

Εθνικό Μετσόβιο Πολυτεχνείο  
Σχολή ΗΜ&ΜΥ  
Αλγόριθμοι και Πολυπλοκότητα  
7<sup>ο</sup> εξάμηνο  
Ακαδημαϊκή περίοδος: 2015-2016



---

## 2<sup>η</sup> Σειρά Γραπτών Ασκήσεων

---

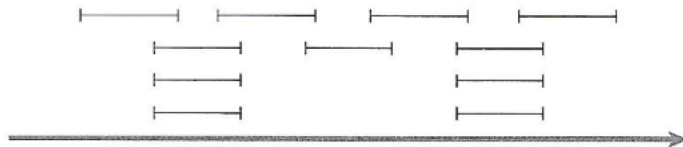
Ψαρουδάκη Ελένη  
Α.Μ.: 03112080

7 Δεκεμβρίου 2015

## Άσκηση 1: Δρομολόγηση Μαθημάτων

(α)

1. Το κριτήριο αποτυγχάνει. Αυτό γίνεται φανερό στο παρακάτω στιγμιότυπο.



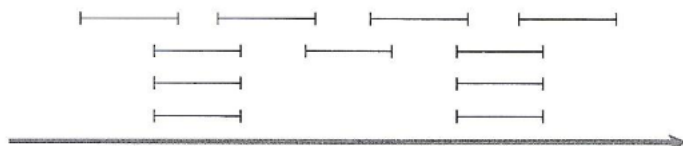
Σύμφωνα με το παραπάνω στιγμιότυπο, το μάθημα με τις λιγότερες επικαλύψεις είναι το δεύτερο στοιχείο της δεύτερης σειράς. Στο παραπάνω στιγμιότυπο όμως είναι ξεκάθαρο πως η βέλτιστη λύση προκύπτει από το συνδιασμό των τεσσάρων μαθημάτων της πρώτης σειράς. Η πρώτη μας επιλογή θα οδηγήσει σε μία λύση με 3 μαθήματα, πράγμα που την καθιστά μη βέλτιστη.

2. Το κριτήριο αποτυγχάνει. Αυτό γίνεται φανερό στο παρακάτω στιγμιότυπο.



Σύμφωνα με αυτό θα αφαιρεθεί ένα από τα δύο μαθήματα που βρίσκονται στην επάνω σειρά και στη συνέχεια και το δεύτερο. Έτσι θα οδηγηθούμε σε μία λύση που περιλαμβάνει μόνο το μάθημα που βρίσκεται στην δεύτερη γραμμή. Ωστόσο η βέλτιστη λύση προκύπτει από το συνδιασμό των δύο μαθημάτων που βρίσκονται στην επάνω σειρά.

3. Και αυτό το κριτήριο αποτυγχάνει. Θα χρησιμοποιήσουμε το ίδιο στιγμιότυπο που χρησιμοποιήσαμε στο 1.



Παρατηρούμε πως υπάρχουν 8 διαφορετικά μαθήματα που εμφανίζουν το μέγιστο αριθμό επικαλύψεων (4). Δύο από αυτά (το δεύτερο και το τρίτο της πρώτης σειράς) ανήκουν στην βέλτιστη λύση. Αν ο αλγόριθμος επιλέξει να αφαιρέσει ένα από αυτά τα δύο, τότε δεν θα οδηγηθούμε σε βέλτιστη λύση. Αυτό όμως δεν μπορούμε να το ελέγξουμε, γιατί εξαρτάται από τη μορφή του στιγμιότυπου κάθε φορά. Επομένως το κριτήριο αποτυγχάνει.

(β)

Για το πρόβλημα αυτό θα ταξινομήσουμε τα μαθήματα σε αύξουσα σειρά σύμφωνα με το χρόνο ολοκλήρωσης (θα βρίσκονται σε μορφή κυκλικής λίστας για να βολεύουν στη λύση της άσκησης). Θα εφαρμόσουμε το κριτήριο άπληστης επιλογής για το ελάχιστο χρόνο ολοκλήρωσης. Το επόμενο μάθημα δρομολογείται εάν είναι εφικτό ή αγνοείται αν όχι. Σταματάμε όταν βρούμε το πρώτο μάθημα που η ώρα ολοκλήρωσης του ξεπερνάει την αρχή του πρώτου μαθήματος.

