Αλγόριθμοι και Πολυπλοκότητα

Αρχοντία Γιαννοπούλου
Όλγα Φουρτουνέλλη

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Δυναμικός Προγραμματισμός

Σακίδιο

Μέγιστη Κοινή Υπακολουθία

Αλγοριθμικά Μοντέλα

Απληστία. Χτίσε μια λύση σταδιακά, βελτιστοποιώντας μυωπικά κάποιο τοπικό κριτήριο.

Διαίρει και κυρίευε. Διάσπασε ένα πρόβλημα σε δύο υποπροβλήματα, λύσε κάθε υποπρόβλημα ανεξάρτητα, και συνδύασε τις λύσεις των υποπροβλημάτων για να δημιουργήσεις την λύση του αρχικού προβλήματος.

Αλγοριθμικά Μοντέλα

Απληστία. Χτίσε μια λύση σταδιακά, βελτιστοποιώντας μυωπικά κάποιο τοπικό κριτήριο.

Διαίρει και κυρίευε. Διάσπασε ένα πρόβλημα σε δύο υποπροβλήματα, λύσε κάθε υποπρόβλημα ανεξάρτητα, και συνδύασε τις λύσεις των υποπροβλημάτων για να δημιουργήσεις την λύση του αρχικού προβλήματος.

Δυναμικός προγραμματισμός.

Αλγοριθμικά Μοντέλα

Απληστία. Χτίσε μια λύση σταδιακά, βελτιστοποιώντας μυωπικά κάποιο τοπικό κριτήριο.

Διαίρει και κυρίευε. Διάσπασε ένα πρόβλημα σε δύο υποπροβλήματα, λύσε κάθε υποπρόβλημα ανεξάρτητα, και συνδύασε τις λύσεις των υποπροβλημάτων για να δημιουργήσεις την λύση του αρχικού προβλήματος.

Δυναμικός προγραμματισμός. Διάσπασε ένα πρόβλημα σε μια σειρά από επικαλυπτόμενα υποπροβλήματα, και δόμησε σωστές λύσεις για όλο και μεγαλύτερα υποπροβλήματα.

Σύνοψη

- Χαρακτηρισμός δομής του προβλήματος.
- Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης.
- Υπολογισμός της τιμής της βέλτιστης λύσης.
- Κατασκευή της βέλτιστης λύσης από την υπολογισμένη πληροφορία.

Σακίδιο (Knapsack)

Το πρόβλημα του Σακιδίου

- Μας δίνονται n αντικείμενα και ένα ''σακίδιο''.
- Το αντικείμενο i έχει βάρος $w_i > 0$ κιλά και αξία $v_i > 0$.
- Το σακίδιο έχει αντοχή μέχρι W κιλά.
- Στόχος: Να γεμίσουμε το σακίδιο μεγιστοποιώντας την αξία των αντικειμένων.

Σακίδιο (Knapsack)

Το πρόβλημα του Σακιδίου

- Μας δίνονται n αντικείμενα και ένα ''σακίδιο''.
- Το αντικείμενο i έχει βάρος $w_i > 0$ κιλά και αξία $v_i > 0$.
- Το σακίδιο έχει αντοχή μέχρι W κιλά.
- Στόχος: Να γεμίσουμε το σακίδιο μεγιστοποιώντας την αξία των αντικειμένων.

Άπληστα: Θα μπορούσαμε να γεμίσουμε το σακίδιο επιλέγοντας κάθε φορά το αντικείμενο με το μεγαλύτερο λόγο $\frac{v_i}{w_i}$.

Δεν είναι βέλτιστο!

Σακίδιο (Knapsack)

Το πρόβλημα του Σακιδίου

- Μας δίνονται n αντικείμενα και ένα ''σακίδιο''.
- Το αντικείμενο i έχει βάρος $w_i > 0$ κιλά και αξία $v_i > 0$.
- Το σακίδιο έχει αντοχή μέχρι W κιλά.
- Στόχος: Να γεμίσουμε το σακίδιο μεγιστοποιώντας την αξία των αντικειμένων.

Άπληστα: Θα μπορούσαμε να γεμίσουμε το σακίδιο επιλέγοντας κάθε φορά το αντικείμενο με το μεγαλύτερο λόγο $\frac{v_i}{w_i}$.

Δεν είναι βέλτιστο!

