Programmazione dinamica

Algoritmi e strutture dati

Ugo de'Liguoro, Andras Horvath

1

Sommario

- obiettivi:
  - migliorare la comprensione della complessità degli algoritmi ricorsivi e del modo di riutilizzare la soluzione di sottoproblemi del problema dato
- argomenti
  - Fibonacci con ricorsione, memoization e bottom-up
  - la programmazione dinamica
  - problema LCS (longest common subsequence)

2

Programmazione dinamica (PD) versus Divide-et-Impera (DI)

- PD e DI si basano sulla scomposizione ricorsiva di un problema in sottoproblemi
- DI è efficiente se i sottoproblemi sono indipendenti tra loro
- se i sottoproblemi non sono indipendenti tra loro, DI può richiedere tempi più lunghi del necessario (spesso esponenziali)
- mentre PD (ove applicabile) riesce a ridurli (spesso a polinomiali)

3

Condizioni di applicabilità

- per applicare PD occorre siano verificate le proprietà:

1. **sottostruttura della soluzione** (ottima nel caso di problemi di ottimizzazione): deve esserci una relazione fra le soluzioni (ottimali) degli sottoproblemi e la soluzione (ottimale) del problema
2. **sottoproblemi ripetuti**

4

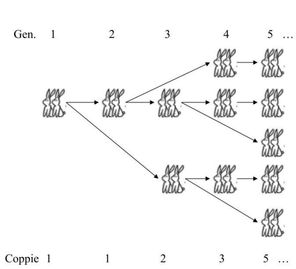
Fasi di sviluppo

1. caratterizzazione della struttura di una soluzione
2. definizione ricorsiva della soluzione
3. eliminare le ripetizioni mediante annotazione dei risultati più semplici (**memoization**)
4. sviluppo di un approccio **bottom-up**, e dunque iterativo

5

Successione di Fibonacci

- Quante coppie di conigli nascono dopo n generazioni a partire da una coppia, se ogni coppia genera una coppia per la gen. succ. ed una per quella ancora succ. e poi muore?



6

Successione di Fibonacci

- Quante coppie di conigli nascono dopo n generazioni a partire da una coppia, se ogni coppia genera una coppia per la gen. succ. ed una per quella ancora succ. e poi muore?
- definita come

$$f_0 = 0, f_1 = 1,$$
$$f_n = f_{n-2} + f_{n-1} \text{ per } n > 1$$

- fase 1 e fase 2 già fatte perché la definizione stessa della quantità che si vuole calcolare e ricorsiva

7

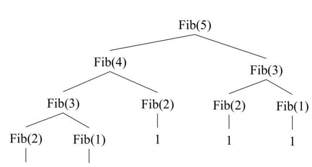
Algoritmo ricorsivo

```
Fib(n)
▷ Pre: n > 0 intero
▷ Post: ritorna l'n-mo numero della sequenza di Fibonacci
if n ≤ 2 then
    f ← 1
else
    f ← Fib(n − 1) + Fib(n − 2)
end if
return f
```

- complessità asintotica della soluzione ricorsiva?
- (differisce leggermente dalla definizione classica: calcola correttamente solo per $n > 0$)

8

Albero delle chiamate



- il numero di nodi al livello soddisfa:

$$N_1 = 1, N_2 = 1,$$
$$N_n = N_{n-2} + N_{n-1} + 1 \text{ per } n > 2$$

9

Albero delle chiamate

- il numero di nodi soddisfa una ricorsione a simile a quella della sequenza di Fibonacci
- ma è facile vedere che:

$$N_n > f_n$$

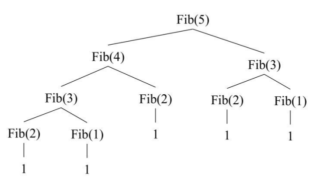
- formula di Binet:

$$f_n = \frac{1}{\sqrt{5}}(\Phi^n - (-1)^n \Phi^{-n}) \text{ con } \Phi = \frac{\sqrt{5}+1}{2}$$

- $\Phi > 1, -1 < (-1)^n \Phi^{-n} < 1$
- $N_n \in \Omega(\Phi^n)$, quindi ha crescita esponenziale
- dunque il tempo di calcolo di Fib è almeno esponenziale

10

Albero delle chiamate



- Fib ci mette così tanto perché ci sono tanti calcoli ripetuti
- fase3: annotiamo (memoizziamo) i risultati ottenuti strada facendo