Επαναλαμβάνουμε αυτή τη διαδικασία για όλα τα μαθήματα. Αν βρούμε καλύτερη λύση κρατάμε την καινούρια.

Η λύση αυτή θα έχει πολυπλοκότητα  $\mathcal{O}(n^2)$  καθώς η greedy λύση για ένα μάθημα απαιτεί χρόνο  $n$  και έχουμε  $n$  μαθήματα.

## (γ)

Στην περίπτωση που τα βάρη έχουν διαφορετική τιμή από την μονάδα, οι greedy αλγόριθμοι δεν δουλεύουν. Επομένως θα χρησιμοποιήσουμε δυναμικό προγραμματισμό. Για αρχή ταξινομούμε τα διαστήματα σύμφωνα με τον χρόνο ολοκλήρωσης ως

$$f_1 \leq f_2 \leq \dots \leq f_n$$

Ως  $p(j)$  ορίζουμε το τελευταίο δυνατό μάθημα  $i < j$  έτσι ώστε τα μαθήματα να είναι μη επικαλυπτόμενα. Υπάρχουν δύο δυνατά σενάρια για κάθε μάθημα. Ο αλγόριθμος να το επιλέξει ή όχι.

- Αν το επιλέξει δεν μπορούμε να διαλέξουμε τα μαθήματα  $p(j), p(j) + 1, \dots, j - 1$  και πρέπει να επιλέξουμε τη βέλτιστη λύση από τα υπόλοιπα μη επικαλυπτόμενα μαθήματα  $1, 2, \dots, p(j)$ .
- Αν δεν το επιλέξουμε πρέπει να επιλέξουμε τη βέλτιστη λύση από τα υπόλοιπα μη επικαλυπτόμενα μαθήματα  $1, 2, \dots, j - 1$

Επομένως προκύπτει η παρακάτω αναδρομική λύση:

$$C(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max = \begin{cases} w_j + C(p(j)) \\ C(j - 1) \end{cases} & \text{otherwise} \end{cases}$$

Το κόστος του αλγόριθμου θα είναι  $\mathcal{O}(n \log n)$  καθώς είναι υποχρεωτική η ταξινόμηση.

## Άσκηση 2: Πομποί και Δέκτες

### (α)

Για την λύση αυτού του προβλήματος θα στηριχτούμε στο γεγονός ότι οι πομποί εκπέμπουν μόνο προς τα ανατολικά και οι δέκτες λαμβάνουν μόνο προς τα δυτικά. Επομένως ζευγάρια στην ακολουθία μπορούν να δημιουργηθούν μόνο αν ο δέκτης βρίσκεται δεξιά του πομπού στην ακολουθία. Παρατηρούμε ότι δεν μας ενδιαφέρει στο δημιουργία ζευγαριών ποιος πομπός συνδέεται με ποιον δέκτη παρά μόνο ο αριθμός των ζευγαριών. Επομένως μπορούμε να ελέγχουμε τα στοιχεία της ακολουθείας και για κάθε πομπό προσθέτουμε ένα στο  $num$  και στην περίπτωση που έχουμε δέκτη ελέγχουμε αν υπάρχουν διαθέσιμοι πομποί. Αν υπάρχουν αυξάνουμε τον αριθμό των ζευγαριών κατά ένα. Ακολουθεί ο αλγόριθμος.

---

**Algorithm 1** Function *Find pairs*

---

```
1: function Function find_pairs(a)
2:    $pairs \leftarrow 0$ 
3:    $num \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $n$  do
5:     if  $a[i] = t$  then
6:        $num \leftarrow num + 1$ 
7:     else
8:       if  $num > pairs$  then
9:          $pairs \leftarrow pairs + 1$ 
```

---

Το κόστος του αλγόριθμου είναι  $\mathcal{O}(n)$  καθώς γίνονται ακριβώς  $n$  συγκρίσεις.

