Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Πρωτη Σειρα Ασκησεων
Αλγοριθμοι και Πολυπλοκοτητα

Αναστασία Χριστίνα Λίβα
03119029

Περιεχόμενα

Άσκηση 1: Πλησιέστερο Ζεύγος Σημείων **2**

Άσκηση 2: Πόρτες Ασφαλείας στο Κάστρο **4**

Άσκηση 3: Κρυμμένος Θησαυρός **4**

Άσκηση 4: Μη Επικαλυπτόμενα Διαστήματα Μέγιστου Συνολικού Μήκους **5**

Άσκηση 5: Παραλαβή Πακέτων **7**

Άσκηση 1: Πλησιέστερο Ζεύγος Σημείων

Ερώτημα Πρώτο:

Αλγόριθμος για τρεις διαστάσεις: Σε έναν πίνακα ταξινομώ τα στοιχεία κατά x. Χωρίζω τον πίνακα στα δύο ξανά και ξανά έως ότου να έχουν μείνει δύο ή τρία σημεία. Βρίσκω την ελάχιστη απόσταση των σημείων αυτών και τη θέτω ίση με δ. Επαναλαμβάνω την ίδια διαδικασία για το άλλο μισό. Υπάρχει πάντα η πιθανότητα τα σημεία με την κοντινότερη απόσταση να βρίσκονται εκατέρωθεν του επιπέδου διαχωρισμού. Παίρνω δύο λωρίδες μήκους δ εκατέρωθεν του επιπέδου διαχωρισμού, όπου δ η ελάχιστη απόσταση που έχω βρει έως τώρα, στις οποίες όλα τα σημεία που βρίσκονται μέσα θα έχουν σχεδόν όμοια x. Ταξινομώ τα σημεία αυτά κατά y, οπότε εφαρμόζω τον αλγόριθμο για δύο διαστάσεις που περιγράφεται παρακάτω μέσα στο y, z *σχεδόν* ορθογώνιο που έχει προκύψει. Συνεχίζω αναδρομικά μέχρι να βρω την ελάχιστη απόσταση.

Σημείωση: Πρόκειται για σχεδόν δισδιάστατο ορθογώνιο διότι η τρίτη διάσταση του έχει μήκος 2δ το οποίο θεωρώ *σχεδόν* αμελητέο.

Αλγόριθμος για δύο διαστάσεις: Ταξινομώ όλα τα στοιχεία κατά τη διάσταση y. Χωρίζω συνεχώς στα δύο, όπως και στον αποπάνω αλγόριθμο, έως ότου έχω δύο ή τρία σημεία, μεταξύ των οποίων βρίσκω την ελάχιστη απόσταση. Η ίδια διαδικασία επαναλαμβάνεται για το άλλο μισό. Πάλι υπάρχει πιθανότητα το κοντινότερο ζεύγος να βρίσκεται εκατέρωθεν, οπότε παίρνω πάλι μια λωρίδα μήκους δ, όπου δ η μικρότερη απόσταση που έχω βρει έως τώρα και διατάσσω κατά z. Γνωρίζω πως για δισδιάστατο σχήμα αρκεί να κοιτάξω για κάθε σημείο τα επόμενα 11 $(11 = 1 \cdot 12 - 1)$ σημεία, οπότε για τον τρισδιάστατο χώρο αρκεί να το συγκρίνω με άλλα $47(47 = 4 \cdot 12 - 1)$.

Σημείωση: Θεωρώντας σχεδόν αμελητέα τη διάσταση x, ουσιαστικά προβάλλω στο επίπεδο που ορίζεται από τα y, z όλα τα σημεία που ανήκουν στο σχεδόν δισδιάστατο σχήμα μου, οπότε χρησιμοποιώ τετράγωνα όπως και στον κανονικό αλγόριθμο δύο διαστάσεων. Υποθέτοντας στο τρισδιάστατο σχήμα κύβους $\frac{\delta}{2} \times \frac{\delta}{2} \times \frac{\delta}{2}$ και εφόσον η διάσταση x εκτείνεται σε μήκος 2δ, πίσω από κάθε τετράγωνο στο σχήμα δυο διαστάσεων βρίσκονται τέσσερις κύβοι, των οποίων τα σημεία προβάλλονται στο τετράγωνο. Συνεπώς έτσι προκύπτει ο αριθμός 47.

