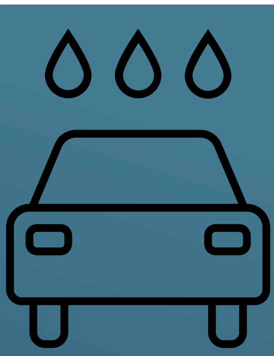


ΓΡΑΜΜΙΚΗ & ΣΥΝΔΥΑΣΤΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

Εργασία #2

Γέροντας Νικόλαος
Α.Μ.: 1092813

Περιεχόμενα:

Άσκηση 1η

Κώδικας 1ης Άσκησης

Θεωρήστε το πρόβλημα γραμμικού προγραμματισμού:

$$\max\ 3x_1 + 11x_2 + 9x_3 - x_4 - 29x_5$$

όταν

$$\begin{aligned} x_2 + x_3 + x_4 - 2x_5 &\leq 4 \\ x_1 - x_2 + x_3 + 2x_4 + x_5 &\geq 0 \\ x_1 + x_2 + x_3 - 3x_5 &\leq 1 \\ x_1 \in \mathbb{R}, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

Ερώτημα α)

Λύστε το πρόβλημα (με κάποιον από τους επιλυτές που αναφέραμε στις διαλέξεις) και περιγράψτε τη βέλτιστη λύση, δηλ. τιμές των μεταβλητών απόφασης και τιμή αντικειμενικής συνάρτησης. Δώστε τις βασικές / μη-βασικές μεταβλητές και τον βέλτιστο βασικό πίνακα. Επίσης ξεχωρίστε τους δεσμευτικούς από τους μη δεσμευτικούς περιορισμούς και μέσω αυτών περιγράψτε τη βέλτιστη κορυφή.

```
--- Άσκηση 1η ---
Ερώτημα α)

- Χρόνος εκτέλεσης: 0.03 δευτερόλεπτα

Βέλτιστη λύση:
x1 = -3.00
x2 = 0.50
x3 = 3.50
x4 = 0.00
x5 = 0.00

Βέλτιστη τιμή της αντικειμενικής συνάρτησης: 28.00

Χαρακτηρισμός των μεταβλητών:
x1: βασική
x2: βασική
x3: βασική
x4: μη-βασική
x5: μη-βασική

Ανάλυση των περιορισμών (δεσμευτικοί / μη δεσμευτικοί):
Constraint_1: 4.00 <= 4.00 -> δεσμευτικός
Constraint_2: 0.00 >= 0.00 -> δεσμευτικός
Constraint_3: 1.00 <= 1.00 -> δεσμευτικός

Άρα, η βέλτιστη κορυφή καθορίζεται από τους περιορισμούς:
Constraint_1, Constraint_2, Constraint_3

Βέλτιστος βασικός πίνακας [B]:
    x1     x2     x3
[  0.00   1.00   1.00 ]
[  1.00  -1.00   1.00 ]
[  1.00   1.00   1.00 ]

Πίνακας [N]:
    x4     x5
[  1.00  -2.00 ]
[  2.00   1.00 ]
[  0.00  -3.00 ]
```

Ερώτημα β)

Επιλέξτε μία βασική και μία μη-βασική μεταβλητή. Περιγράψτε τι θα συμβεί εάν ο συντελεστής της καθεμιάς στην αντικειμενική συνάρτηση (ξεχωριστά) διαταραχθεί κατά ένα ποσό γ. Βρείτε τα διαστήματα ανοχής για τους συγκεκριμένους συντελεστές ώστε να παραμείνει η βέλτιστη λύση στην ίδια κορυφή.

Ισχύει:

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \ \& \ \mathbf{B}^{-1} = \begin{bmatrix} -1 & 0 & 1 \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 1 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \ \& \ \mathbf{N} = \begin{bmatrix} 1 & -2 \\ 2 & 1 \\ 0 & -3 \end{bmatrix} \ \& \ \mathbf{B}^{-1}\mathbf{N} = \begin{bmatrix} -1 & -1 \\ -1 & -2 \\ 2 & 0 \end{bmatrix}$$

Συντελεστές αντικειμενικής: $c_B^T = [3 \quad 11 \quad 9], c_N^T = [-1 \quad -29]$

Βάση της **μεθοδολογίας** που αναφέρεται στο **2° Pdf – σελ. 31/53**, έχουμε:

Βασική

Μία **διαταραχή δ_2 στον συντελεστή του x_2** δεν θα επηρεάσει την βέλτιστη λύση αν:

$$\mathbf{c}_N^T - (\mathbf{c}_B + \delta_2 \mathbf{e}_2)^T \mathbf{B}^{-1}\mathbf{N} \le 0 \Leftrightarrow$$

$$[-1 \quad -29] - [3 \quad 11+\delta_2 \quad 9] \cdot \begin{bmatrix} -1 & -1 \\ -1 & -2 \\ 2 & 0 \end{bmatrix} \le 0 \Leftrightarrow$$

$$[-5+\delta_2 \quad -4+2\delta_2] \le 0 \Leftrightarrow$$

$$\Leftrightarrow \delta_2 \le 5 \ \& \ \boxed{\delta_2 \le 2}$$

Έτσι, αν **ο συντελεστής της x_2 είναι στο διάστημα $(-\infty, 13]$ τότε η βέλτιστη κορυφή δεν αλλάζει**!

μη-Βασική

Μία **διαταραχή δ_4 στον συντελεστή του x_4** δεν θα επηρεάσει την βέλτιστη λύση αν:

$$\mathbf{c}_N^T + \delta_4 \mathbf{e}_4^T - \mathbf{c}_B^T \mathbf{B}^{-1}\mathbf{N} \le 0 \Leftrightarrow$$

$$[-1 \quad -29] + [\delta_4 \quad 0] - [3 \quad 11 \quad 9] \cdot \begin{bmatrix} -1 & -1 \\ -1 & -2 \\ 2 & 0 \end{bmatrix} \le 0 \Leftrightarrow$$

$$[-5+\delta_4 \quad -4] \le 0 \Leftrightarrow$$

$$\Leftrightarrow \boxed{\delta_4 \le 5}$$

Έτσι, αν **ο συντελεστής της x_4 είναι στο διάστημα $(-\infty, 4]$ τότε η βέλτιστη κορυφή δεν αλλάζει**!

Ερώτημα γ)

> Επιλέξτε έναν δεσμευτικό και έναν μη δεσμευτικό περιορισμό και περιγράψτε τι θα συμβεί εάν το δεξιό μέρος του καθενός από αυτούς διαταραχθεί κατά ένα ποσό γ. Βρείτε τα διαστήματα ανοχής που αντιστοιχούν στους δυο αυτούς περιορισμούς.

Ισχύει:

$$B^{-1}b = \begin{bmatrix} -3 \\ 0.5 \\ 3.5 \end{bmatrix}$$

Βάση της **μεθοδολογίας** που αναφέρεται στο **2° Pdf – σελ. 39/53**, έχουμε:

Εισάγω **διαταραχή στον 1° περιορισμό: $\mathbf{x_2 + x_3 + x_4 - 2x_5 \leq 4 + \gamma_1}$**. Έτσι, το διάστημα ανοχής της γ_1 βρίσκεται από τη σχέση:

$$\mathbf{B^{-1}b} + \gamma_1 \mathbf{B^{-1}e_1} \geq 0 \Leftrightarrow$$

$$\begin{bmatrix} -3 \\ 0.5 \\ 3.5 \end{bmatrix} + \gamma_1 \cdot \begin{bmatrix} -1 & 0 & 1 \\ 0 & -\frac{1}{2} & \frac{1}{2} \\ 1 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \geq 0 \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} \gamma_1 \leq -3 \\ 0.5 \geq 0 \\ \gamma_1 \geq -3.5 \end{cases} \Leftrightarrow \boxed{\mathbf{-3.5 \leq \gamma_1 \leq -3}}$$

Δηλαδή, το δεξί μέλος του 1ου (δεσμευτικού) περιορισμού **μπορεί να μεταβληθεί μόνο κατά $\boldsymbol{\gamma_1 \in [-3.5, -3]}$ ώστε να παραμείνει η ίδια βέλτιστη κορυφή**!

Παράλληλα, έστω **διαταραχή μη-δεσμευτικού περιορισμού (προσήμου): $\mathbf{x_2 \geq 0 + \gamma_2}$**. Στην βέλτιστη λύση, ισχύει: $x_2 = 0.5 > 0$ (δηλαδή, ο περιορισμό είναι μη-δεσμευτικός).

Για να παραμείνει εφικτή η ίδια κορυφή, πρέπει:

$$x_2 = 0.5 \geq \gamma_2 \Leftrightarrow \boxed{\mathbf{\gamma_2 \leq 0.5}}$$

Ερώτημα δ)

> Καταστρώστε και λύστε (με κάποιον από τους επιλυτές που αναφέραμε στις διαλέξεις) το δυϊκό του παραπάνω προβλήματος. Ελέγξτε κατά πόσο ισχύουν οι συνθήκες συμπληρωματικής χαλαρότητας για τα δύο αυτά προβλήματα.

Διατύπωση του δυϊκού:

$$
\begin{array}{lrcrr}
 & \textbf{Πρωτεύον} & & & \textbf{Δυϊκό} \\
 & \max Z = 3x_1 + 11x_2 + 9x_3 - x_4 - 29x_5 & \Longrightarrow & & \min Z^* = 4y_1 + y_3 \\
\mathbf{x_1} \in \mathbb{R} & x_2 + x_3 + x_4 - 2x_5 \leq \mathbf{4} & \longrightarrow & \mathbf{y_1 \geq 0} & y_2 + y_3 = \ \ 3 \\
\mathbf{x_2 \geq 0} & x_1 - x_2 + x_3 + 2x_4 + x_5 \geq \mathbf{0} & \longrightarrow & \mathbf{y_2 \leq 0} & y_1 - y_2 + y_3 \geq \ \ 11 \\
\mathbf{x_3 \geq 0} & x_1 + x_2 + x_3 - 3x_5 \leq \mathbf{1} & \longrightarrow & \mathbf{y_3 \geq 0} & y_1 + y_2 + y_3 \geq \ \ 9 \\
\mathbf{x_4 \geq 0} & & \longrightarrow & & y_1 + 2\,y_2 \geq -\ 1 \\
\mathbf{x_5 \geq 0} & & \longrightarrow & & -2\,y_1 + y_2 - 3\,y_3 \geq -29
\end{array}
$$

```
Χρόνος εκτέλεσης: 0.08 δευτερόλεπτα

Βέλτιστη λύση:
  y1 = 6.00
  y2 = -1.00
  y3 = 4.00

  Z* = 28.00
```

Παρατηρούμε πως ικανοποιούνται πλήρως οι συνθήκες του Θεωρήματος 4: Θεώρημα συμπληρωματικής χαλαρότητας!

