## Πρόβλημα 2:

Το δέντρο αναζήτησης που έχουμε στο συγκεκριμένο πρόβλημα, έχει βάθος d και παράγοντα διακλάδωσης b. Η εκφώνηση μας λέει ότι ο κόμβος με το μικρότερο βάθος που αντιστοιχεί σε κατάσταση στόχου έχει βάθος g<=d, οπότε καταλαβαίνουμε ότι ο στόχος μπορεί να είναι σε όλα τα επίπεδα, δηλαδή από τη ρίζα (με βάθος 0) μέχρι και τα φύλλα (με βάθος d) του δέντρου. Συνεπώς στην καλύτερη περίπτωση, δηλαδή ο στόχος να είναι στη ρίζα, θα δημιουργηθεί ένας (1) κόμβος. Στη χειρότερη περίπτωση, δηλαδή ο στόχος να είναι στα φύλλα, οι κόμβοι του κατώτερου επιπέδου θα παραχθούν μία φορά, οι κόμβοι του αμέσως προηγούμενου επιπέδου δύο φορές, οι προηγούμενοι τρεις κ.ο.κ. μέχρι τη ρίζα η οποία θα παραχθεί d + 1 φορές και διότι ο παράγοντας διακλάσωσης είναι b, στο πρώτο επίπεδο θα έχουμε b κόμβος, στο δεύτερο b^2 κόμβους κ.ο.κ. μέχρι το επίπεδο d όπου θα έχουμε b^d κόμβους. Άρα στη χειρότερη περίπτωση έχουμε:

$$1 \cdot b^d + 2 \cdot b^{(d-1)} + \ldots + (d-1) \cdot b^2 + d \cdot b + (d+1) \text{ κόμβους}$$

Ενώ στην καλύτερη: 1 κόμβο

## Πρόβλημα 3:

Η συνάρτηση είναι παραδεκτή εφ’ όσον η τιμή της ευρετικής για κάθε κόμβο είναι μικρότερη ή ίση από το πραγματικό κόστος που χρειάζεται για να φτάσουμε στον στόχο από αυτόν τον κόμβο, δηλαδή h(node) <= realcost(node->goal). Επίσης είναι συνεπής διότι η τιμή της ευρετικής για κάθε κόμβο είναι μικρότερη ή ίση από το κόστος που χρειάζεται να φτάσουμε στους γείτονες του συν της τιμής της ευρετικής για τους γείτονες αυτούς, δηλαδή h(node) <= cost(node->neighbor) + h(neighbor).

Παρακάτω θα εφαρμόσω τους ζητούμενους αλγόριθμους και όταν ο αλγόριθμος δεν μπορεί να διακρίνει ποιον κόμβο θα διαλέξει τους τοποθετώ στο σύνορο με τέτοιο τρόπο ώστε να βγουν από αυτό με αλφαβητική σειρά:

Για τον **αλγόριθμο πρώτα σε πλάτος (BFS)** θα χρησιμοποιήσουμε ουρά για το σύνορο(fringe). Έστω Q η ουρά αυτή:

Q [] Κάθε φορά που θα βγάζουμε έναν κόμβο από την ουρά θα θεωρείται visited και θα ελέγχουμε αν είναι ο στόχος.

Ξεκινάμε από τον κόμβο o103 και τον τοποθετούμε στο σύνορο:

Q [o103]

Ύστερα βγάζουμε τον o103 από το σύνορο, ελέγχουμε αν είναι κόμβος στόχου και αφού δεν είναι, παίρνουμε τους γείτονες του ts, b3, o109 και αν δεν τους έχουμε ξαναεπισκεφθεί τους βάζουμε στο σύνορο, με αλφαβητική σειρά σύμφωνα με την εκφώνηση, οπότε έχουμε:

Q [b3, o109, ts]

