ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

1η ΣΕΙΡΑ ΓΡΑΠΤΩΝ ΑΣΚΗΣΕΩΝ

Διδάσκοντες:

Άρ. Παγουρτζής
Δ. Φωτάκης
Δ. Σούλιου
Π. Γροντάς

Ειρήνη Δόντη
ΑΜ 03119839

7ο εξάμηνο

Αθήνα 2022

Άσκηση 1: Πλησιέστερο Ζεύγος Σημείων

(a) Ταξινομούμε τα σημεία x_i, y_i και z_i με i = 1, 2,….,n σε πίνακες X, Y και Z αντίστοιχα. Διαχωρίζουμε τα σημεία σε δύο υποπίνακες Xleft και Xright, βάσει των ταξινομημένων τιμών των x σημείων. Με αναδρομή, βρίσκουμε το πλησιέστερο ζεύγος σημείων για τους υποπίνακες Xleft και Xright αντίστοιχα (οπότε το χρονικό κόστος διαμορφώνεται ως 2T(n/2) μέχρι στιγμής). Θεωρούμε δ το πλησιέστερο διάστημα του ζεύγους σημείων είτε εξ'ολοκλήρου στον υποπίνακα Xleft είτε στον Xright. Στην περίπτωση που τα πλησιέστερα σημεία αντιστοιχούν σε σημεία των Xleft και Xright, χρειαζόμαστε μία συνθήκη, αντίστοιχα με το πρόβλημα σε δύο διαστάσεις. Όμως, στην περίπτωση των τριών διαστάσεων, δεν έχουμε μία μεσαία γραμμή με μήκος 2δ, αλλά έχουμε μία πλάκα πλάτους 2δ που εκτείνεται στις y και z διαστάσεις. Για κάθε σημείο της πλάκας, υπάρχει περιορισμένος αριθμός σημείων στη γειτονική περιοχή, ώστε τα σημεία στην πλάκα που είναι στο Xleft (ή αντίστοιχα στο Xright) να βρίσκονται το λιγότερο σε διάστημα δ. Θεωρούμε ότι το όριο είναι Π σημεία. Η εύρεση των πλησιέστερων σημείων μπορεί να εκτελεστεί με χρόνο τάξης O(n), έχοντας ένα σημείο στον Xleft υποπίνακα και ένα στον Xright υποπίνακα. Χρειάζεται πρόσβαση στους ταξινομημένους Y και Z πίνακες, ώστε να βρεθούν τα Π σημεία που βρίσκονται πλησιέστερα στα Y και Z σημεία σε χρόνο O(n). Οπότε, ο παραπάνω αλγόριθμος για τις 3 διαστάσεις, ικανοποιεί τη σχέση T(n) = 2T(n/2) + O(n) + O(n), με το οποίο αποδεικνύεται, μέσω Master Theorem, ότι ο παραπάνω αλγόριθμος μπορεί να εκτελεστεί σε χρόνο $\Theta(n^{\log_2 2}\log n) = O(n\log n)$.

(b) Εφόσον το δ^* είναι περιορισμένο στο διάστημα $[\ell, c\ell]$, τότε σε κάθε σημείο μπορούμε να θεωρήσουμε ότι περικλείεται με μία σφαίρα ακτίνας δ^* στην οποία θα ζουν όλα τα σημεία με τα οποία θα έχει απόσταση μικρότερη ή ίση με $c\ell$. Μέσα σε αυτή τη σφαίρα, υπάρχει σταθερός αριθμός σημείων, έστω n και συνεπώς θα γίνουν n συγκρίσεις. Με άλλα λόγια, οι συγκρίσεις θα έχουν χρονικό κόστος της τάξης Θ(n). Πιο συγκεκριμένα, μπορούμε να χωρίσουμε τη σφαίρα σε κύβους πλευράς $c\ell$, θεωρώντας ότι s_i είναι το εκάστοτε σημείο,

σε αυτές τις υποδιαιρέσεις, με τη i-οστή μικρότερη διάσταση. Εφαρμόζοντας την ίδια μέθοδο με εκείνη των Closest Pair of points, ισχύει ότι, εντός της σφαίρας, υπάρχει σταθερός αριθμός σημείων και ως εκ τούτου, για κάθε σημείο, θα γίνουν όσες συγκρίσεις όσες και ο αριθμός σημείων n που το περικλείει, δηλαδή η υπολογιστική πολυπλοκότητα του αλγορίθμου είναι της τάξης $\Theta(n)$.

