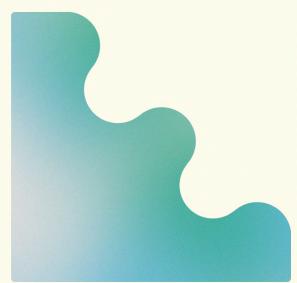


Analisi e Progetto di Algoritmi

879276 * A.A. 2023/2024



haru
879276



Analisi e Progetto di Algoritmi

Appunti del Corso ≈ Anno Accademico 2023-2024 ≈ 879276

25.09.2023

La **programmazione dinamica**, argomento principale del corso, è una tecnica algoritmica per risolvere i problemi basata sulla ricorsione

Ripasso - Notazione

Sia una funzione T che ha come dominio N , il cui singolo valore rappresenta la grandezza dell'input, e dove $T(n)$ rappresenta il costo computazionale dell'algoritmo

Usiamo $T(n)$ per valutare il costo di un algoritmo

$\Theta(T(n))$ è una funzione $f(n)$ tale che da un certo valore in poi, $c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ovvero, $f(n)$ è infinitamente limitata da $g(n)$

$f(n)$ si dice invece $O(g(n))$ se $g(n)$ la limita superiormente

$f(n)$ si dice invece $\Omega(g(n))$ se $g(n)$ la limita inferiormente

Ripasso - Tempo di Calcolo

Il tempo di calcolo di un algoritmo **iterativo** si calcola nel seguente modo:

$x = 1$	1
for $i = 2$ to n	$n - 2 + 1 + 1$
for $j = 1$ to i	$\sum_{i=2}^n [i - 1 + 1 + 1] = \sum_{i=2}^n [i + 1] = (n(n + 1))/2 + (n - 1)$
$x = x + i + j$	$\sum_{i=2}^n = (n(n + 1))/2 - 1$
return x	1

$$T(n) = 1 + n + ((n(n+1))/2 + (n-1)) + ((n(n+1))/2 - 1) + 1 = \Theta(n^2)$$

Il tempo di calcolo di un algoritmo **ricorsivo** si calcola nel seguente modo:

fatt(n)		
if n = 0 return 1	$T(n) = 2$	se $n = 0$
return n * fatt(n - 1)	$2 + T(n - 1)$	se $n > 0$

$$T(n) = 2 + T(n - 1) = 2 + (2 + T(n - 2)) = 2 + 2 + 2 + \dots T(0) = 2 + 2 + 2 + \dots + 2 = 2 + 2n$$

26.09.2023

$X = \langle x_1, \dots, x_n \rangle$ sequenza ordinata di n simboli

$Z = \langle z_1, \dots, z_k \rangle$ è una sottosequenza di X se e solo se esiste una sequenza $I \subset i_1, \dots, i_k$ strettamente crescente di indici tali che per ogni j in I , $z_i = x_{i_j}$

$X = \langle a, b, c, b, d, a, b \rangle \quad n = 7$

$Z = \langle b, c, d, b \rangle$ è una sottosequenza di X poiché esiste $I = \langle 2, 3, 5, 7 \rangle$

Programmazione Dinmica - LCS

LCS - Longest Common Subsequence

$X = \langle x_1, \dots, x_m \rangle$ Sequenze di simboli sullo stesso alfabeto

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

Z sottosequenza sia di X che di Y tale che $|Z| = \max(|W|)$, con W sottosequenza sia di X che di Y

Algoritmo Brute-Force $\Omega(2^m)$

```
for i=1 to  $2^m$ 
    genera i-esima sequenza W di X
    se W è sottosequenza di Y
        se W è più lunga della sequenza Z più lunga trovata fin ora
            Z = W
```

Il brute-force è un algoritmo combinatorio dalle basse prestazioni, è necessario scrivere la risoluzione in termini ricorsivi per ottimizzare la complessità dell'algoritmo

Per farlo, è necessario individuare un **sottoproblema** generico

Il **sottoproblema** che rappresenta il passo ricorsivo per l'LCS sarà trovare una sottosequenza Z comune ad un prefisso di X di lunghezza i e un prefisso di Y di lunghezza j . Ogni sottoproblema è quindi caratterizzato dalla coppia (i, j)

