APA Modulo 1 Lezione 9

Elena Zucca

26 marzo 2020

Tecniche algoritmiche: ripasso divide-et-impera

Schema generale

```
div_imp(P) //dim P = n
  if (n < n_0) risolvi P direttamente
  else
    dividi P in sottoproblemi P_1, ... , P_k di dim < n
    div_imp (P_1)
    ...
    div_imp (P_k)
    costruisci la soluzione di P
   a partire dalle soluzioni di P_1, ... , P_k
```

Correttezza e complessità

- correttezza si basa su induzione forte:
  - l'algoritmo è corretto sui problemi base
  - assumendo che sia corretto sui sottoproblemi (di dimensione minore) lo è sul problema P
- analisi complessità si basa su risolvere relazioni di ricorrenza

Esempi visti: ricerca binaria ricorsiva

```
binary_search(x,a,inf,sup)
  if (inf <= sup)
    mid = (inf + sup)/2
    if (x < a[mid]) return binary_search(x,a,inf,mid-1)
    else if (x > a[mid])
      return binary_search(x,a,mid+1,sup)
    else return true
  return false
```

- problema di partenza: ricerca in $(0, n-1)$
  sottoproblemi: ricerca in $(\mathtt{inf}, \mathtt{sup})$, $0 \leq \mathtt{inf}, \mathtt{sup} \leq n-1$
- base: nessun elemento $(\mathtt{inf} > \mathtt{sup})$
- passo induttivo: ricerca in $(\mathtt{inf}$,$\mathtt{mid}$-1) oppure $(\mathtt{inf}+1$,$\mathtt{sup})$ (dim. $n/2$)
- relazione di ricorrenza: $T(n) = 1 + T(n/2)$

Esempi visti: Hanoi

```
hanoi(n, source, aux, dest)
      if (n = 1) move(source, dest)
      else
        hanoi(n-1, source, dest, aux)
        move(source, dest)
        hanoi(n-1, aux, source, dest)
```

- problema di partenza: Hanoi n
  sottoproblemi: Hanoi k, $1 \leq k \leq n$
- base: un elemento
- passo induttivo: Hanoi $n-1$
- relazione di ricorrenza: $T(n) = 1 + 2T(n-1)$

Programmazione dinamica

Esempio introduttivo: numeri di Fibonacci

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_{i+1} = fib_i + fib_{i-1}$$

0, 1, 1, 2, 3, 5, 8, ...

Complessità

- la definizione induttiva fornisce ovvio algoritmo ricorsivo (divide-et-impera)
- relazione di ricorrenza:

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

Risolviamo:

Dato che ovviamente $T(n+1) > T(n)$ per $n > 2$,

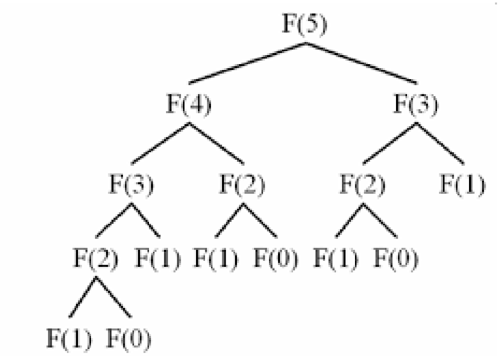
$$T(n) > 2T(n-2) + 1$$

Risolviamo:

$$T(n) > 2(2T(n-4)+1) + 1 = 2^2 T(n-4) + 2^1 + 2^0 > \ldots >$$
$$2^i T(n-2i) + 2^{i-1} + \ldots + 2^1 + 2^0$$
$$\textit{(ultimo termine per } i = k \textit{ se } n = 2k\textit{)}$$
$$> 2^k + \ldots + 2^0 = 2^{k+1} - 1 = 2^{\frac{n}{2}+1} - 1$$

l'albero delle chiamate ricorsive è infatti simile a quello per le torri di Hanoi ma con $n/2$ livelli

- si ha quindi un algoritmo esponenziale
- infatti, l'algoritmo ricorsivo ricalcola inutilmente i risultati parziali, come risulta evidente provando a scrivere l'albero delle chiamate.



