ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ/ΚΩΝ & ΜΗΧ/ΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μάθημα: "Αλγόριθμοι και Πολυπλοκότητα"

Λουκάς Άγγελος 03119877

Σειρά Ασκήσεων 1

1. Πλησιέστερο Ζεύγος Σημείων

α) Θα εργαστούμε με παρόμοιο τρόπο όπως και στο δυσδιάστατο πρόβλημα.

1. Προκειμένου να ταξινομήσουμε τα δεδομένα σύμφωνα με κάθε διάσταση, δημιουργούμε πρώτα τρία διατεταγμένα σύνολα στο χρόνο $O(nlogn)$: P_x, P_y και P_z. Υπολογίζουμε την επιφάνεια διαχωρισμού B, η οποία χωρίζει το σύνολο των σημείων στη μέση ανάλογα με το πού βρίσκεται κάθε σημείο στη διάσταση x, χρησιμοποιώντας το σύνολο Px.

2. Με βάση την διαχωριστική επιφάνεια B κατασκευάζουμε τα σύνολα L_x,L_y, L_z and R_x,R_y, R_z όπου στα σύνολα L_d παραμένουν μόνο στα στοιχεία αριστερά της B και στα σύνολα R_d τα στοιχεία δεξιά της B. Τα στοιχεία παραμένουν ταξινομημένα με βάση τους αντίστοιχους άξονες. Χρονική πολυπλοκότητα του βήματος αυτού: $O(n)$

3. Αναδρομικά υπολογίζουμε το ζητούμενο για κάθε ένα από τα υποσύνολα σε χρόνο $2T(n/2)$. Έστω δ_1, δ_2 οι ελάχιστες αποστάσεις δυο σημείων στα υποσύνολα αυτά και $\delta = \min(\delta 1, \delta 2)$.

4. Η λωρίδα (του δισδιάστατου προβλήματος) αντικαθιστάται με έναν παράλληλο V χώρο που απέχει δ από τη διαχωριστική λωρίδα L στην x διάσταση.

    - Σε χρόνο $O(n)$, χρησιμοποιώντας το σύνολο P_y, υπολογίζουμε το S_y που είναι το σύνολο των σημείων στον χώρο V ταξινομημένα κατά y. Ξεκινώντας από την αρχή, συγκρίνουμε το κάθε σημείο με τα 11 επόμενά του και αν κάποια απόσταση είναι μικρότερη από δ, ανανεώνουμε το δ. Οι συγκρίσεις θέλουν χρόνο $O(n)$.
    - Επαναλαμβάνουμε το προηγούμενο βήμα για τον άξονα z.

Ορθότητα του βήματος 4: Βασιζόμαστε στην ορθότητα του αλγορίθμου για τις δυο διαστάσεις. Πιο συγκεκριμένα, δημιουργούμε ένα grid στον V με ελάχιστη υποδιαίρεση κύβους πλευράς $\delta/2$. Συνεπώς, κάθε κύβος ορίζει έναν χώρο όπου μπορεί να πέσει το πολύ ένα σημείο. Αυτό σημαίνει ότι κάθε σημείο πρέπει να συγκριθεί με όλα τα στοιχεία που βρίσκονται εντός κύβου με κέντρο τον κύβο που ανήκει το σημείο αυτό και πλευρές μήκους $2\delta + \delta/2$. Αυτά είναι τα σημεία στα διατεταγμένα σύνολα Sy, Sz που απέχουν το πολύ 12 θέσεις μεταξύ τους. Καθώς η σύγκριση με τα προηγούμενα ολοκληρώθηκε στα προηγούμενα βήματα, περιοριζόμαστε στη σύγκριση των 11 πρώτων σημείων αυτών των συνόλων μεταξύ τους.