Ακολουθώντας την ίδια διαδικασία, βγάζουμε τον b3, κι αφού δεν είναι στόχος, παίρνουμε τους γείτονες του b1, b4, που δεν έχουμε επισκεφθεί ξανά, οπότε μπαίνουν στο σύνορο:

Q [o109, ts, b1, b4]

Τώρα βγάζουμε τον o109, με γείτονες τους o111, o119, που δεν ανήκουν στους visited και οπότε μπαίνουν στο σύνορο:

Q [ts, b1, b4, o111, o119]

Τώρα τον ts, με γείτονα τον mail:

Q [b1, b4, o111, o119, mail]

Τώρα τον b1, με γείτονες b2, c2 και έχουμε:

Q [b4, o111, o119, mail, b2, c2]

Τώρα τον b4, με γείτονα τον o109 τον οποίο έχουμε ήδη επισκεφθεί οπότε δεν τον βάζουμε στο σύνορο, οπότε πάμε κατευθείαν στο επόμενο βήμα και βγάζουμε τον o111, που δεν έχει γείτονες, οπότε βγάζουμε τον επόμενο της ουράς, δηλαδή τον o119, με γείτονες το storage, o123, οπότε έχουμε:

Q [mail, b2, c2, o123, storage]

Τώρα βγάζουμε τον mail, που δεν έχει γείτονες, οπότε βγάζουμε και τον b2, με γείτονα τον b4, τον οποίο έχουμε ήδη επισκεφθεί, οπότε βγάζουμε και τον c2, με γείτονες c1, c3, οι οποίοι μπαίνουν στο σύνορο:

Q[o123, storage, c1, c3]

Τώρα βγάζουμε τον o123, με γείτονες o125, r123 και τους βάζουμε στο σύνορο:

Q [storage, c1, c3, o125, r123]

Τώρα βγάζουμε το storage, που δεν έχει γείτονες, οπότε βγάζουμε τον c1, με γείτονα τον c3, τον οποίο τον βάζουμε στο σύνορο παρόλο που είναι ήδη μέσα, επειδή δεν ανήκε στους visited. Ύστερα βγάζουμε τον c3, που δεν έχει γείτονες, οπότε βγάζουμε τον o125 που κι αυτός με τη σειρά του δεν έχει γείτονες, οπότε στο τέλος βγάζουμε τον r123, ο οποίος είναι ο στόχος και τελειώνει η αναζήτηση. Συνεπώς η σειρά με την οποία βγαίνουν οι κόμβοι από το σύνορο είναι η εξής:

**o103, b3, o109, ts, b1, b4, o111, o119, mail, b2, c2, o123, storage, c1, c3, o125, r123**

Για τον **αλγόριθμο πρώτα σε βάθος(DFS)** θα χρησιμοποιήσουμε στοίβα για το σύνορο, στην οποία κορυφή θα θεωρείται το δεξί άκρο, δηλαδή θα βγάζουμε το τελευταίο στοιχείο πρώτα. Έστω S η στοίβα αυτή:

S []  Κάθε φορά που θα βγάζουμε έναν κόμβο από την ουρά θα θεωρείται visited και θα ελέγχουμε αν είναι ο στόχος.

Παίρνουμε τον o103 και τον βάζουμε στο σύνορο:

S [o103]

Τον αφαιρούμε και αφού δεν είναι στόχος, παίρνουμε τους γείτονες και τους βάζουμε στο σύνορο:

S [ts, o109, b3]

Αφαιρούμε τον b3, και ελέγχουμε αν είναι στόχος (που δεν είναι) και παίρνουμε τους γείτονες του και τους βάζουμε στο σύνορο:

S [ts, o109, b4, b1]

Αφαιρούμε τον b1, και βάζουμε τον c2 και b2:

S [ts, o109, b4, c2, b2]

Αφαιρούμε τον b2, και βάζουμε τον b4 ξανά:

S [ts, o109, b4, c2, b4]

Αφαιρούμε τον b4 και βάζουμε ξανά τον o109:

