ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

Σχεδιασμός αλγορίθμων

JON KLEINBERG • ÉVA TARDOS

Επιστημονική επιμέλεια ελληνικής έκδοσης
Χρήστος Δ. Ζαρολιάγκης, *Πανεπιστήμιο Πατρών*

Περιεχόμενα

με τον αλγόριθμο του Απώτατου Μελλοντικού Στοιχείου. Θα εξετάσουμε αυτή την ανάλυση, καθώς και την ανάλυση μιας τυχαιοποιημένης παραλλαγής της αρχής LRU, όταν θα επιστρέψουμε στο πρόβλημα της χρήσης κρυφής μνήμης στο Κεφάλαιο 13.

4.4 Συντομότερες διαδρομές σε ένα γράφημα

Κάποιοι από τους βασικούς αλγορίθμους για γραφήματα βασίζονται σε αρχές άπληστου σχεδιασμού. Στην ενότητα αυτή θα εφαρμόσουμε έναν άπληστο αλγόριθμο στο πρόβλημα της εύρεσης των συντομότερων διαδρομών, ενώ στην επόμενη ενότητα θα εξετάσουμε την κατασκευή γεννητικών δένδρων ελάχιστου κόστους.

Το πρόβλημα

Όπως έχουμε δει, τα γραφήματα χρησιμοποιούνται συχνά για τη μοντελοποίηση δικτύων στα οποία ταξιδεύει κανείς από το ένα σημείο στο άλλο — διασχίζοντας μια ακολουθία λεωφόρων μέσω κόμβων, ή διασχίζοντας μια ακολουθία συνδέσμων επικοινωνίας μέσω ενδιάμεσων δρομολογητών. Κατά συνέπεια, ένα βασικό αλγοριθμικό πρόβλημα είναι ο προσδιορισμός της συντομότερης διαδρομής μεταξύ των κόμβων ενός γραφήματος. Μπορεί να το θέσουμε αυτό ως ερώτηση σημείου προς σημείο: Δεδομένων των κόμβων u και v, ποια είναι η συντομότερη διαδρομή u-v; Ή μπορεί να ζητήσουμε περισσότερες πληροφορίες: Δεδομένου ενός *κόμβου εκκίνησης* s, ποια είναι η συντομότερη διαδρομή από τον s προς κάθε άλλο κόμβο;

Η συγκεκριμένη διευθέτηση του προβλήματος συντομότερης διαδρομής είναι η εξής. Μας δίνεται ένα κατευθυνόμενο γράφημα $G = (V, E)$, με έναν καθορισμένο κόμβο εκκίνησης s. Υποθέτουμε ότι ο s διαθέτει διαδρομή προς όλους τους κόμβους του γραφήματος G. Κάθε ακμή e έχει μήκος $l_e \geq 0$, το οποίο δηλώνει το χρόνο (ή την απόσταση, ή το κόστος) που απαιτείται για τη διάσχιση της ακμής e. Για μια διαδρομή P, το *μήκος της* P — που συμβολίζεται με $l(P)$ — είναι το άθροισμα των μηκών όλων των ακμών της P. Ο σκοπός μας είναι να προσδιορίσουμε τη συντομότερη διαδρομή από τον s προς οποιονδήποτε άλλον κόμβο του γραφήματος. Θα πρέπει να αναφέρουμε ότι, αν και το πρόβλημα καθορίζεται για ένα κατευθυνόμενο γράφημα, μπορούμε να χειριστούμε την περίπτωση του μη κατευθυνόμενου γραφήματος αντικαθιστώντας απλώς κάθε μη κατευθυνόμενη ακμή $e = (u, v)$ μήκους l_e με δύο κατευθυνόμενες ακμές (u, v) και (v, u), κάθε μία μήκους l_e.

Σχεδιασμός του αλγορίθμου

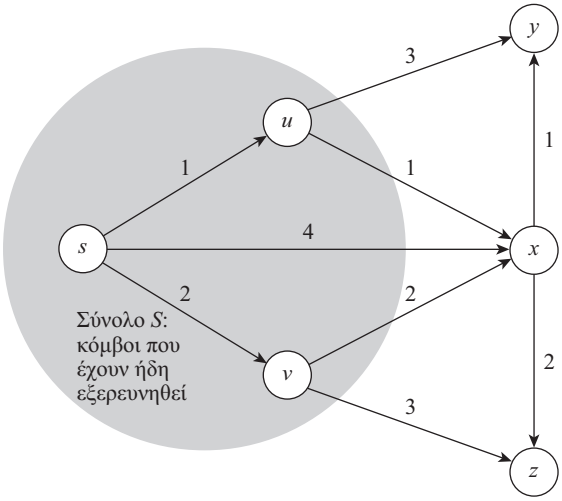
Το 1959 ο Edsger Dijkstra πρότεινε έναν πολύ απλό άπληστο αλγόριθμο για την επίλυση του προβλήματος των συντομότερων διαδρομών μίας αφετηρίας (single-source shortest-paths). Θα ξεκινήσουμε με την περιγραφή ενός αλγορίθμου ο οποίος απλώς προσδιορίζει το *μήκος* της συντομότερης διαδρομής από τον s προς οποιονδήποτε

