Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Ροή Λ, 7ο εξάμηνο

Αλγόριθμοι και Πολυπλοκότητα

Τέταρτη σειρά γραπτών ασκήσεων

Σπουδαστής
Παπασκαρλάτος Αλέξανδρος (Α.Μ.: 03111097)

Ημερομηνία Υποβολής: 3 Φεβρουαρίου 2019

Σε κάθε ερώτημα όπου ζητείται αλγόριθμος, η αντίστοιχη απάντηση θα έχει 4 βασικά μέρη:

Ιδέα του αλγορίθμου
Η ιδέα για τον αλγόριθμο και κάποια επεξήγηση και ανάλυση

Αλγόριθμος
Η υλοποίηση του αλγορίθμου πιο αναλυτικά

Ορθότητα
Απόδειξη ορθότητας για τον αλγόριθμο και ίσως επεξήγηση.

Πολυπλοκότητα
Υπολογισμός πολυπλοκότητας του αλγορίθμου

Ωστόσο, στην πράξη κάποια από τα μέρη μπορεί να “λείπουν”.
Η Ορθότητα θα παραλείπεται όταν εξηγείται επαρκώς στην Ιδέα του αλγορίθμου και τα μέρη Ιδέα του αλγορίθμου και Αλγόριθμος θα συμπυκνώνονται σε ένα όταν είναι πιο βολικό.

<u>Άσκηση 1:</u> "Προτάσεις Φίλων"

<u>Αλγόριθμος</u>

Έστω δεδομένο i.

Για κάθε j, θέλουμε να μάθουμε αν υπάρχει μονοπάτι από το i στο j μήκους $2 \le l \le k$, τέτοιο ώστε $t(p) \ge \beta_l$.

Μας αρκεί να βρούμε το μέγιστο ως προς τη συνολική εμπιστοσύνη μονοπάτι και αν αυτό δίνει μεγαλύτερη συνολική εμπιστοσύνη από β_l , τότε προτείνουμε τον j για φίλο.

Συνεπώς, θέλουμε να μεγιστοποιήσουμε το t(p) όπως αυτό δίνεται.

Οι αλγόριθμοι που γνωρίζουμε "παίζουν" με αθροίσματα βαρών και ελαχιστοποίηση, ενώ εμάς μας έχει δοθεί σχέση με γινόμενα. Θα δοκιμάσουμε να λογαριθμήσουμε.

$$t(p) = \prod_{q=0}^{l-1} t(q, q+1) \ge \beta_l \quad \Rightarrow \quad \sum_{q=0}^{l-1} log\, t(q, q+1) \ge log\, \beta_l \quad \Rightarrow \quad \sum_{q=0}^{l-1} \frac{1}{log\, t(q,q+1)} \le -log\, \beta_l$$

Θα δημιουργήσουμε κατάλληλο κατευθυνόμενο γράφημα G με κορυφές όλους τους χρήστες.

Για κάθε i,j είναι t(i,j) < 1. Θεωρούμε πως η κορυφή i συνδέεται με την j ανν t(i,j) ≠ 0.

Θέτουμε το βάρος κάθε ακμής e=(i,j) ως $w(e) = log\frac{1}{t(i,j)} > 0$,

καθώς αν υπάρχει ακμή (i,j), είναι 0< t(i,j) < 1.

Όπως δείξαμε, στο γράφημα G, με δεδομένη κορυφή i, θέλουμε να βρούμε τα συντομότερα μονοπάτια ως προς το συνολικό βάρος από το i προς κάθε j με l πλήθος ακμών και να εξετάσουμε αν $\Sigma w(p) \le -\log \beta_l$.

Αρκεί να τρέξουμε τον αλγόριθμο Bellman-Ford στο G ως δυναμικό προγραμματισμό, όπως αυτός περιγράφεται στις διαφάνειες του μαθήματος.

Επειδή για κάθε e, w(e)>0, σίγουρα δεν προκύπτουν αρνητικοί κύκλοι.

Κάθε στήλη r του πίνακα στον αλγόριθμο αντιστοιχεί σε μονοπάτια από την κορυφή i προς όλες τις άλλες κορυφές με r με το πολύ t ακμές. Εκτελούμε τον αλγόριθμο για k στήλες.

