Algoritmi e Strutture Dati

Introduzione alla Programmazione Dinamica

Maria Rita Di Berardini, Emanuela Merelli[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

Tecniche di Programmazione

Tecniche di progettazione e analisi di algoritmi efficienti:

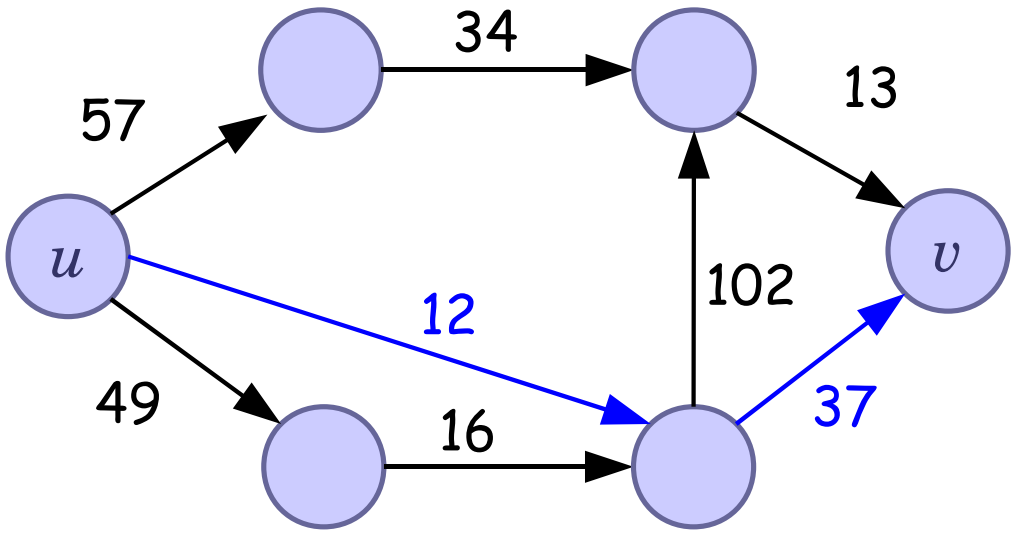
1. Divide-et-Impera
2. Programmazione Dinamica (PD, for short)
3. Algoritmi golosi

Le ultime due tecniche di progettazione di algoritmi sono più sofisticate del Divide-et-Impera, ma consento di risolve in maniera efficiente molti problemi computazionali

In particolare, vedremo come queste tecniche di programmazione vengono usate per risolvere **problemi di ottimizzazione**

Problemi di Ottimizzazione: un esempio

I nodi del seguente grafo rappresentano delle città, gli archi collegamenti tra coppie di città; ogni arco $\langle u, v \rangle$ ha associato un peso che rappresenta la distanza tra u e v



Problema: determinare il cammino minimo tra due nodi dati u e v

Problemi di Ottimizzazione

Risolvere un problema di ottimizzazione significa determinare, nello spazio delle soluzioni ammissibili, la soluzione con valore ottimo (massimo o minimo)

Una soluzione ottima viene costruita effettuando una serie di scelte: ogni scelta determina uno o più sottoproblemi da risolvere

La programmazione dinamica, come il metodo Divide-et-Impera, risolve un problema combinando soluzioni di sottoproblemi, ma può essere applicata anche quando i sottoproblemi non sono indipendenti e, quindi, uno stesso sottoproblema può comparire più volte

Il concetto chiave è quello di memorizzare le soluzioni dei sottoproblemi per usarle nel caso in cui uno di essi si dovesse ripresentare

Il processo di sviluppo

Il processo di sviluppo di un algoritmo di PD può essere suddiviso in quattro fasi

1. caratterizzazione della struttura di una soluzione ottima (proprietà della sottostruttura ottima)
2. definizione di un metodo ricorsivo per il calcolo del valore di una soluzione ottima
3. calcolo del valore di una soluzione ottima secondo uno schema bottom-up (dal basso verso l'alto)
4. costruzione della soluzione ottima dalle informazioni raccolte nella frase precedente

La fase 4 può essere omessa se viene richiesto solo il valore di una soluzione ottima e non la sua effettiva costruzione

Parte I

Moltiplicazione di una sequenza di matrici

Il problema

Data una sequenza di n matrici $\langle A_1, A_2, A_n \rangle$ vogliamo calcolare il prodotto

$$A_1 A_2 \cdots A_n$$

Per risolvere questo problema dobbiamo:

- definire una opportuna "parentesizzazione", ossia porre le opportune parentesi per eliminare ogni possibile ambiguitá sul modo (ordine) con cui moltiplicare le matrici
- usare come subroutine l'algoritmo standard per il prodotto di due matrici

MATRIXMULTIPLY(A, B)

```
1  if columns[A] ≠ rows[B]
2     then error: "dimensioni non compatibili"
3     else for i ← 1 to rows[A]
4             do for j ← 1 to columns[B]
5                   do C[i, j] ← 0
6                      for k ← 1 to columns[A]
7                      do C[i, j] ← C[i, j] + (A[i, k] · B[k, j])
8             return C
```

A e B devono essere compatibili: $columns[A] = rows[B]$. Se A è una matrice $p \times q$ e B è una matrice $q \times r$, allora $A \cdot B$ è una matrice $p \times r$. Calcolare questo prodotto richiede pqr prodotti scalari

