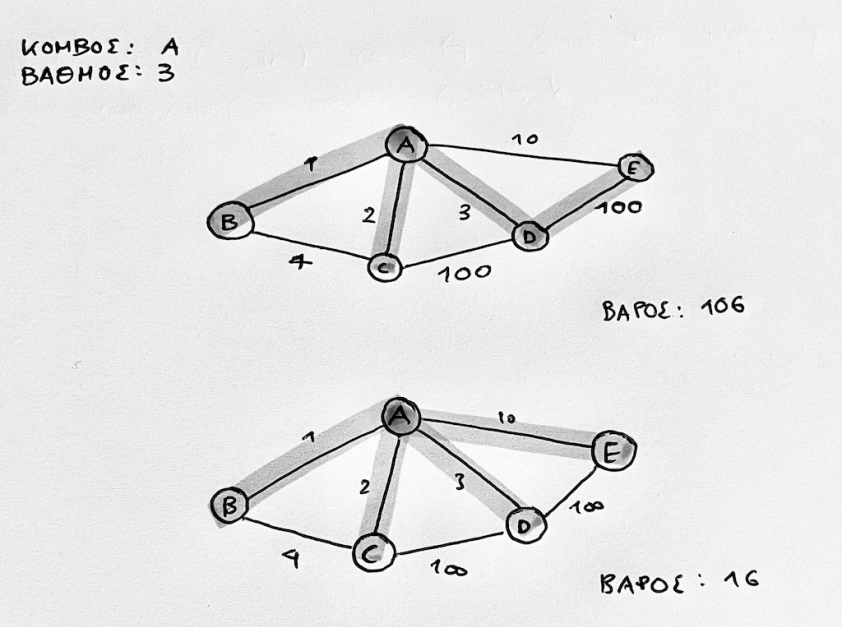
Γιώργος Αγγέλης el18030
2022

Αλγόριθμοι - 3η Σειρά Ασκήσεων

Άσκηση 1

1.

Η άπληστη επιλογή των μικρότερων ακμών για την κορυφή s (με συγκεκριμένο βαθμό) δεν οδηγεί απαραίτητα στο καλύτερο αποτέλεσμα (MST) αφού προκειμένου να αποφευχθεί δημιουργία κύκλου μπορεί να επιλεχθεί ακμή με μεγάλο βάρος που κανονικά δεν θα επιλεγόταν όπως στο παράδειγμα παρακάτω:



2.

Αναζητούμε το νέο MST:

Προσθέτουμε την τιμή X στις ακμές του κόμβου s.

Το X παίρνει τιμές {X=i-j, i κάθε ακμή που δεν είναι ακμή του ζητούμενου κόμβου και j είναι}.

Δοκιμάζουμε όλες τις τιμές του X αφού τις ταξινομήσουμε με δυαδική αναζήτησή.

Όσο αυξάνεται το X μειώνεται η πιθανότητα μια ακμή που αφορά τον συγκεκριμένο κόμβο να συμπεριλαμβάνεται στο MST και το αντίθετο.

Τα μικρά X επιτρέπουν σε όλες τις ακμές του s να συμμετάσχουν στο MST.
Τα μεγάλα X τείνουν να μην επιτρέπουν σε ακμές του s να συμμετάσχουν στο MST.

Για κάθε X της δυαδικής αναζήτησης κάνουμε Kruskal και βρίσκουμε το MST (χωρίς περιορισμούς).

Αν ο βαθμός του κόμβου είναι μικρότερος από τον ζητούμενο το X πρέπει να μικρύνει.

Στο MST* που δημιουργήθηκε τσεκάρουμε το βαθμό του επιλεγμένου κόμβου και αν είναι αυτός που ψάχνω σταματάω τον αλγόριθμο, αλλιώς συνεχίζω.

Απόδειξη Ορθότητας:

Ο αλγόριθμος για κάθε X επιλέγει αλλάζει τις ακμές του υπό δημιουργία MST με σκοπό να οδηγήσει εντέλει στο ζητούμενο MST:
Οι προσπίπτουσες στον ζητούμενο κόμβο ακμές αλλάζουν πάντα όταν αύξηση με το κατάλληλο X μιας ακμής που προσπίπτει στον ζητούμενο κόμβο αυτή γίνει ίση με μια άλλη ακμή του γράφου.

Πολυπλοκότητα:

Έστω i αριθμός των ακμών του γράφουν και j ο αριθμός των κόμβων που προσπίπτουν στον κόμβο s.

