Laurea Magistrale in Informatica A.A. 2020/2021
Università degli Studi di Milano-Bicocca

Appunti pattern matching

Marta Pelusi
@Marta629

Indice

1 Pattern matching

1.1 Pattern matching esatto - PME

Input: testo T di lunghezza n e un pattern P di lunghezza m definiti su un alfabeto Σ.
Output: tutte le occorrenze esatte di P in T, cioè tutte le posizioni i di T in cui $T[i, i + m - 1]$ è esattamente uguale a P.

Complessità $O(m \times n)$

1.2 Pattern matching approssimato - PMA

Input: testo T di lunghezza n e un pattern P di lunghezza m definiti su un alfabeto Σ e soglia di errore k.
Output: tutte le occorrenze approssimate di P in T, cioè tutte le posizioni i di T in cui finisce (almeno) una sottostringa che ha con P una distanza di edit al più k.

Distanza di edit. Date due stringhe s_1, s_2 si definisce **distanza di edit** il minimo numero di operazioni di sostituzione, cancellazione e inserimento di un simbolo che trasformano s_1 in s_2 (o viceversa).

Concatenazione. La **concatenazione** di una stringa x con un simbolo σ dà origine alla stringa $x\sigma$.

Bordo. Il **bordo** di una stringa x è il più lungo prefisso proprio di x che occorre come suffisso di x.
Il **bordo** della concatenazione tra una stringa x e un simbolo σ è $B(x\sigma) = B(x)\sigma$.

2 Ricerca esatta con automa a stati finiti

2.1 Funzione di transizione

La **funzione di transizione** del pattern P è così definita

$$\delta : \{0, 1, ..., m\} \times \Sigma \to \{0, 1, ..., m\}$$

tale che:

$\delta(j, \sigma) = j + 1 \Leftrightarrow j < m \wedge P[j + 1] = \sigma$
$\delta(j, \sigma) = k \Leftrightarrow P[j + 1] \neq \sigma \vee j = m$, per $k = |B(P[1, j]\sigma)|$

2.2 Scansione del testo

Dopo aver letto il simbolo $T[q]$ abbiamo tre casi:

1. $T[q] = P[1, j + 1]$, $j < m$: il prefisso di P di lunghezza $j + 1$ occorre in T in posizione $q - (j + 1) + 1$

2. $T[q] \neq P[1, j+1]$
   (a) $j < m,\ k \leqslant j$: il prefisso di P di lunghezza k occorre in T in posizione $q - k + 1$
   (b) $j = m,\ k \leqslant m$: il prefisso di P di lunghezza k occorre in T in posizione $q - k + 1$
   (c) $j < m,\ k \leqslant j$: il prefisso di P di lunghezza k occorre in T in posizione $q - k + 1$

- $j \leqslant m,\ f \leqslant m$: il prefisso di P di lunghezza f occorre in T in posizione $q - f + 1$
  se $f = m$ allora P occorre in T alla posizione $q - m + 1$

Complessità $O(n)$

Pseudocodice

```
Procedure scan-text(delta,T,m)
begin
  n=|T|
  s=0
  for q=1 to n do
    f=delta(s,T[q])
    if f=m then
      output q-m+1
    s=f
end
```

3 Ricerca esatta con algoritmo di Knuth-Morris-Pratt - KMP

3.1 Prefix-function

Dato un pattern P di lunghezza m, la **prefix-function** ϕ è così definita:

$$\phi : \{0, 1, ..., m\} \rightarrow \{-1, 0, 1, ..., m-1\}$$

tale che:

$\phi(j) = |B(P[1, j])| = |B(P[1, j-1]P[j])|$, se $1 \leqslant j \leqslant m$
$\phi(j) = -1$, se $j = 0$
$\phi(j) = 0$, se $j = 1$

Se si raggiunge un valore k_p tale per cui $P[k_p + 1] = P[j]$, allora $\phi(j) = k_p + 1$. Se non si raggiunge mai un k_p tale per cui $P[k_p + 1] = P[j]$, allora si raggiungerà ad un certo punto il valore $k_p - 1$ che implica che il bordo di $P[1, j]$ è nullo, quindi $\phi(j) = 0$, cioè ancora calcolabile come $k_p + 1$.

Algoritmo di calcolo di ϕ

```
begin
  m=|P|
  phi(0)=-1
  phi(1)=0
  for j=2 to m do
    k=phi(j-1)
    while k >= 0 and P[k+1] != P[j] then
      k=phi(k)
    phi(j)=k+1
  return phi
end
```

3.2 Spostamento di W

Si assuma che:

- la finestra W si trovi in posizione i su T
- il primo carattere di P che determina un mismatch con T è in posizione j¿1 (il simbolo di mismatch su Tsarà quindi in posizione $i + j - 1$)

Per definizione il valore $k = \phi(j - 1)$ è la lunghezza del bordo del prefisso $P[1, j - 1]$

- $P[1, k]$ ha un'occorrenza su T che inizia in posizione i
- $P[1, k]$ ha un'occorrenza su T che inizia in posizione $p = i + j - \phi(j - 1) - 1$

