Appunti di
Teoria della Computazione

A cura di:
Francesco Refolli
Matricola 865955

Anno Accademico 2023-2024

Part I

Rizzi

Chapter 1

Pattern Matching

1.1 Programma

- Ricerca con Automa a Stati Finiti
  - Ricerca esatta
  - Confronto di simboli
  - Preprocessing: pattern $\rightarrow \theta(m|S|)$
  - Ricerca: scansione del testo $\rightarrow \theta(n)$
  - Tempo totale $\rightarrow \theta(m|S| + n)$
- Algoritmo di Knuth-Morris-Pratt (KMP)
  - Ricerca esatta
  - Confronto di simboli
  - Preprocessing: pattern $\rightarrow \theta(m)$
  - Ricerca: scansione del testo $\rightarrow \theta(n)$
  - Tempo totale $\rightarrow \theta(m + n)$
- Algoritmo di Baeza-Yates e Gonnet (BYG)
  - Ricerca esatta
  - Paradigma SHIFT-AND
  - Preprocessing: pattern $\rightarrow \theta(|S| + m)$
  - Ricerca: scansione del testo $\rightarrow \theta(n)$
  - Tempo totale $\rightarrow \theta(|S| + m + n)$
- Algoritmo di Wu-Manber (WM)
  - Ricerca approssimata
  - Paradigma SHIFT-AND
  - Preprocessing: pattern $\rightarrow \theta(|S| + m)$
  - Ricerca: scansione del testo $\rightarrow \theta(kn)$
  - Tempo totale $\rightarrow \theta(|S| + m + kn)$
- Ricerca con Suffix Array (SA)
  - Ricerca esatta

  - Confronto di simboli
  - Preprocessing: testo (costruzione di SA)
  - Ricerca $\rightarrow O(mlogn)$

- Ricerca con Burrows-Wheeler Transform/FM-index
  - Ricerca esatta
  - Preprocessing: testo (costruzione di BWT/FM-index)
  - Ricerca $\rightarrow \theta(m)$

1.2 Definizioni

Pattern Matching cercare un motivo all'interno di un oggetto più o meno complesso.

Pattern Matching su Stringhe cercare all'interno di un testo T le occorrenze di un pattern P.

String Matching Esatto cercare occorrenze esatte di P in T

String Matching Approssimato cercare occorrenze approssimate di P in T

Stringa giustapposizione (operatore $*$) di simboli su un alfabeto Σ (lunghezza di $X = |X|$).

Stringa nulla ϵ stringa composta da zero simboli.

Sottostringa $X[i,j] = X[i.j] = X[i:j] = X[i]X[i+1]...X[j]$, gli indici partono da 1.

Sottostringa propria $i \neq 1 \wedge j \neq |X|$.

Prefisso $X[1,j]$

Suffisso $X[i,|X|]$

Occorrenza Esatta Una posizione i del testo T tale che $T[i,i+m-1] = P$ è un'occorrenza esatta di P in T.

Occorrenza Approssimata Una posizione i del testo T tale che esista almeno una sottostringa $S = T[i-L+1,i]$ tale che $ED(P,S) \leq k$, con k soglia di errore, è un'occorrenza Approssimata di P in T. Nota bene: se $ED(P,S) \geq |P| - L$ allora i non è mai un'occorrenza approssimata.

Match Due simboli α, β sono un **match** sse $\alpha = \beta$

Mismatch Due simboli α, β sono un **match** sse $\alpha \neq \beta$

1.3 Algoritmo Banale per R.E.

Uso una finestra W di lunghezza $m = |P|$ che scorre su T da sinistra a destra. Il cursore scorre lungo la finestra e il pattern P verificando un match. In caso di mismatch la finistra si sposta a destra di 1 e il cursore riparte dalla prima posizione della finestra W e dal primo carattere del pattern P.

---

procedure TRIVIAL-EXACT-OCCURRENCES(P, T)
  $n \leftarrow |T|$
  $m \leftarrow |P|$
  $i \leftarrow 1$
  while $i \leq n - m + 1$ **do**
    $j \leftarrow 1$
    while $P[i] = P[i + j - 1] \wedge j \leq m$ **do**
      $j \leftarrow j + 1$
    end while
    if j = m+1 **then**
      output i
    end if
  end while
end procedure

1.4 Algoritmo Banale per R.A.

Uso una finestra W di lunghezza variabile $m \in [m-k, m+k]$ che scorre su T da sinistra a destra. La posizione iniziale della finestra è $i = m - k$, la lunghezza iniziale pure $m - k$. Se la finestra non evidenzia un'occorrenza approssimata di P in T allora scorro a destra di una posizione.

Fondamentalmente la differenza concettuale rispetto all'algoritmo banale della ricerca esatta è che la finestra è trascinata da destra invece che da sinistra, e allungata a sinistra ogni volta fino a che si può.

---

procedure TRIVIAL-APPROX-OCCURRENCES(T, P, k)
  $n \leftarrow |T|$
  $m \leftarrow |P|$
  $i \leftarrow m - k$
  while $i \leq n$ **do**
    $L \leftarrow m - k$
    while $L \leq m + k \wedge i - L + 1 \geq 1$ **do**
      if $ED(T[i - L + 1, i], P) \leq k$ **then**
        output i
      end if
      $L \leftarrow L + 1$
    end while
    $i \leftarrow i + 1$
  end while
end procedure

Chapter 2

Ricerca con Automa a Stati Finiti

2.1 Definizioni

Bordo di una stringa X $B(X)$ è il più lungo prefisso proprio di X che occorre come suffisso di X.

- $B(aaaccbbaac) = \epsilon$
- $B(\mathbf{ababa}ba) = ababa$
- $B(\mathbf{aaaaaaa}a) = \mathbf{aaaaaaa}$

2.2 Automa a Stati Finiti

Dato un pattern P di lungheza m si costruisce un automa a stati finiti $A = (Q, \Sigma, \delta, q_0, F)$ con

- $Q = 0, 1, 2, 3....m$
- Σ, alfabeto di P
- $\delta Q \times \Sigma \to Q$, funzione di transizione
- q_0, stato iniziale
- $F = m$, stato accettante

Gli stati rappresentano la quantità di caratteri consecutivi in match del pattern P sul testo letto T.

2.3 Funzione di Transizione

$\forall (j, \sigma) \in Q \times \Sigma$, $\delta(j, \sigma)$ rappresenta lo stato a cui si arriva da j attraverso il carattere σ. Questo può essere uno stato in avanti $j \to j+1$ nel caso di un match, oppure $j \to j' \wedge j' < j$ nel caso di un mismatch. Questo passo indietro è deciso in base al bordo della sottostringa letta fin'ora (i match + il carattere di mismatch). Il passo indietro è effettuato anche in caso si abbia riconosciuto tutta la stringa.

definizione di δ

- $\delta(j, \sigma) = j + 1$ sse $j < m \wedge P[j+1] \neq \sigma$
- $\delta(j, \sigma) = |B(P[1, j] \ * \ \sigma)|$ sse $j = m \vee P[j+1] \neq \sigma$

La ragione di passo indietro siffatto è che ragionevolmente un'occorrenza esatta di quel pattern non può avvenire se non in una posizione corrispondente con l'inizio del massimo prefisso.

2.3.1 Esempio

	1	2	3	4	5	6	7
P	a	c	a	c	b	a	c

δ(j,σ)	a	b	c	d
0	1	0	0	0
1	1	0	2	0
2	3	0	0	0
3	1	0	4	0
4	3	5	0	0
5	6	0	0	0
6	1	0	7	0
7	3	0	0	0

Figure 2.1: Esempio di Funzione di Transizione per un Pattern P

2.4 Costruzione della Funzione di Transizione

La funzione δ può essere costruita iterativamente riutilizzando δ_i per il calcolo di δ_{i+1}, dove δ_i è la funzione di transizione per il carattere i-esimo.

Per calcolare la funzione di transizione di δ_j a partire da δ_{j-1}

- chiamo k lo stato in cui andrebbe l'automa dallo stato $j-1$ con il nuovo j-esimo carattere del pattern
- cambio il valore di δ in $j-1, P[j]$ in modo che proceda allo stato successivo (j)
- per ogni simbolo in Σ copio il valore della riga k-esima nella riga j-esima.

