

# Informatica per le Biotecnologie

## Algoritmica Lezione 5

### 1. Programmazione dinamica

Paradigma per la costruzione di algoritmi **alternativo alla ricorsività**, utilizzato se il problema ammette una definizione ricorsiva ma la sua trasformazione in un algoritmo causa ripetizioni degli stessi calcoli da parte di diverse chiamate ricorsive.

RICERCA-BINARIA e MERGE-SORT comprendono due chiamate ricorsive su sottoinsiemi disgiunti di circa  $n/2$  dati nella loro *formulazione*, e in ogni *esecuzione* il primo ne richiede una sola e il secondo ne richiede due.

I due algoritmi sono molto efficienti perché le chiamate ricorsive agiscano su *sottoinsiemi disgiunti* dei dati e l'algoritmo non ripete operazioni già eseguite sugli stessi dati. Vediamo due esempi in cui ciò non si verifica.

Il calcolo dei *numeri di Fibonacci* e l'allevamento dei conigli

$F_i$  è il numero di coppie presenti al tempo  $i$ .

L'allevamento è inizialmente vuoto ( $F_0 = 0$ );

vi si introduce una coppia appena nata al tempo

$i = 1$  ( $F_1 = 1$ ); questa diviene fertile e si accop-

pia al tempo  $i = 2$  ( $F_2 = 1$ ); e ne genera un'altra

al tempo  $i = 3$  ( $F_3 = 2$ ).

$F_i$  è pari al numero di coppie presenti al tempo precedente  $i - 1$ , più le nuove nate che sono tante quante ne esistevano al tempo  $i - 2$ .

$$F_0 = 0, \quad F_1 = 1, \quad F_i = F_{i-1} + F_{i-2}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, .....

La crescita è **esponenziale in  $i$** . All'inizio è piuttosto lenta, poi i valori aumentano sempre più velocemente. Si ha  $F_i \approx \frac{1}{\sqrt{5}}\Phi^i$ , ove  $\Phi = \frac{1+\sqrt{5}}{2} \approx 1,62$  è detto *sezione aurea*.

Algoritmo ricorsivo per calcolare  $F_i$  in funzione di  $i$  applicando la definizione ricorsiva:

**FIB**( $i$ )

if ( $i == 0$ ) return 0;

