Καμπουγέρης Χαράλαμπος Α.Μ: el20098

1η Σειρά Γραπτών Ασκήσεων 2023-2024

Άσκηση 1: Πλησιέστερο Ζεύγος Σημείων

a) Έστω P το σύνολο των n σημείων του χώρου που μας δίνονται. Θέλουμε κάθε αναδρομική κλήση να διαθέτει μια ταξινόμηση κατά x,y,z. Συνεπώς πριν ξεκινήσει η αναδρομή δημιουργούμε λίστες P_x, P_y, P_z, όπου P_i λίστα των στοιχείων ταξινομημένων σε αύξουσα σειρά κατά i-συντεταγμένη. Η ταξινόμηση έχει κόστος $O(n\log n)$.

Θεωρούμε επίπεδο L το οποίο χωρίζει «στα δύο» το χώρο και ορίζουμε τις 6 λίστες: $Q_x, R_x, Q_y, R_y, Q_z, R_z$ όπου η Q_i περιέχει τα σημεία που βρίσκονται αριστερά του L και R_i τα σημεία που βρίσκονται δεξιά του L, ταξινομημένα σε αύξουσα σειρά κατά την i-συντεταγμένη. Ο σχηματισμός αυτός γίνεται σε χρόνο $O(n)$ διατρέχοντας τις λίστες P_x, P_y, P_z.

Θεωρούμε ότι στο τέλος της αναδρομής γνωρίζουμε ένα πλησιέστερο ζεύγος σημείων στον αριστερό χώρο (Q), έστω q_0, q_1. Όπως γνωρίζουμε ένα πλησιέστερο ζεύγος r_0, r_1 στο R. Θεωρούμε $\delta = \min\{d(q_0,q_1), d(r_0,r_1)\}$.

Μένει να εξετάσουμε τι συμβαίνει με ζεύγη σημείων εκ των οποίων το ένα βρίσκεται στον Q και το άλλο στον R. Αρκεί να περιορίσουμε την αναζήτηση μας σε περιοχή απόστασης δ από το επίπεδο L.

Έστω S το σύνολο των σημείων αυτών. Αν υπάρχουν σημεία s, s' με $d(s,s') < \delta$ τότε αυτά θα ανήκουν στο S. Δημιουργούμε λίστες S_y, S_z, δηλαδή λίστες με τα σημεία του S ταξινομημένα ως προς y και z αντίστοιχα. Για τη δημιουργία αυτών των λιστών, αρκεί να διασχίσουμε μια φορά τις λίστες P_y και P_z, σε χρόνο $O(n)$.

Αρκεί να αποδείξουμε ότι κάθε σημείο στο σύνολο S αρκεί να εξεταστεί για ελάχιστη απόσταση με σταθερό αριθμό σημείων. Χωρίζουμε το σύνολο S σε κύβους ακμής δ/2.

Κάθε κύβος περιέχει το πολύ ένα σημείο, διότι αν περιείχε 2 αυτά θα είχαν απόσταση το πολύ $\sqrt{3}\,\delta/2$, όσο δηλαδή η διαγώνιος του, ... άτοπο γιατί $\sqrt{3}\,\delta/2 < \delta$. Ισχυριζόμαστε ότι κάθε σημείο στο S αρκεί να ελεγχθεί με τα επόμενα 63 σημεία του ως προς το y και τα επόμενα 63 ως προς το z.

Απόδειξη ισχυρισμού

Έστω ότι τα σημεία s, s' με $d(s,s') < \delta$ χωρίζονται από τουλάχιστον 64 κουτιά. Τότε θα υπάρχουν τουλάχιστον 3 σειρές του S μεταξύ τους, συνεπώς αυτά θα έχουν απόσταση τουλάχιστον $3\delta/2 > \delta$, άτοπο.