Άσκηση 2η

Κώδικας 2ης Άσκησης

Θεωρήστε το πρόβλημα γραμμικού προγραμματισμού:

$$\min\ z = x_1 + x_2$$

όταν

$$\begin{aligned} 2x_1 + 3x_2 + x_3 + x_4 &\leq 0 \\ -x_1 + x_2 + 2x_3 + x_4 &= 6 \\ 3x_1 + x_2 + 4x_3 + 2x_4 &\geq 3 \\ x_1 \leq 0,\ x_2,\ x_4 \geq 0,&\ x_3 \in \mathbb{R} \end{aligned}$$

Ερώτημα α)

Καταστρώστε το δυϊκό του παραπάνω προβλήματος. Λύστε το δυϊκό πρόβλημα με τη βοήθεια κατάλληλης βιβλιοθήκης γραμμικού προγραμματισμού της python. Περιγράψτε (αλγεβρικά και γεωμετρικά) τη βέλτιστη λύση, εφ' όσον υπάρχει (**Σημ.** δώστε τιμή αντικειμενικής συνάρτησης, τιμές μεταβλητών, κορυφή, δεσμευτικοί/μη-δεσμευτικοί περιορισμοί.)

Για να καταστρώσουμε το δυϊκό πρόβλημα, θα πρέπει αρχικά να φέρουμε το πρωτεύον στη τυπική του μορφή. Έτσι, καταλήγουμε στον εξής μετασχηματισμό:

$$\boxed{\max Z^* = 6y_2 + 3y_3}$$

$$\boxed{\begin{array}{rcrcrcrl} 2y_1 & - & y_2 & + & 3y_3 & \geq & 1 & \{x_1 \leq 0\} \\ 3y_1 & + & y_2 & + & y_3 & \leq & 1 & \{x_2 \geq 0\} \\ y_1 & + & 2y_2 & + & 4y_3 & = & 0 & \{x_3 \in \mathbb{R}\} \\ y_1 & + & y_2 & + & 2y_3 & \leq & 0 & \{x_4 \geq 0\} \end{array}} \ \& \ \boxed{\begin{array}{l} y_1 \leq 0 \\ y_2 \in \mathbb{R} \\ y_3 \geq 0 \end{array}}$$

{Pdf #2 - Δυϊκή Θεωρία – Ανάλυση Ευαισθησίας | Σελίδες 7/53 & 28/53}

Αποτέλεσμα εκτέλεσης κώδικα:

```
--- Άσκηση 2η ---
Ερώτημα α)

- Δυϊκό πρόβλημα:
Χρόνος εκτέλεσης: 0.03 δευτερόλεπτα

Βέλτιστη λύση:
y1 = 0.00
y2 = -0.40
y3 = 0.20

Βέλτιστη τιμή της αντικειμενικής συνάρτησης: -1.80

Ανάλυση των περιορισμών (δεσμευτικοί / μη δεσμευτικοί):
Constraint_x1: 1.00 >= 1.00 -> δεσμευτικός
Constraint_x2: -0.20 <= 1.00 -> μη δεσμευτικός
Constraint_x3: 0.00 = 0.00 -> δεσμευτικός
Constraint_x4: 0.00 <= 0.00 -> δεσμευτικός

Άρα, η βέλτιστη κορυφή καθορίζεται από τους περιορισμούς:
Constraint_x1, Constraint_x3, Constraint_x4

Επαλήθευση της λύσης του δυϊκού προβλήματος:
Complete! Ο έλεγχος ήταν επιτυχής!
```

Ερώτημα β)

Χρησιμοποιήστε τη λύση του δυϊκού που βρήκατε στο ερώτημα (α) και τη δυϊκή θεωρία για να βρείτε αλγεβρικά τη συμπληρωματική λύση του πρωτεύοντος που σας δόθηκε παραπάνω. Περιγράψτε λεπτομερώς τις διαδικασίες που ακολουθείτε και τις αποφάσεις που παίρνετε. Περιγράψτε κι εδώ τη βέλτιστη λύση, όπως και στο προηγούμενο ερώτημα.

Σύμφωνα με το **Θεώρημα 4: Θεώρημα συμπληρωματικής χαλαρότητας** και δεδομένου ότι ο περιορισμός που αντιστοιχεί στη μεταβλητή x_2 του δυϊκού προβλήματος είναι μη δεσμευτικός, συμπεραίνουμε ότι:

$$\boxed{\mathbf{x_2 = 0}}$$

Ομοίως, ισχύει ότι $\mathbf{y_3 = 0.2 > 0}$, άρα **ο περιορισμός** $\mathbf{3x_1 + x_2 + 4x_3 + 2x_4 \geq 3}$ του πρωτεύοντος ικανοποιείται υποχρεωτικά ως ισότητα (**δεσμευτικός**)!

Παρατηρούμε επιπλέον ότι $\mathbf{y_1 = 0}$, παρότι αποτελεί **βασική μεταβλητή**. Συνεπώς, το δυϊκό πρόβλημα είναι εκφυλισμένο. Άρα, σύμφωνα με το **Θεώρημα/Πίνακα της σελίδας 49/53 του [Pdf #2]**, το πρωτεύον πρόβλημα διαθέτει πολλαπλές βέλτιστες λύσεις!

Σύμφωνα με το **Θεώρημα Ισχυρής Δυϊκότητας [σελ. 14/53]** και δεδομένης της βέλτιστης λύσης του δυϊκού προβλήματος, προκύπτει:

$$c^T x^0 = b^T y^0 \Leftrightarrow x_1 + x_2 = -1.8 \overset{x_2=0}{\Longleftrightarrow} \boxed{\mathbf{x_1 = -1.8}}$$

Έτσι, έχουμε:

$$\begin{cases} 2\cdot(-1.8) + 3\cdot 0 + x_3 + x_4 \leq 0 \\ -(-1.8) + 0 + 2x_3 + x_4 = 6 \\ \mathbf{3\cdot(-1.8) + 0 + 4x_3 + 2x_4 = 3} \end{cases} \Leftrightarrow \begin{cases} x_3 + x_4 \leq 3.6 \\ 2x_3 + x_4 = 4.2 \end{cases} \Leftrightarrow \boxed{\mathbf{x_3 \geq \frac{3}{5}} \;\&\; \mathbf{x_4 = \frac{1}{5}(21 - 10x_3)}}$$

Δηλαδή:

$$\boxed{\begin{array}{lcc} \mathbf{x_1} & = & \mathbf{-1.8} \\ \mathbf{x_2} & = & \mathbf{0} \\ \mathbf{x_3} & \geq & \mathbf{0.6} \\ \mathbf{x_4} & \leq & \mathbf{3} \end{array}}$$

Βέλτιστη λύση πρωτεύοντος

με Z = -1.8

Άσκηση $3^η$

Θεωρήστε το πρόβλημα γραμμικού προγραμματισμού:

$$\max z = 6x_1 + x_2 - x_3 - x_4$$

όταν

$$\begin{aligned} x_1 + 2x_2 + x_3 + x_4 &\le 5 \\ 3x_1 + x_2 - x_3 &\le 8 \\ x_2 + x_3 + x_4 &= 1 \\ x_1, x_2 \in \mathbb{R} \text{ και } x_3, x_4 &\ge 0 \end{aligned}$$

Χρησιμοποιήστε τη δυϊκή θεωρία για να εξετάσετε αν η λύση $\mathbf{x} = (3, -1, 0, 2)$ είναι βέλτιστη για το παραπάνω πρόβλημα γ.π., όμως χωρίς την εφαρμογή του αλγορίθμου Simplex (ή οποιουδήποτε επιλυτή) είτε στο ίδιο το πρόβλημα είτε στο δυϊκό του.

Έλεγχος της λύσης:

$$\boxed{\begin{array}{rll} 3 + 2\cdot(-1) + 0 + 2 = 3 \le 5 & ☑ & [\text{μη δεσμευτικός}] \\ 3\cdot 3 + (-1) - 0 = 8 \le 8 & ☑ & [\text{δεσμευτικός}] \\ -1 + 0 + 2 = 1 = 1 & ☑ & [\text{δεσμευτικός}] \end{array}} \; \& \; \boxed{\begin{array}{ll} x_1 = 3 & \in \mathbb{R} \\ x_2 = -1 & \in \mathbb{R} \\ x_3 = 0 & \ge 0 \\ x_4 = 2 & \ge 0 \end{array}}$$

με $\mathbf{Z = 6 \cdot 3 + (-1) - 0 - 2 = 15}$

Δυϊκό:

$$\min Z = 5 \cdot y_1 + 8 \cdot y_2 + y_3$$

$$\boxed{\begin{array}{lcl} \Pi_1 \le 5 & \rightarrow & y_1 \ge 0 \\ \Pi_2 \le 8 & \rightarrow & y_2 \ge 0 \\ \Pi_3 = 1 & \rightarrow & y_3 \in \mathbb{R} \end{array}} \; \& \; \boxed{\begin{array}{rcrcrcrl} y_1 & + & 3y_2 & & & = & 6 & \{x_1 \in \mathbb{R}\} \\ 2y_1 & + & y_2 & + & y_3 & = & 1 & \{x_2 \in \mathbb{R}\} \\ y_1 & - & y_2 & + & y_3 & \ge & -1 & \{x_3 \ge 0\} \\ y_1 & & & + & y_3 & \ge & -1 & \{x_4 \ge 0\} \end{array}}$$

Με βάση την λογική που εφαρμόστηκε και στην $2^η$ άσκηση, έχουμε:

- ✓ **Ο $1^{ος}$ Περιορισμός του πρωτεύοντος είναι γνήσια ανισότητα, άρα: $y_1 = 0$**
- ✓ Ισχύει $\mathbf{x_4 = 2 > 0}$, οπότε: $\mathbf{y_1 + y_3 = -1}$ [*Θεώρημα 4: Θεώρημα συμπληρωματικής χαλαρότητας*]

Δηλαδή,

$$\mathbf{y_1 + y_3 = -1} \overset{\mathbf{y_1=0}}{\Longrightarrow} \boxed{\mathbf{y_3 = -1}}$$

Επομένως:

$$2y_1 + y_2 + y_3 = 1 \Rightarrow 2 \cdot 0 + y_2 + (-1) = 1 \Leftrightarrow \boxed{\mathbf{y_2 = 2}}$$

Όμως $y_1 - y_2 + y_3 \ge -1 \Leftrightarrow 0 - 2 + (-1) \ge -1 \Leftrightarrow \boxed{-3 \ge -1}$ ✗, δηλαδή ο περιορισμός δεν είναι εφικτός!

Συνεπώς, η λύση $\mathbf{x} = (3, -1, 0, 2)$ δεν είναι βέλτιστη, αφού το δυϊκό δεν είναι εφικτό!

[Θεώρημα Ισχυρής Δυϊκότητας]

Άσκηση $4^{η}$

Κώδικας 4ης Άσκησης

> Ένα μεγάλο εστιατόριο ορίζει τις βάρδιες εργασίας για τους σερβιτόρους του έτσι ώστε ο καθένας να εργάζεται για 5 συνεχόμενες ημέρες της εβδομάδας και να παίρνει 2 συνεχόμενες ημέρες ρεπό. Για παράδειγμα οι σερβιτόροι μιας βάρδιας μπορεί να εργαστούν από Κυριακή έως Πέμπτη και στη συνέχεια να πάρουν ρεπό την Παρασκευή και το Σάββατο. Έστω ότι απαιτείται να βρίσκονται στο εστιατόριο κατ' ελάχιστον 8 σερβιτόροι κάθε Δευτέρα, Τρίτη, Τετάρτη και Πέμπτη, 15 σερβιτόροι κάθε Παρασκευή και Σάββατο και 10 σερβιτόροι κάθε Κυριακή. Ο στόχος του μάνατζερ του εστιατορίου είναι να ικανοποιήσει αυτήν την απαίτηση με το μικρότερο δυνατό πλήθος σερβιτόρων.