S [ts, o109, b4, c2, o109]

Αφαιρούμε τον o109 και βάζουμε τους o111 και o 119:

S [ts, o109, b4, c2, o119, o111]

Βγάζουμε τον o111, που δεν έχει γείτονες, οπότε βγάζουμε τον o119, με γείτονες storage, o123:

S [ts, o109, b4, c2, storage, o123]

Βγάζουμε τον o123, και βάζουμε τους o125, r123:

S [ts, o109, b4, c2, storage, r123, o125]

Βγάζουμε τον o125 που δεν έχει γείτονες, οπότε βγάζουμε και τον r123, ο οποίος είναι ο στόχος μας και τελειώνει η αναζήτηση. Συνεπώς η σειρά με την οποία βγαίνουν οι κόμβοι από το σύνορο είναι:

**o103, b3, b1, b2, b4, o109, o111, o119, o123, o125, r123**

Για τον **αλγόριθμο πρώτα σε βάθος με επαναληπτική εκβάθυνση(IDS)** θα εκτελέσουμε αναζήτηση πρώτα σε βάθος με μέγιστο αυξανόμενο βάθος, ξεκινώντας από 0.

Βάθος = 0

Βάζουμε και βγάζουμε τον o103 και αφού δεν είναι στόχος, ξαναεκτελούμε αναζήτηση με βάθος = 1.

Βάζουμε τον o103

S [o103]

Τον αφαιρούμε και βάζουμε τους γείτονες:

S [ts, o109, b3]

Βγάζουμε με τη αλφαβητική σειρά τους b3, o109, ts και βλέπουμε ότι δεν είναι στόχοι. Ξεκινάμε από την αρχή πάλι, γιατί φτάσαμε σε μέγιστο βάθος, αλλά με μέγιστο βάθος 2.

Βάζουμε τον o103

S [o103]

Τον αφαιρούμε και βάζουμε τους γείτονες:

S [ts, o109, b3]

Βγάζουμε τον b3 και βάζουμε τους γείτονες:

S [ts, o109, b4, b1]

Βγάζουμε τον b1, που δεν είναι στόχος, αντίστοιχα και τον b4, αλλά δε βάζουμε τους γείτονες τους γιατί έχουμε φτάσει στο μέγιστο βάθος. Μετά βγάζουμε τον o109 και βάζουμε τους o111 και o119:

S [ts, o119, o111]

Βγάζουμε με τη σειρά τους o111, o119 αλλά δεν προχωράμε παρακάτω γιατί φτάσαμε μέγιστο βάθος, οπότε βγάζουμε τον ts και βάζουμε τον γείτονα mail:

S [mail]

Βγάζουμε τον mail και σταματάει η αναζήτηση. Ξαναρχίζουμε αφού δεν έχουμε βρει στόχο, με βάθος αυτή τη φορά ίσο με 3.

Βάζουμε τον o103:

S [o103]

Τον βγάζουμε και βάζουμε τους γείτονες:

S [ts, o109, b3]

Βγάζουμε τον b3 και βάζουμε τους γείτονες:

S [ts, o109, b4, b1]

Βγάζουμε τον b1 και βάζουμε τους γείτονες:

S [ts, o109, b4, c2, b2]

Βγάζουμε με τη σειρά τους b2, c2 αλλά δε βάζουμε τους γείτονες γιατί φτάσαμε σε μέγιστο βάθος. Οπότε βγάζουμε τον b4 αλλά δε βάζουμε τον o109 γιατί είναι ήδη στην κορυφή της στοίβας:

S [ts, o109]

Τώρα τον αφαιρούμε, και βάζουμε τους γείτονες o111, o119:

S [ts, o119, o111]

Βγάζουμε τον o111 αλλά δεν έχει γείτονες οπότε βγάζουμε τον o119 και βάζουμε τους γείτονες:

S [ts, storage, o123]

