Programmazione Dinamica (3)

- Alcuni problemi su stringhe

Sottosequenza Comune massima (LCS longest common substring) (cap 15.4)

In applicazioni di Biologia e soprattutto di Bioinformatica richiedono spesso algoritmi su STRINGHE di dati testuali o numerici.

Ad esempio dati due segmenti di DNA

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

possiamo valutarne la SIMILARITÁ in vari modi, secondo le esigenze dell'applicazione stessa.

- Per esempio una SOTTOSEQUENZA di una stringa di lunghezza n

$$S = s_1 s_1 s_2 s_3 \cdots s_{(n-1)} s_n$$

è una stringa $s_{i_1} s_{i_2} s_{i_3} \ldots s_{i_\ell}$ $\quad 0 \le i_1 < i_1 < i_2 < i_3 \cdots < i_\ell < n$ che induce

Quindi è una stringa ottenuta da S prendendo IN ORDINE alcuni caratteri di S ANCHE NON CONSECUTIVI.

E.s. ABCBDAB di lunghezza 7 ha una sottosequenza

BCDB di lunghezza 4

Def Una sottosequenza comune di due stringhe S_1 e S_2 è una stringa s tale che
- s è sottosequenza di S_1
- s è sottosequenza di S_2

Due stringhe possono essere considerate molto simili se hanno una sottosequenza comune molto lunga

PROBLEMA Date due stringhe X e Y, trovate una sottosequenza comune di lunghezza massima.

E.s. una sottosequenza comune è visualizzata nell'esempio del DNA

E.s. $X = A\,B\,C\,B\,D\,A\,B$

$Y = B\,D\,C\,A\,B\,A$

- Delle sottosequenze comuni sono BCA, BCBA, BDAB
- Non ci sono sottosequenze comuni di lunghezza ≥ 5

Identifichiamo una sottostruttura ottima della soluzione per impostare un algoritmo

Sia $z_1, z_2 \cdots z_t$ una sottostringa di lunghezza MASSIMA tra

$X = x_1 \cdots x_m \qquad m = |X|$

$Y = y_1 \cdots y_n \qquad n = |Y|$

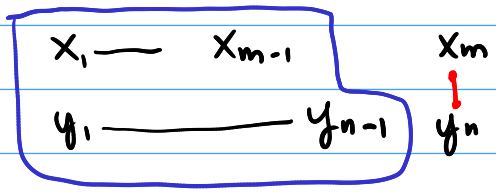
- Se $x_m = y_n$ allora $z_1 \cdots z_{(t-1)}$ è una LCS di $x_1 — x_{(m-1)}$ e $y_1 — y_{(n-1)}$

  perché se ce ne fosse una di lunghezza $\geq t$ allora la LCS da X e Y avrebbe lunghezza $\geq t+1$

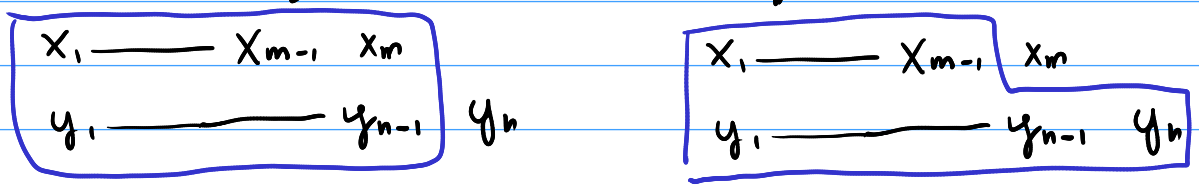
- Se $x_m \neq y_n$ allora abbiamo $x_m \neq z_t$ oppure $y_n \neq z_t$ (o entrambi)
  - Se $x_m \neq z_t$ allora $Z = z_1 — z_t$ è una LCS di $x_1 — x_{(m-1)}$ e $y_1 — y_n$ [Possiamo escludere x_m]
  - Se $y_n \neq z_t$ allora $Z = z_1 — z_t$ è una LCS di $x_1 — x_m$ e $y_1 — y_{(n-1)}$ [escludiamo y_n]

