Εξηγήσεις των προβλημάτων του πρώτου εσωτερικού προκριματικού διαγωνισμού στο ΕΜΠ για τον ICPC

Algoholics ECE-NTUA

Αύγουστος 2018

Περιεχόμενα

Εισαγωγή

Στις 16 Μαΐου 2018 πραγματοποιήθηκε ο πρώτος εσωτερικός διαγωνισμός της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου με σκοπό την συγκρότηση ομάδων που θα εκπροσωπήσουν την σχολή στον διεθνή διαγωνισμό της ACM, ICPC.

Το παρόν έγγραφο, περιέχει επεξηγήσεις των λύσεων για τα δύο προβλήματα που χρησιμοποιήθηκαν στον διαγωνισμό. Ενδεικτικός κώδικας λύσεων, καθώς και εκφωνήσεις μπορούν να βρεθούν στην παρακάτω διεύθυνση:

`https://github.com/algoholics-ntua/ntua-acm-icpc/tree/master/2018`

Τα αποτελέσματα του διαγωνισμού καθώς και άλλες σχετικές πληροφορίες μπορούν να βρεθούν στην ακόλουθη σελίδα:

`https://www.hackerrank.com/contests/ntua-acmicpc-qualification/challenges`

Κεφάλαιο 1

Τραπεζικές Οφειλές

Εκφώνηση: `https://github.com/algoholics-ntua/ntua-acm-icpc/blob/master/2018/problem1/problem1.pdf`

Κάθε τραπεζική εντολή (Π, Τ1, Χ1, Τ2, Χ2) την αντιμετωπίζουμε σαν δύο ξεχωριστά συμβάντα ως εξής:

- (Π, Τ1, Χ1), η τράπεζα Τ1 χάνει Π χρηματικές μονάδες την χρονική στιγμή Χ1.
- (Π, Τ2, Χ2), η τράπεζα Τ2 λαμβάνει Π χρηματικές μονάδες την χρονική στιγμή Χ2.

Συνεπώς, έχουμε κάποια συμβάντα πλέον τα οποία εξελίσσονται στον χρόνο.Τα ταξινομούμε χρονικά και τα επεξεργαζόμαστε με την σειρά.

Είναι εύκολο να δει κανείς, ότι το να βρούμε το περισσότερο που πρέπει να δανειστεί μια τράπεζα για να μπορέσει να πραγματοποιήσει τις τραπεζικές εντολές, ισοδυναμεί με το μεγαλύτερο χρέος που θα έχει μια τράπεζα κάποια χρονική στιγμή. Συνεπώς, το μόνο που αρκεί να κάνουμε, είναι να διατηρούμε κάθε χρονική στιγμή για κάθε τράπεζα πόσο 'μέσα' έχει μπει και στο τέλος να διαλέξουμε την μεγαλύτερη από αυτές τις τιμές.

Το παραπάνω μπορεί να γίνει εύκολα σε γραμμικό χρόνο και συνεπώς έχουμε πολυπλοκότητα: $O(nlogn)$, λόγω της ταξινόμησης.

Κεφάλαιο 2

Όριο Ταχύτητας

Εκφώνηση: `https://github.com/algoholics-ntua/ntua-acm-icpc/blob/master/2018/problem2/problem2.pdf`
Παρατηρούμε πως, αν σε έναν δρόμο που διανύουμε γνωρίζουμε το όριο ταχύτητας σίγουρα πρέπει να ταξιδέψουμε με ταχύτητα ίση με το όριο, προκειμένου να πετύχουμε βέλτιστο χρόνο.

Η δυσκολία εδώ πέρα, έγκειται στο γεγονός ότι πρέπει να θυμόμαστε ακριβώς ποια είναι κάθε φορά η ταχύτητα μας προκειμένου να μπορέσουμε να χρησιμοποιήσουμε κάποιον από τους γνωστούς αλγορίθμους εύρεσης συντομότερων μονοπατιών. Είναι διαφορετικό δηλαδή να διανύουμε έναν δρόμο δεδομένου πως πριν κινούμασταν με 10 χιλιόμετρα ανά ώρα και διαφορετικό αν κινούμασταν πριν με 20.

Κάνουμε μια μικρή παραλλαγή στον αλγόριθμο του Dijkstra , αντί να διατηρούμε έναν πίνακα αποστάσεων $d(u)$: ο ελάχιστος χρόνος για να φτάσουμε στην κορυφή u διατηρούμε έναν δισδιάστατο πίνακα $d(u, S)$: ο ελάχιστος χρόνος για να φτάσουμε στην κορυφή u δεδομένου πως η προηγούμενη ταχύτητα που είχαμε ήταν S χιλιόμετρα την ώρα. Είναι προφανές, ότι σε μια κορυφή, φτάνουμε με κάποια ακέραια ταχύτητα και συνεπώς υπάρχει για κάθε κορυφή κάποιο S τέτοιο ώστε το $d(u, S)$ να περιλαμβάνει την βέλτιστη απάντηση.

Από εκεί και πέρα, αρκεί να τρέξουμε τον αλγόριθμο του Dijkstra με την διαφορά ότι πλέον το state μας αποτελείται από ζεύγη κορυφών-ταχυτήτων. Αφού υπολογίσουμε την βέλτιστη απάντηση (ελάχιστος χρόνος), αρκεί να τυπώσουμε και ένα βέλτιστο μονοπάτι χρησιμοποιώντας το δέντρο συντομότερων μονοπατιών.

Πολύ αναλυτική περιγραφή του αλγορίθμου του Dijkstra καθώς και διάφορων αλγορίθμων εύρεσης συντομότερων μονοπατιών μπορούν να βρεθούνε στα κεφάλαια 24-25 του Introduction to Algorithms των Cormen, Leiserson, Rivest, Stein.