Τελικά κάθε σημείο του S, αρκεί να ελεγχθεί με τα 63 επόμενα του ως προς y και με τα 63 επόμενα του ως προς z, σε χρόνο $O((63+63)n) = O(n)$, που επιτυγχάνεται με προσπέλαση των λιστών S_y, S_z.

Τέλος, ο αλγόριθμος θα κρατήσει τη μικρότερη από τις αποστάσεις των σημείων S, την συγκρίνει με το δ και επιστρέφει τη μικρότερη.

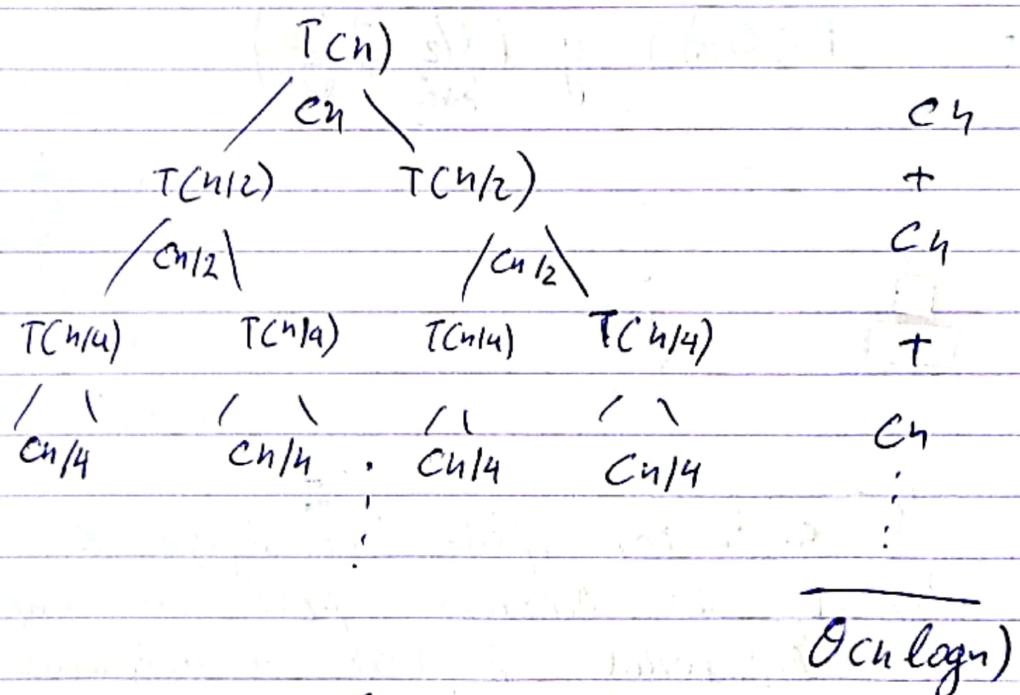
Απόδειξη Ορθότητας

Η ορθότητα έγκειται στο γεγονός ότι θα εξεταστούν για ελάχιστη απόσταση και σημεία που βρίσκονται στον ίδιο ημίχωρο αλλά και σημεία που βρίσκονται σε διαφορετικό.

Ανάλυση Πολυπλοκότητας.

Έστω $T(n)$ ο χρόνος για την εύρεση της ελάχιστης απόστασης. Σε κάθε αναδρομή ~~κοβο~~ μοιράζουμε τα σημεία σε $n/2$ και $n/2$, οπότε $T(n/2)$ ο χρόνος για την εύρεση ελάχιστης απόστασης στον αριστερό υπόχωρο και $T(n/2)$ για τον δεξί αντίστοιχα. Επίσης ο αλγόριθμος εκτελεί λειτουργίες που κοστίζουν γραμμικό χρόνο για την εξέταση ζευγών εκατέρωθεν του επιπέδου L.

Συνεπώς η αναδρομική σχέση: $T(n) \le 2\cdot T(n/2) + \Theta(n)$ περιγράφει τον αλγόριθμο.