Il confronto tra i simboli di T e di P con W nella nuova posizione p può ripartire dalle posizioni:

- $p = i + j - \phi(j - 1) - 1$ su T
- $\phi(j - 1) + 1 = k + 1$ su P

```
KMP(P,T,phi)
begin
  m=|P|
  n=|T|
  j=0
  for q=1 to n do
    while j >= 0 and P[j+1] |= T[q] then
      j=phi(j)
    j=j+1
    if j=m then
      return q-m+1
end
```

4 Ricerca esatta con algoritmo di Baeza-Yates e Gonnet, paradigma shift-and - byg

4.1 Tabella B

La **parola** B_σ per $\sigma \in \Sigma$ è definita come

$$B_\sigma = b_1 b_2 ... b_m$$

tale che:

$$B_\sigma[i] = b_i \Leftrightarrow P[i] = \sigma$$

Complessità $O(|\Sigma| + m)$

```
Procedura Compute-B(P)
begin
  m=|P|
  foreach sigma in Sigma do
    B_sigma=00...0
  M=10...0
  for i=1 to m do
    sigma=P[i]
    B_sigma=M OR B_sigma
    M=RSHIFT(M)
  return the table B of the words B_sigma
end
```

4.2 Parola D_j

La **parola** D_j è così definita

$$D_j = d_1 d_2 ... d_m$$

tale che:

$d_i = D_j[i] = 1 \Leftrightarrow P[1, i] = suff(T[1, j]) \Leftrightarrow$ P ha un'occorrenza che inizia in j-m+1 e finisce in j
$D_j[i] = D_{j-1}[i-1] \wedge B_{T[j]}[i]$, con $i \geqslant 1$

Complessità $O(|\Sigma| + m + n)$

```
Producedure BYG(P,T)
begin
  B=Compute-B(P)
  n=|T|
  D=00...00
  M=00...01
  for j=1 to n do
    sigma=T[j]
    D=RSHIFT1(D) and B_sigma
    if (D and M) = 00...01 then
      return j-m+1 %occorrenza di P in T
end
```

5 Ricerca approssimata con algoritmo di Wu e Manber, paradigma shift-and - wm

Parola B. La **Parola** B_σ si calcola nello stesso modo dell'algoritmo di BYG.

5.1 Parola D_j^h

La **parola** D_j^h è così definita

$$D_j^h = d_1 d_2 ... d_m$$

tale che:

$d_i = D_j^h[i] = 1 \Leftrightarrow P[1,i] = suff_h(T[1,j]) \Leftrightarrow$ P ha un'occorrenza che inizia in $j-m+1$ e finisce in j
Per $i \geqslant 1$ si ha:
$D_j^h[i] =$

$$\begin{gathered}(RSHIFT1(D_{j-1}^h[i-1]) \wedge B_{T[j]}[i]) \vee \\ RSHIFT1(D_{j-1}^{h-1}[i-1]) \vee \\ D_{j-1}^{h-1}[i] \vee \\ RSHIFT1(D_j^{h-1}[i-1])\end{gathered}$$

Nota: $D_j^h[1] = 1$ per $h > 0,\ j > 0$

6 Strutture di indicizzazione di un testo, Suffix-Array, BWT

Rotazione di indice j. La **rotazione di indice** j è la concatenzazione del suffisso $T[j,|T|]$ con il prefisso $T[1,j-1]$.

Esempio: $T = ggtcagtc\$$, allora $T[3,|T|]\ T[1,2] = tcagtc\$gg$ è la rotazione di indice 3.

6.1 Suffix Array - SA

Il **Suffix Array SA** di un testo T \$-terminato di lunghezza n è un array S di n interi tale che $S[i] = j$ se e solo se il suffisso di indice j è l'i-esimo suffisso nell'ordinamento lessicografico dei suffissi di T.
$T[S[i],n]$ è l'i-esimo suffisso di T.

$$i < i' \Rightarrow T[S[i],n] < T[S[i'],n]$$

Per calcolare il SA di un testo bisogna:

1. elencare i suffissi di T per indice crescente

2. ordinare lessicograficamente i suffissi
3. la prima colonna contenente i primi simboli è chiamata F
4. i numeri dopo l'ordinamento saranno il SA

Complessità $O(n \log n)$.

6.2 Burrows-Wheeler Transform - BWT

La **BWT B** di un testo T è un array di lunghezza n tale che $B[i]$ è il simbolo che precede l'i-esimo suffisso di T nell'ordinamento lessicografico dei suffissi di T. $B[i]$ precede $T[S[i], n]$, allora $B[i]$ è il simbolo $T[S[i] - 1]$.

Costruzione della **BWT** di un testo T:

1. elencare i suffissi di T per indice crescente
2. ordinare lessicograficamente i suffissi
3. nella colonna della BWT si avrà il simbolo che precede il simbolo in F
   se non si ha il testo è data per forza l'intera matrice delle rotazioni; in quel caso la BWT è l'ultima sua colonna

Complessità $O(n)$.