Η ταξινόμηση των X γίνεται σε O(|i||j| * log(|i||j|)).
Κάνουμε δυαδική αναζήτησή στο σύνολο των X σε O(log(|i||j|)) ≤ O(2*log(|i|)).
Kruskal σε O(|i| * log(|i|)).
Άρα συνολικά O(|i||j| * log(|i||j|) + 2|i| * $\log^2$(|i|)).

Άσκηση 2

1.

Ο αλγόριθμος που θα εφαρμόσουμε αποτελεί παραλλαγή του γνωστού αλγορίθμου Bellman-Ford.
Ο Bellman-Ford βρίσκει τα ελάχιστα μονοπάτια από έναν αρχικό κόμβο σε κάθε άλλο κόμβο, και ας υπάρχουν αρνητικές ακμές.

Χάριν αναγωγής του προβλήματος σε πρόβλημα γράφου θεωρούμε ότι τα θετικά, μηδενικά και αρνητικά βάρη του αντιστοιχούν στο φίλτρο, το κενό δωμάτιο και το τέρας, τοποθετούνται στις ακμές και όχι στις κορυφές.

Η απλή εφαρμογή του Bellman-Ford θα οδηγούσε στο ελάχιστο μονοπάτι αντικαθιστώντας κατά τις επαναλήψεις ακμές με ακμές μικρότερης απόλυτης τιμής, κάτι που δεν μας καλύπτει αφού στη δική μας περίπτωση όσο μεγαλύτερη η συνολική τιμή ενός μονοπατιού τόσο το καλύτερο.

Αρχικά αρχικοποιούμε κάθε ακμή στη τιμή -∞.

Ο αλγόριθμος μας ξεκινάει από τον κόμβο s με τιμή r και ελέγχει τους γειτονικούς του τους οποίους πιθανώς (όπως θα δούμε παρακάτω) ενημερώνει με την τιμή που συναντάει στις επόμενες ακμές, κατόπιν πάει στον επόμενο κόμβο κλπ κλπ.

Η τιμή ενός κόμβου ανανεώνεται αν οδηγούμαι σε αυτόν από ακμή με μεγαλύτερη θετική τιμή. Αν υπάρχει μονοπάτι προς τον κόμβο αυτό με μικρότερη συνολική θετική τιμή ΔΕΝ ενημερώνεται το κόστος προσπέλασης του κόμβου αυτού, όπως θα γινόταν στον κανονικό Bellman-Ford. Αν έχουμε συνολικά αρνητική τιμή από συγκεκριμένη ακμή για τον κόμβο αυτόν απορρίπτεται το μονοπάτι.

Αν στο τέλος φτάσουμε στον κόμβο t με τιμή p≥0 θα έχουμε r-ασφαλή λαβύρινθο G.

2.

Εφαρμόζουμε τον αλγόριθμο του ερωτήματος (1) |V| - 1 φορές. Εφαρμόζουμε τον αλγόριθμο μία επιπλέον φορά. Πλέον αν φτάσουμε στον κόμβο t με αρνητική τιμή δεν απορρίπτουμε τον G αμέσως αφού αν μετά την επιπλέον εκτέλεση του αλγορίθμου η τιμή του τελικού μονοπατιού αλλάξει (αυξηθεί) αυτό σημαίνει ότι υπάρχει θετικός κύκλος που έχει πρόσβαση στον κόμβο t. Τότε ο λαβύρινθος G είναι r-ασφαλής.

3.

Αθροίζουμε όλα τα βάρη των τεράτων, έστω mon. Ορίζουμε r=mon. Με binard search τρέχουμε τον παραπάνω αλγόριθμο για R στο σύνολο [0, r]. Βρίσκουμε το κατάλληλο r σε

Άσκηση 3

1.

Ο αλγόριθμος με τον οποίο θα προσεγγίσουμε το πρόβλημα είναι άπληστος. Αφού τοποθετήσουμε κάθε σταθμό ανεφοδιασμού σε μία γραμμή ανάλογα τη θέση του, ξεκινώντας από το τέλος μέσω ενός stack (για τις συγκρίσεις) βρίσκουμε το index του επόμενου σταθμού με πιο φτηνό καύσιμο από τον τρέχοντα σταθμό και το αποθηκεύουμε στη λίστα A.

