

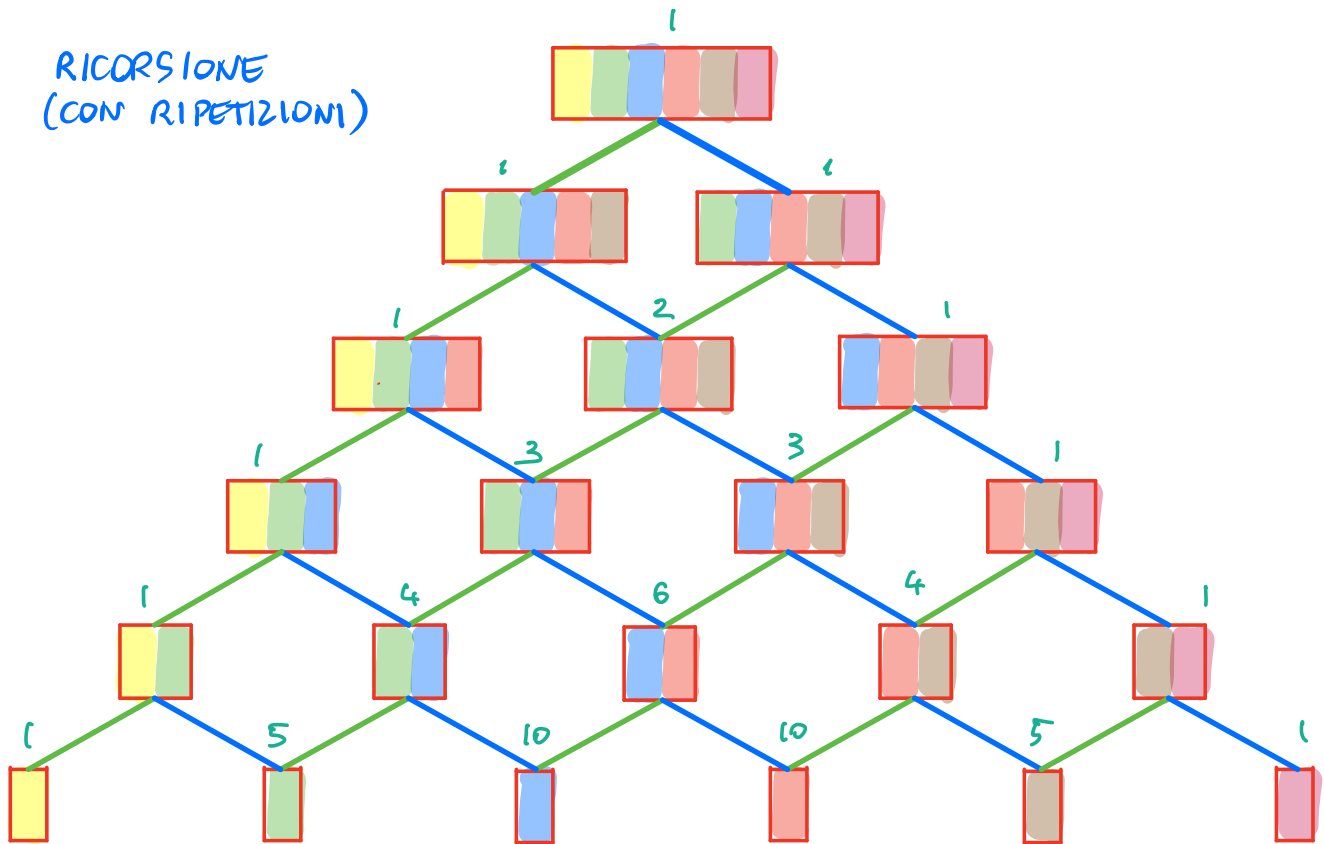
ELEMENTI DELLA PROGRAMMAZIONE DINAMICA

0 CARATTERIZZAZIONE DELLA STRUTTURA DI UNA SOLUZIONE OTTIMA (SOTTOSTRUTTURA OTTIMA)

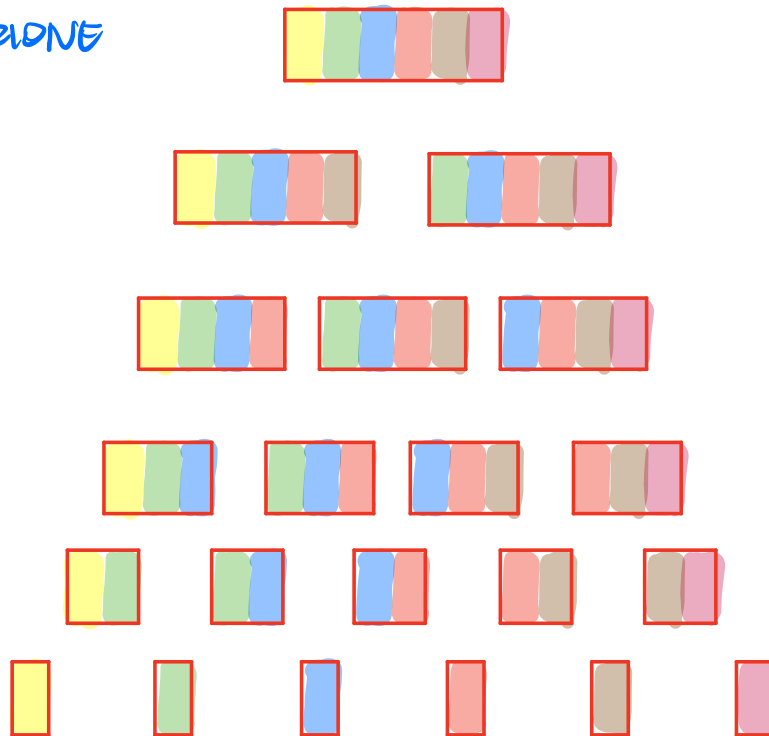
(SCHEMA TIPICO)

1. SI DIMOSTRA CHE UNA SOLUZIONE CONSISTE NEL FARE UNA SCELTA, CHE LASCIA UNO O PIÙ SOTTOPROBLEMI DA RISOLVERE
2. SI SUPPONE TEMPORANEAMENTE DI CONOSCERE TALE SCELTA