Με βάση το δέντρο αναδρομής που φαίνεται παραπάνω προκύπτει χρόνος εκτέλεσης $O(n\log n)$.

β) Αρχικά διατυπώνουμε λύση του προβλήματος για τις 2 διαστάσεις και στη συνέχεια θα γενικεύσουμε.

Γνωρίζουμε τις συντεταγμένες των n σημείων ως προς x, y. Δηλαδή έχουμε $(x_1, x_2, ..., x_n)$ και

$(y_1, y_2, \ldots, y_n)$. Εφόσον γνωρίζουμε την προσέγγιση του δ^* με κάτω όριο το l θα κατανείμουμε τα σημεία σε buckets ως προς x και y διαιρώντας την κάθε συντεταγμένη με $\frac{l}{2}$, όπου d η διάσταση

Έχουμε τα εξής buckets:

→ $x: [0, l/2)$, $y: [0, l/2)$, δηλαδή περιέχει το σημείο με x οπωσδήποτε στο διάστημα $[0, l/2)$ και y οπωσδήποτε στο διάστημα $[0, l/2)$

→ $x: [0, l/2)$ $y: [\frac{l}{2}, l)$

⋮

Με αυτόν τον τρόπο εξασφαλίζουμε ότι σε κάθε bucket θα περιέχεται μόνο ένα σημείο, γιατί αν περιέχονταν 2 τότε η απόσταση τους θα ήταν το πολύ όσο η διαγώνιος ενός κουτιού, δηλαδή

$\frac{\sqrt{2}l}{2} < l$ ... , άτοπο

- Τα buckets φτιάχνονται σε γραμμικό χρόνο, γιατί αρκεί ένα πέρασμα στη λίστα σημείων.

Θεωρούμε ότι υπάρχει σημείο στο bucket $x: [0, l/2)$, $y: [0, l/2)$

Έστω ότι εξετάζουμε το σημείο στο bucket $x: [0, l/2)$, $y: [0, l/2)$. Εφόσον σε κάθε bucket υπάρχει το πολύ ένα σημείο, το σημείο αυτό πρέπει να εξετασεί με σημεία που απέχουν

x-απόσταση έως $c \cdot l$ και y-απόσταση έως $c \cdot l$.

Συνεπώς το σημείο πρέπει να εξετασεί με $4c \cdot 4c = 16c^2$ σημεία. (Εξετάζουμε $2c$ αριστερά, $2c$ δεξιά, $2c$ πάνω, $2c$ κάτω).

Επαναλαμβάνοντας για n σημεία προκύπτει πολυπλοκότητα $O(n \cdot 16c^2)$

Γενικεύοντας για 3-διαστάσεις, πλέον τα buckets θα περιέχουν και 2-συντεταγμένα.
Τα buckets τώρα σχηματίζονται ως κύβοι ακμής $l/2$, όπου και πάλι κάθε κύβος περιέχει το πολύ ένα σημείο (διαγώνιος κύβου $\frac{\sqrt{3}\,l}{2} < l$).

Τώρα κάθε σημείο θα εξετασεί με $4c \cdot 4c \cdot 4c = 64c^3$ σημεία, άρα έχουμε πολυπλοκότητα $O(n \cdot 64c^3)$

- Γενικεύοντας για $d \geq 2$ διαστάσεις, αρκεί κάθε φορά να επιλέγουμε κατάλληλη ακτίνα του bucket ώστε να χωράει το πολύ 1 σημείο μέσα (π.χ $\frac{l}{2\sqrt{d}}$). Αρκεί μετά να υπολογίσουμε όλα τα σημεία που απέχουν απόσταση μικρότερη από cl. Όπως είδαμε ο αριθμός αυτός είναι πεπερασμένος και ανάλογος του c και του d.

Με βάση την παραπάνω ανάλυση η υπολογιστική πολυπλοκότητα είναι $O(c^d n)$.