Il tempo è $\theta(m|\Sigma|)$

procedure BUILD-SIGMA(δ, P, $Sigma$)
  $m \leftarrow |P|$
  for $\sigma \in \Sigma$ **do**
    $\delta(0, \sigma) \leftarrow 0$
  end for
  for $j = 1$ to m **do**
    $k \leftarrow \delta(j-1, P[j])$
    $\delta(j-1, P[j]) \leftarrow j$
    for $\sigma \in \Sigma$ **do**
      $\delta(j, \sigma) \leftarrow \delta(k, \sigma)$
    end for
  end for
end procedure

2.5 Scansione del Testo

Il tempo è lineare in quanto è una basilare scansione del testo in $\theta(N)$

---

procedure ASF-EXACT-OCCURENCES(δ, T, m)
    $n \leftarrow |T|$
    $j \leftarrow 0$
    for $i \leftarrow 1$ to n **do**
        $j \leftarrow \delta(j, T[i])$
        if $j = m$ **then**
            output $i - m + 1$
        end if
    end for
end procedure

2.5.1 Esempio

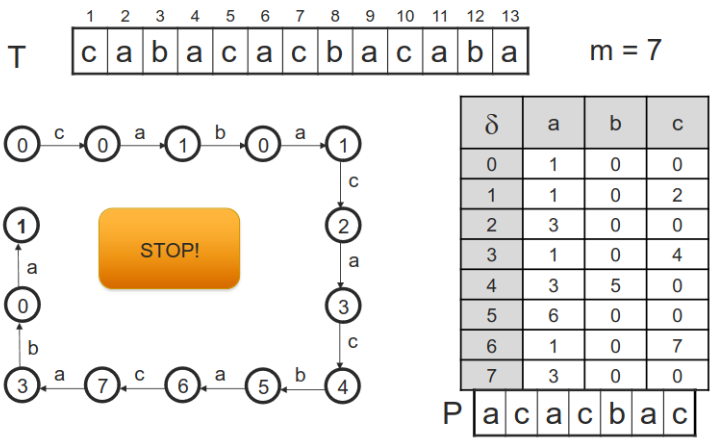


Figure 2.2: Esempio di Funzione di Scansione del Testo

2.6 Esercizi

2.6.1 1

Durante la ricerca esatta del pattern *acbdccbd* con l'ASF si arriva allo stato 6 dopo avere letto un certo simbolo nel testo. Quale simbolo?

soluzione $j = 6 \Rightarrow T[i] = P[j]$, quindi $P[j] = c$

2.6.2 2

L'esecuzione per la ricerca esatta del pattern *acbdacad*, automa si trova allo stato 6. Che simbolo del testo viene letto dopo, se l'algoritmo passa allo stato successivo?

soluzione $j' = j + 1 \Rightarrow T[j+1] = P[j+1]$, quindi $P[j+1] = a$

2.6.3 3

Dato il pattern *dccdbcd*, si può dire che l'esecuzione dell'automa su un determinato testo non arriva mai allo stato 0, dopo avere letto il simbolo *d* a partire da uno stato diverso da 0? In caso affermativo, specificare quali sono i possibili stati di arrivo.

soluzione Per ogni $j \neq 0$, il pezzo del pattern da cui calcolare il bordo sarebbe siffatto: $d...d$, quindi il bordo non sarebbe mai nullo, ergo non si arriva mai allo stato zero. In particolare:

- $1 \rightarrow 1$
- $2 \rightarrow 1$
- $3 \rightarrow 4$
- $4 \rightarrow 1$
- $5 \rightarrow 1$
- $6 \rightarrow 7$

2.6.4 4

Durante la ricerca esatta di un pattern P con automa a finiti, si passa dallo stato 6 allo stato 4 dopo avere il simbolo g nel testo. Dimostrare che $P[1] = g$.

soluzione Se $B(P[1,6]g) = 4$ allora $P[1,4] = T[i-3,i]$, ovvero che $P[4] = g$. Ma se ero nello stato 6 allora il testo aveva in posizione T[i-6,i-1] un match di 6 caratteri sul pattern. Se $P[4] = g$, allora $T[i-6+4-1] = T[i-3] = g$, ovvero, ricordando che $P[1,4] = T[i-3,i]$, $T[i-3] = g = P[1]$.

2.6.5 5

Dato il pattern *accabaccba*, l'automa arriva allo stato 4 dopo avere letto il simbolo a. Determinare i possibili stati di partenza.

soluzione in totale sono due i casi:

- da 3 leggendo a si arriva in 4
- da 8 leggendo a si arriva in 4 perchè la lunghezza del bordo di $B($**acca**$bacca) =$ **acca**

2.6.6 6

Dire, motivando la risposta, se la seguente catena si può verificare durante l'esecuzione di un automa a stati finiti per la ricerca esatta di un pattern.

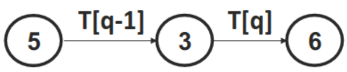


Figure 2.3: Catena di Computazione in Esame

soluzione Il problema della catena è che i passi indietro possono essere anche grossi ma i passi in avanti sono sempre in avanti di 1. Quindi la catena è sbagliata.

2.6.7 7

Dire, motivando la risposta, se la seguente catena si può verificare durante l'esecuzione di un automa a stati finiti per la ricerca esatta di un pattern. Se non è possibile, renderla "plausibile" correggendo uno dei tre simboli di transizione. Supporre che l'ultimo simbolo letto sia in posizione i del testo.

Figure 2.4: Catena di Computazione in Esame

- Dal primo stato si ricava che $P = xxxxa$
- Dal secondo che $P = xxxb$
- Dal terzo che $P = xxb$

Ora però bisogna verificare che i bordi delle transizioni siano corrette.

- $5 \xrightarrow{b} 4$ implica che $B(P[1,5]b) = 4$, ovvero che $P[1,4] = xxab$
- $4 \xrightarrow{b} 3$ implica che $B(P[1,4]b) = 3$, ovvero che $P[1,3] = abb$

Ora incrociamo queste informazioni con quelle ricavate prima:

- $P = xxxxa$
- $P = xxxb$
- $P = xxb$
- $P = abb$
- $P = xxab$ conflitto con $P = abb$

TODO: continuare con il renderlo plausibile

Chapter 3

Algoritmo KMP

3.1 Definizioni

Prefix Function del pattern P di lunghezza m: $\phi : [0, m] \to [-1, m]$. $\phi(j) = |B(P[1, j])|$ se $j \geq 1$ altrimenti $\phi(j) = -1$. È la lunghezza del bordo del prefisso del pattern di lunghezza j.

3.2 L'Algoritmo

L'algoritmo ricalca quello banale a finestra per la ricerca esatta, con la differenza che la funzione ϕ viene utilizzata per fare un salto ottimizzato quando si ottiene un mismatch. Infatti in caso di mismatch:

- i è la posizione della finestra W
- j è la posizione di mismatch sul pattern P
- $P[1, j-1]$ è il prefisso di match
- $i + j - 1$ è la posizione del mismatch su T
- La nuova posizione della finestra è $i \equiv i + j - phi(j-1) - 1$
- Il confronto riparte dalla posizione $j \equiv phi(j-1) + 1$ sul pattern
- Il confronto riparte dalla posizione $k = i + j - 1$ sul testo

Un caso particolare è quando $j = m+1$ ovvero il confronto arriva oltre l'ultima posizione del pattern. Si restituisce i come occorrenza del pattern e il confronto riparte da $i' = i+m-phi(m)$ sul testo e da $j' = phi(m) + 1$ sul pattern.

