Elementi di Bioinformatica

Gianluca Della Vedova

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

30 ottobre 2023

Pattern matching su suffix array

Occorrenza P in T

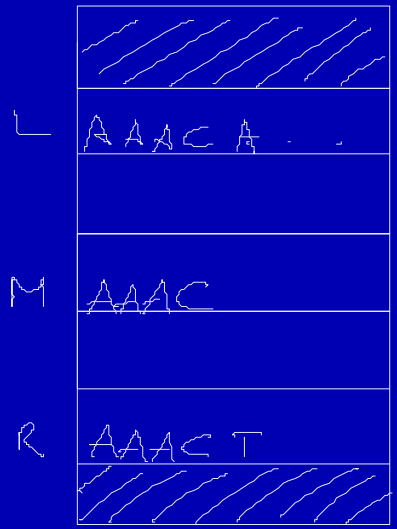
Suffissi di T che iniziano con P

Ricerca in SA

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

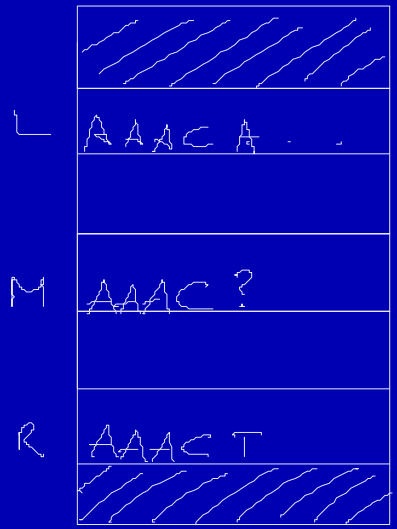
Accelerante 1

- 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



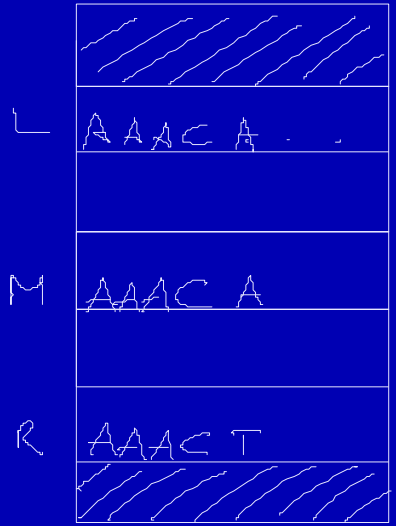
Accelerante 1

- 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



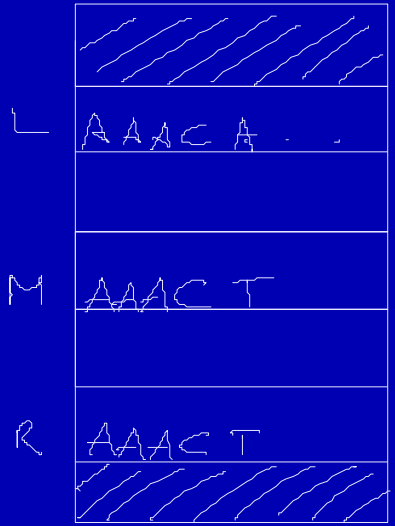
Accelerante 1

- 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



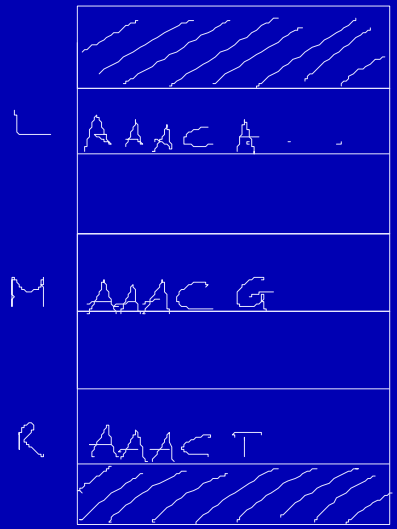
Accelerante 1

- 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

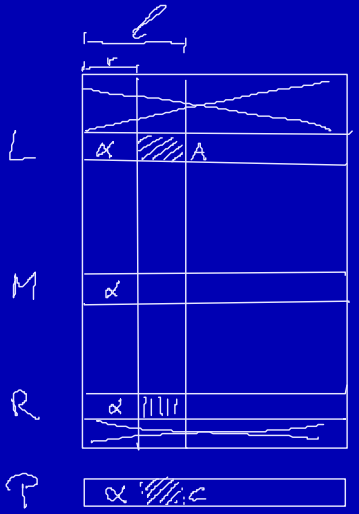


Accelerante 1

- 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



