Δυναμικός προγραμματισμός (Dynamic Programming)

Χαρακτηριστικά γνωρίσματα

- Η βέλτιστη λύση του προβλήματος περιέχει βέλτιστες λύσεις των υπο-προβλημάτων του.
- Επικαλυπτόμενα υπό-προβλήματα.

Θα εξετάσουμε:

- Αριθμούς Fibonacci.
- Πολλαπλασιασμός αλυσίδας πινάκων.
- Μεγαλύτερη κοινή υπό-ακολουθία.
- Ελάχιστα μονοπάτια για όλα τα ζεύγη κόμβων.

Αριθμοί Fibonacci

Ο n-οστός *αριθμός Fibonacci* $F(n)$ ορίζεται από την αναδρομική σχέση:

$$F(0) = 0$$
$$F(1) = 1$$
$$F(n) = F(n-1) + F(n-2) \qquad \text{για } n \geq 2$$

$F(2) = 1,\ F(3) = 2,\ F(4) = 3,\ F(5) = 5,\ F(6) = 8,\ F(7) = 13,\ F(8) = 21,\ \cdots$

Ένας προφανής αλγόριθμος για τον υπολογισμό του $F(n),\ n \geq 1$

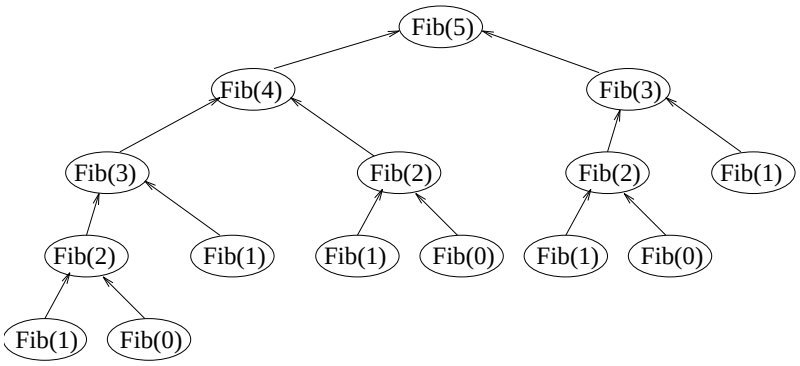
Fibinacci(n)
if $n < 2$ **then return** (n)
else return $(Fibonacci(n-1) + Fibonacci(n-2))$

Ανάλυση $T(n) = \begin{cases} 1 & n < 2 \\ T(n-1) + T(n-2) & n \geq 2 \end{cases}$

$$\Longrightarrow \quad T(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \quad \Longrightarrow \quad T(n) = \Theta(\phi^n)$$

$\phi = \frac{1+\sqrt{5}}{2} = 1.61803, \quad \hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803$

• Για $n = 5$, το 'δένδρο αναδρομής' (recursion tree) είναι το:



Υπολογίζουμε το : $F(3)$ 2 φορές,
$F(2)$ 3 φορές,
$F(1)$ 5 φορές,
$F(0)$ 3 φορές.

Υπάρχουν πολλοί περιττοί υπολογισμοί!

Μία καλύτερη λύση

• Υπολόγισε το $F(n)$ ξεκινώντας από τη βάση (bottom-up).

$Non\_Recursive\_Fibonacci(n)$
/* $A[1..n]$ is an array in which we store $F(1) \cdots F(n)$ */
$A[0] = 0$
$A[1] = 1$
for $i = 2$ **to** n **do**
    $A[i] = A[i-1] + A[i-2]$
return $A[n]$

Ανάλυση: $\Theta(n)$

Πολλαπλασιασμός αλυσίδας πινάκων

Δίδεται μία ακολουθία (αλυσίδα) $< A_1,\ A_2,\ \cdots,\ A_n >$ από n πίνακες και θέλουμε να υπολογίσουμε το γινόμενο $A_1 \times A_2 \times \cdots \times A_n$.

- $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$, δηλ., ο πολλαπλασιασμός πινάκων είναι προσεταιριστικός.

  $\Longrightarrow$ Όλες οι παρενθετικοποιήσεις δίνουν το ίδιο αποτέλεσμα.

Παράδειγμα Για $n = 4$, υπάρχουν 5 διακριτές παρενθετικοποιήσεις:

$$(A_1 \times (A_2 \times (A_3 \times A_4)))$$

$$(A_1 \times ((A_2 \times A_3) \times A_4))$$

$$((A_1 \times A_2) \times (A_3 \times A_4))$$

$$((A_1 \times (A_2 \times A_3)) \times A_4)$$

$$(((A_1 \times A_2) \times A_3) \times A_4)$$

Ορισμός Ένα γινόμενο πινάκων είναι *πλήρως παρενθετικοποιημένο* εάν είναι ένας απλός πίνακας ή το γινόμενο δύο πλήρως παρενθετικοποιημένων γινομένων που περικλείεται από παρενθέσεις.