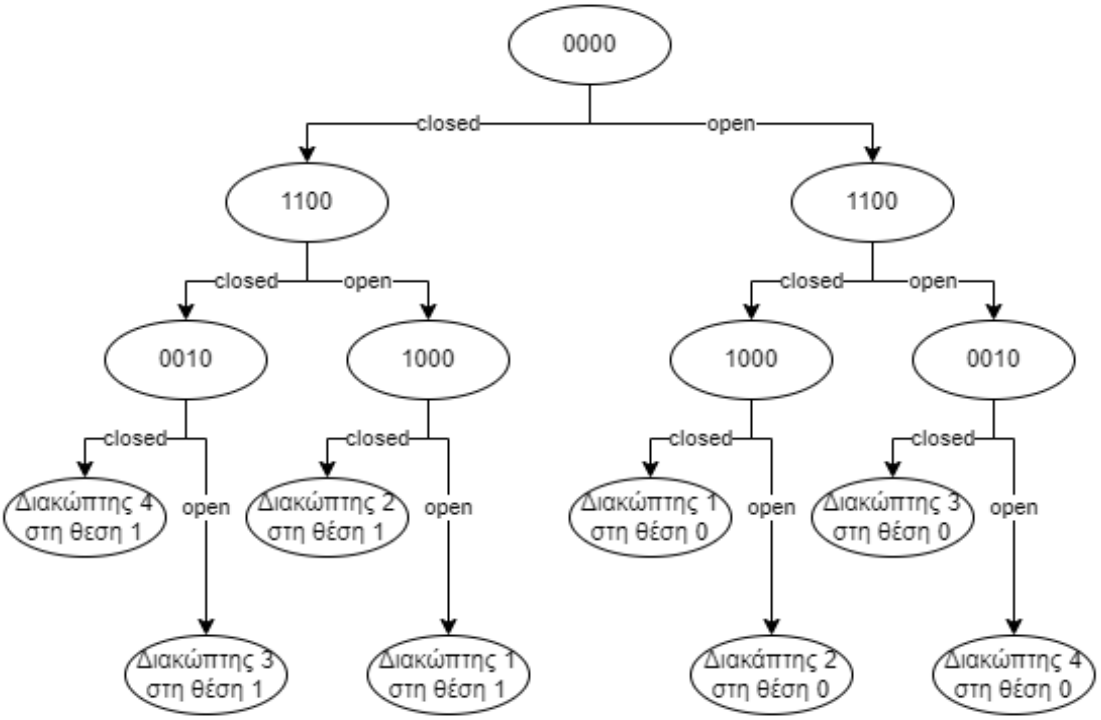
Ορθότητα. Έστω P το σύνολο των σημείων. Θα χρησιμοποιήσουμε επαγωγή στον πληθικό αριθμό του συνόλου. Για σύνολα με $|P| \leq 3$ η μικρότερη απόσταση υπολογίζεται μέσω υπολογισμού και σύγκρισής όλων των ανά-δυο αποστάσεων. Υποθέτουμε ότι τα δ1, δ2 υπολογίζονται ορθά από την αναδρομή. Στην συνέχεια έχουμε δύο περιπτώσεις:

- Η μικρότερη απόσταση βρίσκεται ανάμεσα σε σημεία που βρίσκονται και τα δυο στο ίδιο μισό του της διαχωριστικής επιφάνειας L. Σε αυτή την περίπτωση το αποτέλεσμα έχει ήδη υπολογιστεί από την αναδρομή και δεν αλλάζει από την διαδικασία του βήματος 3.
- Η μικρότερη απόσταση βρίσκεται ανάμεσα σε δύο σημεία που βρίσκονται σε διαφορετικές μεριές της L. Σε αυτή την περίπτωση η ορθότητα του αλγορίθμου βασίζεται στην ορθότητα του βήματος 3 που εξηγήθηκε παραπάνω.

Πολυπλοκότητα. Από Master Theorem: $T(n) \leq 2T\ (\ n/2\ ) + O(n) \Rightarrow \boldsymbol{T(n) = O(nlogn)}$

2. Πόρτες Ασφαλείας στο Κάστρο

Θα λύσουμε το πρόβλημα χρησιμοποιώντας την στρατηγική Decrease and Conquer. Η στρατηγική μπορεί να μοντελοποιηθεί με ένα δυαδικό δέντρο συγκρίσεων όπως τα παρακάτω:



Στο παράδειγμα αυτό χρησιμοποιήσαμε μόνο 4 πόρτες ασφαλείας. Υποθέτουμε πως αλλαγή διακοπτών από τον έναν κόμβο στον άλλον είναι στιγμιαία και ισούται με έναν έλεγχο αν η πόρτα παρέμεινε κλειστή ή όχι.