Ερώτημα α)

Μοντελοποιήστε το παραπάνω πρόβλημα ως πρόβλημα ακέραιου γραμμικού προγραμματισμού.

- Ορίζω 7 δυαδικές μεταβλητές:

$\mathbf{x_i} \in \mathbb{Z}_0^+$**, όπου i = 1, …, 7 και κάθε** $\mathbf{x_i}$ **αντιστοιχεί σε έναν σερβιτόρο** με συγκεκριμένο τύπο εβδομαδιαίου προγράμματος 5 εργάσιμων ημερών και 2 ημερών ρεπό. Αναλυτικότερα:

	Δευτέρα	Τρίτη	Τετάρτη	Πέμπτη	Παρασκευή	Σάββατο	Κυριακή
x_1	✓	✓	✓	✓	✓	✗	✗
x_2	✗	✓	✓	✓	✓	✓	✗
x_3	✗	✗	✓	✓	✓	✓	✓
x_4	✓	✗	✗	✓	✓	✓	✓
x_5	✓	✓	✗	✗	✓	✓	✓
x_6	✓	✓	✓	✗	✗	✓	✓
x_7	✓	✓	✓	✓	✗	✗	✓

Συνάρτηση κόστους:

$$\boxed{\mathbf{Z = \min\left\{\sum_{i=1}^{7} x_i\right\}}}$$

Περιορισμοί:

$$
\begin{array}{ccccccccccccccccl}
x_1 & & & & & + & x_4 & + & x_5 & + & x_6 & + & x_7 & \geq & 8 & [\mathbf{Δευτέρα}] \\
x_1 & + & x_2 & & & & & + & x_5 & + & x_6 & + & x_7 & \geq & 8 & [\mathbf{Τρίτη}] \\
x_1 & + & x_2 & + & x_3 & & & & & + & x_6 & + & x_7 & \geq & 8 & [\mathbf{Τετάρτη}] \\
x_1 & + & x_2 & + & x_3 & + & x_4 & & & & & + & x_7 & \geq & 8 & [\mathbf{Πέμπτη}] \\
x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & & & & & \geq & 15 & [\mathbf{Παρασκευή}] \\
 & & x_2 & + & x_3 & + & x_4 & + & x_5 & + & x_6 & & & \geq & 15 & [\mathbf{Σάββατο}] \\
 & & & & x_3 & + & x_4 & + & x_5 & + & x_6 & + & x_7 & \geq & 10 & [\mathbf{Κυριακή}]
\end{array}
$$

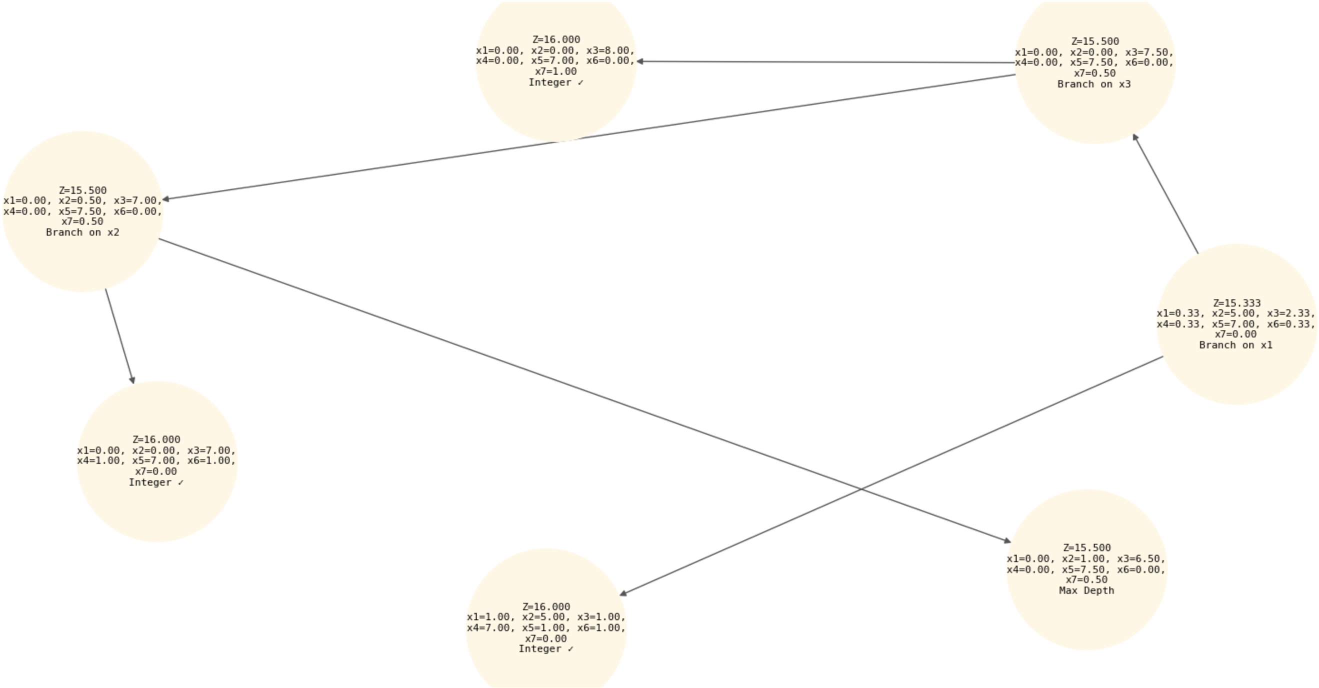
Ερώτημα β)

> Λύστε το πρόβλημα με τη βοήθεια του αλγορίθμου Branch & Bound. Για την επίλυση των προβλημάτων γραμμικού προγραμματισμού στους κόμβους του γράφου χρησιμοποιήσετε κάποιον από τους solvers γραμμικού προγραμματισμού που αναφέρθηκαν στις διαλέξεις μας. Σε περίπτωση πολλαπλών βέλτιστων λύσεων βρείτε τουλάχιστον τρεις και σχολιάστε τις διαφορές τους.

Αποτέλεσμα εκτέλεσης κώδικα:

```
-> Εκκίνηση Branch & Bound

Κόμβος N0 (Βάθος 0) - Προέλευση: ROOT
Τιμή αντικειμενικής συνάρτησης: Z = 15.333
Τιμές μεταβλητών: x1=0.33, x2=5.00, x3=2.33, x4=0.33, x5=7.00, x6=0.33, x7=0.00
- Κατάσταση: Branch στη x1 = 0.333

Κόμβος N1 (Βάθος 1) - Προέλευση: N0
Τιμή αντικειμενικής συνάρτησης: Z = 15.500
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=7.50, x4=0.00, x5=7.50, x6=0.00, x7=0.50
- Κατάσταση: Branch στη x3 = 7.500

Κόμβος N2 (Βάθος 2) - Προέλευση: N1
Τιμή αντικειμενικής συνάρτησης: Z = 15.500
Τιμές μεταβλητών: x1=0.00, x2=0.50, x3=7.00, x4=0.00, x5=7.50, x6=0.00, x7=0.50
- Κατάσταση: Branch στη x2 = 0.500

Κόμβος N3 (Βάθος 3) - Προέλευση: N2
Τιμή αντικειμενικής συνάρτησης: Z = 16.000
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=7.00, x4=1.00, x5=7.00, x6=1.00, x7=0.00
- Κατάσταση: Ακέραια λύση
-> Νέα βέλτιστη ακέραια λύση! GUB = 16.000

Κόμβος N4 (Βάθος 3) - Προέλευση: N2
Τιμή αντικειμενικής συνάρτησης: Z = 15.500
Τιμές μεταβλητών: x1=0.00, x2=1.00, x3=6.50, x4=0.00, x5=7.50, x6=0.00, x7=0.50
- Κατάσταση: Μέγιστο βάθος 3 – ΤΕΡΜΑΤΙΣΜΟΣ

Κόμβος N5 (Βάθος 2) - Προέλευση: N1
Τιμή αντικειμενικής συνάρτησης: Z = 16.000
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=8.00, x4=0.00, x5=7.00, x6=0.00, x7=1.00
- Κατάσταση: Ακέραια λύση

Κόμβος N6 (Βάθος 1) - Προέλευση: N0
Τιμή αντικειμενικής συνάρτησης: Z = 16.000
Τιμές μεταβλητών: x1=1.00, x2=5.00, x3=1.00, x4=7.00, x5=1.00, x6=1.00, x7=0.00
- Κατάσταση: Ακέραια λύση

-> Χρόνος εκτέλεσης: 0.218 δευτερόλεπτα

--- Τελικό Αποτέλεσμα
Βέλτιστη ακέραια λύση: Z = 16
x = {x1 = 0, x2 = 0, x3 = 7, x4 = 1, x5 = 7, x6 = 1, x7 = 0}
```

Γραφική αναπαράσταση του «Δέντρου» που δημιουργείται:



Άσκηση 5η

Κώδικας 5ης Άσκησης

Θεωρήστε το πρόβλημα ακέραιου γραμμικού προγραμματισμού:

$$\max\ 34x_1 + 29x_2 + 2x_3$$

όταν

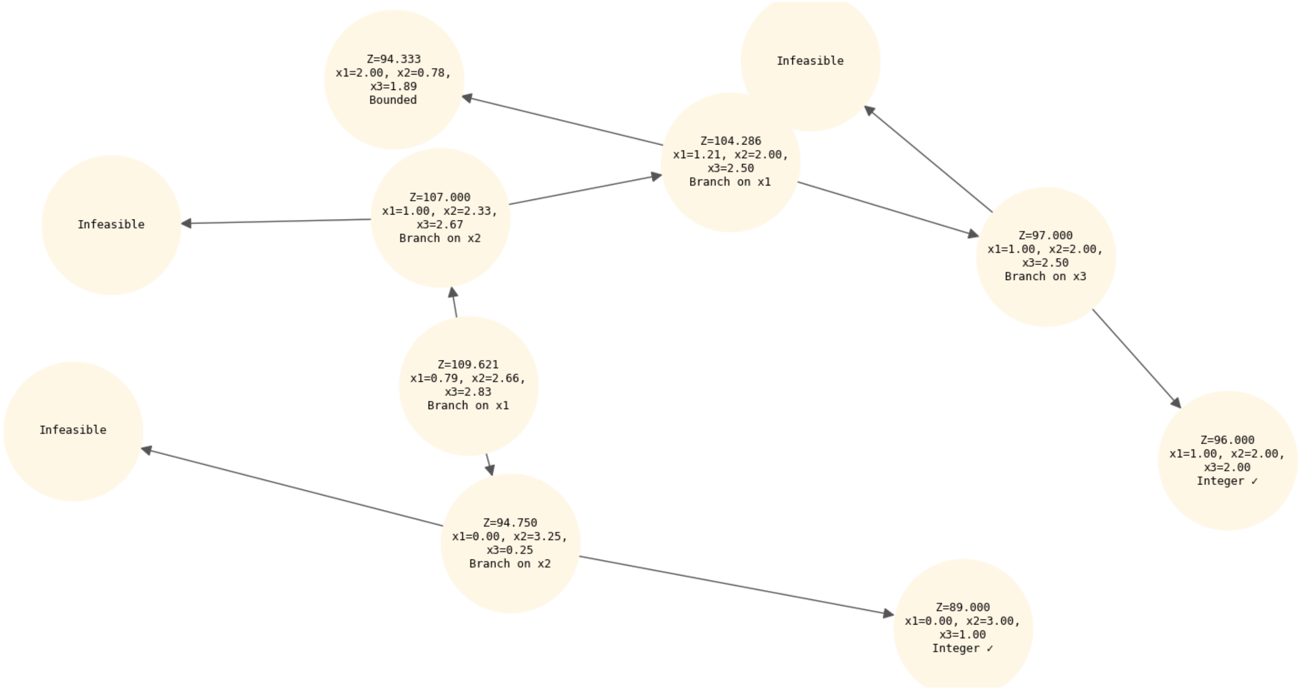
$$7x_1 + 5x_2 - x_3 \le 16$$
$$-x_1 + 3x_2 + x_3 \le 10$$
$$-x_2 + 2x_3 \le 3$$
$$x_1, x_2, x_3 \ge 0 \ \text{και ακέραιοι}$$