άλλον κόμβο του γραφήματος· κατόπιν είναι εύκολο να παραγάγουμε και τις ίδιες τις διαδρομές. Ο αλγόριθμος διατηρεί ένα σύνολο S από κορυφές u για τις οποίες έχουμε προσδιορίσει την απόσταση της συντομότερης διαδρομής $d(u)$ από τον κόμβο s· αυτό είναι το "εξερευνημένο" τμήμα του γραφήματος. Αρχικά $S = \{s\}$ και $d(s) = 0$. Τώρα, για κάθε κόμβο $v \in V - S$, προσδιορίζουμε τη συντομότερη διαδρομή που μπορεί να κατασκευαστεί αν διατρέξουμε μια διαδρομή μέσω του εξερευνημένου τμήματος S προς κάποιον κόμβο $u \in S$, και στη συνέχεια ακολουθήσουμε την ακμή (u, v). Με άλλα λόγια, εξετάζουμε την ποσότητα $d'(v) = \min_{e=(u, v):u \in S} d(u) + l_e$. Επιλέγουμε τον κόμβο $v \in V - S$ για τον οποίο αυτή η ποσότητα είναι η ελάχιστη, προσθέτουμε τον v στο S, και ορίζουμε ότι το $d(v)$ είναι η τιμή $d'(v)$.

```
Αλγόριθμος του Dijkstra (G, l)
Έστω S το σύνολο των εξερευνημένων κόμβων
     For each u ∈ S, αποθηκεύουμε μια απόσταση d(u)
Αρχικά S = {s} και d(s) = 0.
While S≠V
     Διάλεξε έναν κόμβο v ∉ S με τουλάχιστον μία ακμή από το S για τον
        οποίο το d'(v) = min_{e=(u,v):u∈S} d(u) + l_e είναι όσο το δυνατόν μικρότερο
     Πρόσθεσε το v στο S και όρισε d(v) = d'(v)
EndWhile
```

Είναι εύκολο να παραγάγουμε τις διαδρομές s-u που αντιστοιχούν στις αποστάσεις τις οποίες βρίσκει ο αλγόριθμος του Dijkstra. Καθώς κάθε κόμβος v προστίθεται στο σύνολο S, καταγράφουμε απλώς την ακμή (u, v) με την οποία επιτυγχάνεται η τιμή $\min_{e=(u, v):u \in S} d(u) + l_e$. Η διαδρομή P_v αναπαριστάνεται έμμεσα από τις ακμές αυτές: αν (u, v) είναι η ακμή που έχουμε αποθηκεύσει για τον κόμβο v, τότε η P_v είναι απλώς (αναδρομικά) η διαδρομή P_u ακολουθούμενη από την ακμή (u, v). Με άλλα λόγια, για να κατασκευάσουμε την P_v, ξεκινάμε απλώς από τον κόμβο v· ακολουθούμε την ακμή που έχουμε αποθηκεύσει για τον v κατά την αντίστροφη κατεύθυνση, ώστε να φτάσουμε στον κόμβο u· μετά ακολουθούμε προς την αντίστροφη κατεύθυνση την ακμή που έχουμε αποθηκεύσει για τον κόμβο u ώστε να φτάσουμε στον πρόκατοχό του, κ.ο.κ. μέχρι να φθάσουμε στον s. Σημειώστε ότι θα πρέπει να φτάσουμε τελικά στον κόμβο s, επειδή η αντίστροφη πορεία μας από τον κόμβο v επισκέπτεται συνεχώς κόμβους που έχουμε προσθέσει σε προηγούμενα βήματα στο S.

Για να αποκτήσουμε μια καλύτερη αίσθηση για το τι κάνει ο αλγόριθμος, εξετάστε το στιγμιότυπο εκτέλεσής του που φαίνεται στην Εικόνα 4.7. Στο σημείο που φαίνεται στην εικόνα έχουν εκτελεστεί δύο επαναλήψεις: η πρώτη πρόσθεσε τον κόμβο u και η δεύτερη πρόσθεσε τον κόμβο v. Στην επανάληψη που πρόκειται να εκτελεστεί θα προστεθεί ο κόμβος x, επειδή επιτυγχάνει την μικρότερη απόσταση $d'(x)$· χάρη στην ακμή (u, x) έχουμε $d'(x) = d(u) + l_{ux} = 2$. Παρατηρήστε ότι η προσπάθεια να προστεθεί ο κόμβος y ή ο κόμβος z στο σύνολο S σε αυτό το σημείο θα οδηγήσει σε λανθασμένη τιμή για τις αποστάσεις των συντομότερων διαδρομών τους· τελικά, οι κόμβοι αυτοί θα προστεθούν λόγω των ακμών τους από τον κόμβο x.



Εικόνα 4.7 Ένα στιγμιότυπο της εκτέλεσης του αλγορίθμου του Dijkstra. Ο επόμενος κόμβος που θα προστεθεί στο σύνολο S είναι ο x, λόγω της διαδρομής μέσω του u.

Ανάλυση του αλγορίθμου

Σε αυτό το παράδειγμα βλέπουμε ότι ο αλγόριθμος του Dijkstra κάνει το σωστό και αποφεύγει τις συνεχείς παγίδες: η επαύξηση του συνόλου S με λανθασμένο κόμβο μπορεί να οδηγήσει σε υπερεκτίμηση της απόστασης συντομότερης διαδρομής προς αυτόν τον κόμβο. Το ερώτημα είναι το ακόλουθο: Είναι πάντοτε αληθές ότι, όταν ο αλγόριθμος του Dijkstra προσθέτει έναν κόμβο v, παίρνουμε πραγματικά την απόσταση της συντομότερης διαδρομής προς τον v;

