ELEMENTI DI PROGRAMMAZIONE DINAMICA

I CARATTERI CONTRADDISTINTIVI PERCHE' LA PROGRAMMAZIONE DINAMICA RISULTI APPLICABILE CON SUCCESSO SONO:

- SOTTOSTRUTTURA OTTIMA DELLE SOLUZIONI
- DIMENSIONE POLINOMIALE DELLO SPAZIO DEI SOTTOPROBLEMI

PASSI FONDAMENTALI PER LA RISOLUZIONE DI PROBLEMI DI PROGRAMMAZIONE DINAMICA

1. CARATTERIZZAZIONE DELLA STRUTTURA DI UNA SOLUZIONE OTTIMA
2. DEFINIZIONE RICORSIVA DEL VALORE DI UNA SOLUZIONE OTTIMA
3. CALCOLO BOTTOM-UP DEL VALORE DI UNA SOLUZIONE OTTIMA
4. CALCOLO DI UNA SOLUZIONE OTTIMA

- 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)

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 $\longrightarrow$ $n_2 = \ell - 1$

(ℓ E' LA LUNGHEZZA DELLA SEQUENZA)

GENERALMENTE, LA COMPLESSITÀ 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 $\longrightarrow$

$$\left.\begin{array}{l} DIM(SPAZIO\_SOTTOPROBLEMI) = \Theta(n) \\ n_2 = 2 \end{array}\right] \rightsquigarrow \Theta(n)$$

* SEQUENZE DI MATRICI $\longrightarrow$

$$\left.\begin{array}{l} DIM(SPAZIO\_SOTTOPROBLEMI) = \Theta(n^2) \\ n_2 = O(n) \end{array}\right] \rightsquigarrow O(n^3)$$

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

- PRIMA : VENGONO TROVATE LE SOLUZIONI OTTIME DEI SOTTO PROBLEMI

- 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 $u, v \in V$

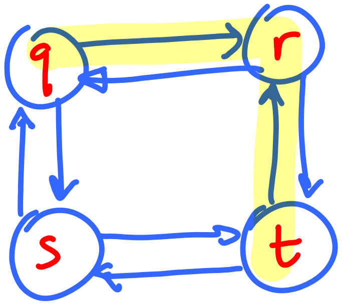
PROBLEMA 1 TROVARE UN CAMMINO (SEMPLICE) DA u A v IN G FORMATO DAL MINOR NUMERO DI ARCHI (TALE CAMMINO, OVVIAMENTE, SARA' SEMPLICE)

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

- IL <u>PROBLEMA 1</u> PRESENTA UNA SOTTOSTRUTTURA OTTIMA

- IL <u>PROBLEMA 2</u> NON PRESENTA UNA SOTTOSTRUTTURA OTTIMA

INFATTI, SI CONSIDERI IL GRAFO



* IL CAMMINO $q \to r \to t$ E' UN CAMMINO SEMPLICE MASSIMO DA q A t

* MA $q \to r$ NON E' UN CAMMINO SEMPLICE MASSIMO DA q A r

INFATTI $q \to s \to t \to r$ E' UN CAMMINO SEMPLICE DA q A r PIÙ LUNGO DEL CAMMINO $q \to r$!

* NEL CASO DEL PROBLEMA 2, NON SOLO NON VALE LA PROPRIETÀ DELLA SOTTOSTRUTTURA OTTIMA, MA ANCHE NON RISULTA POSSIBILE ASSEMBLARE UNA SOLUZIONE VALIDA DEL PROBLEMA DALLE SOLUZIONI DEI SOTTOPROBLEMI.

INFATTI, SE COMBINIAMO I CAMMINI SEMPLICI MASSIMI

$q \to s \to t \to r$ E $r \to q \to s \to t$

SI OTTIENE IL CAMMINO

$q \to s \to t \to r \to q \to s \to t$

CHE NON È SEMPLICE!