sotto problema

Quindi se $X_m = Y_n$



se $X_m \neq Y_n$ escludo Y_n due casi escludo X_m



OSS Per $X_m \neq Y_n$ i due casi non sono disgiunti

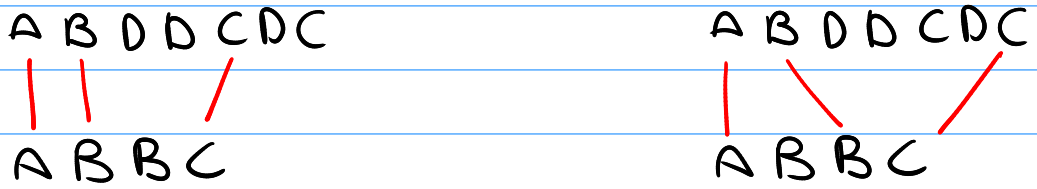
- La stessa soluzione può essere ottima per entrambi i sottoproblemi

OSS Guardate l'esempio

$X =$ A B D D C D C

$Y =$ A B B C

osservate che qui $X_m = Y_n$ Tuttavia esistono anche soluzioni ottime ottenibili escludendo X_m.



La nostra caratterizzazione della sottostruttura ottima non prende in considerazione associazioni come quella a sinistra, ma solo come quella a destra. Eppure l'ottimalità del risultato non è pregiudicata

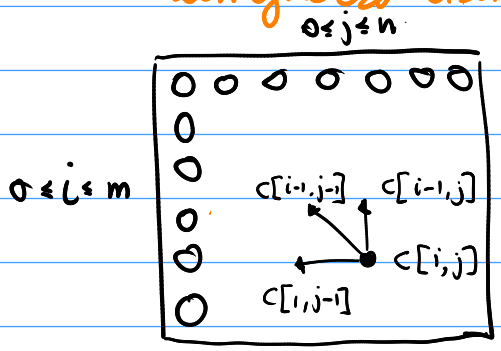
Soluzione ricorsiva

Sia $c[i,j]$ la lunghezza della LCS di $X_1 - X_i$ e $Y_1 - Y_j$

allora

$$c[i,j] = \begin{cases} 0 & \text{se } i=0 \text{ o } j=0 \\ 1 + c[i-1, j-1] & \text{se } x_i = y_j \\ \max\left(c[i-1,j], c[i,j-1] \right) & \end{cases}$$

Calcolo Bottom-up della lunghezza ottima



i valori di $c[i,j]$ possono essere calcolati riga per riga o colonna per colonna, utilizzando spazio

$$\Theta(\min\{n,m\})$$

Calcolo della soluzione ottima

Come in altri esempi, una soluzione ottima può essere calcolata tenendo traccia delle scelte fatte durante il calcolo bottom-up

- Calcolando $c[i,j]$ =
  - usiamo $c[i-1,j-1]$ abbiamo preso x_i e y_j ↖
  - usiamo $c[i,j-1]$ abbiamo scartato y_j ←
  - usiamo $c[i-1,j]$ abbiamo scartato x_i ↑
- Possiamo memorizzare in una seconda matrice B le direzioni ↑, ↖, ←
  del calcolo.
- I cavalieri della sottosequenza sono quelli che corrispondono a ↖

		j	0	1	2	3	4	5	6
i			y_j	B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

X = A B C B D A B

Y = B D C A B A

- Lo spazio necessario per calcolare $C[m,n]$ è $\Theta(\min\{n,m\})$
  tuttavia se vogliamo ricostruire la soluzione,

  Q quanto spazio è necessario ?

Implementazione completa in python

```
30 def LCS(X,Y):
31     m = len(X)
32     n = len(Y)
33     C = creatematrix(m+1,n+1,fill=0)
34     B = creatematrix(m+1,n+1,fill=" ")
35     for i in range(1,m+1):
36         for j in range(1,n+1):
37             if X[i-1] == Y[j-1]:
38                 C[i][j] = C[i-1][j-1]+1
39                 B[i][j] = "↖"
40             elif C[i-1][j] >= C[i][j-1]:
41                 C[i][j] = C[i-1][j]
42                 B[i][j] = "↑"
43             else:
44                 C[i][j] = C[i][j-1]
45                 B[i][j] = "←"
46     return extractLCS(B,X,Y)
```

indici delle stringhe partono da 0

La funzione seguente estrae la sottosequenza visitando la matrice B

```
16 def extractLCS(B,X,Y):
17     i,j = len(B)-1, len(B[0])-1
18     data = []
19     while i>0 and j>0:
20         if B[i][j]== "↖":
21             data.append(X[i-1])
22             i-=1
23             j-=1
24         elif B[i][j]== "↑":
25             i-=1
26         elif B[i][j]== "←":
27             j-=1
28     return "".join(reversed(data))
```

ESERCIZIO (15.4-1) Calcolate la LCS di

$X = 10010101$ e $Y = 0101110110$

Domanda Qual è la complessità dell'algoritmo LCS?