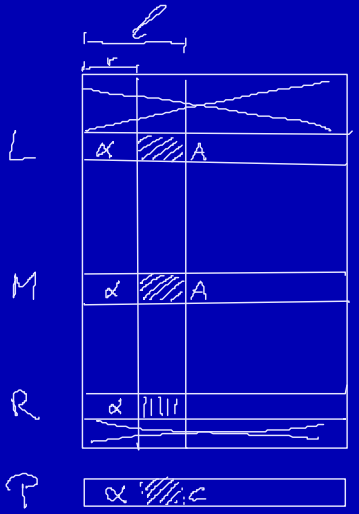
Accelerante 2



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

1. Caso 1: l > r

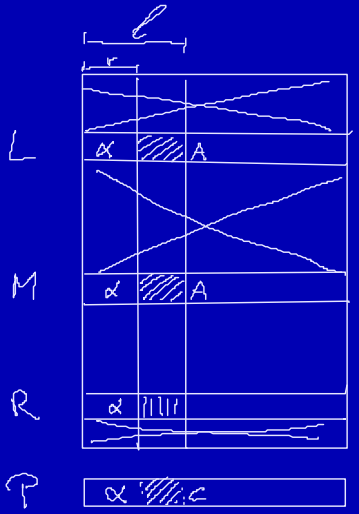
Accelerante 2



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

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

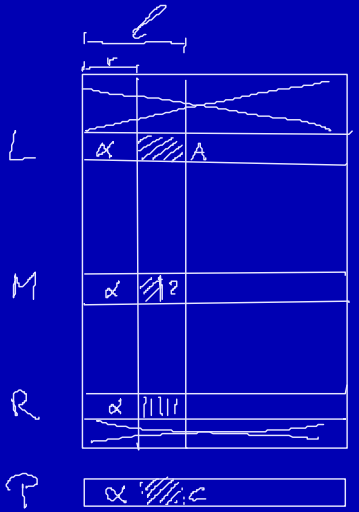
Accelerante 2



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

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

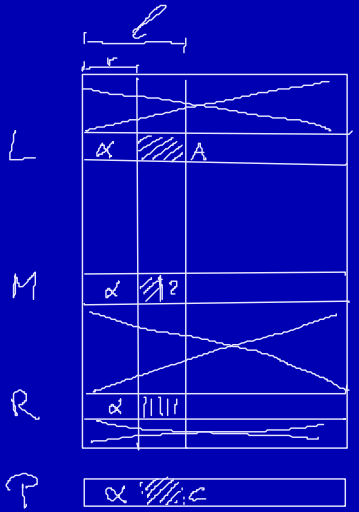
Accelerante 2



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

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

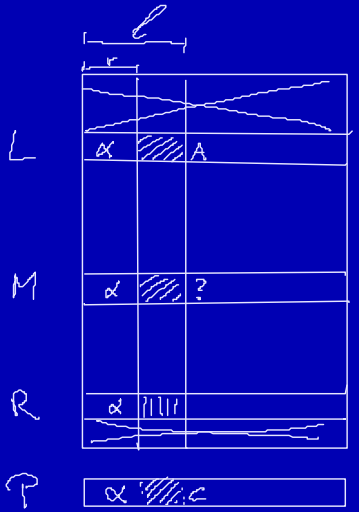
Accelerante 2



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

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

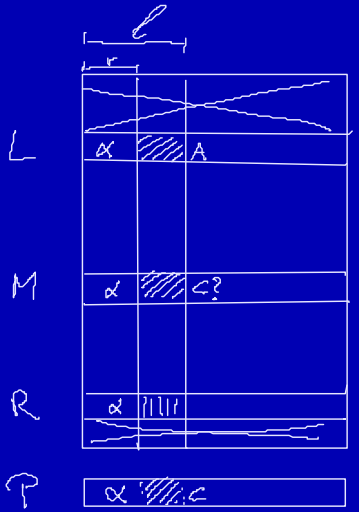
Accelerante 2