Οπότε αφού ολοκληρώσουμε τον αλγόριθμο μπορούμε να κάνουμε άλλο ένα πέρασμα τον πίνακα και σε κάθε στήλη r, ανν το κελί j έχει αποθηκευμένο το μήκος $\Sigma w(i,j) \le -\log \beta_l$, τότε προτείνουμε τον j για φίλο (πολύ καλό παιδί).

Φυσικά, πρέπει να προσέξουμε να μην προτείνουμε τον ίδιο j δύο φορές ακόμα και αν ικανοποιεί τη συνθήκη σε περισσότερες από μία στήλες.

<u>Ορθότητα</u>

Η ορθότητα του Bellman-Ford θεωρείται δεδομένη.

Η προσαρμογή του ζητουμένου, ώστε να επιλύεται κατάλληλα με BF, αναλύθηκε ήδη.

<u>Πολυπλοκότητα</u>

Για αρχικοποίηση των αποστάσεων της κάθε κορυφής, θέλουμε O(n).

Σε κάθε iteration εξετάζουμε για την κάθε κορυφή, όλες τις ακμές της.

Συνεπώς, εξετάζουμε όλες τις ακμές.

Επειδή $l \le k$, θα κάνουμε συνολικά k iterations.

Τέλος, κάνουμε ένα πέρασμα σε όλα τα στοιχεία του πίνακα μεγέθους $k \cdot m$.

Επομένως, έχουμε συνολικά O(n + k m).

Άσκηση 2: “Τροποποίηση του Αλγορίθμου Dijkstra”

α.

Ιδέα του αλγορίθμου

Δημιουργούμε παραλλαγή του αλγορίθμου Dijkstra όπως αυτός παρουσιάζεται στις διαφάνειες του μαθήματος.
Θεωρούμε πίνακα A με θέσεις [0,1,2,...,(n-1)C].
Το κελί i του A θα αντιστοιχεί σε απόσταση i από την αρχική κορυφή s.
Συγκεκριμένα, σε κάθε βήμα, το κελί i θα περιέχει μία λίστα κορυφών, όπου όλες οι κορυφές της λίστας θα απέχουν i (εκτιμώμενη απόσταση) από την κορυφή s.
Θεωρούμε δείκτη a, ο οποίος θα δείχνει στο κελί από το οποίο θα επιλέγουμε κατάλληλο u.
Θεωρούμε δείκτη b, ο οποίος θα δείχνει στο εκάστοτε στοιχείο u της λίστας ενός κελιού.

Αλγόριθμος

Ακολουθούμε την εξής διαδικασία:

i. Αρχικά, θέτουμε την κορυφή s στο κελί 0 και αφήνουμε όλα τα υπόλοιπα κελιά άδεια.
   Θέτουμε τον a να δείχνει στο κελί 0.
   Θέτουμε τον b στο null.

ii. Επιλέγουμε το κελί στο οποίο δείχνει ο a.
   Εάν το κελί δεν είναι άδειο, θέτουμε τον b στο πρώτο στοιχείο της λίστας του κελιού.
   Διαφορετικά, περνάμε στο βήμα (v).

iii. Φτάνουμε εδώ ανν ο b δεν είναι null, δηλαδή δείχνει σε κάποια κορυφή.
   Επιλέγουμε αυτήν την κορυφή ως u και εκτελούμε αυτό το σημείο όπως στον κανονικό Dijkstra.
   Εάν ανανεώσουμε την απόσταση ενός γείτονα v, τότε μεταφέρουμε τον v στο νέο κατάλληλο κελί στον πίνακα A (αν η νέα απόσταση είναι i, το μεταφέρουμε στο κελί i), στο τέλος της λίστας που υπάρχει στο κελί.

iv. Μετακινούμε τον δείκτη b κατά μία θέση δεξιά.
   Εάν ο b δεν είναι null, περνάμε στο βήμα (iii).

v. Φτάνουμε εδώ ανν ο b είναι null.
   Μετακινούμε το δείκτη a μία θέση δεξιά.
   Εάν ο a ξεπέρασε τα όρια του πίνακα A, τερματίζουμε.
   Διαφορετικά, περνάμε στο βήμα (ii).