Il problema

Un prodotto di matrici si dice completamente parentesizzato se consiste di singole matrici o di prodotti fra coppie di matrici tutti racchiusi da parentesi

Ex: se la sequenza è $\langle A_1, A_2, A_3, A_4 \rangle$, il prodotto $A_1 A_2 A_3 A_4$ può essere completamente parentesizzato in cinque modi distinti:

$A_1 \cdot (A_2 \cdot (A_3 \cdot A_4))$	le parentesizzazioni sono tutte funzionalmente
$A_1 \cdot ((A_2 \cdot A_3) \cdot A_4)$	equivalenti, ossia, restituiscono lo stesso risultato
$(A_1 \cdot A_2) \cdot (A_3 A_4)$	poichè la moltiplicazione fra matrici è associativa
$(A_1 \cdot (A_2 \cdot A_3)) \cdot A_4$	tuttavia la parentesizzazione influisce sull'efficienza
$((A_1 A_2) \cdot A_3) \cdot A_4$	dell'algoritmo

Parentesizzazione ed efficienza dell'algoritmo

Consideriamo la sequenza di tre matrici $\langle A_1, A_2, A_3 \rangle$ le cui dimensioni sono, rispettivamente, 10×100, 100×5 e 5×50

Se moltiplichiamo secondo lo schema di parentesizzazione $((A_1 A_2) A_3)$

- eseguiamo $10 \cdot 100 \cdot 5 = 5000$ prodotti scalari per il calcolo di $A_1 A_2$, ottenendo una matrice di di dimensione 10×5
- $10 \cdot 5 \cdot 50 = 2500$ prodotti scalari per il calcolo di $(A_1 A_2) A_3$
- In totale $5000 + 2500 = 7500$ prodotti scalari

Se moltiplichiamo secondo lo schema di parentesizzazione $(A_1 (A_2 A_3))$

- eseguiamo $100 \cdot 5 \cdot 50 = 25000$ prodotti scalari per il calcolo di $A_2 A_3$, ottenendo una matrice di dimensione 100×50
- $10 \cdot 100 \cdot 50 = 50000$ prodotti scalari per il calcolo di $A_1 (A_2 A_3)$
- In totale $25000 + 50000 = 75000$ prodotti scalari

Problema della moltiplicazione della sequenza di matrici

Data una sequenza di n matrici

$$\langle A_1, A_2, \cdots, A_n \rangle$$

determinare uno schema di parentesizzazione completa di $A_1 A_2 \cdots A_n$ che minimizza il numero di prodotti scalari

L'obiettivo è quello di determinare l'ordine di moltiplicazione delle matrici che minimizza il numero di prodotti scalari

Dimensioni delle matrici: rappresentate da un vettore $p = \langle p_0, p_1, \cdots, p_n \rangle$ dove,per ogni $i = 1, 2, ..., n$, la matrice A_i ha dimensione $p_{i-1} \times p_i$

Una prima soluzione consiste nell'analizzare tutti i possibili schemi di parentesizzazione (tecnica "bruta") e scegliere quello ottimo

Numero di possibili parentesizzazione

Sia $P(n)$ il numero di parentesizzazioni complete distinte di una sequenza di n matrici

- Se $n = 1$, banalmente, $P(n) = 1$
- Se $n \geq 2$ un prodotto completamente parentesizzato è il prodotto di due sottoprodotti completamente parentesizzati e la suddivisione può avvenire tra la k-esima e $k+1$-esima matrice per un qualsiasi $k = 1, 2, \cdots, n-1$

$$P(n) = \begin{cases} 1 & \text{se } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{se } n \geq 2 \end{cases}$$

Poichè $P(n) = \Omega(2^n)$, la tecnica bruta non è inadeguata

Proviamo ad usare il Divide-et-Impera

Data una sequenza di n matrici $A_1, A_2, ..., A_n$ determinare uno schema di parentesizzazione che minimizzi il numero di prodotti scalari

- Possiamo originare due sottoproblemi distinti $A_1 \cdots A_k \mid A_{k+1} \cdots A_n$ (abbiamo scelto $1 \leq k < n$)

Proviamo ad usare il Divide-et-Impera

Data una sequenza di n matrici $A_1, A_2, ..., A_n$ determinare uno schema di parentesizzazione che minimizzi il numero di prodotti scalari

- Possiamo originare due sottoproblemi distinti $A_1 \cdots A_k \mid A_{k+1} \cdots A_n$ (abbiamo scelto $1 \leq k < n$)
- determinare uno schema di parentesizzazione ottimo per il prodotto $A_1 A_2 \cdots A_k$,
- determinare uno schema di parentesizzazione ottimo per il prodotto $A_{k+1} A_{k+2} \cdots A_n$,

Proviamo ad usare il Divide-et-Impera

Data una sequenza di n matrici $A_1, A_2, ..., A_n$ determinare uno schema di parentesizzazione che minimizzi il numero di prodotti scalari

- Possiamo originare due sottoproblemi distinti $A_1 \cdots A_k \mid A_{k+1} \cdots A_n$ (abbiamo scelto $1 \leq k < n$)
- determinare uno schema di parentesizzazione ottimo per il prodotto $A_1 A_2 \cdots A_k$,
- determinare uno schema di parentesizzazione ottimo per il prodotto $A_{k+1} A_{k+2} \cdots A_n$,
- la combinazione di queste due soluzione fornisce banalmente uno schema di parentesizzazione per $A_1 A_2 \cdots A_n$, ma chi ci garantisce che tale schema sia minimo??

