DOMANDA: Illustrare l'Algoritmo di String Matching (ASM)

RISPOSTA:
L'Algoritmo di String Matching risolve il cosiddetto problema del "matching", ovvero l'individuazione di una certa stringa di caratteri (detta "stringa chiave") all'interno di un'altra data stringa di caratteri (detta "stringa testo"). Da un punto di vista astratto, il problema è assimilabile a un problema di ricerca ("searching"). Ci sono diverse varianti del problema del matching, che differiscono per l'output che viene richiesto: per esempio, l'indice di inizio della prima occorrenza, il numero delle occorrenze, la sostituzione dell'occorrenza della stringa chiave nella stringa testo con un'altra stringa assegnata (il classico "trova e sostituisci" dei text editor), etc.

L'ASM ha come dati di input due stringhe, cioè la "stringa chiave" e la "stringa testo", e ha come dato di output il numero delle volte (detto numero delle occorrenze) in cui la stringa chiave compare come sottostringa all'interno della stringa testo.

L'ASM è basato sull'idea di effettuare un numero di passi uguale alla lunghezza della stringa testo (più precisamente, alla lunghezza della stringa testo meno la lunghezza della stringa chiave +1, per evitare inutili confronti nelle iterazioni finali), risolvendo a ogni passo il problema di determinare l'uguaglianza tra la stringa chiave e la sottostringa della stringa testo che ha come posizione inziale il passo e come lunghezza la lunghezza della stringa chiave.

Un esempio aiuta a capire la dinamica dell'ASM. Si consideri una stringa `testo` di `m`=7 caratteri e una stringa `chiave` di `n`=2 caratteri

`testo`	G	T	G	A	T	G	T
	i =0	1	2	3	4	5	6
`chiave`	T	G					
	0	1					

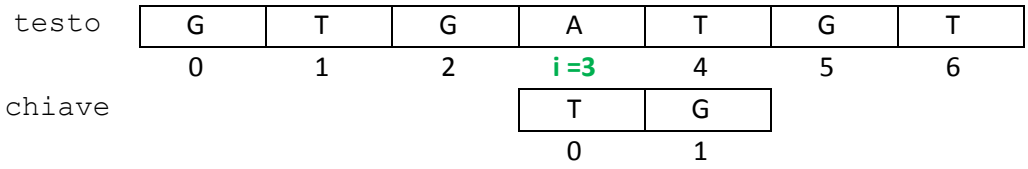
Al primo passo (`i`=0), poiché la sottostringa di inizio 0 e lunghezza 2 di `testo` è diversa dalla stringa `chiave`, allora non c'è alcuna occorrenza e bisogna continuare il processo iterativo

`testo`	G	T	G	A	T	G	T
	0	**i =1**	2	3	4	5	6
`chiave`		T	G				
		0	1				

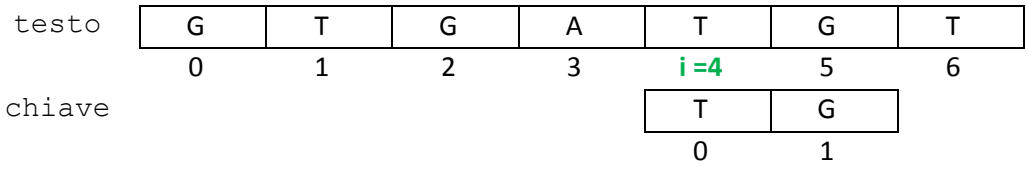
Al secondo passo (`i`=1), poiché la sottostringa di inizio 1 e lunghezza 2 di `testo` è uguale alla stringa `chiave`, allora bisogna incrementare il contatore delle occorrenze e poi continuare il processo iterativo

`testo`	G	T	G	A	T	G	T
	0	1	**i =2**	3	4	5	6
`chiave`			T	G			
			0	1			

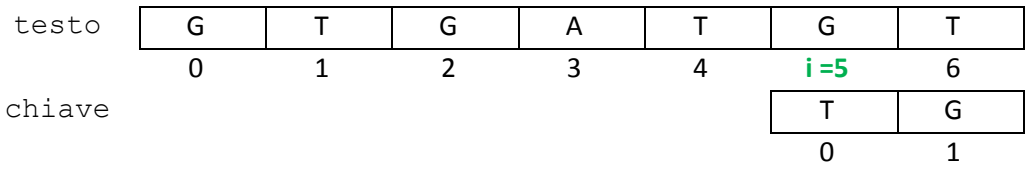
Al terzo passo (`i`=2), poiché la sottostringa di inizio 2 e lunghezza 2 di `testo` è diversa dalla stringa `chiave`, allora non c'è alcuna occorrenza e bisogna continuare il processo iterativo



