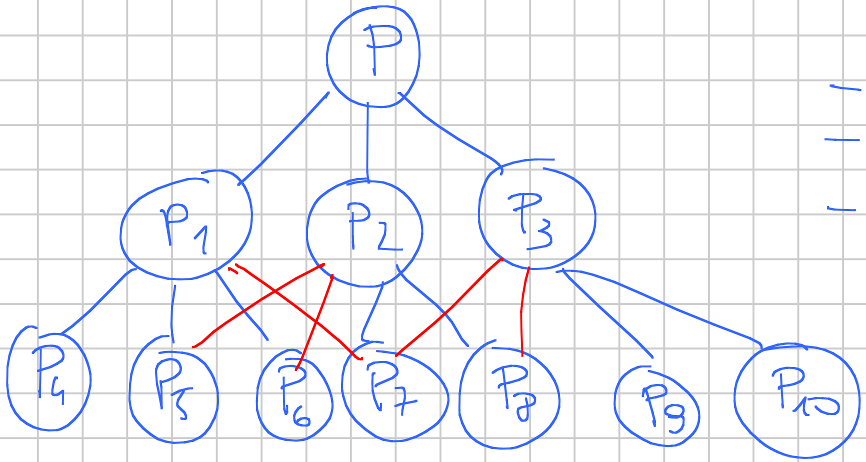
PROGRAMMAZIONE DINAMICA



- sottoproblemi indipendenti
- operano su sottoinsiemi disgiunti di dati
- ogni sottoproblema incontrato è risolto <u>una sola</u> volta

⇒ Ricorsione D&I

OK

sottoproblemi NON indipendenti.

es: P_7 occorre nella decomposizione di P_1, P_2 e P_3

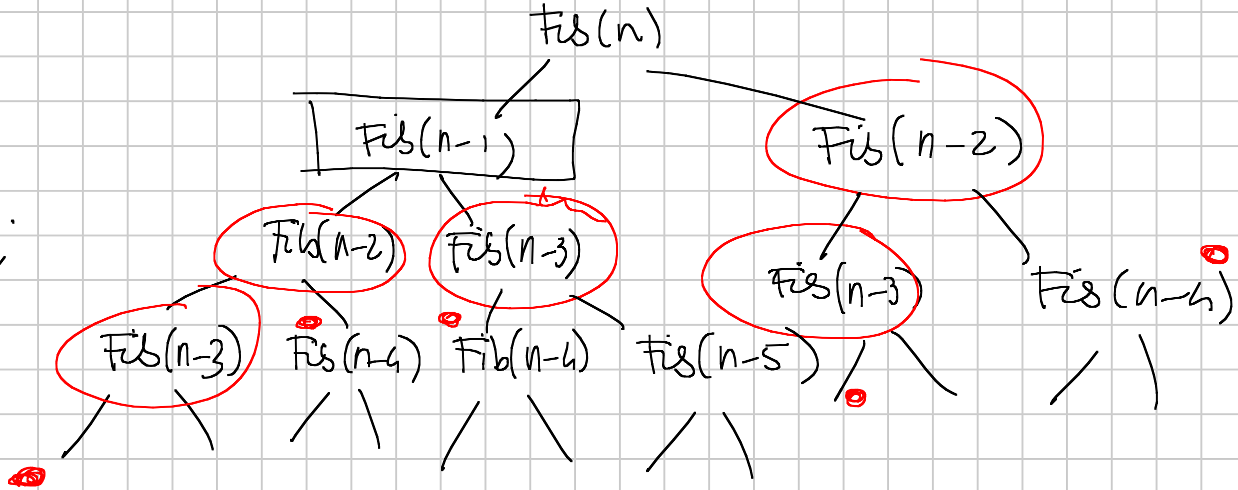
⇒ con la ricorsione si risolve 3 volte!!

↳ D&I, Ricorsione → inefficienti

ESEMPIO: Calcolo dei numeri di Fibonacci

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n \geq 2 \end{cases}$$

```
Fib(n)    // calcola Fn
    if (n == 0)  return 0;
    if (n == 1)  return 1;
    return Fib(n-1) + Fib(n-2);
```



il sottoproblema Fib(n-k) si incontra e si risolve F_{k+1} volte

$k = n$ Fib(0) F_{n+1} volte $\simeq \phi^n$

$$T(n) := \begin{cases} \Theta(1) & n=0,1 \\ T(n-1)+T(n-2)+\Theta(1) & n>1 \end{cases}$$

$$T(n) = T(n-1)+T(n-2)+\underline{\Theta(1)} > T(n-1)+T(n-2) > T(n-2)+T(n-2)$$

$$T(n) > \underbrace{2T(n-2)} > 2\,\underbrace{2T(n-4)} = \underbrace{4\,T(n-4)}$$