Άσκηση 2: Πόρτες Ασφαλείας στο Κάστρο

Θεωρούμε ότι αρχίζουμε με όλους τους διακόπτες ανοιχτούς (τυχαίος συνδυασμός), δηλαδή έστω στο 0 (1 για κλειστό).
Ο αλγόριθμος που θα ακολουθήσουμε βασίζεται στη δυαδική αναζήτηση.
Αρχικά, πρέπει να βρούμε το διακόπτη και τη θέση για την πρώτη πόρτα.
Αντιστρέφουμε τους διακόπτες του πρώτου μισού, → δοκιμάζουμε την εξής ακολουθία:

$$\overbrace{11\ldots1}^{n/2}\overbrace{00\ldots0}^{n/2}$$

Αν η πρώτη πόρτα άνοιξε (αν ήταν κλειστή) ή αν έκλεισε (αν ήταν ανοιχτή) συνεχίζουμε την αναζήτηση στο αριστερό μισό αντιστρέφοντας ξανά τα πρώτα μισά, δηλαδή δοκιμάζουμε την ακολουθία:

$$\overbrace{00\ldots0}^{n/4}\overbrace{11\ldots1}^{n/4}\overbrace{00\ldots0}^{n/2}$$

Αν το κλείδωμα την δεν άλλαξε συνεχίζουμε στο δεξί μισό :

Συνεχίζουμε αναδρομικά τη διαδικασία μέχρι να βρούμε τον διακόπτη και τη θέση του, που ανοίγει την πρώτη πόρτα.
Επαναλαμβάνουμε τη διαδικασία για όλες τις πόρτες (n). Λόγω χωρισμού των k διακοπτών κάθε φορά στα 2 σε κάθε βήμα της αναζήτησης, έχουμε χρόνο $\log(k)$, ο οποίος είναι και βέλτιστος όπως προκύπτει απ'το δέντρο αναζητήσεων. Η διαδικασία γίνεται n φορές, καθώς δεν μπορούμε να δούμε πίσω από πόρτες και άρα να χωρίσουμε το πρόβλημα σε υπό-προβλήματα.

Η πολυπλοκότητα είναι $\lceil \log_2 n \rceil + \lceil \log_2 (n-1) \rceil + \dots + 1 = \Theta(\log_2 n!) = \Theta(n \log n)$

Άσκηση 3: Κρυμμένος Θησαυρός

Περιγραφή αλγορίθμου

Στο i-οστό βήμα πηγαίνουμε από τη θέση που είμαστε στη θέση a^i με ακο. Για $i = 2k$:

$$a^i = a^{2k} > 0$$
$$a^{i+1} = a \cdot a^{2k} < 0$$

Για $k \to \infty$ οι ποσότητες στο $+\infty$ και $-\infty$ αντίστοιχα

Άρα υπάρχει k ώστε: $a^{i+1} \leq x \leq a^i$. Οπότε ο θησαυρός θα βρεθεί.

Υπολογισμός της συνολικής απόστασης που διανύουμε

Στο βήμα i_{final}, ξεκινάμε από τα $a^{i_{final}-1}$ φτάνουμε στο 0 και έπειτα διανύουμε απόσταση x μέχρι να σταματήσουμε. Οπότε μπορούμε να πούμε ότι

$$|a|^{i_{final}} \leq |a||x|. \quad (1)$$

Άρα η συνολική απόσταση που διανύουμε είναι:

$$2|a| + 2|a|^2 + \dots + 2 \cdot |a|^{i_{final}-1} + x = 2 \cdot |a| \cdot \frac{|a|^{i_{final}} - 1}{|a| - 1} + x$$

$$\overset{(1)}{\leq} 2|a| \cdot \frac{|a| \cdot |x| - 1}{|a| - 1} + x \leq 2|a| \frac{|a| \cdot |x|}{|a| - 1} + x \leq$$