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

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

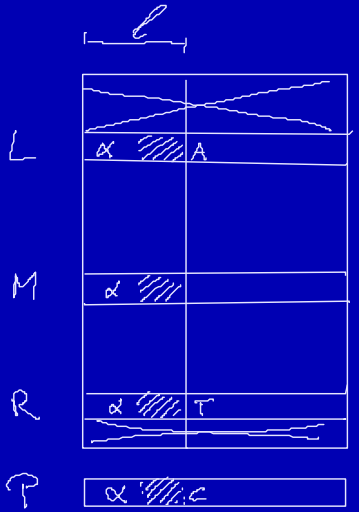
Accelerante 2



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 $P[l + 1 :]$, $M[l + 1 :]$

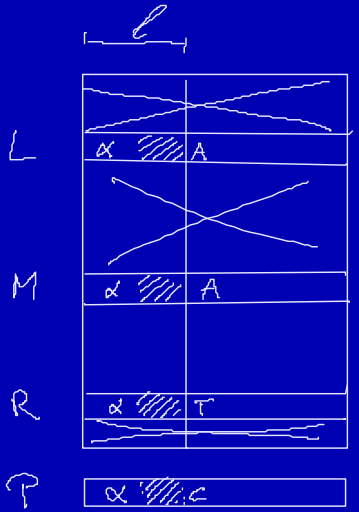
Accelerante 2



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: $l = r$

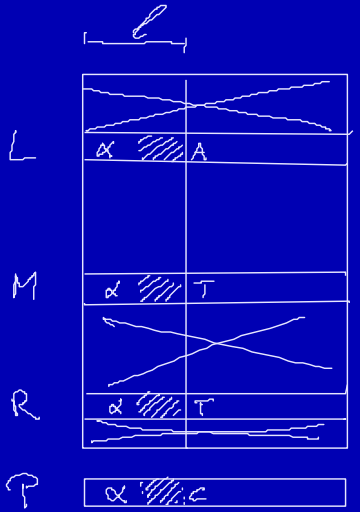
Accelerante 2



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: $l = r$
   - $Lcp(L, M) > l$

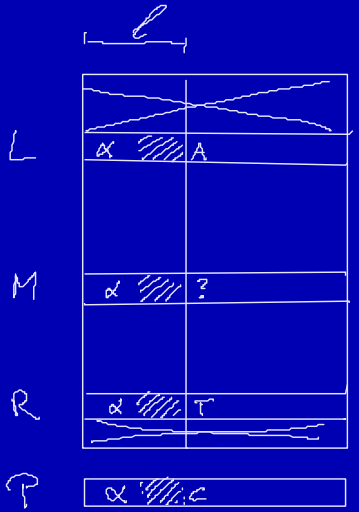
Accelerante 2



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 $\lceil \log_2 n \rceil$: $R = L + 1$, $\mathrm{Lcp}(h, h+1)$

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 $\lceil \log_2 n \rceil$: $R = L + 1$, $\mathrm{Lcp}(h, h+1)$
- Iterazione $\lceil \log_2 n \rceil - 1$: aggrego i risultati dell'iterazione $\lceil \log_2 n \rceil$