3. FATTA LA SCELTA, SI DETERMINANO I SOTTOPROBLEMI DA CONSIDERARE E LO SPAZIO PIÙ PICCOLO DI SOTTOPROBLEMI RISULTANTE
4. SI DIMOSTRA CHE LE SOLUZIONI DEI SOTTOPROBLEMI ALL'INTERNO DI UNA SOLUZIONE OTTIMA SONO OTTIME (TECNICA CUT-PASTE)

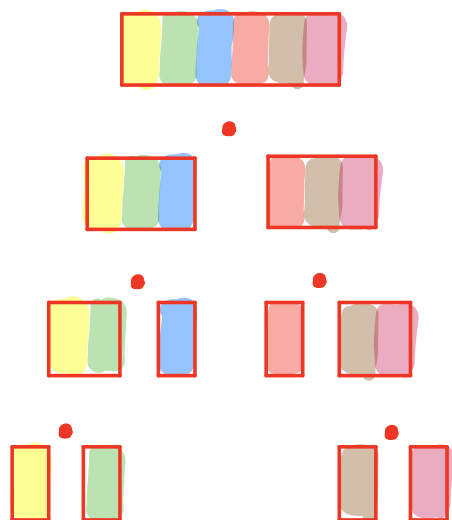
RICORSIONE
(CON RIPETIZIONI)



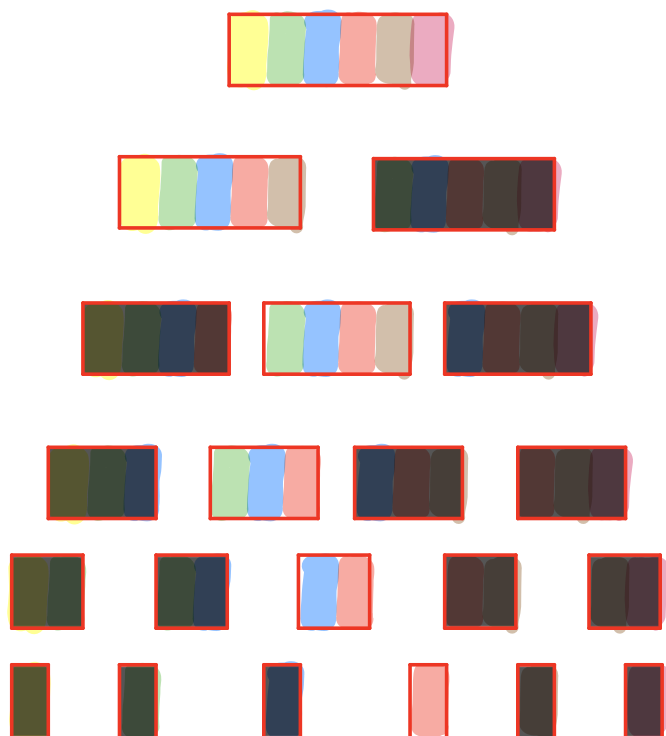
PROGRAMMAZIONE
DINAMICA



SOLUZIONE
(CON ORACOLO)



?



