

Titolo unità didattica: Stringhe ed elaborazione di testi [09]

Titolo modulo : Algoritmi per l'elaborazione di testi [02-T]

Algoritmi di analisi di testi e di individuazione di *pattern*

Argomenti trattati:

- ✓ algoritmo di analisi delle componenti di un testo
- ✓ algoritmo di string matching
- ✓ algoritmo di matching migliore

Prerequisiti richiesti: AP-07-11-T

problema:

dato un testo (stringa), che contiene uno scritto in lingua italiana, determinare il numero di **parole** contenute nel testo (senza segni di interpunzione)

dati di input: la stringa (array **testo**), la sua lunghezza (variabile **m**)

dato di output: il numero di parole nel testo (variabile **numero_parole**)

costrutto ripetitivo: 1 ciclo **for**

operazione ripetuta (al generico passo **i**):

considerare l'**i**-simo carattere della stringa **testo**, determinare se è iniziale di una parola e incrementare opportunamente i contatori

una **parola** è una sottostringa racchiusa tra due caratteri spazio



testo

n	e	l		m	e	z	z	o			d	e	l		
---	---	---	--	---	---	---	---	---	--	--	---	---	---	--	--

c	a	m	m	i	n		d	i		n	o	s	t	r	a
---	---	---	---	---	---	--	---	---	--	---	---	---	---	---	---

v	i	t	a		m	i		r	i	t	r	o	v	a	i
---	---	---	---	--	---	---	--	---	---	---	---	---	---	---	---

testo



n	e	l		m	e	z	z	o			d	e	l		
c	a	m	m	i	n		d	i		n	o	s	t	r	a
v	i	t	a		m	i		r	i	t	r	o	v	a	i

false

in_parola

0

numero_parole

testo



n	e	l		m	e	z	z	o			d	e	l		
c	a	m	m	i	n		d	i		n	o	s	t	r	a
v	i	t	a		m	i		r	i	t	r	o	v	a	i

true

in_parola

1

numero_parole

testo

i
↓

n	e	l		m	e	z	z	o			d	e	l		
c	a	m	m	i	n		d	i		n	o	s	t	r	a
v	i	t	a		m	i		r	i	t	r	o	v	a	i

false

in_parola

1

numero_parole

testo



n	e	l		m	e	z	z	o			d	e	l	
---	---	---	--	---	---	---	---	---	--	--	---	---	---	--

c	a	m	m	i	n		d	i		n	o	s	t	r	a
---	---	---	---	---	---	--	---	---	--	---	---	---	---	---	---

v	i	t	a		m	i		r	i	t	r	o	v	a	i
---	---	---	---	--	---	---	--	---	---	---	---	---	---	---	---

true

in_parola

2

numero_parole

testo



n	e	l		m	e	z	z	o			d	e	l		
c	a	m	m	i	n		d	i		n	o	s	t	r	a
v	i	t	a		m	i		r	i	t	r	o	v	a	i

false

in_parola

2

numero_parole

testo



n	e	l		m	e	z	z	o			d	e	l		
c	a	m	m	i	n		d	i		n	o	s	t	r	a
v	i	t	a		m	i		r	i	t	r	o	v	a	i

false

in_parola

2

numero_parole

testo



n	e	l		m	e	z	z	o		d	e	l			
c	a	m	m	i	n		d	i		n	o	s	t	r	a
v	i	t	a		m	i		r	i	t	r	o	v	a	i

true

in_parola

3

numero_parole

```
int conta_parole(char testo[], int m) {
    int i, numero_parole;
    logical in_parola;
    char c;
    numero_parole = 0;
    in_parola = false;
    for (i=0; i < m; i++) {
        c = testo[i] ;
        if (c = ' ')
            { in_parola = false ; }
        else if ( !in_parola ) {
            in_parola = true ;
            numero_parole = numero_parole + 1 ;
        }
    }
    return numero_parole ;
}
```

ATTENZIONE: da modificare in C

i-simo carattere della stringa **testo**

problema: (**string matching**)