INFATTI, I SOTTOPROBLEMI PER TROVARE UN CAMMINO SEMPLICE MASSIMO NON SONO INDIPENDENTI

- RIPETIZIONE DEI SOTTOPROBLEMI

- LO SPAZIO DEI SOTTOPROBLEMI DEVE ESSERE "PICCOLO", COSICCHE' UN ALGORITMO RICORSIVO RISOLVERA' 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

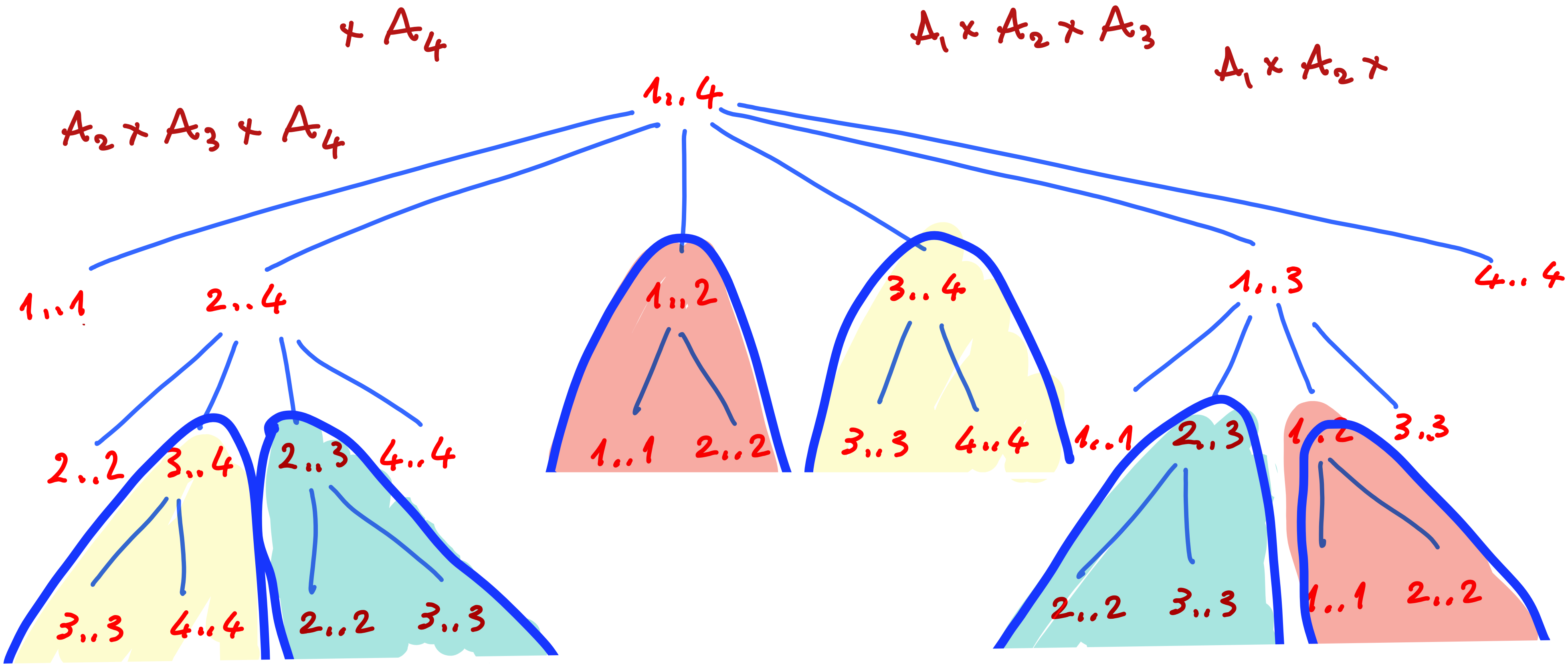
ES. SOLUZIONE RICORSIVA AL PROBLEMA DELLA MOLTIPLICAZIONE DI UNA SEQUENZA DI MATRICI:

- RECURSIVE_MATRIX_CHAIN (p, i, j)

