Studieremo alcune tecniche per il progetto di algoritmi e di strutture dati:

Programmazione dinamica

Algoritmi golosi

Analisi ammortizzata

Vedremo poi alcuni tipi di strutture dati importanti per le applicazioni:

B-alberi

Strutture dati per insiemi disgiunti

Programmazione Dinamica

Esempio: taglio delle aste

Problema del taglio delle aste

E' data un'asta metallica di lunghezza n che deve essere tagliata in pezzi di lunghezza intera (con un costo di taglio trascurabile).

Per ogni lunghezza $l = 1,\ldots,n$ è dato il prezzo p_l a cui si possono vendere i pezzi di quella lunghezza.

Si vuole decidere come tagliare l'asta in modo da rendere massimo il ricavo della vendita dei pezzi ottenuti.

Esempio

l	1	2	3	4	5	6	7	8	9	10
p_l	1	5	8	9	10	17	17	20	24	30

Lunghezza asta n	Ricavo massimo r_n	Suddivisione ottima
1	1	1
2	5	2
3	8	3
4	10	2+2
5	13	2+3
6	17	6
7	18	1+6 o 2+2+3
8	22	2+6
9	25	3+6
10	30	10

Un'asta di lunghezza n può essere tagliata in 2^{n-1} modi distinti in quanto abbiamo una opzione tra tagliare o non tagliare in ogni posizione intera $1,\ldots,n-1$.

Ad esempio per $n = 4$ abbiamo i seguenti 8 modi

Suddivisioni

4	**1+1+2**
1+3	**1+2+1**
2+2	**2+1+1**
3+1	**1+1+1+1**

In generale il ricavo massimo r_n o è il costo p_n dell'asta intera oppure si ottiene effettuando un primo taglio in posizione i e quindi sommando i ricavi massimi del primo e del secondo pezzo, ossia

$$r_n = r_i + r_{n-i}$$

Quindi

$$r_n = \begin{cases} p_1 & \text{se } n = 1 \\ \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1) & \text{se } n > 1 \end{cases}$$

Osserviamo che la soluzione ottima del problema di ottiene da soluzioni ottime di sottoproblemi. Diciamo che il problema ha ***sottostruttura ottima***.

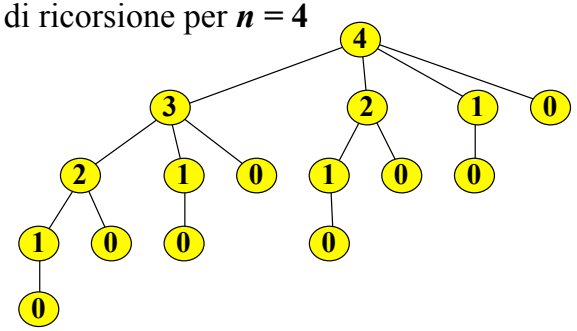
Otteniamo una struttura ricorsiva più semplice se invece di scegliere la posizione i di un primo taglio intermedio scegliamo la lunghezza i del primo pezzo per cui $r_n = p_i + r_{n-i}$

$$r_n = \begin{cases} 0 & \text{se } n = 0 \\ \max\limits_{1 \le i \le n}(p_i + r_{n-i}) & \text{se } n > 0 \end{cases}$$

```
Cut-Rod(p, n)
  if n == 0
     return 0
  q = -1
  for i =1 to n
     q = max(q, p[i]+Cut-Rod(p, n-i))
  return q
```

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

Albero di ricorsione per $n = 4$



Lo stesso problema di dimensione **2** viene risolto due volte, quello di dimensione **1** quattro volte e quello di dimensione **0** otto volte.

Questo spiega la complessità 2^n.

Ripetizione dei sottoproblemi!!

Possiamo ridurre la complessità evitando di risolvere più volte gli stessi problemi.

Un primo modo per farlo è dotare l'algoritmo di un blocco note in cui ricordare le soluzioni dei problemi già risolti: metodo top-down con annotazione.

Un secondo modo per farlo è calcolare prima i problemi più piccoli memorizzandone le soluzioni e poi usare tali soluzioni per risolvere i problemi più grandi: metodo bottom-up.