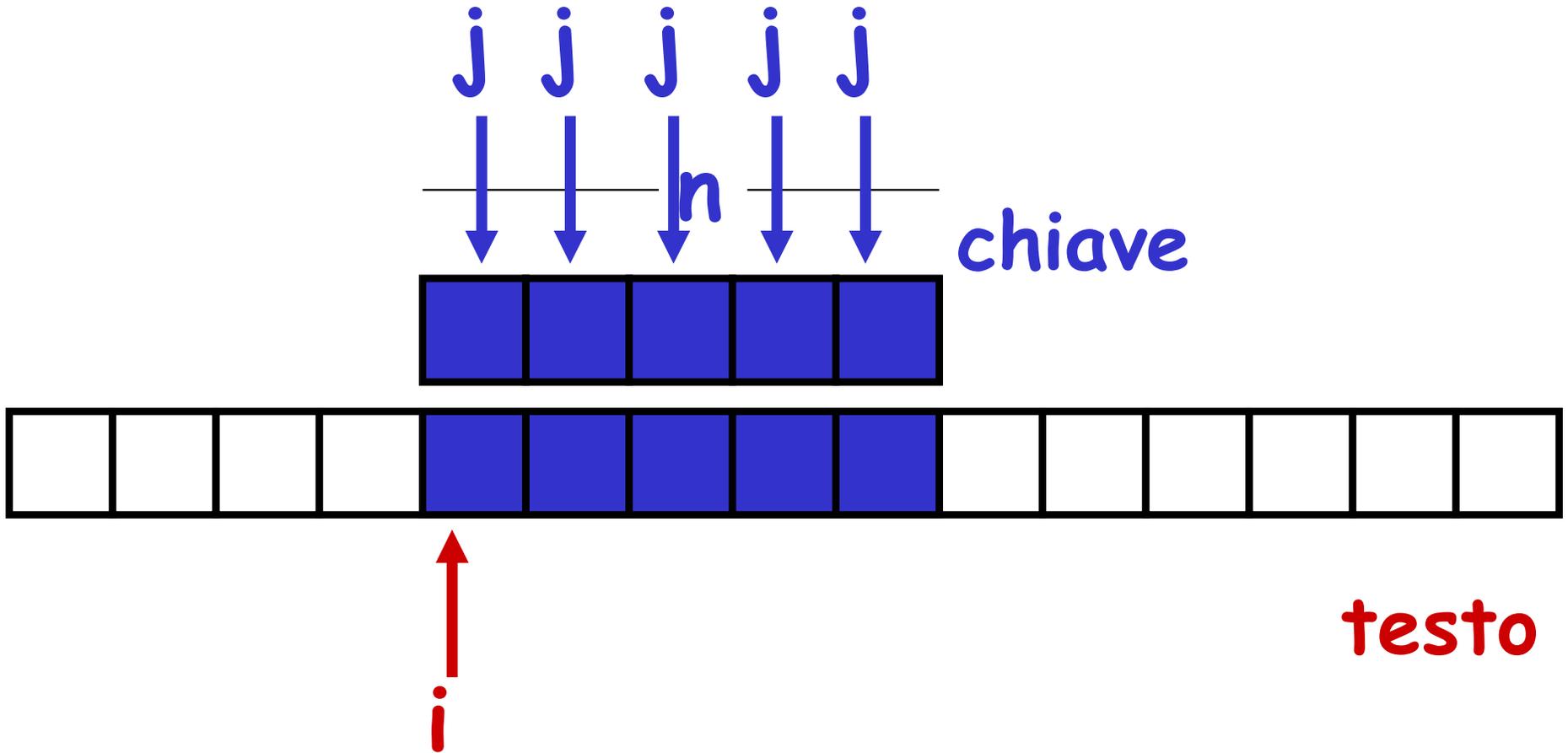
calcolo del numero di volte in cui una stringa data (chiave) compare come sottostringa in un'altra stringa data (testo)