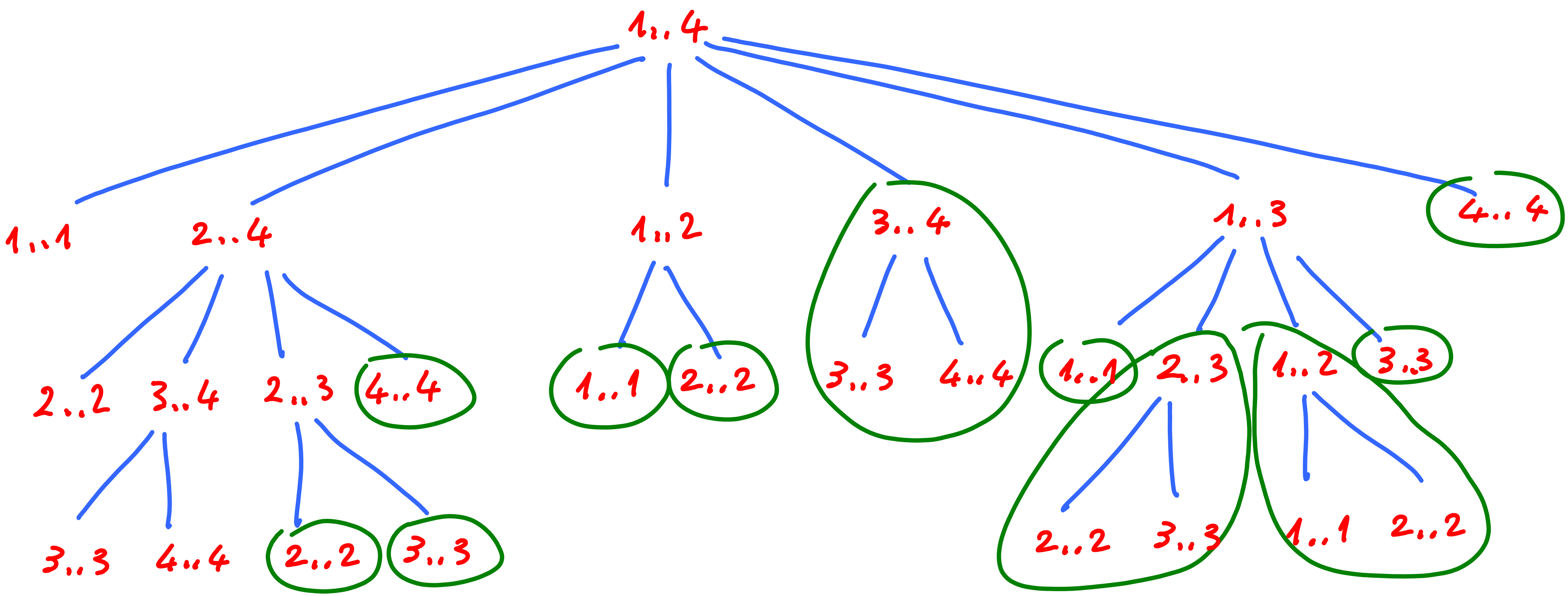
```
if i = j then
    return 0
m[i,j] := +∞
for k := i to j-1 do
    q := RECURSIVE_MATRIX_CHAIN (p, i, k)
         + RECURSIVE_MATRIX_CHAIN (p, k+1, j) + p_{i-1} p_k p_j
    if q < m[i,j] then
        m[i,j] := q
return m[i,j]
```

$$m[i,j] = \begin{cases} 0 & i=j \\ \min\limits_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & i<j \end{cases}$$