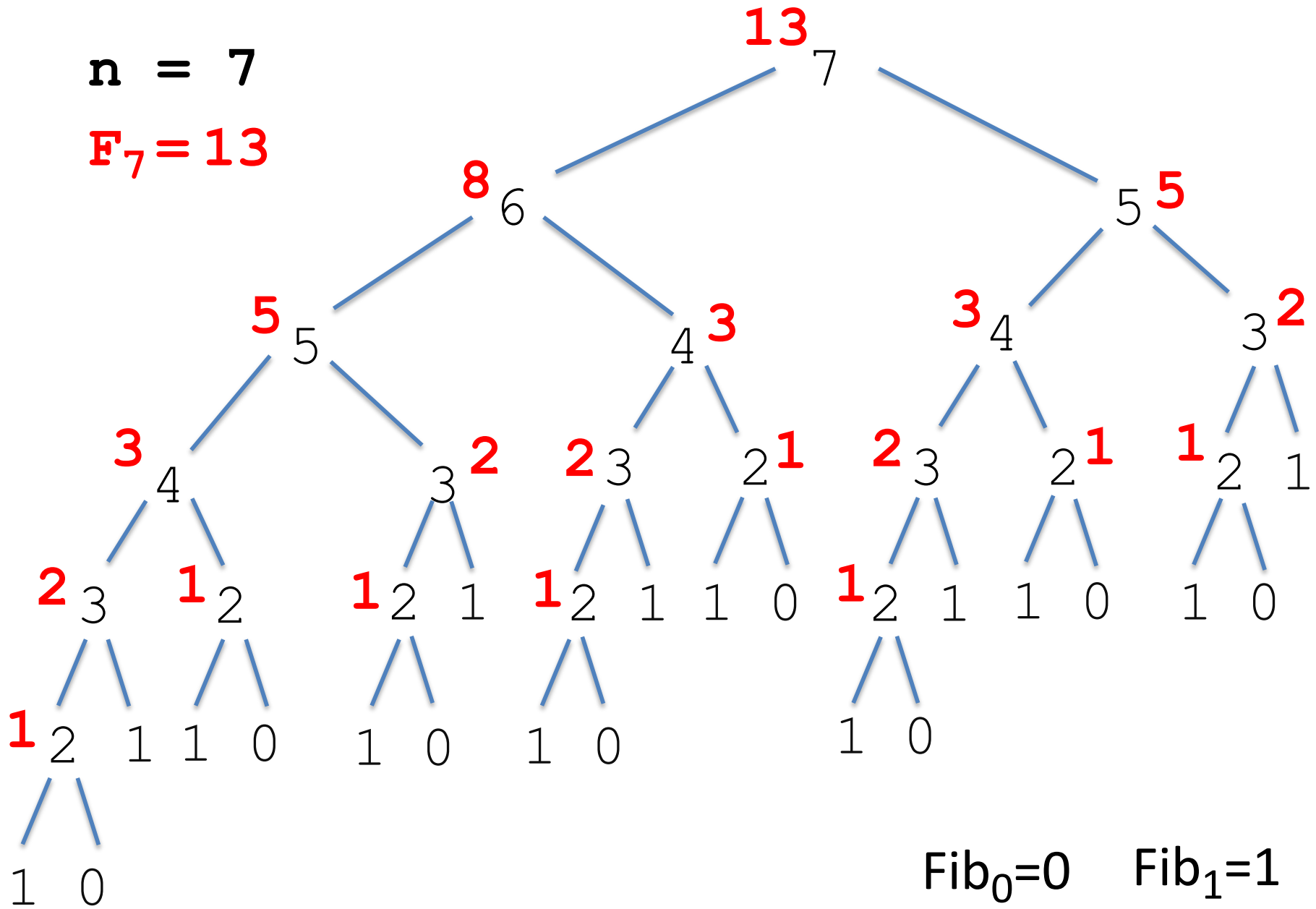
if ( $i == 1$ ) return 1;

return ( **FIB**( $i - 1$ ) + **FIB**( $i - 2$ ) );

$F_n$  si calcola attraverso la chiamata **FIB**( $n$ ).

**n = 7**

**$F_7 = 13$**





$$T(0) = c_0, \quad T(1) = c_1, \quad \text{con } c_0 \text{ e } c_1 \text{ costanti}$$

$$T(n) = T(n-1) + T(n-2) + c_2, \quad \text{con } c_2 \text{ costante}$$

$$T(n) > 2T(n-2) + c_2$$

Vedi dispensa

$$> 2(2T(n-4) + c_2) + c_2 = 2^2 T(n-4) + 3c_2$$

$$> 2^2(2T(n-6) + c_2) + 3c_2 = 2^3 T(n-6) + 7c_2$$

$$> \dots$$

$$> 2^k T(n-2k) + (2^k - 1)c_2 > \dots$$

dove  $T(n-2k)$  tende a  $T(0)$  per  $n$  pari e a  $T(1)$

per  $n$  dispari, quindi  $T(n) > c' 2^{(n-1)/2}$

Con la **Programmazione Dinamica** si conservano valori già calcolati per utilizzarli di nuovo se ciò è richiesto

Il calcolo di  $F_n$  è semplicissimo: si costruiscono a uno a uno gli elementi della successione fino all'ennesimo, partendo da **0, 1** e calcolando ogni elemento come la somma dei due precedenti.

$$F_0 = 0, \quad F_1 = 1, \quad F_i = F_{i-1} + F_{i-2}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, .....

**Il tempo richiesto è di ordine  $\theta(n)$**



Un problema simile è quello del calcolo dei **coefficienti binomiali**  $\binom{n}{m}$  per due interi  $0 \leq m \leq n$

Per un insieme arbitrario,  $\binom{n}{m}$  è il numero di **combinazioni** di  $n$  elementi in gruppi di  $m$ .