3.3 Confronto

Classe	ASF	KMP
Spazio	$O(m\|\Sigma\|)$	$O(m)$
Tempo	$O(m\|Sigma\| + n)$	$O(n + m)$
Preprocessing di P	$O(m\|\Sigma\|)$	$O(m)$
Scansione di T	$O(n)$	$O(n)$

- Automa
  - efficiente per pattern piccoli

  - richiede più tempo e memoria per pattern grandi
  - ricerca di P in testi diversi

- KMP
  - efficiente per pattern grandi
  - richiede più tempo per pattern piccoli

Chapter 4

Algoritmo BYG

Inizia adesso una classe di algoritmi che non effettuano più un confronto esplicito tra i simboli del pattern ma dipendono da operazioni bitwise effettuate in parallelo.

In particolare **Baeza-Yates-Gonnet** segue il paradigma **SHIFT-AND**.

4.1 Definizioni

Word di Bit 10101, il bit più significativo è a sinistra, il meno significativo è a destra.

AND and logico bitwise

OR or logico bitwise

RSHIFT shift dei bit di una posizione verso destra con il bit più significativo posto a 0

RSHIFT1 shift dei bit di una posizione verso destra con il bit più significativo posto a 1

4.2 Preprocessing del Pattern

Si calcolano $|\Sigma|$ words di m bit in tempo $\theta(|\Sigma| + m)$.

- tutte le words B_σ sono inizializzate a m bit a 0
- si crea una maschera M di m bit tutti a zero tranne il più significativo
- si esegue una scansione di P da sinistra a destra e per ogni posizione j di P si eseguono le operazioni bitwise:
-
  - $B_{P[j]} = M\ OR\ B_{P[j]}$
  - $M = RSHIFT(M)$
- ovvero $\forall j \in [1, |P|]$, $B_\sigma[j] = 1 \Leftrightarrow P[j] = \sigma$
- ovvero si setta ad 1 il j-esimo bit della word corrispondente al carattere $\sigma = P[j]$

P	a	b	c	a	b	a	Σ={a,b,c,d}

B_a	100101
B_b	010010
B_c	001000
B_d	000000

Figure 4.1: Esempio di words costruite sul pattern P

4.3 Algoritmo

Il testo viene scansionato dalla prima all'ultima posizione come nel caso dell'automa, ma per ogni posizione i del testo T viene calcolata una word D_i di m bit; ogni occorrenza di P in T avrà un 1 nel bit meno significativo della corrispondente word D_i (occorrenza in $i - m + 1$).

Word D_i $D_i[j] = 1 \Leftrightarrow P[1, j] = suff(T[1, i])$. Ovvero sse $P[i, j] = T[i - j + 1, i]$.

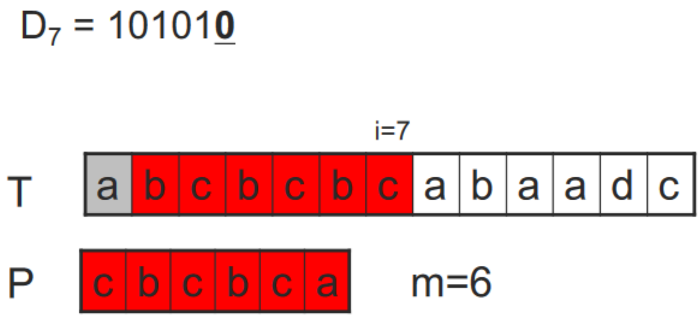


Figure 4.2: Esempio di word D_i

- D_0 è inizializzata con m bit a 0
- $\forall i \in [1, n]$ la word D_i è calcolata a partire da D_{i-1}
- con $j = 1$, $D_i[1] = 1\ AND\ B_{T[i]}[1]$
- $\forall j \in [2, m]$, $D_i[j] = D_{i-1}[j-1]\ AND\ B_{T[i]}[j]$

Semplificando un pò si arriva a poter comporre $D_i = RSHIFT1(D_{i-1})\ AND\ B_{T[i]}$.
Il tutto in tempo $\theta(n)$.

4.4 Esercizi

4.4.1 1

Una word D_i dell'algoritmo di BYG è uguale a 11111 e si riferisce al simbolo $T[i] = c$ sul testo. Si chiede di specificare il pattern P.

soluzione $D_i[j] \Leftrightarrow P[1,j] = suff(T[1,i])$ quindi se $D_i = 11111$ allora $\forall j \in [1,5]$, $P[1,j] = suff(T[1,j])$. Quindi $P = ccccc$.

4.4.2 2

Una word D_i dell'algoritmo di BYG è uguale a 01001. Si chiede di specificare la lunghezza del bordo del pattern.

soluzione La lunghezza del bordo del pattern corrisponde alla posizione $k < m$ del bit a 1 più a destra, visto che $D_i[k]$ è il massimo suffisso di T che ha match con il massimo prefisso di uguale lunghezza su P. Non è l'ultima posizione visto che il bordo deve essere un prefisso proprio. Quindi il bordo è lungo 2.

4.4.3 3

Sia $D_7 = 0000$ una word dell'algoritmo di BYG per $P = catg$ e un dato testo T. Sapendo che $T[5,7] = atg$, si può dire con certezza che in posizione 4 di T non c'è il simbolo c?

soluzione Se per assurdo $T[4] = c$, allora $T[4,7] = P$, ma allora $D_7[4] = 1$, tuttavia questo bit è uguale a zero, quindi $T[4] \leq c$.

Se mancasse l'informazione sul testo **non si potrebbe dire con certezza**.

4.4.4 4

Alla i-esima iterazione dell'algoritmo di BYG per cercare $P = aabaa$ (di cui si conosce la tabella B), viene calcolata la word $D_i = 11000$. Sapendo che la word D_{i-1} è uguale a D_i, specificare il simbolo di T in posizione i.

- $B_a = 11011$
- $b_b = 00100$

soluzione $D_i = RSHIFT1(D_{i-1})\ AND\ B_{T[i]}$, allora $11000 = RSHIFT1(11000)\ AND\ B_{T[i]} = 11100\ AND\ B_{T[i]}$, allora $B_{T[i]} = B_a$, quindi $T[i] = a$.

Chapter 5

Algoritmo WM

È molto simile a BYG, ma questo permette di ottenere tutte le occorrenze approssimate a meno di k errori.

5.1 Algoritmo

Si utilizza ancora la tabella B_σ con $\sigma \in \Sigma$ costruita in tempo $\theta(|\Sigma| + m)$. Ma le word D_i assumono un significato diverso: $\forall h \in [0, k]$ dico che $D_i^h[j] = 1$ sse $P[1, j]$ è uguale ad un suffisso di $T[1, i]$ a meno di h errori. Come funzione di errore stiamo ancora usando la distanza di edit ED.

In questo caso il passo di calcolo della D_i è riprodotto in $k + 1$ iterazioni, (D_i^0 è uguale all word di BYG) e se $D_i^k[m] = 1$, allora $i - m + 1$ è una occorrenza approssimata.

D_i^h può essere calcolata a partire da D_i^{h-1} in tempo costante:

- D_i^0 è calcolata come in BYG.
- $D_i^h = (D_{i-1}^h[j-1] \ AND \ B_{T[i]}[j]) \ OR \ D_{i-1}^{h-1}[j-1] \ OR \ D_{i-1}^{h-1}[j] \ OR \ D_i^{h-1}[j-1]$
- ovvero l'occorrenza esatta **oppure** correzione con sostituzione **oppure** correzione con rimozione da T **oppure** correzione con rimozione da P

Astraendo dal carattere particolare del pattern j:
$(RSHIFT1(D_{i-1}^h) \ AND \ B_{T[i]}) \ OR \ RSHIFT1(D_{i-1}^{h-1}) \ OR \ D_{i-1}^{h-1} \ OR \ RSHIFT1(D_i^{h-1})$. La scansione ha tempo $\theta(kn)$.

5.2 Esercizi

5.2.1 1

La parola 0100 è la parola D_i^0 dell'algoritmo di ricerca approssimata di Wu e Manber. Dire se 1011 può essere la parola D_i^1.

soluzione $D_1^0[2] = 1 \Rightarrow P[1, 2] = suff_0(T[1, i]) \Rightarrow P[1, 2] = suff_1(T[1, i])$. In generale $D_i^h[j] = 1 \Rightarrow D_i^{h+1}[j] = 1$

5.2.2 2

La parola D_i^0 dell'algoritmo di ricerca approssimata di Wu e Manber è uguale a 10100. Indicare quali bit della parola D_i^1 è possibile dedurre da D_i^0.

soluzione innanzitutto $D_i^h[j] = 1 \Rightarrow D_i^{h+1}[j] = 1$ quindi $D_i^1 = 1x1xx$. Poi $D_i^h[j] = 1 \Rightarrow D_i^{h+1}[j+1] = 1$. Di conseguenza $D_i^1 = 1111x$.

