Capitolo 3

L'algoritmo Boyer-Moore e le sue varianti

Prima di passare alla trattazione degli algoritmi utilizzati per effettuare i test di confronto, dobbiamo per forza di cose descrivere l'idea che propone l'algoritmo Boyer-Moore, sulle quali procedure prenderanno spunto i prossimi algoritmi.

3.1 Boyer-Moore

La procedura di Boyer & Moore è considerata la più efficiente nelle applicazioni usuali. L'algoritmo presenta tre caratteristiche fondamentali:

- Right-to-Left Scan;
- Good-Suffix Rule;
- Bad-Character Rule;

Right-to-Left Scan

La procedura esplora i caratteri del pattern da destra verso sinistra, partendo dall'ultimo elemento a destra; il confronto viene effettuato con i caratteri del testo che sono nella finestra corrente.
Non appena c'è una discordanza tra un carattere del pattern con un carattere del testo alla posizione i, viene ritornato il valore $i + 1$ dell'ultima posizione in cui i caratteri del pattern e del testo erano uguali.

Good-Suffix Rule

Quando viene effettuato il controllo dei caratteri partendo da destra verso sinistra, ricaviamo informazioni su quanti caratteri del pattern coincidono con i caratteri della finestra corrente.
Non appena arriviamo ad un punto in cui non sono più uguali, vediamo se nel pattern ci sono altri segmenti che hanno gli stessi caratteri del pezzo di pattern che ha fatto match con il testo. Se tale segmento esiste, l'euristica del buon suffisso ci indica di scorrere il pattern a destra fino ad allineare tale segmento con i caratteri del testo che hanno fatto match; se ne esistono più di uno prendiamo il segmento più a destra (per evitare di saltare eventuali shift validi); se non ne esistono allora troviamo la lunghezza p del più lungo prefisso del pattern che è anche suffisso del pattern stesso, dopodichè si fa scorrere il pattern di $m - p$ posizioni.
Supponendo sempre che ci sia una discordanza tra $P[i]$ e $T[i + s]$, allora, $P[i + 1..m - 1] = T[i + s + 1..s + m - 1] = u$ e $P[i] \neq T[i + s]$. L'euristica del buon suffisso consiste nell'allineare il segmento T[i+s+1 .. s+m-1] = P[i+1 .. m-1] con la sua occorrenza più a destra in P che è preceduta da un carattere differente da P[i]. Se tale segmento non esiste, lo shift consiste nell'allineare il più lungo suffisso v di $T[i + s + 1..s + m - 1]$ con un prefisso corrispondente di P.

$\text{gs}_p(j) = min\{0 < k \leq m : P[j - k..m - k - 1] \sqsupset P \text{ AND } k \leq j - 1 \text{ AND } P[j - 1] \neq P[j - 1 - k]\}$

Il *min* eviterà di saltare shift validi; $P[j - k..m - k - 1] \sqsupset P$ prende il più lungo prefisso di P_{j-k} che è suffisso di P; $k \leq j - 1$ eviterà di uscire fuori dal pattern, $P[j-1] \neq P[j-1-k]\}$ eviterà di trovare un'occorrenza a destra in P preceduta dallo stesso carattere.

Bad-Character Rule

Durante il confronto dei caratteri del testo nella finestra che parte dalla posizione s con quelli del pattern, arriviamo ad una posizione i in cui $T[i + s] \neq P[i]$.
L'euristica del carattere discordante propone di trovare l'occorrenza più a destra di $T[i + s]$ nel pattern P.
Se il carattere corrispondente a $T[i + s]$ si trova nel pattern P nelle posizioni precedenti allora viene shiftato il pattern fino all'allineamento dei due caratteri uguali; se $T[i + s]$ non si trova nel pattern P, allora si avanza tutto il

pattern fino alla posizione $T[s+i+1]$; se $T[i]$ si nelle posizioni successive a $P[i]$ allora dovrebbe essere eseguito uno shift all'indietro!

$$bc_p(T[i]) = max(\{0 \leq k < m : P[k] = T[i+s]\} \cup \{-1\})$$

Il max ci consente di prendere sempre il carattere più a destra. Il prossimo avanzamento sarà quindi per:

$$i - bc_p(T[i])$$

I risultati della Bad-Character-Rule e del Good-Suffix-Rule possono essere salvate in apposite tabelle che chiameremo $bmBc$ e $bmGs$ che possono essere precomputate in tempo $O(m+n)$ e richiedono spazio in $O(m+\sigma)$ dove σ è la cardinalità dell'alfabeto Σ.
La complessità temporale della fase di ricerca è quadratica ma al più $3n$ confronti tra caratteri del testo vengono eseguiti per ricerche su un pattern non periodico. Su grandi alfabeti (relativamente alla lunghezza del pattern) l'algoritmo è estremamente veloce. L'algoritmo seguente implementa questa procedura.

Algorithm 4 BOYER-MOORE(P, m, T, n)

```
1: ▷Preprocessing
2: precomputa le tabelle bmBc e bmGs per il pattern P
3: ▷Searching
4: s ⇐ 0
5: while s ≤ n − m do
6:     for i = m − 1 to 0 AND P[i] = T[s + i] do
7:     end for
8:     if i < 0 then
9:         print s
10:        s ⇐ s + bmGs[0]
11:    else
12:        s ⇐ s + MAX(bmGs[i], bmBc[T[s + i]] − m + i + 1)
13:    end if
14: end while
```

Esempio

P = GCAGAGAG

c	A	C	G	T
$bmBc[c]$	1	6	2	8

i	0	1	2	3	4	5	6	7
$x[i]$	G	C	A	G	A	G	A	G
$suff[i]$	1	0	0	2	0	4	0	8
$bmGs[i]$	7	7	7	2	7	4	7	1

Figura 3.1: Le tabelle $bmBc$ e $bmGs$ che usa l'algoritmo BOYER-MOORE per il pattern GCAGAGAG.

3.2 TUNED BOYER-MOORE

Il TUNED BOYER-MOORE è una implementazione di una versione semplificata dell'algoritmo BOYER-MOORE il quale usa solamente l'euristica del Bad-Character, ed è molto veloce nella pratica. La parte più costosa in termini di tempo in BOYER-MOORE è il controllo che il carattere del pattern fa match con il carattere della window. Per evitare di fare questo lavoro troppo spesso, è possibile svolgere diversi shift prima di confrontare effettivamente i caratteri. L'algoritmo utilizza la funzione dello shift salvato in $bmBc$ per trovare $P[m-1]$ in T e continuare lo shift fino a trovarlo, fare ciecamente tre shift in fila. Questo richiede di salvare il valore di $bmBc[P[m-1]]$ in una variabile $shift$ e poi impostare $bmBc[x[m-1]]$ a 0. Questo richiede anche di aggiungere m occorrenze di $P[m-1]$ alla fine del testo. Quando $P[m-1]$ viene trovato, gli altri $m-1$ caratteri vengono controllati e viene applicato uno shift di lunghezza $shift$.

Caratteristiche principali di questo algoritmo quindi sono l'utilizzo della sola euristica del carattere discordante di BOYER-MOORE per la computazione dello shift; tale semplificazione rende TUNED BOYER-MOORE più facile da implementare rispetto all'originale e lascia inalterata la velocità dell'algoritmo che risulta avere buone prestazioni nella pratica.
Lo pseudocodice dell'algoritmo TUNED BOYER-MOORE è rappresentato nella seguente figura.