LA SOTTOSTRUTTURA OTTIMA VARIA IN FUNZIONE DEL TIPO DI PROBLEMA IN DUE MODI:

- NUMERO n_1 DI SOTTOPROBLEMI UTILIZZATI IN UNA SOLUZIONE OTTIMA

ES. CATENE DI MONTAGGIO $\longrightarrow n_1 = 1$

SEQUENZE DI MATRICI $\longrightarrow n_1 = 2$

- NUMERO n_2 DI SCELTE PER DETERMINARE QUALI SOTTOPROBLEMI UTILIZZARE

ES. CATENE DI MONTAGGIO $\longrightarrow n_2 = 2$

SEQUENZE DI MATRICI $\longrightarrow n_2 = l - 1$

(l E' LA LUNGHEZZA DELLA SEQUENZA)

GENERALMENTE, LA COMPLESSITA' DI UN ALGORITMO DI
PROGRAMMAZIONE DINAMICA DIPENDE DA DUE FATTORI:

- DIMENSIONE DELLO SPAZIO DEI SOTTOPROBLEMI
- NUMERO DI SCELTE DA CONSIDERARE PER OGNI SOTTOPROBLEMA

ES. * CATENE DI MONTAGGIO \rightarrow

$$\left. \begin{array}{l} \text{DIM(SPAZIO_SOTTOPROBLEMI)} = \textcircled{4}(m) \\ n_2 = 2 \end{array} \right\} \rightsquigarrow O(m)$$

* SEQUENZE DI MATRICI \rightarrow

$$\left. \begin{array}{l} \text{DIM(SPAZIO_SOTTOPROBLEMI)} = \textcircled{4}(m^2) \\ n_2 = O(m) \end{array} \right\} \rightsquigarrow O(m^3)$$

LA PROGRAMMAZIONE DINAMICA USA LA SOTTOSTRUTTURA OTTIMA SECONDO UNO SCHEMA **BOTTOM-UP**, CIOE'

- PRIMA : VENGONO TROVATE LE SOLUZIONI OTTIME DEI SOTTOPROBLEMI
- DOPO : VIENE TROVATA UNA SOLUZIONE OTTIMA DEL PROBLEMA.

DI SOLITO IL COSTO DELLA SOLUZIONE DEL PROBLEMA E' PARI AI COSTI PER RISOLVERE I SOTTOPROBLEMI
PIÙ UN COSTO DIRETTAMENTE IMPUTABILE ALLA SCELTA STESSA

BISOGNA STARE ATTENTI A NON ASSUMERE L'ESISTENZA DELLA SOTTOSTRUTTURA OTTIMA ANCHE QUANDO NON E' POSSIBILE FARLO!

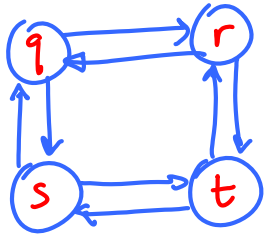
ESEMPIO SIA $G=(V,E)$ UN GRAFO ORIENTATO E
SIANO $u, v \in V$

PROBLEMA 1 TROVARE UN CAMMINO DA u A v IN G
FORNITO DAL MINOR NUMERO DI ARCHI
(TALE CAMMINO, OVVIAMENTE, SARA' SEMPLICE)

PROBLEMA 2 TROVARE UN CAMMINO SEMPLICE DA u A v
IN G FORNITO DAL MAGGIOR NUMERO DI ARCHI

- IL PROBLEMA 1 PRESENTA UNA SOTTOSTRUTTURA OTTIMA
- IL PROBLEMA 2 **NON** PRESENTA UNA SOTTOSTRUTTURA OTTIMA

INFATTI, SI CONSIDERI IL GRAFO



- * IL CAMMINO $q \rightarrow r \rightarrow t$ E' UN CAMMINO SEMPLICE MASSIMO DA q A t
- * MA $q \rightarrow r$ NON E' UN CAMMINO SEMPLICE MASSIMO DA q A r
 INFATTI $q \rightarrow s \rightarrow t \rightarrow r$ E' UN CAMMINO SEMPLICE DA q A r PIU' LUNGO DEL CAMMINO $q \rightarrow r$!