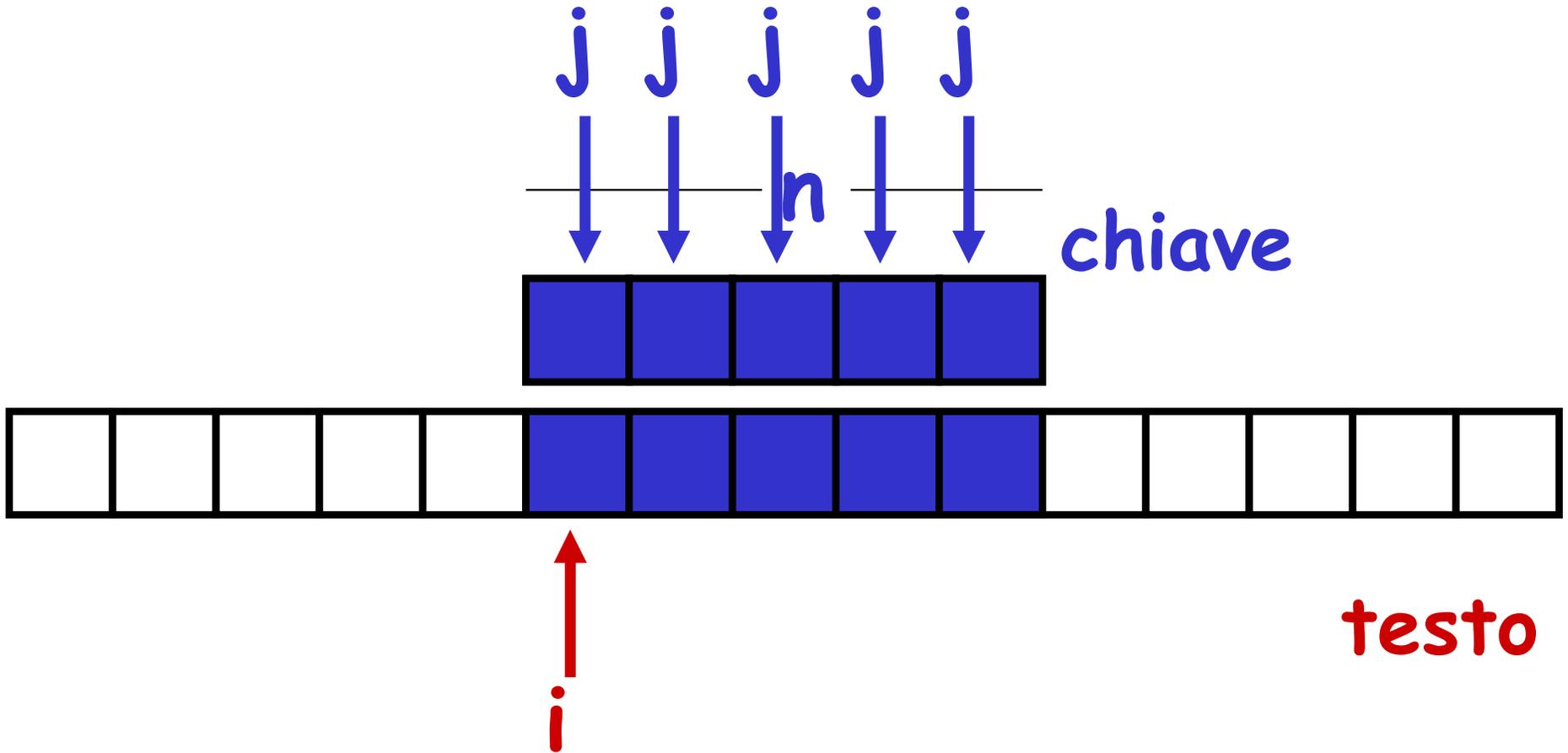
dati di input: la stringa chiave (array) **chiave**, la sua lunghezza (variabile **n**), la stringa testo (array **testo**), la sua lunghezza (variabile **m**)

