Elementi di Bioinformatica

Gianluca Della Vedova

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

30 novembre 2018

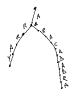
Gianluca Della Vedova Elementi di Bioinformatica

Trie

Trie

- Albero
- Query: parola ∈ dizionario
- archi etichettati
- Percorso radice-foglia = parola

Dizionario ABRACADABRA ARRAY **ABRA**

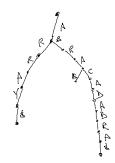


Gianluca Della Vedova Elementi di Bioinformatica 2/1

Trie

Terminatore \$ non appartiene all'alfabeto

Dizionario ABRACADABRA\$ ARRAY\$ ABRA\$



1/1

Gianluca Della Vedova Elementi di Bioinformatica

Suffix tree

Definizione

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

BANANA\$

Gianluca Della Vedova Elementi di Bioinformatica

Suffix tree 2: Definizione

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

Problemi

- Spazio O(n²)
- Puntatori al testo (posizioni)
- Spazio 20n bytes

Gianluca Della Vedova Elementi di Bioinformatica

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	-

Gianluca Della Vedova

Elementi di Bioinformatica

Elementi di Bioinformatica

Da Suffix tree a Suffix array

- $\, \bullet \,$ Visita depth-first diST
- o archi uscenti di ogni nodo in ordine lessicografico
- Lcp[i] = string-depth dilca(i, i + 1)

Da Suffix array a Suffix tree

- Lcp = 0: partizione SA
- o corrispondono ai figli della
- ricorsione prendendo i

Gianluca Della Vedova Elementi di Bioinformatica

Sottostringa comune più lunga di due stringhe

Suffix tree generalizzato

Due stringhe s_1 e s_2

- Suffix tree generalizzato = insieme di stringhe
- $ST(s_1\$_1s_2\$_2)$
- Nodo x con foglie di s_1 e s_2
- Sottostringa di s₁ e s₂
- $ST(s_1\$s_2\$)$
- Max string-depth

BANANA\$ s_2 :
PANNA\$

10/1

Gianluca Della Vedova Elementi di Bioinformatica

Gianluca Della Vedova Elementi di Bioinformatica

Pattern matching su suffix array

Occorrenza P in TSuffissi di T che iniziano con P

Ricerca in SA

- Ricerca dicotomica
- Tempo $O(m \log n)$ caso pessimo
- Controllare tutto *P* ad ogni iterazione
- log₂n iterazioni

Gianluca Della Vedova Elementi di Bioinformatica

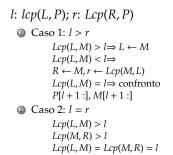
Accelerante 1

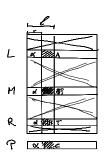
- Intervallo *SA*(*L*, *R*) di *SA*
- Elemento mediano M
- Tutti i suffissi in SA(L, R) iniziano con uno stesso prefisso lungo Lcp(SA[L], SA[R])
- Non confrontare con i primi Lcp(SA[L], SA[R]) caratteri



Gianluca Della Vedova Elementi di Bioinformatica 12/1

Accelerante 2





9/1

11/1

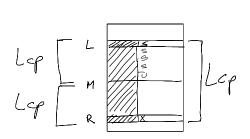
Gianluca Della Vedova Elementi di Bioinformatica

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: $L = h_{\frac{n}{2k-1}}$, $R = (h+1)_{\frac{n}{2k-1}}$
- 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*: $Lcp(h_{\frac{n}{2^{k-1}}}, (h+1)_{\frac{n}{2^{k-1}}})$
- $t = \frac{n}{2}$, $Lcp(2ht + 1, (2h + 2)t)) = min\{Lcp(2ht + 1, (2h + 1)t), Lcp((2h + 1)t + 1, (2h + 2)t), Lcp((2h + 1)t, (2h + 1)t + 1)\}$

Gianluca Della Vedova Elementi di Bioinformatica 14/1

Accelerante 3: calcolo Lcp in tempo O(n)



Passaggio da *s* a *z* deve esistere

13/1

Acceleranti 2: Osservazione

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

Gianluca Della Vedova Elementi di Bioinformatica 15/1 Gianluca Della Vedova Elementi di Bioinformatica 16/1

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) $\Leftrightarrow \bigcup_{i=0,1,2} \text{suffissi}(T_i)$

Gianluca Della Vedova Elementi di Bioinformatica

17/1

Costruzione suffix array: ricorsione

- ① Ricorsione su T_0T_1
- ② suffissi(T_0T_1) \Leftrightarrow suffissi(T_0), suffissi(T_1)
- 3 suffissi(T_0T_1) \Leftrightarrow suffissi(T_2)
- ⑤ $T[3i + 2:] = T[3i + 2]T[3i + 3:] = T[3i + 2]T_0[i + 1:]$
- \odot suffissi(T_0) ordinati
- Radix sort
- **Solution** Fusione suffissi(T_0T_1), suffissi(T_2)

Gianluca Della Vedova Elementi di Bioinformatica

18/1

22/1

24/1

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:]
- $\P[3i]T_1[i:] \iff T[3j+2]T_0[j+1:]$

Gianluca Della Vedova Elementi di Bioinformatica 19/1

Costruzione suffix array: fusione

Confronto suffisso di T_1 e T_2

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

Gianluca Della Vedova Elementi di Bioinformatica

KS

- 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
- 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
- Yuta Mori. SAIS https://sites.google.com/site/yuta256/

Gianluca Della Vedova Elementi di Bioinformatica

Sottostringa comune più lunga

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

- Suffix tree generalizzato
- ② Vettore $C_x[1:k]$ per ogni nodo x
- G $C_x = \bigvee C$ sui figli di C
- \bigcirc Nodo z, C_z = tutti 1
- ⑥ Tempo O(kn)
- ② n: summa lunghezze $|s_1| + \cdots + |s_k|$

Gianluca Della Vedova Elementi di Bioinformatica

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

- $N_x[i] \ge 1 \Rightarrow D_x[i] = N_x[i] 1$
- $N_x[i] D_x[i] = C_x[i]$

Gianluca Della Vedova Elementi di Bioinformatica Gianluca Della Vedova Elementi di Bioinformatica

Sottostringa comune più lunga di *k* stringhe

Gestione ST

- ${\color{blue} \bullet}$ Visita depth-first diST
- \bullet L_i : lista ordinata delle foglie di s_i
- Per ogni coppia x, y consecutiva in L_i

 - ② $D_z[i] =$
 - 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.

26/1

Gianluca Della Vedova Elementi di Bioinformatica 25/1 Gianluca Della Vedova Elementi di Bioinformatica