

6.6 Sequence Alignment



Programmazione dinamica: caratteristiche

1. La soluzione al problema originale si può ottenere da soluzioni a **sottoproblemi**
2. Esiste una **relazione di ricorrenza** per la funzione che dà il valore ottimo ad un sottoproblema

3. Le soluzioni ai sottoproblemi sono calcolate una sola volta e via via memorizzate in una **tabella**

Due implementazioni possibili:

- Con annotazione (*memoized*) o *top-down*
- Iterativa o *bottom-up*

Dynamic Programming Summary

Recipe.

Characterize structure of problem.

Recursively define value of optimal solution.

Compute value of optimal solution.

Construct optimal solution from computed information.

Dynamic programming techniques.

Binary choice: weighted interval scheduling, **sequence alignment**

Multi-way choice: segmented least squares, esempio "canoa"

Adding a new variable: knapsack.

Dynamic programming over intervals: RNA secondary structure.

Top-down vs. bottom-up: different people have different intuitions.

E' capitato anche a voi?

Di digitare sul computer una parola in maniera sbagliata (per esempio usando un dizionario sul Web):

AGORITNI

E sentirsi chiedere:

«Forse cercavi **ALGORITMI**?»

Come fanno a capirlo? Sanno veramente cosa abbiamo in mente?

Non trovando AGORITNI sul vocabolario ha cercato una parola «**simile**», «**vicina**» presente nel vocabolario.

String Similarity

How similar are two strings?

ocurrance

occurrence

o c - u r r a n c e

o c c u r r e n c e

1 mismatch, 1 gap

o c u r r a n c e -

o c c u r r e n c e

6 mismatches, 1 gap

o c - u r r - a n c e

o c c u r r e - n c e

0 mismatches, 3 gaps

Edit Distance

Applications.

Basis for Unix diff

spam filter

Speech recognition

Computational biology (sequenze di simboli nel DNA rappresentano proprietà degli organismi)

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

Gap penalty δ ; mismatch penalty α_{pq} (you may assume $\alpha_{pp}=0$).

Cost = sum of gap and mismatch penalties.

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

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

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

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

$$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$$

$$2\delta + \alpha_{CA}$$

Edit Distance

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

Gap penalty δ ; mismatch penalty α_{pq} (you may assume $\alpha_{pp}=0$).

Cost = sum of gap and mismatch penalties.

Esempio:

$$\delta = 3$$

$\alpha_{AT}=\alpha_{CG}=3$; gli altri mismatch penalità =1 (tranne $\alpha_{pp}=0$).

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

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

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

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

$$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA} = 5$$

$$2\delta + \alpha_{CA} = 7$$

Sequence Alignment

Goal: Given two strings $X = x_1 x_2 \dots x_m$ and $Y = y_1 y_2 \dots y_n$ find alignment of minimum cost.

Def. An **alignment** M is a set of ordered pairs $x_i - y_j$ such that each item occurs in at most one pair and no crossings.

Def. The pair $x_i - y_j$ and $x_{i'} - y_{j'}$ **cross** if $i < i'$, but $j > j'$.

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i: x_i \text{ unmatched}} \delta + \sum_{j: y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Ex: CTACCG **VS** TACATG.

Sol: $M = x_2 - y_1, x_3 - y_2, x_4 - y_3, x_5 - y_4, x_6 - y_6$.

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G

-	T	A	C	A	T	G
	y_1	y_2	y_3	y_4	y_5	y_6

Come risolverlo?

Ricerca esaustiva: proviamo tutti gli allineamenti possibili?

Proviamo con l'approccio della **programmazione dinamica**.

Cominciamo dal basso e poi guardiamo una soluzione ottimale.

Cominciamo «dal basso»

X = ALGORITMI

Y = AGORITNI

costo mismatch = 4 , costo gap = 2

A	A	A	-	-	A
A	A	-	A	A	-

$$\min \{ 0, 2+2, 2+2 \} = 0$$

A	-	A	A	-	A	-	-	-	-	A	-
AG	A	G	A	G	-	A	G	A	-	G	G

$$\min \{ 2+4, 0+2, 2+2+2, 2+2+2, 2+2+2 \} = 2$$

-	-	A
A	G	-

Altre possibilità?

Come classifico le varie possibilità?