Per l'insieme  $\{a, b, c, d, e\}$  vi sono i  $\binom{5}{2} = 10$  gruppi di due elementi :  $ab, ac, ad, ae, bc, bd, be, cd, ce, de$

# Il Triangolo di Tartaglia

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

$\binom{5}{2} = 10$  è l'elemento in riga 5 e colonna 2.

Rappresentiamo i sottoinsiemi di  $\{a,b,c,d,e\}$  con un vettore di appartenenza **V**. Per i sottoinsiemi di **2** elementi:

	0	1	2	3	4	
A	a	b	c	d	e	
V	1	1	0	0	0	ab
V	1	0	1	0	0	ac
V	1	0	0	1	0	ad
V	1	0	0	0	1	ae
V	0	1	1	0	0	bc
V	0	1	0	1	0	bd
V	0	1	0	0	1	be
V	0	0	1	1	0	cd
V	0	0	1	0	1	ce
V	0	0	0	1	1	de

$$\binom{5}{2} = 10$$

La somma su ogni riga  $i$  è  
pari al numero di sottoinsiemi  
di un insieme di  $i$  elementi

quindi tale somma è  $2^i$

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

1

2

4

8

16

32

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

$$\binom{n}{m} = 1 \quad \text{per } m = 0 \text{ e } m = n$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

per  $0 < m < n$

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

$$\binom{n}{m} = 1 \quad \text{per } m = 0 \text{ e } m = n$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

per  $0 < m < n$



Un algoritmo ricorsivo basato sulla formula

**BIN**( $i, j$ )

**if** ( $j == 0$ ) **return** 1;

**if** ( $i == j$ ) **return** 1;

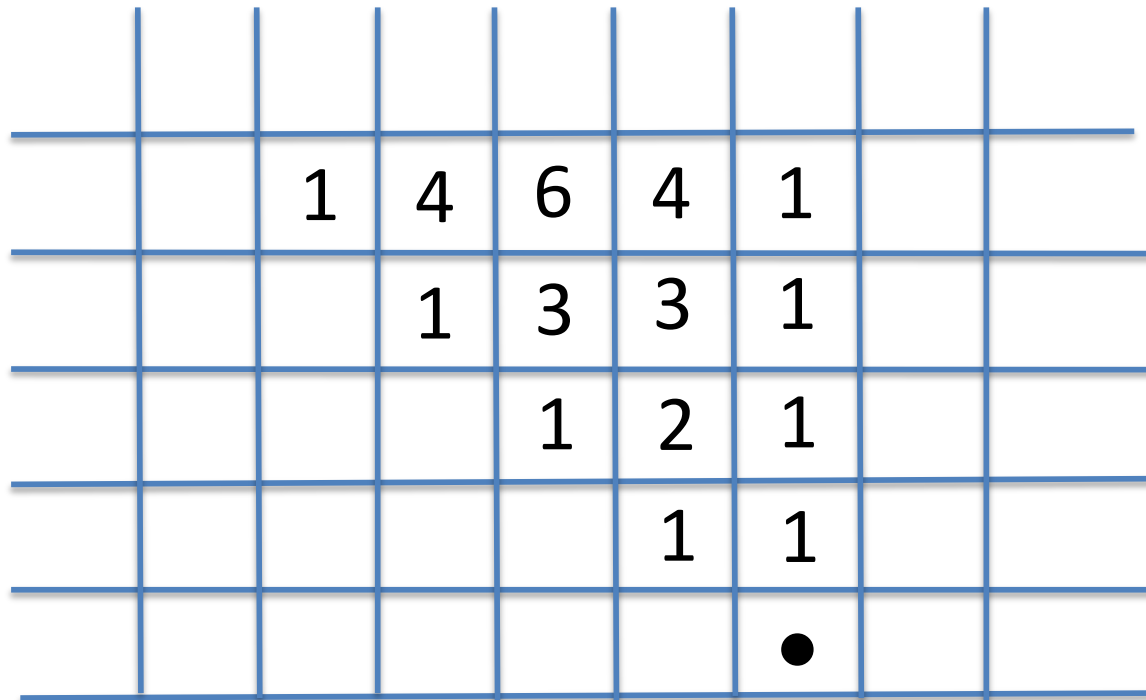
**return** ( **BIN**( $i - 1, j$ ) + **BIN**( $i - 1, j - 1$ ) );