Θα απαντήσουμε τώρα σε αυτό το ερώτημα αποδεικνύοντας την ορθότητα του αλγορίθμου, και δείχνοντας ότι οι διαδρομές P_u είναι πράγματι οι συντομότερες διαδρομές. Ο αλγόριθμος του Dijkstra είναι άπληστος, με την έννοια ότι πάντοτε σχηματίζουμε τη νέα συντομότερη διαδρομή s-v που μπορούμε να δημιουργήσουμε από μια διαδρομή του S που ακολουθείται από μία μόνο ακμή. Θα αποδείξουμε την ορθότητα του αλγορίθμου αυτού χρησιμοποιώντας μια παραλλαγή του πρώτου μας στυλ ανάλυσης: θα δείξουμε ότι ο αλγόριθμος "υπερτερεί" από όλες τις άλλες λύσεις αποδεικνύοντας επαγωγικά ότι κάθε φορά που επιλέγει μία διαδρομή προς έναν κόμβο v, αυτή η διαδρομή είναι η συντομότερη από κάθε άλλη δυνατή διαδρομή προς τον κόμβο v.

(4.14) *Θεωρήστε το σύνολο S σε κάθε σημείο της εκτέλεσης του αλγορίθμου. Για κάθε $u \in S$, η διαδρομή P_u είναι η συντομότερη διαδρομή s-u.*

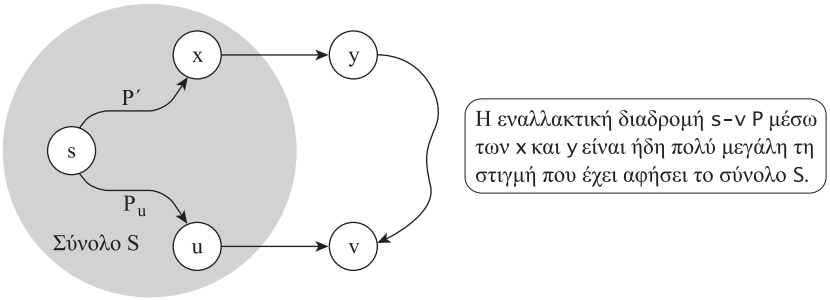
Σημειώστε ότι αυτό το γεγονός αποδεικνύει αμέσως την ορθότητα του αλγορίθμου του Dijkstra, καθώς μπορούμε απλώς να το εφαρμόσουμε στο σημείο τερματισμού του αλγορίθμου, όπου το S περιλαμβάνει όλους τους κόμβους.

Απόδειξη. Θα αποδείξουμε αυτήν την πρόταση με επαγωγή ως προς το μέγεθος του S. Η περίπτωση $|S| = 1$ είναι εύκολη, αφού τότε έχουμε $S = \{s\}$ και $d(s) = 0$. Υποθέστε

ότι ο ισχυρισμός ισχύει όταν $|S| = k$ για κάποια τιμή $k \geq 1$· αυξάνουμε τώρα το μέγεθος του S σε $k + 1$ με την προσθήκη του κόμβου v. Έστω ότι (u, v) είναι η τελική ακμή της διαδρομής s-v P_v.

Λόγω της υπόθεσης της επαγωγής, η P_u είναι η συντομότερη διαδρομή s-u για οποιοδήποτε $u \in S$. Θεωρήστε τώρα κάθε άλλη διαδρομή s-v P'· θέλουμε να δείξουμε ότι έχει τουλάχιστον το ίδιο μήκος με την P_v. Για να φθάσει στον κόμβο v, αυτή η διαδρομή P πρέπει *κάπου* να αφήσει το σύνολο S· έστω ότι y είναι ο πρώτος κόμβος της P που δεν ανήκει στο S, και έστω $x \in S$ ο κόμβος ακριβώς πριν από τον y.

Η κατάσταση αυτή φαίνεται στην Εικόνα 4.8, και η ουσία της απόδειξης είναι πολύ απλή: η P δεν μπορεί να είναι συντομότερη από την P_v επειδή έχει ήδη τουλάχιστον το ίδιο μήκος με την P_v τη στιγμή που άφησε το σύνολο S. Πράγματι, στην επανάληψη k+1 ο αλγόριθμος του Dijkstra πρέπει να έχει εξετάσει την προσθήκη του κόμβου y στο σύνολο S μέσω της ακμής (x, y) και απέρριψε αυτή την επιλογή έναντι της προσθήκης του v. Αυτό σημαίνει ότι δεν υπάρχει διαδρομή από τον s στον y μέσω του x που να είναι συντομότερη από την P_v. Όμως η μερική διαδρομή της P μέχρι τον κόμβο y είναι μια τέτοια διαδρομή, οπότε αυτή η μερική διαδρομή έχει τουλάχιστον το ίδιο μήκος με την P_v. Επειδή τα μήκη των ακμών δεν είναι αρνητικά, η πλήρης διαδρομή P έχει τουλάχιστον το ίδιο μήκος με την P_v.



Εικόνα 4.8 Η συντομότερη διαδρομή P_v και μία εναλλακτική s-v διαδρομή P μέσω του κόμβου y.