ESERCIZIO (15.4-3) Scrivete una versione top-down di LCS che abbia la stessa complessità (quindi usando la memoizzazione)

ESERCIZIO (15.4-4)

- Mostrate come calcolare la lunghezza della LCS mantenendo in memoria solo

$$2 \cdot \min(m,n) + O(1)$$

celle della tabella C

- Mostrate come farlo usando solo $\min(m,n) + O(1)$ celle

ESERCIZIO (15.2) Trovate un algoritmo che, data una stringa S di n caratteri,

trovi la più lunga sottosequenza palindroma

DISTANZA DI EDITING (esercizio 15-5(a))

Una forma più complessa per indicare la similarità tra due stringhe è la distanza di editing.

- Date due stringhe X e Y, quante operazioni di editing sono necessarie per trasformare X in Y

- Cosa vuol dire OPERAZIONI DI EDITING va specificato, per dare una risposta

Es. se le operazioni a disposizione sono la sostituzione di un carattere o la sua cancellazione

- BASTA ⟶ CASA con 2 operazioni

  BASTA ⟶ CASTA ⟶ CASA

Vediamo una descrizione più sistematica del processo

Date X e Y immaginiamo un processo che legga e "consumi" i caratteri di X e man mano scriva su un buffer Z secondo una scelta tra un insieme di regole. X è letta, e Z è scritta, in modo prettamente sequenziale. Alla fine del processo vogliamo che in Z ci sia una copia di Y

- Abbiamo una stringa $X = x_1 x_1 \dots x_m$ [input]
- Un indice $i = 1$ [indice in X]
- Un array Z di lunghezza n con $n = |Y|$ [output]
- Un indice $j = 1$ [indice in Z]

e abbiamo una SERIE DI OPERAZIONI, ognuna con un COSTO FISSO

Le operazioni ammissibili sono alcune o tutte tra le seguenti

COPIA: Imposta $Z[j] \leftarrow X[i]$ e incrementa i e j di 1

SOSTITUISCI(c): Imposta $Z[j] \leftarrow c$ e incrementa i e j di 1

CANCELLA: Incrementa i di 1

INSERISCI(c): Imposta $Z[j] \leftarrow c$ e incrementa j

SCAMBIA: Imposta $Z[j] \leftarrow X[i+1]$ e $Z[j+1] \leftarrow X[i]$ e incrementa i e j di 2

TERMINA: deve essere l'ultima operazione. i è settata a m

Esempio algorithm → altruist

	X:	Z
	algorithm	_
copia	algorithm	a_
copia	algorithm	al_
sostituisci t	algorithm	alt_
cancella	algorithm	alt_
copia	algorithm	altr_
inserisci u	algorithm	altru_
inserisci i	algorithm	altrui_
inserisci s	algorithm	altruis_
SCAMBIA	algorithm	altruisti_
inserisci c	algorithm	altruistic_
TERMINA	algorithm_	altruistic_

Lo scopo é partire dalla stringa X e arrivare alla Y, minimizzando il costo totale delle operazioni

Ma il costo totale non é definito se non definiamo

1. Quali operazioni sono permesse
2. Il costo di ogni operazione

Queste scelte danno luogo a molte variazioni di questo problema e altri problemi posso essere formalizzati come suoi casi speciali

ESERCIZIO Supponiamo di voler scrivere se Y é una sottosequenza di X
Come possiamo usare MINIMUM EDIT DISTANCE per farlo?

