Supersequenza comune minimale

Una stringa P è una supersequenza di una stringa T se T è una sottosequenza di P. Scrivere un algoritmo che restituisce la lunghezza della *supersequenza comune minimale* di due stringhe P, T, ovvero la più piccola supersequenza di entrambe le stringhe. Discutere correttezza e complessità dell'algoritmo proposto.

Esempio: L'unica supersequenza comune minimale di `AB` e `BC` è `ABC`, e la sua lunghezza è pari a 3.

Esempio: Esistono due supersequenze comuni minimali di `DAB` e `DCB`, ovvero `DACB` e `DCAB`, e la loro lunghezza è pari a 4.

I Promessi Sposi

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive, par che renda ancor più sensibile all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda rincomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni."

Quante volte questo testo contiene la sottosequenza "lucia"?

I Promessi Sposi

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive, par che renda ancor più sensibile all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda rincomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni."

Quante volte questo testo contiene la sottosequenza "lucia"?
Alcune considerazioni:

- Due sottosequenze sono diverse (e quindi vanno contate separatamente) se esiste almeno una differenza negli insiemi di caratteri utilizzati.
- Esempio: "did you go" contiene due volte la sottosequenza "dog"....

Vettori ordinati

Si scriva un algoritmo che preso in input n e k, restituisca il numero totale di vettori distinti di lunghezza n, contenenti valori interi compresi fra 1 e k, ordinati dalla relazione $\leq$. Si discuta la correttezza e la complessità dell'algoritmo proposto.
Ad esempio, dati $n = 4$ e $k = 3$, questi sono i possibili vettori ordinati:

$$[1,1,1,1],[1,1,1,2],[1,1,1,3],[1,1,2,2],[1,1,2,3],$$
$$[1,1,3,3],[1,2,2,2],[1,2,2,3],[1,2,3,3],[1,3,3,3],$$
$$[2,2,2,2],[2,2,2,3],[2,2,3,3],[2,3,3,3],[3,3,3,3]$$

e quindi il valore da restituire è 15.

Quadrato binario

Sia $A[1 \ldots n, 1 \ldots n]$ una matrice di valori booleani 0/1. Scrivere un algoritmo che restituisce la dimensione del più grande quadrato composto da valori 1. Ad esempio, nella matrice seguente, i quadrati di dimensione massima (ve ne sono due, di cui uno evidenziato in rosso) hanno dimensione pari a 4.

```
1 0 1 0 1 0 0
1 0 1 1 1 1 0
0 1 1 1 1 1 0
0 0 1 1 1 1 0
1 1 1 1 1 1 0
1 1 1 1 1 1 0
```

Spoiler alert!

Supersequenza comune minimale

La lunghezza della più lunga supersequenza comune può essere calcolata tramite la seguente espressione ricorsiva:

$$DP[i][j] = \begin{cases} i & \text{se } i \geq 0 \wedge j = 0 \\ j & \text{se } i = 0 \wedge j \geq 0 \\ DP[i-1][j-1] + 1 & \text{se } i > 0 \wedge j > 0 \wedge p_i = t_j \\ \min\{DP[i-1][j], DP[i][j-1]\} + 1 & \text{se } i > 0 \wedge j > 0 \wedge p_i \neq t_j \end{cases}$$

- Nel caso una delle stringhe sia vuota, tutti i caratteri dell'altra stringa devono essere presenti nella supersequenza, da cui i due casi base.
- Se gli ultimi caratteri delle due stringhe sono uguali, si considere il sottoproblema in cui viene rimosso l'ultimo caratteri di entrambi e si aggiunge $+1$ per contare questo carattere.
- Nel caso siano diversi, si prenderanno i due sottoproblemi in cui si rimuove l'ultimo carattere di una delle stringhe e si aggiunge $+1$ per contare questo carattere, prendendo il valore più piccolo fra i due.

Supersequenza comune minimale

```
scs(Item[] P, Item[] T, int n, int m)
int[][] DP = new int[0...n][0...m]
for i = 0 to n do
    DP[i][0] = i
for j = 0 to m do
    DP[0][j] = j
for i = 1 to n do
    for j = 1 to m do
        if P[i] == T[j] then
            DP[i][j] = DP[i − 1][j − 1] + 1
        else
            DP[i][j] = min(DP[i − 1][j], DP[i][j − 1]) + 1
return DP[n][m]
```

I Promessi Sposi

Input: Testo $T[1 \dots n]$, pattern $P[1 \dots m]$
Sia $DP[i][j]$ il numero di occorrenze del prefisso j-esimo del pattern $P(j)$ come sottosequenza del prefisso i-esimo del testo $T(i)$.

$$DP[i][j] = \begin{cases} 0 & i = 0 \wedge j > 0 \\ 1 & j = 0 \\ DP[i-1][j] + DP[i-1][j-1] & i > 0 \wedge j > 0 \wedge T[i] = P[j] \\ DP[i-1][j] & \text{altrimenti} \end{cases}$$

I Promessi Sposi

Utilizziamo un vettore $DP[0 \ldots m]$ invece che una matrice, in quanto il valore si ottiene a partire dalla sola riga precedente.