Αυτό μας δίνει την πλήρη απόδειξη· μπορούμε να διατυπώσουμε τον ισχυρισμό της προηγούμενης παραγράφου χρησιμοποιώντας τις ακόλουθες ανισότητες. Έστω P' η μερική διαδρομή της P από τον κόμβο s στον κόμβο x. Επειδή $x \in S$, γνωρίζουμε από την υπόθεση της επαγωγής ότι η P_x είναι η συντομότερη διαδρομή s-x (μήκους $d(x)$), οπότε $l(P') \geq l(P_x) = d(x)$. Έτσι η μερική διαδρομή της P από τον κόμβο y έχει μήκος $l(P') + l(x, y) \geq d(x) + l(x, y) \geq d'(y)$, και η πλήρης διαδρομή P έχει τουλάχιστον το ίδιο μήκος με αυτή τη μερική διαδρομή. Τέλος, επειδή ο αλγόριθμος του Dijkstra επέλεξε τον κόμβο v σε αυτή την επανάληψη, γνωρίζουμε ότι $d'(y) \geq d'(v) = l(P_v)$. Ο συνδυασμός αυτών των ανισοτήτων δείχνει ότι $l(P) \geq l(P') + l(x, y) \geq l(P_v)$. ■

Ας αναφέρουμε δύο παρατηρήσεις σχετικά με τον αλγόριθμο του Dijkstra και την ανάλυσή του. Πρώτον, ο αλγόριθμος δεν βρίσκει πάντοτε τις συντομότερες διαδρομές αν κάποιες από τις ακμές μπορούν να έχουν αρνητικά μήκη. (Βλέπετε σε ποιο σημείο "χαλάει" η απόδειξη;) Πολλές εφαρμογές συντομότερης διαδρομής περιλαμβάνουν αρ-

νητικά μήκη ακμών, και σε αυτή την περίπτωση απαιτείται ένας πιο πολύπλοκος αλγόριθμος — αυτός των Bellman και Ford. Θα δούμε αυτόν τον αλγόριθμο όταν θα εξετάσουμε το θέμα του δυναμικού προγραμματισμού.

Η δεύτερη παρατήρηση είναι ότι ο αλγόριθμος του Dijkstra είναι, κατά μία έννοια, ακόμη απλούστερος απ′ ό,τι τον περιγράψαμε εδώ. Ο αλγόριθμος του Dijkstra είναι στην πραγματικότητα μια "συνεχής" εκδοχή του αλγορίθμου αναζήτησης πρώτα κατά πλάτος για τη διάσχιση ενός γραφήματος, και το κίνητρό του είναι η ακόλουθη φυσική διαισθητική ιδέα. Υποθέστε ότι οι ακμές του γραφήματος G σχηματίζουν ένα σύστημα σωληνώσεων γεμάτων με νερό, που συνδέονται στους κόμβους· κάθε ακμή e έχει μήκος l_e και σταθερή διατομή. Υποθέστε τώρα ότι πέφτει μία ακόμα σταγόνα νερού στον κόμβο s και ξεκινά ένα κύμα από τον s. Καθώς το κύμα επεκτείνεται από τον κόμβο s με σταθερή ταχύτητα, η διογκούμενη σφαίρα του μετώπου του κύματος φθάνει στους κόμβους κατά αύξουσα σειρά της απόστασής τους από τον s. Είναι εύκολο να πιστέψουμε (και ισχύει φυσικά) ότι η διαδρομή που ακολούθησε το μέτωπο του κύματος για να πάει σε κάποιον κόμβο v είναι η συντομότερη διαδρομή. Πράγματι, είναι εύκολο να δούμε ότι αυτή ακριβώς είναι η διαδρομή προς τον v που βρίσκει ο αλγόριθμος του Dijkstra, και ότι οι κόμβοι που ανακαλύπτονται από το "διογκούμενο νερό" είναι στην ίδια σειρά με αυτήν που ανακαλύπτονται από τον αλγόριθμο του Dijkstra.

Υλοποίηση και χρόνος εκτέλεσης Για να ολοκληρώσουμε την ανάλυσή μας για τον αλγόριθμο του Dijkstra, θα εξετάσουμε το χρόνο εκτέλεσής του. Υπάρχουν n-1 επαναλήψεις του βρόχου `While` για ένα γράφημα με n κόμβους, καθώς κάθε επανάληψη προσθέτει ένα νέο κόμβο v στο S. Η επιλογή του σωστού κόμβου v με αποδοτικό τρόπο είναι ένα πιο δύσκολο ζήτημα. Η πρώτη εντύπωση είναι ότι κάθε επανάληψη θα πρέπει να εξετάσει κάθε κόμβο $v \notin S$, και να διατρέξει όλες τις ακμές μεταξύ του S και του v για να προσδιορίσει την ελάχιστη ποσότητα $\min_{e=(u, v):u \in S} d(u) + l_e$, έτσι ώστε να μπορεί να επιλέξει τον κόμβο v για τον οποίο αυτή η ποσότητα είναι η μικρότερη. Για ένα γράφημα με m ακμές ο υπολογισμός αυτών των ελάχιστων τιμών μπορεί να χρειαστεί $O(m)$ χρόνο, οπότε αυτό θα οδηγήσει σε μία υλοποίηση που εκτελείται σε $O(mn)$ χρόνο.

Μπορούμε όμως να τα πάμε καλύτερα αν χρησιμοποιήσουμε τις σωστές δομές δεδομένων. Πρώτον, θα διατηρούμε ρητά τις τιμές των ελάχιστων ποσοτήτων $d′(v) = \min_{e=(u, v):u \in S} d(u) + l_e$ για κάθε κόμβο $v \in V - S$, αντί να τις υπολογίζουμε εκ νέου σε κάθε επανάληψη. Μπορούμε να βελτιώσουμε ακόμα περισσότερο την απόδοση διατηρώντας τους κόμβους $V - S$ σε μια ουρά προτεραιότητας, με τις τιμές $d′(v)$ ως κλειδιά τους. Οι ουρές προτεραιότητας εξετάστηκαν στο Κεφάλαιο 2· είναι δομές δεδομένων που έχουν σχεδιαστεί να διατηρούν ένα σύνολο n στοιχείων, το καθένα από τα οποία διαθέτει ένα κλειδί. Σε μια ουρά προτεραιότητας μπορούμε με αποδοτικό τρόπο να εισάγουμε στοιχεία, να διαγράφουμε στοιχεία, να αλλάζουμε το κλειδί ενός στοιχείου, και να εξάγουμε το στοιχείο με το ελάχιστο κλειδί. Θα χρειαστούμε την τρίτη και την τέταρτη από τις παραπάνω λειτουργίες: τις λειτουργίες `ChangeKey` και `ExtractMin`.