Versione top-down con annotazione:

```
Memoized-Cut-Rod(p, n)
  for i = 0 to n          // inizializza il blocco note
    r[i] = -1
  return Cut-Rod-Aux(p, n, r)

Cut-Rod-Aux(p, j, r)
  if r[j] ≥ 0      // il problema è già stato risolto
    return r[j]
  if j == 0
    q = 0
  else q = -1
      for i = 1 to j
        q = max(q, p[i] + Cut-Rod-Aux(p, j-i, r))
  r[j] = q
  return q
```

$T(n) = \Theta(n^2)$

Versione bottom-up:

```
Bottom-Up-Cut-Rod(p, n)
  r[0] = 0      // il problema più semplice
  for j = 1 to n
    q = -1
    for i = 1 to j
      q = max(q, p[i] + r[j-i])
    r[ j] = q
  return r[n]
```

$T(n) = \Theta(n^2)$

Versione bottom-up estesa per calcolare la soluzione ottima e non solo il suo valore

```
Extended-Bottom-Up-Cut-Rod( p, n)
  r[0] = 0
  for j = 1 to n
    q = -1
    for i = 1 to j
      if q < p[i]+r[ j - i]
        q = p[i]+r[ j - i]
        s[ j] = i      // memorizzo il taglio ottimo
    r[ j] = q
  return r ed s
```

La seguente procedura calcola e stampa la soluzione ottima:

```
Print-Cut-Rod-Solution( p, n)
  (r, s) = Extended-Bottom-Up-Cut-Rod( p, n)
  j = n
  while j > 0
    print s[j]
    j = j - s[j]
```

Moltiplicazione di matrici

L'algoritmo per moltiplicare due matrici A e B di dimensioni $p \times q$ e $q \times r$ è:

```
Matrix-Multiply(A, B)
  for i = 1 to A.rows
    for j = 1 to B.columns
      C[i, j] = 0
      for k = 1 to A.columns
        C[i, j] = C[i, j] + A[i, k] B[k, j]
  return C
```

Esso richiede $p \times q \times r$ prodotti scalari

Problema della moltiplicazione di matrici

Si deve calcolare il prodotto

$$A_1 A_2 \ldots A_n$$

di n matrici di dimensioni

$$p_0 \times p_1 , p_1 \times p_2 , \ldots , p_{n-1} \times p_n$$

Poiché il prodotto di matrici è associativo possiamo calcolarlo in molti modi.

Esempio:

Per calcolare il prodotto $A_1 A_2 A_3$ di 3 matrici di dimensioni **200×5**, **5×100**, **100×5** possiamo:

a) moltiplicare A_1 per A_2 (**100000** prodotti scalari) e poi moltiplicare per A_3 la matrice **200×100** ottenuta (**100000** prodotti scalari).

In totale **200000** prodotti scalari.

b) moltiplicare A_2 per A_3 (**2500** prodotti scalari) e poi moltiplicare A_1 per la matrice **5×5** ottenuta (**5000** prodotti scalari).

In totale **7500** prodotti scalari.

Vogliamo trovare il modo per minimizzare il numero totale di prodotti scalari.

In quanti modi possiamo calcolare il prodotto?

Tanti quante sono le parentesizzazioni possibili del prodotto $A_1 A_2 \ldots A_n$.

Ad esempio per $n = 4$:

$$(A_1 (A_2 (A_3 A_4)))$$
$$(A_1 ((A_2 A_3) A_4))$$
$$((A_1 A_2) (A_3 A_4))$$
$$((A_1 (A_2 A_3)) A_4)$$
$$(((A_1 A_2) A_3) A_4)$$

Il numero $P(n)$ di parentesizzazioni possibili del prodotto $A_1 A_2 \ldots A_n$ di n matrici si esprime ricorsivamente come segue:

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

Si può dimostrare che $P(n)$ cresce in modo esponenziale.

Quindi, tranne per valori di n molto piccoli, non è possibile enumerare tutte le parentesizzazioni.

Passo 1: ***struttura di una parentesizzazione ottima***

Supponiamo che una parentesizzazione ottima di $A_1 A_2 \ldots A_n$ preveda come ultima operazione il prodotto tra la matrice $A_{1..k}$ (prodotto delle prime k matrici $A_1 \ldots A_k$) e la matrice $A_{k+1..n}$ (prodotto delle ultime n-k matrici $A_{k+1} \ldots A_n$).

