INTRODUZIONE

INTRODUCIAMO LA TECNICA ATTRAVERSO L'ESEMPIO DEI NUMERI DI FIBONACCI.

$$\left.\begin{array}{l} fib_0 = 0 \\ fib_1 = 1 \end{array}\right\} \text{CASI BASE}$$

$$fib_{i+1} = fib_i + fib_{i-1}$$

L'OVVIO ALGORITMO RICORSIVO PER CALCOLARLI HA RELAZIONE DI RICORRENZA: $\boxed{T(n) = T(n-1) + T(n-2) + \underbrace{\Theta(1)}_{\text{COMPLESSITÀ SOMMA}}}$

ALGORITMO

```
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]
```

DEFINIZIONE PROGRAMMAZIONE DINAMICA: SI BASA SU UNA DEFINIZIONE RICORSIVA SU SOTTOPROBLEMI PIÙ PICCOLI (AD ESEMPIO DIVIDE ET IMPERA) UTILIZZANDO LA TECNICA BOTTOM-UP, OSSIA, PER PRIMA COSA SI RISOLVONO I SOTTOPROBLEMI BASE E A PARTIRE DA QUESTI SI RISOLVONO I SUCCESSIVI FINO AD ARRIVARE A QUELLO RICHIESTO, MEMORIZZANDO I RISULTATI INTERMEDI. IN QUESTO MODO, NEL CASO IN CUI UN SOTTOPROBLEMA VENGA UTILIZZATO PER RISOLVERE MOLTI PROBLEMI DI LIVELLO SUPERIORE, È POSSIBILE CALCOLARNE LA SOLUZIONE UNA VOLTA SOLA, E QUINDI L'APPROCCIO RISULTA VANTAGGIOSO. UN ESEMPIO È LA LONGEST COMMON SEQUENCE (LCS)

LONGEST COMMON SEQUENCE

CONSIDERIAMO ORA IL PROBLEMA DELLA PIÙ LUNGA SOTTOSEQUENZA COMUNE. UNA SOTTOSEQUENZA DI UNA SEQUENZA S È UNA SEQUENZA DI ELEMENTI DI S PRESI NELLO STESSO ORDINE, OSSIA È INDIVIDUATA DA UNA SEQUENZA DI INDICI DI S.

PROBLEMA: DATE DUE SOTTOSEQUENZE TROVARE UNA SOTTOSEQUENZA COMUNE DI LUNGHEZZA MASSIMA.

X = A B D A C C A B

Y = B A A D C B B C C B

NOTA: NON SI POSSONO INCROCIARE LE LINEE

ALGORITMI

BRUTE-FORCE: GENERA TUTTE LE SOTTOSEQUENZE DI X CHE SONO 2^n, OGNUNA COSTA n PERCHÉ CONFRONTO CON OGNI ELEMENTO DI Y E QUINDI COMPLESSITÀ $\boxed{n \times 2^n}$.

L'APPROCCIO MIGLIORE È LA FORMULAZIONE INDUTTIVA. L'IDEA È QUELLA DI NON RICOMINCIARE DA 0 OGNI VOLTA, MA RIUSARE I RISULTATI PARZIALI. DEFINIAMO $LCS(i, j)$ COME LA PIÙ LUNGA SOTTOSEQUENZA COMUNE CONSIDERANDO SOLO I PRIMI i CARATTERI DELLA PRIMA STRINGA (X) E I PRIMI j DELLA SECONDA STRINGA (Y). ABBIAMO 3 CASI:

1) CASO BASE

$$LCS(0, j) = [\,] \quad E \quad LCS(i, 0) = [\,]$$

SE UNA DELLE DUE STRINGHE È VUOTA, NON POSSONO ESSERCI CARATTERI IN COMUNE.

2) I caratteri coincidono $X(i) = Y(j)$

Se l'ultimo carattere che stiamo guardando è uguale in entrambe le stringhe, allora quel carattere fa sicuramente parte della soluzione ottimale.

$LCS(i-1, j-1) \cdot (i, j)$ — Significa: prendi il risultato precedente e aggiungi questa coppia di indici

3) I caratteri non coincidono

Se gli ultimi caratteri sono diversi, non possono far parte della soluzione.

Dobbiamo scartarne uno, ma non sappiamo quale sia la scelta migliore.

$$\max\big(LCS(i-1, j),\ LCS(i, j-1)\big)$$

Algoritmo completo:

$LCS(0, j) = [\,]$

$LCS(i, 0) = [\,]$

per $i \neq 0$, $j \neq 0$

se $X(i) = Y(j)$ $\quad LCS(i, j) = LCS(i-1, j-1) \cdot (i, j)$

altrimenti $\quad LCS(i, j) = \max\big(LCS(i-1, j),\ LCS(i, j-1)\big)$

La complessità è come torri di Hanoi, ovvero algoritmo esponenziale.

$$\boxed{T(n, m) = T(n-1, m) + T(n, m-1) + 1}$$

Migliorare algoritmo

L'algoritmo LCS può essere migliorato abbassando la complessità, costruendo una matrice di grandezza $n+1$ righe ed $m+1$ colonne, quel $+1$ è dato dalla sequenza vuota:

		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

Sempre tutti 0

Se riga e colonna sono uguali, allora prendo l'elemento in diagonale a sinistra e aggiungo 1.

Se riga e colonna sono diversi allora prendo il massimo tra il valore sopra ed a sinistra

Ci dice quanto è lunga la sequenza

SE VIENE CHIESTA SOLO LA LUNGHEZZA DELLA SEQUENZA POSSO FERMARMI A 4, ALTRIMENTI SE MI VIENE CHIESTO ANCHE QUALE' LA SEQUENZA SEGUO L'ORDINE DELLE FRECCE IN DIAGONALE. NEL NOSTRO ESEMPIO LA SEQUENZA E': A C A B

PSEUDO CODICE

```
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] = ↑
```

COMPLESSITA'

$O(m\,n)$