1 I compitino, 18 novembre 2016 B

Si richiede un algoritmo che sfrutta la tecnica della programmazione dinamica per risolvere il seguente problema: si considerino due sequenze s_1 e s_2 di lettere dell'alfabeto italiano, di lunghezza n e m rispettivamente.

A ogni lettera dell'alfabeto è associato un numero naturale, mediante una funzione f prefissata.

Si vuole calcolare la lunghezza di una più lunga sottosequenza comune a s_1 e s_2 nella quale i caratteri appaiono in ordine lessicografico crescente e i numeri ad essi associati appaiono in ordine decrescente.

1.1 Variabili che servono per risolvere il problema

La programmazione dinamica serve per risolvere un problema considerando di aver già risolto tutti i sottoproblemi (prefissi), e potendo usare la soluzione di essi per calcolare la soluzione del problema.

Per immagazzinare le soluzioni dei sottoproblemi viene utilizzata una matrice c, di dimensione $n \times m$. A ogni sottoproblema i, j corrisponde una casella $c[i, j]$ che contiene il valore di una delle sottosequenze comuni più lunghe che rispettano i vincoli dell'algoritmo.

Con s_1 di lunghezza n e s_2 di lunghezza m, il problema finale avrà dimensione $n,\ m$. Si definiscono indici i, j che indicano ogni sottoproblema, tale che $0 \leq i \leq n$ e $0 \leq j \leq m$. In totale, considerando anche i casi limite con prefisso che corrisponde alla stringa vuota, si hanno $(n+1)(m+1)$ sottoproblemi.

Le sottostringhe $s_{1,i}$ e $s_{2,j}$ sono prefissi di lunghezza rispettivamente i e j delle stringhe di partenza s_1 e s_2 di lunghezza n e m, con $0 \leq i \leq n$ e $0 \leq j \leq m$.

In questo caso, il problema è definito come:
trovare la lunghezza di una delle più lunghe sottosequenze di s_1, s_2 tale che i caratteri siano in ordine lessicografico crescente e i numeri associati in ordine decrescente.

Ogni sottoproblema è definito come:
trovare la lunghezza di una delle più lunghe sottosequenze di $s_{1,i}$, $s_{2,j}$, con $0 \leq i \leq n$ e $0 \leq j \leq m$, tale che i caratteri siano in ordine lessicografico crescente e i numeri associati in ordine decrescente.

Per risolvere questo problema, nel caso in cui $s_{1,i} = s_{2,j}$ è necessario conoscere il valore numerico dell'ultimo carattere della più lunga sottosequenza comune precedente, in modo da poterlo confrontare con il valore corrente.

Il problema viene esteso:
trovare la lunghezza di una delle più lunghe sottosequenze di $s_{1,i}$, $s_{2,j}$ tale che i caratteri siano in ordine lessicografico crescente e i numeri associati in ordine decrescente ***e che termini con*** *$s_{1,i}$, $s_{2,j}$ (nel caso in cui esse coincidano).*

La variabile associata a ogni problema è:
$c[i, j]$ che contiene la lunghezza di una delle più lunghe sottosequenze di $s_{1,i}$, $s_{2,j}$ che abbia lettere crescenti e numeri associati decrescenti (nel caso in cui $s_{1,i}, s_{2,j}$ coincidano), e contiene 0 altrimenti.

Si ricorda che ognuna di queste variabili è considerabile come una **black-box**: si può utilizzare ma non è possibile conoscerne il contenuto (si assume di aver risolto i sottoproblemi di dimensione minore).

1.2 Equazione di ricorrenza

Un algoritmo ricorsivo non è efficace, perché calcolerebbe più volte lo stesso risultato, quindi si ricorre alla programmazione dinamica per risolvere il problema.

Le soluzioni dei sottoproblemi non sono ancora note, ma è possibile utilizzarle per calcolare le soluzioni successive. L'unico caso conosciuto è il caso limite: se una delle due sequenze è vuota, la lunghezza della più lunga sottosequenza è 0 ($i = 0 \vee j = 0$).

Questo si può esprimere nel seguente modo:

$$c[i, 0] = 0 \; con \; \{0 \leq i \leq n\}, \; c[0, j] = 0 \; con \; \{0 \leq j \leq m\}$$

Si introduce il sottoproblema tale per cui $s_{1,i} \neq s_{2,j}$: in questo caso, non è possibile trovare la più lunga sottosequenza comune che termini con $s_{1,i}, s_{2,j}$ con valore maggiore di 0. Di conseguenza si assegna 0 alla casella corrispondente.

$$c[i, j] = 0 \text{ con } s_{1,i} \neq s_{2,j}$$

Avendo risolto il caso con il prefisso vuoto, i sottoproblemi da risolvere diventano al più mn. L'equazione di ricorrenza per un generico sottoproblema i, j è:

$$c[i, j] = \begin{cases} 1 + max\{c[h, k]\} \text{ con } \{1 \leq h < i, \; 1 \leq k < j\} & \text{se } s_{1,h} < s_{1,i}, \; f(s_{1,i}) < f(s_{1,h}) \\ 1 & \text{altrimenti} \end{cases}$$

1.3 Soluzione del problema

La più lunga sottosequenza comune crescente con numeri decrescenti tra $s_{1,i}$ e $s_{2,j}$ è data dal massimo valore tra le soluzioni delle più lunghe sottosequenze comuni per i sottoproblemi di dimensione minori.

La soluzione del problema corrisponde al massimo tra tutte le caselle della matrice.

$$max\{c[i, j] \mid 0 \leq i \leq n \wedge 0 \leq j \leq m\}$$

L'algoritmo funziona in questo modo: in ogni casella di c viene inserito 0 se i due caratteri in posizione i, j sono diversi, altrimenti viene cercato il massimo nella sottomatrice precedente utilizzando due indici ausiliari h, k. Il valore corrispondente deve rispettare le seguenti condizioni: il numero precedente dev'essere maggiore, e la lettera precedente dev'essere minore della lettera associata.

Il risultato ottenuto viene incrementato di 1, perché è stato trovato un altro carattere uguale.

1.4 Algoritmo in pseudocodice

Viene illustrata la soluzione bottom-up mediante un algoritmo iterativo, risolvendo ogni sottoproblema una volta sola.

Algorithm 1 LGCS con numeri decrescenti

```
function LGCS NUMERI DECRESCENTI(s_1, s_2)
    for i ← 0 to n do
        c[i, 0] ← 0
    end for
    for j ← 0 to m do
        c[0, j] ← 0
    end for
    max ← 0
    for i ← 1 to n do
        for i ← 1 to n do
            if s_1[i] ≠ s_2[j] then
                c[i, j] ← 0
            else
                temp ← 0
                for h ← 1 to i − 1 do
                    for k ← 1 to j − 1 do
                        if s_{1,i} > s_{1,h} ∧ f(s_{1,i}) < f(s_{1,h}) ∧ c[h, k] > temp then
                            temp ← c[h, k]
                        end if
                    end for
                end for
                c[i, j] ← 1 + temp
            end if
            if c[i, j] > max then
                max = c[i, j]
            end if
        end for
    end for
end function
```

1.5 Valutazione del tempo di esecuzione

Il tempo di esecuzione di questo algoritmo è approssimativamente $O(n^2m^2)$. Ciò deriva dai cicli *for* innestati: la matrice viene interamente analizzata casella per casella, e poi c'è la ricerca del massimo valore nella sottomatrice.

Lo spazio utilizzato è $\Theta(nm)$, per la matrice.