Continuiamo «dal basso»

X = ALGORITMI

Y = AGORITNI

A	A	A	-	-	A
A	A	-	A	A	-

A
AG

-	A
A	G

G corrisponde a A

- 3 casi:
1. G corrisponde a A
 2. G corrisponde a spazio
 3. A corrisponde a spazio

A	-
A	G

A	-	-
-	A	G

-	A	-
A	-	G

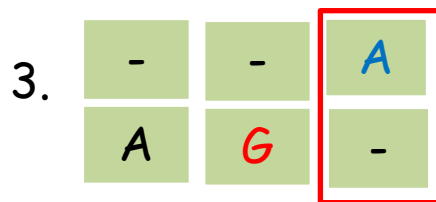
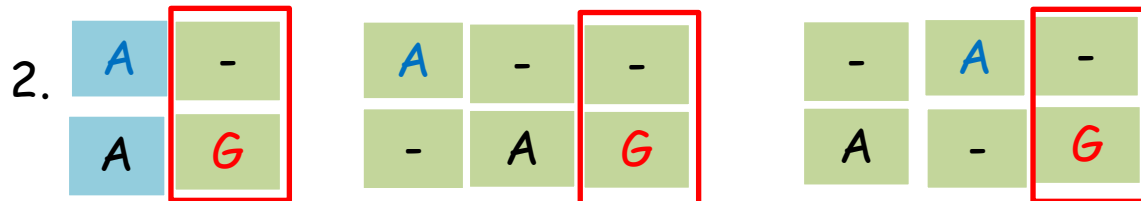
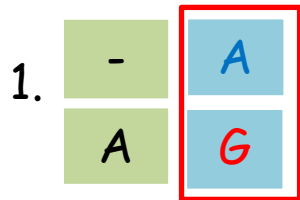
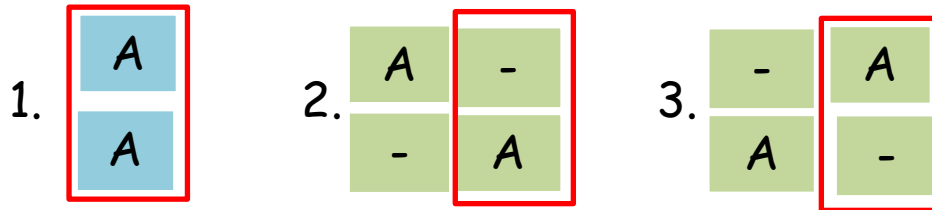
G corrisponde a uno spazio

-	-	A
A	G	-

G corrisponde a un carattere a sinistra di A e
A corrisponde a uno spazio

Continuiamo «dal basso»

$$X = x_1 x_2 \dots x_m \text{ e } Y = y_1 y_2 \dots y_n$$



3 casi:

1. y_n corrisponde a x_m
2. y_n corrisponde a spazio
3. x_m corrisponde a spazio

Posso esprimere il costo minimo per allineare X con Y rispetto a costi minimi per allineare stringhe più corte?

Continuiamo «dal basso»

X = ALGORITMI Y = AGORITNI costo mismatch = 4 , costo gap = 2

1.

A
A

 2.

A	-
-	A

 3.

-	A
A	-

 $\min \{ 0, 2+2, 2+2 \} = 0$

$$\min \{ 2+4, 0+2, 2+2+2, 2+2+2, 2+2+2 \} = 2 =$$

1.

-	A
A	G

 $\min \{ 2+4,$

2.

A	-
A	G

A	-	-
-	A	G

-	A	-
A	-	G

 $\min \{ 0, 2+2, 2+2 \} + 2,$

3.

-	-	A
A	G	-

 $4 + 2 \} = 2$

Sotto-problemi

Il problema di allineare due stringhe di lunghezza m ed n si riconduce al problema di allineare due stringhe, di cui **almeno una è più corta**.

In generale, il **sotto-problema** che ci troveremo a dover risolvere è quello di **allineare due prefissi** delle stringhe di partenza, di lunghezza qualsiasi:

$$\begin{array}{c} x_1 x_2 \dots x_i \\ y_1 y_2 \dots y_j \end{array}$$

Il generico sotto-problema sarà definito dagli indici i e j .

I **casi base** si avranno quando una (almeno) delle due stringhe è vuota; in tal caso per i simboli dell'altra stringa si potranno avere solo gap

Soluzione ottimale OPT

Consideriamo un allineamento ottimale OPT delle stringhe

$$x_1 x_2 \dots x_m \quad e \quad y_1 y_2 \dots y_n$$

Guardiamo l'allineamento in OPT degli **ultimi caratteri x_m e y_n** .