Proviamo ad usare il Divide-et-Impera

Data una sequenza di n matrici $A_1, A_2, ..., A_n$ determinare uno schema di parentesizzazione che minimizzi il numero di prodotti scalari

- Possiamo originare due sottoproblemi distinti $A_1 \cdots A_k \mid A_{k+1} \cdots A_n$ (abbiamo scelto $1 \leq k < n$)
- determinare uno schema di parentesizzazione ottimo per il prodotto $A_1 A_2 \cdots A_k$,
- determinare uno schema di parentesizzazione ottimo per il prodotto $A_{k+1} A_{k+2} \cdots A_n$,
- la combinazione di queste due soluzione fornisce banalmente uno schema di parentesizzazione per $A_1 A_2 \cdots A_n$, ma chi ci garantisce che tale schema sia minimo??
- **chi ci garantisce che combinando soluzioni ottime dei sottoproblemi $A_1 A_2 \cdots A_k$ e $A_{k+1} A_{k+2} \cdots A_n$, otteniamo uno soluzione ottima per $A_1 A_2 \cdots A_n$??**

Algoritmi di Programmazione Dinamica

La Programmazione dinamica è una variante del Divide-et-Impera pensata per risolvere problemi di ottimizzazione

Può applicata con successo solo se il problema (di ottimizzazione) in esame ci consente di ottenere una soluzione ottima combinando soluzioni ottime di sottoproblemi

Proprietà della sottostruttura ottima: ogni soluzione ottima di un (sotto)problema deve contenere al suo interno soluzioni ottime di sottoproblemi (è indispensabile per poter combinare soluzioni ottime di sottoproblemi)

Fase 1: struttura della parentesizzazione ottima

Generico sottoproblema: determinare uno schema di parentesizzazione ottimo per il prodotto $A_i \cdots A_j$ dove $1 \leq i \leq j \leq n$

- Sottoproblemi banali ($i = j$), il prodotto $A_i \cdots A_j$ contiene una sola matrice
- Sottoproblemi banali ($i < j$) il prodotto $A_i \cdots A_j$ contiene almeno due matrici

Notazione: di qui in avanti scriveremo $A_{i,j}$ per indicare la matrice di dimensioni $p_{i-1} \times p_j$ che otteniamo calcolando il prodotto $A_i \cdots A_j$

Dimostriamo che ogni sottoproblema non banale verifica la proprietà della sottostruttura ottima

Fase 1: struttura della parentesizzazione ottima

Una qualsiasi parentesizzazione di $A_i \cdots A_j$ suddivide il prodotto fra le matrici A_k e A_{k+1} per qualche $i \leq k < j$

(corrisponde a calcolare prima $A_{i,k}$, poi $A_{k+1,j}$ ed, infine, moltiplicare queste due matrici per ottenere il prodotto finale $A_{i,j}$)

Se tale parentesizzazione è **ottima**, deve contenere parentesizzazioni **ottime** per i sottoproblemi $A_i \cdots A_k$ e $A_{k+1} \cdots A_n$

Lo dimostriamo per **assurdo**, utilizzando una tecnica che prende il nome di "**taglia e incolla**"

Fase 1: struttura della parentesizzazione ottima

Assumiamo, per assurdo, che la parentesizzazione p per $A_i \cdots A_k$ all'interno della parentesizzazione ottima per $A_i \cdots A_j$ non sia ottima

Allora, esiste un'altra parentesizzazione p' di $A_i \cdots A_k$ il cui costo costo (i.e. numero di prodotti scalari) minore

Sostituendo p con p' all'interno della parentesizzazione ottima di $A_i \cdots A_j$ (tagliamo la vecchia ed incolliamo la nuova, di qui il nome "taglia e incolla" per questa tecnica) otteniamo una parentesizzazione di $A_i \cdots A_j$ con costo minore della soluzione ottima. Impossibile

Questo prova che la parentesizzazione di $A_i \cdots A_k$ all'interno della parentesizzazione ottima di $A_i \cdots A_j$ è ottima

Fase 1: struttura della parentesizzazione ottima

In maniera analoga, possiamo dimostrare che la parentesizzazione del prodotto $A_{k+1} \cdots A_j$ all'interno della parentesizzazione ottima di $A_i \cdots A_j$ è ottima

In altri termini, la soluzione ottima del problema contiene al suo interno soluzioni ottime di sottoproblemi (il problema ha una sottostruttura ottima)

Possiamo usare la proprietà della sottostruttura ottima per costruire la soluzione ottima a partire dalle soluzioni ottime dei sottoproblemi

Quanti sottoproblemi dobbiamo risolvere?