Al quarto passo (`i`=3), poiché la sottostringa di inizio 3 e lunghezza 2 di `testo` è diversa dalla stringa `chiave`, allora non c'è alcuna occorrenza e bisogna continuare il processo iterativo



Al quinto passo (`i`=4), poiché la sottostringa di inizio 4 e lunghezza 2 di `testo` è diversa dalla stringa `chiave`, allora non c'è alcuna occorrenza e bisogna continuare il processo iterativo



Al sesto passo (`i`=5), poiché la sottostringa di inizio 5 e lunghezza 2 di `testo` è uguale alla stringa `chiave`, allora bisogna incrementare il contatore delle occorrenze. L'algoritmo termina perché è stata eseguita l'ultima iterazione (la (`m-n+1`)-sima, cioè la sesta iterazione, con l'indice di ciclo `i` che ha raggiunto il valore 5, cioè 7-2).

Questa è l'implementazione in C dell'ASM:

```
int string_matching(char chiave[],char testo [])
{
  int n, m, i, conta_chiave;
  n = strlen(chiave);
  m = strlen(testo);
  conta_chiave = 0;
  for (i=0; i <= m-n; i++)
     if(strncmp(chiave,&testo[i],n) == 0)
        conta_chiave++;
  return conta_chiave;
}
```

Un breve commento al codice. Le lunghezze delle due stringhe in input sono determinate dalla function **`strlen`** della libreria `string`. Il `for` su `i` da 0 a `m-n` gestisce i passi dell'ASM. Lo scopo del passo `i` è risolvere il problema di determinare l'uguaglianza tra la (sotto)stringa della stringa `chiave` di lunghezza `n` e la sottostringa della stringa `testo` che ha come posizione inziale `i` e come lunghezza `n`. Si noti che poiché è coinvolta una sottostringa della stringa `testo`, si deve usare la function della libreria `string` che effettua il confronto tra due sottostringhe, ovvero la **`strncmp`** (non si può usare la function **`strcmp`**, che invece confronta due stringhe).
Si ricorda che la function **`strncmp`** ha tre parametri di input: la prima sottostringa (più precisamente, il puntatore all'indirizzo iniziale della prima sottostringa), che è proprio l'indirizzo base della stringa `chiave`; la seconda sottostringa (più precisamente, il puntatore all'indirizzo iniziale della seconda

sottostringa), che è l'indirizzo dell'elemento in posizione `i` della stringa `testo`; la lunghezza delle due sottostringhe da confrontare, che nel nostro caso è `n`. La function **`strncmp`** ha un unico parametro di output: un numero intero, che vale 0 se le due sottostringhe di input sono uguali, vale un numero negativo se la prima sottostringa precede la seconda nell'ordine alfabetico, mentre vale un numero positivo se la prima sottostringa segue la seconda nell'ordine alfabetico.

Analizziamo la complessità dell'ASM. La dimensione computazionale è individuata dalle due quantità m e n, rispettivamente la lunghezza della stringa testo e la lunghezza della stringa chiave. La complessità di spazio è $m+n$, perché l'algoritmo lavora "in place".
Passiamo alla complessità di tempo. L'operazione dominante è l'operazione di confronto tra un elemento della stringa testo e un elemento della stringa chiave. Tali confronti sono eseguiti all'interno della function **`strncmp`**. Poiché la **`strncmp`** determina l'uguaglianza di due sottostringhe, cioè di due array, il numero di confronti per ogni chiamata della function **`strncmp`** è al più la lunghezza n delle sottostringhe in input. La function **`strncmp`** è chiamata una sola volta per ogni passo del ciclo `for`, che effettua $(m-n+1)$ passi. Quindi si può concludere che il numero totale di confronti è al più $(m-n+1)\ n$.
Si noti che si tratta di una complessità di caso peggiore.
In conclusione, la complessità di tempo dell'ASM è
$T(m,n)=(m-n+1)\cdot n=O(m\cdot n)$ confronti, al più.