Πώς θα υλοποιήσουμε τον αλγόριθμο του Dijkstra με τη χρήση μιας ουράς προτεραιότητας; Τοποθετούμε τους κόμβους V σε μία ουρά προτεραιότητας με το $d'(v)$ ως κλειδί για $v \in V$. Για να επιλέξουμε τον κόμβο v που θα πρέπει να προστεθεί στο σύνολο S, χρειαζόμαστε τη λειτουργία `ExtractMin`. Για να δείτε πώς γίνεται η ενημέρωση των κλειδιών, θεωρήστε μια επανάληψη στην οποία ο κόμβος v προστίθεται στο S, και έστω $w \notin S$ ένας κόμβος που παραμένει στην ουρά προτεραιότητας. Τι πρέπει να κάνουμε για να ενημερώσουμε την τιμή του $d'(w)$; Αν το (v, w) δεν είναι ακμή, τότε δεν χρειάζεται να κάνουμε τίποτε: το σύνολο των ακμών που εξετάζονται στην ελάχιστη ποσότητα $\min_{e=(u,w):u\in S} d(u) + l_e$ είναι ακριβώς το ίδιο με πριν και μετά την προσθήκη του v στο S. Από την άλλη πλευρά, αν $e' = (v, w) \in E$, τότε η νέα τιμή για το κλειδί είναι $\min(d'(w), d(v) + l_{e'})$. Αν $d'(w) > d(v) + l_{e'}$ τότε πρέπει να χρησιμοποιήσουμε τη λειτουργία `ChangeKey` για να μειώσουμε κατάλληλα το κλειδί του κόμβου w. Αυτή η λειτουργία `ChangeKey` μπορεί να συμβεί το πολύ μία φορά για κάθε ακμή, όταν το τέλος της ακμής e' προστίθεται στο S. Συνοψίζοντας, έχουμε το ακόλουθο αποτέλεσμα.

(4.15) *Αν χρησιμοποιήσουμε ουρά προτεραιότητας, μπορούμε να υλοποιήσουμε τον αλγόριθμο του Dijkstra για ένα γράφημα με n κόμβους και m ακμές έτσι ώστε να εκτελείται σε χρόνο $O(m)$, συν το χρόνο για τις n λειτουργίες* `ExtractMin` *και τις m λειτουργίες* `ChangeKey`.

Αν χρησιμοποιήσουμε την υλοποίηση των ουρών προτεραιότητας που βασίζεται σε σωρούς, την οποία εξετάσαμε στο Κεφάλαιο 2, κάθε λειτουργία ουράς προτεραιότητας μπορεί να εκτελεστεί σε χρόνο $O(\log n)$. Έτσι ο συνολικός χρόνος για την υλοποίηση είναι $O(m \log n)$.

4.5 Το πρόβλημα του Ελάχιστου Γεννητικού Δένδρου

Θα εφαρμόσουμε τώρα το επιχείρημα ανταλλαγής στα πλαίσια του δεύτερου θεμελιώδους προβλήματος για γραφήματα: το πρόβλημα του Ελάχιστου Γεννητικού Δένδρου (Minimum Spanning Tree Problem).

Το πρόβλημα

Υποθέστε ότι έχουμε ένα σύνολο τοποθεσιών $V = \{v_1, v_2, ..., v_n\}$ και θέλουμε να δομήσουμε ένα δίκτυο επικοινωνιών επάνω σε αυτές. Το δίκτυο θα πρέπει να είναι συνεκτικό — θα πρέπει να υπάρχει μια διαδρομή ανάμεσα σε κάθε ζευγάρι κόμβων — όμως επιθυμούμε να το κατασκευάσουμε με όσο το δυνατόν μικρότερο κόστος, ικανοποιώντας πάντα την προηγούμενη απαίτηση.

Για κάποια ζευγάρια (v_i, v_j) μπορούμε να κατασκευάσουμε έναν άμεσο σύνδεσμο μεταξύ των v_i και v_j με συγκεκριμένο κόστος $c(v_i, v_j) > 0$. Έτσι μπορούμε να αναπαραστήσουμε το σύνολο των δυνατών συνδέσμων που μπορούμε να δημιουργήσουμε χρησιμοποιώντας ένα γράφημα $G = (V, E)$, με θετικό *κόστος* c_e για κάθε ακμή $e =$

(v_i, v_j). Το πρόβλημα είναι να βρεθεί ένα υποσύνολο των ακμών $T \subseteq E$ έτσι ώστε το γράφημα (V, T) να είναι συνεκτικό και το συνολικό κόστος $\sum_{e \in T} c_e$ όσο το δυνατόν μικρότερο. (Θα υποθέσουμε ότι το αρχικό γράφημα G είναι συνεκτικό· διαφορετικά δεν υπάρχει λύση.)

Μια βασική παρατήρηση είναι η ακόλουθη.