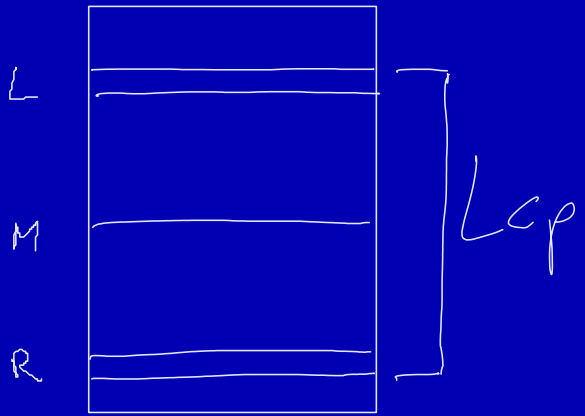
Accelerante 3: calcolo Lcp in tempo $O(n)$

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

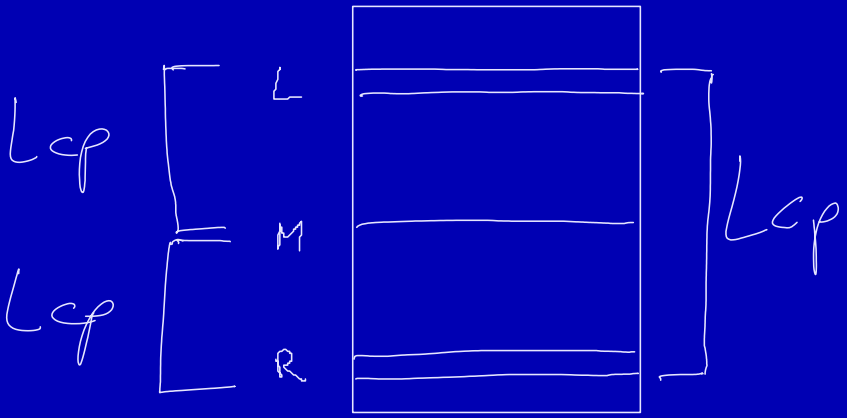
Accelerante 3: calcolo Lcp in tempo $O(n)$

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

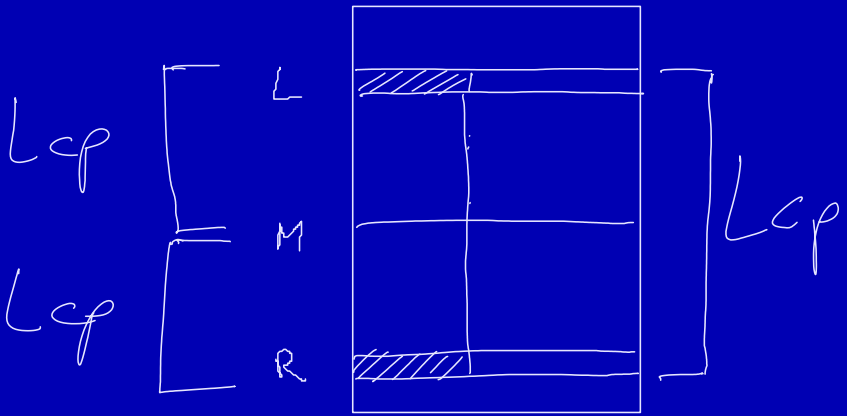
Accelerante 3: calcolo Lcp in tempo O(n)



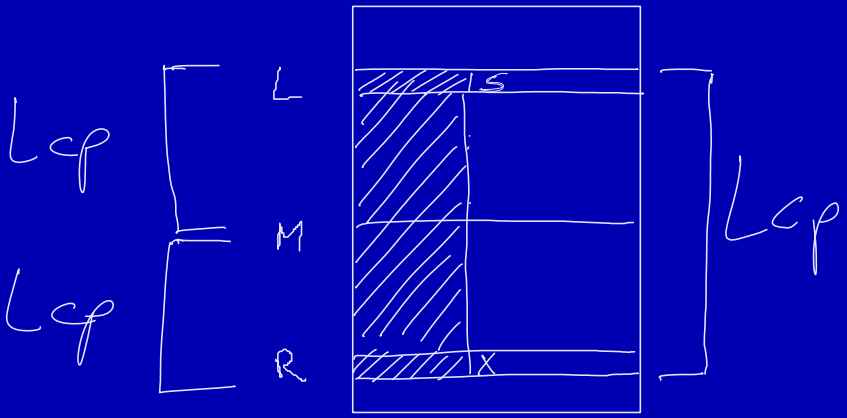
Accelerante 3: calcolo Lcp in tempo O(n)