$\binom{n}{m}$  si calcola con la chiamata **BIN**( $n, m$ )

$T(n, 0) = c_0, T(n, n) = c_1,$  **per qualsiasi**  $n$

$T(n, m) = T(n-1, m) + T(n-1, m-1) + c_2,$  **per**  $0 < m < n$

Si può controllare espandendo la formula che  $T(n,m)$  ha valore esponenziale in  $m$



	1	4	6	4	1	
		1	3	3	1	
			1	2	1	
				1	1	
					•	

questi numeri rappresentano quante volte si ricalcola uno stesso elemento

Allocando il triangolo in una matrice  $M$ , possiamo calcolare  $M[n,m]$  per righe: ogni elemento della riga  $i$  è la somma di due elementi della riga  $i-1$  precedentemente memorizzati.

$M$

		$m$						
$i$	$j$	0	1	2	3	4	5	6
	0	1						
1	1	1	1					
2	2	1	2	1				
3	3	1	3	3	1			
4	4	1	4	6	4	1		
5	5	1	5	10	10	5	1	
6	6	1	6	15	20	15	6	1

$$T(n,m) = \theta(nm)$$

## 2. Confronto tra sequenze

Molti problemi di **confronto fra sequenze** si incontrano nella elaborazione di testi e nella **biologia molecolare**.

## ALLINEAMENTO TRA SEQUENZE

A	T	T	G	C		C	C	A
	G	T	G	A	A	C		A

Iniziamo con lo studio del problema di EDIT DISTANCE il cui schema algoritmico sarà poi impiegato per risolvere molti altri problemi:

Date due sequenze di caratteri  $X, Y$   
trovare un allineamento tra di esse di  
minima distanza (*edit distance*) tra  $X$  e  $Y$



- nelle due sequenze possano essere inseriti spazi indicati con -

```
A T T G C - C C A
- G T G A A C - A
```

Lo spazio indica la perdita di un carattere in una sequenza o l'inserzione di un nuovo carattere in posizione corrispondente nell'altra sequenza

- la *distanza* è la somma delle distanze tra coppie di caratteri o spazi, uno in X e l'altro in Y, in posizioni corrispondenti nell'allineamento, dove:

caratteri uguali (**match**) hanno **distanza 0**

caratteri diversi (**mismatch**) hanno **distanza 1**

carattere e spazio allineati hanno **distanza 1**

- non si considerano spazi in posizioni corrispondenti di X,Y

A L B E R O  
 L A B B R O  
 • • •

A L - B E R O  
 - L A B B R O  
 • • •

- A L B E R O  
 L A B B - R O  
 • • •

- A L B E R O  
 L A - B B R O  
 • • •

Quattro allineamenti ottimi  
 con Edit Distance uguale a 3

$$\mathbf{X} = \mathbf{x}_1 \mathbf{x}_2 \dots \mathbf{x}_n, \quad \mathbf{Y} = \mathbf{y}_1 \mathbf{y}_2 \dots \mathbf{y}_m$$

In una matrice **M** di  
dimensioni  $(n+1) \times (m+1)$   
i caratteri di **X** e **Y**  
corrispondono alle  
righe tra 1 e n e alle  
colonne tra 1 e m

		L	A	B	B	R	O
	Ø	1	2	3	4	5	6
A	1						
L	2						
B	3						
E	4						
R	5						
O	6						

**M**

$M[i,j]$  indica la edit distance  
 tra il prefisso  $x_1 x_2 \dots x_i$  di  $X$   
 e il prefisso  $y_1 y_2 \dots y_j$  di  $Y$