Chapter 6

Suffix Array

Questa è la prima classe di algoritmi che non effettuano il **preprocessing** sul pattern ma sul testo, creando una vera e propria indicizzazione multipurpose dello stesso.

6.1 Suffix Array

È una struttura dati che rappresenta l'ordine lessicografico dei suffissi di un testo T, in spazio $\theta(n\ logn)$. Permette l'identificazione di occorrenze esatte in tempo $\theta(m\ logn)$.

Come nota: al 2003 è possibile costruire il Suffix Array in tempo $\theta(n)$.

$\forall i \in [1, |n|]$, $SA[i] = q$ sse $T[q, |n|]$ è l'i-esimo suffisso del testo T in ordine lessicografico.

6.1.1 Esempio

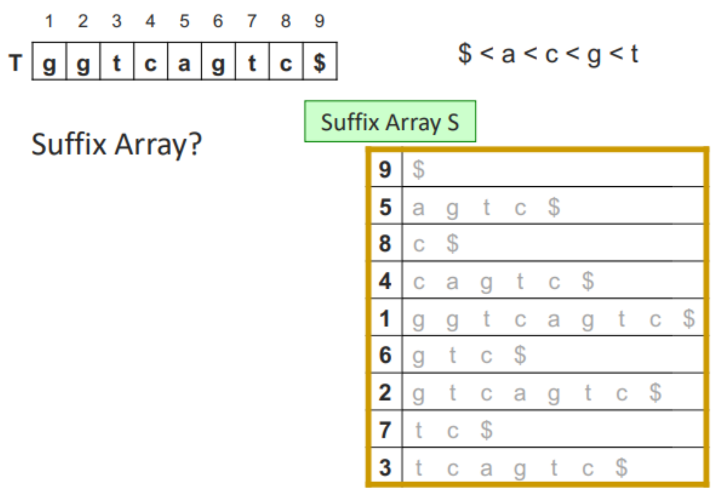


Figure 6.1: Esempio di SA

6.1.2 Ricerca Esatta con SA

A questo punto la ricerca di pattern P diventa una ricerca binaria sul Suffix Array per determinare la posizione di suffisso che ha un match sul pattern. Se il pattern occorre k allora è anche prefisso di k suffissi di T.

Chapter 7

BWT

7.1 Burrows-Wheeler Transform

È una struttura dati che rappresenta una permutazione reversibile di un T in $\theta(n\ log\Sigma)$ spazio.

La costruzione della BWT coincide con l'ordinamento di tutte le permutazioni del testo T in ordine lessicografico.

definizione una permutazione T_q è uguale alla concatenazione di $T[1, q-1] * T[q, |n|]$. Di conseguenza $T[q-1]$ è il primo simbolo di T_q, e $T[q-1, |T|]$ contiene $T[q, |T|]$.

Formalmente è l'array tale che $\forall i \in [1, |T|]$, $BWT[i] = \sigma$ sse T_q è la i-esima rotazione in ordine lessicografico di T e σ è l'ultimo carattere di T_q, ovvero $T[q-1] = \sigma$ (con q = 1, $T[n] = \sigma$).

definizione F è l'array con l'ordine lessicografico dei simboli di T. Servirà dopo.

7.1.1 Esempio

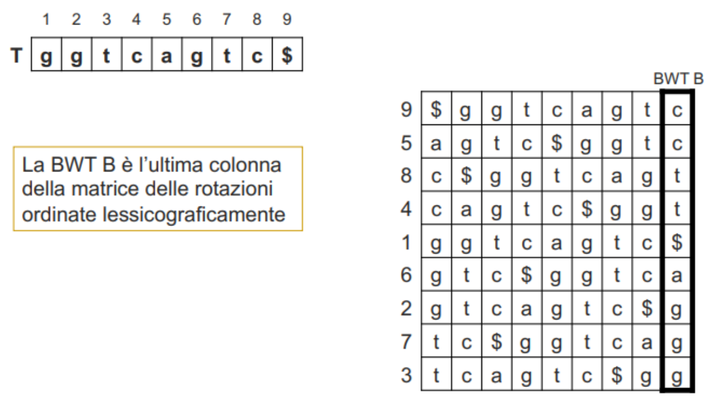


Figure 7.1: Esempio di BWT

7.2 Reversibile

È possibile riottenere il testo T a partire dalla BWT tramite l'array F. Visto che la BTW gli ultimi simboli delle permutazioni di T, allora $F[i] = T[q] \Rightarrow BWT[j] = T[q-1]$.

- Si inizializza il puntatore $i = 1$ sulle prime posizioni dei due array
- Siccome $\$ = min\Sigma$ allora $BWT[1] = T[n]$, in ogni caso

- ad ogni passo scrivo all'indietro $BWT[i]$ sul testo da riempire T e i diventa la posizione su F della q-esima occorrenza di $BWT[i]$ ($\sigma = BWT[i]$ è la q-esima occorrenza di σ in BTW).
- si continua così fino a che $BWT[i] = \$$, in quel caso il testo è terminato.
- come metodo di verifica della correttezza di una BWT semplicemente non si deve ottenere \$ prima di aver riempito $|T|$ caselle di T.

Come nota: la corrispondenza della q-esima occorrenza di σ su F con la q-esima occorrenza di σ su BWT si chiaman **Last-First Mapping**.

7.3 Calcolo di BWT da SA

Se conosco il testo T posso passare da BWT e SA agilmente.

definizione Visto che SA lista gli indici dei simboli in F, si può costruire la BWT come $\forall i \in [1, |T|]$, $BWT[i] = T[SA[i] - 1]$.

7.4 Q-Intervallo

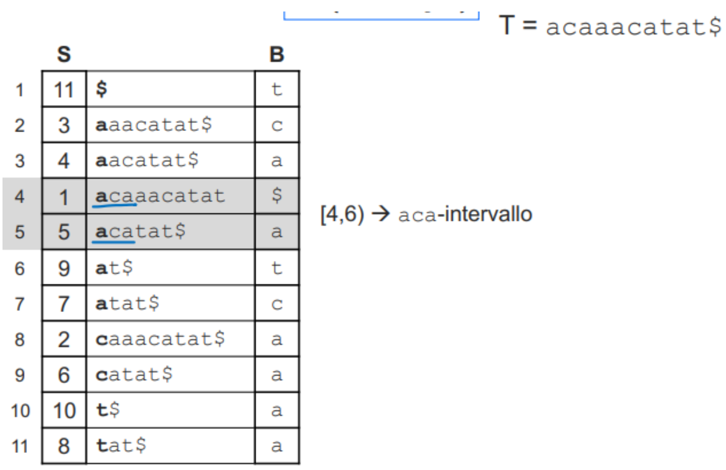
Q è una sottostringa di P (nell'algoritmo generalmente un suffisso)

Q-Intervallo su BWT è l'intervallo di posizioni $[b, e)$ sulla BWT tali che i suffissi successivi ai simboli nella BWT condividano strettamente il prefisso Q

Q-Intervallo su SA è l'intervallo di posizioni $[b, e)$ sul SA tali che i suffissi corrispondenti condividano strettamente il prefisso Q

il numero di occorrenze di Q in T è uguale a $(e - b)$.