Κατόπιν διατρέχουμε την A. Εάν από έναν σταθμό που βρίσκεται το όχημα μπορώ να φτάσω στον επόμενο που βρίσκεται στη λίστα A (και άρα έχει το ελάχιστο κόστος) με το καύσιμο που είχα ή μόλις έχω προμηθευτεί τότε μεταβαίνω σε αυτόν. Εάν δεν μπορώ να φτάσω διότι ο επόμενος σταθμός με οικονομικό καύσιμο βρίσκεται εκτός εμβέλειας του ντεπόζιτου, τότε (και επειδή βρίσκομαι σε σταθμό με οικονομικό καύσιμο) γεμίζω το ντεπόζιτο (αφού ξέρω ότι δεν υπάρχει άμεσα επόμενος σταθμός με μικρότερο κόστος) και πηγαίνω στον αμέσως επόμενο σταθμό (ανεξάρτητα από την τιμή που αυτός παρέχει το καύσιμο). Εκεί ελέγχω για τον επόμενο σταθμό με οικονομικό καύσιμο και ου το καθεξής.

Ορθότητα:
Ο αλγόριθμος είναι άπληστος και επειδή γνωρίζει εκ των προτέρων τους σταθμούς ανεφοδιασμού με το ελάχιστο κόστος θα επιλέξει εκείνους. Σε περίπτωση που δεν γίνεται να τους προσπελάσει λόγω μη επάρκειας σε καύσιμο ελέγχει σταθμό - σταθμό για βέλτιστες επιλογές έχοντας υπόψιν τη πληροφορία από την A).

Πολυπλοκότητα:
Ταξινόμηση σε O(nlogn), όπου n οι σταθμοί και O(n) η δημιουργία του stack και οι αναζητήσεις. Σύνολο max{O(nlogn), O(n)}=> Πολυπλοκότητα O(nlogn).

2.

Πηγή στην οποία βασίστηκε η λύση: https://www.cs.umd.edu/users/samir/grant/gas-j.pdf

Για οποιοδήποτε τυχαίο κατευθυνόμενο γράφο G θα βρούμε το μονοπάτι που οδηγεί στο ελάχιστο κόστος. Αυτό γίνεται με δυναμικό προγραμματισμό σε $O(n^2)$ και συγκεκριμένα:

Έστω:

C(i,b) = ελάχιστο κόστος για να μεταβούμε από τον σταθμό ανεφοδιασμού i στον τελικό κόμβο t με b λίτρα βενζίνης.

Έστω:

G(i) = { οι δυνατές τιμές του καυσίμου που μπορεί να υπάρχουν στο ντεπόζιτο για τον σταθμό i }
g: το υπόλοιπο σε καύσιμο

Προφανώς η βασική σχέση της αναδρομής είναι:

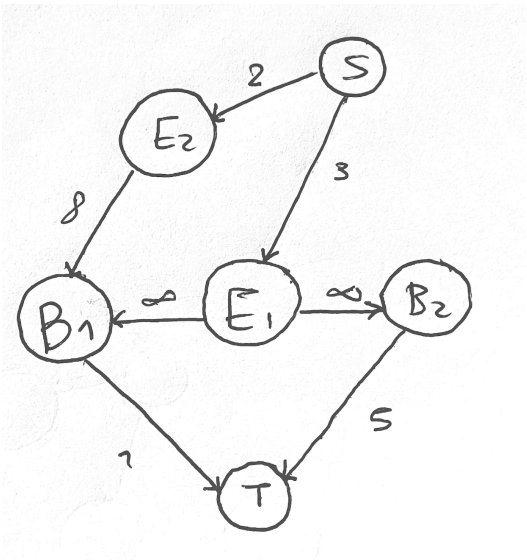
C(i, 0) = 0, i=t
$C(i, g) = \min_{d_{ij} \le B}(a, b)$

$a = C(j, 0) + (d_{ij}-g)c_i,\ c_j \le c_i \wedge g \le d_{ij}$
$b = C(j, B-d_{ij}) + (B-g)c_i,\ c_j > c_i$