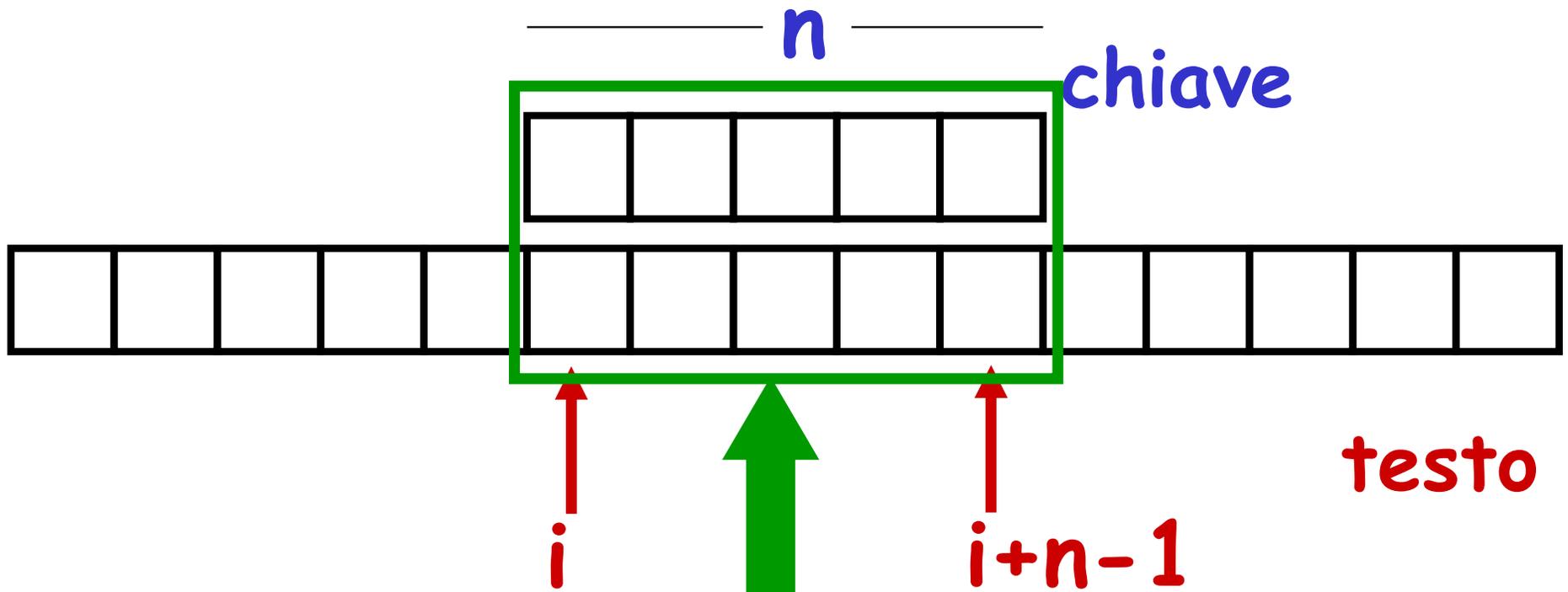
dato di output: numero di occorrenze (variabile **conta_chiave**)

costrutto ripetitivo: 1+1 cicli innestati

operazione ripetuta (al generico passo **i**, ciclo **for** esterno): confrontare la stringa **chiave** e la sottostringa (inizio **i** e lunghezza **n**) della stringa **testo**







determinare se queste due
stringhe sono uguali

```
logical uguale_stringhe (char a[], char b[], int n) {  
    int i;  
    logical uguale;  
    i = 0 ;  
    uguale = true;  
    while (uguale && i < n) {  
        if (a[(i) != b[(i)])  
            { uguale = false;}  
        i = i+1;  
    }  
    return uguale;  
}
```

n confronti
(al più)
(tra caratteri delle due
stringhe)

è una variante della function **uguaglianza_array**

```
int string_matching(char chiave[], int n, char testo[],
                  int m) {
    int i, conta_chiave;
    conta_chiave = 0 ;
    for (i=0; i <= m-n; i++) {
        if (uguale_stringhe(chiave, testo+i, n) )
            { conta_chiave = conta_chiave + 1; }
    }
    return conta_chiave ;
}
```

sottostringa di
inizio i della
stringa **testo**

notazione di
porzione di array

$n*(m-n+1)$
confronti
(al più)
(tra caratteri delle due
stringhe)

problema: (**matching migliore**)

