Αλγόριθμοι και Πολυπλοκότητα

1η Σειρά Γραπτών Ασκήσεων

Ιωάννης Τσαντήλας, 03120883

Άσκηση 1

- *Ερώτημα 1*

Αρχικά, ταξινομούμε τα σημεία με βάση την συντεταγμένη x. Αναδρομικά, χωρίζουμε την ταξινόμηση στην μέση και βρίσκουμε την ελάχιστη απόσταση μεταξύ του κάθε μισού (έστω δ_δ και δ_α). Θέτουμε δ = min(δ_δ, δ_α). Για την ενδιάμεση περιοχή, παίρνουμε μία «φέτα» (εν αντιστοιχία με την «λωρίδα» στο δυσδιάστατο πρόβλημα) που απέχει από το σύνορο (έστω x=L) απόσταση δ = min(δ_δ, δ_α).

Ψάχνουμε δηλαδή στο διάστημα x=[L-δ, L+δ] εάν υπάρχει μία απόσταση μικρότερη από τη δ. Χωρίζουμε την εν λόγω «φέτα» σε «grid» από κυβάκια μήκους δ/2 (εν αντιστοιχία με τα τετράγωνα στο δυσδιάστατο πρόβλημα). Για ένα σημείο-κυβάκι μέσα στο grid, χρειάζεται να εξετάσουμε τα αμέσως γύρω του – που θα απέχουν το πολύ δ - δηλαδή 4*4*3-1 = 47 κουτάκια στην χειρότερη περίπτωση.

Όσον αφορά την πολυπλοκότητα:

- Η ταξινόμηση ως προς x (π.χ. με quicksort) χρειάζεται O(n*logn),
- Η εύρεση της ελάχιστης απόστασης δ στα δύο μισά ξεχωριστά χρειάζεται 2*T(n/2),
- Η διαγραφή όλων των σημείων που απέχουν απόσταση παραπάνω από δ από το επίπεδο x=L χρειάζεται O(n),
- Η εξέταση των υπολοίπων 47 σημείων στην «φέτα» χρειάζεται O(n).

Συνολικά:

$$T(n) \leq 2 * T\left(\frac{n}{2}\right) + O(n * logn) + 2 * O(n) = 2 * T\left(\frac{n}{2}\right) + O(n * logn) \rightarrow$$

$$\rightarrow T(n) = O(n * \log^2 n)$$

- *Ερώτημα 2*

Χωρίζουμε τον δυσδιάστατο χώρο σε «grid» μήκους l και τοποθετούμε τα σημεία στο αντίστοιχο κελί τους. Για κάθε σημείο θα πρέπει να ψάξουμε την γύρω περιοχή του, η οποία είναι ένα τετράγωνο μήκους «**2cl**» (το σημείο βρίσκεται στο κέντρο, επομένως ψάχνουμε δεξιά-αριστερά κατά «**cl**» και πάνω-κάτω κατά «**cl**»), συνολικά **(2cl)²**. Για n σημεία, θα κάνουμε **n*(2cl)²** επαναλήψεις.

Αντίστοιχα, για 3 διαστάσεις, για ένα σημείο θα πρέπει να ελέγξουμε την γύρω περιοχή του, η οποία είναι ένας κύβος μήκους «**2cl**» (το σημείο βρίσκεται στο κέντρο, επομένως ψάχνουμε δεξιά-αριστερά κατά «**cl**», πάνω-κάτω κατά «**cl**» και μέσα-έξω κατά «**cl**»), συνολικά **(2cl)³**. Για n σημεία, θα κάνουμε **n*(2cl)³**.

Επομένως, για d διαστάσεις, εύλογα, θα χρειαστούμε $\boldsymbol{O(n * (2cl)^d)}$.

Άσκηση 2

Αρχικά, έχουμε στην κάτω θέση όλους τους διακόπτες και εξετάζουμε την κατάσταση της πρώτης πόρτας. Κλείνοντας τους πρώτους μισούς, εξετάζουμε εάν η κατάσταση της πρώτης πόρτας άλλαξε. Εάν άλλαξε, αυτό σημαίνει πως ο διακόπτης της είναι στο πρώτο μισό, διαφορετικά είναι στο δεύτερο μισό. Στο μισό που προσδιορίσαμε που είναι, αλλάζουμε τους πρώτους μισούς του (δηλαδή το ένα τέταρτο του συνόλου) και πάλι εξετάζουμε εάν η κατάσταση της πόρτας άλλαξε. Συνεχίζουμε να κόβουμε σε μισά μέχρι να βρούμε τον πρώτο διακόπτη.