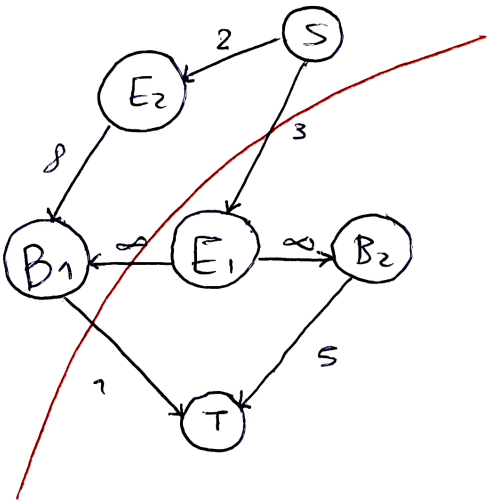
Από το C προκύπτει το ελάχιστο κόστος. Για να βρούμε το ελάχιστο μονοπάτι περνάμε τις (i, g) ως κορυφές σε ένα νέο γράφημα με ακμές τις τιμές που βγάζει η σχέση του δυναμικού προγραμματισμού. Εκτελούμε Dijkstra και βρίσουκε το ελάχιστο μονοπάτι. $O(n^2)$ κορυφές στον νέο γράφο επί n πιθανούς σταθμούς ανεφοδιασμού πολυπλοκότητα $O(n^3)$.

Άσκηση 4

(α)



(β)



Το πρόβλημα μπορεί να αναχθεί σε πρόβλημα max flow. Για να βρούμε το max flow αναζητούμε το min cut.

Πριν αναλύσουμε τον αλγόριθμο διαπιστώνουμε ότι τόσο οι θέσεις των ανταρτών και των βάσεων τους, όσο και οι θέσεις των κυβερνητικών δυνάμεων και άρα και οι μεταξύ τους αποστάσεις μας είναι γνωστές από την εκφώνηση. Άρα είναι εφικτή η οπτική απεικόνιση με την μορφή γράφου.

Για να διαχειριστούμε το συγκεκριμένο πρόβλημα κατασκευάζουμε έναν γράφο G με κορυφές τις βάσεις Bi και τους αντάρτες Ej συν δύο επιπλέον κόμβους s και t.

Κάθε βάση Bi συνδέεται με τους αντάρτες Ej με τους οποίους βρίσκεται σε κατάλληλη εμβέλεια ≤ d. Σε κάθε τέτοια ακμή τοποθετούμε τιμή ∞, αφού δεν σκοπεύουμε να λάβουμε υπόψιν μας τη συγκεκριμένη ακμή κατά την εύρεση του min cut.

Ενώνουμε τον κόμβο s με όλους τους κόμβους ανταρτών Ei.

Ενώνουμε τον κόμβο t με όλους τους κόμβους βάσεων Bi.

Είναι εμφανές ότι αναζητούμε τον βέλτιστο συνδυασμό ενεργειών, που θα δώσει το μικρότερο άθροισμα, και άρα η εφαρμογή min cut στο γράφο θα μας δώσει τις ακμές (και άρα τις ενέργειες) που απαιτούνται προκειμένου να προκύψει το ελάχιστο κόστος. Είναι προφανές ότι αγνοούμε σε αυτή τη διαδικασία τις ακμές που ενώνουν τις βάσεις με τους αντάρτες αφού δεν μας ενδιαφέρει η συγκεκριμένη πληροφορία ως ποσοτικό μέγεθος, παρά μόνο ως ποιοτικό. Τονίζουμε δηλαδή ότι δεν λαμβάνουμε υπόψιν ακμές με μη πραγματικές τιμές.

Αναπαράσταση της υλοποίησης μας φαίνεται στα σχήματα (α), (β).

Ορθότητα:
Αν ο αλγόριθμος δεν επέστρεφε το βέλτιστο συνδυασμό λύσεων αυτό σημαίνει πως θα υπήρχε άλλο min cut, άτοπο.

Πολυπλοκότητα:
Πρώτα ταξινομούμε στην ευθεία βάσεις, αντάρτες και κυβερνητικές δυνάμεις σε O((i+j+k)log(i+j+k)), όπου i+j το άθροισμα των Βάσεων και ανταρτών.

Για τη δημιουργία του γράφου G χρειαζόμαστε γραμμικό χρόνο O(i+j), όπου i+j+k το άθροισμα των Βάσεων και ανταρτών και κυβερνητικών δυνάμεων.

Για τη μέγιστη ροή χρησιμοποιήσαμε τον αλγόριθμο Ford-Fulkerson σε O(|E|*max_flow), E οι ακμές του τελικού γράφου.

Αθροιστικά:

Πολυπλοκότητα: O((i+j+k)log(i+j+k) + (i+j) + |E|*max_flow).

ΤΕΛΟΣ