Le parentesizzazioni di $A_1 \ldots A_k$ e di $A_{k+1} \ldots A_n$ sono parentesizzazioni ottime per il calcolo di $A_{1..k}$ e di $A_{k+1..n}$.

Perché?

Passo 2: **soluzione ricorsiva**

Prendiamo come sottoproblemi il calcolo dei prodotti parziali $A_{i..j}$ delle matrici $A_i \ldots A_j$.

Ricordiamo che la generica matrice A_i ha dimensioni $p_{i-1} \times p_i$.

Di conseguenza la matrice prodotto parziale $A_{i..j}$ è una matrice $p_{i-1} \times p_j$ con lo stesso numero p_{i-1} di righe della prima matrice A_i e lo stesso numero p_j di colonne dell'ultima matrice A_j .

Se $i = j$ allora $A_{i..j} = A_i$ ed $m[i,i] = 0$.

Se $i < j$ allora $A_{i..j} = A_i \ldots A_j$ si può calcolare come prodotto delle due matrici $A_{i..k}$ e $A_{k+1..j}$ con k compreso tra i e j-1.

Il costo di questo prodotto è $p_{i-1} p_k p_j$.

Quindi

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

Passo 3

Esempio

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

		35	15	5	10	20	25	p
		1	2	3	4	5	6	j
30	1	$A_{1..1}$ m 0 k	$A_{1..2}$ 15750 1	$A_{1..3}$ 7900 1	$A_{1..4}$ 9400 3	$A_{1..5}$ 11900 3	$A_{1..6}$ 15125 3	
35	2		$A_{2..2}$ m 0 k	$A_{2..3}$ 2625 2	$A_{2..4}$ 4375 3	$A_{2..5}$ 7125 3	$A_{2..6}$ 10500 3	
15	3			$A_{3..3}$ m 0 k	$A_{3..4}$ 750 3	$A_{3..5}$ 1500 4	$A_{3..6}$ 5375 3	
5	4				$A_{4..4}$ m 0 k	$A_{4..5}$ 1000 4	$A_{4..6}$ 3500 5	
10	5					$A_{5..5}$ m 0 k	$A_{5..6}$ 5000 5	
							$A_{6..6}$ m 0 k	

$A_{1..1}A_{2..6}$: $0+10500+30\times35\times25 = 36750$
$A_{1..2}A_{3..6}$: $15750+5375+30\times15\times25 = 32375$
$A_{1..3}A_{4..6}$: $7900+3500+30\times5\times25 = 15150$
$A_{1..4}A_{5..6}$: $9400+5000+30\times10\times25 = 21900$
$A_{1..5}A_{6..6}$: $11900+0+30\times20\times25 = 26900$

Passo 3: *calcolo del costo minimo*

```
Matrix-Chain-Order(p, n)
  for i = 1 to n
     m[i, i] = 0
  for j = 2 to n
     for i = j-1 downto 1
        m[i, j] = ∞
        for k = i to j-1
           q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
           if q < m[i, j]
              m[i, j] = q
              s[i, j] = k
  return m, s
```

Complessità: $O(n^3)$

Passo 4

Esempio

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

	1	2	3	4	5	6	j
1	$A_{1..1}$ m s	$A_{1..2}$ 1	$A_{1..3}$ 1	$A_{1..4}$ 3	$A_{1..5}$ 3	$A_{1..6}$ 3	
2		$A_{2..2}$ m s	$A_{2..3}$ 2	$A_{2..4}$ 3	$A_{2..5}$ 3	$A_{2..6}$ 3	
3			$A_{3..3}$ m s	$A_{3..4}$ 3	$A_{3..5}$ 3	$A_{3..6}$ 3	
4				$A_{4..4}$ m s	$A_{4..5}$ 4	$A_{4..6}$ 5	
5					$A_{5..5}$ m s	$A_{5..6}$ 5	
6						$A_{6..6}$ m s	
i							