Βγάζουμε τον o123 αλλά δε βάζουμε τους γείτονες γιατί φτάσαμε στο μέγιστο βάθος και μετά βγάζουμε τον storage που δεν έχει γείτονες, οπότε βγάζουμε και τον ts και βάζουμε τον mail, τον οποίο αφαιρούμε και σταματάμε. Αρχίζουμε πάλι με μέγιστο βάθος 4:

Βάζουμε τον o103:

S [o103]

Τον βγάζουμε και βάζουμε τους γείτονες:

S [ts, o109, b3]

Βγάζουμε τον b3 και βάζουμε τους γείτονες:

S [ts, o109, b4, b1]

Βγάζουμε τον b1 και βάζουμε τους γείτονες:

S [ts, o109, b4, c2, b2]

Βγάζουμε τον b2 και βάζουμε τον b4:

S [ts, o109, b4, c2, b4]

Βγάζουμε τον b4 και βάζουμε τον γείτονα του:

S [ts, o109, b4, c2, o109]

Βγάζουμε τον o109 και βάζουμε τους γείτονες:

S [ts, o109, b4, c2, o119, o111]

Βγάζουμε τον o111, που δεν έχει γείτονες, οπότε βγάζουμε κατευθείαν τον o119 και βάζουμε τους γείτονες:

S [ts, o109, b4, c2, storage, o123]

Βγάζουμε τον o123 και βάζουμε τους γείτονες:

S [ts, o109, b4, c2, storage, r123, o125]

Βγάζουμε τον o125 που δεν έχει γείτονες και βγάζουμε απ'ευθείας τον r123, που είναι ο στόχος οπότε και σταματάμε τον αλγόριθμο. Συνεπώς η σειρά με την οποία αφαιρούνται οι κόμβοι από το σύνορο είναι η εξής:

**o103, o103, b3, o109, ts, o103, b3, b1, b4, o109, o111, o119, ts, mail, o103, b3, b1, b2, c2, b4, o109, o111, o119, o123, storage, ts, mail, o103, b3, b1, b2, b4, o109, o111, o119, o123, o125, r123**

Για τον **αλγόριθμο άπληστης αναζήτησης πρώτα στον καλύτερο** θα χρησιμοποιήσουμε για σύνορο ουρά προτεραιότητας, με αριθμό προτεραιότητας για τον κάθε κόμβο βάσει της τιμής της ευρετικής συνάρτησης για τον κόμβο αυτό. Θα αφαιρούμε τον κόμβο με τον μικρότερο αριθμό, δηλαδή τη μεγαλύτερη προτεραιότητα. Έστω PQ η ουρά αυτή:

PQ []

Ξεκινάμε με τον o103:

PQ [(o103, 21)]

Τον αφαιρούμε και παίρνουμε τους γείτονες:

PQ [(ts, 23), (b3, 17), (o109, 24)]

Αφαιρούμε τον b3 επειδή έχει μεγαλύτερη προτεραιότητα και βάζουμε τους γείτονες:

PQ [(ts, 23), (o109, 24), (b1, 13), (b4, 18)]

Αφαιρούμε τον b1 και βάζουμε τους γείτονες:

PQ [(ts, 23), (o109, 24), (b4,18), (b2, 15), (c2, 10)]

Αφαιρούμε τον c2 και βάζουμε τους γείτονες:

PQ [(ts, 23), (o109, 24), (b4, 18), (b2, 15), (c1, 6), (c3, 12)]

Αφαιρούμε τον c1 αλλά δε βάζουμε τον c3 γιατί είναι ήδη στο τέλος της ουράς. Οπότε αφαιρούμε και τον c3, όντας ο επόμενος με τη μεγαλύτερη προτεραιότητα, ο οποίος δεν έχει γείτονες κι έτσι αφαιρούμε και τον b2, αλλά δε βάζουμε τον b4 επειδή υπάρχει ήδη. Ύστερα αφαιρούμε τον b4 και παίρνουμε τον o109, ο οποίος είναι ήδη στην ουρά προτεραιότητας, οπότε δεν τον ξαναβάζουμε. Στο επόμενο