Esempi di impostazione dei pesi:

- COPIA → 0
- INSERISCI / CANCELLA / SCAMBIA / SOSTITUISCI / TERMINA → 1

LCS come variante di MINIMUN EDIT DISTANCE

- COPIA → -1
- INSERISCI / CANCELLA → 0
- SCAMBIA / SOSTITUISCI / TERMINA → non ammessi

ESERCIZIO Dimostrare che LCS puó essere espresso come la variante di Min Edit Distance descritta sopra

OSSERVAZIONE Il numero possibile di operazioni é finito, poiché in INSERISCI(c) e SOSTITUISCI(c) c deve essere per forza Y[j]

SOTTOSTRUTTURA OTTIMA

Impostiamo l'algoritmo per MED, descrivendo la sottostruttura ottima

- Definiamo $D(i,j)$ la minima distanza di editing tra $x_1 - x_i$ e $y_1 - y_j$ così $D(n,m)$ è la risposta cercata
- Caso base $D(0,0) = 0$

- L'ultima azione può essere un termina, ma solo se $j = n$
  QUESTO CASO LO CONSIDERIAMO DOPO

- Altrimenti l'ultima azione in una soluzione di costo $D(i,j)$
  - CANCELLAZIONE di x_i
    che segue una sequenza ottima che porta $x_1 - x_{(i-1)} \longrightarrow y_1 - y_j$
  - INSERIMENTO che segue l'editing da $x_1 - x_i$ a $y_1 - y_{(j-1)}$
  - COPIA / SOSTITUISCI che segue l'editing da $x_1 - x_{i-1}$ a $y_1 - y_{(j-1)}$
  - SCAMBIA che segue l'editing da $x_1 - x_{i-2}$ a $y_1 - y_{(j-2)}$

Va osservato che alcune di queste azioni di edit possono non essere fattibili (e.g. copia o scambio)

Non abbiamo considerato l'azione TERMINA.
In questo caso l'azione termina si può effettuare solo se in Z ci sta una copia di Y, quindi $j = n$, e $i < m$.

Impostazione alla ricorsione

$D(0,0) = 0$

$D(i,j) =$ Minimo tra:

- $D(i-1,j)$ + costo (CANCELLAZIONE) — se $i<m$
- $D(i,j-1)$ + costo (INSERIMENTO)
- $D(i-1,j-1)$ + costo (SOSTITUISCI)
- $D(i-1,j-1)$ + costo (COPIA) — se $X[i] = Y[j]$
- $D(i-2,j-2)$ + costo (SCAMBIA) — se applicabile
- $D(t,j)$ + COSTO (TERMINA) — per $t<m$ se $i=m$, $j=n$

Solo per $D(m,n)$

ESERCIZIO Completate questo schema e scrivete l'algoritmo

- Valutatene la complessità in termini di tempo
- Valutatene la complessità in termini di spazio

ALLINEAMENTO DI SEQUENZE DI DNA (esercizio 15-5 (b))

- Un metodo per misurare la similarità di sequenze di DNA è lo studio dell'allineamento di queste sequenze
- Per esempio due sequenze possono essere allineate inserendo spazi nelle due sequenze (anche in cima e in fondo) per poi misurare la loro similarità.

E.s. x = GATCGGCAT y = CAATGTGAATC

allineate

```
G ATCG GCAT
CAAT GTGAATC
• • • • • • • • • •
```

- -2 punti (spazio)
- +1 punto (uguali)
- -1 punto (diversi)

Punteggio totale: -4

ESERCIZIO Trovare un sottoinsieme delle 6 operazioni e degli assegnamenti per i punteggi tale che il problema di trovare un allineamento ottimo si riduca al problema della distanza minima di editing tra le due stringhe.

ESERCIZIO Distanza di editing e simmetrica.

Normalmente una misura di distanza è tale dal punto di vista matematico se, tra altre cose, è simmetrica.

Ovvero $dist(X,Y) = dist(Y,X)$

① Osservate che a seconda dei punteggi e delle operazioni usate, la distanza minima di editing può essere o non essere simmetrica

② Dimostrate che con i pesi appropriati, le regole COPIA, INSERISCI, SCAMBIA, CANCELLA, SOSTITUISCI danno luogo ad una distanza minima simmetrica

③ Osservate che qualunque punteggio P si dia alla regola "termina", esistono X_p, Y_p per cui la distanza minima non è simmetrica.