- tuttavia, è banale scrivere un algoritmo iterativo lineare con la tecnica della *programmazione dinamica*:

```
Fibonacci(n)
  fib = array con indici 0..n-1
  fib[0] = 0
  fib[1] = 1
  for (i=2; i < n; i++)
    fib[i] = fib[i-1]+ fib[i-2]
  return fib[n-1]
```

Caratteristiche della programmazione dinamica

NB: "programmazione" nel senso di *pianificazione*, come per la programmazione lineare

- come in divide et impera:
  - si ricava la soluzione di un problema dalle soluzioni di sottoproblemi più piccoli
  - correttezza per induzione aritmetica completa sulla dimensione dei problemi
- Ma:
  - bottom-up invece di top-down
  - per prima cosa si risolvono i sottoproblemi base
  - poi via via i successivi fino a quello richiesto, memorizzando i risultati intermedi
  - conveniente se un sottoproblema viene utilizzato per risolvere molti problemi di livello superiore
  - è possibile calcolarne la soluzione una volta sola, e quindi l'approccio risulta vantaggioso

Longest Common Subsequence (LCS)

- problema: date due sequenze trovare una sottosequenza comune di lunghezza massima (individuata da una sequenza di coppie di indici)
- esempi reali: biologia (trovare la più lunga sottosequenza comune a due sequenze di DNA), sicurezza informatica (individuare, in un log costituito da una sequenza di comandi, le sottosequenze che indicano la presenza di un possibile attacco al sistema), etc.

Esempio

AGCCGGATCGAGT

TCAGTACGTTA

una sottosequenza comune di lunghezza massima è:

AGCGTA

Un'altra sottosequenza comune di lunghezza massima, per le stesse due sequenze, è:

AGTCGA

Infatti:

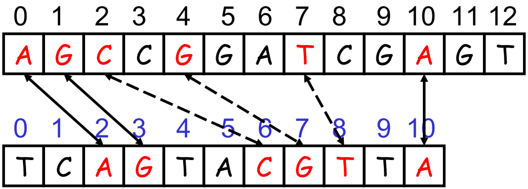
AGCCGGATCGAGT

TCAGTACGTTA

Come rappresentiamo una sottosequenza?

come sequenza di coppie di indici

La più lunga sottosequenza comune di:



è la sottosequenza comune AGCGTA indicata da:

[(0, 2), (1, 3), (2, 6), (4, 7), (7, 8), (10, 10)]

Algoritmo “brute force”

- algoritmo ingenuo: date s_1 lunga m ed s_2 lunga n
- genero tutte le sottosequenze di s_1
  controllando se ognuna è sottosequenza di s_2
  tenendo traccia della lunghezza max
- le sottosequenze di s_1 sono 2^m
- per ognuna occorrono nel caso peggiore n passi per controllare se è anche sottosequenza di s_2
- la complessità è quindi $n \times 2^m$
- esponenziale!

Formulazione induttiva della soluzione: base

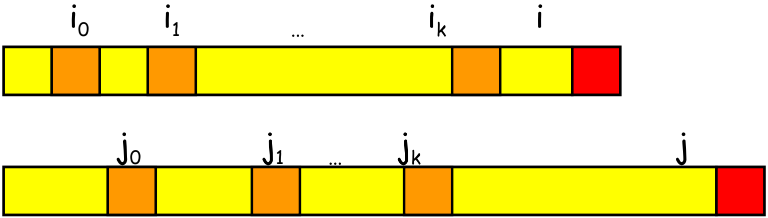
- siano $X[1..m]$ e $Y[1..n]$ sono le due sequenze
- sottoproblema $LCS(i,j)$: sottosequenza comune di lunghezza massima tra i prefissi $X[1..i]$ e $Y[1..j]$.
- problema di partenza $LCS(m,n)$
- base della definizione induttiva: una delle due sequenze è vuota

$$LCS(0,j) = [\,]\ per\ 0 \leq j \leq n$$
$$LCS(i,0) = [\,]\ per\ 0 \leq i \leq m$$

Passo induttivo

- esaminiamo $X(i)$ e $Y(j)$, due casi:
  - se $X(i) = Y(j)$
    le sottosequenze comuni di $X[1..i]$ e $Y[1..j]$ sono tutte quelle di $X[1..i-1]$ e $Y[1..j-1]$, e quelle ottenute aggiungendo in fondo a una di queste la coppia (i, j)
    quindi $LCS(i,j) = LCS(i-1, j-1) \cdot (i,j)$