Sono possibili 2 casi: x_m e y_n sono in corrispondenza oppure no; nel secondo caso almeno uno dei 2 è in corrispondenza di uno spazio vuoto (altrimenti crossing).

CASO 1: x_m e y_n sono in corrispondenza:

$x_1 x_2 \dots x_{m-1}$	x_m
$y_1 y_2 \dots y_{n-1}$	y_n

Caso 2a: x_m non corrisponde a nessun carattere:

$x_1 x_2 \dots x_{m-1}$	x_m
$y_1 y_2 \dots y_n$	-

Caso 2b: y_n non corrisponde a nessun carattere:

$x_1 x_2 \dots x_m$	-
$y_1 y_2 \dots y_{n-1}$	y_n

Sequence Alignment: Problem Structure

Def. $OPT(i, j)$ = min cost of aligning strings $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$.

Case 1: OPT matches x_i - y_j .

pay mismatch for x_i - y_j + min cost of aligning two strings
 $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 \dots y_{j-1}$

Case 2a: OPT leaves x_i unmatched.

pay gap for x_i and min cost of aligning $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 \dots y_j$

Case 2b: OPT leaves y_j unmatched.

pay gap for y_j and min cost of aligning $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_{j-1}$

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Implementazione con algoritmo iterativo

Uso una tabella M per contenere $OPT(i,j)$ per $i=0,1,\dots,m$, $j=0,1,\dots,n$.

Comincio con l'inserire i valori dei casi base, $i=0$ e $j=0$, una riga e una colonna.

m						
...						
i						
...						
0						
	0	...	j	...	n	

m	x					
...	x					
i	x					
...	x					
0	x	x	x	x	x	x
	0	...	j	...	n	

m	x					x
...	x					
i	x		x	x		
...	x		x	x		
0	x	x	x	x	x	x
	0	...	j	...	n	

Scorro tutte le altre celle della tabella in modo che quando devo riempire la cella $M[i,j]$, i valori necessari al calcolo siano già inseriti nelle corrispondenti celle. I valori necessari sono $M[i-1,j-1]$, $M[i,j-1]$ e $M[i-1,j]$. Posso scorrere per righe da 1 a m e per colonne da 1 a n .

La soluzione al problema di allineare X e Y sarà in $M[m,n]$, che restituisco alla fine.

Sequence Alignment: Algorithm

```
Sequence-Alignment( $m, n, x_1x_2\dots x_m, y_1y_2\dots y_n, \delta, \alpha$ ) {  
    for  $j = 0$  to  $n$   
         $M[0,j] = j\delta$   
    for  $i = 0$  to  $m$   
         $M[i,0] = i\delta$   
    for  $i = 1$  to  $m$   
        for  $j = 1$  to  $n$   
             $M[i,j] = \min(\alpha[x_i, y_j] + M[i-1, j-1],$   
                         $\delta + M[i-1, j],$   
                         $\delta + M[i, j-1])$   
    return  $M[m, n]$   
}
```

Analysis. $\Theta(mn)$ time and space.

Esempio

Esempio: X = mean, Y = name

$\delta = 2$

costo mismatch fra vocali differenti=1

costo mismatch fra consonanti differenti=1

costo mismatch fra vocale e consonante=3

$M[0,0]=0, M[0,1]=2, M[0,2]=4, \dots$

$M[1,0]=2, M[2,0]=4, \dots$

$M[1,1]=\min\{\alpha_{ma} + M[0,0], \delta + M[0,1], \delta + M[1,0]\} = \min\{1+0, 2+2, 2+2\} = 1$

$M[1,2]=\min\{\alpha_{me} + M[0,1], \delta + M[0,2], \delta + M[1,1]\} = \min\{3+2, 2+4, 2+1\} = 3$

$M[1,3]=\min\{\alpha_{mn} + M[0,2], \delta + M[0,3], \delta + M[1,2]\} = \min\{0+4, 2+6, 2+3\} = 4$

.....