(4.16) *Έστω T μια λύση ελάχιστου κόστους για το παραπάνω πρόβλημα σχεδιασμού δικτύου. Τότε το (V, T) είναι ένα δένδρο.*

Απόδειξη. Εξ ορισμού, το (V, T) πρέπει να είναι συνεκτικό· θα δείξουμε επίσης ότι δεν περιέχει κύκλους. Πράγματι, υποθέστε ότι περιείχε έναν κύκλο C, και έστω e οποιαδήποτε ακμή του C. Ισχυριζόμαστε ότι το $(V, T - \{e\})$ εξακολουθεί να είναι συνεκτικό, καθώς κάθε διαδρομή που προηγουμένως χρησιμοποιούσε την ακμή e μπορεί τώρα να ακολουθήσει "τη μεγάλη πορεία" μέσω του υπόλοιπου τμήματος του κύκλου C. Αυτό σημαίνει ότι το $(V, T) - \{e\}$ είναι επίσης έγκυρη λύση του προβλήματος και έχει μικρότερο κόστος — άτοπο. ■

Αν επιτρέψουμε σε κάποιες ακμές να έχουν μηδενικό κόστος (δηλαδή, υποθέτουμε μόνο ότι τα κόστη c_e είναι μη αρνητικά), τότε η λύση στο πρόβλημα του σχεδιασμού δικτύου με ελάχιστο κόστος μπορεί να έχει πρόσθετες ακμές — ακμές που έχουν κόστος 0 και που προαιρετικά θα μπορούσαν να διαγραφούν. Αλλά ακόμα και σε αυτή την περίπτωση υπάρχει πάντοτε μία λύση ελάχιστου κόστους που είναι δένδρο. Ξεκινώντας από οποιαδήποτε βέλτιστη λύση, θα μπορούσαμε να διαγράφουμε ακμές σε κύκλους μέχρι να έχουμε δένδρο· με μη αρνητικές ακμές, το κόστος δεν θα αυξανόταν στη διάρκεια αυτής της διαδικασίας.

Θα ονομάζουμε ένα υποσύνολο $T \subseteq E$ *γεννητικό δένδρο* (spanning tree) του G αν το (V, T) είναι δένδρο. Η πρόταση (4.16) λέει ότι ο στόχος του προβλήματος σχεδιασμού δικτύου μπορεί να διατυπωθεί ως εύρεση του φθηνότερου γεννητικού δένδρου για το γράφημα· γι′ αυτόν το λόγο ονομάζεται γενικά *πρόβλημα του Ελάχιστου Γεννητικού Δένδρου* (Minimum Spanning Tree Problem). Αν το G δεν είναι ένα πολύ απλό γράφημα, τότε θα έχει εκθετικά πολλά διαφορετικά γεννητικά δένδρα, των οποίων οι δομές μπορεί να είναι πολύ διαφορετικές η μία από την άλλη. Έτσι δεν είναι καθόλου προφανής ο αποδοτικός τρόπος εύρεσης του φθηνότερου δένδρου μεταξύ όλων αυτών των επιλογών.

Σχεδιασμός αλγορίθμων

Όπως και με τα προηγούμενα προβλήματα που συναντήσαμε, είναι εύκολο να βρούμε μια σειρά από φυσικούς άπληστους αλγορίθμους για το πρόβλημα. Όμως παραδόξως, και ευτυχώς, αυτή είναι η περίπτωση όπου *πολλοί* από τους πρώτους άπληστους αλγορίθμους που δοκιμάζει κανείς αποδεικνύονται σωστοί: καθένας από αυτούς επιλύει με βέλτιστο τρόπο το πρόβλημα. Θα εξετάσουμε τώρα μερικούς από αυτούς τους αλγορίθμους και μετά θα ανακαλύψουμε, μέσω ενός ωραίου ζεύγους επιχειρημάτων ανταλλαγής, κάποιες από τις αιτίες στις οποίες οφείλεται αυτή η πληθώρα των απλών, βέλτιστων αλγορίθμων.

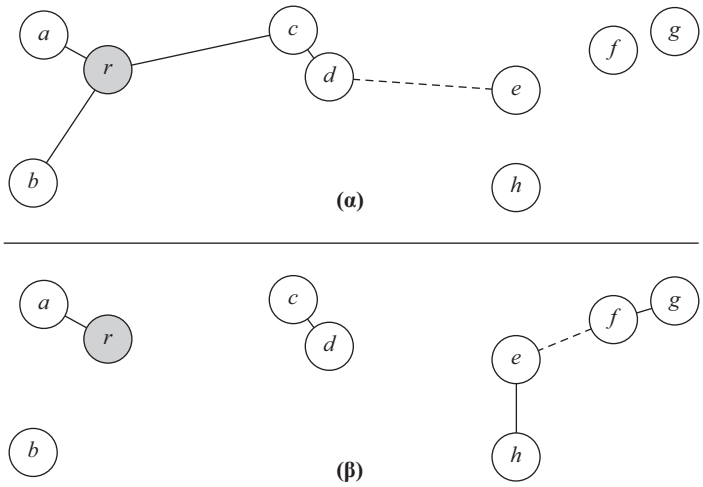
Ας δούμε τρεις άπληστους αλγορίθμους, όπου και οι τρεις βρίσκουν σωστά ένα ελάχιστο γεννητικό δένδρο.

