Rabin-Karp
Report - Progetto laboratorio algoritmi

Nunzio Fornitto

0.1 Introduzione

Gli algoritmi di pattern matching su stringhe, chiamati anche algoritmi di confronto fra stringhe sono una classe importante degli algoritmi sulle stringhe che provano a individuare una posizione all'interno di una stringa più grande o di un testo, in cui una o più stringhe solitamente più piccole (dette anche pattern) si trovano.

0.1.1 Formalizzazione problema corrispondenza tra stringhe

Si assume che il testo sia un array $\mathcal{T}[1\ ..\ n]$ di lunghezza n e che il pattern sia un array $\mathcal{P}[1\ ..\ m]$ di lunghezza $m \leq n$. Inoltre si suppone che gli elementi di $\mathcal{P}$ e di $\mathcal{T}$ siano dei caratteri presi da un alfabeto finito Σ.
Un esempio puo' essere il seguente: Σ={0,1} oppure Σ={a,b,....,z}.$\mathcal{P}$ e $\mathcal{T}$ sono chiamati ***stringhe*** di caratteri. Si dice che il pattern(sottostringa) $\mathcal{P}$ è presente a cominciare dalla posizione s+1 nel testo $\mathcal{T}$ se $0 \leq s \leq n-m$ e $\mathcal{T}[s+1..s+m] = \mathcal{P}[1..m]$.

0.2 Rabin-Karp

*L'**algoritmo di Rabin–Karp*** è uno degli algoritmi di pattern matching su stringhe creato da *Michael O. Rabin e Richard M. Karp* nel 1987, utilizza l'hashing per trovare uno qualsiasi di un insieme di stringhe di pattern in un testo. Un semplice algoritmo di ricerca di stringhe confronterebbe il pattern dato con tutte le posizioni nel testo. Ciò comporterebbe una complessità di $O(nm)$ dove n = lunghezza del testo e m = lunghezza del pattern. Rabin-Karp migliora questo concetto utilizzando il fatto che il confronto degli hash di due stringhe può essere fatto in tempo lineare ed è molto più efficiente del confronto dei singoli caratteri.Fornendo una complessità runtime nel caso medio di $O(n+m)$.

*Sia **m** la lunghezza del pattern e sia **n** la lunghezza del testo su cui effettuare la ricerca.*

ALGORITMO	TEMPO DI PREPROCESSING	TEMPO DI ESECUZIONE
Ricerca stringhe Naïve	0 (no preprocessing)	θ***((n-m+1) m)***
Rabin-Karp	θ***(m)***	**Medio** θ***(n+m)***
Rabin-Karp	θ***(m)***	**Pessimo** θ***((n-m+1) m)***

0.2.1 Descrizione algoritmo

Dato il pattern $\mathcal{P}[1 .. m]$ si denota con p la sua corrispondente funzione hash e dato un testo $\mathcal{T}[1 .. n]$ si denota con t_s la corrispondente funzione hash della sottostringa $\mathcal{T}[s+1 .. s+m]$ di lunghezza m, con s=0,1,...,n-m. Quindi si ha che se t_s=p e questo accade se e solo se $\mathcal{T}[s+1 .. s+m]$=$\mathcal{P}[1 .. m]$ allora si dirà che il pattern appare nel testo con uno spostamento s.

Per il calcolo di p(funzione hash per il pattern) in un tempo ragionevole ovvero $O(m)$ si può utilizzare la seguente regola(regola di Horner):

$H = c_1 \times b^{m-1} + c_2 \times b^{m-2} + ... + c_m \times b^0$.

Con c = caratteri nella stringa, m = lunghezza del pattern, b = costante(Solitamente il valore b è il numero 256 corrispondente al numero di caratteri totali presenti nella tabella ASCII) .

Per il calcolo di t_s(funzione hash per la sottostringa del testo)si osserva che, dopo aver calcolato t_0 a partire da $\mathcal{T}[1..m]$ per calcolare i rimanenti valori t_1,t_2,...t_{n-m}, t_{s+1} può essere calcolata come segue:

$H = (H_p - C_p \times b^{m-1}) \times b + C_n$.

Con H_p = Hash precedente, b = costante, m = dimensione della nuova sottostringha considerata, C_p = carattere precedente "più significativo",C_n nuovo carattere inserito "meno significativo".