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



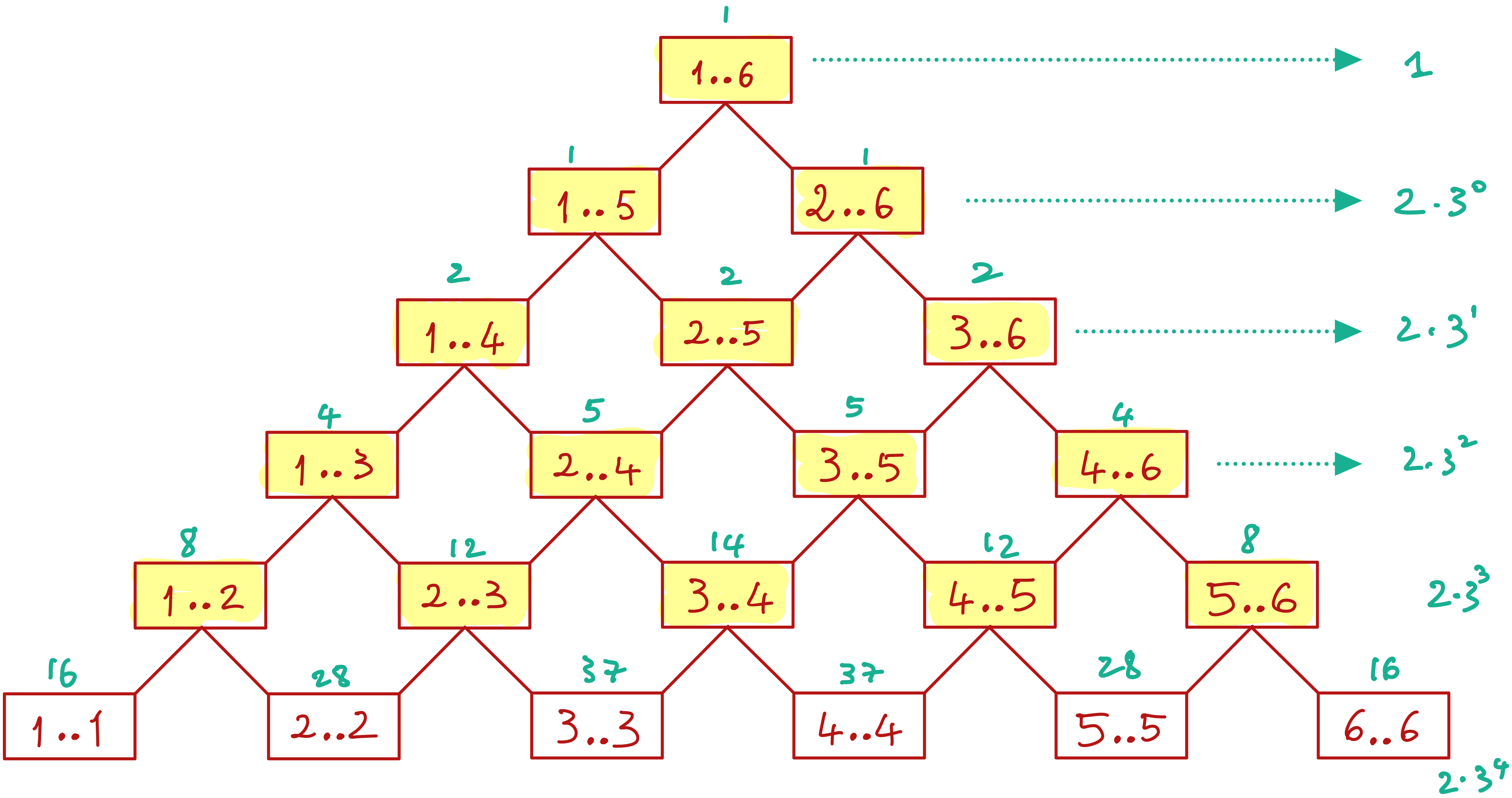
ALBERO DI RICORSIONE DI RECURSIVE_MATRIX_CHAIN (p, 1, 4)



SOTTOPROBLEMI RIPETUTI

COMPLESSITA' DI RECURSIVE_MATRIX_CHAIN

(GRAFO DELLE CHIAMATE)



PER UNA SEQUENZA DI n MATRICI, VIENE EFFETTUATO IL SEGUENTE NUMERO DI CHIAMATE A RECURSIVE_MATRIX_CHAIN:

$$1 + 2 \cdot 3^0 + 2 \cdot 3^1 + 2 \cdot 3^2 + \dots + 2 \cdot 3^{n-2}$$

$$= 1 + 2 \cdot \sum_{i=0}^{n-2} 3^i$$

$$= 1 + 2 \cdot \frac{3^{n-1} - 1}{2}$$

$$= 3^{n-1}.$$

PERTANTO RECURSIVE_MATRIX_CHAIN HA COMPLESSITÀ $\Omega(3^n)$.