$$\leq \left(\frac{2 \cdot |a|^2}{|a| - 1} + 1 \right) |x|$$

Οπότε έχουμε φτάσει στη μορφή $c|x|$ με

$$c = \frac{2|a|^2}{|a|-1} + 1 .$$

Αρκεί να ελαχιστοποιήσουμε τη συνάρτηση $c = \frac{2a^2}{a-1} + 1$, με $a > 1$

$$\frac{dc}{da} = \frac{2a(a-2)}{(a-1)^2} , \quad \frac{dc}{da} = 0 \Leftrightarrow a = 2$$

$$\frac{d^2c}{da^2} = \frac{4}{(a-1)^3} \qquad \text{Για } a=2: \left.\frac{d^2c}{da^2}\right|_{a=2} = 4>0$$

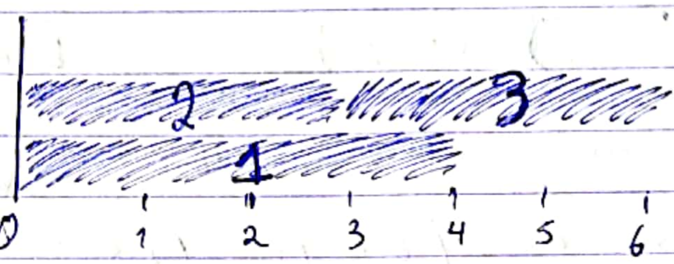
Οπότε η συνάρτηση παρουσιάζει ελάχιστο για $a=2$ το $c(2) = 9$.

Οπότε η βέλτιστη απόσταση επιτυγχάνεται για βάση $a = 2$, και στη χειρότερη περίπτωση θα διανύσουμε $9|x|$ βήματα για να βρούμε το θησαυρό.

Άσκηση 4: Μη Επικαλυπτόμενα Διαστήματα Μεγίστου Συνολικού Μήκους

1) α)

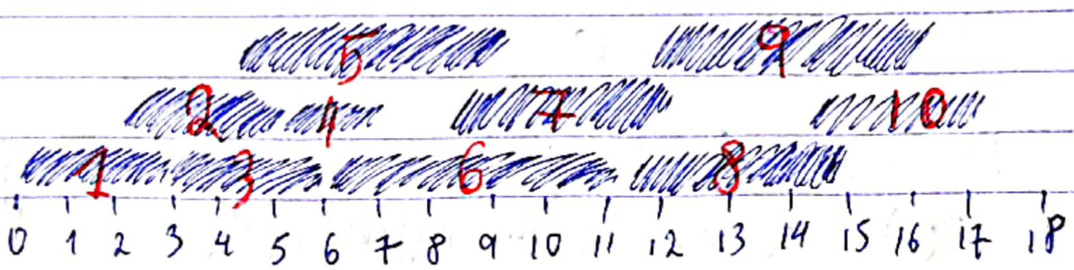
i	1	2	3
s_i	0	0	3
f_i	4	3	6



Αν χρησιμοποιήσουμε για κριτήριο επιλογής το διαθέσιμο διάστημα $[s_i, f_i)$ με μέγιστο μήκος $(f_i - s_i]$ τότε δεν οδηγούμαστε στη βέλτιστη λύση, καθώς βρίσκει ως βέλτιστη λύση το εξής: $[0,4)$ ενώ η βέλτιστη λύση είναι: $[0,3)$, $[3,6)$.

β) Από διαφάνειες του μαθήματος παίρνουμε:

i	1	2	3	4	5	6	7	8	9	10
s_i	0	2	3	5	4	6	8	11	11	14
f_i	3	5	6	7	9	11	12	15	16	17



Αν χρησιμοποιήσουμε για κριτήριο επιλογής το διαθέσιμο διάστημα $[s_i, f_i)$ με ελάχιστο χρόνο ολοκλήρωσης f_i, τότε δεν οδηγούμαστε στη βέλτιστη λύση καθώς βρίσκει ως βέλτιστη την εξής:

$[0,3), [3,6), [6,11), [11,15)$ Ενώ η βέλτιστη

είναι: $[0,3), [3,6), [6,11), [11,16)$.