Λύστε το με τη βοήθεια του του αλγορίθμου Branch & Bound. Για την επίλυση των προβλημάτων γραμμικού προγραμματισμού στους κόμβους του γράφου χρησιμοποιήσετε κάποιον από τους solvers γραμμικού προγραμματισμού που αναφέρθηκαν στις διαλέξεις μας.

Αποτέλεσμα εκτέλεσης κώδικα:

```
-> Εκκίνηση Branch & Bound

Κόμβος N0 (Βάθος 0) - Προέλευση: ROOT
Τιμή αντικειμενικής συνάρτησης: Z = 109.621
Τιμές μεταβλητών: x1=0.79, x2=2.66, x3=2.83
- Κατάσταση: Branch στη x1 = 0.793

Κόμβος N1 (Βάθος 1) - Προέλευση: N0
Τιμή αντικειμενικής συνάρτησης: Z = 94.750
Τιμές μεταβλητών: x1=0.00, x2=3.25, x3=0.25
- Κατάσταση: Branch στη x2 = 3.250

Κόμβος N2 (Βάθος 2) - Προέλευση: N1
Τιμή αντικειμενικής συνάρτησης: Z = 89.000
Τιμές μεταβλητών: x1=0.00, x2=3.00, x3=1.00
- Κατάσταση: Ακέραια λύση
-> Νέα βέλτιστη ακέραια λύση! GUB = 89.000

Κόμβος N3 (Βάθος 2) - Προέλευση: N1
- Κατάσταση: Μη εφικτό – Απορρίπτεται

Κόμβος N4 (Βάθος 1) - Προέλευση: N0
Τιμή αντικειμενικής συνάρτησης: Z = 107.000
Τιμές μεταβλητών: x1=1.00, x2=2.33, x3=2.67
- Κατάσταση: Branch στη x2 = 2.333

Κόμβος N5 (Βάθος 2) - Προέλευση: N4
Τιμή αντικειμενικής συνάρτησης: Z = 104.286
Τιμές μεταβλητών: x1=1.21, x2=2.00, x3=2.50
- Κατάσταση: Branch στη x1 = 1.214

Κόμβος N6 (Βάθος 3) - Προέλευση: N5
Τιμή αντικειμενικής συνάρτησης: Z = 97.000
Τιμές μεταβλητών: x1=1.00, x2=2.00, x3=2.50
- Κατάσταση: Branch στη x3 = 2.500

Κόμβος N7 (Βάθος 4) - Προέλευση: N6
Τιμή αντικειμενικής συνάρτησης: Z = 96.000
Τιμές μεταβλητών: x1=1.00, x2=2.00, x3=2.00
- Κατάσταση: Ακέραια λύση
-> Νέα βέλτιστη ακέραια λύση! GUB = 96.000

Κόμβος N8 (Βάθος 4) - Προέλευση: N6
- Κατάσταση: Μη εφικτό – Απορρίπτεται

Κόμβος N9 (Βάθος 3) - Προέλευση: N5
Τιμή αντικειμενικής συνάρτησης: Z = 94.333
Τιμές μεταβλητών: x1=2.00, x2=0.78, x3=1.89
- Κατάσταση: Bound (Z = 94.333 < GUB = 96.000)

Κόμβος N10 (Βάθος 2) - Προέλευση: N4
- Κατάσταση: Μη εφικτό – Απορρίπτεται

-> Χρόνος εκτέλεσης: 0.299 δευτερόλεπτα

--- Τελικό Αποτέλεσμα
Βέλτιστη ακέραια λύση: Z = 96
x = {x1 = 1, x2 = 2, x3 = 2}
```

Γραφική αναπαράσταση του «Δέντρου» που δημιουργείται:



Άσκηση 6η

Κώδικας 6ης Άσκησης

> Μια εταιρεία courier θέλει να μεγιστοποιήσει τα συνολικά ημερήσια έσοδά της. Για την παράδοση των δεμάτων, η εταιρεία διαθέτει ένα αυτοκίνητο με όγκο έντεκα (μονάδες όγκου). Υπάρχουν τα εξής δέματα για παράδοση: το δέμα 1 με όγκο δύο, το δέμα 2 με όγκο τρία, το δέμα 3 με όγκο τέσσερα, το δέμα 4 με όγκο έξι, και το δέμα 5 με όγκο οκτώ. Τα έσοδα από την παράδοση των δεμάτων είναι 10, 14, 31, 48, και 60, αντίστοιχα.

Ερώτημα α)

> Μοντελοποιήστε αυτό το πρόβλημα με τη βοήθεια ενός μοντέλου ακέραιου γραμμικού προγραμματισμού. Τι τύπος προβλήματος προκύπτει; Λύστε το με τη μέθοδο branch-and-bound και σχεδιάστε το δέντρο που προκύπτει.

Μοντελοποίηση του προβλήματος:

Έστω $x_i \in \{0, 1\}$, όπου **μονάδα σημαίνει ότι το πακέτο i έχει φορτωθεί στο αυτοκίνητο ενώ 0 όχι**. Επίσης, ορίζω **α_i ως τον όγκο** και **c_i ως τα έσοδα** από το πακέτο i. Έτσι, προκύπτει ο παρακάτω πίνακας:

i	1	2	3	4	5
α_i	2	3	4	6	8
c_i	10	14	31	48	60

Ισχύει/Θέλουμε:

$$\mathbf{Z = \max\{10x_1 + 14x_2 + 31x_3 + 48x_4 + 60x_5\}}$$

$$\mathbf{2x_1 + 3x_2 + 4x_3 + 6x_4 + 8x_5 \leq 11 \; \{μονάδες\ όγκου\}}$$

Δηλαδή, έχει προκύψει ένα **πρόβλημα τύπου 0-1 Knapsack**!

```
-> Εκκίνηση Branch & Bound

Κόμβος N0 (Βάθος 0) - Προέλευση: ROOT
Τιμή αντικειμενικής συνάρτησης: Z = 86.500
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=1.00, x4=1.00, x5=0.12
- Κατάσταση: Branch στη x5 = 0.125

Κόμβος N1 (Βάθος 1) - Προέλευση: N0
Τιμή αντικειμενικής συνάρτησης: Z = 84.000
Τιμές μεταβλητών: x1=0.50, x2=0.00, x3=1.00, x4=1.00, x5=0.00
- Κατάσταση: Branch στη x1 = 0.500

Κόμβος N2 (Βάθος 2) - Προέλευση: N1
Τιμή αντικειμενικής συνάρτησης: Z = 83.667
Τιμές μεταβλητών: x1=0.00, x2=0.33, x3=1.00, x4=1.00, x5=0.00
- Κατάσταση: Branch στη x2 = 0.333

Κόμβος N3 (Βάθος 3) - Προέλευση: N2
Τιμή αντικειμενικής συνάρτησης: Z = 79.000
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=1.00, x4=1.00, x5=0.00
- Κατάσταση: Ακέραια λύση
-> Νέα βέλτιστη ακέραια λύση! GUB = 79.000

Κόμβος N4 (Βάθος 3) - Προέλευση: N2
Τιμή αντικειμενικής συνάρτησης: Z = 77.500
Τιμές μεταβλητών: x1=0.00, x2=1.00, x3=0.50, x4=1.00, x5=0.00
- Κατάσταση: Bound (Z = 77.500 < GUB = 79.000)

Κόμβος N5 (Βάθος 2) - Προέλευση: N1
Τιμή αντικειμενικής συνάρτησης: Z = 81.250
Τιμές μεταβλητών: x1=1.00, x2=0.00, x3=0.75, x4=1.00, x5=0.00
- Κατάσταση: Branch στη x3 = 0.750

Κόμβος N6 (Βάθος 3) - Προέλευση: N5
Τιμή αντικειμενικής συνάρτησης: Z = 72.000
Τιμές μεταβλητών: x1=1.00, x2=1.00, x3=0.00, x4=1.00, x5=0.00
- Κατάσταση: Bound (Z = 72.000 < GUB = 79.000)

Κόμβος N7 (Βάθος 3) - Προέλευση: N5
Τιμή αντικειμενικής συνάρτησης: Z = 81.000
Τιμές μεταβλητών: x1=1.00, x2=0.00, x3=1.00, x4=0.83, x5=0.00
- Κατάσταση: Branch στη x4 = 0.833

Κόμβος N8 (Βάθος 4) - Προέλευση: N7
Τιμή αντικειμενικής συνάρτησης: Z = 55.000
Τιμές μεταβλητών: x1=1.00, x2=1.00, x3=1.00, x4=0.00, x5=0.00
- Κατάσταση: Bound (Z = 55.000 < GUB = 79.000)

Κόμβος N9 (Βάθος 4) - Προέλευση: N7
- Κατάσταση: Μη εφικτό – Απορρίπτεται

Κόμβος N10 (Βάθος 1) - Προέλευση: N0
Τιμή αντικειμενικής συνάρτησης: Z = 84.000
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=0.00, x4=0.50, x5=1.00
- Κατάσταση: Branch στη x4 = 0.500

Κόμβος N11 (Βάθος 2) - Προέλευση: N10
Τιμή αντικειμενικής συνάρτησης: Z = 83.250
Τιμές μεταβλητών: x1=0.00, x2=0.00, x3=0.75, x4=0.00, x5=1.00
- Κατάσταση: Branch στη x3 = 0.750

Κόμβος N12 (Βάθος 3) - Προέλευση: N11
Τιμή αντικειμενικής συνάρτησης: Z = 74.667
Τιμές μεταβλητών: x1=1.00, x2=0.33, x3=0.00, x4=0.00, x5=1.00
- Κατάσταση: Bound (Z = 74.667 < GUB = 79.000)
```

```
Κόμβος N13 (Βάθος 3) - Προέλευση: N11
- Κατάσταση: Μη εφικτό – Απορρίπτεται

Κόμβος N14 (Βάθος 2) - Προέλευση: N10
- Κατάσταση: Μη εφικτό – Απορρίπτεται

--- Τελικό Αποτέλεσμα
Βέλτιστη ακέραια λύση: Z = 79
x = {x1 = 0, x2 = 0, x3 = 1, x4 = 1, x5 = 0}
```

Αναλυτικότερα η λύση:

$$\frac{c_4}{\alpha_4} = 8 \;>\; \frac{c_3}{\alpha_3} = 7.75 \;>\; \frac{\mathbf{c_5}}{\mathbf{\alpha_5}} = \mathbf{7.50} \;>\; \frac{c_1}{\alpha_1} = 5 \;>\; \frac{c_2}{\alpha_2} = 4.67$$

Greedy

Οπότε **επιλέγουμε τα 2 πακέτα με το μεγαλύτερο όφελος [x_4 & x_3]**, για τα οποία ο συνολικός τους όγκος δεν υπερβαίνει την χωρητικότητα του αυτοκινήτου!