0 RIPETIZIONE DEI SOTTOPROBLEMI

- LO SPAZIO DEI SOTTOPROBLEMI DEVE ESSERE "PICCOLO", NEL SENSO CHE UN ALGORITMO RICORSIVO RISOLVE RIPETUTAMENTE GLI STESSI PROBLEMI
- GLI ALGORITMI DI PROGRAMMAZIONE DINAMICA SFRUTTANO I SOTTOPROBLEMI RIPETUTI RISOLVENDO CIASCUN SOTTOPROBLEMA UNA SOLA VOLTA, MEMORIZZANDO LA SOLUZIONE IN UNA TABELLA

ES. SOLUZIONE RICORSIVA AL PROBLEMA DELLA MOLTIPLICAZIONE DI UNA SEQUENZA DI MATRICI:

RECURSIVE_MATRIX_CHAIN(p, i, j)

if $i = j$ then
return 0

$m[i, j] := +\infty$

for $k := i$ to $j-1$ do

$q := \text{RECURSIVE_MATRIX_CHAIN}(p, i, k)$

$+ \text{RECURSIVE_MATRIX_CHAIN}(p, k+1, j) + p_{i-1} p_k p_j$

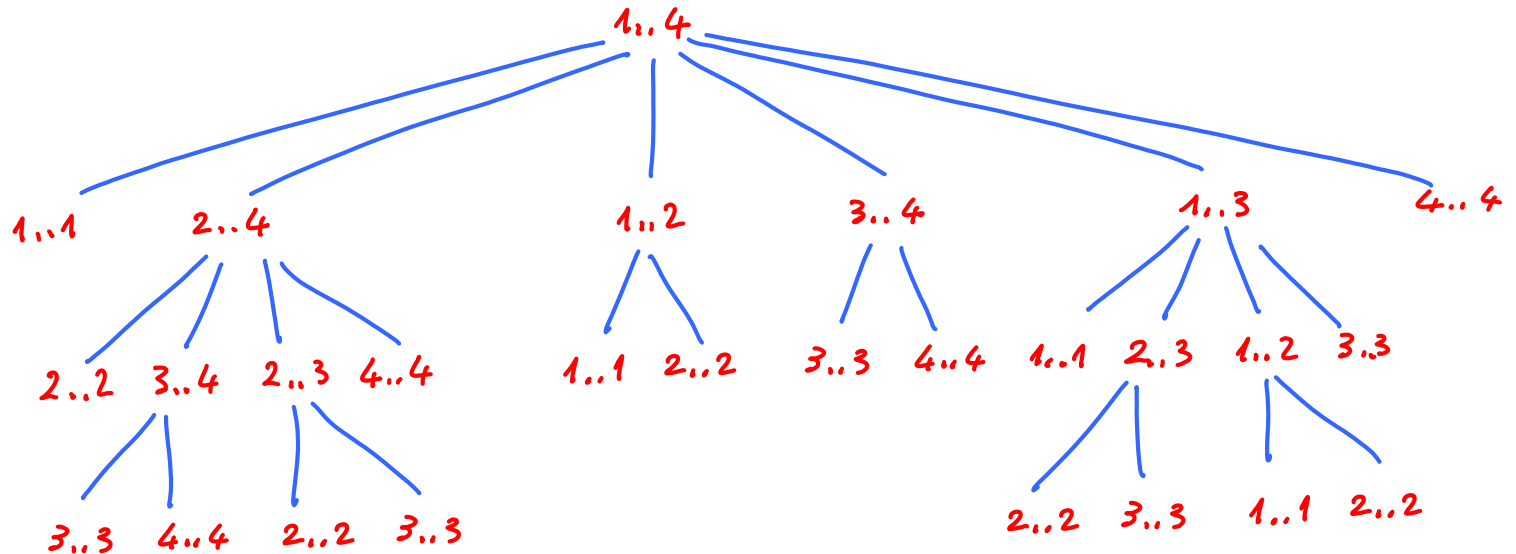
if $q < m[i, j]$ then

$m[i, j] := q$

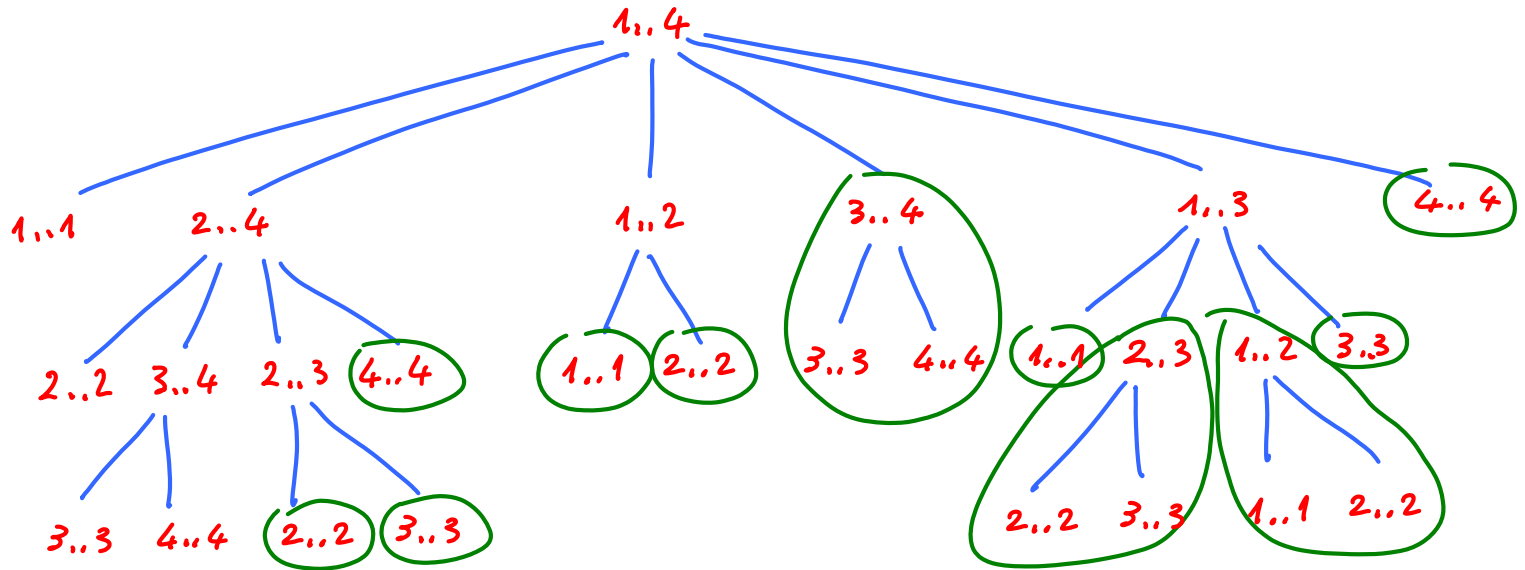
return $m[i, j]$

$$m[i, j] = \begin{cases} 0 & i=j \\ \min_{i \leq k < j} (m[i, k] + m[k+1, j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

ALBERO DI RICORSIONE DI `RECURSIVE_MATRIX_CHAIN(p, 1, 4)`



ALBERO DI RICORSIONE DI `RECURSIVE_MATRIX_CHAIN(p, 1, 4)`