7.4.1 Esempio


T = acaaacatat$

	S		B
1	11	$	t
2	3	aaacatat$	c
3	4	aacatat$	a
4	1	acaaacatat	$
5	5	acatat$	a
6	9	at$	t
7	7	atat$	c
8	2	caaacatat$	a
9	6	catat$	a
10	10	t$	a
11	8	tat$	a

[4,6) → aca-intervallo

Figure 7.2: Esempio di Q-intervalli

7.5 Backward Extension

La **backward extension** di un Q-intervallo $[b, e)$ con un simbolo σ è il σQ-intervallo. Ricavarlo è semplice perchè la BWT contiene l'informazione del simbolo precedente al suffisso i-esimo. Quindi per ogni posizione dell'intervallo $[b, e)$ dove $BWT[i] = \sigma$ si usa il **Last First Mapping** per risalire all'indice nel SA del suffisso che inizia con quel simbolo. Il nuovo intervallo $[b', e')$ e' formato dagli estremi indici ricavati in questo modo.

- $b' = LF($ più piccola posizione in $[b, e]$ tale che $B[i] = \sigma)$
- $e' = LF($ più grande posizione in $[b, e]$ tale che $B[i] = \sigma) + 1$

7.5.1 Esempio

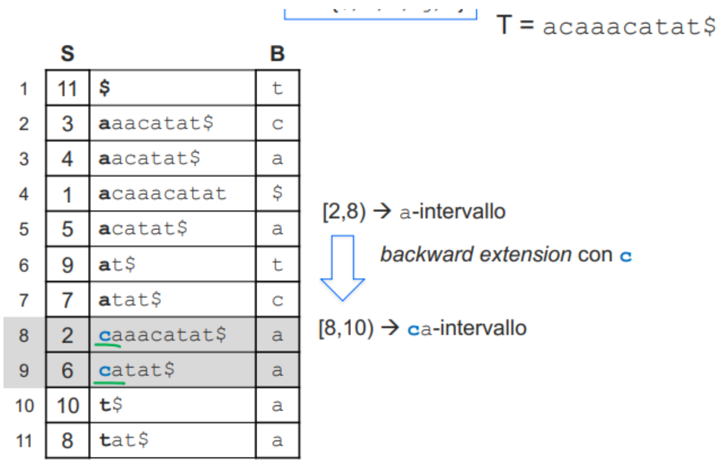


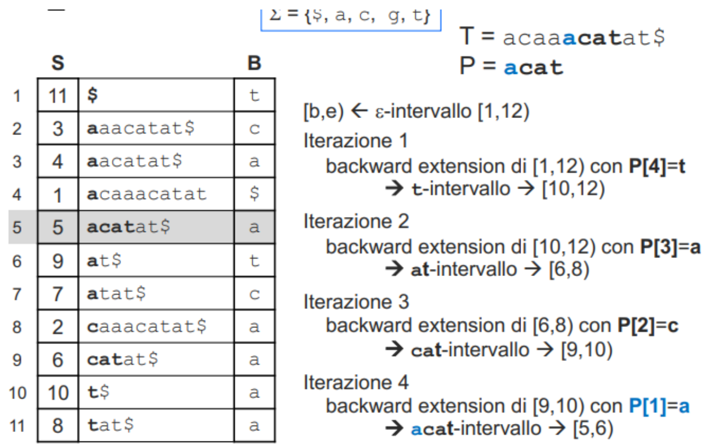
Figure 7.3: Esempio di Backward Extension

7.6 Algoritmo di Ricerca

- Si parte con il suffisso vuoto del pattern P, $Q \equiv \epsilon$, il Q-intervallo è $[1, n + 1)$.
- Quindi si inizializza $i = |P|$
- ad ogni passo si, $P[i] = \sigma$ viene concatenato a Q e si effettua la backward extension calcolando il Q-intervallo di σQ.
- se il Q-intervallo diventa l'intervallo vuoto allora P non ha occorrenze in T.
- una volta che $Q = P$ se l'intervallo è non vuoto allora ci sono $(e - b)$ occorrenze in T, e gli indici sul SA indicano le posizioni dei suffissi su T, ovvero delle occorrenze su T.

Il tutto in tempo $O(n\, m)$, ma si può ottimizzare ulteriormente.

7.6.1 Esempio



	S		B
1	11	$	t
2	3	aaacatat$	c
3	4	aacatat$	a
4	1	acaaacatat	$
5	5	acatat$	a
6	9	at$	t
7	7	atat$	c
8	2	caaacatat$	a
9	6	catat$	a
10	10	t$	a
11	8	tat$	a

Figure 7.4: Esempio di Ricerca con BWT e SA

Chapter 8

FM-Index

8.1 FM-Index

Fornisce una rappresentazione della **BWT** tramite due funzioni numerice (Occ e C) per ottenere la ricerca di occorrenze esatte in tempo lineare rispetto al pattern $\theta(m)$.

8.1.1 C

$C(\sigma)\ :\ \Sigma \rightarrow [0, n]$, è la funzione che restituisce il numero di simboli della BWT che sono $< \sigma$.

8.1.2 Occ

$Occ(i, \sigma)\ :\ [1, n+1] \times \Sigma \rightarrow [0, n]$, è la funzione che restituisce il numero di simboli nell'$i-1$-esimo prefisso della BWT, $BWT[1, i-1]$, che sono $= \sigma$.

8.1.3 Esempio

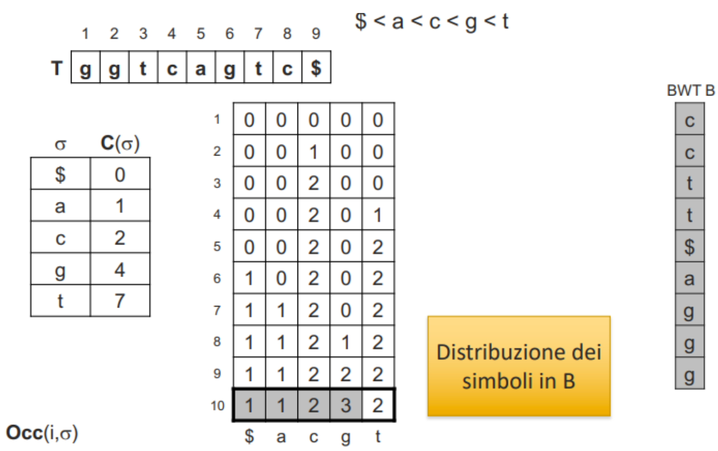


Figure 8.1: Esempio di Ricerca con Occ e C

8.1.4 Costruzione

La funzione Occ si può costruire facilmente scandendo la BWT simbolo per simbolo, calcolando $Occ[i] = Occ[i-1] + [0...1...0]$ in corrispondenza con il simbolo σ sulla cella $BWT[i]$. Notare che $Occ[0] = [0...0]$.

A quel punto C è costruito con una scansione lineare di $Occ[n]$ calcolando $C[\sigma'] = C[\sigma] + Occ[n][\sigma]$, con $C[\$] = 0$.

La costruzione di queste due tabelle richiede tempo $\theta(|\Sigma|n)$.

8.2 Last First Mapping

La **FM-Index** è rilevante perchè permette di calcolare in tempo costante la funzione $LF(i)$ che viene usata dagli algoritmi su BWT (ricostruzione del testo T e ricerca di pattern P).

La formula è $LF(i) = C(BWT[i]) + Occ(i, B[i]) + 1$, si sommano:

- il numero di simboli precedenti a σ
- il numero di carateri uguali a σ che lo precedono nella BWT
- 1 perchè la funzione Occ indica il numero di simboli $q-1$ che precedono σ, ma noi vogliamo la q-esima occorrenza

8.2.1 Backward Extension

In questo modo la formula per il nuovo Q-intervallo della ricerca con BWT si semplifica:

- siccome per definizione non esistono occorrenze di σ in $BWT[b, i_0 - 1]$, allora il numero di simboli σ in $BWT[1, i_0 - 1]$ è uguale al numero di simboli in $BWT[1, b - 1]$, ovvero a $Occ(b, \sigma)$
- per la stessa ragione si può riscrivere $Occ(i_f, \sigma) = Occ(e, \sigma)$

per ottenere:

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

Ora questa estensione del Q-intervallo è fatta in tempo costante, e la ricerca del pattern ha tempo lineare rispetto al pattern $O(m)$.

8.3 Self Index

Dire che **FM-index** sia un **self-index** significa che esprime informazioni sui dati e li indicizza allo stesso tempo.