Ορθότητα

Η ορθότητα προκύπτει άμεσα από την ορθότητα του απλού Dijkstra. Η ουσία δεν αλλάζει.

Αρκεί να αναφέρουμε πως με την κίνηση του a, θα παίρνουμε πάντα το κελί με την ελάχιστη εκτιμώμενη απόσταση τέτοιο ώστε να έχει στοιχεία που δεν έχουμε ακόμα εξετάσει.
Αυτό ισχύει, καθώς παίρνουμε τα κελιά σε αύξουσα σειρά και εξετάζουμε κάθε στοιχείο μέσα στο εκάστοτε κελί.

Επίσης, επειδή δεν έχουμε αρνητικά βάρη, όταν θα επιλέξουμε στοιχείο u, δεν είναι εφικτό να μετακινήσουμε ένα γείτονα v σε μικρότερο κελί, οπότε σίγουρα δεν προσπερνάμε στοιχεία.
Ακόμα και αν έχουμε μια ακμή (u,v) με βάρος 0, η v θα μπει στο τέλος της λίστας στην οποία ανήκει το u, οπότε δεν θα την προσπεράσει ο b, θα εξεταστεί με τη σειρά της (βλ. βήμα iii).

Επιπλέον, αναφέρουμε πως εφόσον κάθε ακμή μπορεί να έχει βάρος το πολύ C, το μέγιστο μήκος που μπορεί να έχει ένα μονοπάτι είναι $(n-1) \cdot C$.
Συνεπώς, αρκεί το μέγεθος που θέσαμε για τον πίνακα A.

Πολυπλοκότητα

Για το βήμα (iii) χρειαζόμαστε συνολικά O(m) όπως και στον απλό Dijkstra, αφού εν τέλει θα έχουμε λάβει ως u όλα τα στοιχεία του γραφήματος (ακριβώς μία φορά) και για το καθένα εξετάζουμε όλους τους γείτονες του.
Σε κάθε εκτέλεση του (iii) αντιστοιχεί και μια μετακίνηση στοιχείου σε O(1) (διαγραφή από λίστα και προσθήκη σε λίστα) και μια μετακίνηση του δείκτη b σε O(1).

Ο a μετακινείται (n-1)C φορές, άρα έχουμε O(nC).

Επομένως, συνολικά έχουμε O(nC+m).

β Όταν υλοποιούμε Dijkstra με binary heap η πολυπλοκότητα προκύπτει Θ((m+n)logn).
Το logn οφείλεται στο ύψος του σωρού μεγέθους n (καθώς αυτό καθορίζει το χρόνο της extract-min, της insert και της decrease-key).
Μπορούμε αντί για σωρό να χρησιμοποιήσουμε δέντρο AVL (αυτο-ισορροπούμενο δυαδικό δέντρο αναζήτησης). (Με πονάει, καθώς οι σωροί είναι υπέροχοι…)

Στο AVL οι λειτουργίες αυτές χρειάζονται επίσης χρόνο logn, άρα ασυμπτωτικά δε "χαλάει" η πολυπλοκότητα όταν χρησιμοποιούμε AVL ως σειρά προτεραιότητας.
Στη συγκεκριμένη περίπτωση, βολεύει περισσότερο το AVL, γιατί έχουμε εύκολο τρόπο να διαχειριζόμαστε όμοιες τιμές.
Συγκεκριμένα, κάθε κόμβος του AVL θα διαθέτει μια λίστα όλων των κορυφών του γραφήματος που αντιστοιχούν σε αυτήν την τιμή εκτιμώμενης απόστασης. Σε σωρό αυτό δεν είναι εύκολο να γίνει καθώς σε κάθε προσθήκη, θα πρέπει κάπως να βρίσκουμε τον κόμβο με όμοια τιμή.
Έτσι, εφόσον στο AVL που περιγράψαμε, κάθε κόμβος θα έχει μοναδική τιμή D, θα έχουμε αναγκαία το πολύ D_{max}-D_{min} κόμβους στο δέντρο ανά πάσα στιγμή.
Αρκεί να αποδείξουμε πως D_{max}-$D_{min} \leq 2^C$, οπότε κάθε extract-min και decrease-key θα χρειάζεται log(D_{max}-D_{min}) ≤ C και η συνολική πολυπλοκότητα προκύπτει O((n + m)C).