βήμα αφαιρούμε τον ts και παίρνουμε τους γείτονες του και τους βάζουμε στην ουρά προτεραιότητας:

PQ [(o109, 24), (mail, 26)]

Αφαιρούμε τον o109 και βάζουμε τους γείτονες:

PQ [(mail, 26), (o111, 27), (o119, 11)]

Αφαιρούμε τον o119 και βάζουμε τους γείτονες:

PQ [(mail, 26), (o111, 27), (storage, 12), (o123, 4)]

Αφαιρούμε τον o123 και βάζουμε τον r123:

PQ [(mail, 26), (o111, 27), (storage, 12), (r123, 0)]

Αφαιρούμε τον r123 και σταματάμε διότι είναι κόμβος στόχου. Συνεπώς η σειρά με την οποία αφαιρούνται οι κόμβοι είναι:

**o103, b3, b1, c2, c1, c3, b2, b4, ts, o109, o119, o123, r123**

Για τον **αλγόριθμο A* με ευρετική συνάρτηση,** θα χρησιμοποιήσουμε πάλι την ευρετική που χρησιμοποιήθηκε και προηγουμένως και για σύνορο μια ουρά προτεραιότητας, με μόνη διαφορά ότι ο αριθμός αυτός θα είναι η τιμή της ευρετικής συν το κόστος για να πάμε στον κόμβο αυτό (ο αριθμός αυτός θα ανανεώνεται στις περιπτώσεις που το κόστος για να πάμε στον κόμβο αυτό είναι μικρότερο από το προηγούμενο):

Ξεκινάμε κλασικά με τον o103:

PQ [(o103, 21 + 0)]

Τον αφαιρούμε και παίρνουμε τους γείτονες: ts -> 23 + 8 = 31, b3 -> 17 + 4 = 21, o109 -> 24 + 12 = 36 και τους βάζουμε στην ουρά:

PQ [(ts, 31), (b3, 21), (o109, 36)]

Αφαιρούμε τον b3 και παίρνουμε: b1 -> 13 + 4 = 17, b4 -> 18 + 7 = 25:

PQ [(ts, 31), (o109, 36), (b1, 17), (b4, 25)]

Αφαιρούμε τον b1 και παίρνουμε: b2 -> 15 + 6 = 21, c2 -> 10 + 3 = 13:

PQ [(ts, 31), (o109, 36), (b4, 25), (b2, 21), (c2, 13)]

Αφαιρούμε τον c2 και παίρνουμε: c3 -> 12 + 6 = 18, c1 -> 6 + 4 = 10

PQ [(ts, 31), (o109, 36), (b4, 25), (b2, 21), (c3, 18) , (c1, 10)]

Αφαιρούμε τον c1, με γείτονα τον c3, που υπάρχει ήδη, αλλά αυτή τη φορά το κόστος μετάβασης είναι 8, οπότε ο αριθμός προτεραιότητας γίνεται 12 + 8 = 20, μεγαλύτερος από πριν άρα δεν αλλάζουμε κάτι.

Αφαιρούμε τον c3, αλλά δεν έχει γείτονες οπότε αφαιρούμε και τον b2, ο οποίος έχει γείτονα μόνο τον b4, οποίος είναι ήδη στην ουρά προτεραιότητας αλλά θα ανανεώσουμε τον αριθμό του γιατί αυτή τη φορά το κόστος μετάβασης σε αυτόν είναι 3, δηλαδή συνολικά 18 + 3 = 21 (άρα μικρότερο από το 25 που είχαμε πριν), οπότε έχουμε:

PQ [(ts, 31), (o109, 36), (b4, 21)]