Πολυπλοκότητα Αλγορίθμου: Ως γνωστόν ο *αλγόριθμος δύο διαστάσεων* έχει πολυπλοκότητα τάξης $O(mlogm)$, όπου m το πλήθος των σημείων που ανήκουν

στο σχεδόν δισδιάστατο σχήμα που εξετάζω. Θεωρώντας πως έχω n σημεία συνολικά προκύπτει ότι ο *Αλγόριθμος τριών διαστάσεων* θα καλέσει τον *Αλγόριθμο δύο διαστάσεων* συνολικά $logn$ φορές. Προκύπτει συνεπώς η ακόλουθη σχέση:

$$T(n) = 2T(n/2) + O(nlogn)$$

η οποία σύμφωνα με το Master Theorem έχει λύση $O(n \log^2 n)$.

Ερώτημα Δεύτερο:

Δημιουργώ στο χώρο που μου δίνεται ένα πλέγμα του οποίου κάθε δομική μονάδα (τετράγωνα για χώρο δύο διαστάσεων, κύβοι για χώρο τριών διαστάσεων κ.ο.κ.) θα έχει διαστάσεις μήκους $\frac{l}{2}$, οπότε κάθε δομική μονάδα του πλέγματος θα περιέχει *το πολύ* ένα σημείο. Γνωρίζω επίσης πως δε χρειάζεται να συγκρίνω οποιαδήποτε δύο σημεία ανήκουν σε δομικές μονάδες οι οποίες απέχουν περισσότερο από $2c \cdot l$ δομικές μονάδες σε οποιαδήποτε διάσταση, αφού ψάχνω την ελάχιστη απόσταση και γνωρίζω εκ των προτέρων πως αυτή είναι μικρότερη του $c \cdot l$. Συγκρίνοντας κάθε σημείο με όλα τα σημεία που ανήκουν στο χώρο που ορίζεται από τις κοντινότερες $2c \cdot l$ δομικές μονάδες σε οποιαδήποτε διάσταση γύρω του και κρατώντας κάθε φορά την ελάχιστη απόσταση, βρίσκω την ελάχιστη απόσταση εκτελώντας $(4c+1)^d - 1$ πράξεις.

Για να ελέγξω μόνο τα γειτονικά σημεία και να αποφύγω την άσκοπη «περιπλάνηση» στις κενές δομικές μονάδες του πλέγματος που κοστίζει σε χρόνο θα χρησιμοποιήσω πίνακες Hash σε d επίπεδα. Στο πρώτο επίπεδο του Hash Table μου βάζω τη συντεταγμένη x κάθε σημείου. Στο επόμενο επίπεδο έχω όσους πίνακες Hash όσες και οι διακριτές τιμές της συντεταγμένης x και οι συνολικές μου θέσεις ισούνται με τον συνολικό αριθμό των σημείων. Κάθε συντεταγμένη y του δεύτερου επιπέδου αντιστοιχίζεται στη συντεταγμένη x του πρώτου σύμφωνα με τις συντεταγμένες των σημείων, οπότε πρακτικά κάθε y που αντιστοιχεί σε ίδια τιμή του x βρίσκεται στον ίδιο πίνακα Hash. Επαναλαμβάνω τη διαδικασία αυτή για όλες τις διαστάσεις δημιουργώντας για κάθε διάσταση i όσους πίνακες Hash όσες και οι διακριτές τιμές της διάστασης $i-1$.

Πολυπλοκότητα Αλγορίθμου: Η κατασκευή της δομής Hash που περιέγραψα νωρίτερα έχει γραμμική πολυπλοκότητα, ενώ η γραμμική αναζήτηση σε αυτή τη δομή έχει τάξη $O(d) = O(1)$.
Ο αλγόριθμος που περιγράφηκε εκτελεί $(4c+1)^d - 1$ πράξεις, συνεπώς η συνολική μου πολυπλοκότητα ανέρχεται σε:

$$O(n) + O(nd[(4c+1)^d - 1]) = O(n) + O(dc^d n) = O(c^d n)$$

Άσκηση 2: Πόρτες Ασφαλείας στο Κάστρο

Αλγόριθμος Ανακατασκευής: Η λογική επίλυσης του εν λόγω προβλήματος στηρίζεται στην ίδια λογική με αυτή της δυαδικής αναζήτησης. Πιο συγκεκριμένα παίρνοντας κάθε πόρτα με τη σειρά, για κάθε πόρτα i εκτελώ την ακόλουθη διαδικασία: Πατάω τους μισούς διακόπτες συγχρόνως και βλέπω εάν αλλάζει η κατάσταση της πόρτας i. Αν δεν αλλάξει πατάω τους άλλους μισούς. Πηγαίνω στο μισό που αλλάζει, το χωρίζω στα δύο και επαναλαμβάνω την ίδια διαδικασία έως ότου να καταλήξω στον διακόπτη που αλλάζει την κατάσταση της συγκεκριμένης πόρτας. Όταν βρω τον διακόπτη που αντιστοιχεί στην πόρτα i αποθηκεύω την τιμή {0 για κάτω, 1 για πάνω} που αντιστοιχεί στο άνοιγμα της πόρτας i στη θέση i πίνακα $switches$ μήκους n και φροντίζω για τις επόμενες αναζητήσεις να κρατώ κάθε πόρτα της οποίας το διακόπτη γνωρίζω ανοιχτή. Επαναλαμβάνοντας n φορές τη διαδικασία αυτή παίρνω το σωστό συνδυασμό διακοπτών.

Πολυπλοκότητα Αλγορίθμου: Εφαρμόζω ουσιαστικά n δυαδικές αναζητήσεις, μία για κάθε πόρτα. Μια δυαδική αναζήτηση έχει πολυπλοκότητα της τάξης $O(logn)$ συνεπώς η συνολική πολυπλοκότητα του αλγορίθμου μου είναι $O(nlogn)$.

Άσκηση 3: Κρυμμένος Θησαυρός

Αλγόριθμος: Θεωρώ πως η θέση του θησαυρού βρίσκεται στη θέση x. Ξεκινάω να «σαρώνω» το δρόμο σε στάδια. Συμβατικά θεωρώ πως για περιττά στάδια κινούμαι προς τα αριστερά (αρνητικοί αριθμοί) και για άρτια στάδια προς τα δεξιά (θετικοί αριθμοί). Άρα θεωρώντας τον αριθμό d περιττό αριθμό και βρισκόμενη στο σημείο 0, στο στάδιο d θα μεταβώ λ^d βήματα αριστερά και εάν δε βρω το θησαυρό θα ξαναγυρίσω στο 0. Στη συνέχεια θα μεταβώ λ^{d+1} βήματα δεξιά και εάν δε βρω το θησαυρό θα επιστρέψω στο σημείο 0. Συνεχίζω τη διαδικασία αυτή έως ότου βρω το θησαυρό.

Σημείωση: Το λ είναι *ακέραιος* αριθμός $\neq 1$. Θα αποδείξω παρακάτω πως η πιο αποδοτική τιμή για το λ είναι το 2.

Σημείωση: Ως στάδιο θεωρώ τη συνολική κίνηση που κάνω κάθε φορά από τη θέση 0 μέχρι να ξαναγυρίσω στη θέση 0 ενώ ως βήμα κάθε μετατόπιση από έναν ακέραιο στο διπλανό του (πχ $3 \rightarrow 4$).

Αποδοτικότερη Τιμή για το λ: Έστω $k \in \mathbb{N}$ τέτοιο ώστε $\lambda^k < |x| \leq \lambda^{k+1}$. Θεωρώντας πως ο θησαυρός βρίσκεται στη θέση $|x|$ και έχοντας ξεκινήσει με τη

«λάθος» φορά θα έχω διανύσει διάστημα ίσο με

$$s = \sum_{i=1}^{k+1} 2 \cdot \lambda^i + |x| = 2 \cdot \frac{\lambda(\lambda^{k+1} - 1)}{\lambda - 1} + |x| < 2 \cdot \frac{\lambda(\lambda \cdot |x| - 1)}{\lambda - 1} + |x| = \frac{\lambda^2 \cdot |x| - \lambda}{\lambda - 1} + |x|$$

Θέλω να ελαχιστοποιήσω την ποσότητα αυτή οπότε πρέπει να πάρω τη μικρότερη δυνατή τιμή για το λ, καθώς το λ^2 πολλαπλασιάζει το $|x|$ το οποίο θα είναι στη βέλτιστη περίπτωση το μεγαλύτερο από τα βήματα που θα κάνουμε. Άρα η κατάλληλη τιμή για το λ είναι το 2.