- Ένας απλός αλγόριθμος ξεκινά χωρίς καθόλου ακμές και δομεί ένα γεννητικό δένδρο με διαδοχική εισαγωγή ακμών από το E κατά αύξουσα σειρά κόστους. Καθώς προχωρούμε μέσω των ακμών με αυτή τη σειρά, εισάγουμε κάθε ακμή e εφόσον δεν δημιουργεί κύκλο όταν προστίθεται στις ακμές που έχουν ήδη εισαχθεί. Αν όμως η εισαγωγή της e θα οδηγούσε σε κύκλο, τότε απλώς απορρίπτουμε την e και συνεχίζουμε. Αυτή η προσέγγιση ονομάζεται *αλγόριθμος του Kruskal*.
- Ένας άλλος απλός άπληστος αλγόριθμος μπορεί να σχεδιαστεί σε αναλογία με τον αλγόριθμο του Dijkstra για διαδρομές, αν και στην πράξη είναι ακόμα πιο απλός στον προσδιορισμό από τον αλγόριθμο του Dijkstra. Ξεκινάμε με έναν κόμβο ρίζας s και προσπαθούμε, με άπληστο τρόπο, να δημιουργήσουμε ένα δένδρο που ξεκινά από τον s και εκτείνεται προς τα έξω. Σε κάθε βήμα προσθέτουμε απλώς τον κόμβο που μπορεί να προσαρτηθεί με όσο το δυνατόν μικρότερο κόστος στο μερικό δένδρο που ήδη έχουμε.

  Πιο συγκεκριμένα, διατηρούμε ένα σύνολο $S \subseteq V$ για το οποίο έχει ήδη κατασκευαστεί ένα γεννητικό δένδρο. Αρχικά $S = \{s\}$. Σε κάθε επανάληψη προσαυξάνουμε το S κατά έναν κόμβο, προσθέτοντας τον κόμβο v που ελαχιστοποιεί το "κόστος προσάρτησης" $\min_{e=(u,v):u\in S} c_e$, και συμπεριλαμβάνοντας στο γεννητικό δένδρο την ακμή $e = (u,v)$ που επιτυγχάνει αυτή την ελάχιστη τιμή. Αυτή η προσέγγιση ονομάζεται *αλγόριθμος του Prim*.
- Τέλος, μπορούμε να σχεδιάσουμε έναν άπληστο αλγόριθμο εκτελώντας μια "αντίστροφη" εκδοχή του αλγορίθμου του Kruskal. Συγκεκριμένα, ξεκινάμε με ολόκληρο το γράφημα (V, E) και αρχίζουμε να διαγράφουμε ακμές κατά φθίνουσα σειρά κόστους. Καθώς φθάνουμε σε κάθε ακμή e (ξεκινώντας από την πιο ακριβή), τη διαγράφουμε εφόσον αυτό δεν θα επιφέρει τη μη συνεκτικότητα του γραφήματος που έχουμε. Προς αναζήτηση καλύτερου ονόματος, αυτή η προσέγγιση γενικά ονομάζεται *Αλγόριθμος Αντίστροφης Διαγραφής* (Reverse-Delete Algorithm) (απ′ όσα μπορούμε να γνωρίζουμε, δεν έχει πάρει το όνομα κάποιου συγκεκριμένου ατόμου).

Για παράδειγμα, η Εικόνα 4.9 παρουσιάζει τις τέσσερις πρώτες ακμές που προστέθηκαν από τους αλγορίθμους των Prim και Kruskal, αντίστοιχα, σε ένα γεωμετρικό στιγμιότυπο του προβλήματος του Ελάχιστου Γεννητικού Δένδρου όπου το κόστος κάθε ακμής είναι ανάλογο της γεωμετρικής απόστασης στο επίπεδο.

Το γεγονός ότι καθένας από αυτούς τους αλγορίθμους εγγυάται την παραγωγή μιας βέλτιστης λύσης υπονοεί μια "ανθεκτικότητα" στο πρόβλημα του Ελάχιστου Γεννητικού Δένδρου — υπάρχουν πολλοί τρόποι για να πάρουμε την απάντηση. Στη συνέχεια θα εξερευνήσουμε τις αιτίες για τις οποίες τόσο πολλοί διαφορετικοί αλγόριθμοι παράγουν γεννητικά δένδρα ελάχιστου κόστους.



Εικόνα 4.9 Παράδειγμα εκτέλεσης των αλγορίθμων Ελάχιστου Γεννητικού Δένδρου των (α) Prim και (β) Kruskal για την ίδια είσοδο. Οι 4 πρώτες ακμές που προστέθηκαν στο γεννητικό δένδρο εμφανίζονται με συνεχείς γραμμές· η επόμενη ακμή που θα προστεθεί εμφανίζεται με διακεκομμένη γραμμή.

Ανάλυση των αλγορίθμων

Όλοι αυτοί οι αλγόριθμοι πραγματοποιούν τη συνεχή εισαγωγή ή διαγραφή ακμών σε μία μερική λύση. Έτσι, για να τους αναλύσουμε, είναι χρήσιμο να έχουμε στη διάθεσή μας κάποια βασικά στοιχεία που να λένε πότε είναι "ασφαλές" να συμπεριλάβουμε μια ακμή στο ελάχιστο γεννητικό δένδρο, ή αντίστοιχα πότε είναι ασφαλές να απαλείψουμε μια ακμή με το σκεπτικό ότι δεν θα ήταν δυνατόν να ανήκει στο ελάχιστο γεννητικό δένδρο. Για τους σκοπούς της ανάλυσης αυτής, θα κάνουμε την απλουστευμένη υπόθεση ότι όλα τα κόστη των ακμών διαφέρουν μεταξύ τους (δηλαδή, δεν υπάρχουν δύο ίδια κόστη). Αυτή η υπόθεση κάνει ευκολότερη τη διατύπωση των προτάσεων που ακολουθούν· στη συνέχεια της ενότητας θα δείξουμε πώς μπορεί να απαλειφθεί εύκολα αυτή η υπόθεση.