### (β)

Για το ερώτημα αυτό θα χρησιμοποιήσουμε δυναμικό προγραμματισμό με την παρακάτω αναδρομική σχέση.

$$C(i, r) = \begin{cases} C(i-1, r-1) + R_i & \text{if } r > \frac{i-1}{2} \\ C(i-1, r) + T_i & \text{if } r = 0 \text{ \& } i \neq 1 \\ \min = \begin{cases} C(i-1, r) + T_i \\ C(i-1, r-1) + R_i \end{cases} & \text{otherwise} \end{cases}$$

με αρχικές συνθήκες:

- $i = n$
- $r = \frac{n}{2}$
- $C(1, 0) = T_1$

Ως  $i$  ορίζουμε το στοιχείο στο οποίο βρισκόμαστε κάθε φορά που κυμαίνεται από  $n$  έως 1 και ως  $r$  τον αριθμό των δεκτών που έχουμε επιλέξει μέχρι και την στιγμή  $i$ . Ξεκινάμε από το τελευταίο στοιχείο (γνωρίζουμε πως θα είναι δέκτης) και υπάρχει ο περιορισμός, πως για κάθε χρονική στιγμή πρέπει να υπάρχουν  $r \leq \frac{i}{2}$  δέκτες. Επομένως αν  $r > \frac{i-1}{2}$  γνωρίζουμε πως το τελευταίο στοιχείο  $i$  θα πρέπει αναγκαστικά να είναι δέκτης. Αν  $r \leq \frac{i-1}{2}$  το στοιχείο  $i$  μπορεί είτε να είναι δέκτης είτε πομπός. Θέλουμε το  $\min$  από την κάθε περίπτωση με το αντίστοιχο κόστος της επιλογής  $i$ .

Για την επίλυση θα χρησιμοποιηθεί ένας πίνακας  $n \cdot \frac{n}{2}$  ο οποίος στην αρχή έχει όλα τα στοιχεία του μηδενικά (θεωρούμε πως δεν μπορεί να υπάρξει μηδενική κατανάλωση

ισχύος). Ο πίνακας αυτός δεν θα γεμίσει ολόκληρος, καθώς ο αλγόριθμος δεν θα αποκτήσει πρόσβαση σε ορισμένες θέσεις του πίνακα, εξαιτίας της αναδρομικής σχέσης. Ο αλγόριθμος καλείται για  $i = n$  και  $r = \frac{n}{2}$ .

---

**Algorithm 2** Function *Find pairs dynamic*

---

```

1: function find_pairs_dynamic( $i, r$ )
2:   if  $r = 0 \ \& \ i = 1$  then
3:      $C[1, 0] \leftarrow T_1$ 
4:   else if  $r = 0 \ \& \ i \neq 1$  then
5:     if  $C[i - 1, r] = 0$  then
6:       call find_pairs_dynamic( $i - 1, r$ )
7:        $C[i, r] \leftarrow C[i-1, r] + T_i$ 
8:     else if  $r > \frac{i-1}{2}$  then
9:       if  $C[i - 1, r - 1] = 0$  then
10:        call find_pairs_dynamic( $i - 1, r - 1$ )
11:         $C[i, r] \leftarrow C[i-1, r-1] + R_i$ 
12:     else
13:       if  $C[i - 1, r] = 0$  then
14:        call find_pairs_dynamic( $i - 1, r$ )
15:       if  $C[i - 1, r - 1] = 0$  then
16:        call find_pairs_dynamic( $i - 1, r - 1$ )
17:       if  $C[i - 1, r] + T_i < C[i - 1, r - 1] + R_i$  then
18:         $C[i, r] \leftarrow C[i-1, r] + T_i$ 
19:       else
20:         $C[i, r] \leftarrow C[i-1, r-1] + R_i$ 
21:   return

```

---

Γνωρίζουμε πως η υπολογιστική πολυπλοκότητα είναι ίση με  $\mathcal{O}(n^2)$  καθώς προκύπτει από το γινόμενο του statespace ( $\frac{n^2}{2}$ ) με τον χρόνο υπολογισμού σε κάθε βήμα (σταθερός).