Γενικά, για $d \geq 2$ διαστάσεις, ο αλγόριθμος είναι υποπρόβλημα της $d - 1$ διάστασης. Συγκεκριμένα, χωρίζουμε το χώρο σε δύο υποχώρους και ορίζουμε μεσαίο χώρο S. Με αναδρομή, λύνουμε το πρόβλημα στους δύο υποχώρους. Προβάλλουμε τα σημεία εντός μία πλάκας με πάχος δ γύρω από τον χώρο S, θεωρώντας τον χώρο που προκύπτει ως S'. Με αναδρομή, το πρόβλημα λύνεται στον χώρο S' στην $d - 1$ διάσταση. Βάσει των παραπάνω, ο αλγόριθμος ικανοποιεί τη σχέση $T(n, d) = 2T(n/2, d) + T(n, d - 1) + O(n)$ σε αντιστοιχία με τα παραπάνω. Οπότε, βάσει του Master Theorem, η υπολογιστική πολυπλοκότητα είναι της τάξης $O(n(\log n)^{d-1})$.

<u>Άσκηση 2: Πόρτες Ασφαλείας στο Κάστρο</u>

Ανακατασκευάζουμε το χάρτη, δοκιμάζοντας διαφορετικές διαμορφώσεις για τους διακόπτες. Εφόσον μπορούμε να γνωρίζουμε μέχρι την πρώτη πόρτα που παραμένει κλειστή, γνωρίζουμε τη θέση στην οποία η πρώτη πόρτα παραμένει κλειστή.
Η μέθοδος που θα ακολουθήσουμε, είναι η μέθοδος Διαίρει-και-Βασίλευε. Συγκεκριμένα, για αρχή, δοκιμάζουμε να ανεβοκατεβάσουμε με τη σειρά τους διακόπτες. Συγκεκριμένα, θεωρούμε πίνακα $A[1\ldots n]$ ο οποίος περιέχει με σειρά τα δεδομένα 0 ή 1 για το αν ο διακόπτης κάθε πόρτας είναι κατεβασμένος ή ανεβασμένος, χωρίς να γνωρίζουμε ποια θέση ανοίγει ή κλείνει τον διακόπτη κάθε πόρτας. Στην περίπτωση που κάποιος διακόπτης δεν ανοιγοκλείνει το φως κάποιας πόρτας μέχρι εκείνης που είναι κλειστή, η πόρτα που αντιστοιχεί στο διακόπτη αυτόν βρίσκεται μετά την πόρτα που είναι κλειστή (οπότε, δεν μπορούμε να δούμε να ανοιγοκλείνει φως από κάποια πόρτα). Στη συνέχεια, το αρχικό σύνολο διακοπτών (δηλαδή τα στοιχεία του πίνακα $A[1\ldots n]$), χωρίζεται σε δύο σύνολα: το υποσύνολο διακοπτών που γνωρίζουμε ότι βρίσκονται πριν τη κλειστή πόρτα και το υποσύνολο διακοπτών που βρίσκονται μετά από την κλειστή πόρτα. Συνεχίζουμε την ίδια διαδικασία ανεβοκατεβάζοντας τους διακόπτες για τα δύο υποσύνολα και χωρίζοντας

τα σε μικρότερα υποσύνολα με τον ίδιο τρόπο, ώστε να βρούμε αναδρομικά ποιος διακόπτης αντιστοιχεί σε ποια πόρτα και ποια θέση αυτού του διακόπτη διατηρεί την αντίστοιχη πόρτα ανοιχτή. Η ορθότητα αποδεικνύεται, παρόμοια με εκείνης της Mergesort, από το γεγονός ότι θα προκύψουν ταξινομημένα στοιχεία κατά σειρά και θα έχει βρεθεί η θέση του διακόπτη που ανάβει το φως της εκάστοτε πόρτας. Ο χρόνος εκτέλεσης του παραπάνω αλγορίθμου, στη χειρότερη περίπτωση, ικανοποιεί τη σχέση T(n) = 2T(n/2) + Θ(n) = {υπολογισμός αριστερού & δεξιού τμήματος} + {συγχώνευση τμημάτων}. Οπότε, οι συνολικές διαμορφώσεις που απαιτεί ο παραπάνω αλγόριθμος (βάσει του Master Theorem) είναι O($n^{\log_2 2}$ logn) = O(nlogn).

