

Leczione 7

SHORTEST PATH

input: un grafo $G = (V, E)$, due vertici $u, v \in V$

goal: trovare un cammino da u a v di lunghezza minima.

LONGEST SIMPLE PATH

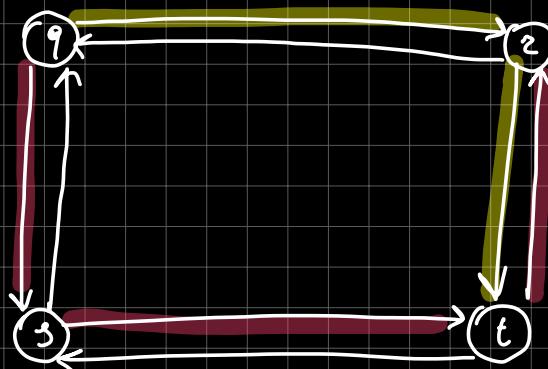
input: un grafo $G = (V, E)$, due vertici $u, v \in V$

goal: trovare un cammino simplice da u a v di lunghezza massima.

SHORTEST PATH ha la proprietà di sottostruttura ottima.

LONGEST SIMPLE PATH non ha la proprietà di sottostruttura ottima, e non è possibile risolverlo mediante programmatore dinamico.

Consideriamo ad esempio il seguente grafo



e i vertici q e t .

Il cammino semplice da q a t è $q \rightarrow r \rightarrow t$.

$q \rightarrow r$ è il cammino semplice più lungo fra q ed r ? No!

Inoltre, $q \rightarrow s \rightarrow t \rightarrow r$.

Nou soltanto, il problema non gode delle proprietà di sostituzione ottime, ma anche nou c'è detto che soluzioni ottime dei sottoproblemis dicono una soluzione ottime al problema originario, infatti:

$$(q \rightarrow s \rightarrow t \rightarrow z) \circ (z \rightarrow q \rightarrow s \rightarrow t)$$

↓ ↓
 sol ottima
 del sottoproblemis
 $LSP(G, q, z)$
 \downarrow
 sol ottima
 del sottoproblemis
 $LSP(G, s, t)$

↓
 Non è una sol ottima
 a $LSP(G, q, t)$.

L'errore e' rappresentato dal fatto che le soluzioni dei sottoproblemis non sono compatibili.

OSS: le SHORTEST PATH, MATRIX MULTIPLICATION, ROD CUTTING i sottoproblemis generati sono compatibili.

7.1 OVERLAPPING SUBPROBLEMS

Si dice che un problema ha sottoproblemis sovrapposti (overlapping subproblems) se un algoritmo ricorsivo per il problema risolve lo stesso sottoproblemis ripetutamente. Lo spazio dei sottoproblemis deve essere "piccolo" (polinomiale).

7.2 LONGEST COMMON SUBSEQUENCE (LCS)

Formalmente, data una sequenza $X = \langle x_1, x_2, \dots, x_m \rangle$, diciamo che $Z = \langle z_1, z_2, \dots, z_k \rangle$ è una sottosequenza (sottosequenza) di X se esiste una sequenza crescente di indici $\langle i_1, \dots, i_k \rangle$ di X tale che $\forall j=1, \dots, k$

$$x_{i_j} = z_j.$$

Esempio: $Z = \langle B C D B \rangle$ è una sottosequenza di $X = \langle A \underset{2}{B} \underset{3}{C} \underset{4}{B} \underset{5}{D} \underset{6}{A} \underset{7}{B} \rangle$ con sequenze di indici $\langle 2, 3, 5, 7 \rangle$.

Date due sequenze X e Y , diciamo che la sequenza Z è una sottosequenza comune a X e Y se Z è sottosequenza di entrambi.

LONGEST COMMON SUBSEQUENCE (LCS)

input: due sequenze $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$.

goal: trovare una sottosequenza comune Z di X e Y di lunghezza massima.

Esempio: $X = \langle A \underset{2}{B} \underset{3}{C} \underset{4}{B} \underset{5}{D} \underset{6}{A} \underset{7}{B} \rangle$, $Y = \langle \underset{1}{B} \underset{2}{D} \underset{3}{C} \underset{4}{A} \underset{5}{B} \underset{6}{A} \rangle$

$Z_1 = \langle B C A \rangle$ è una sottoseq. comune ma non è di lunghezza massima.