$$> 4\cdot 2\cdot T(n-6) = 2^3\,\underbrace{T(n-2\cdot 3)} > 2^3\cdot 2\cdot T(n-2\cdot 3-2) =$$

$$> 2^4\,T(n-2\cdot 4) \quad --- \quad > 2^i\,T(n-2\cdot i)$$

n pari, $i = \frac{n}{2}$ $\quad T(n) > 2^{\frac{n}{2}}\,T\left(n-2\cdot\frac{n}{2}\right) = 2^{\frac{n}{2}}\cdot \underbrace{T(0)}_{\Theta(1)}$

n dispari, $i = \frac{n-1}{2}$ $\quad T(n) > 2^{\frac{n-1}{2}}\,T\left(n-\frac{n-1}{2}\cdot 2\right) = 2^{\frac{n-1}{2}}\,\underbrace{T(1)}_{\Theta(1)}$

$$\Rightarrow \quad T(n) > 2^{\lfloor \frac{n}{2} \rfloor}\cdot\Theta(1)$$

$$T(n) = \Omega\left(2^{\lfloor \frac{n}{2} \rfloor}\right)$$

Versione PD (iterativa)

0	1	1	2	3	5	8	13	21	34	55	89
0	1	2	3	4	5	6	7	8	9	10	11

F_{11}

FibPD(n)

```
F = nuovo array di dim n+1
F[0] = 0;
F[1] = 1;
for i = 2 to n
    F[i] = F[i-1] + F[i-2];
return F[n];
```

$$T(n) = \underbrace{\Theta}_{\text{operazioni}}(n)$$

$$S(n) = \Theta(n)$$

Titolo nota 26/02/2016

Fib2(n)

if (n == 0) return 0;
if (n == 1 OR n == 2) return 1;
a = b = 1
for i = 3 to n
{ c = a + b; a = b; b = c }
return b;

$T(n) = \Theta(n)$ # operazioni

$S(n) = \Theta(1)$

OSSERVAZIONE

I = istanza di input = n

$|I|$ = # bit per scrivere il valore n = $\Theta(\log n)$

esponenziale in $|I| = \Theta(\log n)$

lineare nel valore n

algoritmo "pseudopolinomiale"

ESERCIZIO

Calcolare F_n con $\Theta(\log n)$ operazioni

Suggerimento $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ x induzione

$$\boxed{A^{n-1} = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix}}$$

PD (problemi di ottimizzazione)

1. SOTTOSTRUTTURA OTTIMA

   la soluzione ottima del problema deriva dalle soluzioni ottime dei sottoproblemi

2. SOVRAPPOSIZIONE (RIPETIZIONE) dei sottoproblemi

PD: struttura dell'algoritmo

4 fasi

(1) Definizione dei sottoproblemi, dimensionamento della tabella

Fibonacci: sottoproblema: calcolo di F_i

tabella: array di dim $n+1$ $(0, 1, \dots, n)$

(2) Soluzione diretta dei sottoproblemi elementari e memorizzazione del risultato nella tabella

$F(0) = 0, \quad F(1) = 1$

(3) definizione della regola ricorsiva per ottenere la soluzione di un sottoproblema a partire dalle soluzioni dei sottoproblemi già risolti
(regola di riempimento della tabella) $F(i) = F(i-1) + F(i-2)$

4 Restituzione del risultato del problema originario

return F(n)

LCS (longest common subsequence)

trovare la più lunga sottosequenza comune a due stringhe

Definizioni

X, Y, Z stringhe

Z è Sottosequenza (S) di X se si può ottenere da X ~~dei~~ cancellando uno o più caratteri

X = SPI~~E~~~~G~~A~~R~~~~E~~ → SPIA

Z = SPIA è S di X

Z è CS (S comune) a X e Y se è S di X e di Y

X = SPIEGARE, Y = OSPITARE

Z = SPIA è SC di X e Y

Z è una LCS se è una CS a X e Y con il maggior numero di caratteri

$X = SPIEGARE$ $Y = OSPITARE$

$Z = SPIARE$ è LCS

\# sottosequenze di una stringa di m caratteri è 2^m

X stringa di m caratteri $X = x_1 x_2 \dots x_m$

Y stringa di n caratteri $Y = y_1 y_2 \dots y_n$

X_i = i-esimo prefisso di X : stringa composta dai primi i caratteri di X

$X = SPIE\underbrace{GARE}$ $X_4 = SPIE$

Y_j = j-esimo prefisso di Y

TEOREMA (Sottostruttura ottima di una LCS)

Siano $X = x_1 x_2 \dots x_m$ e $Y = y_1 y_2 \dots y_n$ due stringhe.