Άσκηση 3: Φόρτιση Ηλεκτρικών Αυτοκινήτων

Θεωρούμε ότι ο πίνακας αφίξεων των αυτοκινήτων A[1,…,n] έχει ταξινομημένα στοιχεία σε αύξουσα σειρά. Οπότε, σε χρόνο τάξης O(n), μπορούμε να δημιουργήσουμε τους πίνακες X[1,…,n] & Au[1,…,n] που υποδηλώνουν τις χρονικές στιγμές των αφίξεων και τα αυτοκίνητα που χρειάζονται φόρτιση αντίστοιχα. Θεωρούμε sum το πλήθος των χρονικών στιγμών, το οποίο είναι της τάξης O(n). Εφαρμόζουμε Δυαδική Αναζήτηση στο πλήθος των υποδοχών s και για το εκάστοτε s, ακολουθούμε τον εξής αλγόριθμο:

Για i από 1 ως sum-1:

Στην περίπτωση που X[i+1] ≥ X[i] + d, συνέχισε τη ροή εκτέλεσης. /* Στην περίπτωση που την επόμενη στιγμή οι αφίξεις είναι περισσότερο από d μονάδες χρόνου μακριά, συνεχίζεται η ροή εκτέλεσης*/

Αλλιώς, στην περίπτωση που Au[i] > (s)*d, εκτέλεσε δυαδική αναζήτηση στο δεξί μέρος του πίνακα. /*Στην αντίθετη περίπτωση, μπορούν να φορτιστούν όλα τα αμάξια μέσα στο χρόνο d.*/

Αλλιώς Au[i+1] = Au[i+1] + max{Au[i] – (s)*(X[i+1] – X[i]), 0} /* Στην περίπτωση που την επόμενη στιγμή οι αφίξεις είναι λιγότερο από d μονάδες χρόνου μακριά, πρέπει να μεταφέρουμε τα αυτοκίνητα που θα απομείνουν.*/

Στην περίπτωση που Au[sum] > (s)*d, εκτέλεσε δυαδική αναζήτηση στο δεξί μέρος του πίνακα.
Αλλιώς, εκτέλεσε δυαδική αναζήτηση στο αριστερό μέρος του πίνακα.

Για κάθε χρονική στιγμή άφιξης, εξυπηρετούνται s αυτοκίνητα, ενώ τα υπόλοιπα μεταφέρονται στην επόμενη χρονική στιγμή όπου επαναλαμβάνουμε τη διαδικασία. Αν υπάρξει χρονική στιγμή στην οποία υπάρχουν παραπάνω αυτοκίνητα από αυτά που μπορούμε να εξυπηρετήσουμε, χρειαζόμαστε παραπάνω υποδοχές. Στην περίπτωση που οι υποδοχές επαρκούν, αναζητούμε μία βέλτιστη λύση. Επαληθεύουμε τον παραπάνω αλγόριθμο με τον εξής τρόπο: Έστω ότι υπάρχουν t υποδοχές, η καθεμία από την οποία έχει ή δεν έχει καθόλου αυτοκίνητα αριθμημένα με τον αύξοντα αριθμό της υποδοχής. Κάθε υποδοχή, αναπαριστά μία χρονική στιγμή, στην οποία υπάρχουν αρχικά τα αυτοκίνητα που κατέφθασαν εκείνη τη χρονική στιγμή. Κάθε αυτοκίνητο με αριθμό k πρέπει να βρίσκεται το πολύ στην υποδοχή k + d - 1. Ο παραπάνω αλγόριθμος ελέγχει την περίπτωση που κάποια υποδοχή έχει παραπάνω από s αυτοκίνητα, όπου s ο υποψήφιος αριθμός υποδοχών και κατανέμει τα υπόλοιπα στην επόμενη υποδοχή με αναδρομή, ώστε το καθένα να έχει το μέγιστο s αυτοκίνητα. Στην περίπτωση που τα αυτοκίνητα δεν είναι πάνω από (s)*d, τότε εξυπηρετούνται όλα στον ζητούμενο χρόνο. Οπότε, ο αλγόριθμος ελέγχει πόσα έχει να κατανείμει στην αμέσως επόμενη υποδοχή που δεν είναι κενή. Στην επόμενη χρονική στιγμή (ή στην επόμενη υποδοχή), ελέγχεται αν μπορούν να εξυπηρετηθούν όλα τα αυτοκίνητα εντός d χρονικών στιγμών, δηλαδή αν ισχύει ότι τα αυτοκίνητα δεν είναι πάνω από (s)*d. Τα αυτοκίνητα που φορτίζονται στη δεύτερη υποδοχή είναι το μέγιστο (d - 1)*s (έχει ήδη περάσει η πρώτη χρονική στιγμή) και συνεπώς μπορούν να εξυπηρετηθούν για d - 1 χρονικές στιγμές. Τα αυτοκίνητα που υπήρχαν ήδη στην δεύτερη υποδοχή, μπορούν να εξυπηρετηθούν το αργότερο σε d – 1 + 2 = d + 1 χρονικές στιγμές, δηλαδή θα έχουν φορτιστεί τη χρονική στιγμή d + 2 (δεκτό). Από την παραπάνω επαγωγική ανάλυση, επαληθεύεται η ορθότητα του αλγορίθμου. Η δημιουργία των πινάκων χρειάζεται χρόνο τάξης O(n) και η Δυαδική Αναζήτηση χρειάζεται χρόνο O(logn). Η λύση, λοιπόν, είναι βέλτιστη, καθώς χρησιμοποιούμε Δυαδική Αναζήτηση με πολυπλοκότητα τάξης O(nlogn). Συγκεκριμένα, η παραπάνω υλοποίηση είναι η βέλτιστη δυνατή, καθώς εργαζόμαστε με τον εξής τρόπο: Στην περίπτωση που υπάρχουν αφίξεις την ίδια χρονική στιγμή, τότε ελέγχουμε εκείνες που είναι μεγαλύτερες σε πλήθος από το d (με άλλα λόγια, αν