- **caso 1**: X[i] = Y[j]



Come si vede chiaramente dal disegno,

se LCS(i-1, j-1) = [(i_0, j_0), (i_1, j_1), ... (i_k, j_k)]

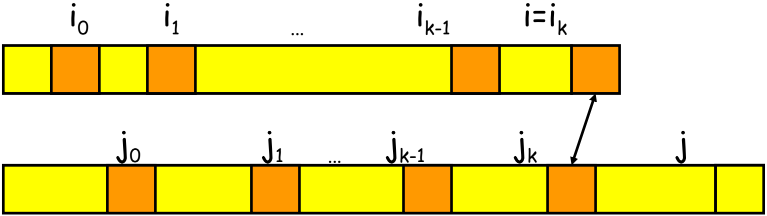
allora LCS(i, j) = [(i_0, j_0), (i_1, j_1), ... (i_k, j_k), (i, j)].

Passo induttivo

- se $X(i) \neq Y(j)$
  le sottosequenze comuni di $X[1..i]$ e $Y[1..j]$ non possono includere la coppia (i, j)
  sono quindi sottosequenze comuni di $X[1..i-1]$ e $Y[1..j]$, oppure sottosequenze comuni di $X[1..i]$ e $Y[1..j-1]$
  quindi $LCS(i, j)$ è la più lunga fra $LCS(i-1, j)$ e $LCS(i, j-1)$

Graficamente

l'elemento finale di una sottosequenza comune non può essere la coppia (i, j), ma può essere una coppia (i, j') con $j' < j$, oppure (i', j) con $i' < i$. Esempio:



Riassumendo:

- base

$$LCS(0,j) = [\,]\ per\ 0 \leq j \leq n$$
$$LCS(i,0) = [\,]\ per\ 0 \leq i \leq m$$

- passo induttivo

$$per\ i \neq 0,\ j \neq 0:$$
$$LCS(i,j) = LCS(i-1,j-1) \cdot (i,j)\ se\ X(i) = Y(j)$$
$$LCS(i,j) = max(LCS(i-1,j), LCS(i,j-1))\ altrimenti$$

- si noti che in questo caso la dimensione del problema è una coppia di numeri, che ordinamento consideriamo?
- prodotto, ossia $(i,j) \leq (i',j')$ se $i \leq i'$ e $j \leq j'$)

Se ci interessa la lunghezza:

- base

$$
\begin{array}{l}
LCS(0,j) = 0 \textit{ per } 0 \leq j \leq n \\
LCS(i,0) = 0 \textit{ per } 0 \leq i \leq m
\end{array}
$$

- passo induttivo

$$
\begin{array}{l}
\textit{per } i \neq 0,\ j \neq 0\textit{:} \\
LCS(i,j) = LCS(i-1,j-1) + 1 \textit{ se } X(i) = Y(j) \\
LCS(i,j) = max(LCS(i-1,j), LCS(i,j-1)) \textit{ altrimenti}
\end{array}
$$

Relazione di ricorrenza

- $$T(n, m) = T(n-1, m) + T(n, m-1) + 1$$
- considerando la somma $k = n + m$ si ha
  $T(k) = 2T(k-1) + 1$ relazione di ricorrenza delle torri di Hanoi
- si può però dare un algoritmo migliore con la programmazione dinamica
- infatti ogni sottoproblema è utilizzato nella soluzione di più problemi di dimensione superiore

Algoritmo di programmazione dinamica

- matrice `LCS` con m+1 righe ed n+1 colonne
- prima riga e colonna per la sequenza vuota, di lunghezza 0
- si riempie riga per riga (o colonna per colonna)
- la casella `LCS(m,n)` conterrà la soluzione
- in ogni casella non serve tutta la LCS ma basta lunghezza e simbolo convenzionale per tre casi, per esempio:
  - se si ha lo stesso carattere su riga e colonna $\nwarrow$ e lunghezza $+ 1$
  - altrimenti, $\uparrow$ o $\leftarrow$ e lunghezza uguale
- la LCS si ricostruisce all'indietro
- se si è interessati solo alla lunghezza, basta costruire la matrice delle lunghezze