2)

$$A = \{[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n)\},$$

$$w_1 \quad w_2 \quad \ldots \quad w_n$$

ταξινομημένα τα με: $f_1 < f_2 < \ldots < f_n$

και για τη λύση: όπου OPT το βέλτιστο.

$$\max \begin{cases} w_n + OPT(p(n)) & \text{όπου προσθέτω το } [s_n, f_n) \\ OPT(n-1) & \text{όπου το αγνοώ} \end{cases}$$

μέγιστο βάρος των n πρώτων διαστημάτων. $p(n) = \arg\max_j \{f_j \leq s_n\}$

$$OPT(i) = \max \begin{cases} w_i + OPT(p(i)) \\ OPT(i-1) \end{cases}$$

τελευταίο διάστημα που δεν επικαλύπτεται με το $[s_i, f_i)$

Με $OPT(0) = 0$

Με χρόνο $O(n\log n)$ αν χρειάζεται ταξινόμηση, ενώ $O(n)$ αν είναι ήδη ταξινομημένα.

Άσκηση 5: Παραλαβή Πακέτων

Αλγόριθμος

i) Θα λύσουμε το πρόβλημα με greedy αλγόριθμο. Στην άπληστη λύση ταξινομούμε τα πηλίκα p_i/w_i κατά αύξουσα σειρά και επιλέγουμε κάθε φορά το πακέτο με το μικρότερο πηλίκο για εξυπηρέτηση. Χωρίς βλάβη της γενικότητας υποθέτουμε ότι μετά από ταξινόμηση επαναπροσδιορίζουμε τους δείκτες i ώστε να ισχύει

$$p_1/w_1 \leq p_2/w_2 \leq \dots \leq p_n/w_n.$$

Απόδειξη ορθότητας

Ο greedy αλγόριθμος επιλέγει πρώτο για εξυπηρέτηση το δέμα του 1ου πελάτη. Ο συνολικός βεβαρυμένος χρόνος εξυπηρέτησης ισούται με:

$$G_1 = w_1 p_1 + w_2(p_1+p_2) + \dots + w_n(p_1+p_2+\dots+p_n) \quad (1)$$

Έστω ότι σε κάποιο βήμα, χωρίς βλάβη γενικότητας στο πρώτο, ο αλγόριθμος επιλέγει το 2ο δέμα αντί για το πρώτο. Ο χρόνος θα είναι:

$$G_2 = w_2 p_2 + w_1(p_1+p_2) + \dots + w_n(p_1+p_2+\dots+p_n) \quad (2)$$

$$(2)-(1) \Rightarrow G_2 - G_1 = w_1(p_1+p_2-p_1) - w_2(p_1+p_2-p_2) =$$

$$= w_1 p_2 - w_2 p_1 \geq 0, \quad \text{γιατί ισχύει } \frac{p_1}{w_1} \leq \frac{p_2}{w_2}.$$

Συνεπώς η αλλαγή οδηγεί σε αύξηση του χρόνου εξυπηρέτησης και επομένως ο greedy αλγόριθμος είναι

Βέλτιστος. Κατά τον ίδιο τρόπο μπορούμε να δείξουμε ότι η εναλλαγή οποιονδήποτε δύο διαδοχικών πελατών i, j με $p_i/w_i \leq \frac{p_j}{w_j}$ και εξυπηρέτηση πρώτα του j, οδηγεί σε αύξηση του χρόνου.

Πολυπλοκότητα

Ο αλγόριθμος ταξινομεί τους πελάτες κατά p_i/w_i.