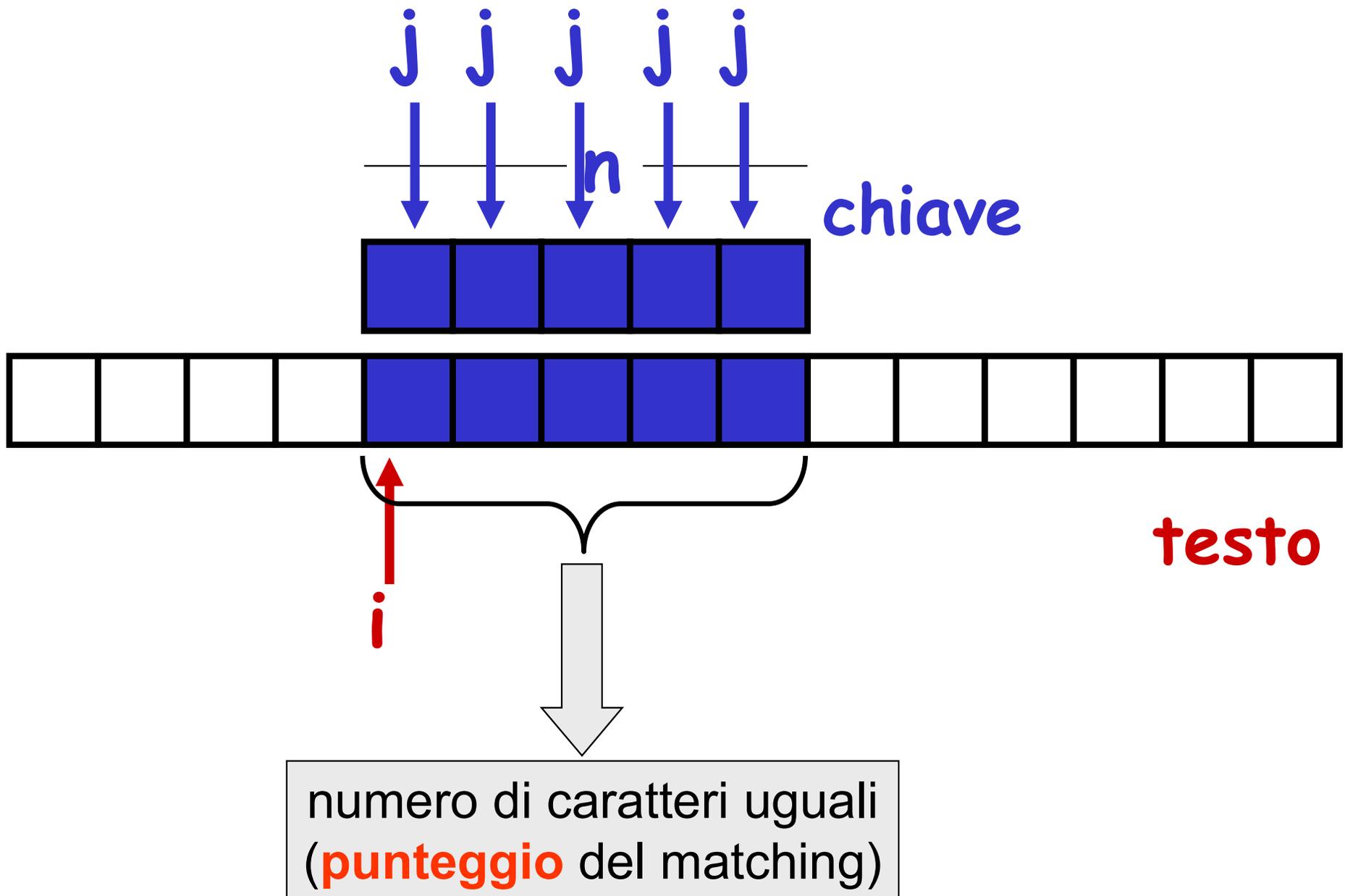
determinare la sottostringa di una stringa data (testo) che ha il maggior numero di caratteri (di ugual posto) uguali a quelli di un'altra stringa data (chiave)

