

Elementi di Bioinformatica

Gianluca Della Vedova

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

10 ottobre 2019

Occorrenza P in T

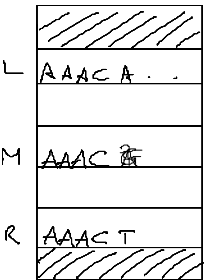
Suffissi di T che iniziano con P

Ricerca in SA

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

Accelerante 1

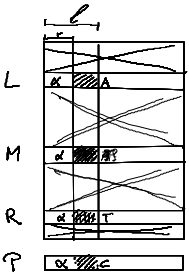
- Intervallo $SA(L, R)$ di SA
- Elemento mediano M
- Tutti i suffissi in $SA(L, R)$ iniziano con uno stesso prefisso lungo $Lcp(SA[L], SA[R])$
- Non confrontare con i primi $Lcp(SA[L], SA[R])$ caratteri



Accelerante 2

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

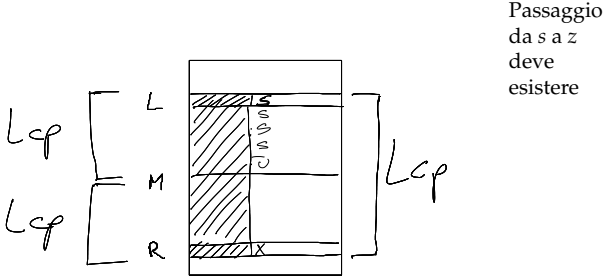
- ① Caso 1: $l > r$
 $Lcp(L, M) > l \Rightarrow L \leftarrow M$
 $Lcp(L, M) < l \Rightarrow$
 $R \leftarrow M, r \leftarrow Lcp(M, L)$
 $Lcp(L, M) = l \Rightarrow$ confronto
 $P[l + 1 :], M[l + 1 :]$
- ② Caso 2: $l = r$
 $Lcp(L, M) > l$
 $Lcp(M, R) > l$
 $Lcp(L, M) = Lcp(M, R) = l$



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

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



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

Costruzione suffix array: ricorsione

- 1 Ricorsione su T_0T_1
- 2 $\text{suffissi}(T_0T_1) \Leftrightarrow \text{suffissi}(T_0), \text{suffissi}(T_1)$
- 3 $\text{suffissi}(T_0T_1) \Leftrightarrow \text{suffissi}(T_2)$
- 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 $\text{suffissi}(T_0)$ ordinati
- 7 Radix sort
- 8 Fusione $\text{suffissi}(T_0T_1), \text{suffissi}(T_2)$

Costruzione suffix array: fusione

Confronto suffisso di T_0 e T_2

- 1 $T_0[i:] \leq T_2[j:]$
- 2 $T[3i:] \leq T[3j+2:]$
- 3 $T[3i]T[3i+1:] \leq T[3j+2]T[3j+3:]$
- 4 $T[3i]T_1[i:] \leq T[3j+2]T_0[j+1:]$

Costruzione suffix array: fusione

Confronto suffisso di T_1 e T_2

- 1 $T_1[i:] \leq T_2[j:]$
- 2 $T[3i+1:] \leq T[3j+2:]$
- 3 $T[3i+1]T[3i+2:] \leq T[3j+2]T[3j+3:]$
- 4 $T[3i+1]T[3i+2]T[3i+3:] \leq T[3j+2]T[3j+3]T[3j+4:]$
- 5 $T[3i+1]T[3i+2]T_0[i+1:] \leq T[3j+2]T[3j+3]T_1[j+1:]$

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 $\{s_1, \dots, s_k\}$

- 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 x
- 5 Nodo z , $C_z =$ tutti 1
- 6 Tempo $O(kn)$
- 7 n : somma lunghezze $|s_1| + \dots + |s_k|$

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 $\text{lca}(x, y)$ 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
- 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 $z \leftarrow \text{lca}(x, y)$
 - 2 $D_z[i] =$
 - 3 Aggiorna C_z

Licenza d'uso

Quest'opera è soggetta alla licenza Creative Commons:

Attribuzione-Condividi allo stesso modo 4.0.

(<https://creativecommons.org/licenses/by-sa/4.0/>).

Sei libero di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire, recitare e modificare quest'opera alle seguenti condizioni:

- **Attribuzione** — Devi attribuire la paternità dell'opera nei modi indicati dall'autore o da chi ti ha dato l'opera in licenza e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
- **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.