<u>Απόδειξη με επαγωγή πως D_{max}-$D_{min} \leq 2^C$ σε κάθε iteration i του αλγορίθμου</u>
<u>Επαγωγική Βάση</u>
Αρχικά, μπαίνει μόνο το στοιχείο s, οπότε D_{max}-D_{min} = 0.

<u>Επαγωγική Υπόθεση</u>
Στο i-οστό iteration ισχύει πως D_{max}-$D_{min} \leq 2^C$

<u>Επαγωγικό Βήμα</u>
Στο (i+1)-οστό iteration έχουμε D'_{max} και D'_{min} .
Στο i-οστό iteration, επιλέγεται ως u, το στοιχείο με D(u) = D_{min} .
Δεν αυξάνεται η απόσταση κανενός στοιχείου που είναι ήδη στο δέντρο.
Μπορεί επιπλέον να προστεθεί νέο στοιχείο v στο δέντρο, το οποίο πρόκειται για γείτονα του u, οπότε από εκφώνηση e = (u,v), w(e) ≤ 2^C ⇒ D(v) ≤ D(u) + 2^C

Επίσης, από τη λειτουργία του Dijkstra, γνωρίζουμε πως το νέο ελάχιστο στοιχείο δεν μπορεί να είναι μικρότερο από το προηγούμενο.
Συνεπώς, είναι $D'_{max} \leq \max\{D_{max}, D_{min} + 2^C\}$
και $D'_{min} \geq D_{min} \Leftrightarrow - D'_{min} \leq - D_{min}$

Διακρίνουμε περιπτώσεις: α. $D'_{max} - D'_{min} \leq D_{max} - D_{min} \leq 2^C$
β. $D'_{max} - D'_{min} \leq D_{min} + 2^C - D_{min} = 2^C$

Βλέπουμε πως και στις δύο περιπτώσεις, είναι $D'_{max} - D'_{min} \leq 2^C$, οπότε σε κάθε περίπτωση αποδείξαμε το ζητούμενο.

Επομένως, όπως προείπαμε $D_{max} - D_{min} \leq 2^C$ ⇒ log(D_{max}-D_{min}) ≤ C και η συνολική πολυπλοκότητα με ουρά προτεραιότητας δέντρο AVL, είναι O((n + m)C).

<u>Άσκηση 4:</u> "Διαφημίσεις στο Διαδίκτυο"

<u>Ιδέα του αλγορίθμου</u>

Πρόκειται για πρόβλημα μέγιστης ροής.
Σε κάθε επισκέπτη αντιστοιχεί ακριβώς μία διαφήμιση.
Επίσης, στην εκάστοτε εταιρία i αντιστοιχούν το πολύ c_i διαφημίσεις.
Επιπλέον, σε κάθε εταιρία αντιστοιχεί ένα σύνολο κατηγοριών, και σε κάθε επισκέπτη αντιστοιχεί ένα σύνολο κατηγοριών.

<u>Αλγόριθμος</u>

ι. Θεωρούμε 5-μερές γράφημα G (στην πράξη είναι και διμερές).
Το πρώτο μέρος (Α) είναι μία κορυφή s (πηγή).
Το δεύτερο μέρος (Β) είναι m κορυφές που αντιστοιχούν στις εταιρίες.
Το τρίτο μέρος (Γ) είναι k κορυφές που αντιστοιχούν στις κατηγορίες.
Το τέταρτο μέρος (Δ) είναι n κορυφές που αντιστοιχούν στους επισκέπτες.
Το πέμπτο μέρος (Ε) είναι η κορυφή t (προορισμός).

ιι. Θέτουμε ακμές στο G ως εξής:
Για κάθε εταιρία i, συνδέουμε με ακμή χωρητικότητας c_i την κορυφή s με την αντίστοιχη κορυφή-εταιρία i του Β μέρους.