- «1» συμβολίζει την αντιστροφή της αρχικής θέσης του κάθε διακόπτη
- «0» είναι η αρχική θέση του κάθε διακόπτη.

Ξεκινώντας από της αρχική διάταξη όπου όλοι οι διακόπτες βρίσκονται στην θέση 0 μπορούμε να δούμε πως οι αλλαγές διακοπτών ισούνται με την μικρότερη ακέραια τιμή που υπερβαίνει το $\log_2(n)$ όπου n οι διακόπτες που ελέγχονται. Οι διακόπτες που έχουμε βρει ποια πόρτα ανοίγουν δεν χρειάζεται να μπαίνουν στην δοκιμή της επόμενης πόρτας.

Με την παραπάνω στρατηγική ο αριθμός των δοκιμών θα είναι :

$$\log 2+2\cdot\log 4+4\cdot\log 8+\ldots+\frac{n}{2\cdot logn}=\log\left(2\cdot 4^{2}\cdot\ldots\cdot n^{\frac{n}{2}}\right)=O\left(n\cdot logn\right)$$

3. Κρυμμένος Θησαυρός

Για την προσέγγιση του προβλήματος του κρυμμένου θησαυρού, ένας ενδιαφέρων τρόπος να μοντελοποιήσουμε την αναζήτηση μας είναι να χρησιμοποιήσουμε μια πολυωνυμική ακολουθία αποστάσεων, με κάθε καινούρια θέση $f_n = 2^n$. Η επιλογή αυτή οφείλεται στην ιδέα ότι προσφέρει έναν συστηματικό και αποτελεσματικό τρόπο για τη διερεύνηση ενός ευρέος φάσματος πιθανών θέσεων για τον θησαυρό. Επιπλέον, αυτή η ακολουθία παρουσιάζει μια ισορροπημένη εναλλαγή μεταξύ θετικών και αρνητικών βημάτων, αντικατοπτρίζοντας το ταξίδι μας στην αναζήτηση του μπρος-πίσω.
Θα θεωρήσουμε την χειρότερη περίπτωση όπου $d=f_n+c$ είναι η θέση του θησαυρού , για μία μικρή τιμή του c και με $n \geq 1$ που δηλώνει την n-οστή καινούρια θέση. Για να βρεθεί αυτή η θέση d θα χρειαστεί να μετακινηθούμε στην αντίθετη κατεύθυνση στην θέση f_{n+1} και ύστερα στην σωστή κατεύθυνση προς την θέση f_{n+2}. Η συνολική απόσταση θα είναι:

$$2\cdot\sum_{j=0}^{n+1}(f_j)+d=2\cdot\sum_{j=0}^{n+1}(2^j)+2^n+c$$

$$=2\cdot(2^{n+2}+1)+2^n+c \qquad \text{αφού} \quad \sum_{j=0}^{n-1}(2^j)+1=2^n$$

$$=2\cdot 2^{n+2}+2+2^n+c$$

$$=8\cdot 2^n+2+2^n+c$$

$$=9\cdot 2^n+2+c$$

$$\leqslant 9\cdot(2^n+c) \qquad \text{αφού } c\geqslant 1\Rightarrow 8\cdot c\geqslant 8\Rightarrow 9\cdot c\geqslant 8+c\Rightarrow 9\cdot c\geqslant 2+c$$