La funzione Occ permette di ricostruire la BWT, infatti per ottenere il simbolo di $BWT[i]$ si guarda all'indice del simbolo dell'unico 1 all'interno di $(Occ[i + 1] - Occ[i])$.

Part II

Esami

Chapter 9

Note

Fare molti esercizi di calcolo della BWT, del SA e altre robe operative visuali.

Chapter 10

29 Gennaio 2021

10.1 Esercizio 5

Dare la definizione di parola D_j per la ricerca esatta con algoritmo di Baeza-Yates-Gonnet. Con riferimento al pattern P = *aca* e al testo T = *gtccata* specificare (motivando la risposta) la parola D_6.

soluzione In BYG la word D_j è utilizzata per individuare le occorrenze esatte di un pattern P. Formalmente è una word di m bit dove $\forall i\ D_j[i] = 1$ sse $P[1,i] = suff(T[1,j])$ ovvero se il prefisso i-esimo del pattern è uguale ad un suffisso del prefisso j-esimo di T. Per costruzione, un'occorrenza esatta è identificabile verificando che il bit $D_j[m] = 1$. In quel caso l'occorrenza di P si trova all'indice $j - m + 1$. La word D_j è calcolata ad ogni iterazione tramite $D_j = RSHIFT1(D_{j-1}) \wedge B_{T[i]}$, dove $\forall \sigma \in \Sigma$, B_σ è l'array di bit dove $B_\sigma[i] = 1 \Leftrightarrow P[i] = \sigma$.

Di conseguenza, $D_6[i] = 1$ sse $P[1,i] = T[1,6]$. In questo caso $D_6[1] = 0$ ($a \neq suff(gtccat) \Rightarrow P[1,1] \neq suff(T[1,6])$), $D_6[2] = 0$ ($ac \neq suff(gtccat) \Rightarrow P[1,1] \neq suff(T[1,6])$), $D_6[1] = 0$ ($aca \neq suff(gtccat) \Rightarrow P[1,1] \neq suff(T[1,6])$). Quindi $D_6 = 000$.

10.2 Esercizio 6

Dare la definizione di funzione di transizione per la ricerca esatta con automa a stati finiti specificando insieme di partenza (dominio) e insieme di arrivo (codominio). Scegliere (e specificare) un pattern, un testo e un alfabeto e mostrare: la funzione di transizione e l'esecuzione dell'algoritmo di ricerca esatta con automa a stati finiti.

soluzione La funzione di transizione $\delta : Q \times \Sigma \rightarrow Q$, dove Q è insieme degli stati $Q = \{0, ...m\}$ dove $q \in Q$ rappresenta la corrispondenza esatta tra il suffisso del testo $T[i-q+1, i]$ e il prefisso q-esimo $P[1,q]$. È così definita:

- $\sigma = T[i]$, il carattere appena letto sul testo
- $\delta(q, \sigma) = q + 1$ sse $q < m \wedge P[q+1] = \sigma$
- $\delta(q, \sigma) = |B(P[1,q]\sigma)|$ sse $q = m \vee P[q+1] \neq \sigma$ (funzione di fallimento)

Prendo come esempio il pattern $P = abc$ e il testo $T = aaabca$.

δ	a	b	c
0	1	0	0
1	1	2	0
2	1	0	3
3	1	0	0

La computazione sul testo T: $q_0 \to [a]q_1 \to [a]q_1 \to [a]q_1 \to [b]q_2 \to [c]q_3 \to [a]q_1$.

10.3 Esercizio 7

Definire la BWT di un testo e specificarla per un testo scelto a piacere.

soluzione L'alfabeto di un testo per la BWT è esteso con il carattere temrinale \$. Quindi la BWT è definita come l'array di $n+1$ simboli dove $BWT[i] = \sigma$ sse σ è l'ultimo simbolo della i-esima permutazione in ordine lessicografico.

Dato il testo $T = abca\$$, sull'alfabeto $\Sigma = \{a, b, c, \$\}$, le permutazioni in ordine lessicografico sono:

- $\$abca$
- $a\$abc$
- $abca\$$
- $bca\$a$
- $ca\$ab$

Quindi la BWT del testo è uguale a $ac\$ab$.

Chapter 11

4 Febbraio 2021

11.1 Esercizio 5

Dare la definizione di funzione di fallimento (prefix-function) dell'algoritmo KMP facendo attenzione a specificare dominio e codominio.

soluzione La prefix-function $\phi : [0, m] \to [-1, m]$ ha valore $\phi(j) = |B(P[1, j])|$ sse $j \geq 1$ altrimenti $\phi(j) = -1$. È utilizzata per calcolare un salto della finestra ottimizzato quando si ottiene un mismatch oppure è stata riconosciuta un'occorrenza del pattern P su T.

11.2 Esercizio 6

Si consideri la seguente parola $D_j^0 = 010110$ dell'algoritmo di Wu e Manber. Indicare (spiegando la motivazione) quali sono i bit determinabili della parola D_j^1.

soluzione $D_j^1[2] = 1 \wedge D_j^1[4] = 1 \wedge D_j^1[5] = 1$ perchè $D_j^i[q] = 1 \Rightarrow D_j^{i+1}[q] = 1$, se il prefisso q-esimo occorre con al massimo i errori allora è anche vero che occorre con al massimo $i + 1$ errori tautologicamente.

Inoltre visto che $D_j^i[q] = 1 \Rightarrow D_j^{i+1}[q + 1]$ (il carattere q+1-esimo extra viene rimosso potendomi sbagliare su un simbolo in più) allora $D_j^0[2] = 1 \Rightarrow D_j^1[3] = 1$ e $D_j^0[5] = 1 \Rightarrow D_j^1[6] = 1$.

Inoltre visto che $ED(P[1, 1], T[j - 1, j]) \leq 1$ (se sono due simboli diversi ne sostituisco uno dei due) allora posso anche dire che $D_j^1[1] = 1$.

In totale allora $D_j^1 = 111111$.

11.3 Esercizio 7

Si consideri il pattern $P = acacaca$. Durante l'esecuzione dell'algoritmo KMP, di ricerca di P in un testo, la finestra passa dalla posizione i alla posizione $p = 21$. Sapendo che il più lungo prefisso di P che occorre in posizione i del testo è *acaca*, trovare la posizione iniziale i.

soluzione Se il più lungo prefisso di P che occorre in posizione i è lungo 5 allora il mismatch è avvenuto in posizione $j = 6$ sul pattern P. Di conseguenza la nuova posizione della finestra $p = i + j - \phi(j - 1) - 1 = 21$. Per definizione di prefix function $\phi(j - 1 = 5) = 3$. Quindi $i + 6 - 3 - 1 = 21 \Rightarrow i + 2 = 21 \Rightarrow i = 19$.

Chapter 12

16 Febbraio 2022

12.1 Esercizio 8

Si dia la definizione di parola D_i^h dell'algoritmo di Wu e Manber, specificando il significato dell'apice h e del pedice i (NB: tutte le notazioni utilizzate devono essere adeguatamente spiegate). Fare un esempio per un testo e un pattern a piacere. Specificare inoltre (motivando adeguatamente la risposta) la parola D_0^4 per un generico pattern lungo 8 simboli.

soluzione Mi ha lasciato un pò **perplesso**. Vuole seriamente che risponda $D_0^4 = 11110000...$?

12.2 Esercizio 9

Durante l'esecuzione dell'algoritmo KMP la finestra si trova in posizione 20 e il primo mismatch tra pattern e testo si trova in posizione 24 sul testo. Specificare la posizione sul testo da cui riparte il confronto dopo lo spostamento della finestra e perche' non si puo' derivare (con i dati a disposizione) la corrispondente posizione sul pattern (da cui riparte il confronto).

soluzione Se il primo mismatch sul testo è in posizione 24 allora la posizione di mismatch sul pattern è $j = 5$. Il confronto riparte dalla posizione $i + j - 1 = 20 + 5 - 1 = 24$ sul testo. Non è possibile calcolare la posizione di ripartenza sul pattern senza conoscere la lunghezza del bordo del massimo prefisso riconosciuto in posizione $i = 20$ sul testo, visto che quindi non si può conoscere il numero di caratteri che si assume l'algoritmo abbia già letto per la prossima possibile occorrenza di P su T.