### Άσκηση 3: Τηλεκατευθυνόμενο Drone

Στην άσκηση αυτή θα χρησιμοποιήσουμε δυναμικό προγραμματισμό με την παρακάτω αναδρομική σχέση:

$$P(x, i) = P(x - \hat{s}_i, i - 1) \cdot p_i + P(x + \hat{s}_i, i - 1) \cdot (1 - p_i)$$

με αρχικές συνθήκες:

- $x = k$
- $i = n$
- $P(0, 0) = 1$
- $P(x, 0) = 0 \ \forall x \neq 0$

Οι τελευταίες δύο συνθήκες προκύπτουν από τον περιορισμό να ξεκινάμε μόνο από τη θέση  $x = 0$ .

Ως  $x$  ορίζουμε τη θέση στην οποία βρίσκεται το drone τη χρονική στιγμή  $i$ . Στη θέση αυτή μπορούμε να πάμε:

- αν βρισκόμαστε στη θέση  $x - \hat{s}_i$  την χρονική στιγμή  $i - 1$  και κινηθούμε με το σωστό σήμα
- αν βρισκόμαστε στη θέση  $x + \hat{s}_i$  την χρονική στιγμή  $i - 1$  και κινηθούμε με το λανθασμένο σήμα

Η πιθανότητα να βρισκόμαστε στη θέση  $x$  προκύπτει λοιπόν από το άθροισμα των πιθανοτήτων να κινηθούμε με σωστό ή λανθασμένο σήμα, πολλαπλασιασμένο κάθε φορά με την πιθανότητα να βρισκόμαστε στην κατάλληλη θέση.

Για την επίλυση χρησιμοποιούμε έναν πίνακα  $P$  με  $2n$  στήλες (όρια  $-n$  έως  $n$ ) και  $n$  γραμμές ( $0$  έως  $n$ ). Ο πίνακας θεωρούμε πως είναι αρχικοποιημένος στην τιμή  $-1$ , καθώς δεν υπάρχει αρνητική πιθανότητα. Και πάλι ο πίνακας δεν θα συμπληρωθεί ολόκληρος, καθώς θα τον συμπληρώσουμε αναδρομικά και μερικές από τις θέσεις δεν θα μας χρειαστούν, καθώς σε κάθε γραμμή γεμίζουν οι διπλάσιες θέσεις. Ακολουθεί ο αλγόριθμος γεμίσματος του πίνακα.

---

**Algorithm 3** Function *Find possibility*

---

```
1: function Function find_possibility( $x, i$ )
2:   if  $x = 0$  &  $i = 0$  then
3:      $P[0, 0] \leftarrow 1$ 
4:   else if  $i = 0$  then
5:      $P[x, 0] \leftarrow 0$ 
6:   else
7:     if  $P[x - s[i], i - 1] = -1$  then
8:       call find_possibility( $x - s[i], i - 1$ )
9:     if  $P[x + s[i], i - 1] = -1$  then
10:      call find_possibility( $x + s[i], i - 1$ )
11:      $P[x, i] \leftarrow P[x - s[i], i - 1] \cdot p[i] + P[x + s[i], i - 1] \cdot (1 - p[i])$ 
12:   return
```

---

Η υπολογιστική πολυπλοκότητα είναι ίση με  $\mathcal{O}(n^2)$  καθώς προκύπτει από το γινόμενο του statespace (μέγιστο δυνατό  $2 \cdot n^2$ ) με τον χρόνο υπολογισμού σε κάθε βήμα (σταθερός).

## Άσκηση 4: Βγάζοντας Βόλτα το Σκύλο

Για την άσκηση αυτή θα χρησιμοποιήσουμε δυναμικό προγραμματισμό με την παρακάτω αναδρομική σχέση.

$$D(n, m) = \max \left\{ \begin{array}{ll} \begin{cases} D(n, m-1) & \text{if } n=1 \\ D(n-1, m) & \text{if } m=1 \end{cases} & \\ \min \left\{ \begin{array}{l} D(n-1, m-1) \\ D(n-1, m) \\ D(n, m-1) \end{array} \right\} & \text{otherwise} \\ d(p_n, q_m) & \end{array} \right.$$