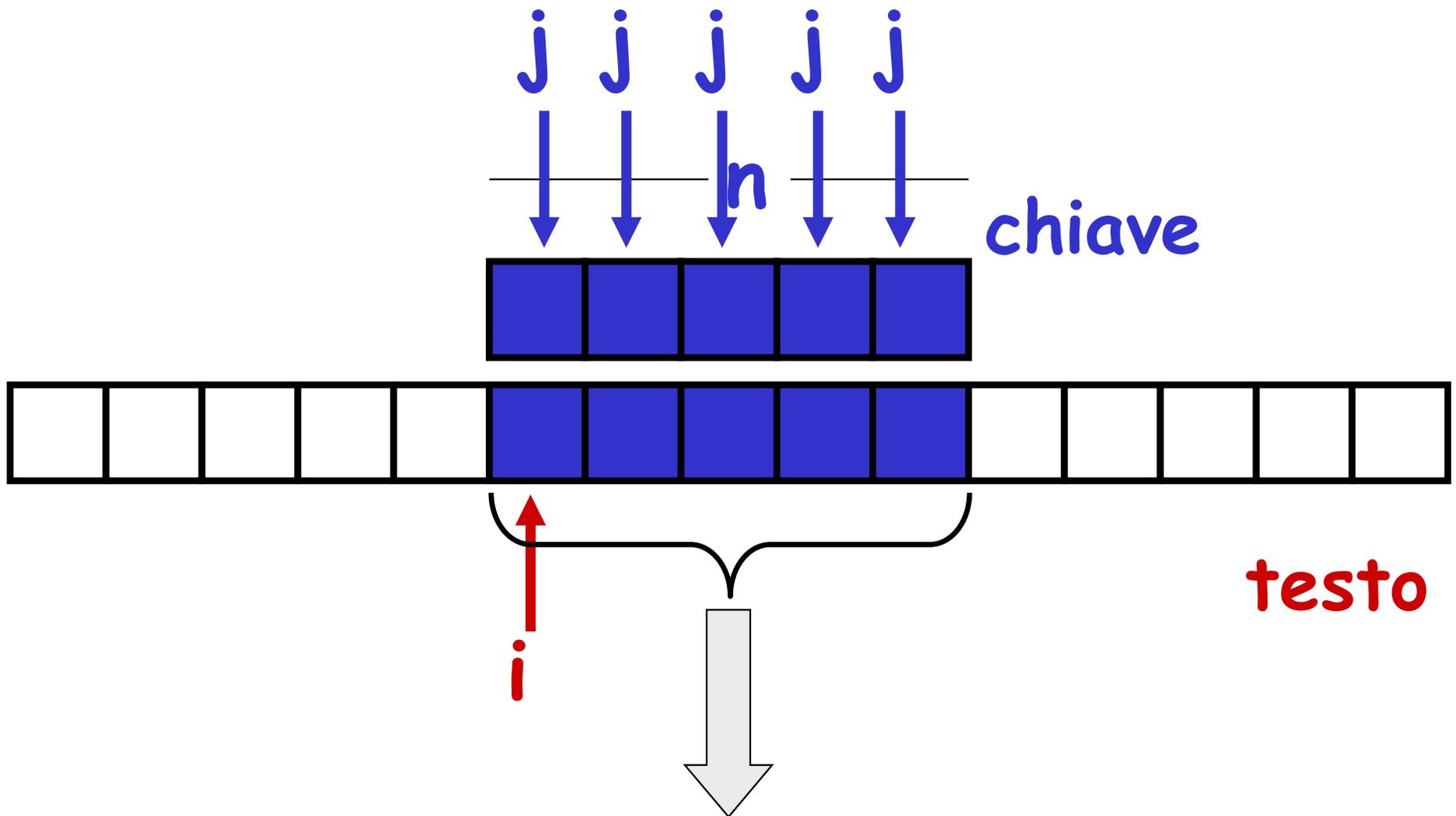
dati di input: la stringa chiave (array **chiave**) la sua lunghezza (variabile **n**), la stringa testo (array **testo**), la sua lunghezza (variabile **m**)

dato di output: l'indice della sottostringa di matching migliore (variabile **indice**)