Una volta capito che è possibile fornire una soluzione ottima in termini di soluzioni di sottoproblemi, è importante capire quanti sottoproblemi dobbiamo risolvere

Generico sottoproblema: determinare il costo della parentesizzazione ottima di $A_i \cdots A_j$ per $1 \leq i \leq j \leq n$

Quanti sono? Fissato $i \in [1, n]$, j varia nell'intervallo $[i, n]$ (e può essere scelto in $n - i + 1$ modi diversi). Il numero totale dei sottoproblemi è

$$\sum_{i=1}^{n} n - i + 1 = \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = \Theta(n^2)$$

Un numero ragionevole

Fase 2: una soluzione ricorsiva

Vediamo ora come definire il **costo** della soluzione ottima a partire dal costo delle soluzioni ottime dei sottoproblemi (il costo verrà inizialmente espresso da una funzione ricorsiva)

Fase 2: una soluzione ricorsiva

Vediamo ora come definire il **costo** della soluzione ottima a partire dal costo delle soluzioni ottime dei sottoproblemi (il costo verrà inizialmente espresso da una funzione ricorsiva)

Sia $m(i, j)$ il minimo numero di prodotti scalari per calcolare $A_{i,j}$.

Fase 2: una soluzione ricorsiva

Vediamo ora come definire il **costo** della soluzione ottima a partire dal costo delle soluzioni ottime dei sottoproblemi (il costo verrà inizialmente espresso da una funzione ricorsiva)

Sia $m(i,j)$ il minimo numero di prodotti scalari per calcolare $A_{i,j}$.

- Se $i = j$ (il prodotto $A_i \cdots A_j$ contiene una sola matrice) il problema è banale e $m(i,i) = 0$ per ogni $i = 1, \cdots, n$

Fase 2: una soluzione ricorsiva

Vediamo ora come definire il **costo** della soluzione ottima a partire dal costo delle soluzioni ottime dei sottoproblemi (il costo verrà inizialmente espresso da una funzione ricorsiva)

Sia $m(i,j)$ il minimo numero di prodotti scalari per calcolare $A_{i,j}$.

- Se $i = j$ (il prodotto $A_i \cdots A_j$ contiene una sola matrice) il problema è banale e $m(i,i) = 0$ per ogni $i = 1, \cdots, n$
- Se $i < j$ (problema non banale) sfruttiamo la sottostruttura ottima. **Assumiamo che la parentesizzazione ottima suddivida** $A_i \cdots A_j$ **tra** A_k **e** A_{k+1} per un qualche $i \leq k < j$. Allora:

Fase 2: una soluzione ricorsiva

Vediamo ora come definire il **costo** della soluzione ottima a partire dal costo delle soluzioni ottime dei sottoproblemi (il costo verrà inizialmente espresso da una funzione ricorsiva)

Sia $m(i,j)$ il minimo numero di prodotti scalari per calcolare $A_{i,j}$.

- Se $i = j$ (il prodotto $A_i \cdots A_j$ contiene una sola matrice) il problema è banale e $m(i,i) = 0$ per ogni $i = 1, \cdots, n$
- Se $i < j$ (problema non banale) sfruttiamo la sottostruttura ottima. **Assumiamo che la parentesizzazione ottima suddivida** $A_i \cdots A_j$ **tra** A_k **e** A_{k+1} per un qualche $i \leq k < j$. Allora:

$$m(i,j) = m(i,k) + m(k+1,j) + p_{i-1} p_k p_j$$

dove $m(i,k)$ e $m(k+1,j)$ sono il numero minimo di prodotti scalari per calcolare $A_{i,k}$ e $A_{k+1,j}$ (otteniamo delle matrici $p_{i-1} \times p_k$ e $p_k \times p_j$, risp.); $p_{i-1} p_k p_j$ è il numero di prodotti scalari necessari per calcolare $A_{i,k} \cdot A_{k+1,j}$

Fase 2: una soluzione ricorsiva

Chi ci dice quale è il valore di k che corrisponde alla parentesizzazione ottima? (Ricordo che $i \leq k < j$)

Fase 2: una soluzione ricorsiva

Chi ci dice quale è il valore di k che corrisponde alla parentesizzazione ottima? (Ricordo che $i \leq k < j$)

Calcoliamo tutti i valori della forma

$$m(i,k) + m(k+1,j) + p_{i-1}p_k p_j$$

al variare di k in $[i,j)$ e prendiamo il **minimo**.

Ricapitolando, se $i < j$:

$$m(i,j) = \min_{i \leq k < j} \left\{ m(i,k) + m(k+1,j) + p_{i-1}p_k p_j \right\}$$

Fase 2: una soluzione ricorsiva

La definizione ricorsiva di $m(i,j)$ diventa:

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

Calcolo dei costi ottimi

Potremmo scrivere un'algoritmo ricorsivo che calcoli di $m(i, j)$

Tuttavia, questo algoritmo ricorsivo ha una complessità esponenziale e, quindi, non è migliore della tecnica a forza bruta che controlla tutti gli schemi di parentesizzazione

La ragione: nei diversi cammini dell'algoritmo ricorsivo (dell'albero di ricorsione) uno stesso sottoproblema può essere risolto più volte

Esempio: per calcolare $m(2, 5)$ abbiamo bisogno (tra l'altro) di calcolare $m(2, k)$ per ogni $2 \leq k < 5$ e quindi $m(2, 2)$, $m(2, 3)$ e $m(2, 4)$