Για κάθε εταιρία i και σύνολο κατηγοριών S_i συνδέουμε με ακμή χωρητικότητας c_i την κορυφή-εταιρία i του Β μέρους με τις αντίστοιχες κορυφές-κατηγορίες του Γ μέρους.

Για κάθε κατηγορία j, συνδέουμε με ακμή χωρητικότητας 1, την κορυφή-κατηγορία j του Γ μέρους με τις κορυφές-επισκέπτες l του Δ μέρους, ανν ο επισκέπτης l ανήκει στην κατηγορία j.

Για κάθε επισκέπτη l, συνδέουμε με ακμή χωρητικότητας 1 την κορυφή-επισκέπτη l του Δ μέρους με την κορυφή προορισμό t.

ιιι. Υπολογίζουμε τη μέγιστη s-t ροή στο G.
Ανν η μέγιστη ροή ισούται με n, τότε είναι πράγματι εφικτό το γεγονός που περιγράφεται.

Ορθότητα

Σύμφωνα με τον τρόπο που αντιστοιχήσαμε τις κορυφές του G με τις εταιρίες, τις κατηγορίες και τους ανθρώπους, καθώς και τις ακμές που ορίσαμε και τη χωρητικότητα αυτών, βλέπουμε τα εξής.

Από την εταιρία i, θα "περάσουν" το πολύ c_i διαφημίσεις καθώς αυτή είναι η χωρητικότητα της ακμής από την s στη κορυφή-εταιρία i.

Από την εταιρία i, θα φτάσουμε μόνο σε κατηγορίες του συνόλου S_i.

Από την κατηγορία j, θα φτάσουμε μόνο σε επισκέπτες οι οποίοι ανήκουν στην κατηγορία αυτή.

Από τον κάθε επισκέπτη l θα "περάσει" το πολύ μία διαφήμιση καθώς αυτή είναι η χωρητικότητα από την κορυφή-επισκέπτη l στην κορυφή-προορισμό t, άρα δεν παραβιάζεται κάποιος κανόνας.

Επειδή αναζητούμε τη μέγιστη ροή, ιδανικά θέλουμε να "περάσει" μία διαφήμιση από κάθε επισκέπτη, οπότε και θα φτάσει ροή μεγέθους n στον προορισμό t.

Σημείωση: Στην πραγματικότητα, οι ακμές από το Β στο Γ και από το Γ στο Δ, θα μπορούσαν να έχουν οσοδήποτε μεγαλύτερη χωρητικότητα (μέχρι και ∞) από αυτή που τους αναθέσαμε.

Έτσι κι αλλιώς, έχουμε κατάλληλα bottleneck από το Α στο Β και από το Δ στο Ε.

Κάτι τέτοιο βέβαια ίσως να είναι λίγο πλεοναστικό (αν και εξίσου ορθό).

Η χωρητικότητα που τους αναθέσαμε είναι η ελάχιστη ασφαλής τιμή.

Πολυπλοκότητα

Έστω Q_j το σύνολο των επισκεπτών που ανήκουν στην κατηγορία j.

Για τη δημιουργία του γράφου G, θέτουμε n+m+k+2 κορυφές.

Επιπλέον, θέτουμε $m + \sum_{i=1}^{m} |S_i| + \sum_{i=1}^{m} |Q_i| + n$ ακμές.

Σημειώνουμε πως $\sum_{i=1}^{m} |S_i| \le m \cdot k$ και $\sum_{i=1}^{m} |Q_i| \le k \cdot n$.

Συνεπώς, για τη δημιουργία του γράφου G χρειαζόμαστε
$O(n + m + k + 2 + m + m \cdot k + k \cdot n + n) = O(m \cdot k + k \cdot n)$.

Για τον υπολογισμό της μέγιστης ροής (αποδοτικά), χρειαζόμαστε $O(|V|\ |E|\ \log|V|)$ ή $O(|V|^3)$ (σύμφωνα με τις διαφάνειες του μαθήματος), οπότε αυτό το κομμάτι σαφώς υπερισχύει έναντι της δημιουργίας του γράφου.

Επομένως, έχουμε συνολική πολυπλοκότητα $O((n+m+k)^3)$.