$Z_2 = \langle B C A B \rangle$ è comune e di lunghezza max.

Potremmo pensare di risolvere LCS mediante un approccio
brute-force: enumerando tutte le sottosequenze di X e controllando
per ogni sottosequenza se essa è pure una sottosequenza
di Y , tenendo traccia delle più lunghe sottosequenze trovate.
Questa sottosequenza di X corrisponde ad una sottosequenza dei
suoi suffici $\{1, 2, \dots, m\}$. Poiché X ha 2^m sottosequenze,
questo approccio richiede tempo esponenziale.

7.2.1 LCS ha la proprietà di SottostruTTura ottima

Data una sequenza $X = \langle x_1, x_2, \dots, x_m \rangle$, chiamiamo
i-esimo prefisso di X la sequenza $X_i = \langle x_1, x_2, \dots, x_i \rangle$, per
 $i \in \{1, \dots, m\}$.

THM (SOTTOSTRUTTURA Ottima di LCS): Siano $X = \langle x_1, \dots, x_m \rangle$ e
 $Y = \langle y_1, \dots, y_n \rangle$ due sequenze e sia $Z = \langle z_1, \dots, z_k \rangle$ una LCS di
 X e Y .

- (1) Se $x_m = y_n \Rightarrow z_k = x_m = y_n$ e Z_{k-1} è una LCS di X_{m-1} e Y_{n-1} .
- (2) Se $x_m \neq y_n$ e $z_k \neq x_m \Rightarrow Z$ è una LCS di X_{m-1} e Y .
- (3) Se $x_m \neq y_n$ e $z_k \neq y_n \Rightarrow Z$ è una LCS di X e Y_{n-1} .

Proof:

- (1) Se $z_k \neq x_m$ allora potremmo appenderci $x_m = y_n$ a Z
per ottenere una sottosequenza comune a X e Y di
lunghezza $k+1$, contraddicendo l'ipotesi che Z sia
la più lunga sottosequenza comune a X e Y . Per ciò,

necessariamente $z_k = x_m = y_n$.

Adesso vogliamo mostrare che z_{k-1} è una sequenza comune a X_{m-1} e Y_{n-1} di lunghezza massima, $k-1$. Supponiamo che W sia una LCS di X_{m-1} e Y_{n-1} di lunghezza $> k-1$. Allora, appendendo x_m a W otterremmo una sequenza comune a X e Y di lunghezza $> k$, che è una contraddizione.

(2) Se $z_k \neq x_m$ e ci fosse una sequenza W comune a X_{m-1} e Y di lunghezza $> k$, allora W sarebbe comune anche a $X_m = X$ e Y , contraddicendo l'ipotesi che Z è una LCS di X e Y .

(3) Dimostrazione simmetrica (quindi, analogo) al caso (2). \square

Il teorema precedente ci dice che:

- Se $x_m = y_n$, allora dobbiamo trovare una LCS di X_{m-1} e Y_{n-1} e poi appendere $x_m = y_n$ a questo; \rightsquigarrow 1 sotto problema
- Se $x_m \neq y_n$, allora dobbiamo trovare $LCS(X_{m-1}, Y)$ e $LCS(X, Y_{n-1})$, e poi prendere la più lunga tra queste due sequenze. Si osservi che entrambi i sottoproblem contengono il sottoproblema $LCS(X_{m-1}, Y_{n-1})$. \rightsquigarrow 2 sotto problemi
 \downarrow
overlapping subproblems

Definiamo una tabella c di dimensioni $(m+1) \times (n+1)$.

$$c[i, j] := \text{lunghezza della LCS di } X_i \text{ e } Y_j. \quad \forall i, j: i \geq 0, j \geq 0$$
$$c[i, 0] = c[0, j] = 0 \quad \forall i, j \geq 0.$$

Questo spiega ad avere una delle due sequenze di lunghezza zero.

Poiché LCS ha le proprietà di sottostruttura ottime, possiamo scrivere

$$\star \quad c[i,j] = \begin{cases} 0 & i=0 \text{ o } j=0 \\ c[i-1,j-1] + 1 & i,j > 0 \text{ e } x_i = y_j \\ \max\{c[i-1,j], c[i,j-1]\} & i,j > 0 \text{ e } x_i \neq y_j \end{cases}$$

Le formule \star suggeriscono un algoritmo ricorsivo con tempo esponenziale per LCS.