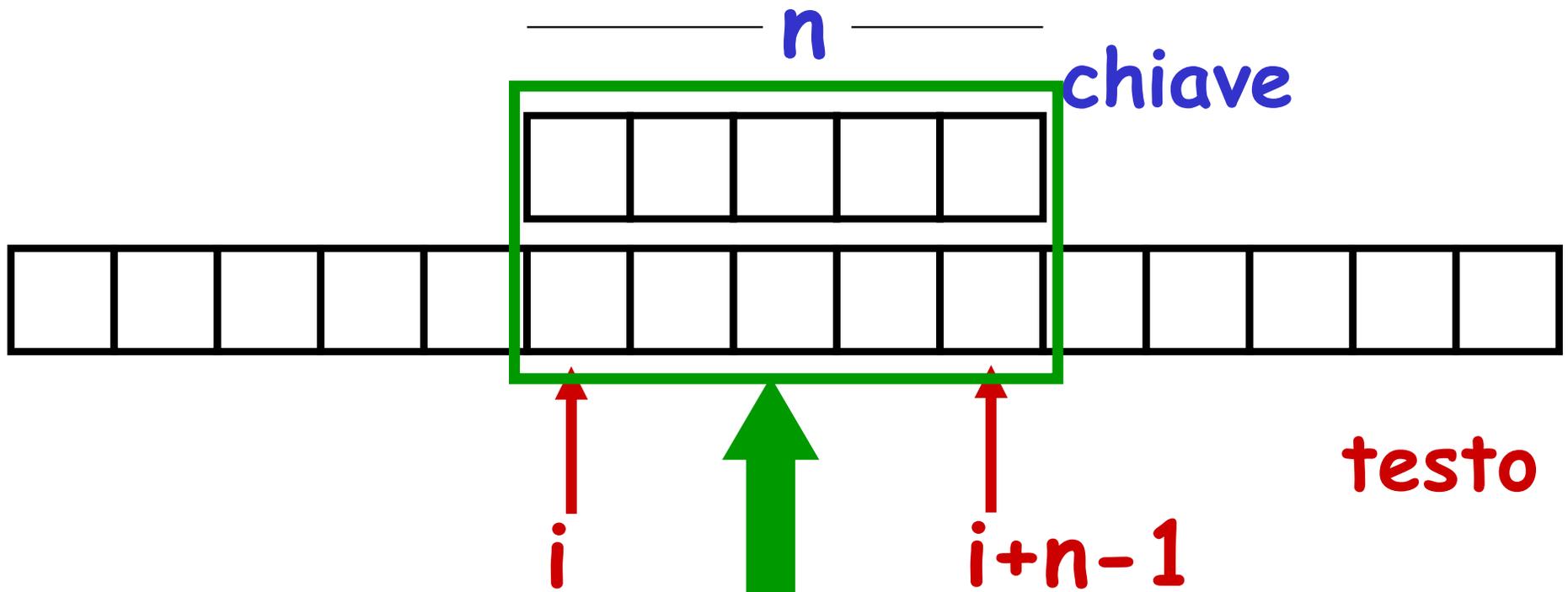
costrutto ripetitivo: 1+1 cicli **for** innestati

operazione ripetuta (al generico passo **i**, ciclo **for** esterno): determinare il punteggio del *matching* tra la stringa **chiave** e la sottostringa (inizio **i** e lunghezza **n**) della stringa **testo**





numero di caratteri uguali
(**punteggio** del matching)



determinare il numero di
caratteri uguali di queste
due stringhe

```
int punteggio_matching(char a[], char b[], int n) {  
    int i, n_caratteri_uguali;  
    n_caratteri_uguali = 0;  
    for (i=0; i < n; i++) {  
        if (a[i] == b[i] )  
            { n_caratteri_uguali = n_caratteri_uguali +1; }  
    }  
    return n_caratteri_uguali ;  
}
```

n

confronti

(tra caratteri delle
due stringhe)

```
int matching_migliore(char chiave[], int n, char testo[],
                    int m) {
    int i, punteggio_max, punteggio, indice;
    punteggio_max = 0 ;
    for (i=0; i <= m-n; i++) {
        punteggio = punteggio_matching(chiave, testo+i, n) ;
        if (punteggio > punteggio_max)
        {
            punteggio_max = punteggio ;
            indice = i ;
        }
    }
    return indice ;
}
```

$n \cdot (m - n + 1)$
confronti
(tra caratteri delle
due stringhe)

sottostringa di
inizio i della
stringa **testo**