I **casi base** della ricorsione sono tutte le coppie (i, j) dove $i = 0$ o $j = 0$

In questi casi $Z = \langle \rangle$ è sempre sottosequenza di almeno o X_i o Y_j , o entrambi

$(i, j) \quad i = 0 \text{ or } j = 0$

$Z = \langle \rangle$

$i = 0$

Ci sono dunque $m + n + 1$ coppie che rientrano nel caso base, le restanti sono $m \cdot n$

In totale le coppie sono $(m+1) \cdot (n+1)$

Per la generica coppia (i, j) con $i, j > 0$, la soluzione Z dipende dal simbolo corrente:

27.09.2023

$(i, j) \quad i, j > 0$

se $X_i = Y_j$

$Z = f(i-1, j-1) \mid X_i$

altrimenti

$Z = \max(f(i-1, j), f(i, j-1))$

Data la coppia di valori (i, j) , in generale, per poterlo risolvere nel passo ricorsivo ho bisogno di avere la soluzione a tre sottoproblemi: $(i-1, j)$ $(i, j-1)$ $(i-1, j-1)$

LCS-Ricorsivo (X, Y, i, j)

```

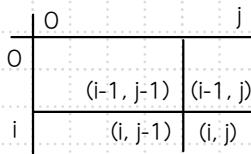
if i = 0 or j = 0
    return <>
if X[i] = Y[j]
    return LCSR(X, Y, i-1, j-1) | Xi
A = LCSR(X, Y, i-1, j)
B = LCSR(X, Y, i, j-1)
if A.len > B.len
    return A
return B

```

Equazione ricorsiva

$$\begin{aligned}
f(i, j) = & \quad f(i-1, j-1) | X_i & \text{se } X_i = Y_j \\
& f(i-1, j) & \text{se } |f(i-1, j)| > f(i, j-1) \\
& f(i, j-1) & \text{se } |f(i, j-1)| \geq f(i-1, j)
\end{aligned}$$

Per costruire la soluzione iterativa dinamica, ragioniamo in modo **bottom-up**, partendo dal sottoproblema più piccolo



I sottoproblemi necessari al problema (i, j) sono quelli a lui adiacenti
I case base si trovano lungo tutta la prima riga e la prima colonna

Le soluzioni sono salvate in memoria in un array bidimensionale S della forma sopra, di dimensioni $m \times n$, dove $S[i, j]$ rappresenta la soluzione $Z(i, j)$

LCS-Iterativo (X, Y)

```

m = X.len
n = Y.len
for j = 0 to n
    S[0, j] = <>
for i = 0 to m
    S[i, 0] = <>

```

```

for i = 1 to m
    for j = 1 to n
        if X[i] = Y[j]
            S[i, j] = S[i-1, j-1] | X[i]
        else
            if S[i-1, j].len > S[i, j-1].len
                S[i, j] = S[i-1, j]
            else
                S[i, j] = S[i, j-1]

```

La proprietà della sottostruttura ottima dell'LCS è:

$$i = 0 \text{ o } j = 0 \quad Z(i, j) = <>$$

$$i > 0 \text{ e } j > 0$$

$$X_i = Y_j \quad Z(i, j) = Z(i-1, j-1) | X_i$$

$$X_i \neq Y_j \quad Z(i, j) = Z(i-1, j) \text{ se } |Z(i-1, j)| > |Z(i, j-1)|$$

$$Z(i, j-1) \text{ se } |Z(i-1, j)| \leq |Z(i, j-1)|$$

Si dice **sottostruttura** in quanto per ogni problema è immediato visualizzare e memorizzare la soluzione dei sottoproblemi ad esso adiacenti

Riformulando il problema e chiedendo come soluzione la **lunghezza** della sottosequenza comune più lunga anziché la sottosequenza stessa, si pesa di meno sulla memoria in quanto nella matrice S vengono memorizzate solo le lunghezze e non le sequenze intere

L-LCS (X, Y, i, j)

```
if i = 0 o j = 0
    return 0
if X[i] = Y[j]
    return L-LCS(X, Y, i-1, j-1) + 1
return max(L-LCS(X, Y, i-1, j), L-LCS(X, Y, i, j-1))
```

L-LCS-Iterativo (X, Y)