Ερώτημα β)

Διατυπώστε το δυϊκό του αρχικού χαλαρωμένου προβλήματος μαζί και τις αντίστοιχες σχέσεις συμπληρωματικής χαλαρότητας (complementary slackness) για τα δύο προβλήματα. Χρησιμοποιήστε τις σχέσεις αυτές και τη βέλτιστη λύση του (χαλαρωμένου) προβλήματος από το (α) για να προσδιορίσετε μία βέλτιστη λύση του δυϊκού προβλήματος.

Βάση του **[Pdf #2] – σελ. 39/53**, έχουμε:

$$\mathbf{2x_1 + 3x_2 + 4x_3 + 6x_4 + 8x_5 \leq 11\ [\pi]}$$

και

$$\mathbf{x_i < 1\ (i = 1, \dots, 5)\ [\mu_i]}$$

Όπου:

$\boldsymbol{\pi \geq 0}$: **δυϊκή μεταβλητή (σκιώδης τιμή) του περιορισμού χωρητικότητας**
$\boldsymbol{\mu_i \geq 0}$: **σκιώδης τιμή ορίου «το πολύ 1» για κάθε x_i**

Δυϊκό:

$$\boxed{\min\left\{11\pi + \sum_{i=1}^{5}\mu_i\right\}} \;\&\; \boxed{\begin{array}{lcr} 2\pi + \mu_1 & \geq & 10 \\ 3\pi + \mu_2 & \geq & 14 \\ 4\pi + \mu_3 & \geq & 31 \\ 6\pi + \mu_4 & \geq & 48 \\ 8\pi + \mu_5 & \geq & 60 \\ \pi, \mu_i \geq 0 & & \end{array}}$$

Επομένως:

Χαλαρωμένο!

$\mathbf{i}$	$\mathbf{x_i}$	$\boldsymbol{\mu_i}$	$\boldsymbol{\alpha_i\pi + \mu_i\ ?\ c_i}$
1	0	0	ελεύθερη
2	0	0	ελεύθερη
3	1	ελεύθερη	$=$
4	1	ελεύθερη	$=$
5	$\mathbf{0.125\ (> 0)}$	0	$=$

Δηλαδή:

$$\begin{cases} 8\pi = 60 \\ 4\pi + \mu_3 = 31 \\ 6\pi + \mu_4 = 48 \end{cases} \Leftrightarrow \boxed{\begin{cases} \boldsymbol{\pi = 7.5} \\ \boldsymbol{\mu_3 = 1} \\ \boldsymbol{\mu_4 = 3} \end{cases}}$$

Τελικά, η δυϊκή λύσης είναι:

$$\begin{bmatrix} \boldsymbol{\pi} \\ \boldsymbol{\mu_1} \\ \boldsymbol{\mu_2} \\ \boldsymbol{\mu_3} \\ \boldsymbol{\mu_4} \\ \boldsymbol{\mu_5} \end{bmatrix} = \begin{bmatrix} \mathbf{7.5} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{3} \\ \mathbf{0} \end{bmatrix}$$

$$\boldsymbol{\text{με } Z^* = 86.5}$$

Ερώτημα γ)

Με τη βοήθεια κάποιου γνωστού solver για ακέραιο γραμμικό προγραμματισμό προσδιορίστε τη βέλτιστη ακέραια λύση του δυϊκού που βρήκατε στο ερώτημα (β). Ελέγξτε αν ισχύουν οι σχέσεις συμπληρωματικής χαλαρότητας για τις βέλτιστες ακέραιες λύσεις του πρωτεύοντος και του αντίστοιχου δυϊκού προβλήματος.

```
Χρόνος εκτέλεσης: 0.028 δευτερόλεπτα

--- Δυϊκό Πρόβλημα (Ακέραιη Επίλυση) ---

Κατάσταση: Optimal
Βέλτιστη τιμή Z = 88.000

Τιμές μεταβλητών:
y1 = 8, y2 = 0, y3 = 0, y4 = 0, y5 = 0, y6 = 0
```

Οι σχέσεις συμπληρωματικής χαλαρότητας παραβιάζονται γιατί η λύση δεν προέρχεται από το LP relaxation, αλλά από ένα ακέραιο δυϊκό που δεν σχετίζεται μέσω ισχυρής δυϊκότητας με το πρωτεύον!

Τμήμα κώδικα

Κώδικας 1ης Άσκησης

```python
# lin_prog_code_HW2_1a.py

from time import time
import numpy as np
import pulp

def define_and_solve_lp() -> pulp.LpProblem:
    model = pulp.LpProblem('LP1', pulp.LpMaximize)

    # Μεταβλητές απόφασης
    x1 = pulp.LpVariable('x1', cat = 'Continuous')
    x2 = pulp.LpVariable('x2', lowBound = 0)
    x3 = pulp.LpVariable('x3', lowBound = 0)
    x4 = pulp.LpVariable('x4', lowBound = 0)
    x5 = pulp.LpVariable('x5', lowBound = 0)

    # Ορισμός της αντικειμενικής συνάρτησης
    model += (3 * x1 + 11 * x2 + 9 * x3 - x4 - 29 * x5, 'Objective')

    # Περιορισμοί
    model += (      x2 + x3 +      x4 - 2 * x5 <= 4, 'Constraint 1')
    model += (x1 - x2 + x3 + 2 * x4 +      x5 >= 0, 'Constraint 2')
    model += (x1 + x2 + x3           - 3 * x5 <= 1, 'Constraint 3')

    # Επίλυση του μοντέλου/προβλήματος
    model.solve(pulp.PULP_CBC_CMD(msg = False))

    return model;

def print_solution(model: pulp.LpProblem) -> tuple:
    # --- Αποτελέσματα
    print('\nΒέλτιστη λύση:')
    var_values = {v.name: v.varValue for v in model.variables()}
    for (name, value) in var_values.items():
        print(f"{name} = {value:.2f}")

    temp = f'{pulp.value(model.objective):.2f}'
    print(f'\nΒέλτιστη τιμή της αντικειμενικής συνάρτησης: {temp}')

    # --- Χαρακτηρισμός μεταβλητών
    basic_vars = []
    print('\nΧαρακτηρισμός των μεταβλητών:')
    for (name, value) in var_values.items():
        status = 'βασική' if abs(value) > 1e-6 else 'μη-βασική'
        print(f"{name}: {status}")
        if status == 'βασική':
            basic_vars.append(name)
```

```
    # --- Ανάλυση περιορισμών
    print('\nΑνάλυση των περιορισμών (δεσμευτικοί / μη δεσμευτικοί):')
    (constraint_status, B_rows, constraint_names) = ({}, [], [])
    for (cname, cons) in model.constraints.items():
        lhs = sum(
            var_values[v.name] * coef for (v, coef) in cons.items()
        )
        rhs = -cons.constant  # Pulp stores as lhs - rhs ≤ 0!!!
        relation = ('<=' if (cons.sense == -1) else \
                    ('>=' if (cons.sense == 1) else '='))

        binding = abs(lhs - rhs) < 1e-6
        status = 'δεσμευτικός' if binding else 'μη δεσμευτικός'
        print(f"{cname}: {lhs:.2f} {relation} {rhs:.2f} -> {status}")
        constraint_status[cname] = status

        if binding:
            B_rows.append([cons.get(v, 0) for v in model.variables() \
                               if v.name in basic_vars])
            constraint_names.append(cname)

    if constraint_names:
        print('\nΆρα, η βέλτιστη κορυφή καθορίζεται από τους περιορισμούς:')
        print(', '.join(constraint_names))

    # --- Πίνακας B
    B_matrix = np.array(B_rows)
    print('\nΒέλτιστος βασικός πίνακας [B]:')
    print_matrix(B_matrix, basic_vars)

    return (basic_vars, constraint_status, B_matrix);

def find_matrix_N(model: pulp.LpProblem,
                  basic_vars: list,
                  constraint_status: dict) -> tuple:
    non_basic_vars = [
        v.name for v in model.variables() if v.name not in basic_vars
    ]
    N_rows = []

    for (cname, cons) in model.constraints.items():
        if constraint_status[cname] == 'δεσμευτικός':
            N_rows.append(
                [cons.get(v, 0) for v in model.variables() \
                 if v.name in non_basic_vars]
            )

    N_matrix = np.array(N_rows)

    return (non_basic_vars, N_matrix);
```

```python
def analyze_perturbation(model: pulp.LpProblem,
                         basic_vars: list,
                         non_basic_vars: list,
                         constraint_status: dict,
                         B_matrix: np.ndarray,
                         N_matrix: np.ndarray) -> None:
    # Πίνακας b από τους δεσμευτικούς περιορισμούς
    b = np.array([
        -model.constraints[name].constant
        for name in constraint_status
        if constraint_status[name] == 'δεσμευτικός'
    ])

    B_inv = np.linalg.inv(B_matrix) # B⁻¹

    # Συντελεστές αντικειμενικής συνάρτησης
    c_all = {v.name: coef for (v, coef) in model.objective.items()}
    c_B = np.array([c_all.get(v, 0) for v in basic_vars])
    c_N = np.array([c_all.get(v, 0) for v in non_basic_vars])

    # --- Ανάλυση συντελεστή βασικής μεταβλητής
    print("\n-> Διαστήματα ανοχής για βασικό συντελεστή:")
    (var, i) = (basic_vars[0], 0) # Επιλέγω την x1 βασική μεταβλητή

    e_k = np.zeros(len(basic_vars))
    e_k[i] = 1
    ekT_Binv = e_k @ B_inv
    ekT_Binv_N = ekT_Binv @ N_matrix

    cN_minus_cB_BinvN = c_N - (c_B @ B_inv @ N_matrix)

    delta_lows = []
    delta_highs = []
    for (ci, ai) in zip(cN_minus_cB_BinvN, ekT_Binv_N):
        if abs(ai) < 1e-8:
            continue;
        bound = ci / ai
        if ai > 0:
            delta_highs.append(bound)
        else:
            delta_lows.append(bound)

    δ_min = max(delta_lows)  if delta_lows  else -np.inf
    δ_max = min(delta_highs) if delta_highs else  np.inf
    (δ_min, δ_max) = (δ_max, δ_min) if δ_min > δ_max else (δ_min, δ_max)

    print(f" - {var}: δ ∈ [{δ_min:.2f}, {δ_max:.2f}]")
```

```python
    # --- Ανάλυση συντελεστή μη-βασικής μεταβλητής
    print("\n-> Διαστήματα ανοχής για μη-βασικό συντελεστή:")
    (var, i) = (non_basic_vars[0], 0) # Επιλέγω την x4 μη-βασική

    delta_lows = []
    delta_highs = []
    for (ci, ei) in zip(cN_minus_cB_BinvN, e_k):
        if abs(ei) < 1e-8:
            continue;
        bound = -ci / ei
        if ei > 0:
            delta_highs.append(bound)
        else:
            delta_lows.append(bound)

    δ_min = max(delta_lows)  if delta_lows  else -np.inf
    δ_max = min(delta_highs) if delta_highs else  np.inf
    (δ_min, δ_max) = (δ_max, δ_min) if δ_min > δ_max else (δ_min, δ_max)

    print(f" - {var}: δ ∈ [{δ_min:.2f}, {δ_max:.2f}]")

    return;

# --- Helpers
def print_matrix(matrix: np.ndarray, vars: list) -> None:
    # Για την καλύτερη εμφάνιση των πινάκων!
    print(4 * ' ' + (5 * ' ').join(vars))
    for row in matrix:
        row_str = '  '.join(f"{val:>5.2f}" for val in row)
        print(f"[ {row_str} ]")

    return;

def main():
    print('\n--- Άσκηση 1η ---')
    print('Ερώτημα α)')

    start = time()
    model = define_and_solve_lp()
    print(f'\n- Χρόνος εκτέλεσης: {time() - start:.2f} δευτερόλεπτα')

    (basic_vars, constraint_status, B_matrix) = print_solution(model)

    (non_basic_vars, N_matrix) = find_matrix_N(model, basic_vars, constraint_status)
    print('\nΠίνακας [N]:')
    print_matrix(N_matrix, non_basic_vars)

    return;

if __name__ == '__main__':
    main()
```

```python
# lin_prog_code_HW2_1d.py

from time import time
import pulp

def define_dual_lp():
    dual = pulp.LpProblem('DualProblem', pulp.LpMinimize)

    # Δυϊκές μεταβλητές
    y1 = pulp.LpVariable('y1', lowBound = 0) # ≤-περιορισμός
    y2 = pulp.LpVariable('y2', upBound  = 0) # ≥-περιορισμός
    y3 = pulp.LpVariable('y3', lowBound = 0) # ≤-περιορισμός

    # Αντικειμενική
    dual += (4 * y1 + y3, 'W')

    # Περιορισμοί στήλης
    dual += (             y2 +     y3 ==   3, 'Constraint 1')
    dual += (     y1 -    y2 +     y3 >=  11, 'Constraint 2')
    dual += (     y1 +    y2 +     y3 >=   9, 'Constraint 3')
    dual += (     y1 + 2 * y2          >=  -1, 'Constraint 4')
    dual += (-2 * y1 +    y2 - 3 * y3 >= -29, 'Constraint 5')

    dual.solve(pulp.PULP_CBC_CMD(msg = False))

    return dual;

if __name__ == '__main__':
    start = time()
    model = define_dual_lp()
    print(f'\nΧρόνος εκτέλεσης: {time() - start:.2f} δευτερόλεπτα\n')

    print('Βέλτιστη λύση:')
    for v in model.variables():
        print(f'  {v.name} = {v.varValue:.2f}')
    print(f'\n  Z* = {pulp.value(model.objective):.2f}')
```