Ora, per calcolare $m(2, 4)$ abbiamo bisogno di calcolare $m(2, k)$ per $2 \leq k < 4$ e quindi di nuovo $m(2, 2)$ e $m(2, 3)$

I problemi $m[2, 2]$ e $m[2, 3]$ vengono risolti due volte

Calcolo dei costi ottimi

Calcoliamo il costo ottimale applicando un metodo tabulare (usa delle tabelle) bottom-up

Assumiamo di dover moltiplicare n matrici A_i di dimensioni $p_{i-1} \times p_i$ per $i = 1, 2, ..., n$. L'input dell'algoritmo è una sequenza $p = \langle p_0, p_1, ..., p_n \rangle$ con $length[p] = n + 1$

La procedura usa due tabelle tabelle ausiliarie

- una tabella $m[1..n, 1..n]$ per memorizzare i costi $m(i, j)$
- una tabella $s[1..n, 1..n]$ memorizza l'indice k cui corrisponde il costo ottimo di $m[i, j]$. Questa tabella verrà usata in fase di costruzione della soluzione ottima

Calcolo dei costi ottimi

- Per implementare correttamente l'algoritmo bottom-up dobbiamo capire quali posizioni della tabella sono utilizzate per il calcolo di $m(i, j)$. Procediamo in base alla lunghezza della sequenza $A_i \cdots A_j$

Calcolo dei costi ottimi

- Per implementare correttamente l'algoritmo bottom-up dobbiamo capire quali posizioni della tabella sono utilizzate per il calcolo di $m(i, j)$. Procediamo in base alla lunghezza della sequenza $A_i \cdots A_j$
- (len $= 1$): $m[i, i] = 0$, per $i = 1, 2, ..., n$, non dipendono da nessuna posizione della tabella (casi base)

Calcolo dei costi ottimi

- Per implementare correttamente l'algoritmo bottom-up dobbiamo capire quali posizioni della tabella sono utilizzate per il calcolo di $m(i,j)$. Procediamo in base alla lunghezza della sequenza $A_i \cdots A_j$
- (len $= 1$): $m[i,i] = 0$, per $i = 1, 2, ..., n$, non dipendono da nessuna posizione della tabella (casi base)
- (len $= 2$): $m[i, i+1]$, con $i = 1, 2, ..., n-1$.

$$\min_{i \leq k < i+1} \left\{ m[i,k] + m[k+1, i+1] + p_{i-1} p_k p_{i+1} \right\}$$

Calcolo dei costi ottimi

- Per implementare correttamente l'algoritmo bottom-up dobbiamo capire quali posizioni della tabella sono utilizzate per il calcolo di $m(i,j)$. Procediamo in base alla lunghezza della sequenza $A_i \cdots A_j$
- (len $= 1$): $m[i,i] = 0$, per $i = 1, 2, ..., n$, non dipendono da nessuna posizione della tabella (casi base)
- (len $= 2$): $m[i, i+1]$, con $i = 1, 2, ..., n-1$.

$$\min_{i \leq k < i+1} \left\{ m[i,k] + m[k+1, i+1] + p_{i-1} p_k p_{i+1} \right\}$$

dipende da $m[i,i]$ e $m[i+1, i+1]$, cioè da posizioni della tabella corrispondenti al costo del prodotto di sequenze lunghe $1 < len = 2$

Calcolo dei costi ottimi

- (len $= 3$): $m[i, i+2]$ con $i = 1, 2, ..., n-2$.

$$\min_{i \leq k < i+2} \left\{ m[i,k] + m[k+1, i+2] + p_{i-1} p_k p_{i+2} \right\}$$

Calcolo dei costi ottimi

- (len $= 3$): $m[i, i+2]$ con $i = 1, 2, ..., n-2$.

$$\min_{i \le k < i+2} \left\{ m[i,k] + m[k+1, i+2] + p_{i-1} p_k p_{i+2} \right\}$$

che dipende da:

  - $m[i,i]$ e $m[i+1, i+2]$ per $k = i$
  - $m[i, i+1]$ e $m[i+2, i+2]$ per $k = i+1$

ossia, da posizioni della tabella corrispondenti al costo del prodotto di sequenze lunghe 1 o 2 $< len = 3$

Calcolo dei costi ottimi

In generale, sequenze la cui lunghezza è $h+1 \geq 2$ corrispondono a posizioni della tabella $m[i, i+h]$ con $h = 1, ..., n-1$ e $i = 1, 2, ..., n-h$.

$$m[i, i+h] = \min_{i \leq k < h+i} \left\{ m[i, k] + m[k+1, i+h] + p_{i-1} p_k p_{i+2} \right\}$$

Calcolo dei costi ottimi

In generale, sequenze la cui lunghezza è $h+1 \geq 2$ corrispondono a posizioni della tabella $m[i, i+h]$ con $h = 1, ..., n-1$ e $i = 1, 2, ..., n-h$.

$$m[i, i+h] = \min_{i \leq k < h+i} \left\{ m[i, k] + m[k+1, i+h] + p_{i-1} p_k p_{i+2} \right\}$$

Al variare di k nell'intervallo $i \leq k < i+h$, dipende da:

- $m[i, k]$ (prodotto di sequenze lunghe $k - i + 1$)
- $m[k+1, i+h]$ (prodotto di sequenze lunghe $(i+h) - (k+1) - 1 = i + h - k$)

Calcolo dei costi ottimi

In generale, sequenze la cui lunghezza è $h+1 \geq 2$ corrispondono a posizioni della tabella $m[i, i+h]$ con $h = 1, ..., n-1$ e $i = 1, 2, ..., n-h$.

$$m[i, i+h] = \min_{i \leq k < h+i} \left\{ m[i, k] + m[k+1, i+h] + p_{i-1} p_k p_{i+2} \right\}$$

Al variare di k nell'intervallo $i \leq k < i+h$, dipende da:

- $m[i, k]$ (prodotto di sequenze lunghe $k - i + 1$)
- $m[k+1, i+h]$ (prodotto di sequenze lunghe $(i+h) - (k+1) - 1 = i + h - k$)

Ora:

- $k < i + h$ implica $k - i + 1 < (i + h) - i + 1 = h + 1$
- $k \geq i$ implica $i + h - k \leq i + h - i = h < h + 1$

Calcolo dei costi ottimi

In generale, sequenze la cui lunghezza è $h+1 \geq 2$ corrispondono a posizioni della tabella $m[i, i+h]$ con $h = 1, ..., n-1$ e $i = 1, 2, ..., n-h$.

$$m[i, i+h] = \min_{i \leq k < h+i} \left\{ m[i, k] + m[k+1, i+h] + p_{i-1} p_k p_{i+2} \right\}$$

Al variare di k nell'intervallo $i \leq k < i+h$, dipende da:

- $m[i, k]$ (prodotto di sequenze lunghe $k - i + 1$)
- $m[k+1, i+h]$ (prodotto di sequenze lunghe $(i+h) - (k+1) - 1 = i + h - k$)

Ora:

- $k < i + h$ implica $k - i + 1 < (i + h) - i + 1 = h + 1$
- $k \geq i$ implica $i + h - k \leq i + h - i = h < h + 1$

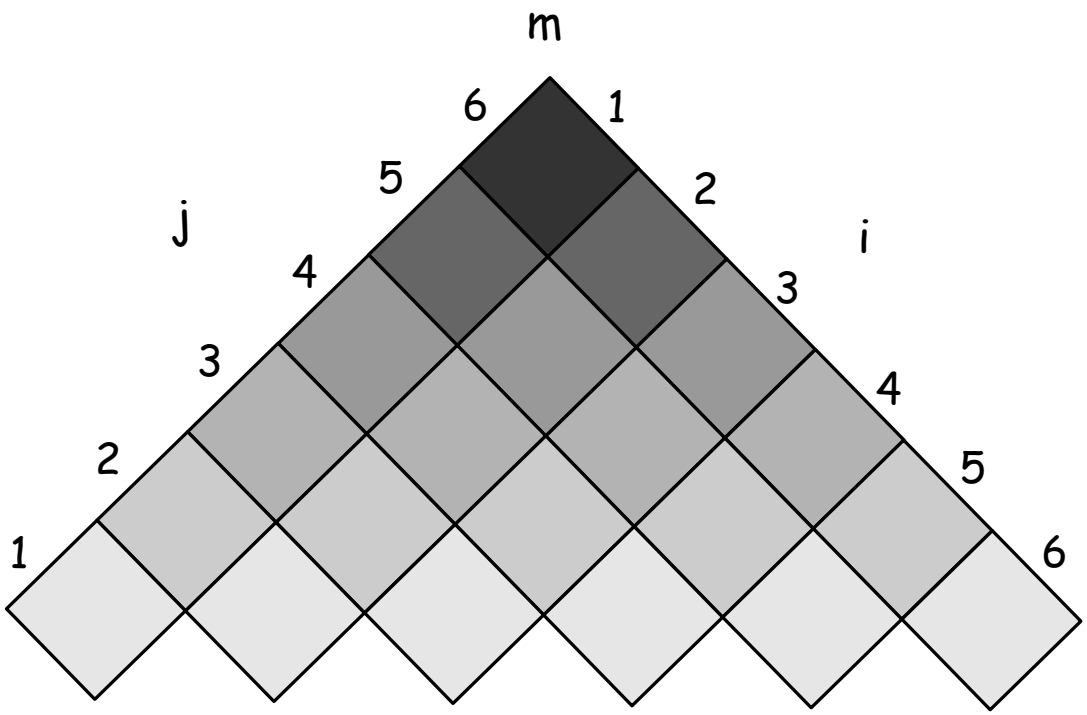
Calcolo dei costi ottimi

Riempiamo nell'ordine:

- tutte le posizioni $m[i, i]$ per $i = 1, ..., n$ (facile vanno poste a zero)
- tutte le posizioni $m[i, i + 1]$ per $i = 1, ..., n - 1$
- tutte le posizioni $m[i, i + 2]$ per $i = 1, ..., n - 2$
- tutte le posizioni $m[i, i + 3]$ per $i = 1, ..., n - 3$

  (in generale, tutte le posizioni della forma $m[i, i + h]$ con $h = 1, ..., n - 1$) ...
- una sola posizione della tabella corrisponde ad una sequenza di lunghezza n cioè $m[1, n]$ e viene riempita per ultima