$M[4,4]=\min\{\alpha_{ne} + M[3,3], \delta + M[3,4], \delta + M[4,3]\} = \min\{3+5, 2+5, 2+4\} = 6$

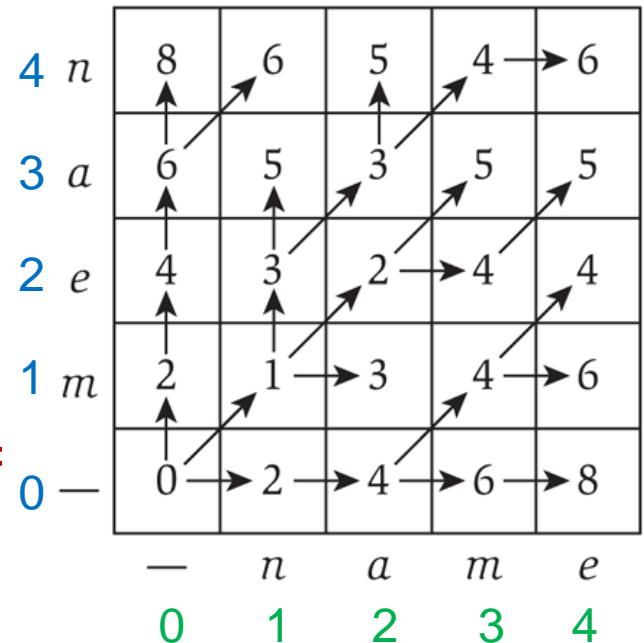


Figure 6.18 The OPT values for the problem of aligning the words *mean* to *name*.

Ricostruzione dell'allineamento

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Esempio: X = mean, Y = name

$\delta = 2$

costo mismatch fra vocali differenti=1

costo mismatch fra consonanti differenti=1

costo mismatch fra vocale e consonante=3

La freccia nella casella (i,j) proviene dalla casella usata per ottenere il minimo

$$\begin{aligned} M[4,4] &= \\ &= \min\{\alpha_{ne} + M[3,3], \delta + M[3,4], \delta + M[4,3]\} = \\ &= \min\{3 + 5, 2 + 5, 2 + 4\} = 6 \end{aligned}$$

Inserisco freccia da $M[4,3] \rightarrow M[4,4]$

Per ricostruire l'allineamento seguiamo il percorso all'indietro nella matrice

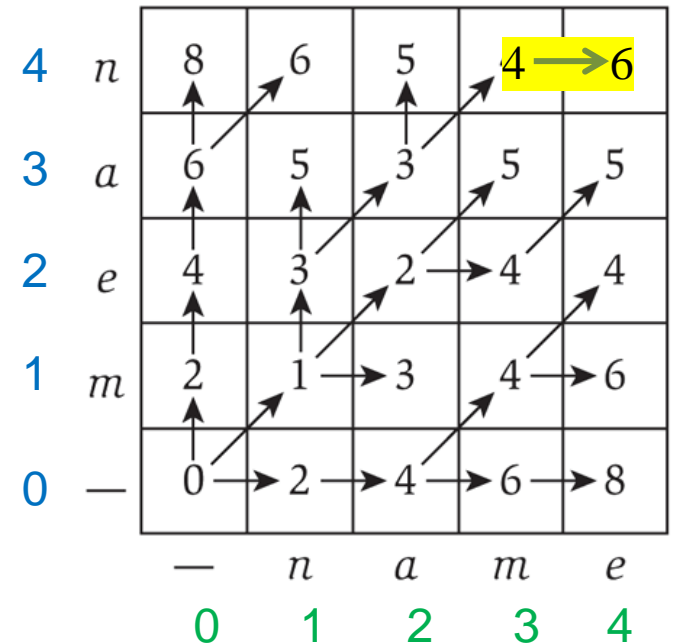


Figure 6.18 The OPT values for the problem of aligning the words *mean* to *name*.

Ricostruzione soluzione ottimale

Per ricostruire un **allineamento ottimale** seguiamo il percorso all'indietro nella matrice