Grazie alla seguente formula si evita efficacemente il calcolo intero della funzione hash per le varie sottostringhe presenti nel testo, proprio perchè sfrutta il fatto che l'hash di una nuova sottostringa è uguale all'hash della sottostringa precedente nel testo a meno di un carattere, infatti si toglie dal calcolo dell'hash precedente il valore hash del carattere rimosso, si moltiplica il valore risultante per la b che è una costante comunemente chiamata base(serve per ripristinare l'ordine corretto degli esponenti dei caratteri precedentemente non toccati) e infine si aggiunge il valore del nuovo carattere.

Un esempio può essere il seguente:
Trovare il pattern "135" nel testo testo "2135" con base b=10(Nell'esempio si suppone di utilizzare solo gli interi anziché le conversioni di caratteri)
Calcolo hash del pattern:
H(135) $= 1 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 =$ *135* .
Quindi calcoliamo l'hash dei primi m=3 caratteri del testo, quindi "213":
H(213) $= 2 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 =$ *213* .
Chiaramente non si ha una corrispondenza tra l'hash del pattern e l'hash dei

primi 3 caretteri della sottostringa del testo, quindi si fa scorrere di una posizione la sottostringa del testo rilasciando il primo carattere della sottostringa precedentemente calcolata, procedendo come detto si avrà la seguente nuova sottostringa: "135", si potrebbe procedere al calcolo della funzione hash per la nuova sottostringa ma si osserva che, proprio perchè nella nuova funzione hash ci saranno due valori hash precedentemente calcolati, nella precedente funzione hash si può applicare la seguente formula:

$H = (213 - 200) \times 10 + 5 = \mathit{135}$.

Utilizzando questo tipo di approccio si possono trovare tutte le occorrenze del pattern $\mathcal{P}[1 \,..\, m]$ nel testo $\mathcal{T}[1 \,..\, n]$ in tempo $O(n+m)$.

Si nota che per il calcolo della funzione hash p del pattern e quindi delle varie t_s sottostringhe che hanno la stessa dimensione del pattern vi è un problema, ovvero quando il pattern p è formato da un numero elevato di caratteri, l'ipotesi che ogni operazione aritmetica sul pattern p venga effettuata in tempo costante è insoddisfatta. Per evitare ciò esiste un metodo che calcola p e le t_s modulo un appropropriato modulo q,(il q usato è solitamente un numero primo) garantendo tempo $O(n+m)$. Il valore viene calcolato utilizzando l'aritmetica modulare per assicurarsi che i valori hash possano essere memorizzati in una variabile intera. Quando verrà calcolato uno spostamento per cui il valore hash p del pattern da cercare è uguale al valore hash t_s della sottostringa all'interno del testo, prima di stabilire a partire da dove è effettivamente presente il pattern si dovrà procedere al confronto dei caratteri del pattern con quelli della sottostringa nel testo,nel caso in cui i valori hash siano uguali ma la sottostringa non sia uguale al pattern ricercato ci sarà un cosiddetto ***colpo mancato***.
Qalunque spostamento per cui quindi $t_s \equiv p \pmod{q}$ dovrà sempre essere controllato per capire se questo spostamento è realmente valido.

La procedura prende in input il testo T, il pattern da ricercare P la base d da usare che tipicamente è la cardinalità dell'alfabeto finito Σ, e il numero primo q.

Pseudocodice dell'algoritmo

RABIN-KARP-MATCHER(*T,P,d,q*)

```
1  n ← length[T]
2  m ← length[P]
3  h ← d^(m−1) mod q
4  p ← 0
5  t_0 ← 0
6  for i←0 to m
7      do p ← (dp +P[i]) mod q
8         t_0 ← (dt_0 +T[i]) mod q
9  for s←0 to n-m
10     do if p = t_s
11           then if P[1 .. m] = T[s + 1 .. s + m]
12                then "Il pattern appare nel testo T con spostamento" s
13        if s < n − m
14           then t_{s+1} = (d(t_s − T[ s + 1 ]h) + T[ s + m + 1 ]) mod q
```

Spiegazione pratica dell'algoritmo con un esempio :

Si suppone di voler trovare il pattern P="DC", nel testo T="ABDCB". Quindi si ha il pat="DC" e il txt="ABDCB", rispettivamente con dimensione M=2 ed N=5. Come numero primo q si prende il numero 11, e come base d che è la cardinalità dell'alfabeto si ha 256. Quindi si hanno i seguenti dati iniziali:

txt="ABDCB"

pat="DC"

q=11

M=2, N=5.