							∅	L	A	B	B	R	O	
							∅	∅	1	2	3	4	5	6
x <sub>1</sub> x <sub>2</sub> x <sub>3</sub> x <sub>4</sub>	A	L	-	B	E									
y <sub>1</sub> y <sub>2</sub> y <sub>3</sub>	-	L	A	B	-		A	1						
	•		•		•		L	2						
M[4,3] = 3							B	3						
							E	4			3			
							R	5						
							O	6						3

M[n,m] = 3 è la  
edit distance  
tra X e Y

La riga e la colonna  $\emptyset$  corrispondono  
a prefissi vuoti, cioè X e Y non sono  
ancora stati esaminati

							∅	L	A	B	B	R	O	
							∅	∅	1	2	3	4	5	6
x <sub>1</sub> x <sub>2</sub> x <sub>3</sub> x <sub>4</sub>	A	L	B	E			A	1						
(Y)	—	—	—	—			L	2						
	•	•	•	•			B	3						
M[4,0]	=	4					E	<u>4</u>						
							R	5						
							O	6						



# Come si calcola la matrice?

	<b>Y</b>	L	A	B	<u>B</u>	R	O
<b>X</b>	0	1	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3

riga e colonna  $\emptyset$

$$M[\emptyset, j] = j \quad \text{per } \emptyset \leq j \leq m$$

$$M[i, \emptyset] = i \quad \text{per } \emptyset \leq i \leq n$$

detta  **$p(i, j)$**  la distanza tra  $x_i$  e  $y_j$

$p(i, j) = 0$  per  $x_i = y_j$  (match)

$p(i, j) = 1$  per  $x_i \neq y_j$  (mismatch)

carattere-spazio hanno distanza = 1

$$M[i,j] = \min\{M[i,j-1]+1, \\ M[i-1,j]+1, \\ M[i-1,j-1]+p(i,j)\}$$

per  $1 \leq i \leq n, 1 \leq j \leq m$

Mossa orizzontale

Mossa verticale

Mossa diagonale

	∅	L	A	B
∅	∅	1	2	3
A	1	1	1	2
L	2	1	2	2
B	3	2	2	2
E	4	3	3	3

	Ø	L	A	B	B	R	O
Ø	Ø	1	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3

L'algoritmo descritto a parole segue lo schema di programmazione dinamica:

ha **complessità quadratica**  $\Theta(n\ m)$  perché costruisce una matrice  $M$  di dimensioni  $(n+1) \times (m+1)$ , e il valore in ciascuna cella è calcolato in tempo costante;

il suo programma ED impiega due vettori  $X$ ,  $Y$  di dimensioni  $n+1$  e  $m+1$  che contengono le due sequenze a partire dalla cella 1 (la cella 0 non è utilizzata).

ED (X, Y)

```
for (i=0, i≤ n, i++) M[i, 0]=i;
for (j=0, j≤ m, j++) M[0, j]=j;
for (i=1, i≤ n, i++)
    for (j=1, j≤ m, j++)
        { if (X[i]=Y[j]) p=0 else p=1;
          M[i, j]= min (M[i-1, j]+1,
                        M[i-1, j-1]+p,
                        M[i, j-1]+1); }

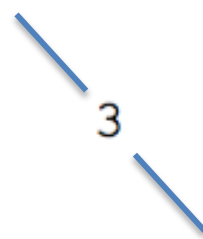
return M[n, m];
```



Gli **allineamenti** per la edit distance  
 si ricostruiscono **risalendo nella matrice**  
 dalla casella  $M[n,m]$  alla  $M[\emptyset,\emptyset]$ :

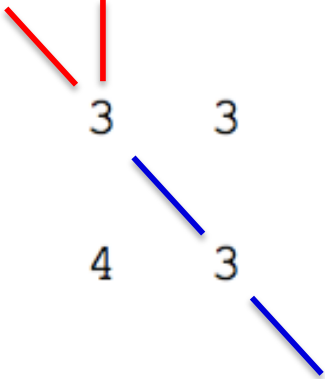
da ogni casella  $M[i,j]$  si risale a una  
 tra le tre che la precedono, invertendo  
 la regola usata per costruire  $M[i,j]$ .

