Lezione 9

9.1 DEFINIZIONE RICORSIVA DELLA LUNGHEZZA DI UNA LCS.

Definiamo $c[i,j]$ come la lunghezza della LCS per $X_i = \langle x_1, \dots, x_i \rangle$ e $Y_j = \langle y_1, \dots, y_j \rangle$.

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

$X = \langle x_1, x_2, \dots, x_m \rangle$ $\quad Z = \langle z_1, \dots, z_k \rangle = LCS(X,Y)$

$Y = \langle y_1, y_2, \dots, y_n \rangle$

- se $x_m = y_n \rightarrow z_k = x_m = y_n$ e ottengo LCS per X_{m-1} e Y_{n-1}
- se $x_m \neq y_n$
  - STUDIO LCS per X_{m-1} e Y
  - STUDIO LCS per X e Y_{n-1}

Basandoci su $(\star)$ possiamo scrivere un algoritmo ricorsivo.
Inoltre, lo spazio dei sottoproblemi ha dimensione polinomiale (proprietà dei sottoproblemi ripetuti), infatti

il numero di sottoproblemi distinti è uguale al numero di $c[i,j]$ distinti, $0 \le i \le m$, $0 \le j \le n$ $\Rightarrow \Theta(mn)$.

$\rightarrow$ POSSIAMO USARE UN APPROCCIO BASATO SULLA PROGRAMMAZIONE DINAMICA

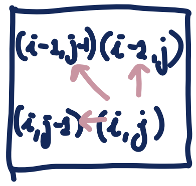
La procedura LCS LENGTH prende in input due sequenze $X = \langle x_1, x_2, \dots, x_m \rangle$ e $Y = \langle y_1, y_2, \dots, y_n \rangle$ assieme alle loro lunghezze, memorizza i valori di $c[i,j]$ in una tabella $c[0:m, 0:n]$ e ne calcola gli elementi in ordine di riga crescente da sx a dx. Mantiene pure la tabella $b[1:m, 1:n]$ di "frecce".

Il tempo di questa procedura è $\Theta(mn)$, dato che ogni elemento è calcolato in $\Theta(1)$.

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

1. Inizializziamo $c[0:m, 0:n]$ e $b[1:m, 1:n]$
2. for $i=0$ to m
3. $\quad c[i,0]=0$
4. for $j=1$ to n

```
        c[0,j] = 0
for i=1 to m
        for j=1 to n
                if xi == yj
                        c[i,j] = c[i-1,j-1]+1
                        b[i,j] = "↖"
                else if c[i-1,j] ≥ c[i,j-1]
                        c[i,j] = c[i-1,j]
                        b[i,j] = "↑"
                else    c[i,j] = c[i,j-1]
                        b[i,j] = "←"
return c and b
```



Una volta costruita la tabella b possiamo risalire a una LCS di X, Y.
Iniziamo da b[m, n] e ci muoviamo seguendo le frecce.

<u>ESEMPIO:</u> $X = \langle A\ B\ C\ B\ D\ A\ B \rangle$, $m = 7$

$Y = \langle B\ D\ C\ A\ B\ A \rangle$, $n = 6$

	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

lunghezza di una LCS(X,Y)

```
PRINTLCS(b, X, i, j)
if i==0 or j==0
    return // lunghezza di una LCS è zero
if b[i,j] == "↖"
    PRINTLCS(b, X, i-1, j-1)
    print "x_i"
else if b[i,j] = "↑"
```

7. PRINTLCS$(b, X, i-1, j)$
8. else PRINTLCS$(b, X, i, j-1)$

PRINTLCS impiega $\Theta(m+n)$-TIME perché decrementa almeno uno tra m ed n ad ogni passo temporale.

Si può migliorare il codice di PrintLCS? Si, eliminando la tabella b. Infatti $c[i,j]$ dipende solo da $c[i-1,j-1]$, $c[i,j-1]$ e $c[i-1,j]$. Dato $c[i,j]$ si può determinare in tempo $\Theta(1)$ quale tra questi tre valori è stato usato per $c[i,j]$. Si può così ricostruire una LCS in tempo $\Theta(m+n)$ usando una procedura simile a PRINTLCS, risparmiando $\Theta(m \cdot n)$-SPACE, ma comunque lo spazio usato non diminuisce asintoticamente perché comunque la tabella c richiede $\Theta(mn)$-SPACE.

Si può ridurre asintoticamente lo spazio usato per LENGTHLCS perché esso ha bisogno solo di due righe alla volta: quella corrente e quella precedente. Questo funziona se vogliamo conoscere <u>SOLO la lunghezza di una LCS</u>.

9.2 ELEMENTI DELLA STRATEGIA GREEDY

Un algoritmo greedy ottiene una soluzione ottima ad un problema di ottimizzazione attraverso una successione di scelte localmente ottime. Ad ogni "bivio" l'algoritmo fa la scelta che sembra migliore al momento.

È una strategia euristica che non sempre produce una soluzione ottima.

COME SVILUPPARE UN ALGORITMO GREEDY

1. Formulare il problema di ottimizzazione come uno nel quale si fa una scelta e si rimane con un unico sottoproblema. Decidere l'euristica (scelta).

2. Dimostrare che esiste sempre una soluzione ottima che contiene la scelta greedy così che la scelta greedy è "sicura".

3. Dimostrare la proprietà di sottostruttura ottima.

9.2.1 PROPRIETÀ DI SCELTA GREEDY

La proprietà di scelta greedy vale se <u>esiste una soluzione ottima che contiene la scelta</u> greedy ad ogni iterazione.

Essa ci assicura di poter assemblare una soluzione ottima globale facendo scelte (greedy) che sono localmente ottime.

COME SI DIMOSTRA CHE UN PROBLEMA HA LA PROPRIETÀ DI SCELTA GREEDY?

Si esamina una soluzione globale e poi si mostra come modificarla sostituendo la scelta greedy a qualche altra scelta, ottenendo così una soluzione che è ancora ottima.

La scelta fatta da un algoritmo greedy può dipendere dalle scelte precedenti ma non da quelle successive.

La programmazione dinamica risolve i sottoproblemi prima di fare le prime scelte (approccio bottom-up)

vs

La strategia greedy fa le prime scelte prima di risolvere qualunque sottoproblema. (approccio top-down)

Problemi risolubili con programmazione dinamica : vale la prop. di sottostruttura ottima

problemi risolubili con strategia greedy : vale la prop. di sottostruttura ottima + la prop. di scelta greedy

COME SI DIMOSTRA LA PROPRIETÁ DI SOTTOSTRUTTURA OTTIMA SE SAPPIAMO GIÁ CHE VALE LA PROPRIETÁ DI SCELTA GREEDY?

Basta soltanto mostrare che combinando una soluzione al sottoproblema ottima alla scelta greedy si ottiene una sol globalmente ottima.