Αφού έχουμε n διακόπτες και σε αυτή τη διαδικασία χωρίζουμε το σύνολο κάθε φορά στη μέση, θα χρειαστούμε logn βήματα. Επαναλαμβάνουμε την διαδικασία αυτή για τους υπόλοιπους διακόπτες (εξαίροντας βέβαια από την εξέταση αυτούς που έχουμε προσδιορίσει). Η 2η επανάληψη θα χρειαστεί log(n-1) βήματα, η 3η log(n-2) κ.ο.κ.. Συνολικά:

$$\sum_{i=0}^{n-1} \log(n-i) =$$

$$= logn + log(n-1) + log(n-2) + \cdots + log1 =$$
$$= \log[n * (n-1) * (n-2) * \ldots * 1] =$$
$$= \log(n!) = \Theta(nlogn)$$

Άσκηση 3

Ξεκινώντας από το 0 πάμε στο 1 (διανύοντας απόσταση 1). Από ‘δω και στο εξής θα πολλαπλασιάζουμε την συντεταγμένη στην οποία βρισκόμαστε με -2. Το αποτέλεσμα αυτής της πράξης είναι η επόμενη συντεταγμένη στην οποία θα κατευθυνθούμε (δηλαδή η αναζήτηση μας θα ακολουθεί τις επόμενες συντεταγμένες: 0, 1, -2, 4, -8, 16, -32, 64, …, δηλαδή: 0, 2^0, -2^1, 2^2, -2^3, 2^4, -2^5, 2^6, …).

Στην χειρότερη περίπτωση το |x| ισούται με μία δύναμη του 2 (διαφορετικά, καθώς θα πηγαίναμε στην αμέσως επόμενη μεγαλύτερη δύναμη θα «γλυτώναμε» βήματα). Έστω, χωρίς βλάβη της γενικότητας, υποθέτουμε πως x = 2^k, με k>0. Τα συνολικά βήματα που κάνουμε για να φτάσουμε σε αυτό είναι:

$$|2^0 - 0| + |-2^1 - 2^0| + |2^2 - (-2^3)| + |-2^3 - 2^4| + \cdots + |2^k - (-2^{k-1})| =$$

$$= 2^0 + |2^1 + 2^0| + |2^2 + 2^1| + |2^3 + 2^4| + \cdots + |2^k + 2^{k-1}| =$$

$$= 1 + \sum_{i=1}^{k} \left(2^i + 2^{i-1}\right) = 1 + 3 * (2^k - 1) = 3 * 2^k - 2 = 3 * |x| - 2 = O(3|x|)$$

Γενικεύοντας το για οποιαδήποτε βάση *a > 1*:

$$1 + \sum_{i=1}^{k} \left(a^i + a^{i-1}\right) =$$

$$= 1 + (a^k - 1) * \frac{a+1}{a-1} =$$

$$= 1 + (|x| - 1) * \frac{a+1}{a-1}$$

Η συνάρτηση $\frac{a+1}{a-1}$ με α θετικό ακέραιο αριθμό, μεγαλύτερο του 1, είναι γνησίως φθίνουσα. Όσο μεγαλύτερο το α, τόσο αυτή πλησιάζει την τιμή 1. Η μεγαλύτερη τιμής της δίνεται για α = 2, η οποία είναι 3. Επομένως, στην χειρότερη περίπτωση, θα κάνουμε το πολύ $O(3|x|)$βήματα, δηλαδή c = 3.

Άσκηση 4

- *Ερώτημα 1*

Θα αποδείξουμε με αντιπαράδειγμα πως ο άπληστος αλγόριθμος που επιλέγει πάντα τα μεγαλύτερο διάστημα είναι λάθος. Εάν έχουμε τα διαστήματα: [0,42), [0,17) και [17,43), ο αλγόριθμος θα διάλεγε το [0,42), ενώ ο μεγαλύτερος δυνατός συνδυασμός είναι ο [0,17) και [17,43).

Επίσης, ο άπληστος αλγόριθμος που επιλέγει πάντα το διάστημα με μικρότερο f_i είναι λάθος. Εάν έχουμε τα διαστήματα: [0,1), [1,17) και [16,42), ο αλγόριθμος θα διάλεγε τα [0,1) και [1,17), ενώ ο μεγαλύτερος δυνατός συνδυασμός είναι ο [0,1) και [16,42).