---

lucia(ITEM[ ] T, ITEM[ ] P, **int** n, **int** m)

```
int[ ] DP = new int[0...n]
int[ ] DP' = new int[0...n]
DP[0] = 1
for j = 1 to m do DP[j] = 0
for i = 1 to n do
    for j = 0 to m do DP'[j] = DP[j]
    for j = 1 to m do
        if T[i] == T[j] then
            DP[j] = DP'[j] + DP'[j − 1]
        else
            DP[j] = DP'[j]
return DP[m]
```

---

5 Maggio e Promessi Sposi

---- Forwarded Message ----
Subject: Sottosequenza Promessi sposi
Date: Fri, 11 Dec 2015 00:25:41 +0100
To: Alberto Montresor <alberto.montresor@unitn.it>

Il 5 maggio nei promessi sposi senza considerare spazi, punteggiatura e numeri, ma considerando gli accenti ci sta:

219754653015166309795736175938257695138575835630252623797897783379476158171917574339842832197562154239662334744219763715818422816009859675872567801017800136565973566045203119340799723612777962220975263078675519750712637479432237655210391918486018747374239424385310182137281795662107004225375841957767155366649493437946947434130448636772119986920548451717813640098831758107771539361489284456030355662857753722957658724970416074137670807561854959314707635812057348445333068267511860816130432217972866059043784081128537958885066938200067286955157502356301533012859308257726902061928895201187397008635926385087734504207402724330995069654934446734109698508036913355586284550592994928187284736568396263368466837671143105426910996086013014180403835018234891339555786532695278342722343647414316040385168266548204572245828285669254568831700065606562316618108110528823557597557235207472652875237693750708795898738364470324968401496146509587976160485483512050002468507459521611876935161859069786239098798726481408600448745883309202330 7

Allego lo script in python che ha generato il risultato (con testo completo dei promessi sposi e del 5 maggio). Ovviamente ha tempi assurdi per svariati motivi.

Vettori ordinati

Sia $DP[n][k]$ il numero di vettori ordinati di lunghezza n, contenenti k valori distinti (compresi fra 1 e k). $DP[n][k]$ può essere calcolato in maniera ricorsiva come segue:

$$DP[n][k] = \begin{cases} 1 & n = 0 \\ \sum_{i=1}^{k} DP[n-1][i] & n > 0 \end{cases}$$

Ovviamente, questo richiede una tabella $O(nk)$, per calcolare ogni elemento delle quale saranno necessarie $O(k)$ operazioni, per un costo totale di $O(nk^2)$.

Vettori ordinati

```
int sortedPermRec(int n, int k, int[][] DP)
if n == 0 then
    return 1
if DP[n][k] < 0 then
    DP[n][k] = 0
    for i = 1 to k do
        DP[n][k] = DP[n][k] + sortedPermRec(n − 1, i, DP)
return DP[n][k]
```

Vettori ordinati

int sortedPerm(**int** n, **int** k)

int[ ][ ] $DP =$ **new int**$[0 \ldots k][1 \ldots k]$
for $i = 1$ **to** n **do**
    for $j = 1$ **to** k **do**
        $DP[i][j] = -1$
return sortedPermRec(n, k, DP)

Vettori ordinati

Una soluzione alternativa, più efficiente, calcola $DP[n][k]$ nel modo seguente:

$$DP[n][k] = \begin{cases} k & n = 1 \\ 0 & k = 0 \\ DP[n-1][k] + DP[n][k-1] & \text{altrimenti} \end{cases}$$

Vettori ordinati

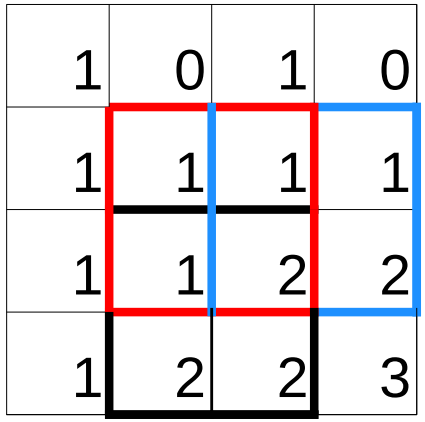
sortedPerm(**int** n, **int** k)

int[ ][ ] DP = **new int**$[0 \ldots k][1 \ldots k]$
for $i = 1$ **to** n **do**
  $DP[i][0] = 0$
for $j = 1$ **to** k **do**
  $DP[1][j] = j$
for $i = 2$ **to** n **do**
  for $j = 1$ **to** k **do**
    $DP[i][j] = DP[i-1][j] + DP[i][j-1]$
return $DP[n][k]$

Ovviamente, questo richiede una tabella $O(nk)$, per calcolare ogni elemento delle quale saranno necessarie $O(1)$ operazioni, per un costo totale di $O(nk)$.

Quadrato binario

1	0	1	0
1	1	1	1
0	1	1	1
1	1	1	1



Quadrato binario

$DP[i][j]$ contiene la dimensione del più grande quadrato composto da soli 1 il cui angolo in basso a destra sia nella posizione (i, j)

$$DP[i][j] = \begin{cases} 0 & A[i][j] = 0 \\ 1 & A[i][j] = 1 \wedge (i = 1 \vee j = 1) \\ \min\{DP[i-1][j], & \\ \quad DP[i-1][j-1], & \\ \quad DP[i][j-1]\} + 1 & \text{altrimenti} \end{cases}$$