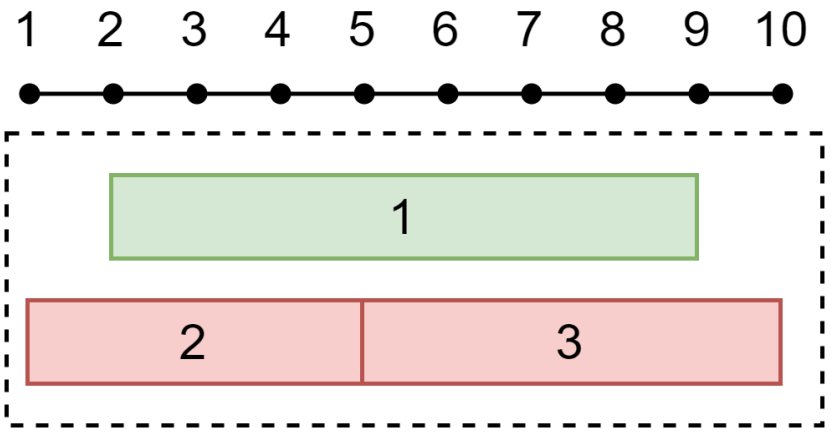
$$=9\cdot d$$

Βλέπουμε πως ο αλγόριθμός μας στην χειρότερη περίπτωση θα μας αναγκάσει να μετακινηθούμε 9 φορές την απόσταση του θησαυρού από την αρχική θέση.

4. Μη Επικαλυπτόμενα Διαστήματα Μέγιστου Συνολικού Μήκους

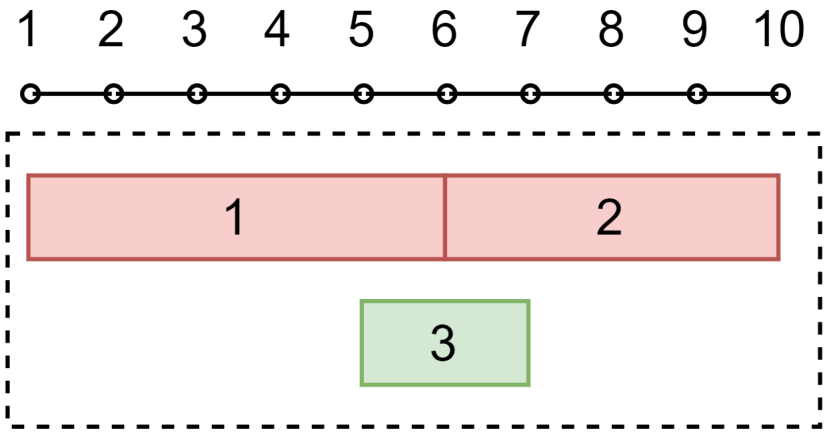
4.1

- Για τον άπληστο αλγόριθμο που επιλέγει κάθε φορά το **μεγαλύτερο** διαθέσιμο διάστημα μπορούμε να δώσουμε το επόμενο αντιπαράδειγμα.



Στην παραπάνω εικόνα μπορούμε να δούμε 3 διαστήματα. Διάστημα_1 (2,9) , Διάστημα_2 (1,5) και Διάστημα_3 (5,10). Θα γίνει ταξινόμηση αυτών των τριών διαστημάτων σε χρόνο O(nlogn) με βάση το μήκος των διαστημάτων και η τελική ταξινομημένη λίστα θα είναι : [Διάστημα_1,Διάστημα_3,Διάστημα_2].
Θα γίνει επιλογή αρχικά του Διάστημα_1 αλλά κανένα άλλο διάστημα δεν θα επιλεγεί καθώς υπάρχουν επικαλύψεις. Μπορούμε να δούμε λοιπόν πως ο αλγόριθμος αυτός δεν οδηγεί απαραίτητα σε σωστή λύση.

- Για τον άπληστο αλγόριθμο που επιλέγει κάθε φορά το **μικρότερο** διαθέσιμο διάστημα μπορούμε να δώσουμε το επόμενο αντιπαράδειγμα.