Proprietà 1. Per ogni posizione i, il simbolo $B[i]$ precede il simbolo $F[i]$ nel testo T.

Proprietà 2 - Last-First mapping. L'r-esimo simbolo σ in B e l'r-esimo simbolo σ in F sono lo stesso simbolo del testo T.

La **Last-First function LF** è a funzione che fa corrispondere ad una posizione i sulla BWT B la posizione j su F in modo tale che $B[i]$ e $F[j]$ siano lo stesso simbolo del testo T.

$$j = LF(i)$$

6.3 Test indexing

Q-intervallo

Def rispetto alla BWT: data la BWT B di un testo T e una stringa Q definitia su Σ (\$ escluso), il Q-intervallo è l'intervallo $[b, e)$ di posizioni che contiene i simboli che precedono i suffissi che condividono Q come prefisso.

Def rispetto al suffix array: dato il suffix array S di un testo T e una stringa Q Σ (\$ escluso), il Q-intervallo è l'intervallo $[b, e)$ di posizioni che contiene gli indici dei suffissi che condividono Q come prefisso.

Nota: $[1, n+1)$ è il Q-intervallo per Q=ϵ, relativo ai suffissi che condividono la stringa nulla come prefisso, cioè tutti i suffissi di T.

Backward extension. Dato un Q-intervallo $[b, e)$ e un simbolo σ, la **backward extension** di $[b, e)$ con σ è il σQ-intervallo.

7 Ricerca esatta con FM-index

7.1 LF-mapping

Dato il simbolo $B[i]$ della BWT B che precede il suffisso di indice $S[i]$, si ha che il suffisso $B[i]\ T[S[i], n]$ è l'r-esimo suffisso che inizia con un simbolo uguale a $B[i]$ se e solo se in $B[1, i]$ esistono esattamente r simboli uguali a $B[i]$.
Quindi, la posizione assoluta j del suffisso $B[i]\ T[S[i], n$ sarà

$$j = p + r$$

dove p è il numero di suffissi che iniziano con un simbolo inferiore a $B[i]$.

La **LF function** è la funzione che, data una posizione i sulla BWT B restituisce la posizione j (nell'ordinamento lessicografico) del suffisso $B[i]\ T[S[i], n]$, cioè

$$j = LS(i) = p + r$$

Osservazioni:

- $B[i]$ è il simbolo di T in posizione $S[i] - 1$
- $B[i]\ T[S[i], n]$ è il suffisso di indice $S[i] - 1$
- j è la posizione di $B[i]\ T[S[i], n]$ nell'ordinamento lessicografico, allora $S[j] = S[i] - 1$

7.2 Calcolo funzioni C e Occ

Dato un testo T di lunghezza n di cui si conosce la BWT B, il suo FM-index è composto dalle due funzioni C e Occ seguenti:

$$C : \Sigma \rightarrow \mathbb{N}$$

con $C(\sigma)$ = numero di simboli in B tali che siano lessicograficamente $< \sigma$.

$$Occ : \{1, 2, 3, ..., n+1\} \times \Sigma \rightarrow \mathbb{N}$$

con $Occ(i, \sigma)$ = numero di simboli uguali a σ in $B[1, i-1]$.
Nota: dalla tabella della funzione Occ si può ricavare la BWT.
p=numero di simboli inferiori a $B[i] = C(B[i])$

r=numero di simboli uguali a $B[i]$ in $B[1, i] = Occ(i+1, B[i]) = Occ(i, B[i]) + 1$

$$j = LF(i) = p + r$$

$$j = C(B[i]) + Occ(i, B[i]) + 1$$

7.3 Q-intervallo

Il Q-intervallo è l'intervallo $[b, e)$ delle posizioni dei suffissi (nell'ordinamento lessicografico) che condividono il prefisso comune Q.

$S[b, e)$: indici dei suffissi che condividono il prefisso comune Q.
$B[b, e)$: simboli che precedono i suffissi che condividono il prefisso comune Q.

7.4 Backward extension

La **backward extension** di un Q-intervallo $[b, e)$ con un simbolo σ è il σQ-intervallo $[b', e')$.

Dato un Q-intervallo $[b, e)$ e il simbolo σ, il σQ-intervallo $[b', e')$ è calcolato come:

$$b' = LF(i_1) = C(B[i_1]) + Occ(i_1, B[i_1]) + 1$$

$$e' = LF(i_k) + 1 = C(B[i_k]) + Occ(i_k, B[i_k]) + 1 + 1$$

dove i_1 è la più piccola posizione in $[b, e)$ tale che $B[i_1] = \sigma$ e
i_k è la più grande posizione in $[b, e)$ tale che $B[i_k] = \sigma$

$$b' = C(\sigma) + Occ(i_1, \sigma) + 1$$

$$e' = C(\sigma) + Occ(i_k, \sigma) + 1 + 1$$

ovvero

$$b' = C(\sigma) + Occ(b, \sigma) + 1$$

$$e' = C(\sigma) + Occ(e, \sigma) + 1$$