- *Ερώτημα 2*

Ταξινομούμε τα διαστήματα σε αύξουσα σειρά του f_i. Χωρίς βλάβη της γενικότητας, θεωρούμε πως $f_1 < f_2 < \cdots < f_n$ και δημιουργούμε δύο πίνακες μεγέθους f_n, τους A και B. Στο A[i] θα αποθηκεύουμε το μέγιστο συνολικό μήκος των διαστημάτων που τελειώνει το πολύ στην χρονική στιγμή i, ενώ στο B[f_i] θα αποθηκεύουμε το s_i. Ξεκινούμε με το (s_1,f_1). Οι θέσεις A[1] έως και A[f_n-1] γεμίζουν με 0, ενώ A[f_1] = f_1-s_1. Επιπλέον, B[f_1] = s_1.

Συνεχίζουμε με τα υπόλοιπα διαστήματα και έστω πως είμαστε στο (s_i,f_i). Γεμίζουμε τις θέσεις του A[f_{i-1}+1] έως και A[f_i-1] με την ίδια τιμή της θέσης A[f_{i-1}], αφού δεν υπάρχει κάποιο άλλο διάστημα που μπορεί να προστεθεί και να οδηγήσει σε μεγαλύτερο μήκος. Στην θέση A[f_i] υπάρχουν δύο ενδεχόμενα, το διάστημα (s_i,f_i) να μπει ή να μην μπει στην ακολουθία. Αρκεί λοιπόν να συγκρίνουμε την τιμή A[f_i-1] – η οποία είναι το μέγιστο μήκος χωρίς το (s_i,f_i) – με το μήκος f_i-s_i+A[s_i], το οποίο είναι το άθροισμα του μήκους του (s_i,f_i) και του μέγιστου δυνατού μήκους των διαστημάτων που τελειώνουν έως στο πολύ s_i.

Θέτουμε $A[f_i] = max\{A[f_i - 1], f_i - s_i + A[s_i]\}$, και εάν είναι το f_i-s_i+A[s_i] – δηλαδή επιλέξαμε το νέο διάστημα, θέτουμε B[f_i] = s_i. Εάν περισσότερα διαστήματα τελειώνουν στην ίδια θέση, τα ελέγχουμε με τη σειρά και κρατάμε την βέλτιστη επιλογή. Τελικά, το A[f_n] θα έχει το μέγιστο δυνατό μήκος.

Για να βρούμε ποια διαστήματα αποτελούν αυτό το μήκος, ξεκινούμε από την A[f_n] και προχωράμε προς την αρχή του πίνακα, ελέγχοντας πότε αλλάζει η τιμή του. Στην τελευταία θέση προτού αλλάξει η τιμή, βρίσκεται το τέλος του προηγούμενου διαστήματος, η αρχή του οποίου μπορεί να βρεθεί από τον πίνακα B. Συνεχίζουμε ομοίως.

Για να ταξινομήσουμε τα διαστήματα (π.χ. με quicksort) χρειαζόμαστε n*logn επαναλήψεις. Για να βρούμε το συνολικό μήκος και τα σωστά διαστήματα χρειαζόμαστε f_n επαναλήψεις αντίστοιχα, ενώ χρειαζόμαστε επιπλέον n επαναλήψεις για να ελέγξουμε κάθε διάνυσμα πριν βάλουμε τις τιμές στον πίνακα A. Συνολικά:

$$O(n * logn + 2 * f_n + n)$$

Άσκηση 5

- *Ερώτημα 1*

Θα ταξινομήσουμε σε φθίνουσα σειρά τους πελάτες με βάση την **αποτελεσματικότητα** τους, η οποία δίνεται από τον λόγο w/p (όπου όσο πιο ψηλό w και χαμηλό p έχουν, τόσο πιο σημαντικοί είναι), και με αυτή τη σειρά θα τους εξυπηρετήσουμε. Για τον υπολογισμό των λόγων w/p χρειαζόμαστε n επαναλήψεις, για την ταξινόμηση τους θέλουμε n*logn επαναλήψεις (π.χ. με την quicksort) και τέλος η εξυπηρέτηση τους χρειάζεται n επαναλήψεις. Συνολικά, ο αλγόριθμος αυτός έχει πολυπλοκότητα O(2n + nlogn) = O(nlogn).