Πότε είναι ασφαλές να συμπεριλαμβάνεται μια ακμή στο Ελάχιστο Γεννητικό Δένδρο; Το κρίσιμο γεγονός για την εισαγωγή μιας ακμής είναι η ακόλουθη πρόταση, την οποία θα αναφέρουμε ως *Ιδιότητα Αποκοπής* (Cut Property).

(4.17) *Υποθέστε ότι όλα τα κόστη των ακμών είναι διαφορετικά. Έστω S ένα τυχαίο υποσύνολο κόμβων που δεν είναι ούτε κενό ούτε ίσο με το V, και έστω ότι η ακμή e = (v, w) είναι η ακμή ελάχιστου κόστους με το ένα άκρο στο S και το άλλο στο V – S. Τότε κάθε ελάχιστο γεννητικό δένδρο θα περιέχει την ακμή e.*

Απόδειξη. Έστω T ένα γεννητικό δένδρο που δεν περιέχει την e· πρέπει να δείξουμε ότι το T δεν έχει το ελάχιστο δυνατό κόστος. Θα το επιτύχουμε αυτό χρησιμοποιώντας ένα επιχείρημα ανταλλαγής: θα προσδιορίσουμε μια ακμή e' στο T που είναι πιο ακριβή από την e, και η οποία έχει την ιδιότητα ότι η ανταλλαγή της e με την e' μας δίνει

"Το βιβλίο των Kleinberg και Tardos είναι το καλύτερο σύγγραμμα προπτυχιακού επιπέδου που έχω δει, και πιστεύω ότι θα αποτελέσει το πρότυπο για μια νέα γενιά βιβλίων με θέμα τους αλγορίθμους... (διαθέτει) μια φρέσκια προσέγγιση, ισχυρή έμφαση στο σχεδιασμό, και μεγάλο πλούτο πρωτότυπων ασκήσεων."

DIETER VAN MELKEBEEK
Πανεπιστήμιο Wisconsin – Madison

"Το βιβλίο *Σχεδιασμός αλγορίθμων* επιτυγχάνει τον τέλειο συνδυασμό διαισθητικής και αυστηρής προσέγγισης· έχει εκπληκτικές και εξαιρετικά ποικίλες εφαρμογές σε όλους τους τομείς της Επιστήμης των Υπολογιστών, ενώ τα προβλήματα που περιέχει αποτελούν όνειρο για κάθε διδάσκοντα..."

ANNA R. KARLIN
Πανεπιστήμιο Washington

"Μια προσπάθεια σύνδεσης των αλγοριθμικών ιδεών με προβλήματα του πραγματικού κόσμου η οποία αποδεικνύεται απίστευτα ισχυρή και εξαιρετικά εκτελεσμένη."

MICHAEL MITZENMACHER
Πανεπιστήμιο Harvard

Επισκεφθείτε μας στο Internet:
www.klidarithmos.gr

ISBN 978-960-461-207-9

Το βιβλίο *Σχεδιασμός αλγορίθμων* ακολουθεί μια νέα προσέγγιση στη διδασκαλία των αλγορίθμων, παρουσιάζοντας τις αλγοριθμικές ιδέες μέσα από εκείνα τα προβλήματα του πραγματικού κόσμου που κατέστησαν αναγκαία τη χρήση αλγορίθμων. Με ξεκάθαρο και άμεσο στυλ γραφής, οι Jon Kleinberg και Eva Tardos διδάσκουν τους σπουδαστές πώς να αναλύουν και να ορίζουν μόνοι τους τα προβλήματα, δείχνοντάς τους ταυτόχρονα τον τρόπο για να αναγνωρίζουν τις σχεδιαστικές αρχές που είναι κατάλληλες σε κάθε διαφορετική περίπτωση. Το βιβλίο βοηθά στην καλύτερη κατανόηση της διαδικασίας σχεδιασμού αλγορίθμων, αλλά και στην εκτίμηση του ρόλου που διαδραματίζουν οι αλγόριθμοι στο ευρύτερο πεδίο της Επιστήμης των Υπολογιστών.

Χαρακτηριστικά του βιβλίου:

- Δίνεται έμφαση στην ανάλυση των προβλημάτων και των τεχνικών σχεδιασμού.
- Ακολουθείται μια δομημένη παιδαγωγική μέθοδος που καθοδηγεί τους σπουδαστές, από τη διατύπωση του προβλήματος μέχρι την ολοκλήρωση των διαδικασιών σχεδιασμού και ανάλυσης του αλγορίθμου.
- Παρουσιάζεται «από πρώτο χέρι» η διαδικασία που ακολουθούν οι επιστήμονες των υπολογιστών για το σχεδιασμό και την εφαρμογή αλγορίθμων, μέσα από μια ευρεία γκάμα λυμένων προβλημάτων.
- Περιλαμβάνονται περισσότερα από 200 καλοδιατυπωμένα προβλήματα για «εργασία στο σπίτι» — με πολλά από αυτά να προέρχονται από γνωστές εταιρείες όπως οι Yahoo® και Oracle®.
- Γίνεται ευρεία κάλυψη των αλγορίθμων που ασχολούνται με NP-δύσκολα προβλήματα και των εφαρμογών της τυχαιοποίησης — δύο τομέων χρήσης των αλγορίθμων με ολοένα αυξανόμενη σημασία.

εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ
Διομήδους 4, Στούντιος Λαρισης, 10440 ΑΘΗΝΑ, Τηλ. 210-5237635