Sia $Z = z_1 z_2 \dots z_k$ una LCS di X e Y. $Z = LCS(X, Y)$

1) $\boxed{x_m = y_n} \Rightarrow \underline{z_k = x_m}$ e $\underline{Z_{k-1} = LCS(X_{m-1}, Y_{n-1})}$

2) $\boxed{x_m \neq y_n} \Rightarrow z_k \neq x_m$ implica $Z_{k-1} = LCS(X_{m-1}, Y)$

3) $\boxed{x_m \neq y_n} \Rightarrow z_k \neq y_n$ implica $Z_{k-1} = LCS(X, Y_{n-1})$

Dimostrazione

1) ~~Ass~~ per assurdo $z_k \neq x_m \Rightarrow W = Zx_m$ W è SC di X e Y

$|W| = k+1$ $\lightning$ $Z = LCS(X, Y)$

$|Z| = k$

dobbiamo dimostrare $Z_{k-1} = LCS(X_{m-1}, Y_{n-1})$

$\rightarrow$ certamente Z_{k-1} è CS di X_{m-1} e Y_{n-1}

per assurdo Z_{k-1} non sia una LCS di X_{m-1} e Y_{n-1}

$|Z_{k-1}| = k-1$

$\Rightarrow \exists W, \quad |W| > k-1, \quad t.c. \quad W = LCS(X_{m-1}, Y_{n-1})$

$\Rightarrow \underbrace{Wx_m}_{Wx_m \text{ è CS a } X \text{ e } Y}, \quad |Wx_m| = |W| + 1 \underline{\underline{>}} (k-1)+1 = \underline{k}$

$\Downarrow$

$Z, |Z| = k,$
è LCS di X e Y

2) $x_m \neq y_n$, $\underbrace{z_k \neq x_m}$

$Z = z_1 z_2 \dots z_k$ è CS di X_{m-1} e Y

dobbiamo dimostrare che Z è anche LCS

supponiamo $\exists W$ t.c. $W = CS(X_{m-1}, Y)$, $|W| > |Z|$

$\Rightarrow W = CS(X_{m-1}, Y)$ è anche $CS(X, Y)$

$|W| > |Z| = k$

Z è $LCS(X, Y)$

3) analogo al 2.

↳ una LCS di due sequenze contiene al suo interno una LCS dei loro prefissi

⤳ SOTTOSTRUTTURA OTTIMA

$$LCS(X,Y) = \begin{cases} x_m = y_n & LCS(X,Y) = LCS(X_{m-1}, Y_{n-1})x_m \\ & |LCS(X,Y)| = |LCS(X_{m-1}, Y_{n-1})| + 1 \\ \cancel{x_m \neq y_n} & \\ x_m \neq y_n & |LCS(X,Y)| = \max\{|LCS(X_{m-1}, Y)|, |LCS(X, Y_{n-1})|\} \end{cases}$$

$LCS(X_{m-1}, Y_{n-1})$

PD

1. sottoproblemi

tabella c, $(m+1) \times (n+1)$

$c[i,j] =$ lunghezza $LCS(X_i, Y_j)$

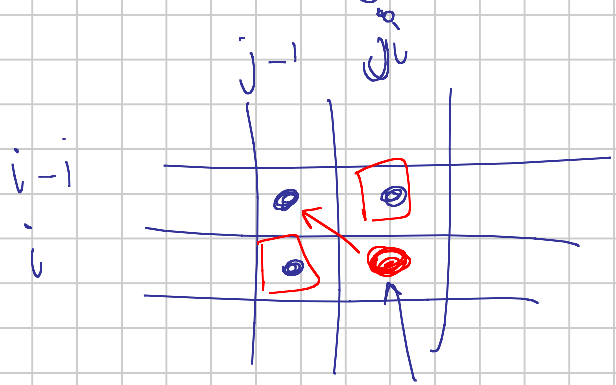
2. sottoproblemi elementari ("prefissi vuoti")

$c[i,0] = c[0,j] = 0$ $\quad 0 \le i \le m$, $0 \le j \le n$

$\forall i, \forall j$ $\quad LCS(X_0, Y_j) = LCS(X_i, Y_0) =$ stringa vuota

(3)

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



(4) return $c[m, n]$

Y = OSPITATA   X = SPIA

C = matrice 5x9

		Ø	O	S	P	I	T	A	T	A
		0	1	2	3	4	5	6	7	8
Ø	0	0	0	0	0	0	0	0	0	0
S	1	0	0	1	1	1	1	1	1	1
P	2	0	0	1	2	2	2	2	2	2
I	3	0	0	1	2	3	3	3	3	3
A	4	0	0	1	2	3	3	4	4	4

$m = 4$ $m+1$ righe indicizzate con i caratteri di X

$n = 8$ $n+1$ colonne, indicizzate con i caratteri di Y.

$|LCS(X,Y)| = 4$