Αφαιρούμε τον b4, ο οποίος έχει γείτονα τον o109, με κόστος μετάβασης 7, δηλαδή έχουμε 24 + 7 = 31, οπότε ανανεώνουμε:

PQ [(ts, 31), (o109, 31)]

Εδώ έχουμε ισοβαθμία αλλά αφαιρούμε τον o109 επειδή αλφαβητικά είναι πρώτος και παίρνουμε τους γείτονες: o111 -> 27 + 4 = 31, o119 -> 11 + 16 = 27:

PQ [(ts, 31), (o111, 31), (o119, 27)]

Αφαιρούμε τον o119 και παίρνουμε: storage -> 12 + 7 = 19, o123 -> 4 + 9 = 13:

PQ [(ts, 31), (o111, 31), (storage, 19), (o123, 13)]

Αφαιρούμε τον o123 και παίρνουμε: r123 -> 0 + 4 = 4, o125 -> 6 + 4 = 10:

PQ [(ts, 31), (o111, 31), (storage, 19), (r123, 4), (o125, 10)]

Αφαιρούμε τον r123 και σταματάμε αφού είναι στόχος. Συνεπώς η σειρά με την οποία αφαιρούνται οι κόμβοι από το σύνορο είναι η εξής:

**o103, b3, b1, c2, c1, c3, b2, b4, o109, o119, o123, r123**

## Πρόβλημα 4:

- **Καταστάσεις:** Το ρομπότ βρίσκεται σε μια αρχική κατάσταση και πρέπει να πάρει πακέτα που μπορεί να βρίσκονται σε οποιαδήποτε θέση και να τα μεταφέρει σε μια άλλη.

- **Αρχική κατάσταση:** Η αρχική κατάσταση είναι η θέση που ξεκινάει το ρομπότ, η οποία είναι η o103, σύμφωνα και με το προηγούμενο πρόβλημα.

- **Συνάρτηση διαδόχων:** Παράγει τις καταστάσεις, δηλαδή τους διαδόχους-κόμβους, που προκύπτουν από τις επιτρεπτές κινήσεις που μπορεί να κάνει το ρομπότ, όπως φαίνεται στον γράφο του προηγούμενου προβλήματος (Πάνω, Κάτω, Αριστερά, Δεξιά, Διαγώνια).

- **Έλεγχος στόχου:** Ελέγχει αν στην τωρινή κατάσταση, έχουν μεταφερθεί όλα τα πακέτα στις θέσεις που πρέπει.

- **Κόστος διαδρομής:** Η κίνηση από κόμβο σε κόμβο έχει ένα συγκεκριμένο κόστος, το οποίο φαίνεται στον γράφο του προηγούμενου προβλήματος, συνεπώς το κόστος της διαδρομής θα είναι το άθροισμα των κοστών των κόμβων της διαδρομής που ακολούθησε το ρομπότ για να μεταφέρει όλα τα πακέτα.

Ο στόχος του προβλήματος είναι να μεταφέρει το ρομπότ όλα τα πακέτα από τις θέσεις που βρίσκονται στον προορισμό τους. Αυτό σημαίνει, ότι το ρομπότ θα διανύει σίγουρα, σε κάθε βήμα, την απόσταση από τη θέση που είναι μέχρι το κοντινότερο πακέτο και μετά την απόσταση από το πακέτο αυτό μέχρι τον προορισμό του πακέτου και μετά απο κει την απόσταση μέχρι το επόμενο κοντινότερο πακέτο κ.ο.κ. Συνεπώς μια καλή ευρετική θα ήταν αυτή η οποία υπολογίζει την απόσταση από τη θέση που βρίσκεται το ρομπότ μέχρι το κοντινότερο πακέτο συν (+) την απόσταση του πακέτου αυτού μέχρι τη θέση που βρίσκεται το μακρινότερο πακέτο. Αυτή η συνάρτηση είναι παραδεκτή επειδή η απόσταση που υπολογίζει είναι πάντα μικρότερη της πραγματικής, διότι το ρομπότ πρέπει επιπλέον να διανύσει και τις αποστάσεις μεταξύ της θέσης που βρίσκονται τα πακέτα και του προορισμού τους. Επίσης, ο αριθμός των κόμβων που θα παράγονται θα είναι σχετικά μικρός, επειδή η τιμή της συνάρτησης που επιστρέφεται είναι αρκετά κοντά στην πραγματική απόσταση, διότι στην απόσταση του κοντινότερου πακέτου προσθέτουμε και τη μέγιστη απόσταση μεταξύ των πακέτων.