Λόγω της ταξινόμησης η πολυπλοκότητα είναι $O(n \log n)$

2) Για αυτήν την υποκατηγορία του προβλήματος το άπληστο κριτήριο δε θα δουλέψει, καθώς αν ταξινομούμε κατά αύξουσα σειρά δεν γνωρίζουμε αν είναι βέλτιστο, το δέμα του πελάτη i να τοποθετηθεί στον 1ο ή στον 2ο εξυπηρετητή. Συνεπώς θα χρησιμοποιήσουμε δυναμικό προγραμματισμό και θα αποφασίζουμε για κάθε δέμα αν θα πάει στον πρώτο ή στον 2ο εξυπηρετητή.

Ταξινομούμε τα δέματα με βάση την ταξινόμηση του ερωτήματος (1) και εξετάζουμε το n-οστό δέμα. Έστω C ο ελάχιστος βεβαρυμένος χρόνος εξυπηρέτησης. Για το C χρειαζόμαστε 2 παραμέτρους, το πλήθος των δεμάτων και το μέσο χρόνο εξυπηρέτησης στον πρώτο εξυπηρετητή, t_1, μόλις δρομολογηθεί το n-οστό πακέτο. Επίσης ορίζουμε t_2 το μέσο χρόνο εξυπηρέτησης στον δεύτερο εξυπηρετητή.

Ισχύει ότι $t_1 + t_2 = \sum_{i=1}^{n} p_i$.

Για το n-οστό αντικείμενο έχουμε 2 επιλογές

→ Θα ανατεθεί στον 1º εξυπηρετητή συνεπώς ο ελάχιστος χρόνος θα προκύψει από τον ελάχιστο χρόνο για n-1 αντικείμενα με μέγιστο χρόνο εξυπηρέτησης στον 1º εξυπηρετητή $t_1 - p_n$, και το βεβαρυμένο χρόνο του n-οστού δέματος, $w_n t_1$.

→ Δε θα ανατεθεί στον 1º εξυπ. αλλά στον 2º. Ο ελάχιστος χρόνος προκύπτει από τον ελάχιστο χρόνο για n-1 αντικείμενα με μέγιστο χρόνο εξυπηρέτησης στον 1º εξυπηρ. t_1 και τον βεβαρυμένο χρόνο του n-οστού δέματος, $w_n t_2$

Οπότε έχουμε της εξής σχέση για τον βεβαρυμένο χρόνο εξυπηρέτησης για n πελάτες

$$C(n, t_1) = \min \begin{cases} C(n-1, t_1 - p_n) + w_n t_1 \\ C(n-1, t_1) + w_n t_2 \end{cases}$$

Η παραπάνω σχέση γενικεύεται για οποιοδήποτε πακέτο i.

Απόδειξη Ορθότητας

Ο αλγόριθμος σε κάθε βήμα αποφασίζει με βέλτιστο τρόπο για το αν το δέμα i τοποθετηθεί στον 1º ή στο 2º εξυπηρετητή, εξετάζοντας όλες τις περιπτώσεις. Έτσι, επιτυγχάνεται η βέλτιση λύση στο πρόβλημα

Ανάλυση Πολυπλοκότητας

Η πολυπλοκότητα του αλγορίθμου είναι $O(n t_1)$. Επίσης ισχύει

$t_1 + t_2 = \sum_{i=1}^{n} p_i$, οπότε είναι $O(n \sum_{i=1}^{n} p_i)$

Γενίκευση για $m \geq 3$ υπαλλήλους

Για κάθε θέση έχουμε m αποφάσεις. Η αναδρομική σχέση για το συνολικό βεβαρυμένο χρόνο εξυπηρέτησης γενικεύεται:

$$C(n, t_1) = \min \begin{cases} C(n-1, t_1 - p_n) + w_n t_1 \\ C(n-1, t_1) + w_n t_2 \\ \vdots \\ C(n-1, t_1) + w_n t_m \end{cases}$$

Η πολυπλοκότητα θα είναι $O(n (\sum_{i=1}^{n} p_i)^{m-1})$