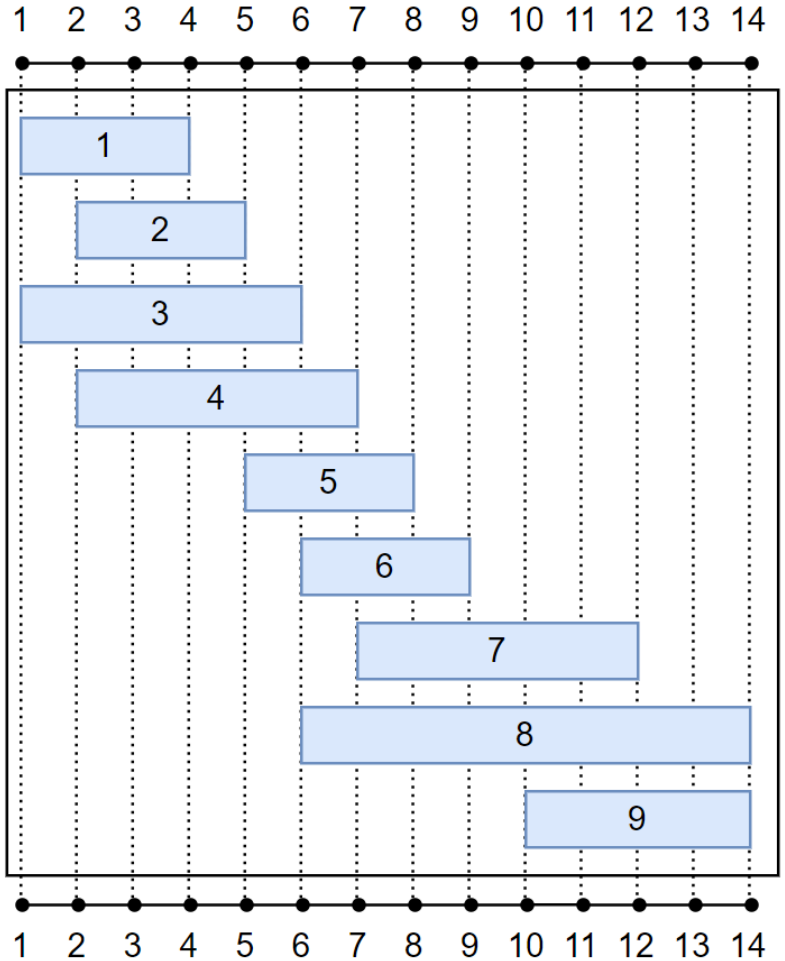


Στην παραπάνω εικόνα μπορούμε να δούμε 3 διαστήματα. Διάστημα_1 (1,6) , Διάστημα_2 (6,10) και Διάστημα_3 (5,7). Θα γίνει ταξινόμηση αυτών των τριών διαστημάτων σε χρόνο O(nlogn) με βάση το μήκος των διαστημάτων και η τελική ταξινομημένη λίστα θα είναι : [Διάστημα_3,Διάστημα_2,Διάστημα_1].
Θα γίνει επιλογή αρχικά του Διάστημα_3 αλλά κανένα άλλο διάστημα δεν θα επιλεγεί καθώς υπάρχουν επικαλύψεις. Μπορούμε να δούμε λοιπόν πως ο αλγόριθμος αυτός δεν οδηγεί απαραίτητα σε σωστή λύση.

4.2

Στο πρόβλημά μας έχουμε n διαστήματα όπου κάθε διάστημα j έχει μέγεθος $[s_j , f_j)$.

1. Αρχικά θα ταξινομήσουμε τα διαστήματα μας με αύξον τιμή την f_j όπως φαίνεται στο παρακάτω σχήμα (αν δύο διαστήματα έχουν την ίδια τιμή f_j τότε μπαίνει πρώτο εκείνο με την μικρότερη τιμή s_j) σε χρόνο O(nlogn).
2. Θα κατασκευάσουμε έναν πίνακα με στήλες που δηλώνουν τον αριθμό j του κάθε διαστηματός μας με βάση την σειρά που έχουν εισαχθεί στην ταξινομημένη μας λίστα.
3. Οι σειρές του πίνακά μας θα είναι :
   - **f_j-s_j** (το μήκος του διαστήματος)
   - **A(j)** (τον αριθμό του τελευταίου διαστήματος k όπου $f_k < f_j$)
   - **C(j)** (το μέγιστο συνολικό διάστημα μέχρι το διάστημα j)