Κώδικας $2^{ης}$ Άσκησης

```python
# lin_prog_code_HW2_2a.py

from time import time
import numpy as np
import pulp

def define_and_solve_lp() -> pulp.LpProblem:
    model = pulp.LpProblem('LP1', pulp.LpMinimize)

    # Μεταβλητές απόφασης
    x1 = pulp.LpVariable('x1', upBound = 0)
    x2 = pulp.LpVariable('x2', lowBound = 0)
    x3 = pulp.LpVariable('x3', cat = 'Continuous')
    x4 = pulp.LpVariable('x4', lowBound = 0)

    # Ορισμός της αντικειμενικής συνάρτησης
    model += (x1 + x2, 'Objective Function')

    # Περιορισμοί
    model += (2 * x1 + 3 * x2 +     x3 +     x4 <= 0, 'Constraint 1')
    model += (   -x1 +     x2 + 2 * x3 +     x4 == 6, 'Constraint 2')
    model += (3 * x1 +     x2 + 4 * x3 + 2 * x4 >= 3, 'Constraint 3')

    # Επίλυση του μοντέλου/προβλήματος
    model.solve(pulp.PULP_CBC_CMD(msg = False))

    return model;

def define_dual_lp() -> pulp.LpProblem:
    dual = pulp.LpProblem('Dual_LP', pulp.LpMaximize)

    # Δυϊκές μεταβλητές
    #   Constraint 1 (<=) -> y1 ≤ 0
    #   Constraint 2 ( =) -> y2 free
    #   Constraint 3 (>=) -> y3 ≥ 0
    y1 = pulp.LpVariable('y1', upBound = 0)
    y2 = pulp.LpVariable('y2', cat = 'Continuous')
    y3 = pulp.LpVariable('y3', lowBound = 0)

    # Αντικειμενική συνάρτηση
    dual += (6 * y2 + 3 * y3, 'Dual_Objective')

    # Περιορισμοί
    dual += (2 * y1 -     y2 + 3 * y3 >= 1, 'Constraint x1')
    dual += (3 * y1 +     y2 +     y3 <= 1, 'Constraint x2')
    dual += (    y1 + 2 * y2 + 4 * y3 == 0, 'Constraint x3')
    dual += (    y1 +     y2 + 2 * y3 <= 0, 'Constraint x4')
```

```python
    dual.solve(pulp.PULP_CBC_CMD(msg = False))

    return dual;

def print_solution_dual(model: pulp.LpProblem) -> dict:
    # --- Αποτελέσματα
    print('\nΒέλτιστη λύση:')
    var_values = {v.name: v.varValue for v in model.variables()}
    for (name, value) in var_values.items():
        print(f'{name} = {value:.2f}')

    temp = f'{pulp.value(model.objective):.2f}'
    print(f'\nΒέλτιστη τιμή της αντικειμενικής συνάρτησης: {temp}')

    # --- Χαρακτηρισμός μεταβλητών
    basic_vars = []
    for (name, value) in var_values.items():
        status = 'βασική' if abs(value) > 1e-6 else 'μη-βασική'
        if status == 'βασική':
            basic_vars.append(name)

    # --- Ανάλυση περιορισμών
    print('\nΑνάλυση των περιορισμών (δεσμευτικοί / μη δεσμευτικοί):')
    (constraint_status, B_rows, constraint_names) = ({}, [], [])
    for (cname, cons) in model.constraints.items():
        lhs = sum(
            var_values[v.name] * coef for (v, coef) in cons.items()
        )
        rhs = -cons.constant  # Pulp stores as lhs - rhs ≤ 0!!!
        relation = ('<=' if (cons.sense == -1) else \
                    ('>=' if (cons.sense == 1) else '='))

        binding = abs(lhs - rhs) < 1e-6
        status = 'δεσμευτικός' if binding else 'μη δεσμευτικός'
        print(f'{cname}: {lhs:.2f} {relation} {rhs:.2f} -> {status}')
        constraint_status[cname] = status

        if binding:
            B_rows.append([cons.get(v, 0) for v in model.variables() \
                               if v.name in basic_vars])
            constraint_names.append(cname)

    if constraint_names:
        print('\nΆρα, η βέλτιστη κορυφή καθορίζεται από τους περιορισμούς:')
        print(', '.join(constraint_names))

    return var_values;

def main():
    print('\n--- Άσκηση 2η ---')
    print('Ερώτημα α)')
```

```python
    # print('\n- Πρωτεύον πρόβλημα:')
    # start = time()
    model = define_and_solve_lp()
    # print(f'Χρόνος εκτέλεσης: {time() - start:.2f} δευτερόλεπτα')
    # print_solution(model)

    print('\n- Δυϊκό πρόβλημα:')
    start = time()
    dual = define_dual_lp()
    print(f'Χρόνος εκτέλεσης: {time() - start:.2f} δευτερόλεπτα')
    dual_var_solution = print_solution_dual(dual)

    # --- Έλεγχος της λύσης του δυϊκού προβλήματος!
    print('\nΕπαλήθευση της λύσης του δυϊκού προβλήματος:')
    assert np.isclose(
        pulp.value(model.objective),
        pulp.value(dual.objective),
        atol = 1e-5
    ), f'Οι τιμές των αντικειμενικών συναρτήσεων δεν είναι ίσες!'
    dual_map = {
        'y1': 'Constraint_1',
        'y2': 'Constraint_2',
        'y3': 'Constraint_3'
    }
    for (y_name, cname) in dual_map.items():
        expected = model.constraints[cname].pi
        actual   = dual_var_solution[y_name]
        assert np.isclose(
            actual,
            expected,
            atol = 1e-5
        ), f'{y_name} = {actual:.2f} != {expected:.2f} = {cname}'
    print('Complete! Ο έλεγχος ήταν επιτυχής!')

    return;

if __name__ == '__main__':
    main()
```