ισχύει η σχέση {συνολικός αριθμός αυτοκινήτων}/({αριθμό εξυπηρετητών}*d) > 1, πρέπει να προστεθεί εξυπηρετητής). Αν ισχύει το προηγούμενο, τότε πρέπει να προστεθεί φορτιστής, αλλιώς συνεχίζουμε τη διαδικασία. Αν δεν υπάρχει ταυτόχρονη άφιξη, τότε ενημερώνονται πόσοι πελάτες έχουν εξυπηρετηθεί σε διάστημα Δt μεταξύ αφίξεων και συνεχίζεται η διαδικασία.

Άσκηση 4: Παραλαβή Πακέτων

1. Η προετοιμασία του δέματος για κάθε πελάτη θα εκτελεστεί σύμφωνα με τη ακολουθία βαρών δια τη μονάδα χρόνου εξυπηρέτησης. Με άλλα λόγια, το πρόβλημα με την ακολουθία $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_i}{p_i}$ (για κάθε i = 1,2,…n) μπορεί να επιλυθεί με χρήση της ταξινόμησης Radixsort σε χρόνο τάξης μεγέθους O(n). Αποδεικνύουμε την ορθότητα με τον εξής τρόπο: Έστω ότι ο αλγόριθμος δεν ακολουθούσε την παραπάνω λογική, οπότε θα υπάρχει ένα ζεύγος πελατών με δείκτες k και j τέτοιοι ώστε ο πελάτης k να εξυπηρετείται πριν από τον πελάτη j χωρίς να μεταβάλλεται η σχέση $\frac{w_k}{p_k} \geq \frac{w_j}{p_j}$ (1). Έστω ότι το ζεύγος πελατών με δείκτες k και j εξυπηρετούνται με αντίστροφη σειρά, δηλαδή να ο πελάτης j να εξυπηρετείται πριν από τον πελάτη k (2). Οπότε, το κόστος και ο χρόνος εξυπηρέτησης δε μεταβάλλονται για τους υπόλοιπους πελάτες. Ο χρόνος εξυπηρέτησης στην αρχική κατάσταση για τον πελάτη k είναι t + p_k και συνεπώς ο χρόνος εξυπηρέτησης του j είναι t + p_k + p_j.
   Στην περίπτωση (1), το συνολικό κόστος διαμορφώνεται ως:
   K_1 = {Προηγούμενο Κόστος} + $w_k(t + p_k) + w_j(t + p_k + p_j)$ (3).
   Στην περίπτωση (2), το συνολικό κόστος διαμορφώνεται ως:
   K_2 = {Προηγούμενο Κόστος} + $w_j(t + p_j) + w_k(t + p_j + p_k)$ (4).
   Αφαιρώντας τις σχέσεις (3) και (4): $K_1 - K_2$ =
   $w_k(t + p_k) + w_j(t + p_k + p_j) - w_j(t + p_j) - w_k(t + p_j + p_k)$ =
   $= p_j p_k(\frac{w_j}{p_j} - \frac{w_k}{p_k}) > 0$
   Το οποίο σημαίνει ότι $K_1 > K_2$, γεγονός που είναι άτοπο, καθώς το κόστος της περίπτωσης K_1 είναι βέλτιστο. Οπότε, η αρχική μας προσέγγιση είναι ορθή.