με αρχικές συνθήκες:

- δείκτης για τη θέση του ανθρώπου:  $n$
- δείκτης για θέση του σκύλου:  $m$
- $D(1, 1) = d(p_1, q_1)$

Οι τελευταίες τέσσερις συνθήκες προκύπτουν από την ανάγκη για τερματισμό της αναδρομικής σχέσης. Στην περίπτωση που κάποιος από τους δύο (είτε ο άνθρωπος είτε ο σκύλος) έχουν φτάσει στη θέση 0, η αναδρομή δεν συνεχίζεται για αυτόν τον συνδιασμό και η απόσταση δεν λαμβάνεται υπόψιν. Αν φτάσουν και οι δύο στη θέση 1 έχουμε φτάσει στο τέλος της διαδρομής, οπότε και πάλι η αναδρομή τερματίζει.

Σε κάθε συνδιασμό θέσεων μπορούμε να πάμε με τρεις διαφορετικούς τρόπους:

- ο άνθρωπος έμεινε ακίνητος και ο σκύλος προχώρησε ένα βήμα
- ο άνθρωπος προχώρησε ένα βήμα και ο σκύλος έμεινε ακίνητος
- και ο άνθρωπος και ο σκύλος προχώρησαν ένα βήμα

Επομένως καθώς αναζητούμε το ελάχιστο δυνατό μήκος λουριού πρέπει να επιλέγουμε κάθε φορά το μέγιστο μεταξύ της απόστασης των θέσεων που βρίσκονται άνθρωπος και σκύλος, με την ελάχιστη δυνατή απόσταση που έχει υπάρξει μέχρι και το προηγούμενο βήμα. Έτσι προκύπτει η παραπάνω αναδρομική σχέση.

---

**Algorithm 4** Function *Find Path*

---

```
1: function Function find_path( $n, m$ )
2:   if  $n = 1$  &  $m = 1$  then
3:      $D[1, 1] \leftarrow d(p[1], q[1])$ 
4:   else if  $n = 1$  &  $m \neq 1$  then
5:     if  $D[n, m - 1] = -1$  then
6:       call find_path( $n, m - 1$ )
7:     if  $D[n, m - 1] < d(p[n], q[m])$  then
8:        $D[n, m] \leftarrow d(p[n], q[m])$ 
9:     else
10:       $D[n, m] \leftarrow D[n, m - 1]$ 
11:   else if  $n \neq 1$  &  $m = 1$  then
12:     if  $D[n - 1, m] = -1$  then
13:       call find_path( $n - 1, m$ )
14:     if  $D[n - 1, m] < d(p[n], q[m])$  then
15:        $D[n, m] \leftarrow d(p[n], q[m])$ 
16:     else
17:        $D[n, m] \leftarrow D[n - 1, m]$ 
18:   else
19:     if  $D[n, m - 1] = -1$  then
20:       call find_path( $n, m - 1$ )
21:      $min \leftarrow D[n, m - 1]$ 
22:     if  $D[n - 1, m] = -1$  then
23:       call find_path( $n - 1, m$ )
24:     if  $D[n - 1, m] < min$  then
25:        $min \leftarrow D[n - 1, m]$ 
26:     if  $D[n - 1, m - 1] = -1$  then
27:       call find_path( $n - 1, m - 1$ )
28:     if  $D[n - 1, m - 1] < min$  then
29:        $min \leftarrow D[n - 1, m - 1]$ 
30:     if  $min < d(p[n], q[m])$  then
31:        $D[n, m] \leftarrow d(p[n], q[m])$ 
32:     else
33:        $D[n, m] \leftarrow min$ 
34:   return
```

---

Η υπολογιστική πολυπλοκότητα είναι ίση με  $\mathcal{O}(n^2)$  καθώς προκύπτει από το γινόμενο του statespace (μέγιστο δυνατό  $2 \cdot n^2$ ) με τον χρόνο υπολογισμού σε κάθε βήμα (σταθερό σύμφωνα με την εκφώνηση).



