Martedì Gaetano M63001226
Salzillo Biagio M63001227

Corso di Algoritmi e Strutture Dati
Homework #3

Problema 3.1
Il palindromo è una sequenza di caratteri che, letta al contrario, rimane invariata. Ad esempio, GAG e MADAM sono palindromi, ma ADAM no. Assumiamo che anche la stringa vuota sia un palindromo. Da qualsiasi stringa non palindromica, si può sempre ottenere una sotto-sequenza palindromica togliendo alcune lettere: ad esempio, data la stringa ADAM, si può rimuovere la lettera M e ottenere ADA. Scrivere un programma per determinare la lunghezza del palindromo più lungo che puoi ottenere da una stringa rimuovendo zero o più caratteri. Il programma deve ricevere una stringa (di lunghezza inferiore a 1000 caratteri) e stampare la lunghezza del palindromo più lungo che si può ottenere rimuovendo zero o più caratteri.

Svolgimento:

L'algoritmo risolutivo è progettato secondo la strategia di Programazione Dinamica.

5 STEPS

1. **Definizione del sotto-problema:**

$$DP(i,j) \;\; con\; i,j = 1, \dots, n$$

$DP(i,j)$ è il sotto-problema che calcola la lunghezza della massima sotto-sequenza palindroma contenuta nella sotto-stringa $x[i:j]$.

$$\#\, sottoproblemi = \Theta(n^2) \tag{1}$$

2. **Scelte del sotto-problema:**

Per le scelte da effettuare per ogni sotto-problema dobbiamo distinguere due casi:

(a) se $x[i] = x[j]$, selezioniamo il sotto-problema $DP(i+1, j-1)$; $\implies$ un'unica scelta

(b) se $x[i] \neq x[j]$, avremo due sotto-problemi $DP(i+1,j)$ e $DP(i,j-1)$, dei quali selezioniamo quello che restituisce la lunghezza della massima sotto-sequenza palindroma. $\implies$ due possibili scelte

$$\#scelte = \Theta(1) \tag{2}$$

3. **Ricorrenza:**

Per questo problema di programmazione dinamica possiamo scrivere la ricorrenza in questo modo:

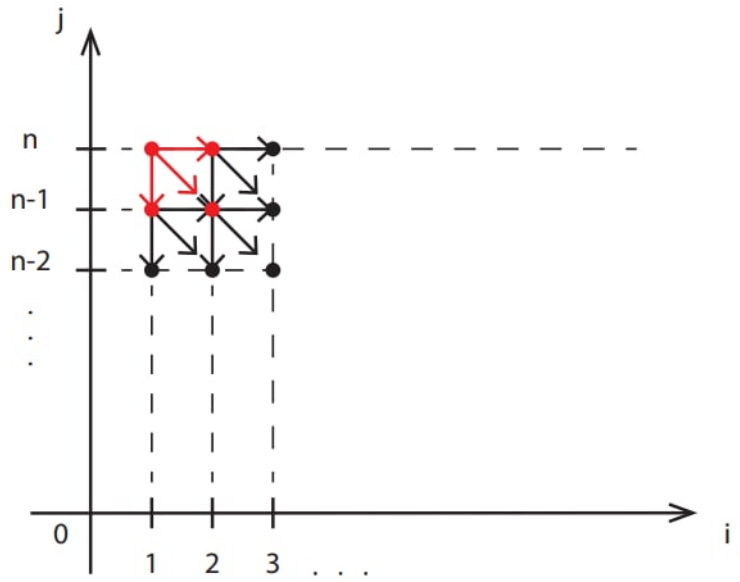
$$DP(i,j) = \begin{cases} 0 & se\ i > j \\ 1 & se\ i = j \\ 2 + DP(i+1, j-1) & se\ x[i] = x[j] \\ max(DP(i+1,j), DP(i,j-1)) & altrimenti \end{cases}$$

con $i, j = 1, \ldots, n$

$$tempo/sottoproblema = \Theta(1)$$

4. **Aciclicità - Memoization:**

   Il grafo dei sotto-problemi è aciclico poiché nella ricorrenza l'indice i viene sempre incrementato e invece l'indice j viene sempre decrementato.



$$T = \#\, sottoproblemi \cdot tempo/sottoproblema = \Theta(n^2) \cdot \Theta(1) = \Theta(n^2)$$

Per la Memoization utilizziamo una matrice di dimensione n x n, inizializzata in questo modo:

(a) $mem[i][j] = 1$ se $i = j$;
(b) $mem[i][j] = 0$ se $i > j$;
(c) $mem[i][j] = -1$ se i$< j$.

Ad ogni chiamata ricorsiva, i risultati vengono salvati in questa matrice, e all'occorrenza prelevati, evitando così di ripetere più volte gli stessi calcoli già effettuati dalle precedenti chiamate.

5. **Problema originario:**

   Il problema originario corrisponde al sotto-problema DP(1,n), quindi non c'è extra-time.

Algoritmo:

```
1: function PALINDROME-SUBSEQUENCE(X)
2:     n ← length[X]
3:     dichiara memo[n][n]
4:
5:     INIT-MEMO(memo, n)
6:
7:     ris ← DP(X, 1, n, memo)
8:     return ris
9: end function
```

```
10:
11: function INIT-MEMO(memo, n)
12:     for i ← 1 to n do
13:         for j ← 1 to n do
14:             if i < j then
15:                 memo[i][j] ← −1
16:             else
17:                 if i = j then
18:                     memo[i][j] ← 1
19:                 else
20:                     memo[i][j] ← 0
21:                 end if
22:             end if
23:         end for
24:     end for
25: end function
26:
27: function DP(X, i, j, memo)
28:     dichiara due variabili a, b
29:
30:     if memo[i][j] = −1 then
31:         if X[i] ≠ X[j] then
32:             a ← DP(X, i + 1, j, memo)
33:             b ← DP(X, i, j − 1, memo)
34:             if a > b then
35:                 memo[i][j] ← a
36:             else
37:                 memo[i][j] ← b
38:             end if
39:         else
40:             memo[i][j] ← 2 + DP(X, i + 1, j − 1, memo)
41:         end if
42:     end if
43:
44:     return memo[i][j]
45: end function
```