$m = X.\text{len}$ $n = Y.\text{len}$ $\text{for } j = 0 \text{ to } n$ $\quad c[0, j] = 0$ $\text{for } i = 0 \text{ to } m$ $\quad c[i, 0] = 0$	$\text{for } i = 1 \text{ to } m$ $\quad \text{for } j = 1 \text{ to } n$ $\quad \quad \text{if } X[i] = Y[j]$ $\quad \quad \quad c[i, j] = c[i-1, j-1] + 1$ $\quad \quad \text{else}$ $\quad \quad \quad c[i, j] = \max(c[i-1, j], c[i, j-1])$
--	--

Per ricostruire la sequenza posso salvarmi una lista di puntatoti (i, j) usati per calcolare la lunghezza massima

02.10.2023 - Esercitazione: Weighted Interrupt Scheduling

		i	p(i)	v_i
1				
2	—	1	0	10
3	—	2	0	2
4	—	3	1	8
5	—	4	2	1
6	—	5	1	1
		6	4	3

Determinare un sottoinsieme di attività eventualmente compatibili che massimizza il valore totale. Nell'esempio, la soluzione sarebbe $\{1, 3, 6\}$

Istanza $n \in \mathbb{N}, X_n = \{1, \dots, n\}$ insieme di n attività

Per ogni $i \in X_n$ si s_i tempo di inizio dell'attività i
 f_i tempo di fine dell'attività i
 v_i valore dell'attività i

Soluzione $S \subseteq X_n$ t.c. $v(S) = \max(v(A) \text{ con } A \subseteq X_n)$
 $\text{comp}(A) = \text{true}$

dove $v() : P(X_n) \rightarrow \mathbb{R} \quad \forall A \subseteq X_n$
 $v(A) = \sum v_i \quad \forall i \in A \quad \text{se } A \neq \emptyset$
 0 se $A = \emptyset$

$\text{comp}(A) = \text{true}$ se A contiene attività
 mutualmente **compatibili**
 false altrimenti

compatibili: $\forall i, j \in A \text{ con } i \neq j \quad [s_i, f_i] \cap [s_j, f_j] = \emptyset$

Il sotto problema è trovare una soluzione ad un **sottoinsieme** X_i di X_n

$$X_i = \{1, \dots, i\} \text{ con } 1 \leq i \leq n$$

$p(i)$ è la precedente attività di indice più alto compatibile con quella presa in considerazione attualmente (i)

$$p(i) = \max\{j \mid t.c. j < i \text{ e } j \text{ compatibile con } i\}$$

$$p(i) = 0 \text{ se } \{j \mid t.c. j < i \text{ e } j \text{ compatibile con } i\} = \emptyset$$

I **casi base** della ritorsione avvengono quando il sottoinsieme S ha uno o zero elementi

$$X_1 = \{1\}$$

$$X_0 = \emptyset$$

$$S_1 = \{1\}$$

$$S_0 = \emptyset$$

$$\text{opt}(1) = v(S_1) = v_1$$

$$\text{opt}(0) = 0$$

Dato un sottoproblema $i > 1$, il **passo ricorsivo** avviene in questo modo:

$S_i =$	$S_p(i) \cup \{i\}$	$\text{se } i \in S_i$	(Questa non è l'equazione di ricorrenza ma la formalizzazione del sotto problema visto che le condizioni non sono algoritmicamente verificabili se non a posteriori)
	$S_{(i-1)}$	$\text{se } i \notin S_i$	
↓			
$S_i =$	$S_p(i) \cup \{i\}$	$\text{se } \text{opt}(p(i) + v_i) \geq \text{opt}(i-1)$	Proprietà della sottostruttura ottima
	$S_{(i-1)}$	altrimenti	

Per la soluzione iterativa dinamica avrò bisogno di un array mono dimensionale M che contenga i valori di $\text{OPT}()$

OPT-D (n)

$M[0] = 0$

$M[1] = v_1$ $S[1] = \{1\}$

for $i = 2$ to n

$M[i] = \max (M[p(i)] + v_i, M[i-1])$

$S[i] = S[p(i)] \cup S[i]$

$S[i-1]$

È necessario però ordinare le attività per tempo di fine prima di eseguire OPT-D