Accelerante 3: calcolo Lcp in tempo O(n)

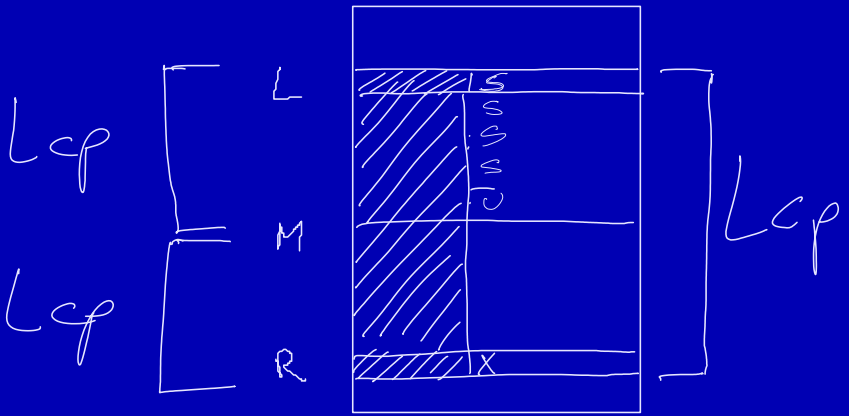


Accelerante 3: calcolo Lcp in tempo O(n)



Accelerante 3: calcolo Lcp in tempo O(n)

Passaggio da s a z deve esistere



Osservazione

- Tempo per trovare un'occorrenza

Osservazione

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

Osservazione

- Tempo per trovare un'occorrenza
- Tempo per trovare tutte le occorrenze?
- $O(n + m + k)$, per k occorrenze | scansione lineare a partire dall'occorrenza trovata

Costruzione suffix array: nuovo alfabeto

- Alfabeto Σ con σ simboli, testo T lungo n
- Aggrego triple di caratteri
- Alfabeto Σ^3 con σ^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

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)

Costruzione suffix array: ricorsione

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_2)

Costruzione suffix array: ricorsione

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_2)
1. $T_2[i:] \approx T[3i+2:]$

Costruzione suffix array: ricorsione

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_2)

1. $T_2[i:] \approx T[3i+2:]$
2. $T[3i+2:] = T[3i+2]T[3i+3:] = T[3i+2]T_0[i+1:]$

Costruzione suffix array: ricorsione

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_2)

1. $T_2[i:] \approx T[3i+2:]$
2. $T[3i+2:] = T[3i+2]T[3i+3:] = T[3i+2]T_0[i+1:]$
3. suffissi(T_0) ordinati

Costruzione suffix array: ricorsione

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_2)

1. $T_2[i:] \approx T[3i+2:]$
2. $T[3i+2:] = T[3i+2]T[3i+3:] = T[3i+2]T_0[i+1:]$
3. suffissi(T_0) ordinati
4. Radix sort

Costruzione suffix array: ricorsione

- Ricorsione su T_0T_1
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_0), suffissi(T_1)
- suffissi(T_0T_1) $\Leftrightarrow$ suffissi(T_2)

1. $T_2[i:] \approx T[3i+2:]$
2. $T[3i+2:] = T[3i+2]T[3i+3:] = T[3i+2]T_0[i+1:]$
3. suffissi(T_0) ordinati
4. Radix sort
5. 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/`

Licenza d'uso