11

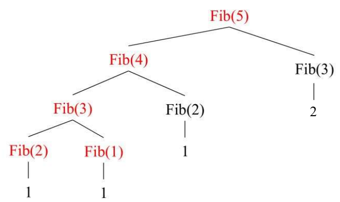
Fib con memoization

```
Fib-Memoization(n, memo)
▷ Pre: n > 0 intero, memo array di dim. > n
▷ Post: ritorna F_n = n-mo numero della sequenza di Fibonacci
if memo[n] ≠ nil then
    return memo[n]
end if      ▷ memo[n] non contiene alcun valore
if n ≤ 2 then
    f ← 1
else
    f ← Fib-Memoization(n−1, memo) + Fib-Memoization(n−2, memo)
end if
memo[n] ← f
return f
```

- spazio utilizzato da Fib-Memoization per l'array *memo* è $\Theta(n)$

12

Fib con memoization



- l'albero ha un sviluppo "lineare verso sinistra" e dunque sia tempo sia spazio di Fib-Memoization sono $\Theta(n)$

13

Fib bottom-up con array

```
Fib-bottomUp(n)
▷ Pre: n > 0 intero
▷ Post: ritorna F_n = n-mo numero della sequenza di Fibonacci
  if n ≤ 2 then
     return 1
  else
     Fib[1..n] sia un array di dimensione n
     Fib[1] ← 1, Fib[2] ← 1
     for i ← 3 to n do        ▷ inv: ∀j < i. Fib[j] = F_j
        Fib[i] ← Fib[i − 1] + Fib[i − 2]
     end for
  end if
  return Fib[n]
```

- tempo di Fib-BottomUp è $\Theta(n)$
- spazio utilizzato da Fib-BottomUp è $\Theta(n)$

14

Fib bottom-up senza array

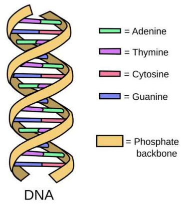
```
Fib-Iter(n)
▷ Pre: n > 0 intero
▷ Post: ritorna F_n = n-mo numero della sequenza di Fibonacci
  if n ≤ 2 then
     return 1
  else
     FibA ← 1, FibB ← 1
     for i ← 3 to n do       ▷ inv: FibA = F_{i−1}, FibB = F_{i−2}
        tmp ← FibA + FibB
        FibB ← FibA
        FibA ← tmp
     end for
  end if
  return FibA
```

- tempo di Fib-Iter è $\Theta(n)$
- spazio utilizzato da Fib-Iter è $\Theta(1)$

15

Massima sottosequenza comune - LCS

- obiettivo: Dati due filamenti di DNA vogliamo misurare la loro somiglianza calcolando la più lunga sottosequenza di basi che hanno in comune



16

Massima sottosequenza comune - LCS

- date due sequenze S_1 e S_2:

S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

- la massima sottosequenza comune è S_3

S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

S_3 = GTCGTCGGAAGCCGGCCGAA

17

Definizioni

- date due sequenze X e Z:
  $X = \langle x_1, \ldots x_m \rangle \qquad Z = \langle z_1, \ldots z_k \rangle$
- $Z \sqsubseteq X$, cioè Z è **sottosequenza** di X, se
  $k \leq m$
  $\exists f : \{1, \ldots, k\} \to \{1, \ldots, m\}$ crescente e t.c. $\forall j \leq k.\ z_j = x_{f(j)}$
- Z prende elementi non necessariamente consecutivi da X
- una sottosequenza Z è LCS(X, Y), cioè Z è **massima sottosequenza comune**, se
  $Z \sqsubseteq X \wedge Z \sqsubseteq Y \wedge Z$ ha lunghezza massima
- Z in generale non è unica

18

Definizioni

- **prefissi**:

  sia $X = \langle x_1, \ldots, x_m \rangle$ allora:

  $X_0 = \langle\rangle$

  $X_i = \langle x_1, \ldots, x_i \rangle$ se $1 \leq i \leq m$

- esempi:

$$
\begin{aligned}
S &= \langle A, G, C, A \rangle \\
S_0 &= \langle\rangle \\
S_1 &= \langle A \rangle \\
S_2 &= \langle A, G \rangle \\
S_3 &= \langle A, G, C \rangle \\
S_4 &= \langle A, G, C, A \rangle
\end{aligned}
$$

19

Proprietà della sottostruttura

- date due sequenze $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$ proviamo a mettere in relazione LCS(X, Y) e la soluzione di problemi LCS che coinvolgono anche prefissi di X e Y
- distinguiamo due casi: $x_m = y_n$ e $x_m \neq y_n$
- $x_m = y_n$:
  - se $Z = \langle z_1, \ldots, z_k \rangle$ è LCS(X, Y) allora quale sarà l'ultimo elemento di Z?
  - l'ultimo elemento di Z dovrebbe essere x_m
  - se $z_k = x_m$ allora Z_{k-1} è LCS di quali prefissi di X e Y?
  - Z_{k-1} è LCS(X_{m-1}, Y_{n-1})