Calcolo dei costi ottimi



Algoritmo per la parentesizzazione ottima

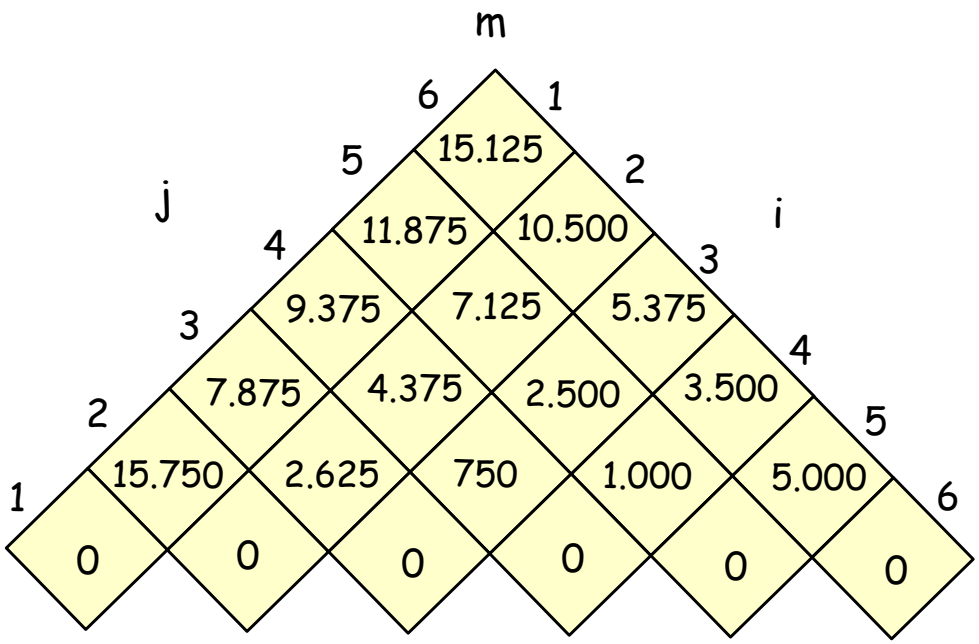
MATRIXCHAINORDER(p)

```
 1  n ← length[p] − 1
 2  for i ← 1 to n
 3  do m[i, i] ← 0
 4  for h ← 1 to n − 1
 5  do for i ← 1 to n − h
 6       do j ← i + h
 7          m[i, j] ← ∞
 8          for k ← i to j − 1
 9          do q ← m[i, k] + m[k + 1, j] + p[i − 1] · p[k] · p[j]
10             if q < m[i, j]
11                then m[i, j] ← q
12                     s[i, j] ← k
13  return m, s
```

Costruzione di una soluzione ottima

```
PrintOptimalParens(s, i, j)
1  if i = j
2     then Stampa "A_i"
3     else Stampa "("
4          PrintOptimalParens(s, i, s[i, j])
5          PrintOptimalParens(s, s[i, j] + 1, j)
6          Stampa ")"
```

Calcolo dei costi ottimi: un esempio



matrice	dimensione	matrice	dimensione
A_1	30×35	A_4	5×10
A_2	35×15	A_5	10×20
A_3	15×5	A_6	20×25

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[1,2] &= \min_{1 \le k < 2} \left\{ m[1,k] + m[k+1,2] + p_0 p_k p_2 \right\} \\
&= m[1,1] + m[2,2] + p_0 p_1 p_2 = 30 \cdot 35 \cdot 15 = 15.750
\end{aligned}
$$

$$
\begin{aligned}
m[2,3] &= \min_{2 \le k < 3} \left\{ m[2,k] + m[k+1,3] + p_1 p_k p_3 \right\} \\
&= m[2,2] + m[3,3] + p_1 p_2 p_3 = 35 \cdot 15 \cdot 5 = 2.625
\end{aligned}
$$

$$
\begin{aligned}
m[3,4] &= \min_{3 \le k < 4} \left\{ m[3,k] + m[k+1,4] + p_2 p_k p_4 \right\} \\
&= m[3,3] + m[4,4] + p_2 p_3 p_4 = 15 \cdot 5 \cdot 10 = 750
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[4,5] &= \min_{4 \le k < 5} \left\{ m[4,k] + m[k+1,5] + p_3 p_4 p_5 \right\} \\
&= m[4,4] + m[5,5] + p_3 p_4 p_5 = 5 \cdot 10 \cdot 20 = 1.000
\end{aligned}
$$

$$
\begin{aligned}
m[5,6] &= \min_{5 \le k < 6} \left\{ m[5,k] + m[k+1,6] + p_4 p_k p_6 \right\} \\
&= m[5,5] + m[6,6] + p_4 p_5 p_6 = 10 \cdot 20 \cdot 25 = 5.000
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[1,3] &= \min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 &= 2.625 + (30 \cdot 35 \cdot 5) \\ & = 7.875 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 &= 15.750 + (30 \cdot 15 \cdot 5) \\ & = 18.000 \end{cases} \\
&= 7.875
\end{aligned}
$$