Κώδικας 4^{ης} Άσκησης

```python
# lin_prog_code_HW2_4b.py

from time import time
import pulp

# Global μεταβλητή κατάστασης
best_Z_int          = float('inf') # Βέλτιστη τιμή αντικειμενικής
                                  # (Global Upper Bound - GUB)
best_x_int_solution = None        # Βέλτιστη ακέραια λύση (dictionary)
node_id_counter     = 0           # Αναγνωριστικό κόμβων για debug!

solver = pulp.PULP_CBC_CMD(msg = False) # Solver χωρίς output

# Συνάρτηση επίλυσης LP κόμβου
def solve_lp_node(branch_constraints):
    model = pulp.LpProblem('Waiter_Scheduling_LP_Node', pulp.LpMinimize)

    # Μεταβλητές απόφασης (x1 έως x7)
    x = [pulp.LpVariable(
        f'x{i}', lowBound = 0, cat = 'Continuous') \
            for i in range(1, 8)
    ]
    (x1, x2, x3, x4, x5, x6, x7) = x # Για ευκολία αναφοράς

    # Αντικειμενική συνάρτηση: ελαχιστοποίηση του πλήθους σερβιτόρων
    model += pulp.lpSum(x)

    # Αρχικοί περιορισμοί (μία γραμμή ανά ημέρα)
    model += x1           + x4 + x5 + x6 + x7 >= 8  # Δευτέρα
    model += x1 + x2           + x5 + x6 + x7 >= 8  # Τρίτη
    model += x1 + x2 + x3           + x6 + x7 >= 8  # Τετάρτη
    model += x1 + x2 + x3 + x4           + x7 >= 8  # Πέμπτη
    model += x1 + x2 + x3 + x4 + x5           >= 15 # Παρασκευή
    model +=      x2 + x3 + x4 + x5 + x6      >= 15 # Σάββατο
    model +=           x3 + x4 + x5 + x6 + x7 >= 10 # Κυριακή

    # Εφαρμογή των περιορισμών διακλάδωσης
    for (idx, op, val) in branch_constraints:
        if op == '<=':
            model += x[idx - 1] <= val
        else:
            model += x[idx - 1] >= val

    # Επίλυση του LP
    model.solve(solver)
```

```python
    if pulp.LpStatus[model.status] == 'Optimal':
        Z = pulp.value(model.objective)
        x_sol = {v.name: v.varValue for v in model.variables()}
        return (Z, x_sol, True);
    else:
        return (float('inf'), None, False);

# Αναδρομική συνάρτηση Branch & Bound!
def branch_and_bound_recursive(
    parent_id, constraints, depth, max_depth = 5
):
    global best_Z_int, best_x_int_solution, node_id_counter

    node_id = f'N{node_id_counter}'
    node_id_counter += 1

    (Z, x_sol, feasible) = solve_lp_node(constraints)

    print(f'\nΚόμβος {node_id} (Βάθος {depth})', end = ' - ')
    print(f'Προέλευση: {parent_id or 'ROOT'}')
    if not feasible:
        print('- Κατάσταση: Μη εφικτό — Απορρίπτεται')
        return;

    print(f'Τιμή αντικειμενικής συνάρτησης: Z = {Z:.3f}')
    print('Τιμές μεταβλητών: ' + \
        ', '.join(f'{k}={v:.2f}' \
            for (k, v) in sorted(
                x_sol.items(), key = lambda x: int(x[0][1:])
            )
        )
    )

    if Z > best_Z_int:
        print(f'- Κατάσταση: Bound (Z = {Z:.3f} > GUB = {best_Z_int:.3f})')
        return;

    (var_idx, frac_val) = get_fractional_variable_info(x_sol)

    if var_idx is None:
        print('- Κατάσταση: Ακέραια λύση')
        if Z < best_Z_int:
            best_Z_int = Z
            best_x_int_solution = x_sol
            print(f'-> Νέα βέλτιστη ακέραια λύση! GUB = {best_Z_int:.3f}')
        return;

    if depth >= max_depth:
        print(f'- Κατάσταση: Μέγιστο βάθος {max_depth} — ΤΕΡΜΑΤΙΣΜΟΣ')
        return;
```

```python
    print(f'- Κατάσταση: Branch στη x{var_idx} = {frac_val:.3f}')

    floor_val = int(frac_val)
    ceil_val = floor_val + 1

    # Διακλάδωση 1: x ≤ floor
    branch_and_bound_recursive(
        node_id,
        constraints + [(var_idx, '<=', floor_val)],
        depth + 1,
        max_depth
    )

    # Διακλάδωση 2: x ≥ ceil
    branch_and_bound_recursive(
        node_id,
        constraints + [(var_idx, '>=', ceil_val)],
        depth + 1,
        max_depth
    )

    return;

# --- Helpers ---
def get_fractional_variable_info(x_dict):
    # Συνάρτηση εύρεσης 1ης μη ακέραιας μεταβλητής
    for i in range(1, 8):
        var = f'x{i}'
        val = x_dict.get(var, 0.)
        if abs(val - round(val)) > 1e-5:
            return (i, val);

    return (None, None);

def main():
    global best_Z_int, best_x_int_solution, node_id_counter

    best_Z_int          = float('inf')
    best_x_int_solution = None
    node_id_counter     = 0

    print('-> Εκκίνηση Branch & Bound')
    start = time()
    branch_and_bound_recursive(None, [], 0, max_depth = 3)
    print(f'\n-> Χρόνος εκτέλεσης: {time() - start:.3f} δευτερόλεπτα')
```

```python
    print('\n--- Τελικό Αποτέλεσμα')
    if best_x_int_solution:
        sorted_x = sorted(
            best_x_int_solution.items(), key = lambda x: int(x[0][1:])
        )
        print(f'Βέλτιστη ακέραια λύση: Z = {best_Z_int:.0f}')
        print('x = {' + \
            ', '.join(f'{k} = {v:.0f}' for (k, v) in sorted_x) + '}'
        )
    else:
        print('Δεν βρέθηκε ακέραια λύση.')

    return;

if __name__ == '__main__':
    main()
```

Κώδικας 5ης Άσκησης

```python
# lin_prog_code_HW2_5.py

from time import time
import pulp

# Global μεταβλητή κατάστασης
best_Z_int          = float('-inf') # Βέλτιστη τιμή αντικειμενικής
                                    # (Global Upper Bound - GUB)
best_x_int_solution = None          # Βέλτιστη ακέραια λύση (dictionary)
node_id_counter     = 0             # Αναγνωριστικό κόμβων για debug!

solver = pulp.PULP_CBC_CMD(msg = False) # Solver χωρίς output

# Συνάρτηση επίλυσης LP κόμβου
def solve_lp_node(branch_constraints):
    model = pulp.LpProblem('BranchAndBoundLP', pulp.LpMaximize)

    # Μεταβλητές απόφασης
    x = [pulp.LpVariable(
        f'x{i}', lowBound = 0, cat = 'Continuous') \
            for i in range(1, 4)
    ]
    (x1, x2, x3) = x # Για ευκολία αναφοράς

    # Αντικειμενική συνάρτηση:
    model += 34 * x1 + 29 * x2 + 2 * x3

    # Περιορισμοί
    model += 7 * x1 + 5 * x2 -     x3 <= 16
    model +=    -x1 + 3 * x2 +     x3 <= 10
    model +=              -x2 + 2 * x3 <= 3

    # Εφαρμογή των περιορισμών διακλάδωσης
    for (idx, op, val) in branch_constraints:
        if op == '<=':
            model += x[idx - 1] <= val
        else:
            model += x[idx - 1] >= val

    # Επίλυση του LP
    model.solve(solver)

    if pulp.LpStatus[model.status] == 'Optimal':
        Z = pulp.value(model.objective)
        x_sol = {v.name: v.varValue for v in model.variables()}
        return (Z, x_sol, True);
    else:
        return (float('-inf'), None, False);
```

```python
# Αναδρομική συνάρτηση Branch & Bound!
def branch_and_bound_recursive(
    parent_id, constraints, depth, max_depth = 5
):
    global best_Z_int, best_x_int_solution, node_id_counter

    node_id = f'N{node_id_counter}'
    node_id_counter += 1

    (Z, x_sol, feasible) = solve_lp_node(constraints)

    print(f'\nΚόμβος {node_id} (Βάθος {depth})', end = ' - ')
    print(f'Προέλευση: {parent_id or 'ROOT'}')
    if not feasible:
        print('- Κατάσταση: Μη εφικτό – Απορρίπτεται')
        return;

    print(f'Τιμή αντικειμενικής συνάρτησης: Z = {Z:.3f}')
    print('Τιμές μεταβλητών: ' + \
        ', '.join(f'{k}={v:.2f}' \
            for (k, v) in sorted(
                x_sol.items(), key = lambda x: int(x[0][1:])
            )
        )
    )

    if Z < best_Z_int:
        print(f'- Κατάσταση: Bound (Z = {Z:.3f} < GUB = {best_Z_int:.3f})')
        return;

    (var_idx, frac_val) = get_fractional_variable_info(x_sol)

    if var_idx is None:
        print('- Κατάσταση: Ακέραια λύση')
        if Z > best_Z_int:
            best_Z_int = Z
            best_x_int_solution = x_sol
            print(f'-> Νέα βέλτιστη ακέραια λύση! GUB = {best_Z_int:.3f}')
        return;

    if depth >= max_depth:
        print(f'- Κατάσταση: Μέγιστο βάθος {max_depth} – ΤΕΡΜΑΤΙΣΜΟΣ')
        return;

    print(f'- Κατάσταση: Branch στη x{var_idx} = {frac_val:.3f}')

    floor_val = int(frac_val)
    ceil_val = floor_val + 1
```

```python
    # Διακλάδωση 1: x ≤ floor
    branch_and_bound_recursive(
        node_id,
        constraints + [(var_idx, '<=', floor_val)],
        depth + 1,
        max_depth
    )

    # Διακλάδωση 2: x ≥ ceil
    branch_and_bound_recursive(
        node_id,
        constraints + [(var_idx, '>=', ceil_val)],
        depth + 1,
        max_depth
    )

    return;

# --- Helpers ---
def get_fractional_variable_info(x_dict):
    # Συνάρτηση εύρεσης 1ης μη ακέραιας μεταβλητής
    for i in range(1, 4):
        var = f'x{i}'
        val = x_dict.get(var, 0.)
        if abs(val - round(val)) > 1e-5:
            return (i, val);

    return (None, None);

def main():
    global best_Z_int, best_x_int_solution, node_id_counter

    best_Z_int          = float('-inf')
    best_x_int_solution = None
    node_id_counter     = 0

    print('-> Εκκίνηση Branch & Bound')
    start = time()
    branch_and_bound_recursive(None, [], 0, max_depth = 20)
    print(f'\n-> Χρόνος εκτέλεσης: {time() - start:.3f} δευτερόλεπτα')
```

```python
    print('\n--- Τελικό Αποτέλεσμα')
    if best_x_int_solution:
        sorted_x = sorted(
            best_x_int_solution.items(), key = lambda x: int(x[0][1:])
        )
        print(f'Βέλτιστη ακέραια λύση: Z = {best_Z_int:.0f}')
        print('x = {' + \
            ', '.join(f'{k} = {v:.0f}' for (k, v) in sorted_x) + '}'
        )
    else:
        print('Δεν βρέθηκε ακέραια λύση.')

    return;

if __name__ == '__main__':
    main()
```

