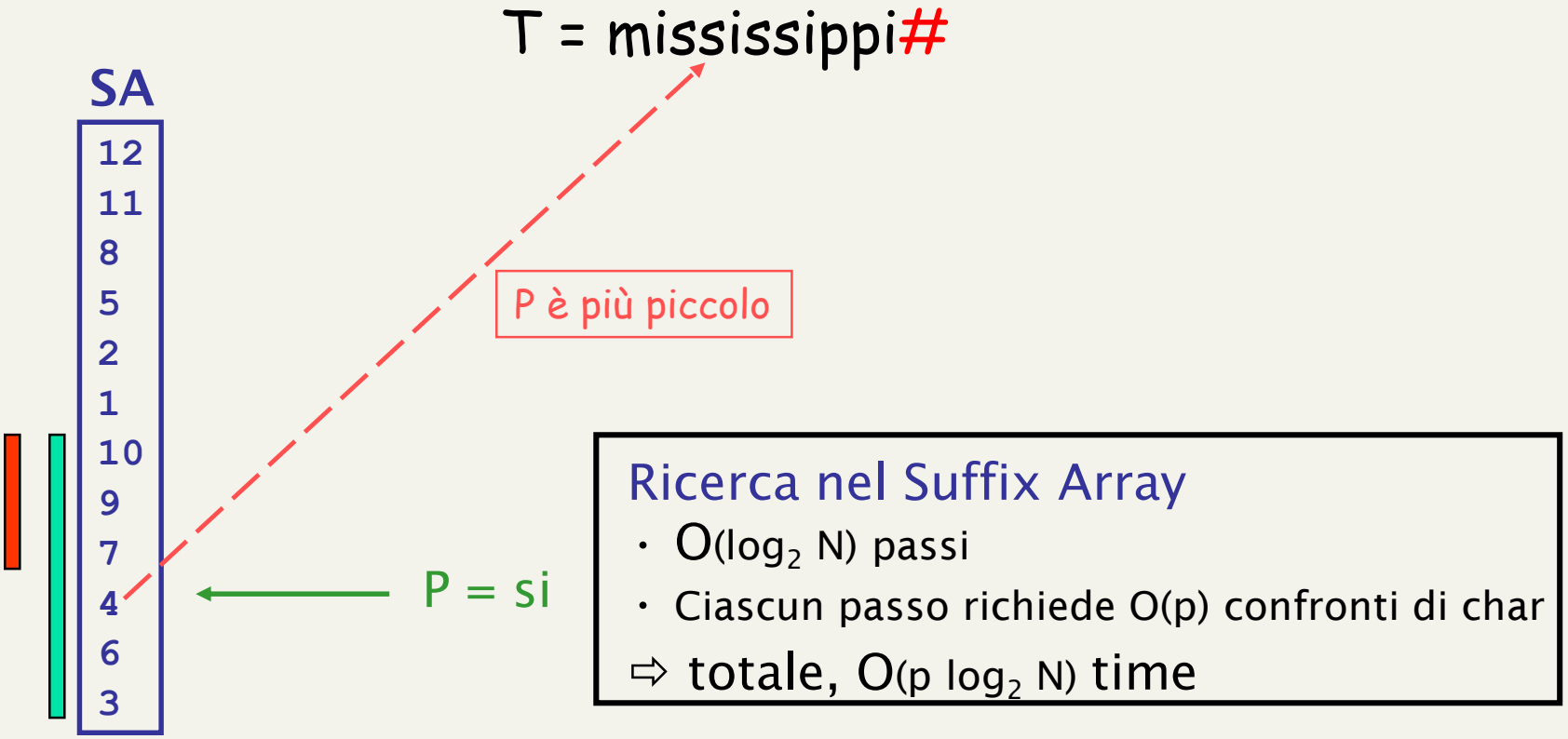
# Cercare un pattern

Ricerca binaria «indiretta» su SA: O(p) time per confronto



# Cercare un pattern

Ricerca binaria «indiretta» su *SA*: *O*(p) time per confronto