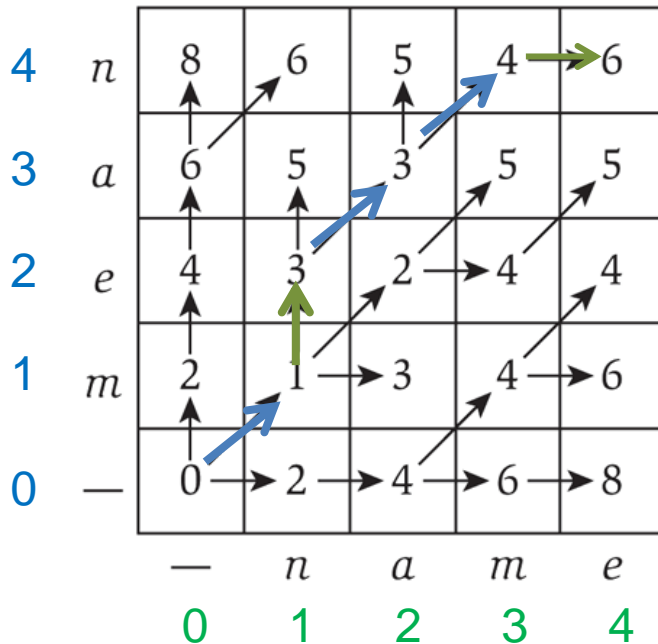
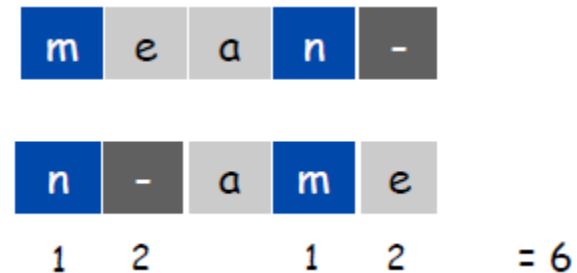


Figure 6.18 The OPT values for the problem of aligning the words *mean* to *name*.



$$M[4,4] = \min\{\alpha_{ne} + M[3,3], \delta + M[3,4], \delta + M[4,3]\} = \min\{3+5, 2+5, 2+4\} = 6$$

Calendario

- Martedì 5 aprile: PD3 - Il problema dello zaino
- Mercoledì 6 aprile: Esercitazione
- Venerdì 8 aprile: Esercitazione

- Martedì 12 aprile ore 14: **Prima prova intercorso**, aule P3 e P4.

- Mercoledì 13 aprile: **niente lezione**

- 14 - 19 aprile: Festività Pasquali

- Mercoledì 20 aprile: Lezione 19: Discussione prima prova
-

Prima prova

In Materiali del corso del team, trovate la prima prova intercorso dell'a.a. 2017/18.

Provate a risolverla, tranne le domande a risposta multipla n° 6, 7, 8.

La prima prova intercorso di quest'anno seguirà la stessa impostazione.

Appello 22 febbraio 2016

3)

Si considerino le stringhe

$X = x_1 x_2 x_3 x_4 x_5 = \text{mamma}$ e $Y = y_1 y_2 y_3 = \text{mia}$.

Se il costo di un *gap* è 5, il costo di un *mismatch* fra due vocali differenti è 3, fra due consonanti differenti è 4 e fra vocale e consonante è 7, allora il costo dell'allineamento $x_1 - y_1$, $x_3 - y_2$, $x_5 - y_3$ è:

A. 7

B. 14

C. 17

D. Nessuna delle risposte precedenti

Esercizio

Si considerino le stringhe

$X = x_1 x_2 x_3 x_4 x_5 = \text{mamma}$ e $Y = y_1 y_2 y_3 = \text{mia}$.

Si supponga che il costo di un *gap* sia 5, il costo di un *mismatch* fra due vocali differenti sia 3, fra due consonanti differenti è 4 e fra vocale e consonante è 7.

Calcolare il costo minimo di un allineamento delle stringhe X e Y, applicando l'algoritmo studiato.

Appello 24 gennaio 2017

Quesito 1 (24 punti)

Si consideri la seguente funzione $c(i,j)$ definita per $1 \leq i, j \leq n$ da:

$$\begin{aligned} c(1,j) &= 1 \text{ per ogni } 1 \leq j \leq n, & c(i,1) &= 3 \text{ per ogni } 2 \leq i \leq n, \\ c(2,j) &= 2 \text{ per ogni } 2 \leq j \leq n, & c(i,2) &= 4 \text{ per ogni } 3 \leq i \leq n, \\ c(i,j) &= \max \{ 3 \times c(i-2, j), c(i, j-1) - 2 \} \text{ per ogni } 3 \leq i, j \leq n. \end{aligned}$$

- Disegnare la matrice c per $n=4$
- Scrivere lo pseudocodice di un algoritmo ricorsivo per il calcolo di $c(n,n)$ e indicarne la complessità di tempo (non è necessaria una analisi dettagliata).
- Scrivere lo pseudocodice di un algoritmo di programmazione dinamica per il calcolo di $c(n,n)$ ed analizzarne la complessità. E' necessario giustificare la risposta.