20

Proprietà della sottostruttura

- date due sequenze $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$ proviamo a mettere in relazione LCS(X, Y) e la soluzione di problemi LCS che coinvolgono anche prefissi di X e Y
- distinguiamo due casi: $x_m = y_n$ e $x_m \neq y_n$
- $x_m \neq y_n$:
  - se $Z = \langle z_1, \ldots, z_k \rangle$ è LCS(X, Y) allora il suo ultimo elemento può essere x_m o y_n o qualcosa altro
  - di sicuro o x_m o y_n non serve per formare Z
  - Z è LCS(X_{m-1}, Y) oppure Z è LCS(X, Y_{n-1}) (o tutti e due)

21

Proprietà della sottostruttura

Lemma 1. Siano $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$; se $Z = \langle z_1, \dots, z_k \rangle$ è LCS(X, Y) e $x_m = y_n$ allora Z_{k-1} è LCS(X_{m-1}, Y_{n-1}) e $z_k = x_m$.

- **dimostrazione per assurdo di** Z_{k-1} è LCS(X_{m-1}, Y_{n-1})**:**
- assumiamo che

$$Z_{k-1} \text{ non è } \mathrm{LCS}(X_{m-1}, Y_{n-1})$$

- Z è LCS(X, Y) e quindi $Z \sqsubseteq X \wedge Z \sqsubseteq Y$ e quindi

$$Z_{k-1} \sqsubseteq X_{m-1} \wedge Z_{k-1} \sqsubseteq Y_{n-1}$$

- se Z_{k-1} non è LCS(X_{m-1}, Y_{n-1}) allora esiste $Z' = \langle z'_1, \dots, z'_h \rangle$ che è LCS(X_{m-1}, Y_{n-1}) e $|Z'| = h > k - 1$ ma allora se $x_m = y_n$ abbiamo

$$\langle z'_1, \dots, z'_h, x_m \rangle \sqsubseteq X \wedge \langle z'_1, \dots, z'_h, x_m \rangle \sqsubseteq Y \wedge |\langle z'_1, \dots, z'_h, x_m \rangle| > k$$

e questo contradice al "$Z = \langle z_1, \dots, z_k \rangle$ è LCS(X, Y)"

22

Proprietà della sottostruttura

Lemma 1. Siano $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$; se $Z = \langle z_1, \dots, z_k \rangle$ è LCS(X, Y) e $x_m = y_n$ allora Z_{k-1} è LCS(X_{m-1}, Y_{n-1}) e $z_k = x_m$.

- **dimostrazione per assurdo di** $z_k = x_m$**:**
- assumiamo per assurdo

$$z_k \neq x_m$$

- allora

$$\langle z_1, \dots, z_k, x_m \rangle \sqsubseteq X \wedge \langle z_1, \dots, z_k, x_m \rangle \sqsubseteq Y \wedge |\langle z_1, \dots, z_k, x_m \rangle| > k$$

e questo contradice al "$Z = \langle z_1, \dots, z_k \rangle$ è LCS(X, Y)"

23

Proprietà della sottostruttura

Lemma 2. Siano $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$; se $Z = \langle z_1, \dots, z_k \rangle$ è LCS(X, Y) e $x_m \neq y_n$ allora Z è LCS(X_{m-1}, Y) oppure Z è LCS(X, Y_{n-1}).

- è evidente che sia cosi perché se $x_m \neq y_n$ allora l'ultimo elemento di Z o non è x_m o non è y_n e quindi l'ultimo elemento di X o l'ultimo elemento di Y non serve per trovare il loro LCS

24

Proprietà della sottostruttura

Teorema. Indicando con LCS(X,Y) una LCS di X ed Y (che in generale non è unica) e supponendo che $X = X_m$ ed $Y = Y_n$ si ha che, per $i \leq m$ e $j \leq n$:

$$\mathrm{LCS}(X_i, Y_j) = \begin{cases} \langle\rangle & \text{se } i = 0 \text{ oppure } j = 0 \\ \mathrm{LCS}(X_{i-1}, Y_{j-1}) \frown x_i & \text{se } x_i = y_j \\ \mathrm{longest}(\mathrm{LCS}(X_{i-1}, Y_j), \mathrm{LCS}(X_i, Y_{j-1})) & \text{se } x_i \neq y_j \end{cases}$$

- segue da Lemma 1 e Lemma 2
- suggerisce anche una procedura ricorsiva
- usando $k = |X| + |Y| = m + n$ la relazione di ricorrenza $T(k) = 2T(k-1) + 1$ fornisce un limite superiore per l'ordine di grandezza del tempo di calcolo
- $T(k) \in \Theta(2^k) = \Theta(2^{m+n})$

25

Definizione ricorsiva

Costruiamo un tabella $c[0..m, 0..n]$ dove $c[i, j]$ sia la lunghezza di LCS(X_i , Y_j):