Η ιδέα είναι σωστή, αφού αν υπάρχουν δύο διαδοχικοί πελάτες i και i+1 για τους οποίους δεν τηρείται ότι $w_i/p_i > w_{i+1}/p_{i+1}$, και ισχύει το ανάποδο, τότε θα πρέπει να τους αλλάξουμε σειρά. Ο συνολικός **βεβαρυμμένος** χρόνος των υπόλοιπων πελατών δεν αλλάζει, επομένως ο χρόνος στις δύο περιπτώσεις είναι:

$$T_{o\lambda} = T_{\upsilon\pi o\lambda} + w_i * (t_{\pi\rho\iota\nu} + p_i) + w_{i+1} * (t_{\pi\rho\iota\nu} + p_i + p_{i+1})$$

$$T'_{o\lambda} = T_{\upsilon\pi o\lambda} + w_{i+1} * (t_{\pi\rho\iota\nu} + p_{i+1}) + w_i * (t_{\pi\rho\iota\nu} + p_i + p_{i+1})$$

Η διαφορά αυτών των ποσοτήτων είναι:

$$T'_{o\lambda} - T_{o\lambda} = w_i * p_{i+1} - w_{i+1} * p_i = p_i * p_{i+1}(\frac{w_i}{p_i} - \frac{w_{i+1}}{p_{i+1}})$$

Και αφού $w_i/p_i < w_{i+1}/p_{i+1}$:

$$T'_{o\lambda} - T_{o\lambda} < 0 \rightarrow T'_{o\lambda} < T_{o\lambda}$$

Άτοπο, αφού υποθέσαμε πως αν αλλάζαμε την σειρά τους, θα είχαμε καλύτερο **βεβαρυμμένο** χρόνο.

- *Ερώτημα 2*

Ακολουθούμε την ίδια τακτική υπολογισμού και ταξινόμησης, αλλά τώρα έχουμε δύο ουρές όπου μπορούμε να τοποθετήσουμε τους πελάτες. Επομένως, η τακτική μας αλλάζει: ο επόμενος πελάτης θα πάει στην ουρά με το μικρότερο συνολικό **βεβαρυμμένο** χρόνο. Χωρίς βλάβη της γενικότητας θεωρούμε πως:

$$\frac{w_1}{p_1} > \frac{w_2}{p_2} > \cdots > \frac{w_n}{p_n}$$

Έστω πως έχουμε δύο ουρές, με συνολικό **απλό** χρόνο T_1 και T_2, δηλαδή τα T ισούται με το άθροισμα των p των πελατών που ανήκουν στην εν λόγω ουρά. Εάν ο πελάτης n πάει στην ουρά 1, τότε στον **βεβαρυμμένο** χρόνο θα προστεθεί w_n*T_1, ενώ το T_1 θα πρέπει να μειωθεί κατά p_n, αφού ο πελάτης n θεωρείται μέρος της ουράς 1. Εάν ο πελάτης n πάει στην ουρά 2, τότε στον **βεβαρυμμένο** χρόνο θα προστεθεί $w_n * \sum_{i=1}^{n}(p_i - T_1)$, ενώ το T_1 θα παραμείνει ίδιο. Σε κάθε περίπτωση, μειώνουμε το n κατά 1 για να πάμε στον προηγούμενο πελάτη. Συνολικά, ο αλγόριθμος δίνεται από την εξίσωση:

$$C(n, T_1) = \min \begin{cases} C(n-1, T_1 - p_n) + w_n * T_1 \\ C(n-1, T_1) + w_n * \sum_{i=1}^{n}(p_i - T_1) \\ C(0, t) = 0 \end{cases}$$

Για να αποφύγουμε άσκοπες επαναλήψεις, αποθηκεύουμε όλα τα C(n,T_1) σε έναν μονοδιάστατο πίνακα μεγέθους $\sum_{i=1}^{n} p_i = \max\{T_1\}$. Χρειάζεται να γεμίσουμε όλες τις θέσεις του, επομένως ο αλγόριθμος μας έχει πολυπλοκότητα $O(n * \sum_{i=1}^{n} p_i)$.

Εάν έχουμε m υπαλλήλους, τότε ο αλγόριθμος θα είναι της μορφής $C(n, T_1, T_2, \dots, T_{m-1})$, όπου - όπως και πριν – δεν κρατάμε τον τελευταίο αφού σε αυτόν πάνε όσοι πελάτες δεν εξυπηρετούνται από τους προηγούμενους. Ομοίως, θα έχουμε έναν πίνακα, μόνο που τώρα θα είναι διαστάσεων $(m-1) \times \sum_{i=1}^{n} p_i$, μία γραμμή για κάθε υπάλληλο. Επομένως, η συνολική πολυπλοκότητα για να γεμίσουμε αυτόν τον πίνακα θα είναι $O(n * (\sum_{i=1}^{n} p_i)^{m-1})$.