			B	B	R	O
	.	.	.	.	.	.
B	.	2	2	2	3	4
E	.	3	3	3	3	4
R	.	4	4	4	3	4
O	.	5	5	5	4	3



Nella costruzione vi possono presentarsi  
**più alternative** relative ad allineamenti  
di uguale edit distance.

			B	B	R	O
	.	.	.	.	.	.
B	.	2	2	2	3	4
E	.	3	3	3	3	4
R	.	4	4	4	3	4
O	.	5	5	5	4	3



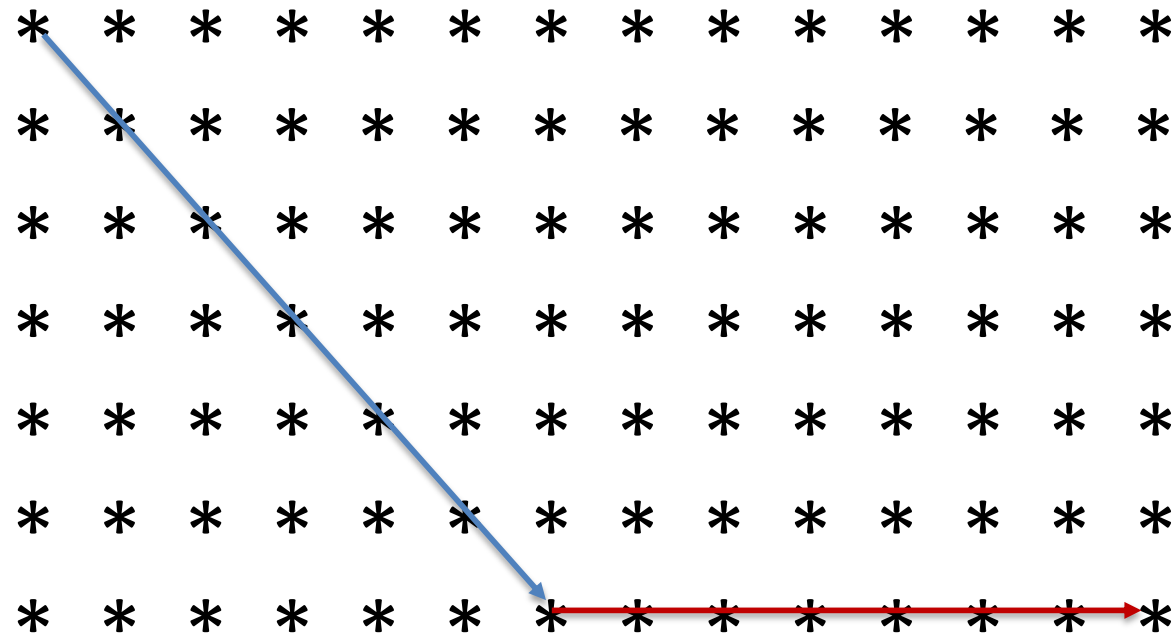
	∅	L	A	B	B	R	O
∅	∅	<u>1</u>	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3

Quali valori limite può assumere  
l'edit distance in funzione di  $n, m$  ?

A shortest path

$m=13$

$n=6$



$$|n-m| = 7$$

**Il valore della edit distance è limitato tra  $|n-m|$  e  $\max(n,m)$ .**

- ogni percorso tra  $M[0,0]$  e  $M[n,m]$  deve includere almeno  $|n-m|$  passi orizzontali o verticali (limite inferiore);
- un percorso con numero minimo di passi e contenente solo spazi o mismatch deve includere anche  $\min(n,m)$  passi diagonali: in totale  $|n-m| + \min(n,m) = \max(n,m)$  (limite superiore).