Poiché LCS ha soltanto $\Theta(m \cdot n)$ sotto problemi distinti (calcolare $c[i,j]$ per $0 \leq i \leq m$ e $0 \leq j \leq n$), la procedura di cui diavante può risolvere LCS con un approccio bottom-up in tempo polinomiale.

LCS-LENGTH(X, Y, m, n)

```

1 let b[1:m, 1:n] and c[0:m, 0:n] be new tables
2 for i = 1 to m
3   c[i, 0] = 0
4 for j = 0 to n
5   c[0, j] = 0
6 for i = 1 to m      // compute table entries in row-major order
7   for j = 1 to n
8     if  $x_i == y_j$ 
9       c[i, j] = c[i - 1, j - 1] + 1
10      b[i, j] = " $\nwarrow$ "
11    elseif c[i - 1, j] ≥ c[i, j - 1]
12      c[i, j] = c[i - 1, j]
13      b[i, j] = " $\uparrow$ "
14    else c[i, j] = c[i, j - 1]
15      b[i, j] = " $\leftarrow$ "
16 return c and b

```

PRINT-LCS(b, X, i, j)

```

1 if i == 0 or j == 0
2   return      // the LCS has length 0
3 if b[i, j] == " $\nwarrow$ "
4   PRINT-LCS( $b, X, i - 1, j - 1$ )
5   print  $x_i$       // same as  $y_j$ 
6 elseif b[i, j] == " $\uparrow$ "
7   PRINT-LCS( $b, X, i - 1, j$ )
8 else PRINT-LCS( $b, X, i, j - 1$ )

```

La procedura LCS-length prende in input due sequenze $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$, e le fa n. lunghezze; calcola gli elementi in ordine di lunghezza crescente.

La procedura contiene pure le tabella $b[1:m, 1:n]$.

$b[i, j]$ è una freccia che punta alle lunghezze massime dei sotto problemi scelti nel calcolo di $c[i, j]$.

LCS-length restituisce le tabelle b e c , su particolare $c[m, n]$ che contiene le lunghezze di una $\text{LCS}(X, Y)$.

Il tempo con pufazionale di LCS-length è $\Theta(mn)$, dato che ogni elemento delle tabelle è calcolato su $\Theta(1)$ -time.

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
1	A	0	0	0	1	$\leftarrow 1$	$\nwarrow 1$
2	B	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\nwarrow 2$
3	C	0	1	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\uparrow 2$
4	B	0	1	1	2	$\uparrow 3$	$\nwarrow 3$
5	D	0	1	2	2	3	$\uparrow 3$
6	A	0	1	2	3	3	$\nwarrow 4$
7	B	0	1	2	3	4	$\uparrow 4$

Una volta costruite le tabelle b e c , possiamo ricavare una LCS di X e Y , utilizzando le procedure PRINT-LCS.

Iniziamo da $b[m, n]$ e muovendoci seguendo le frecce. Ogni " \nwarrow " in $b[i, j]$ implica $x_i = y_j$. Questo ci dà gli elementi della LCS su ordine inverso. La procedura PRINT-LCS la stampa nel giusto ordine.

PRINT-LCS impiega tempo $\Theta(m+n)$ perché decrementa almeno uno tra i e j ad ogni passo temporale.

7.2.2 Si può migliorare il codice?

- Si può eliminare la tabella b . $c[i, j]$ dipende solo da $c[i-1, j-1]$, $c[i-1, j]$ e $c[i, j-1]$. Dato $c[i, j]$, si può determinare (in tempo $\Theta(1)$) quale tra questi tre valori è stato usato per $c[i, j]$ senza guardare b . In questo modo, si può costruire LCS in $\Theta(m+n)$ -time utilizzando una procedura simile a PRINT-LCS. \rightsquigarrow Si esigerebbe $\Theta(mn)$ -space (memoria), ma lo spazio ausiliario per calcolare LCS risulta $\Theta(mn)$, quindi lo spazio richiesto non diventa scese assintoticamente. Tuttavia, c'è richiede pure $\Theta(mn)$ -space.
- Si può ridurre assintoticamente lo spazio usato per LCS-length

perché così usa solo due righe di celle volte:
quelle da calcolare e le precedenti. Questo funziona se
vogliamo conoscere solo le lunghezze della LCS e non
conoscere la sequenza stessa.