## Άσκηση 5: Αναβάθμιση του 242

(α)

Για την άσκηση αυτή θα χρησιμοποιήσουμε δυναμικό προγραμματισμό με την παρακάτω αναδρομική σχέση.

$$C(l, r, p) = \begin{cases} \min \begin{cases} C(l-1, r, 1) + (d_l + d_r) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l-1] + sr[r]) \\ C(l-1, r, 0) + (d_l - d_{l-1}) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l-1] + sr[r]) \end{cases} & \text{if } p = 0 \\ \min \begin{cases} C(l, r-1, 1) + (d_r - d_{r-1}) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l] + sr[r-1]) \\ C(l, r-1, 0) + (d_l + d_r) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l] + sr[r-1]) \end{cases} & \text{if } p = 1 \end{cases}$$

με αρχικές συνθήκες:

- $C(1, 1, 0) = C(0, 1, 1) + d_{r_1} \cdot \frac{50}{60 \cdot 30} \cdot (sr[1] + sl[0])$
- $C(1, 1, 1) = C(1, 0, 0) + d_{l_1} \cdot \frac{50}{60 \cdot 30} \cdot (sl[0] + sr[1])$
- $C(0, 1, 1) = d_{r_1} \cdot \frac{50}{60 \cdot 30} \cdot (sl[0] + sr[0])$
- $C(1, 0, 0) = d_{l_1} \cdot \frac{50}{60 \cdot 30} \cdot (sl[0] + sr[0])$

Η τελική βέλτιστη λύση προκύπτει από το

$$\min = \begin{cases} C(l, r, 1) \\ C(l, r, 0) \end{cases}$$

Οι αρχικές συνθήκες αυτές προκύπτουν από τον περιορισμό της δημιουργίας της πρώτης κίνησης προς τα δεξιά και τα αριστερά μόνο με ένα τρόπο.

Για αρχή μετατρέπουμε το πρόβλημα: θέτουμε ως μηδέν το σημείο εκκίνησης και υπολογίζουμε την απόσταση κάθε σημείου από το κέντρο, αποθηκεύοντας τον ως θετικό αριθμό. Τα σημεία που βρίσκονται αριστερά του σημείου  $k$ , συμβολίζονται με το γράμμα  $l$  και τα σημεία που βρίσκονται δεξιά με το γράμμα  $r$ . Παρατηρώντας το πρόβλημα, κάθε χρονική στιγμή, μας αφορά μόνο ποιο διάστημα έχουμε καλύψει και αν βρισκόμαστε αριστερά ή δεξιά. Επομένως το statespace μας περιέχει μια επιπλέον παράμετρο  $p$  με τιμή 0 αν βρισκόμαστε αριστερά και 1 αν βρισκόμαστε δεξιά.

Ξεκινώντας από το 0 (σημείο  $k$ ) μπορούμε είτε να πάμε αριστερά είτε δεξιά και κάθε φορά η επιλογή έχει κάποιο κόστος το οποίο ισούται με την απόσταση που διανύεται στο συγκεκριμένο βήμα, πολλαπλασιασμένη με την σταθερά  $\frac{50}{30 \cdot 60}$  (προκύπτει από τα δεδομένα) και με τον αριθμό των φοιτητών που βρίσκονται μέσα στο λεωφορείο. Θεωρούμε πως έχουμε ήδη υπολογίσει τον αριθμό αυτόν των φοιτητών και αποθηκεύσει στους πίνακες  $sr$  και  $sl$  τον αριθμό των φοιτητών που βρίσκονται στο λεωφορείο και σκοπεύουν να κατέβουν σε μία από τις επόμενες στάσεις από δεξιά και αριστερά αντίστοιχα.