$$
\begin{aligned}
m[2,4] &= \min \begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 &= 750 + (35 \cdot 15 \cdot 10) \\ & = 6.000 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 &= 2.625 + (35 \cdot 5 \cdot 10) \\ & = 4.375 \end{cases} \\
&= 4.375
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[3,5] &= \min \begin{cases} m[3,3] + m[4,5] + p_2 p_3 p_5 &= 1.000 + (15 \cdot 5 \cdot 20) \\ & = 2.500 \\ m[3,4] + m[5,5] + p_2 p_4 p_5 &= 750 + (15 \cdot 10 \cdot 20) \\ & = 3.750 \end{cases} \\
&= 2.500
\end{aligned}
$$

$$
\begin{aligned}
m[4,6] &= \min \begin{cases} m[4,4] + m[5,6] + p_3 p_4 p_6 &= 5.000 + (5 \cdot 10 \cdot 25) \\ & = 6.250 \\ m[4,5] + m[6,6] + p_3 p_5 p_6 &= 1.000 + (5 \cdot 20 \cdot 25) \\ & = 3.500 \end{cases} \\
&= 3.500
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[1,4] &= \min \begin{cases}
m[1,1] + m[2,4] + p_0 p_1 p_4 &= 4.375 + (30 \cdot 35 \cdot 10) \\
&= 14.857 \\
m[1,2] + m[3,4] + p_0 p_2 p_4 &= 15.750 + 750 + (30 \cdot 15 \cdot 10) \\
&= 21.000 \\
m[1,3] + m[4,4] + p_0 p_3 p_4 &= 7.875 + (30 \cdot 5 \cdot 10) \\
&= 9.375
\end{cases} \\
&= 9.375
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[2,5] &= \min \begin{cases}
m[2,2] + m[3,5] + p_1 p_2 p_5 &= 2.500 + (35 \cdot 15 \cdot 20) \\
&= 13.000 \\
m[2,3] + m[4,5] + p_1 p_3 p_5 &= 2.625 + 1.000 + (35 \cdot 5 \cdot 20) \\
&= 7.125 \\
m[2,4] + m[5,5] + p_1 p_4 p_5 &= 4.375 + (35 \cdot 10 \cdot 20) \\
&= 11.375
\end{cases} \\
&= 7.125
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[3,6] &= \min \begin{cases}
m[3,3] + m[4,6] + p_2 p_3 p_6 &= 3.500 + (15 \cdot 5 \cdot 25) \\
&= 5.375 \\
m[3,4] + m[5,6] + p_2 p_4 p_6 &= 750 + 5.000 + (15 \cdot 10 \cdot 25) \\
&= 9.500 \\
m[3,5] + m[6,6] + p_2 p_5 p_6 &= 2.500 + (15 \cdot 20 \cdot 25) \\
&= 10.000
\end{cases} \\
&= 5.375
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[1,5] &= \min \begin{cases}
m[1,1] + m[2,5] + p_0 p_1 p_5 &= 7.125 + (30 \cdot 35 \cdot 20) \\
&= 28.125 \\
m[1,2] + m[3,5] + p_0 p_2 p_5 &= 15.750 + 2.500 + (30 \cdot 15 \cdot 20) \\
&= 27.250 \\
m[1,3] + m[4,5] + p_0 p_3 p_5 &= 7.875 + 1.000 + (30 \cdot 5 \cdot 20) \\
&= 11.875 \\
m[1,4] + m[5,5] + p_0 p_4 p_5 &= 9.375 + (30 \cdot 10 \cdot 20) \\
&= 15.375
\end{cases} \\
&= 11.875
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[2,6] &= \min \begin{cases}
m[2,2] + m[3,6] + p_1 p_2 p_6 &= 5.375 + (35 \cdot 15 \cdot 25) \\
&= 18.500 \\
m[2,3] + m[4,6] + p_1 p_3 p_6 &= 2.625 + 3.500 + (35 \cdot 5 \cdot 25) \\
&= 10.500 \\
m[2,4] + m[5,6] + p_1 p_4 p_6 &= 4.375 + 5.000 + (35 \cdot 10 \cdot 25) \\
&= 18.125 \\
m[2,5] + m[6,6] + p_1 p_5 p_6 &= 7.125 + (35 \cdot 20 \cdot 25) \\
&= 24.465
\end{cases} \\
&= 10.500
\end{aligned}
$$

Calcolo dei costi ottimi: un esempio

$$
\begin{aligned}
m[1,6] &= \min \begin{cases}
m[1,1] + m[2,6] + p_0 p_1 p_6 &= 10.500 + (30 \cdot 35 \cdot 25) \\
&= 36.750 \\
m[1,2] + m[3,6] + p_0 p_2 p_6 &= 15.750 + 5.375 + (30 \cdot 15 \cdot 25) \\
&= 32.375 \\
m[1,3] + m[4,6] + p_0 p_3 p_6 &= 7.875 + 3.500 + (30 \cdot 5 \cdot 25) \\
&= 15.125 \\
m[1,4] + m[5,6] + p_0 p_4 p_6 &= 9.375 + 5.000 + (30 \cdot 10 \cdot 25) \\
&= 21.875 \\
m[1,5] + m[6,6] + p_0 p_5 p_6 &= 11.875 + (30 \cdot 20 \cdot 25) \\
&= 26.875
\end{cases} \\
&= 15.125
\end{aligned}
$$