SOTTOPROBLEMI RIPETUTI

COMPLESSITA' DI RECURSIVE_MATRIX_CHAIN

- SIA $T(n)$ IL TEMPO IMPIEGATO DA RECURSIVE_MATRIX_CHAIN SU UN'ISTANZA DI DIMENSIONE n
- VERIFICHIAMO CHE $T(n) = \Omega(2^n)$, SI HA:

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1), \text{ PER } n > 1$$

QUINDI:
$$T(n) \geq 2 \sum_{k=1}^{n-1} T(k) + n$$

DIMOSTRIAMO PER INDUZIONE CHE $T(n) \geq 2^{n-1}$.

- $T(1) \geq 1 = 2^0 = 2^{1-1}$

- $$\begin{aligned} T(n+1) &\geq 2 \sum_{k=1}^n T(k) + n+1 \geq 2 \sum_{k=0}^{n-1} 2^k + n+1 \\ &\geq 2(2^n - 1) + n+1 = 2^{n+1} - 2 + n+1 \\ &= 2^{n+1} + n - 1 \geq 2^{n+1} \end{aligned}$$

MEMOIZED-MATRIX-CHAIN (P)

$n := \text{length}[P] - 1$

for $i := 1$ to n do

for $j := i$ to n do

$m[i, j] := +\infty$

return LOOKUP-CHAIN (P, 1, n)

LOOKUP-CHAIN (P, i, j)

if $m[i, j] < +\infty$ then return $m[i, j]$

if $i = j$ then $m[i, j] := 0$

else for $k := i$ to $j - 1$ do

$q := \text{LOOKUP-CHAIN}(P, i, k)$

$+ \text{LOOKUP-CHAIN}(P, k + 1, j)$

$+ P_{i-1} P_k P_j$

if $q < m[i, j]$ then

$m[i, j] := q$

return $m[i, j]$