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

Simultaneamente costruiamo una tabella $b[1..m, 1..n]$ dove $b[i, j]$ punta alla pos. in c corrispondente alla soluzione ottima del sottoproblema LCS(X_i , Y_j):

$$b[i,j] = \begin{cases} \nwarrow & \text{se } x_i = y_j \\ \uparrow & \text{se } c[i-1, j] \geq c[i, j-1] \\ \leftarrow & \text{se } c[i-1, j] < c[i, j-1] \end{cases}$$

26

Algoritmo bottom-up

```
LCS-Bottom-Up(X, Y)
▷ Pre: X = ⟨x1, ... xm⟩, Y = ⟨y1, ..., yn⟩
▷ Post: ritorna le matrici c[0..m, 0..n] e b[1..m, 1..n]
m ← X.length, n ← Y.length
siano c[0..m, 0..n] e b[1..m, 1..n] due nuove tabelle
for j ← 0 to n do        ▷ la prima riga è inizializzata a 0
    c[0, j] ← 0
end for
for i ← 0 to m do        ▷ la prima colonna è inizializzata a 0
    c[i, 0] ← 0
end for
...
```

27

Algoritmo bottom-up

```
...
for i ← 1 to m do
    for j ← 1 to n do
        if x_i = y_j then
            c[i, j] ← c[i − 1, j − 1] + 1
            b[i, j] ← ↖
        else    ▷ x_i ≠ y_j
            if c[i − 1, j] ≥ c[i, j − 1] then
                c[i, j] ← c[i − 1, j]
                b[i, j] ← ↑
            else    ▷ c[i − 1, j] < c[i, j − 1]
                c[i, j] ← c[i, j − 1]
                b[i, j] ← ←
            end if
        end if
    end for
end for
return c, b
```

28

Algoritmo bottom-up

```
LCS-Bottom-Up(X, Y)
▷ Pre: X = ⟨x_1, ... x_m⟩, Y = ⟨y_1, ..., y_n⟩
▷ Post: ritorna le matrici c[0..m, 0..n] e b[1..m, 1..n]
m ← X.length, n ← Y.length
siano c[0..m, 0..n] e b[1..m, 1..n] due nuove tabelle
for j ← 0 to n do        ▷ la prima riga è inizializzata a 0
    c[0, j] ← 0
end for
for i ← 0 to m do        ▷ la prima colonna è inizializzata a 0
    c[i, 0] ← 0
end for
for i ← 1 to m do
    for j ← 1 to n do
        if x_i = y_j then
            c[i, j] ← c[i − 1, j − 1] + 1
            b[i, j] ← ↖
        else    ▷ x_i ≠ y_j
            if c[i − 1, j] ≥ c[i, j − 1] then
                c[i, j] ← c[i − 1, j]
                b[i, j] ← ↑
            else    ▷ c[i − 1, j] < c[i, j − 1]
                c[i, j] ← c[i, j − 1]
                b[i, j] ← ←
            end if
        end if
    end for
end for
return c, b
```

	$\langle\rangle$	A	G	C	A	T
$\langle\rangle$	0	0	0	0	0	0
G	0					
A	0					
C	0					

$\mathrm{LCF}(\langle G, A, C\rangle, \langle A, G, C, A, T\rangle) =$

Complessità: $\Theta(m \cdot n)$

29

Algoritmo bottom-up

```
LCS-Bottom-Up(X, Y)
▷ Pre: X = ⟨x_1, ... x_m⟩, Y = ⟨y_1, ..., y_n⟩
▷ Post: ritorna le matrici c[0..m, 0..n] e b[1..m, 1..n]
m ← X.length, n ← Y.length
siano c[0..m, 0..n] e b[1..m, 1..n] due nuove tabelle
for j ← 0 to n do        ▷ la prima riga è inizializzata a 0
    c[0, j] ← 0
end for
for i ← 0 to m do        ▷ la prima colonna è inizializzata a 0
    c[i, 0] ← 0
end for
for i ← 1 to m do
    for j ← 1 to n do
        if x_i = y_j then
            c[i, j] ← c[i − 1, j − 1] + 1
            b[i, j] ← ↖
        else    ▷ x_i ≠ y_j
            if c[i − 1, j] ≥ c[i, j − 1] then
                c[i, j] ← c[i − 1, j]
                b[i, j] ← ↑
            else    ▷ c[i − 1, j] < c[i, j − 1]
                c[i, j] ← c[i, j − 1]
                b[i, j] ← ←
            end if
        end if
    end for
end for
return c, b
```

	$\langle\rangle$	A	G	C	A	T
$\langle\rangle$	0	0	0	0	0	0
G	0	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$
A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$
C	0	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$

$\mathrm{LCF}(\langle G, A, C\rangle, \langle A, G, C, A, T\rangle) = \langle G, A\rangle$

Complessità: $\Theta(m \cdot n)$

30