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Τα αντικείμενα 3 και 4 έχουν αξία 40 και βάρος 11 ενώ τα αντικείμενα 5, 2, και 1 έχουν αξία 35.

Δυναμικός Προγραμματισμός: Απόπειρα

$\mathrm{OPT}(i) =$ μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$.

Δυναμικός Προγραμματισμός: Απόπειρα

$\mathrm{OPT}(i) =$ μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.

- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.

Δυναμικός Προγραμματισμός: Απόπειρα

$\mathrm{OPT}(i) =$ μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$.

- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.

Δυναμικός Προγραμματισμός: Απόπειρα

$\mathrm{OPT}(i) =$ μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$.
- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Η επιλογή του αντικειμένου i δεν εξαιρεί άμεσα την επιλογή κάποιου άλλου αντικειμένου.

Δυναμικός Προγραμματισμός: Απόπειρα

$\mathrm{OPT}(i) =$ μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$.

- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Η επιλογή του αντικειμένου i δεν εξαιρεί άμεσα την επιλογή κάποιου άλλου αντικειμένου.
  - Χωρίς να γνωρίζουμε τα προηγούμενα αντικείμενα που έχουν επιλεγεί δεν μπορούμε καν να γνωρίζουμε αν υπάρχει χώρος να επιλέξουμε το αντικείμενο i.

Δυναμικός Προγραμματισμός: Απόπειρα

$\mathrm{OPT}(i) =$ μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$.

- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i)$.
  - Η επιλογή του αντικειμένου i δεν εξαιρεί άμεσα την επιλογή κάποιου άλλου αντικειμένου.
  - Χωρίς να γνωρίζουμε τα προηγούμενα αντικείμενα που έχουν επιλεγεί δεν μπορούμε καν να γνωρίζουμε αν υπάρχει χώρος να επιλέξουμε το αντικείμενο i.

Χρειάζεται να ορίσουμε περισσότερα υποπροβλήματα!

Δυναμικός προγραμματισμός: Επιπλέον μεταβλητή

$\mathrm{OPT}(i, w)$ = μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$ με συνολικό βάρος το πολύ w.

Δυναμικός προγραμματισμός: Επιπλέον μεταβλητή

$\mathrm{OPT}(i, w)$ = μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$ με συνολικό βάρος το πολύ w.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.

- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.

Δυναμικός προγραμματισμός: Επιπλέον μεταβλητή

$\mathrm{OPT}(i, w)$ = μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$ με συνολικό βάρος το πολύ w.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$ με βάρος το πολύ w.
- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.

Δυναμικός προγραμματισμός: Επιπλέον μεταβλητή

$\mathrm{OPT}(i, w)$ = μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$ με συνολικό βάρος το πολύ w.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$ με βάρος το πολύ w.
- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Θέτουμε νέο βάρος $w' = w - w_i$.

Δυναμικός προγραμματισμός: Επιπλέον μεταβλητή

$\mathrm{OPT}(i, w)$ = μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$ με συνολικό βάρος το πολύ w.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$ με βάρος το πολύ w.
- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Θέτουμε νέο βάρος $w' = w - w_i$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$ με βάρος το πολύ w'.

Δυναμικός προγραμματισμός: Επιπλέον μεταβλητή

$\mathrm{OPT}(i, w)$ = μέγιστη αξία υποσυνόλων των αντικειμένων $1, 2, \ldots, i$ με συνολικό βάρος το πολύ w.

- Περίπτωση 1: Το i δεν περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$ με βάρος το πολύ w.
- Περίπτωση 2: Το i περιέχεται στη βέλτιστη λύση $\mathrm{OPT}(i, w)$.
  - Θέτουμε νέο βάρος $w' = w - w_i$.
  - Το OPT επιλέγει την καλύτερη λύση ανάμεσα στα αντικείμενα $1, \ldots, i-1$ με βάρος το πολύ w'.

$$\mathrm{OPT}(i, w) = \begin{cases} 0 & \text{εάν } i = 0 \\ \mathrm{OPT}(i-1, w) & \text{εάν } w_i > w \\ \max\{\mathrm{OPT}(i-1, w), v_i + \mathrm{OPT}(i-1, w-w_i)\} & \text{αλλιώς} \end{cases}$$

Αλγόριθμος

Συμπληρώνουμε έναν $n \times W$ πίνακα.