Σημείωση Ο ορισμός αυτός μπορεί να γενικευθεί για κάθε είδους δυαδικού τελεστή.

Άσκηση Να δειχθεί ότι η πλήρη παρενθετικοποίηση μιας μαθηματικής έκφρασης n στοιχείων έχει ακριβώς $n - 1$ ζευγάρια παρενθέσεων. (Υποθέτουμε δυαδικούς τελεστές.)

```
Matrix_multiply(A, B)
if columns(A) ≠ rows(B)
then error "incompatible dimension"
else for i = 1 to rows(A) do
        for j = 1 to columns(B) do
            C[i, j] = 0
            for k = 1 to columns(A) do
                C[i, j] = C[i, j] + A[i, k] · B[k, j]
```

- Εάν ο A είναι ένας $p \times q$ πίνακας και ο B είναι ένας $q \times r$ πίνακας, τότε ο C είναι ένας $q \times r$ πίνακας.
- Ο αλγόριθμος εκτελεί ακριβώς pqr πολλαπλασιασμούς.

Παράδειγμα $A_1 = 10 \times 100, \quad A_2 = 100 \times 5, \quad A_3 = 5 \times 50$

$((A_1 \times A_2) \times A_3)$ χρειάζεται $(10 \cdot 100 \cdot 5) + (10 \cdot 5 \cdot 50) = 7,500$ πολλαπλασιασμούς.
$(A_1 \times (A_2 \times A_3))$ χρειάζεται $(10 \cdot 100 \cdot 50) + (100 \cdot 5 \cdot 50) = 75,000$ πολλαπλασιασμούς.

$\Longrightarrow$ Η σειρά με την οποία εκτελούνται οι πολλαπλασιασμοί των πινάκων είναι πολύ σημαντική!

Ερώτηση Ποιος είναι ο αριθμός των δυνατών παρενθετικοποιήσεων·

Απάντηση Συμβολίζουμε τον αριθμό αυτό με $P(n)$.

$$\underbrace{(A_1 \times A_2 \times \cdots \times A_k)}_{P(k)} \times \underbrace{(A_{k+1} \times \cdots \times A_n)}_{P(n-k)}$$

Μπορούμε να γράψουμε την αναδρομική σχέση:

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

Η λύση είναι η ακολουθία των αριθμών Catalan. $P(n) = C(n-1)$ όπου,

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega(4^n/n^{3/2})$$

(Εκθετικό ως προς n!)

Η δομή της βέλτιστης παρενθετικοποίησης

- $A_{i..j}$ συμβολίζει τον πίνακα που προκύπτει από τον πολλαπλασιασμό $A_i \times \cdots \times A_j$.

Παρατήρηση Μία βέλτιστη παρενθετικοποίηση του $A_1 \times \cdots \times A_n$ χωρίζει το γινόμενο μεταξύ των A_k και A_{k+1} για κάποιο ακέραιο $k,\ 1 \leq k < n$.

$$((A_1 \times A_2 \times \cdots \times A_k) \ \times \ (A_{k+1} \times \cdots \times A_n))$$

Θεώρημα Η παρενθετικοποίηση της υπό-αλυσίδας $A_1 \times \cdots \times A_k$ σε μία βέλτιστη παρενθετικοποίηση της $A_1 \times \cdots \times A_n$ είναι επίσης βέλτιστη.

Απόδειξη Εάν υπάρχει μία καλύτερη παρενθετικοποίηση για το $A_1 \times \cdots \times A_k$, τότε μπορούμε να κατασκευάσουμε μία καλύτερη παρενθετικοποίηση από τη βέλτιστη για το $A_1 \times \cdots \times A_n$. Μία καθαρή αντίφαση.

□

Σημείωση Όμοιο θεώρημα ισχύει για την υπό-αλυσίδα $A_{k+1} \times \cdots \times A_n$.

Μία αναδρομική λύση

- $m[i,j]$ συμβολίζει τον ελάχιστο αριθμό των αναγκαίων πολλαπλασιασμών για τον υπολογισμό του $A_{i..j}$.
- Θέλουμε να βρούμε την παρενθετικοποίηση του χρειάζεται $m[1,n]$ πολλαπλασιασμούς για τον υπολογισμό του $A_{1..n}$.
- Υποθέτουμε ότι ο πίνακας A_i είναι διαστάσεων $[p_{i-1} \times p_i]$.