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:

$$T(1) \geq 1$$

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

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

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

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

$$\geq 2\left(2^n - 1\right) + n + 1 = 2^{n+1} - 2 + n + 1$$

$$= 2^{n+1} + n - 1 \geq 2^n$$

[redacted]

COMPLESSITA' DI MATRIX_CHAIN_ORDER

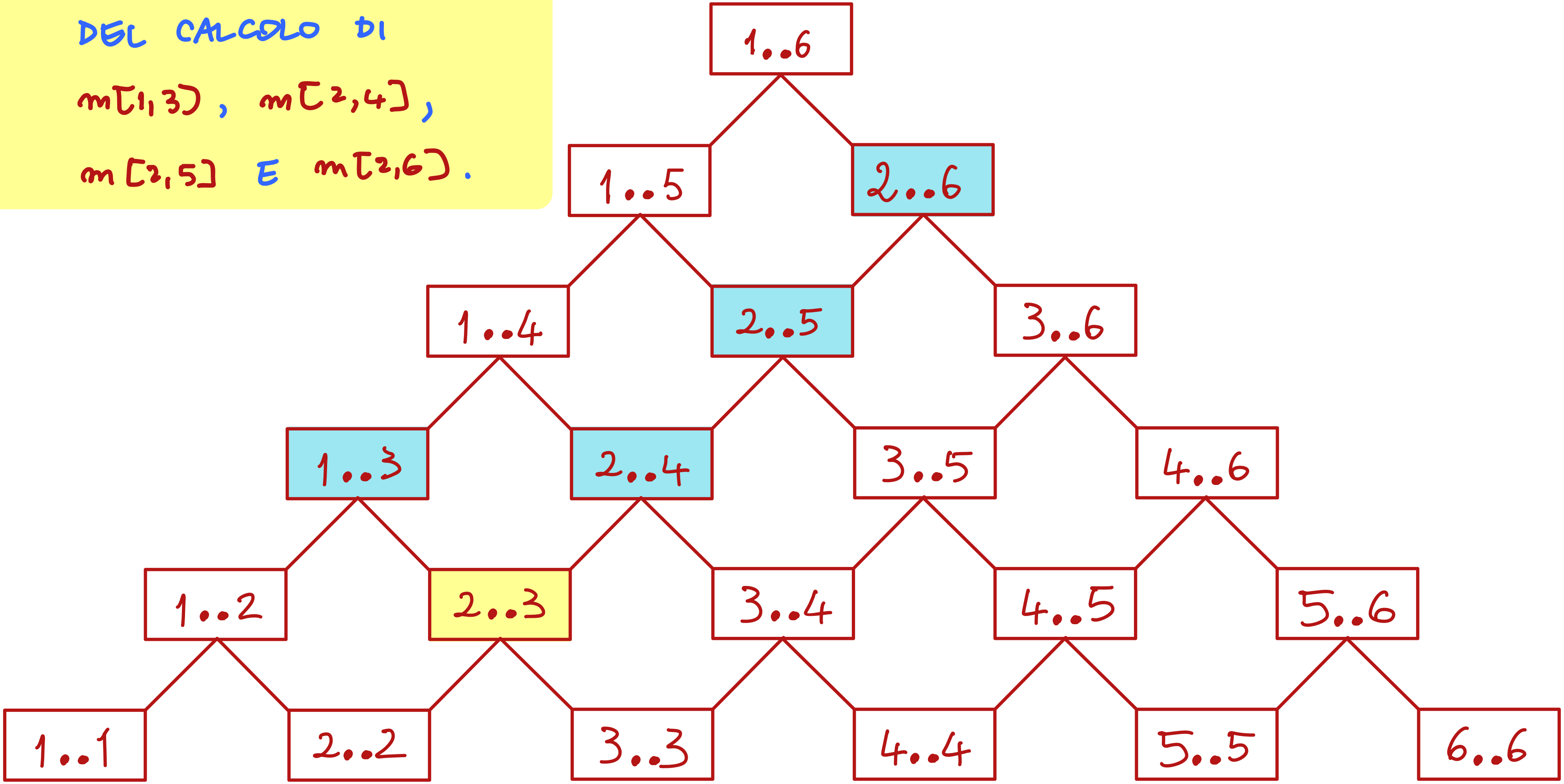
```
MATRIX_CHAIN_ORDER(p)
for i := 1 to n do
    m[i,i] := 0
for Δ := 1 to n-1 do
    for i := 1 to n-Δ do
        j := Δ+i
        m[i,j] := +∞
        for k := i to j-1 do
            q := m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
            if q < m[i,j] then
                m[i,j] := q
                s[i,j] := k
return m, s
```