Αλγόριθμος

Συμπληρώνουμε έναν $n \times W$ πίνακα.

Είσοδος: $n,\ w_1,\ w_2,\ \ldots,\ w_n,\ v_1,\ v_2,\ \ldots,\ v_n,\ W$

για $w = 0$ εως W
    $M[0, w] = 0$

για $i = 1$ εως n
    για $w = 0$ έως W
        εάν $w_i > w$
            $M[i, w] = M[i - 1, w]$
        αλλιώς
            $M[i, w] = \max\{v_i + M[i - 1, w - w_i], M[i - 1, w]\}$

Αλγόριθμος

Συμπληρώνουμε έναν $n \times W$ πίνακα.

Είσοδος: $n,\ w_1,\ w_2,\ \ldots,\ w_n,\ v_1,\ v_2,\ \ldots,\ v_n,\ W$

για $w = 0$ εως W
    $M[0, w] = 0$

για $i = 1$ εως n
    για $w = 0$ έως W
        εάν $w_i > w$
            $M[i, w] = M[i - 1, w]$
        αλλιώς
            $M[i, w] = \max\{v_i + M[i - 1, w - w_i], M[i - 1, w]\}$

Ποιά είναι η βέλτιστη λύση του συνολικού προβλήματος;

Παράδειγμα

W + 1 →

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

OPT: { 4, 3 }
value = 22 + 18 = 40

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Χρόνος εκτέλεσης

Τρέχει σε χρόνο $\Theta(n \times W)$.

Χρόνος εκτέλεσης

Τρέχει σε χρόνο $\Theta(n \times W)$.

- Ο χρόνος δεν είναι πολυωνυμικός στο μέγεθος της εισόδου.

Χρόνος εκτέλεσης

Τρέχει σε χρόνο $\Theta(n \times W)$.

- Ο χρόνος δεν είναι πολυωνυμικός στο μέγεθος της εισόδου.
- Είναι ψεύδοπολυωνυμικός.

Χρόνος εκτέλεσης

Τρέχει σε χρόνο $\Theta(n \times W)$.

- Ο χρόνος δεν είναι πολυωνυμικός στο μέγεθος της εισόδου.
- Είναι ψεύδοπολυωνυμικός.
- Το *πρόβλημα απόφασης* του Σακιδίου είναι NP-πλήρες.

Υπάρχει πολυωνυμικός αλγόριθμος ο οποίους βρίσκει μία λύση που έχει αξία το πολύ 0.01% μακριά από το βέλτιστο.

Μέγιστη Κοινή Υπακολουθία

Δεδομένα: δύο ακολουθίες/συμβολοσειρές $x = x_1 x_2 \cdots x_m$ και $y = y_1 y_2 \ldots y_n$

Ζητούμενο: Το μέγιστο μήκος μίας ακολουθίας $z_1 z_2 \cdots z_k$ η οποία είναι ταυτόχρονα υπακολουθία των x και y

Αλγόριθμοι Πολυπλοκότητα

Δομή του Προβλήματος

$\mathrm{OPT}(i, j) =$ μέγιστο μήκος μίας κοινής υπακολουθίας των $x_1 x_2 \cdots x_i$ και $y_1 y_2 \cdots y_j$.

Δομή του Προβλήματος

$\mathrm{OPT}(i,j)$ = μέγιστο μήκος μίας κοινής υπακολουθίας των $x_1 x_2 \cdots x_i$ και $y_1 y_2 \cdots y_j$.

- $x_i = y_j = z_k$.

- $x_i \neq y_j$.

Δομή του Προβλήματος

$\mathrm{OPT}(i,j)$ = μέγιστο μήκος μίας κοινής υπακολουθίας των $x_1 x_2 \cdots x_i$ και $y_1 y_2 \cdots y_j$.

- $x_i = y_j = z_k$.
  Τότε το τελευταίο στοιχείο της μέγιστης κοινής υπακολουθίας είναι το $z_k = x_i = y_j$ και η υπόλοιπη μέγιστη κοινή υπακολουθία είναι υπακολουθία των $x_1 x_2 \cdots x_{i-1}$ και $y_1 y_2 \cdots y_{j-1}$.

- $x_i \neq y_j$.