---

**Algorithm 5** Function *Find route*

---

```
1: function Function find_route( $l, r, p$ )
2:   if  $l = 1 \ \& \ r = 0 \ \& \ p = 0$  then
3:      $C[1, 0, 0] \leftarrow dl[1] \cdot \frac{50}{60 \cdot 30} \cdot (sl[0] + sr[0])$ 
4:   else if  $l = 0 \ \& \ r = 1 \ \& \ p = 1$  then
5:      $C[0, 1, 1] \leftarrow dr[1] \cdot \frac{50}{60 \cdot 30} \cdot (sl[0] + sr[0])$ 
6:   else if  $l = 1 \ \& \ r = 1 \ \& \ p = 0$  then
7:     if  $C[0, 1, 1] = -1$  then
8:       call find_route(0, 1, 1)
9:      $C[1, 1, 0] \leftarrow C[0, 0, 1] + (dr[1]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[0] + sr[1])$ 
10:  else if  $l = 1 \ \& \ r = 1 \ \& \ p = 1$  then
11:    if  $C[1, 0, 0] = -1$  then
12:      call find_route(1, 0, 0)
13:     $C[1, 1, 1] \leftarrow C[1, 0, 0] + (dl[1]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[1] + sr[0])$ 
14:  else if  $p = 0$  then
15:    if  $C[l - 1, r, 1] = -1$  then
16:      call find_route( $l - 1, r, 1$ )
17:     $C[l, r, 0] \leftarrow C[l - 1, r, 1] + (dl[l] + dr[r]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l - 1] + sr[r])$ 
18:    if  $C[l - 1, r, 0] = -1$  then
19:      call find_route( $l - 1, r, 0$ )
20:    if  $C[l - 1, r, 0] + (dl[l] - dl[l - 1]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l - 1] + sr[r]) < C[l, r, 1]$  then
21:       $C[l, r, 0] \leftarrow C[l - 1, r, 0] + (dl[l] - dl[l - 1]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l - 1] + sr[r])$ 
22:  else
23:    if  $C[l, r - 1, 1] = -1$  then
24:      call find_route( $l, r - 1, 1$ )
25:     $C[l, r, 1] \leftarrow C[l, r - 1, 1] + (dr[r] - dr[r - 1]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l] + sr[r - 1])$ 
26:    if  $C[l, r - 1, 0] = -1$  then
27:      call find_route( $l, r - 1, 0$ )
28:    if  $C[l, r - 1, 0] + (dl[l] + dr[r]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l] + sr[r - 1]) < C[l, r, 0]$  then
29:       $C[l, r, 0] \leftarrow C[l, r - 1, 0] + (dl[l] + dr[r]) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l] + sr[r - 1])$ 
30:  return
```

---

Και πάλι έχουμε  $\mathcal{O}(n^2)$  καθώς προκύπτει από το γινόμενο του statespace ( $2 \cdot n^2$ ) με τον χρόνο υπολογισμού σε κάθε βήμα (σταθερό - απλές πράξεις).

(β)

Αν θεωρήσουμε  $(l,r,t)$  ως state σύμφωνα με το οποίο βρισκόμαστε την χρονική στιγμή στη στάση  $l$  και έχουμε ταξιδέψει έως και την  $r$  η σχέση που προκύπτει θα είναι η ακόλουθη:

$$C(l, r, t) = \begin{cases} \infty & \text{if } t > dl \\ \min \left\{ \begin{array}{l} C(r, l-1, t-t[r, l]) + (d_l + d_r) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l-1] + sr[r]) \\ C(l-1, r, t-t[l-1, l]) + (d_l - d_{l-1}) \cdot \frac{50}{60 \cdot 30} \cdot (sl[l-1] + sr[r]) \end{array} \right\} & \text{otherwise} \end{cases}$$

Η βέλτιστη λύση θα προκύψει από το  $\min$  των παραπάνω για όλα τα πιθανά

$$t = 0, 1, \dots, maximum\_deadline$$

. Η σχέση θα μπορούσε να γραφεί όπως και στο προηγούμενο ερώτημα αλλά στην προκειμένη αρκούμαστε στην αντιμετάθεση των  $l$  με  $r$  (δεν χρησιμοποιούμε παράμετρο  $p$ ). Επιπλέον χρησιμοποιείται ο πίνακας  $t$  που δείχνει το χρόνο μέχρι και την συγκεκριμένη απόσταση (minimum) που ισοδυναμεί με την ταχύτητα επί την απόσταση. Η πολυπλοκότητα είναι  $\mathcal{O}(mnD)$ , δηλαδή ψευδοπολυωνυμική.