Αριθμός Σταδίων/Πολυπλοκότητα Αλγορίθμου: Έστω $k \in \mathbb{N}$ τέτοιο ώστε $2^k < |x| \leq 2^{k+1}$. Τότε ο αλγόριθμός μου χρειάζεται τουλάχιστον $\sum_{i=1}^{k} 2 \cdot 2^i$ βήματα καθώς ο θησαυρός θα βρεθεί είτε στο στάδιο $k+2$ (χείριστη περίπτωση όπου κάνω 2^{k+1} βήματα προς τα αριστερά ενώ ο θησαυρός βρίσκεται στη θέση x με $2^k < x \leq 2^{k+1}$ προς τα δεξιά) είτε στο στάδιο $k+1$ (βέλτιστη περίπτωση που λόγω τύχης κινούμαι στη σωστή κατεύθυνση κατευθείαν). Θεωρώντας τη χειριστή περίπτωση, στο στάδιο $k+1$ περπατώ προς τη λάθος κατεύθυνση οπότε έχω κάνει μια έξτρα απόσταση $2 \cdot 2^{k+1}$ βήματα. Επιστρέφοντας στο σημείο 0, περπατώ άλλα $|x|$ βήματα προς τη σωστή κατεύθυνση. Συνεπώς:

$$\textbf{Συνολικό Διάστημα} = \sum_{i=1}^{k+1} 2 \cdot 2^i + |x| = 2(2^{k+2} - 1) + |x| \leq 8|x| - 2 + |x| \leq 9|x|$$

Άσκηση 4: Μη Επικαλυπτόμενα Διαστήματα Μέγιστου Συνολικού Μήκους:

Ερώτημα Πρώτο:

Μέγιστο Διάστημα: Έστω άπληστος αλγόριθμος με άπληστο κριτήριο την επιλογή του διαστήματος $[s_i, f_i)$ που έχει τη μεγαλύτερη τιμή $f_i - s_i$ σε κάθε επανάληψη. Ο αλγόριθμος αυτός δε θα δώσει το επιθυμητό αποτέλεσμα καθώς μπορώ να βρω αντιπαράδειγμα.

Αντιπαράδειγμα: Έστω 4 διαστήματα:

$$[0, 3),\ [3, 5),\ [6, 8),\ [2, 7)$$

Σε αυτή την περίπτωση ο αλγόριθμος θα επιλέξει το διάστημα $[2, 7)$ μήκους 5 ενώ η σωστή λύση είναι τα τρία πρώτα διαστήματα με συνολικό μήκος 8.

Ελάχιστο Διάστημα: Όμοια θεωρώ τώρα άπληστο αλγόριθμος με άπληστο κριτήριο την επιλογή του διαστήματος $[s_i, f_i)$ που έχει τη μικρότερη τιμή $f_i - s_i$ σε κάθε επανάληψη.

Αντιπαράδειγμα: Έστω 4 διαστήματα:

$$[0, 20), [0, 1), [5, 7), [7, 15)$$

Ο αλγόριθμος θα επιλέξει τα τρία τελευταία διαστήματα με συνολικό μήκος 11 ενώ το πρώτο διάστημα είχε συνολικό μήκος 20.

Ερώτημα Δεύτερο: Αποθηκεύω σε πίνακα n θέσεων $Space[i] = f_i - s_i,\ i \in [0, n-1]$ και ταξινομώ τα διαστήματα ως προς f_i με κάποιον αλγόριθμο ταξινόμησης (mergesort ή quicksort). Αρχικοποιώ έναν μηδενικό πίνακα OPT n στοιχείων. Στο $i-1$-οστό στοιχείο του πίνακα αποθηκεύω την τιμή για το μέγιστο συνολικό μήκος για το ίδιο πρόβλημα θεωρώντας ως input μόνο τα τα i πρώτα ταξινομημένα διαστήματα, δηλαδή τη λύση του μικρότερου προβλήματος για τα διαστήματα $(s_1, f_1), ..., (s_i, f_i)$.