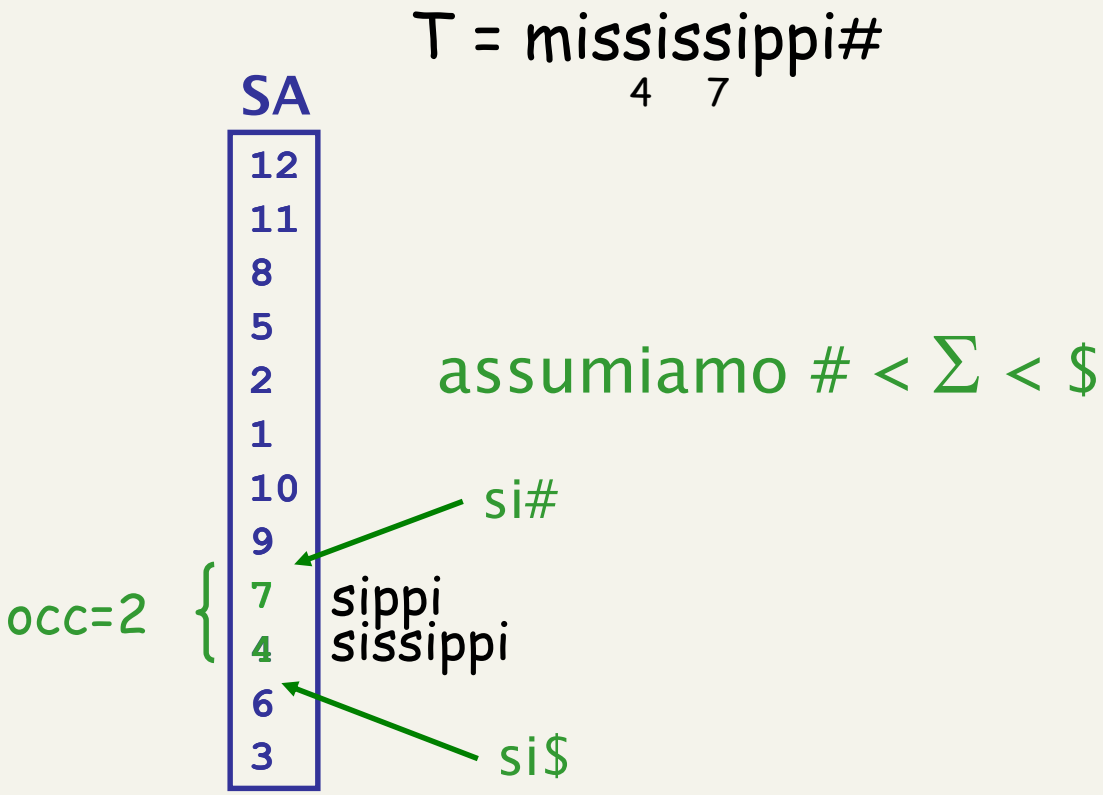



**Ricerca nel Suffix Array**

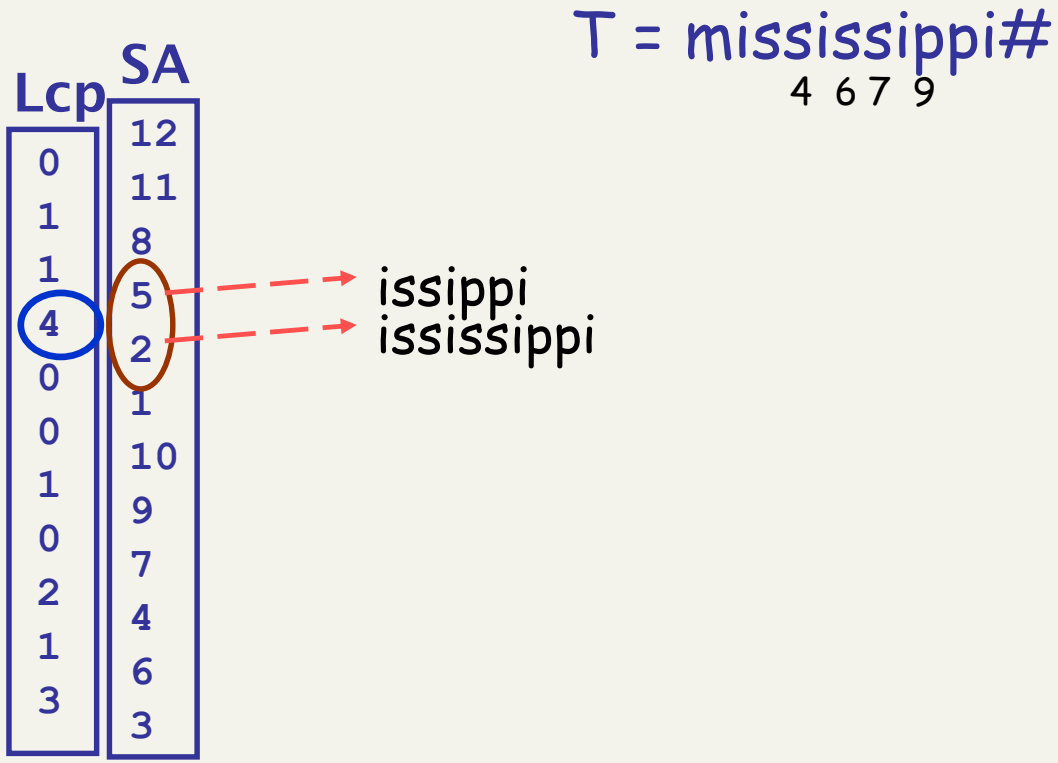
- $O(\log_2 N)$ passi
- Ciascun passo richiede O(p) confronti di char