12.3 Esercizio 10

Definire l'FM-index di un testo (NB: tutte le notazioni utilizzate devono essere adeguatamente spiegate) specificando dominio e codominio delle funzioni.

soluzione Anche qui mi prendete per sfinismento. FM-Index è definito come la coppia di funzioni $Occ : [1, m+1] \times \Sigma \to [0, m]$ ($Occ(i, \sigma)$ è il numero di simboli $= \sigma$ nel prefisso i-1-esimo della BWT: $B[1, i-1]$) e $C : \Sigma \to [0, m]$ ($C(\sigma)$ è il numero di simboli nella BWT inferiori a σ, ricordando che Σ è un insieme totalmente ordinato).

Chapter 13

20 Giugno 2022

13.1 Esercizio 5

Dare la definizione di funzione di transizione per ricercare un pattern P in un testo T tramite automa a stati finiti. In particolare, specificare il dominio e il codominio. Siano poi $P = ccaca$ e $T = bccacaec$ un pattern e un testo definiti su alfabeto $\{a, b, c, d, e\}$.

Si chiede di:

- specificare la funzione di transizione di P
- mostrare l'automa a stati finiti per cercare P in T, evidenziando gli stati che identificano un'occorrenza di P in T; mostrare inoltre il calcolo delle posizioni di inizio di ciascuna occorrenza sulla base dello stato trovato

NOTA BENE: tutte le notazioni usate nel rispondere alla domanda devono essere spiegate altrimenti la risposta non viene valutata.

soluzione

δ	a	b	c	d	e
0	0	0	1	0	0
1	0	0	2	0	0
2	3	0	0	0	0
3	0	0	4	0	0
4	5	0	2	0	0
5	0	0	1	0	0

Lo stato che identifica un'occorrenza è lo stato 5. L'occorrenza si trova in posizione $i-m+1 = i-4$, con i ultimo carattere letto del testo che ha portato allo stato 5.

13.2 Esercizio 6

Dare la definizione di parola Di dell'algoritmo di ricerca esatta basato su paradigma SHIFT-AND.

soluzione L'ho già fatto in un altro esercizio: vattelo a cercare con CTR+F.

13.3 Esercizio 7

Definire il Suffix Array di un testo.

soluzione L'ho già fatto in un altro esercizio: vattelo a cercare con CTR+F.

Chapter 14

15 Luglio 2022

14.1 Esercizio 5

Dare la definizione di funzione di fallimento per la ricerca esatta di un pattern P in un testo T tramite algoritmo KMP. In particolare, specificare il dominio e il codominio. Specificare e spiegare, aiutandosi con uno schema, la formula di calcolo del salto della finestra (dell'algoritmo KMP) durante la scansione del testo.

soluzione Spiego solo la regola di salto perchè la prefix function l'ho fatta un tot di volte.

Data la prefix function $\phi : [0, m] \to [-1, m]$, detta la vecchia posizione della finestra i e la prima posizione di mismatch sul pattern j, la nuova posizione della finestra è $y = i + j - phi(j - 1) - 1$.

Il ragionamento è quanto segue: $j - 1$ è il massimo prefisso del pattern riconosciuto in posizione i sul testo, $\phi(j - 1)$ è la lunghezza del bordo, ovvero del massimo prefisso proprio di $P[1, j - 1]$ uguale ad un suffisso di $P[1, j - 1]$, ovvero il prefisso della prossima possibile occorrenza a cui sono interessato. $j - 1 - \phi(j - 1)$ esprime il numero di caratteri che separano l'inizio della finestra dall'inizio di questa possibile occorrenza. Lo spostamento ottimizzato della finestra esegue questo salto.

14.2 Esercizio 6

Dare la definizione di parola D_i^k dell'algoritmo di Wu e Manber.

soluzione Sarà la quarta volta che lo faccio direi che sia superfluo.

14.3 Esercizio 7

Definire la Burrows-Wheeler Transform di un testo in termini di Suffix Array.

soluzione Per definizione il Suffix Array è l'array degli indici di inizio dei suffissi di un testo in ordine lessicografico. Sempre per definizione la BWT contiene in ogni cella l'ultimo carattere delle rotazioni in ordine lessicografico. Ovvero è uguale al primo carattere del suffisso precedente. Di conseguenza per ottenere la BWT B dal Suffix Array S su un testo T, dico che $B[i] = T[S[i] - 1]$ se $S[i] > 1$ altrimenti $B[i] = T[n]$.

Chapter 15

16 Settembre 2022

15.1 Esercizio 5

Dare la definizione di funzione di transizione per la ricerca esatta di un pattern P in un testo T tramite automa a stati finiti. In particolare, specificare il dominio e il codominio. Si richiede inoltre di spiegare come funziona l'algoritmo di ricerca di P in T utilizzando un esempio di pattern e testo.

soluzione L'ho già fatto in un altro esercizio: vattelo a cercare con CTR+F.

15.2 Esercizio 6

Dare la definizione di parola Di dell'algoritmo di Baeza-Yates-Gonnet.

soluzione L'ho già fatto in un altro esercizio: vattelo a cercare con CTR+F.

15.3 Esercizio 7

Definire l'operazione di backward extension di un Q-intervallo.

soluzione Detto Q una sottostringa del pattern P, il Q-intervallo è l'intervallo di indici sul Suffix Array e sulla BWT i quali suffissi corrispondenti hanno come prefisso Q. L'operazione di backward extension consiste nel calcolare l'intervallo per σQ, con $\sigma \in \Sigma$. Questa operazione può essere fatta in tempo lineare individuando due indici i_1, i_k rispettivamente la prima e l'ultima posizione in cui $BWT[i] = \sigma$ (ricordardo che il simbolo nella BWT è il carattere che precede il primo carattere del suffisso corrispondente all'indice del Suffix Array). Quindi si utilizza il Last-First Mapping per individuare il nuovo intervallo $[b', e') = [LF(i_1), LF(i_k) + 1)$.

In alternativa utilizzando la FM-Index è possibile effettuare questa operazione in tempo costante con $[b', e') = [Occ(b, \sigma) + C(\sigma) + 1, Occ(e, \sigma) + C(\sigma) + 1)$.

Chapter 16

27 gennaio 2023

16.1 Esercizio 1

Dare la definizione di funzione di transizione per la ricerca esatta tramite automa a stati finiti specificando in particolare dominio e codominio.

soluzione La funzione di transizione dell'ASF è $\delta : [0, |P|] \times \Sigma \to [0, |P|]$. Il suo valore è $\delta(q, P[q+1]) = q + 1$ sse $P[q+1] = T[i] \wedge q < m$ e $\delta(q, P[q+1]) = |B(P[q]\sigma)|$ se $P[q+1] \neq T[i] \vee q = m$.

16.2 Esercizio 2

Scegliere un testo e un pattern e fornire su di essi un esempio di parola D_5^1 di Wu e Manber e spiegare cosa rappresenta.

soluzione Prendo $T = abcdabcdaa$ e il pattern $abcdd$.

D_5^1 è la word dove $D_5^1[j] = 1$ sse $P[1, j] = suff_1(T[1, 5])$, ovvero se il j-esimo prefisso di P è uguale ad un suffisso di T[1,5] con al più un errore.

In questo caso $T[1, 5] = abcda$ e $D_5^1 = 11001$, ovvero $abcda$ ha un match con $abcdd$ con $ED(p, t) \leq 1$. In particolare $D_5^1[1] = 1$ perchè $P[1, 1] = T[5, 5]$ e $D_5^1[2] = 1$ perchè $ED(P[1, 2], T[5, 5]) = 1 \leq 1$.

16.3 Esercizio 3

Scegliere un testo e fornire la funzione Occ del suo FM-index, spiegando poi cosa rappresenta.

soluzione Scelgo il testo $T = aabac$ con alfbeto $\Sigma = \{\$,\ a,\ b,\ c\}$.