2. Σε αυτό το υποερώτημα, θα εργαστούμε με δυναμικό προγραμματισμό. Το πρόβλημα είναι παρόμοιο με το πρόβλημα Knapsack, αλλά πρέπει να γίνει χρήση του αποτελέσματος από το 1ο ερώτημα. Θεωρούμε ότι οι πελάτες ταξινομούνται με τη βοήθεια του 1ου ερωτήματος. Το ελάχιστο συνολικό βεβαρυμένο κόστος για τους πρώτους i πελάτες με συνολικό χρόνο εξυπηρέτησης για τον πρώτο είναι L. Οπότε, ισχύει η παρακάτω σχέση:

   C(i, L) = min{ C(i - 1,L - p_i) + Lw_i, C(i - 1,L) +w_i($\sum_{j=1}^{i} p_j - L$) }.

   Χρησιμοποιώντας τη bottom-up προσέγγιση, το τελικό κόστος είναι της τάξης O(n$\sum_{j=1}^{n} p_j$), καθώς χρειάζεται ταξινόμηση, βάσει του τρόπου που αποδείξαμε στο 1ο ερώτημα, πριν εξυπηρετηθεί το σύνολο των πελατών σε μία ουρά.

   Γενικεύουμε την παραπάνω απόδειξη για την περίπτωση που έχουμε τρεις ή περισσότερους υπαλλήλους. Στην περίπτωση που υπάρχουν m υπάλληλοι, θα εξυπηρετούνται m ουρές, οπότε η πολυπλοκότητα, παρόμοια με πριν, είναι της τάξης O(n($\sum_{j=1}^{n} p_j$)^{m}).

<u>Άσκηση 5: Ελάχιστη Διαταραχή Ακολουθίας</u>

1. Η μεγαλύτερη διαφορά σε μία υπακολουθία, προκύπτει από τα μέγιστα (max) και τα ελάχιστα (min) στοιχεία της ακολουθίας. Το μέγιστο (max) ή το ελάχιστο (min) τοποθετούνται στο τέλος, ώστε να υπολογιστούν μία φορά στην ακολουθία. Στην αντίθετη περίπτωση, αν τα στοιχεία min και max βρίσκονταν σε διαφορετικές θέσεις στην ακολουθία, το max-min θα υπολογιζόταν πάνω από μία φορά και συνεπώς το β^* δε θα ήταν η βέλτιστη ακολουθία.
2. Υπολογίζουμε τη βέλτιστη ακολουθία β^* με τη βοήθεια Ελαχίστου Σωρού, ώστε σταδιακά να μεταφέρουμε τα τελευταία στοιχεία σε πιο σημαντικές θέσεις.

   Θεωρούμε ως είσοδο την ακολουθία α, η οποία θα είναι αποθηκευμένη σε έναν πίνακα A[1…n] και αρχικά θα τοποθετηθεί σε μία ουρά προτεραιότητας με κόστος O(n).

Επιπλέον, θεωρούμε έναν πίνακα P[1..n + 1], στον οποίο θα τοποθετήσουμε τα στοιχεία της ουράς σε σωστή σειρά. Υλοποιούμε τον παρακάτω αλγόριθμο:
Για κάθε στοιχείο i της ουράς:

Απομονώνουμε το πρώτο στοιχείο c της ουράς και το αφαιρούμε από την ουρά.

Αν το c στοιχείο δεν είναι ίσο με A[i] ή i είναι ίσο με n:

P[i] = c;

Αλλιώς:

Στο P[i] ανατίθεται το αμέσως επόμενο στοιχείο της ουράς, το οποίο θα αφαιρεθεί από την ουρά και θα προστεθεί το στοιχείο c στην ουρά.

Αν P[n] = A[n]:

Ανταλλαγή στοιχείων P[n - 1] και P[n].

Η συνολική διαταραχή μπορεί να υπολογιστεί με τη βοήθεια του πρώτου ερωτήματος, καθώς πλέον γνωρίζουμε ότι το τελευταίο στοιχείο της βέλτιστης ακολουθίας β^*είναι είτε το μέγιστο είτε το ελάχιστο στοιχείο της ακολουθίας α. Στη θέση i του παραπάνω αλγορίθμου γνωρίζουμε τη μικρότερη τιμή των στοιχείων που μπορούμε να τοποθετήσουμε. Η διαδικασία είναι παρόμοια με την ανταλλαγή των γειτονικών στοιχείων των ζευγών στοιχείων του πίνακα A[1…n]. Εφόσον δεν μπορούμε να καθορίσουμε το τελευταίο στοιχείο σε σταθερό χρόνο, η συνολική πολυπλοκότητα διαμορφώνεται ως O(nlogn).