4. Για τον υπολογισμό του C(j) : Το μέγιστο συνολικό μήκος με τελικό διάστημα το διάστημα j θα ισούται με το **max{ $f_j - s_j$ + C(A(j)) , C(j-1) }** . Το **$f_j - s_j$ + C(A(j))** είναι το μήκος του j συν το μέγιστο συνολικό μήκος των διαστημάτων που χωράνε να μπούνε πριν από το διάστημα j. Το συγκρίνουμε αυτό με το **C(j-1)** καθώς αν έχουμε βρει μεγαλύτερο μήκος για το προηγούμενο διάστημα θα κρατήσουμε εκείνο.

C(j)= max{ $f_j - s_j$ + C(A(j)) , C(j-1) }

	1	2	3	4	5	6	7	8	9
f_j-s_j	3	3	5	5	3	3	5	8	4
A(j)	0	0	0	0	2	3	4	3	6
C(j)	3	3	5	5	6	8	10	13	13

Με τον πίνακα γεμισμένο μπορούμε να βρούμε στην τελική τιμή C(n) το μέγιστο δυνατό μήκος για τα διαστήματά μας. Στο παράδειγμά μας η τελική τιμή ισούται με 13 (διάστημα_3 + διάστημα_8) . Για να βρούμε ποιά διαστήματα το καταφέρανε αυτό θα περάσουμε τις τιμές του C(j). Ξεκινώντας από το C(n) και κατεβαίνοντας πρός το C(1) . Αν βρούμε C(j) > C(j-1) για j στο [1,n] τότε κρατάμε την τιμή του j σε μια λίστα και συνεχίζουμε την αναζήτηση από το C(A(j)). Η λίστα αυτή κρατάει ολα τα διαστήματα που χρειαστήκανε για το μέγιστο συνολικό διάστημα.

5. Παραλαβή Πακέτων

5.1

Έστω:

	Βέλτιστος αλγόριθμος	**Άπληστος αλγόριθμος**
Ο χρόνος	T	T^*
Η σειρά παραλαβής	$\lambda(j)$	$\lambda^*(j)$
Ο βεβαρυμμένος χρόνος για όλα τα δέματα πριν το i – οστο στοιχείο	T_{i-}	T^*_{i-}
Ο βεβαρυμμένος χρόνος για όλα τα δέματα μετά το i – οστο στοιχείο	T_{i+}	T^*_{i+}
Το άθροισμα χρόνων προετοιμασίας για όλα τα δέματα πριν το i – οστο στοιχείο σε μία δρομολόγηση	C_i	C^*_i

Ο άπληστος μας αλγόριθμος:

- Θα κατασκευάσουμε μία λίστα (με όνομα WP) με τους λόγους w_i/p_i και ταξινόμηση των πακέτων αυτών σε φθίνουσα σειρά με βάση την λίστα αυτή.
- Η δρομολόγηση θα γίνει με την σειρά της ταξινόμησης στην παραπάνω μας λίστα άρα $\lambda^*(j)=WP(j)$

Απόδειξη της ορθότητας το άπληστου αλγόριθμου:

- Έστω μία βέλτιστη δρομολόγηση '
- Έστω πως υπάρχει $i_1<i_2$ τα οποία είναι δύο συνεχόμενα πακέτα για το οποία ισχύει $\lambda(i_1)>\lambda(i_2)$ και $\lambda^*(i_1)<\lambda^*(i_2)$
- λ^* =[… ,i_2,i_1,…]
- λ =[… ,i_1,i_2,…]
- Για την βέλτιστη δρομολόγηση θα έχουμε
  - $T^*=T^*_{i2-}+w_{i2}\cdot(C^*_{i2}+p_{i2})+w_{i1}\cdot(C^*_{i2}+p_{i2}+p_{i1})+T^*_{i1+}$
  - $T=T_{i1-}+w_{i1}\cdot(C_{i1}+p_{i1})+w_{i2}\cdot(C_{i1}+p_{i2}+p_{i1})+T_{i2+}$
- Όμως ισχύει πως :
  - $T^*_{i2-}=T_{i1-}$
  - $T^*_{i1+}=T_{i2+}$
  - $C^*_{i2}=C_{i1}$
- Άρα:

$$T^*-T=$$
$$=T^*_{i2-}+w_{i2}\cdot(C^*_{i2}+p_{i2})+w_{i1}\cdot(C^*_{i2}+p_{i2}+p_{i1})+T^*_{i1+}-T_{i1-}-w_{i1}\cdot(C_{i1}+p_{i1})-w_{i2}\cdot(C_{i1}+p_{i2}+p_{i1})-T_{i2+}$$
$$=w_{i2}\cdot(C^*_{i2}+p_{i2})+w_{i1}\cdot(C^*_{i2}+p_{i2}+p_{i1})-w_{i1}\cdot(C_{i1}+p_{i1})-w_{i2}\cdot(C_{i1}+p_{i2}+p_{i1})$$
$$=w_{i2}\cdot(C^*_{i2}+p_{i2}-C_{i1}-p_{i2}-p_{i1})+w_{i1}\cdot(C^*_{i2}+p_{i2}+p_{i1}-C_{i1}-p_{i1})$$
$$=-w_{i2}\cdot p_{i1}+w_{i1}\cdot p_{i2}\geqslant 0 \qquad \text{Αφού } \frac{w_{i2}}{p_{i2}}\leq\frac{w_{i1}}{p_{i1}}$$

- Μπορούμε να δούμε λοιπόν πως ο χρόνος της βέλτιστης λύσης δεν είναι απαραίτητα καλύτερος από τον χρόνο της άπληστης άρα η άπληστη μας λύση είναι βέλτιστη.

Πολυπλοκότητα. Η πολυπλοκότητα είναι όση χρειάζεται για την ταξινόμηση: $O(n \log n)$.

5.2

Θα λύσουμε το πρόβλημα με δυναμικό προγραμματισμό. Αρχικά, παρατηρούμε ότι και σε αυτή την περίπτωση οι εργασίες θα πρέπει να ανατεθούν στους υπαλλήλους με σειρά όπως στο προηγούμενο ερώτημα. Αυτό μπορεί να αποδειχθεί όπως και πριν με Επιχείρημα Ανταλλαγής στην την βέλτιστη δρομολόγηση του κάθε υπαλλήλου ξεχωριστά.
Έστω C(i,P) το βέλτιστος βεβαρυμμένος χρόνος εξυπηρέτησης με τις i πρώτες εργασίες δρομολογημένες και P το συνολικός χρόνος προετοιμασίας των πακέτων που είναι δρομολογημένα στον πρώτο υπάλληλο. Αυτός ο βέλτιστος χρόνος μπορεί να έχει προκύψει με δύο τρόπους:

- Το πακέτο i δρομολογήθηκε στον πρώτο υπάλληλο και άρα προσθέτουμε ο επιπλέον βεβαρυμμένο χρόνο εξυπηρέτησης που προκύπτει από το πακέτο αυτό στον βέλτιστο συνολικό χρόνο $C(i-1,P-p_i)$
- Το πακέτο i δρομολογήθηκε στον δεύτερο υπάλληλο και άρα πρέπει να βρούμε τον συνολικό χρόνο εξυπηρέτησης των πακέτων που βρίσκονται στον δεύτερο υπάλληλο. Επειδή έχουμε μόνο δυο υπαλλήλους αυτή είναι $\sum_{j=1}^{i} p_j - P$. Στην περίπτωση αυτή ο επιπλέον βεβαρυμμένος χρόνος εξυπηρέτησης προστίθεται στον C(i-1,P). Για να μην υπολογίζουμε τα $S_i = \sum_{j=1}^{i} p_j$ κάθε φορά από την αρχή μπορούμε μετά την ταξινόμηση των πακέτων να δημιουργήσουμε σε γραμμικό χρόνο μια λίστα S_i με το άθροισμα των i πρώτων χρόνων προετοιμασίας.