-Si procede con il calcolo della funzione hash:

h = (1*256)%11=3

-Si calcola il valore hash del pattern e quindi nello stesso ciclo for si può calcolare il valore hash della sottostringa presente all'interno del testo della stessa dimensione del pattern, quindi si memorizza in una variabile p il valore hash del pattern e in una variabile t il valore hash della sottostringa di dimensione M.

Iterazione 1:

p =(256∗0+68)%11=2

(68 all'interno delle parentesi è il valore hash del carattere "D" del pat).

t =(256∗0+65)%11=10

(65 all'interno delle parentesi è il valore hash del carattere "A" del txt).

-Si itera nuovamente poichè il pattern è lungo 2.

Iterazione 2:

p =(256∗2+67)%11=7

(2 all'interno delle parentesi è il valore hash del carattere "D" del pat precedente, 67 è il valore hash del carattere "C" del pat).

t =(256∗10+66)%11=8

(10 all'interno delle parentesi è il valore hash del carattere "A" del txt precedente, 66 è il valore hash del carattere "B").
Adesso si ci può fermare. Si ha quindi che il valore hash di "DC" è p=7, mentre quello di "AB" è t=8.
-Si scorre sui restanti caratteri del testo precisamente(n-m). Poichè si è già calcolato il valore hash della prima sottostringa nel testo di dimensione M, basta confrontare il valore hash del pattern con il valore hash della sottostringa per vedere se è uguale o meno.
p=t? NO!(p=7 t=8 non sono uguali i due valori hash).
Quindi si fa scorrere il testo di un carattere e si considera il nuovo carattere "D" del testo, e si va a calcolare il nuovo valore hash per la nuova sottostringa "BD", qui si potrà usare il vecchio valore hash di t rimuovendo da t il valore hash di "A" e aggiungendo il nuovo valore hash di "D".
t inizialmente era 8 il valore hash di "A" era 65, il valore hash di "D" è 68 quindi si ha:
t=256*(8-65*3)+68)%11=2
-Si confronta nuovamente se **p=t**, non è uguale, poichè p=7, t=2, quindi si scorre ancora sul testo andando a considerare il carattere "C" del testo e si va a calcolare il nuovo valore hash per la nuova sottostringa "DC" si potrà usare nuovamente il vecchio valore hash di t rimuovendo da t il valore hash "B" e aggiungendo il nuovo valore hash di "C".
t adesso vale 2 il valore hash di "B" era 66, il valore hash di "C" è 67 quindi si ha:
t=256*(2-66*3)+67)%11=7
Si confronta nuovamente se **p=t, SI!** poichè p=7, t=7, dato che i valori hash del pattern e della sottostringa nel testo sono uguali adesso si procede a verificare che anche i caratteri siano uguali, verificando che siano uguali uno per uno. I caratteri in questo caso risultano essere uguali:
P="DC"
T="ABDCB"
Quindi si stampa l'indice a partire dal quale il pattern P appare nel testo T, in questo caso 2(partendo da 0).
-Si va a calcolare quindi il valore hash per la successiva sottostringa di dimensione M. Si va a considerare il carattere "B" del testo, l'ultimo, e si va a calcolare il nuovo valore hash per la nuova sottostringa "CB" si potrà usare nuovamente il vecchio valore hash di t rimuovendo da t il valore hash "D" e aggiungendo il nuovo valore hash di "B".
t adesso vale 7 il valore hash di "D" era 68, il valore hash di "B" è 66 quindi si ha:
t=256*(7-68*3)+66)%11=3
Si confronta nuovamente se **p=t,** non è uguale poichè p=7, t=3, si prova a calcolare il successivo valore hash per la successiva sottostringa di dimensione M nel testo, si nota che non è possibile farlo poichè si è già scansionato tutto il testo T, di conseguenza non si calcola, il secondo ciclo for termina e si conclude che, il pattern P ha una singola occorrenza nel testo T e appare a partire dall'indice 2.