- **Για i = 0:** Έχω ως δεδομένο εισόδου μόνο το πρώτο διάστημα (s_1, f_1), οπότε η λύση του υποπροβλήματος είναι $f_1 - s_1$ και συνεπώς $Space[0] = OPT[0] = f_1 - s_1$
- **Για i = k:** Βρίσκομαι σε ενδιάμεση κατάσταση όπου λόγω επαγωγής έχω βρει τη βέλτιστη λύση για τα $k-1$ προηγούμενα διαστήματα, οπότε προσθέτοντας στο μέτωπο αναζήτησης το $k-$οστό διάστημα έχω τα δύο ακόλουθα ενδεχόμενα:
  Συνάρτηση compute_opt():
  - Το $[s_k, f_k)$ να μην ανήκει στα διαστήματα που συνιστούν τη βέλτιστη λύση, οπότε η βέλτιστη λύση του k-οστού στιγμιοτύπου για το πρόβλημά μου είναι ίδια με αυτή για το στιγμιότυπο $k-1$.
  - Το $[s_k, f_k)$ να ανήκει στα διαστήματα που συνιστούν τη βέλτιστη λύση, οπότε και η βέλτιστη λύση περιλαμβάνει το $Space[k] = f_k - s_k$ *συν* την πιο *πρόσφατη* λύση που βρήκα νωρίτερα και *δεν επικαλύπτεται* με το διάστημα $Space[k]$ που προσθέτω τώρα (έστω πως η λύση αυτή είναι η λύση του στιγμιοτύπου j), οπότε έχω

$$OPT[k] = Space[k] + OPT[j]$$

Τώρα για κάθε στοιχείο $i \in [1, n-1]$ βρίσκω εάν υπάρχει με δυαδική αναζήτηση στα ταξινομημένα διαστήματα το index του διαστήματος που τελειώνει αμέσως πριν το $i-$οστό χωρίς να το επικαλύπτει ($f_{index} \leq s_i$)

- **Εάν υπάρχει:**

$$OPT[i] = max\{OPT[i-1],\ OPT[index] + Space[i]\}$$

- **Εάν δεν υπάρχει:**

$$OPT[i] = max\{OPT[i-1],\ 0\ +\ Space[i]\}$$

κι έπειτα επιστρέφω το $OPT[n-1]$ ως συνολική απόσταση.

Για να βρω τα επιλεγμένα διαστήματα θεωρώντας πως βρίσκομαι σε ενδιάμεσο στιγμιότυπο j:

- Αν $OPT[j] = OPT[j-1]$ τότε το διάστημα (s_j, f_j) δεν αποτελεί συνιστώσα της βέλτιστης λύσης, οπότε ελαττώνω το j κατά μια μονάδα και συνεχίζω την ίδια διαδικασία για τα υπόλοιπα στιγμιότυπα.
- σε αντίθετη περίπτωση βάζω το διάστημα (s_j, f_j) στη λύση και υπολογίζω το $w = OPT[j] - Space[j]$. Στη συνέχεια με δυαδική αναζήτηση στον πίνακα OPT για το w βρίσκω το index k για το οποίο $OPT[k] = w$, θέτω $j = k$ και συνεχίζω τη διαδικασία αυτή έως ότου βρω όλα τα διαστήματα.

Ορθότητα Αλγορίθμου: Εξ ορισμού έχω πως για μηδενικά διαστήματα έχω $OPT(0) = 0$. Για τυχαίο $j > 0$ θεωρώ πως υπολογίζω σωστά το $OPT[i]$ για κάθε $i < j$. Από την υπόθεση της επαγωγής γνωρίζω ότι η $compute\_opt(p(j)) = OPT(p(j))$ και $compute\_opt(j-1) = OPT[j-1]$, άρα προκύπτει:

$$OPT[j] = max\{Space[j] + compute\_opt(p(j)), compute\_opt(j-1)\} = compute\_opt(j)$$

Σημείωση: Για διάστημα j το $p(j)$ το ορίζω ως τον μεγαλύτερο δείκτη $i < j$ τέτοιο ώστε τα διαστήματα i και j να είναι *ξένα* μεταξύ τους.