$A_{1..6}$

$(A_{1..3}\ A_{4..6})$

$((A_1\ A_{2..3})\ (A_{4..5}\ A_6))$

$((A_1\ (A_2\ A_3))\ ((A_4\ A_5)\ A_6))$

Passo 4:

stampa della soluzione ottima

```
Print-Optimal-Parens(s, i, j)
  if i == j
     print "A_i"
  else
     k = s[i, j]
     print "("
     Print-Optimal-Parens(s, i, k)
     print "×"
     Print-Optimal-Parens(s, k+1, j)
     print ")"
```

Complessità: $O(n)$

Calcolo del prodotto di una sequenza di matrici

```
Matrix-Chain-Multiply(A1...An , i, j, s)
  if i == j
    return Ai
  else
    k = s[i, j]
    A = Matrix-Chain-Multiply(A1...An, i, k, s)
    B = Matrix-Chain-Multiply(A1...An, k+1, j, s)
  return Matrix-Multiply(A, B)
```

Si potrebbe anche usare direttamente la definizione ricorsiva del costo minimo per il prodotto di matrici

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

per calcolarlo ricorsivamente senza usare le matrici ***m*** ed ***s***.

```
Rec-Matrix-Chain-Cost(p, i, j)
  if i = j
    return 0
  else
    cmin = ∞
    for k = i to j-1
      q = Rec-Matrix-Chain-Cost(p, i, k) +
          Rec-Matrix-Chain-Cost(p, k+1, j) + p_{i-1} p_k p_j
      if q < cmin
        cmin = q
    return cmin
```

Complessità $T(n)$ con $n = j-i+1$

$$T(n) = \begin{cases} a & \text{se } n = 1 \\ a + \sum\limits_{h=1}^{n-1}\left(T(h) + T(n-h) + b\right) & \text{se } n > 1 \end{cases}$$

$$T(n) = \begin{cases} a & \text{se } n = 1 \\ a + \sum\limits_{h=1}^{n-1}\left(T(h) + T(n-h) + b\right) & \text{se } n > 1 \end{cases}$$

Per sostituzione si può dimostrare che

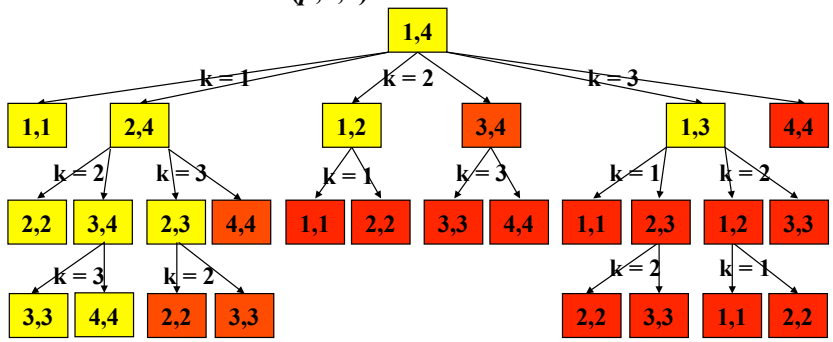
$$T(n) \ge c2^{n-1}$$

dove $c = \min(a,b)$.

Quindi $T(n) = \Omega(2^n)$.

Causa della complessità esponenziale:

***Rec-Matrix-Chain-Cost*(*p*,1,4)**



La complessità diventa esponenziale perché vengono risolti più volte gli stessi sottoproblemi.

Possiamo evitare il ricalcolo dei costi minimi dei sottoproblemi dotando la procedura ricorsiva di un blocco notes (una matrice ***m*** di dimensione ***n*×*n***) in cui annotare i costi minimi dei sottoproblemi già risolti.

```
Memoized-Matrix-Chain-Order(p, n)
  for i = 1 to n
    for j = i to n do
      m[i, j] = ∞
  return Memoized-Chain-Cost(p,1, n, m)
```

```
Memoized-Chain-Cost(p, i, j, m)
  if m[i, j] = ∞
    if i == j
      m[i, j] = 0
    else
      for k = i to j-1
        q = Memoized-Chain-Cost(p, i, k, M)
            + Memoized-Chain-Cost(p, k+1, j, M)
            + p_{i-1} p_k p_j
        if q < m[i, j]
          m[i, j] = q
  return m[i, j]
```

Complessità: $O(n^3)$