## Πρόβλημα 5:

Σύμφωνα με τις διαφάνειες και το βιβλίο, για να είναι πλήρης η αμφίδρομη αναζήτηση, θα πρέπει και οι 2 υπορουτίνες που χρησιμοποιούνται να είναι αναζητήσεις πρώτα σε πλάτος. Αυτό συμβαίνει επειδή για να ελέγξουμε αν έχει βρεθεί λύση, κοιτάμε αν ο κόμβος που πρόκειται να επεκτείνουμε, βρίσκεται στο σύνορο του άλλου. Έτσι, αν το ζεύγος υπορουτινών δεν είναι BFS και οι 2, τότε υπάρχουν πολλές περιπτώσεις όπου ο ένας αλγόριθμος θα έχει βγάλει ήδη από το σύνορο του τους κόμβους, τους οποίους ελέγχει ο άλλος. Για παράδειγμα στην περίπτωση (α), αν η αναζήτηση περιορισμένου βάθους έχει ως μέγιστο βάθος, ένα επίπεδο που είναι μικρότερο από το μισό του βάθους του δέντρου, τότε ο αλγόριθμος θα σταματήσει και το σύνορο του θα είναι άδειο, με αποτέλεσμα ο αλγόριθμος της αναζήτησης πρώτα σε πλάτος, να μη βρει κανέναν κόμβο που να ανήκει στο σύνορο του άλλου, αφού θα είναι κενό. Έτσι δε θα βρεθεί λύση. Αν όμως γενικά, αντί να ελέγχαμε αν ο κόμβος είναι στο σύνορο του άλλου αλγορίθμου και ελέγχαμε αν ανήκει στο σύνολο των κόμβων που έχει ήδη επισκεφθεί ο άλλος αλγόριθμος, τότε πολύ πιθανό να βρίσκαμε λύση σε πολλά από τα ζεύγη των υπορουτινών. Επειδή, όμως, οι απαντήσεις πρέπει να σέβονται το πνεύμα του βιβλίου, η πιο σωστή απάντηση σε αυτήν την περίπτωση είναι ότι **κανένας** αλγόριθμος με τις υπορουτίνες της εκφώνησης δεν είναι πλήρης. Και αφού δεν είναι πλήρης, δεν μπορεί να είναι και βέλτιστος. Γιατί πώς θα εγγυηθούμε ότι η λύση θα είναι η βέλτιστη, όταν δεν μπορούμε να εγγυηθούμε ότι θα βρεθεί λύση;

Για να δούμε αν πραγματικά συναντιούνται οι δύο υπορουτίνες αρκεί, όπως ανέφερα και παραπάνω, να εκτελούμε τον εξής αποδοτικό έλεγχο: όταν φτάνει η μία υπορουτίνα σε έναν κόμβο, πριν τον επεκτείνει, θα ελέγχει αν αυτός ο κόμβος ανήκει στο σύνολο των κόμβων που έχει επισκεφθεί ο άλλος. Αν ναι, τότε θεωρούμε ότι έχουν συναντηθεί, άρα θα έχει βρεθεί και λύση και με αυτόν τον τρόπο θα μπορούσε και τα ζεύγη του ερωτήματος να αποτελούν πλήρεις αλγορίθμους.