Πολυπλοκότητα Αλγορίθμου: Η ταξινόμηση των n διαστημάτων έχει πολυπλοκότητα $O(nlogn)$ και για κάθε διάστημα κάνω μία δυαδική αναζήτηση και κάποιες πράξεις οπότε έχω $O(n(logn + c)) = O(nlogn)$, οπότε η συνολική πολυπλοκότητα του αλγορίθμου προκύπτει $O(nlogn)$.

Άσκηση 5: Παραλαβή Πακέτων

Ερώτημα Πρώτο: Για να δρομολογήσω τα πακέτα έχοντας στη διάθεσή μου έναν υπάλληλο υποθέτοντας ότι έχω n πελάτες, δημιουργώ έναν πίνακα μεγέθους

n. Για κάθε πελάτη i υπολογίζω το $\frac{w_i}{p_i}$, το οποίο και αποθηκεύω στην αντίστοιχη θέση του πίνακα. Στη συνέχεια ταξινομώ τον πίνακα χρησιμοποιώντας mergesort. Διασχίζω τον πίνακα από το τέλος προς την αρχή εξυπηρετώντας λοιπόν πρώτα τους πελάτες με μεγαλύτερα $\frac{w_i}{p_i}$.

Ο άπληστος αλγόριθμός μου περιγράφεται από την ακόλουθη αναδρομική σχέση:

$$G(A_i) = \begin{cases} G(A_i - w_i, p_i) + (\sum_{j=1}^{j=i} p_j)w_i \\ 0, \quad \text{εάν } A_i = \emptyset \end{cases}$$

Άπληστο Κριτήριο: Είναι απόλυτα λογική η χρήση της μεγιστοποίησης του κλάσματος $\frac{w_i}{p_i}$ ως άπληστο κριτήριο της δρομολόγησης αυτής καθώς θέλω να δώσω προτεραιότητα στους πελάτες με μεγάλο βάρος εφόσον έχουν μεγάλη σημασία για την επιχείρησή μου, οπότε τοποθετώ το βάρος στον αριθμητή του κλάσματος αφού μεγάλος αριθμητής συνεπάγεται μεγάλο κλάσμα. Θέλω επίσης να ελαχιστοποιήσω το συνολικό χρόνο εξυπηρέτησης, οπότε βάζω τον χρόνο εξυπηρέτησης p_i στον παρονομαστή του κλάσματος, καθώς μικρός παρονομαστής συνεπάγεται μεγάλο κλάσμα.

Ορθότητα Αλγορίθμου: Ο άπληστος αλγόριθμος έχει κόστος

$$G(A_n) = w_1p_1 + w_2(p_1 + p_2) + ... + w_n(p_1 + p_2 + ... + p_n) =$$
$$p_1 \sum_{1}^{n} w_k + ... + p_i \sum_{k=i}^{n} w_k + ... + p_j \sum_{k=j}^{n} + .. + w_np_n$$

Θεωρώ βέλτιστο αλγόριθμο που μου δίνει ως λύση ακολουθία $C^*(A_i)$, όπου δύο πακέτα i, j είναι $swapped$, άρα κατά συνέπεια ισχύει $\frac{w_i}{p_i} \geq \frac{w_j}{p_j}$. Η δρομολόγηση αυτή έχει κόστος

$$p_1 \sum_{\kappa=1}^{N} w_\kappa + ... + p_j \sum_{k=i}^{N} w_k + p_{i+1}(\sum i + 1^N w_k - w_j + w_i)$$
$$+ ... + p_\lambda(\sum_{\lambda}^{n} w_k - w_j + w_i) + .. + p_i(w_i + \sum_{k=j+1}^{n} w_k) + ... + p_nw_n$$

Τότε

$$C^*(A_n) - G(A_n) = p_j \sum_{i}^{j-1} w_k - p_i \sum_{i+1}^{j} w_k + \sum_{i+1}^{j-1} p_k(w_i - w_j)$$

Για κάθε $\lambda,\ i < \lambda < j$:

$$\frac{w_i}{p_i} > \frac{w_\lambda}{p_\Lambda} > \frac{w_j}{p_j} \Rightarrow w_ip_\lambda > w_\lambda p_i \text{ και } w_\lambda p_j > p_\lambda w_j$$

Ομαδοποιώντας σε δυάδες τους όρους της αφαίρεσης $C^*(A_n) - G(A_n)$ έχω

$$p_j w_i - p_i w_j + p_j w_{i+1} - p_{i+1} w_j + ... + p_{i+1} w_i - p_i w_{i+1} + ... + p_{j-1} w_i - p_i w_{i-1}$$

οπότε συμπεραίνω ότι

$$C^*(A_n) > G(A_n)$$

Συνεπώς προκύπτει πως η βέλτιστη λύση είναι η άπληστη.