Il SA è {6 1 2 4 3 5} e quindi la BWT è $\{a\ a\ b\ c\ a\ \$\}$ e la F è $\{\$\ a\ a\ a\ b\ c\}$.

Dalla BWT calcolo la funzione Occ:

i	\$	a	b	c
1	0	0	0	0
2	0	1	0	0
3	0	2	0	0
4	0	2	1	0
5	0	2	1	1
6	0	2	1	1
7	1	3	1	1

La funzione $Occ : [1, n+1] \times \Sigma \rightarrow [0, n]$ nella sua applicazione $Occ(i, \sigma)$ ha come valore la quantità di simboli σ nell'intervallo $BWT[1, i-1]$ della BWT. Dall'applicazione parziale di questa funzione $Occ(7, \sigma)$ si ottiene la funzione C:

- $C(\$) = 0$
- $C(a) = 1$
- $C(b) = 4$
- $C(c) = 5$

Dove $C(\sigma)$ indica rispettivamente la quantià di simboli $\beta < \sigma$ presenti nella BWT.

Chapter 17

22 Febbraio 2023

17.1 Esercizio 1

Dare la definizione di parola D_i dell'algoritmo di ricerca esatta basato su paradigma SHIFT-AND. Specificare (1) quando la parola indica un occorrenza e (2) come si ottiene la posizione di inizio di tale occorrenza.

soluzione La parola D_i è una sequenza di bit tale che $\forall j \in [1, |P|]$ è vero che $D_i[j] = 1$ **SSE** $P[1, j] = suff(T[1, i])$. Se $D_i[m] = 1$ allora è vero che $P[1, m] = P = suff(T[1, i])$ ovvero che c'è un'occorrenza di P come suffisso di $T[1, i]$. La posizione dell'occorrenza $i' = i - m + 1$, ovvero al primo carattere di tale suffisso.

17.2 Esercizio 2

Dare la definizione di Q-intervallo sia rispetto al Suffix Array che rispetto alla BWT.

soluzione

Il Q-Intervallo rispetto al SA è l'intervallo di indici sul Suffix Array dove i suffissi puntati dalle celle nel range condividono tutti come prefisso la stringa Q.

Il Q-Intervallo rispetto alla BWT è l'intervallo di indici sulla BWT i cui simboli puntati dalle celle precedono nel testo un suffisso di esso che contiene come prefisso la stringa Q.

17.3 Esercizio 3

Spiegare la proprietà di Last-First-mapping aiutandosi con un testo a scelta.

soluzione Prendo il testo $T = aabca$.

La BWT = $\{a\ c\ \$\ a\ a\ b\}$ mentre F = $\{\$\ a\ a\ a\ b\ c\}$.

Ora per ricostruire il testo T mappato dalla BWT così come per la ricerca di un pattern sul testo tramite la BWT, torna utile la cosiddetta proprietà **Last First Mapping**. Ovvero una relazione biettiva tra la q-esima occorrenza di un simbolo σ nella BWT e la q-esima occorrenza di quel simbolo σ nella F e di conseguenza l'indice di inizio della permutazione corrispondente alla cella di F (= indice nella stessa cella del Suffix Array).

Per esempio $LF(B[4]) = F[3]$, oppure $LF(B[6]) = F[5]$.

Chapter 18

21 Giugno 2023

18.1 Esercizio 1

Spiegare come avviene la ricerca esatta di P in T con automa a stati finiti, supponendo di avere già definito la funzione di transizione δ. Specificare in particolar modo come vengono identificate le occorrenze.

soluzione La ricerca esatta consiste in una scansione lineare del testo con l'automa. Ogni carattere è letto e passato alla funzione di transizione per passare dallo stato $q \to q'$. Se dopo una transizione l'automa si trova nello stato m allora ha identificato una occorrenza esatta di P all'indice $i - m + 1$ su T. La ricerca esatta ha tempo $\theta(n)$, con $n = |T|$.

18.2 Esercizio 2

L'esecuzione della ricerca esatta di $P = bbbbbacca$ in un testo T con l'algoritmo KMP si trova con la finestra W in posizione 32. Il più lungo prefisso di P che occorre in posizione 32 è composto da 4 simboli. Calcolare la successiva posizione della finestra e le posizioni dei simboli su T e su P da cui riparte il confronto. Spiegare il procedimento usato.

soluzione Se il più lungo prefisso di P è lungo 4, allora l'indice dell'ultimo match su P é $j - 1 = 4$ e quind $\phi(j - 1) = |B(P[1, 4])| = 3$. Di conseguenza la nuova posizione della finestra é $i' = i + j - phi - 1 = 33$ il confronto riparte dalla posizione $x = i + j - 1 = 36$ sul testo T e dalla posizione $j' = phi(4) + 1 = 4$ sul pattern P.

18.3 Esercizio 3

Dare la definizione di Last-First Function $j = LF(i)$, che realizza la proprietà di Last-First mapping della BWT, spiegando il significato di i e j.

soluzione Formalmente $LF : [1, n] \to [1, n]$: il Dominio è l'insieme degli indici della BWT, il Codominio è l'insieme degli indici su F. La funzione **LF** realizza la corrispondenza $LF(i) = j$ tale che $B[i]$ e $F[j]$ siano lo stesso simbolo sul testo T.

Chapter 19

25 Luglio 2023

19.1 Esercizio 1

Descrivere l'algoritmo di ricerca esatta di un pattern in un testo tramite il paradigma SHIFT-AND, cioè descrivere la fase di preprocessing e la fase di scansione.

soluzione

Il preprocessing consiste nella creazione di una serie di word $\forall \sigma \in \Sigma\ B_\sigma[j] = 1 \Leftrightarrow P[j] = \sigma$. La costruzione avviene linearmente mantenendo una maschera M di $|P|$ bit inizialmente con solo il bit più significativo a 1. $\forall i \in [1, |P|]$, si seleziona la word B_σ corrispondente $\sigma = P[j]$ e si opera $B_\sigma = B_\sigma\ OR\ M$ (settando di fatto il bit corrispondente alla posizione j a 1), quindi si opera sulla maschera $M = RSHIFT(M)$ preparandola per la prossima iterazione. Il tutto in tempo $\theta(|P|)$.

Nella scansione si mantiene una word D_i di $|P|$ bit dove $D_i[j] = 1$ sse il prefisso $P[1, j]$ è uguale ad un suffisso di $T[1, i]$. La word D_0 è settata tutta a zero. Ogni word successiva D_i è calcolata a partire da D_{i-1} tramite $D_i \equiv RSHIFT1(D_{i-1})\ AND\ B_{T[i]}$. Se il bit meno significativo di D_i è uguale a 1 allora $T[i - m + 1, i]$ è un'occorrenza di P su T.

19.2 Esercizio 2

L'esecuzione della ricerca esatta di $P = abcaac$ in testo T con automa a stati finiti passa dallo stato 4 allo stato 2 dopo avere letto un certo simbolo σ di T. Specificare, motivando la risposta, che simbolo è σ.

soluzione Per costruzione della funzione di transizione $\delta(4) = |B(P[1, 4]\sigma)| = 2$, quindi $|B(abca\sigma)| = 2$. Di conseguenza $\sigma = b$, infatti $B(abc\mathbf{ab}) = \mathbf{ab}$.

19.3 Esercizio 3

Sia data la BWT $B = cc\$aaab$ di un testo. Indicare, motivando la risposta, qual è il valore j restituito per $i = 4$ dalla LF-function $j = LF(i)$. (si ricorda che $j = LF(i)$ è la posizione nel Suffix Array del suffisso del testo che inizia con il simbolo $B[i]$)

soluzione Se $B = cc\$aaab$ allora $F = \$aaabcc$. La funzione LF realizza il Last-First Mapping, ovvero la q-esima occorrenza di un simbolo σ su F e la q-esima occorrenza di un simbolo σ su B sono lo stesso simbolo sul testo T. Di conseguenza $B[i] = a \Rightarrow LF(i) = 2$ visto che $F[2]$ è la prima occorrenza di a.