Gli allineamenti ottimi possono essere moltissimi. Per costruirne uno solo l'algoritmo richiede tempo  $\Theta(n+m)$

Il programma ALLINEA usa i vettori  $X$ ,  $Y$  e la matrice  $M$  prodotta da  $ED$ , e due vettori  $ALX[0:k]$ ,  $ALY[0:k]$  con  $k=\max(n,m)$ , che conterranno nelle ultime celle le due sequenze allineate. Per esempio:

	0	1	.	.	.	.	.	.	.	.	k
ALX	.	.	.	.	A	L	-	B	E	R	O
ALY	.	.	.	.	-	L	A	B	B	R	O

$X$ ,  $Y$  e  $M$  sono i dati in ingresso.

$ALX$ ,  $ALY$  e  $h$  sono i dati in uscita.

$h$  indica la posizione nei due vettori da cui comincia l'allineamento perché in genere essi non sono completamente utilizzati

In caso di scelte multiple l'algoritmo dà precedenza a **match-mismatch**, poi a **spazio in  $X$** , infine a **spazio in  $Y$**



ALLINEA (X, Y, M)

k=max(n,m); h=k; i=n; j=m;

while ((i>0) or (j>0))

{if (((i>0) and (j>0))

and ((M[i,j]==M[i-1,j-1]) && (X[i]==Y[j]))

or ((M[i,j]==M[i-1,j-1]+1) and (X[i]!=Y[j])))

{ALX[h]=X[i]; ALY[h]=Y[j]; i=i-1; j=j-1;}

else {if ((j>0) and (M[i,j]==M[i,j-1]+1))

{ALX[h]= -; ALY[h]=Y[j]; j=j-1;}

else if (i>0) //deve essere M[i,j]==M[i-1,j]+1

{ALX[h]=X[i]; ALY[h]= -; i=i-1;}

}

h=h-1;

}

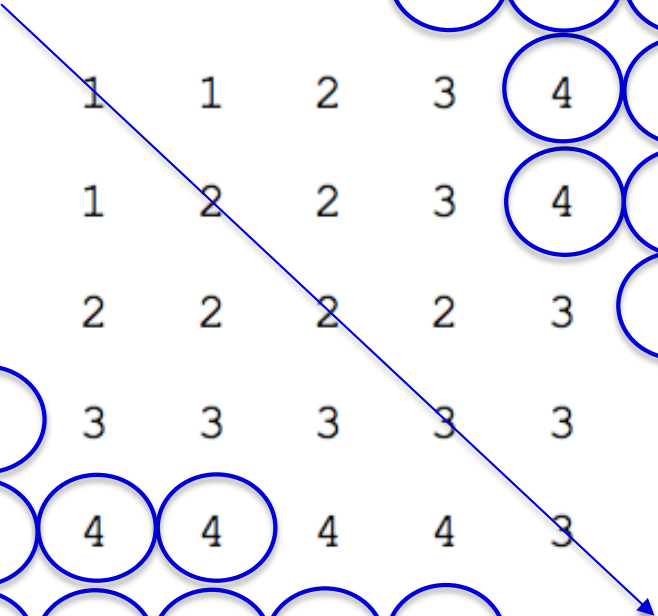
return h;

È possibile che nuovi valori per la distanza tra singoli caratteri o spazi siano più significativi di 0, 1, in dipendenza dal campo di applicazione considerato.

Il calcolo può essere immediatamente adattato inserendo tali nuovi valori nelle relazioni di ricorrenza e negli algoritmi ED e ALLINEA

# Una considerazione conclusiva sulla edit distance

	∅	L	A	B	B	R	O
∅	∅	1	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3



Ricordiamo che la edit distance è  $\leq \max(n,m) = 6$

Tutti questi valori possono essere ignorati nel calcolo della edit distance

	∅	L	A	B	B	R	O
∅	∅	1	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3

Potrei calcolare solo i  
valori in una banda larga  
 $n+1$  attorno alla diagonale  
(o larga  $n$  per  $n$  dispari)

ma la complessità  
dell'algo-ritmo sarebbe  
sempre quadratica

	∅	L	A	B	B	R	O
∅	∅	1	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3

Ma se l'edit distance tollerata fosse  $\leq k$  costante (per es.  $k=3$ ) la banda avrebbe larghezza costante e la complessità diverrebbe  $O(nk)$ , cioè lineare