ACCESSI CONTEGGIATI
A VALORI GIA' CONSOLIDATI
DELLA MATRICE m

GRAFO DEGLI ACCESSI ALLA MATRICE m

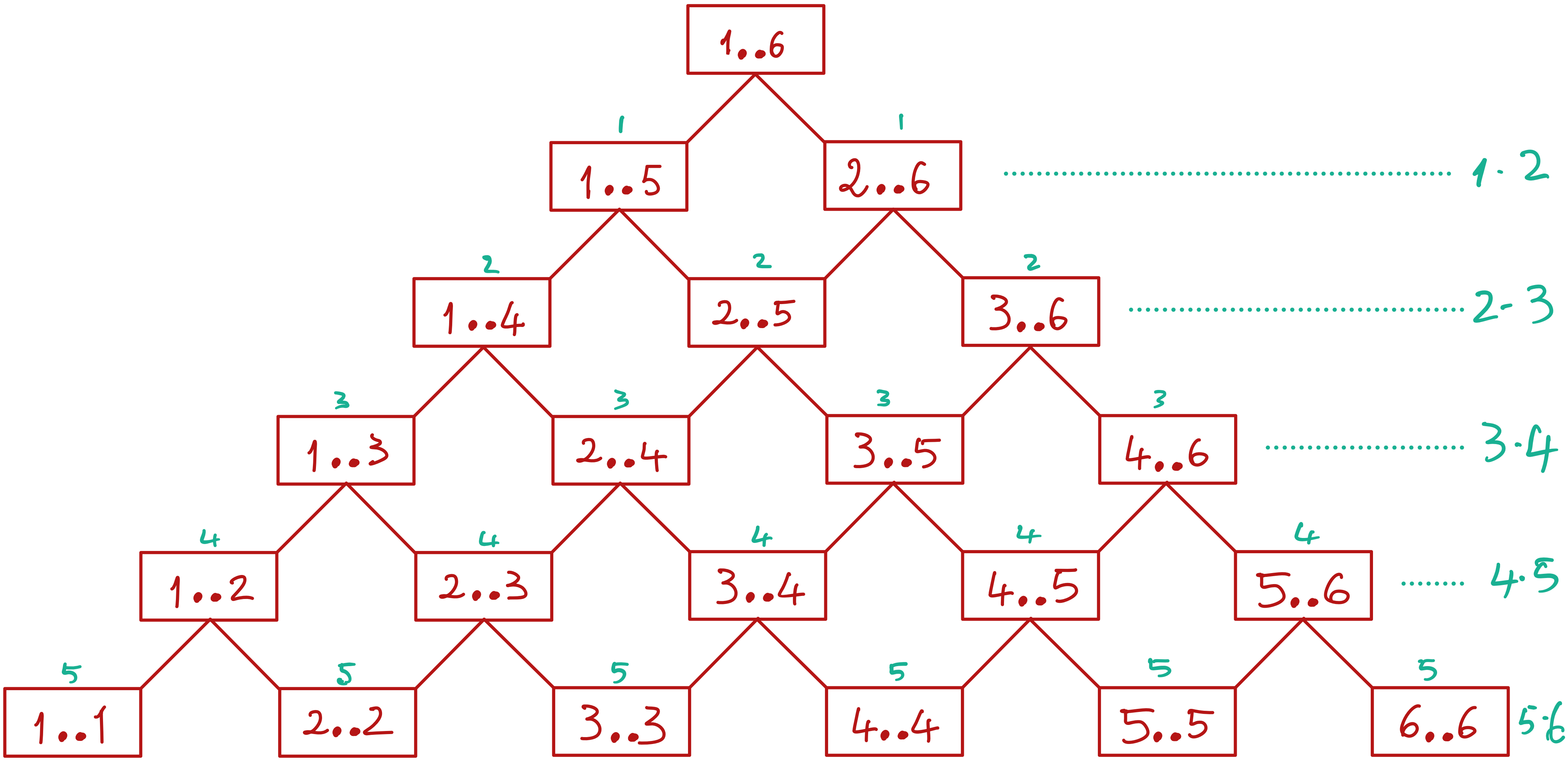
ES.: CI SONO 4 ACCESSI A $m[2,3]$ NEL CORSO DEL CALCOLO DI $m[1,3]$, $m[2,4]$, $m[2,5]$ E $m[2,6]$.