Πολυπλοκότητα Αλγορίθμου: Χρησιμοποιώ mergesort για την ταξινόμηση για την οποία έχω και έπειτα διασχίζω πίνακα n στοιχείων, οπότε:

$$T(n) = O(n) + O(nlogn)$$

και συνεπώς έχω συνολική πολυπλοκότητα $O(nlogn)$.

Ερώτημα Δεύτερο: Για τη δρομολόγηση με δύο εργαζομένους αρχικά ταξινομώ τα $\frac{w_i}{p_i}$. Υποθέτοντας ότι ξεκινώ με $C(n_i \sum_{j=0}^{n} p_j)$ ο αλγόριθμός μου περιγράφεται από την εξής αναδρομική σχέση:

$$C(i, A) = min \left\{ \begin{array}{l} C(i-1, A - p_i) + Aw_i \\ C(i-1, A) + w_0(\sum_{j=0}^{i} p_j - A) \end{array} \right\}$$

$$C(i, 0) = \text{Λύση των } i \text{ με έναν υπάλληλο}$$

$$C(0, A) = 0$$

Α είναι ο συνολικός χρόνος που κάνει ο πρώτος υπάλληλος . Ουσιαστικά για κάθε πελάτη επιλέγω αν θα τον αναθέσω στον πρώτο ή το δεύτερο εργαζόμενο και επειδή ξεκινώ από το τέλος αφαιρώ από το υποπρόβλημα. Για το δεύτερο εργαζόμενο, ο χρόνος ισούται $\sum$ Όλων των χρόνων $- \sum$ Χρόνων του πρώτου

Πολυπλοκότητα Αλγορίθμου: Η πολυπλοκότητα του αλγορίθμου είναι $\mathrm{O}(n^2)$ καθώς ο πίνακας δυναμικού προγραμματισμού που γεμίζεται από την αναδρομική σχέση «φτιάχνεται» από την παράμευρο i που λαμβάνει τιμές από 1 έως n και την παράμετρο που με βήμα p_i φτάνει έως την τιμή $\sum_{i=0}^{n} p_i$, οπότε παίρνει n τιμές.

Σκιαγράφηση Προβλήματος για Περισσότερους Εργαζομένους: Εργάζομαι με παρόμοιο τρόπο γνωρίζοντας ότι εάν διαθέτω m εργαζομένους κάθε πακέτο μπορεί να ανατεθεί σε οποιονδήποτε από αυτούς, συνεπώς κάθε πακέτο θα έχει m επιλογές.

$$C(i, A_1, A_2, ..., A_{d-1}) = min \begin{cases} C(i-1, A_1 - p_i, A_2, ..., A_{d-1}) + \mathrm{A}_1 w_i \\ C(i-1, A_1, A_2 - p_i, ..., A_{d-1}) + \mathrm{A}_2 w_i \\ ... \\ C(i-1, A_1, A_2 - p_i, ..., A_{d-1} - p_i) + \mathrm{A}_{d-1} w_i \\ C(i-1, A_1, A_2 - p_i, ..., A_{d-1}) + w_i(\sum_{j=0}^{i} p_i - \sum_{j=1}^{d-1} A_j \end{cases}$$

όπου τα Α αναπαριστούν το χρόνο που θέλει κάθε υπάλληλος αι ο τελεταίος όρος υπολογίζει το χρόνο του υπαλλήλου d.

Πολυπλοκότητα Αλγορίθμου: $O(n^d \cdot d)$, αφού ο χώρος του προβλήματος έχει μέγεθος

$$size(i) \cdot size(A_1)....size(\mathrm{A}_{d-1}) = n \cdot \underbrace{n \cdot n \cdot n... \cdot n}_{d-1}$$

και για κάθε state στην αναδρομή έχω d επιλογές άρα έτσι προκύπτει το $O(n^d \cdot d)$