⇨ totale, $O(p \log_2 N)$ time

# Occorrenze del pattern

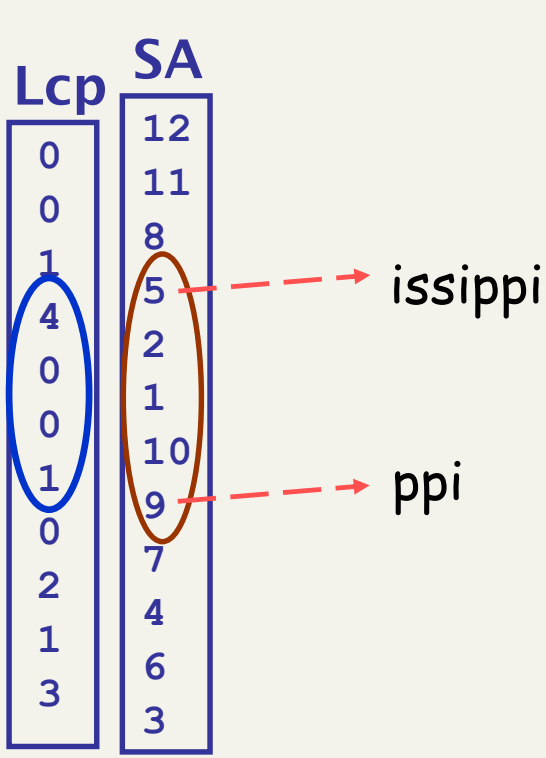


# Text mining



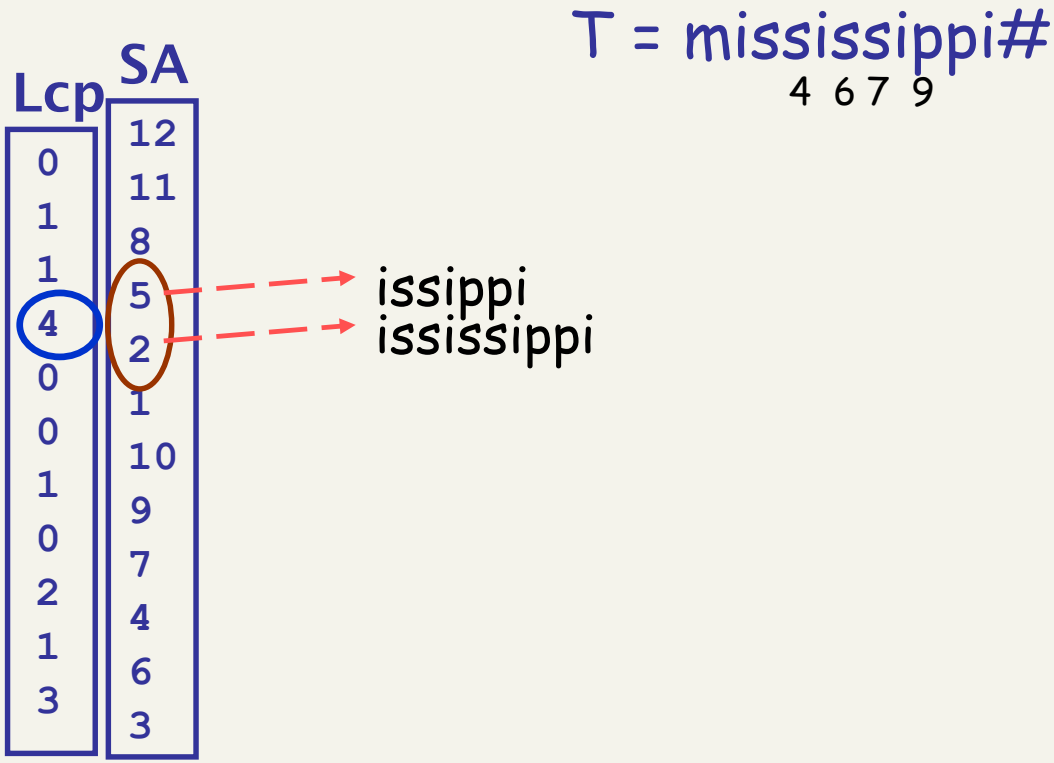

- Array Lcp[h] = lunghezza del prefisso condiviso tra SA[h] e SA[h+1].

# Problema #1



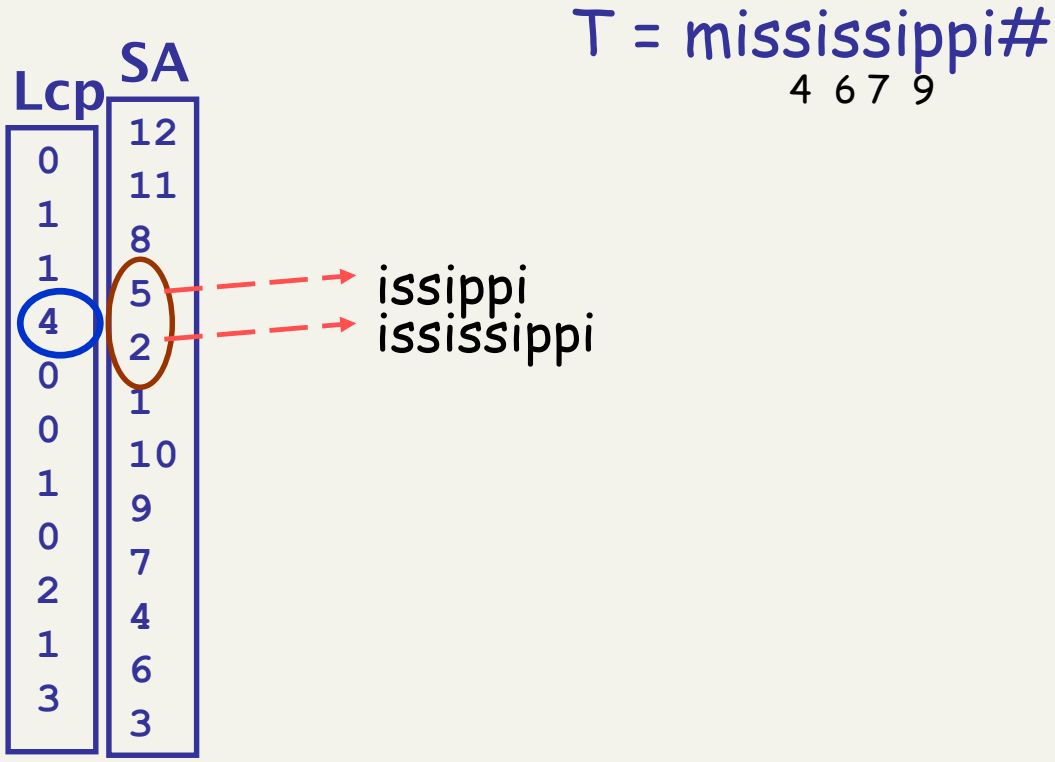

- Quanto è lungo il prefisso comune tra T[i,...] e T[j,...] ?
  - Min del subarray Lcp[h,k] s.t. SA[h]=i e SA[k]=j.

# Problema #2




- Esiste una sottostringa ripetuta di lunghezza ≥ L ? [Ad esempio L=4?]
  - Cerca se esiste «i» per cui Lcp[i] ≥ L

# Problema #3




- Esiste una sottostringa ripetuta di len ≥ L e che occorre ≥ C volte ?
  - Esempio: L = 2 e C=2 ? Oppure, L=2 e C=3 ?
  - Cercare indice «i» t.c. Lcp[i,i+C-2] ha valori ≥ L

# Problema #4: Plagio tra due testi

Dati due file $F_1$ ed $F_2$

- Trovare la più lunga sottostringa condivisa
- Trovare le sottostringhe condivise di lunghezza > L
- Trovare le sottostringhe massimali condivise di lunghezza > L