$1..n \longmapsto (n-1).$



GRAFO DEGLI ACCESSI ALLA MATRICE m

$$m[i,j] = \begin{cases} 0 & i=j \\ \min\limits_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & i<j \end{cases}$$



- PER UNA SEQUENZA DI n MATRICI, VIENE EFFETTUATO IL SEGUENTE NUMERO DI ACCESSI ALLA MATRICE m DURANTE L'ESECUZIONE DI MATRIX_CHAIN_ORDER

$$0\cdot 1 + 1\cdot 2 + 2\cdot 3 + 3\cdot 4 + \cdots + (n-1)n$$

$$= 1(1-1) + 2(2-1) + 3(3-1) + 4(4-1) + \cdots + n(n-1)$$

$$= (1^2-1) + (2^2-2) + (3^2-3) + (4^2-4) + \cdots + (n^2-n)$$

$$= \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} i$$

$$= \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2}$$

$$= \frac{n(n+1)(2n+1-3)}{6}$$

$$= \frac{n(n+1)(2n-2)}{6}$$

$$= \frac{n(n+1)(n-1)}{3}$$

$$= \frac{n(n^2-1)}{3}$$

$$= \frac{n^3-n}{3} = \Theta(n^3)$$

PERTANTO MATRIX_CHAIN_ORDER HA COMPLESSITÀ $\Omega(n^3)$.

INSIEME ALLA COMPLESSITÀ $O(n^3)$ GIÀ OTTENUTA PER ISPEZIONE DELLA PROCEDURA, POSSIAMO QUINDI AFFERMARE CHE MATRIX_CHAIN_ORDER HA COMPLESSITÀ $\Theta(n^3)$.

Da Wikipedia

La memoizzazione è una tecnica di programmazione che consiste nel salvare in memoria i valori restituiti da una funzione in modo da averli a disposizione per un riutilizzo successivo senza doverli ricalcolare.

Memoizzare significa letteralmente "mettere in memoria".

Una funzione può essere "memoizzata" soltanto se soddisfa la trasparenza referenziale, cioè se non ha effetti collaterali e restituisce sempre lo stesso valore quando riceve in input gli stessi parametri.