Δομή του Προβλήματος

$\mathrm{OPT}(i,j)$ = μέγιστο μήκος μίας κοινής υπακολουθίας των $x_1 x_2 \cdots x_i$ και $y_1 y_2 \cdots y_j$.

- $x_i = y_j = z_k$.
  Τότε το τελευταίο στοιχείο της μέγιστης κοινής υπακολουθίας είναι το $z_k = x_i = y_j$ και η υπόλοιπη μέγιστη κοινή υπακολουθία είναι υπακολουθία των $x_1 x_2 \cdots x_{i-1}$ και $y_1 y_2 \cdots y_{j-1}$.

- $x_i \neq y_j$.
  Τότε το πολύ ένα από τα x_i και y_j μπορεί να περιέχεται στη μέγιστη κοινή υπακολουθία και άρα η μέγιστη κοινή υπακολουθία τους είναι είτε υπακολουθία των $x_1 x_2 \cdots x_i$ και $y_1 y_2 \cdots y_{j-1}$ είτε των $x_1 x_2 \cdots x_{i-1}$ και $y_1 y_2 \cdots y_j$

Δομή του Προβλήματος

$\mathrm{OPT}(i,j)$ = μέγιστο μήκος μίας κοινής υπακολουθίας των $x_1x_2\cdots x_i$ και $y_1y_2\cdots y_j$.

- $x_i = y_j = z_k$.
  Τότε το τελευταίο στοιχείο της μέγιστης κοινής υπακολουθίας είναι το $z_k = x_i = y_j$ και η υπόλοιπη μέγιστη κοινή υπακολουθία είναι υπακολουθία των $x_1x_2\cdots x_{i-1}$ και $y_1y_2\cdots y_{j-1}$.

- $x_i \neq y_j$.
  Τότε το πολύ ένα από τα x_i και y_j μπορεί να περιέχεται στη μέγιστη κοινή υπακολουθία και άρα η μέγιστη κοινή υπακολουθία τους είναι είτε υπακολουθία των $x_1x_2\cdots x_i$ και $y_1y_2\cdots y_{j-1}$ είτε των $x_1x_2\cdots x_{i-1}$ και $y_1y_2\cdots y_j$

$$\mathrm{OPT}(i,j) = \begin{cases} 0 & i = 0 \text{ ή } j = 0 \\ 1 + \mathrm{OPT}(i-1, j-1) & i,j > 1 \text{ και } x_i = y_j \\ \max\{\mathrm{OPT}(i-1,j), \mathrm{OPT}(i,j-1)\} & i,j > 1 \text{ και } x_i \neq y_j \end{cases}$$

Αλγόριθμος

ΜΚΥ$(m, n, x_1x_2 \ldots x_m, y_1y_2 \ldots y_n)$
για $j = 0$ έως n
    $M[0, j] = 0$
για $i = 0$ έως m
    $M[i, 0] = 0$

για $i = 1$ εως m
    για $j = 1$ εως n
        εάν $x_i = y_j$
            $M[i, j] = 1 + M[i - 1, j - 1]$
        αλλιώς
            $M[i, j] = \max(M[i, j - 1], M[i - 1, j])$

Αλγόριθμος

ΜΚΥ$(m, n, x_1 x_2 \ldots x_m, y_1 y_2 \ldots y_n)$
για $j = 0$ έως n
    $M[0, j] = 0$
για $i = 0$ έως m
    $M[i, 0] = 0$

για $i = 1$ εως m
    για $j = 1$ εως n
        εάν $x_i = y_j$
            $M[i, j] = 1 + M[i - 1, j - 1]$
        αλλιώς
            $M[i, j] = \max(M[i, j - 1], M[i - 1, j])$

Πολυπλοκότητα:

Αλγόριθμος

ΜΚΥ$(m, n, x_1 x_2 \ldots x_m, y_1 y_2 \ldots y_n)$
για $j = 0$ έως n
    $M[0, j] = 0$
για $i = 0$ έως m
    $M[i, 0] = 0$

για $i = 1$ εως m
    για $j = 1$ εως n
        εάν $x_i = y_j$
            $M[i, j] = 1 + M[i - 1, j - 1]$
        αλλιώς
            $M[i, j] = \max(M[i, j - 1], M[i - 1, j])$

Πολυπλοκότητα: $\Theta(mn)$.