Μπορούμε να υπολογίσουμε το $m[1,n]$ μέσω της αναδρομικής σχέσης:

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

- Ένας αναδρομικός αλγόριθμος βασιζόμενος στην παραπάνω σχέση χρειάζεται **τουλάχιστον** εκθετικό χρόνο για την εκτέλεσή του!

Μία λύση δυναμικού προγραμματισμού

Παρατήρηση Πρέπει να λύσουμε $\Theta(n^2)$ υπό-προβλήματα, ένα για κάθε τιμή του ζεύγους i και j που ικανοποιεί τη σχέση $1 \leq i \leq j \leq n$.

Ιδέα: Θα υπολογίσουμε το $m[1, n]$ ξεκινώντας από τη βάση (bottom-up). Θα υπολογίσουμε όλα τα $m[i, j]$ για αυξανόμενες τιμές του $(j - i)$.

```
Matrix_Chain_order(p)
/* p =< p_0, p_1, ..., p_n >, the dimensions of the matrices */
n = length(p) − 1
for i = 1 to n do
    m[i, i] = 0
for l = 2 to n do                /* l denotes the "length" of A_{i..j} */
    for i = 1 to n − l + 1 do
        j = i + l − 1
        m[i, j] = 0
        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
return m
```

Ανάλυση: $\Theta(n^3)$

Παράδειγμα

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

M (i \ j)	1	2	3	4	5	6
1	0	15,750	7,875	9,375	11,875	15,125
2		0	2,625	4,375	7,125	10,500
3			0	750	2,500	5,375
4				0	1,000	3,500
5					0	5,000
6						0

$m[1, 2] = 30 \cdot 35 \cdot 15 = 15,750$

$$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + p_0 \cdot p_1 \cdot p_3 = 0 + 2,625 + 5,250 = \mathbf{7,875} \\ m[1,2] + m[3,3] + p_0 \cdot p_2 \cdot p_3 = 15,750 + 0 + 2,250 = 18,000 \end{cases}$$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + p_0 \cdot p_1 \cdot p_4 = 0 + 4,375 + 10,500 = 14,875 \\ m[1,2] + m[3,4] + p_0 \cdot p_2 \cdot p_4 = 15,750 + 750 + 4,500 = 21,000 \\ m[1,3] + m[4,4] + p_0 \cdot p_3 \cdot p_4 = 7,875 + 0 + 1,500 = \mathbf{9,375} \end{cases}$$

Ερώτηση Πως θα ανακτήσουμε την παρενθετικοποίηση·

Απάντηση Μέσω ενός επιπρόσθετου πίνακα $split$ όπου, $split[i,j]$ δηλώνει τη θέση όπου χωρίζουμε το γινόμενο $A_{i..j}$.

$Matrix\_Chain\_order\_1(p)$
/* $p =< p_0,\ p_1,\ \cdots,\ p_n >$, the dimensions of the matrices */
$n = length(p) - 1$
for $i = 1$ **to** n **do**
    $m[i,i] = 0$
for $l = 2$ **to** n **do** /* l denotes the "length" of $A_{i..j}$ */
    for $i = 1$ **to** $n - l + 1$ **do**
        $j = i + l - 1$
        $m[i,j] = 0$
        for $k = 1$ **to** $j - 1$ **do**
            $q = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$
            if $q < m[i,j]$ **then**
                $m[i,j] = q$
                $split[i,j] = k$
return $m,\ split$

Ο αλγόριθμος πολλαπλασιασμού αλυσίδας πινάκων

$Matrix\_Chain\_Multiply(A, split, i, j)$
/* Computes $A_{i..j}$. A is the set of matrices */

if $j > i$ **then** $X = Matrix\_Chain\_Multiply(A, split, i, split[i,j])$
    $Y = Matrix\_Chain\_Multiply(A, split, split[i,j] + 1, j)$
    return $Matrix\_Multiply(X, Y)$
else return A_i

- **Στοιχεία δυναμικού προγραμματισμού**
  - Η βέλτιστη λύση του προβλήματος περιέχει βέλτιστες λύσεις των υπό-προβλημάτων του.
  - Επικαλυπτόμενα υπό-προβλήματα.

- **Απομνημόνευση (Memoization)** Χρησιμοποίησε τον αναδρομικό αλγόριθμο, αλλά αποθήκευσε την λύση σε κάθε υπό-πρόβλημα για μελλοντική χρήση. Με τον τρόπο αυτό λύνουμε το κάθε υπό-πρόβλημα μόνο μία φορά.