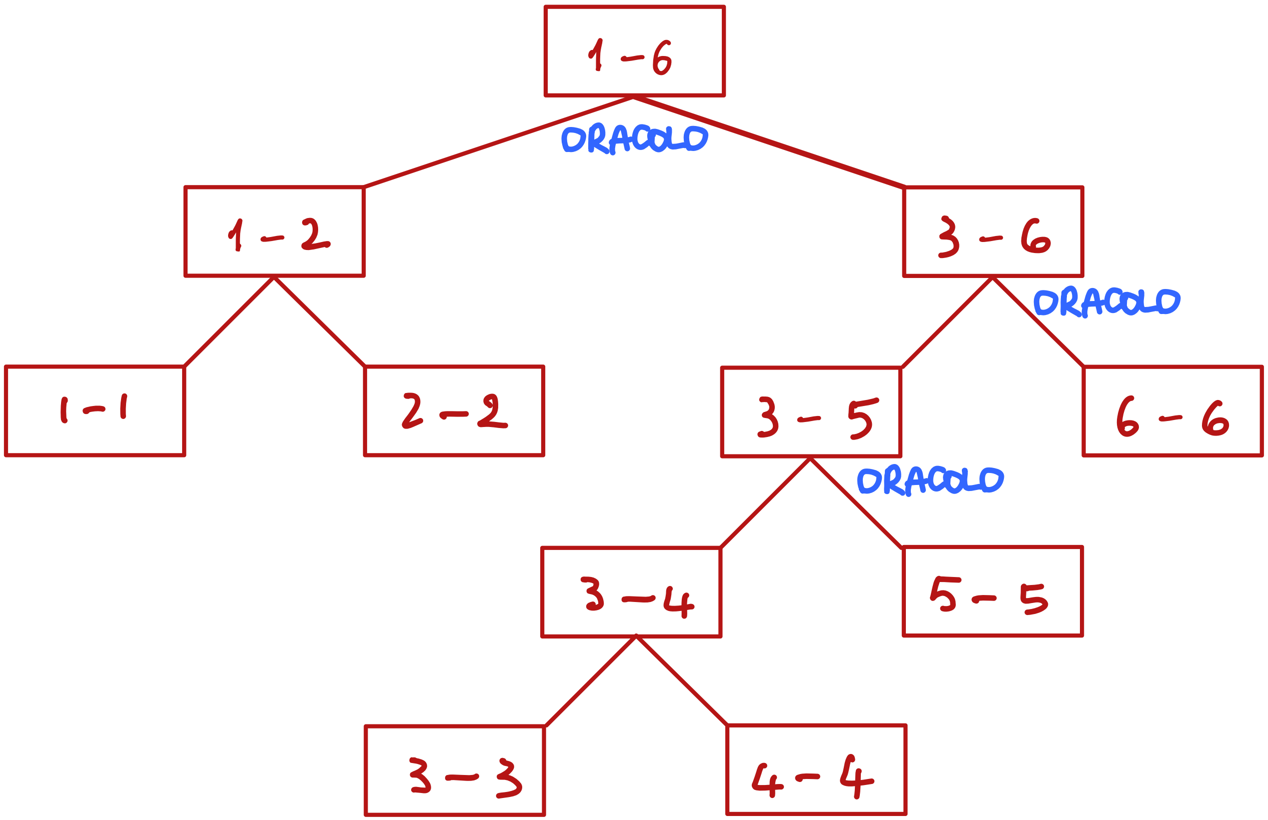
SOLUZIONE RICORSIVA CON MEMOIZZAZIONE

```
MEMOIZED_MATRIX_CHAIN (p)
   n := length[p] - 1
   for i := 1 to n do
       for j := 1 to n do
           m[i,j] := +∞
   return LOOKUP_CHAIN (p, 1, n)
```

```
LOOKUP_CHAIN (p, i, j)
if m[i,j] < +∞ then return m[i,j]
if i = j then m[i,j] := 0
   else for k := i to j-1 do
              q := LOOKUP_CHAIN (p, i, k)
                   + 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]
```

IN ANALOGIA CON MATRIX_CHAIN_ORDER, ANCHE LA PROCEDURA MEMOIZED_MATRIX_CHAIN HA COMPLESSITÀ $\Theta(n^3)$.

SOLUZIONE "IPOTETICA" CON ORACOLO



COMPESSITA' : $O(n)$ + COMPLESSITA' ORACOLO

- ORACOLO "GREEDY"