Esempio

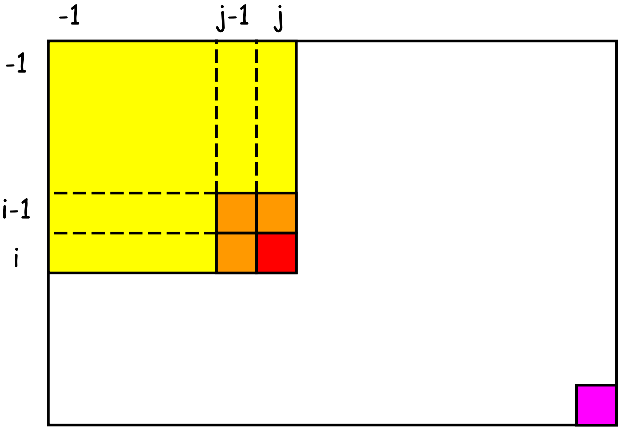
		A	T	C	B	A	B
	0	0	0	0	0	0	0
B	0	0↑	0↑	0↑	↖ 1	← 1	↖ 1
A	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
A	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
T	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
B	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
A	0	↑ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Osservazione

- per calcolare `LCS(i,j)` basta conoscere tre caselle contigue `LCS(i-1,j-1)`, `LCS(i-1,j)`, `LCS(i,j-1)`
- viceversa ogni casella può essere utilizzata per calcolarne altre tre

Graficamente

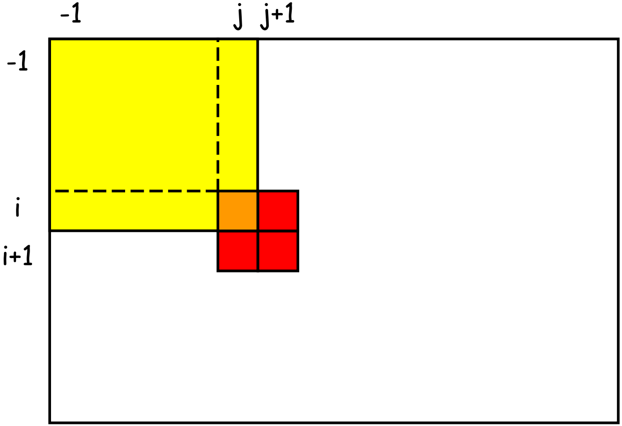
- Per trovare la LCS(i, j) basta conoscere le tre LCS contigue, cioè: LCS(i-1, j-1), LCS(i-1, j), LCS(i, j-1).



Graficamente

Reciprocamente, ogni LCS(i, j) può essere utilizzato per il calcolo di tre altri LCS:

LCS(i+1, j+1), LCS(i, j+1), LCS(i+1, j).



Implementazione

- dato che basta memorizzare solo lunghezza e riferimento, lo spazio necessario è $O(mn)$
- usiamo per semplicità due matrici: la matrice `L` delle lunghezze e la matrice `R` dei riferimenti (coppie di indici, oppure tre valori convenzionali)

Pseudocodice

```
for (i = 0; i <= m; i++) L[i,0] = 0
for (j = 0; j <= n; j++) L[0,j] = 0

for (i = 1; i <= m; i++)
  for (j = 1; j <= n; j++)
    if (X[i] = Y[j])
      L[i,j] = L[i-1,j-1] + 1; R[i,j] = ↖
    else if (L[i,j-1] > L[i-1,j])
        L[i,j] = L[i,j-1; R[i,j] = ←
    else L[i,j] = L[i-1,j; R[i,j] = ↑
```

Osservazioni

- per ricostruire la LCS si parte da `(m,n)` e si va all'indietro seguendo le frecce, in corrispondenza di ogni freccia diagonale si ha un elemento della sequenza (scrivere l'algoritmo per esercizio)
- complessità temporale $T(m,n) = \Theta(mn)$
  costruzione di matrici $m \times n$
- assumendo come al solito $m \sim n$, l'algoritmo è quadratico
  molto meglio dell'algoritmo ingenuo esponenziale
- complessità spaziale $S(m,n) = \Theta(mn)$
  memorizzazione di matrici $m \times n$
- si può ottimizzare lo spazio ottenendo complessità lineare