Κώδικας 6ης Άσκησης

```python
# lin_prog_code_HW2_6a.py

import pulp

# Global μεταβλητή κατάστασης
best_Z_int          = float('-inf') # Βέλτιστη τιμή αντικειμενικής
                                  # (Global Upper Bound - GUB)
best_x_int_solution = None        # Βέλτιστη ακέραια λύση (dictionary)
node_id_counter     = 0           # Αναγνωριστικό κόμβων για debug!

solver = pulp.PULP_CBC_CMD(msg = False) # Solver χωρίς output

# Συνάρτηση επίλυσης LP κόμβου
def solve_lp_node(branch_constraints):
    model = pulp.LpProblem('BranchAndBoundLP', pulp.LpMaximize)

    # Δυαδικές μεταβλητές του LP
    x1 = pulp.LpVariable('x1', lowBound = 0, upBound = 1, cat = 'Continuous')
    x2 = pulp.LpVariable('x2', lowBound = 0, upBound = 1, cat = 'Continuous')
    x3 = pulp.LpVariable('x3', lowBound = 0, upBound = 1, cat = 'Continuous')
    x4 = pulp.LpVariable('x4', lowBound = 0, upBound = 1, cat = 'Continuous')
    x5 = pulp.LpVariable('x5', lowBound = 0, upBound = 1, cat = 'Continuous')
    x = [x1, x2, x3, x4, x5] # Λίστα μεταβλητών για εύκολη αναφορά

    # Αντικειμενική συνάρτηση:
    model += (
        10 * x1 + 14 * x2 + 31 * x3 + 48 * x4 + 60 * x5, 'Objective'
    )

    # Περιορισμοί
    model += 2 * x1 + 3 * x2 + 4 * x3 + 6 * x4 + 8 * x5 <= 11

    # Εφαρμογή των περιορισμών διακλάδωσης
    for (idx, op, val) in branch_constraints:
        if op == '<=':
            model += x[idx - 1] <= val
        else:
            model += x[idx - 1] >= val

    # Επίλυση του LP
    model.solve(solver)

    if pulp.LpStatus[model.status] == 'Optimal':
        Z = pulp.value(model.objective)
        x_sol = {v.name: v.varValue for v in model.variables()}
        return (Z, x_sol, True);
    else:
        return (float('-inf'), None, False);
```

```python
# Αναδρομική συνάρτηση Branch & Bound!
def branch_and_bound_recursive(
    parent_id, constraints, depth, max_depth = 5
):
    global best_Z_int, best_x_int_solution, node_id_counter

    node_id = f'N{node_id_counter}'
    node_id_counter += 1

    (Z, x_sol, feasible) = solve_lp_node(constraints)

    print(f'\nΚόμβος {node_id} (Βάθος {depth})', end = ' - ')
    print(f'Προέλευση: {parent_id or 'ROOT'}')
    if not feasible:
        print('- Κατάσταση: Μη εφικτό — Απορρίπτεται')
        return;

    print(f'Τιμή αντικειμενικής συνάρτησης: Z = {Z:.3f}')
    print('Τιμές μεταβλητών: ' + \
        ', '.join(f'{k}={v:.2f}' \
            for (k, v) in sorted(
                x_sol.items(), key = lambda x: int(x[0][1:])
            )
        )
    )

    if Z < best_Z_int:
        print(f'- Κατάσταση: Bound (Z = {Z:.3f} < GUB = {best_Z_int:.3f})')
        return;

    (var_idx, frac_val) = get_fractional_variable_info(x_sol)

    if var_idx is None:
        print('- Κατάσταση: Ακέραια λύση')
        if Z > best_Z_int:
            best_Z_int = Z
            best_x_int_solution = x_sol
            print(f'-> Νέα βέλτιστη ακέραια λύση! GUB = {best_Z_int:.3f}')
        return;

    if depth >= max_depth:
        print(f'- Κατάσταση: Μέγιστο βάθος {max_depth} — ΤΕΡΜΑΤΙΣΜΟΣ')
        return;

    print(f'- Κατάσταση: Branch στη x{var_idx} = {frac_val:.3f}')

    floor_val = int(frac_val)
    ceil_val = floor_val + 1
```

```python
    # Διακλάδωση 1: x ≤ floor
    branch_and_bound_recursive(
        node_id,
        constraints + [(var_idx, '<=', floor_val)],
        depth + 1,
        max_depth
    )

    # Διακλάδωση 2: x ≥ ceil
    branch_and_bound_recursive(
        node_id,
        constraints + [(var_idx, '>=', ceil_val)],
        depth + 1,
        max_depth
    )

    return;

# --- Helpers ---
def get_fractional_variable_info(x_dict):
    # Συνάρτηση εύρεσης 1ης μη ακέραιας μεταβλητής
    for i in range(1, 6):
        var = f'x{i}'
        val = x_dict.get(var, 0.)
        if abs(val - round(val)) > 1e-5:
            return (i, val);

    return (None, None);

def main():
    global best_Z_int, best_x_int_solution, node_id_counter

    best_Z_int          = float('-inf')
    best_x_int_solution = None
    node_id_counter     = 0

    print('-> Εκκίνηση Branch & Bound')
    branch_and_bound_recursive(None, [], 0, max_depth = 20)

    print('\n--- Τελικό Αποτέλεσμα')
    if best_x_int_solution:
        sorted_x = sorted(
            best_x_int_solution.items(), key = lambda x: int(x[0][1:])
        )
        print(f'Βέλτιστη ακέραια λύση: Z = {best_Z_int:.0f}')
        print('x = {' + \
            ', '.join(f'{k} = {v:.0f}' for (k, v) in sorted_x) + '}'
        )
    else:
        print('Δεν βρέθηκε ακέραια λύση.')

    return;
```

```python
if __name__ == '__main__':
    main()
```

```python
# lin_prog_code_HW2_6c.py

from time import time
import pulp

def define_model_and_solve():
    # --- Μοντέλο Δυϊκού LP (με ακέραιες μεταβλητές) ---
    model = pulp.LpProblem('Dual_Knapsack_LP', pulp.LpMinimize)

    # Μεταβλητές: y1 για τον περιορισμό χωρητικότητας, y2-y6 για upper bounds των x1-x5
    y = [pulp.LpVariable(f'y{i}', lowBound = 0, cat = 'Integer') for i in range(1, 7)]
    (y1, y2, y3, y4, y5, y6) = y # Για ευκολία αναφοράς

    # Αντικειμενική συνάρτηση:
    model += (11 * y1 + y2 + y3 + y4 + y5 + y6, 'Objective')

    # Περιορισμοί (ένας για κάθε x_i)
    model += 2 * y1 + y2 >= 10  # x1
    model += 3 * y1 + y3 >= 14  # x2
    model += 4 * y1 + y4 >= 31  # x3
    model += 6 * y1 + y5 >= 48  # x4
    model += 8 * y1 + y6 >= 60  # x5

    # --- Επίλυση ---
    solver = pulp.PULP_CBC_CMD(msg = False)
    model.solve(solver)

    return (model, y);

def main():
    start = time()
    (model, y) = define_model_and_solve()
    print(f'Χρόνος εκτέλεσης: {time() - start:.3f} δευτερόλεπτα')

    # --- Αποτελέσματα ---
    print('\n--- Δυϊκό Πρόβλημα (Ακέραιη Επίλυση) ---')
    print('\nΚατάσταση:', pulp.LpStatus[model.status])
    print(f'Βέλτιστη τιμή Z = {pulp.value(model.objective):.3f}')

    print('\nΤιμές μεταβλητών:')
    print(', '.join([f'{var.name} = {var.varValue:.0f}' for var in model.variables()]))

    return;

if __name__ == "__main__":
    main()
```

Πρόσθετα βοηθητικά αρχεία κώδικα (helper 4b & 5):

```python
# Γραφική παράσταση για το β ερώτημα της άσκησης 4
# Hardcoded βάση του αποτελέσματος του αρχείου: lin_prog_code_HW2_4b.py

import matplotlib.pyplot as plt
import networkx as nx

# Detailed nodes for the Branch & Bound tree
detailed_nodes = [
    ('N0', 'Z=15.333\nx1=0.33, x2=5.00, x3=2.33,\nx4=0.33, x5=7.00,
x6=0.33,\nx7=0.00\nBranch on x1', None),
    ('N1', 'Z=15.500\nx1=0.00, x2=0.00, x3=7.50,\nx4=0.00, x5=7.50,
x6=0.00,\nx7=0.50\nBranch on x3', 'N0'),
    ('N2', 'Z=15.500\nx1=0.00, x2=0.50, x3=7.00,\nx4=0.00, x5=7.50,
x6=0.00,\nx7=0.50\nBranch on x2', 'N1'),
    ('N3', 'Z=16.000\nx1=0.00, x2=0.00, x3=7.00,\nx4=1.00, x5=7.00,
x6=1.00,\nx7=0.00\nInteger ✓',    'N2'),
    ('N4', 'Z=15.500\nx1=0.00, x2=1.00, x3=6.50,\nx4=0.00, x5=7.50, x6=0.00,\nx7=0.50\nMax
Depth',    'N2'),
    ('N5', 'Z=16.000\nx1=0.00, x2=0.00, x3=8.00,\nx4=0.00, x5=7.00,
x6=0.00,\nx7=1.00\nInteger ✓',    'N1'),
    ('N6', 'Z=16.000\nx1=1.00, x2=5.00, x3=1.00,\nx4=7.00, x5=1.00,
x6=1.00,\nx7=0.00\nInteger ✓',    'N0'),
]

# Create graph
G = nx.DiGraph()
for (node_id, label, parent) in detailed_nodes:
    G.add_node(node_id, label = label)
    if parent:
        G.add_edge(parent, node_id)

# Layout
pos = nx.spring_layout(G, k = 5)

# Plot
plt.figure(figsize = (16, 12))
nx.draw(
    G,
    pos,
    with_labels = False,
    node_color  = '#FFF7E6',
    edge_color  = '#555',
    node_size   = 16000
)
```

```python
labels = nx.get_node_attributes(G, 'label')
nx.draw_networkx_labels(
    G,
    pos,
    labels      = labels,
    font_size   = 8,
    font_family = 'monospace'
)

plt.title(
    'Δέντρο Branch & Bound (3 Ακέραιες Λύσεις)',
    fontsize = 14,
    weight   = 'bold'
)
plt.axis('off')
plt.tight_layout()
plt.show()
```

```python
# Γραφική παράσταση για την άσκηση 5
# Hardcoded βάση του αποτελέσματος του αρχείου: lin_prog_code_HW2_5.py

import matplotlib.pyplot as plt
import networkx as nx

nodes = [
    ('N0', 'Z=109.621\nx1=0.79, x2=2.66,\nx3=2.83\nBranch on x1', None),
    ('N1', 'Z=94.750\nx1=0.00, x2=3.25,\nx3=0.25\nBranch on x2',  'N0'),
    ('N2', 'Z=89.000\nx1=0.00, x2=3.00,\nx3=1.00\nInteger ✓',      'N1'),
    ('N3', 'Infeasible', 'N1'),
    ('N4', 'Z=107.000\nx1=1.00, x2=2.33,\nx3=2.67\nBranch on x2', 'N0'),
    ('N5', 'Z=104.286\nx1=1.21, x2=2.00,\nx3=2.50\nBranch on x1', 'N4'),
    ('N6', 'Z=97.000\nx1=1.00, x2=2.00,\nx3=2.50\nBranch on x3',  'N5'),
    ('N7', 'Z=96.000\nx1=1.00, x2=2.00,\nx3=2.00\nInteger ✓',      'N6'),
    ('N8', 'Infeasible', 'N6'),
    ('N9', 'Z=94.333\nx1=2.00, x2=0.78,\nx3=1.89\nBounded',        'N5'),
    ('N10', 'Infeasible', 'N4'),
]

# Δημιουργία γράφου
G = nx.DiGraph()
for (node_id, label, parent) in nodes:
    G.add_node(node_id, label=label)
    if parent:
        G.add_edge(parent, node_id)

# Θέσεις κόμβων
pos = nx.spring_layout(G, k = 10, iterations = 1000)

# Σχεδίαση κόμβων και ακμών
plt.figure(figsize = (16, 12))
```

```python
nx.draw(
    G,
    pos,
    with_labels = False,
    node_color  = '#FFF7E6',
    edge_color  = '#555',
    node_size   = 12000,
    arrowsize   = 18
)
labels = nx.get_node_attributes(G, 'label')
nx.draw_networkx_labels(
    G,
    pos,
    labels      = labels,
    font_size   = 9,
    font_family = 'monospace'
)

plt.title(
    'Δέντρο Branch & Bound (Άσκηση 5)',
    fontsize = 14,
    weight   = 'bold'
)
plt.axis('off')
plt.tight_layout()
plt.show()
```