Τα παραπάνω μπορούν να περιγράφουν με την αναδρομική σχέση:

$$C(i,P) = \begin{cases} C(i-1,P-p_i) + p_{wi} \\ C(i-1,P) + w_i \cdot (\sum_{j=1}^{i} p_j - P) \qquad \alpha\nu\ i>0\ \kappa\alpha\iota\ P \geqslant 0 \end{cases}$$

Αρχικές συνθήκες:

- C(0,0)=0
- C(0,P)= ∞ , αν P διάφορο του 0
- C(i,P)=∞ , αν P<0

Οι αρχικές συνθήκες b,c μας βοηθάνε να αποκλείσουμε states που είναι αδύνατα. Επειδή αναζητούμε ελάχιστη δυνατή λύση αρχικοποιούμε τα states αυτές στο άπειρο. Η b αποκλείει τις περιπτώσεις να έχουμε μη μηδενικό χρόνο αναμονής χωρίς να έχουν δρομολογηθεί εργασίες. Η c αποκλείει τις περιπτώσεις αρνητικού συνολικού χρόνου προετοιμασίας πακέτων για τον πρώτο υπάλληλο.

Συνεπώς, ο βέλτιστος βεβαρυμμένος χρόνος εξυπηρέτησης είναι:
$C^* = \min\{ C(n,P) \}$

Μπορούμε να ανακτήσουμε την δρομολόγηση από τον πίνακα του DP σε χρόνο O(n) ξεκινώντας από το C^* και ελέγχοντας τα states από τα οποία αυτή προκύπτει.

Για παράδειγμά έστω οι εργασίες:

	w_i	p_i	S_i
1	5	1	1
2	4	2	3
3	1	2	5

Το δυναμικό πρόγραμμα θα υπολογίσει τον παρακάτω πίνακα:

	-2	-1	0	1	2	3	4	5
0	∞	∞	0	∞	∞	∞	∞	∞
1	∞	∞	5	5	∞	∞	∞	∞
2	∞	∞	17	13	13	17	∞	∞
3	∞	∞	22	17	16	16	17	22

Στον οποίο πίνακα έχουμε τις στήλες που είναι η τιμές του P και τις σειρές που είναι οι τιμές του I.

Όπως ήταν αναμενόμενο έχουμε δυο συμμετρικές βέλτιστες λύσεις.

Πολυπλοκότητα. Ο αλγόριθμος αυτός έχει O(nP) διαφορετικά states και κάθε state υπολογίζεται σε σταθερό χρόνο. Άρα η πολυπλοκότητα είναι O(nP).

Για m ≥ 3 επειδή τα αναδρομική σχέση γίνεται:

$$C(i, P_1, P_2, \ldots, P_n) = min\{C(i-1, P_1, \ldots, P_k - p_i, \ldots, P_m) + w_i \cdot P_k\} \quad \text{για } 1 \leq k \leq m$$

Με αρχικές συνθήκες:

- $C(i,0,\ldots,0)=0$
- $C(i,P_1,\ldots,P_n)=\infty$ αν υπάρχει $i|P_i<0$ ή το άθροισμα όλων των P_i για i στο [1,m] να υπερβαίνει το P.

Η βέλτιστη λύση προκύπτει από την σχέση:
$C^* = \min\{C(n,P_1,P_2 \ldots P_m)$ στα states όπου το άθροισμα όλων των P_i για i στο [1,m] να ισούται με το P.

Τώρα έχουμε $O(nP^m)$ καταστάσεις και κάθε μία χρειάζεται O(m) για να υπολογιστεί. Άρα η συνολική πολυπλοκότητα είναι $O(mnP^m)$.
Το δυναμικό πρόγραμμα αυτό λύνει και το πρόβλημα όπου m=2 και έχει ως αποτέλεσμα state space με $O(nP^2)$) καταστάσεις. Αλλά επειδή οι δύο διαστάσεις του state space πρέπει να ικανοποιούν την σχέση $P_1+P_2=P$ μπορούμε να αγνοήσουμε την μία διάσταση και με αυτό τον τρόπο προκύπτει η λύση που παρουσιάστηκε παραπάνω με πολυπλοκότητα O(nP).