

**Εθνικό Μετσόβιο Πολυτεχνείο**



Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Αλγόριθμοι και Πολυπλοκότητα 2016-2017  
3η Σειρά Γραπτών Ασκήσεων

Όνομα: Παναγιώτης Κωστοπαναγιώτης  
A.M: 03115196

## Άσκηση 1. Εφαρμογές BFS και DFS

(α) Αρχικά, στο στάδιο της προεπεξεργασίας, παράγουμε το euler-tour του δέντρου ξεκινώντας από την κορυφή  $r$ . Εφαρμόζουμε δηλαδή μια DFS με αρχική κορυφή  $r$  και για κάθε κόμβο, έστω  $u$ , που επισκεπτόμαστε πρατάμε την πρώτη χρονική στιγμή που τον συναντάμε, καθώς και την τελευταία, έστω  $beg_u, end_u$  αντίστοιχα. Η διαδικασία αυτή απαιτεί γραμμικό χρόνο.

Τώρα, προκειμένου να απαντήσουμε σε ερωτήματα της μορφής  $(u,v)$  αρκεί να ελέγξουμε αν ο κόμβος  $v$  ανήκει στο υπόδεντρο που ορίζει ο κόμβος  $u$ . Προκειμένου να το ελέγξουμε αυτό χρησιμοποιούμε την πληροφορία της προεπεξεργασίας, ελέγχουμε δηλαδή αν  $beg_u \leq beg_v$  και  $end_v \leq end_u$ . Η πρώτη συνθήκη μας εξασφαλίζει ότι επισκεφτήκαμε τον κόμβο  $v$  μετά τον κόμβο  $u$  και η δεύτερη συνθήκη μας εξασφαλίζει ότι ο κόμβος  $v$  εμπεριέχεται στο υπόδεντρο του κόμβου  $u$  και άρα ο κόμβος  $v$  είναι πρόγονος του κόμβου  $u$ . Η απάντηση στα ερωτήματα γίνεται online και σε σταθερό χρόνο.

(β) Προκειμένου να αναπτύξουμε τον αλγόριθμό μας θα αποδείξουμε αρχικά το ακόλουθο.

Λήμμα: Ένας ισχυρά συνεκτικός γράφος θα περιέχει κύκλο περιττού μήκους αν και μόνο αν ο γράφος δεν είναι διμερής.

Απόδειξη:

- Ευθύ: Έστω πως ο γράφος περιέχει κύκλο περιττού μήκους, θα δείξουμε ότι δεν μπορεί να είναι διμερής. Έστω πως ο γράφος είναι διμερής και περιέχει κύκλο περιττού μήκους. Έστω  $A = \{ \alpha_1, \alpha_2, \dots, \alpha_k \}$ ,  $B = \{ \beta_1, \beta_2, \dots, \beta_\lambda \}$  τα δύο ανεξάρτητα σύνολα κόμβων στα οποία χωρίζεται ο γράφος. Επιλέγουμε αυθαίρετα έναν κόμβο  $\alpha_i$  ως κόμβο εκκίνησης του κύκλου περιττού μήκους. Παρατηρούμε πως για να διανύσουμε ένα μονοπάτι από έναν κόμβο στο σύνολο  $A$ , πάλι σε κάποιον κόμβο του συνόλου  $A$  χρειαζόμαστε απαραίτητα άρτιο πλήθος ακμών. Όμως, ο εν λόγω κύκλος μπορεί να θεωρηθεί μονοπάτι από τον κόμβο  $\alpha_i$  στον κόμβο  $\alpha_i$ , συνεπώς πρέπει να έχει άρτιο μήκος. Άτοπο.
- Αντίστροφο: Έστω τώρα πως ο γράφος δεν είναι διμερής, θα δείξουμε τότε πως ο γράφος περιέχει κύκλο περιττού μήκους.

Δεδομένου του παραπάνω, αρκεί πλέον απλά να ελέγξουμε αν ο γράφος μας είναι διμερής προκειμένου να αποφανθούμε για το κατά πόσο περιέχει κύκλο περιττού μήκους. Αυτό μπορεί να γίνει με μια DFS όπως φαίνεται παρακάτω.

Αλγόριθμος: Bipartite Check

Input:  $G = (V, E)$   
Output: YES/NO

$\forall u \in V$   
 $colour_u = NULL$

$dfs(u):$   
 $\forall v \in adj_u$   
 $if colour_u = colour_v$   
 $return NO$   
 $if colour_v = NULL$   
 $colour_v = not colour_u$   
 $dfs(v)$

$choose arbitrary node u$   
 $colour_u = 0$   
 $return dfs(u)$

Η πολυπλοκότητα του αλγορίθμου μας είναι όση και της DFS, δηλαδή  $O(n + m)$ , όπου  $n, m$  το πλήθος των κορυφών και των ακμών του γράφου αντίστοιχα.

## Άσκηση 2. Μια Συνάρτηση Κόστους σε Κατευθυνόμενα Γραφήματα

(α) Εφαρμόζουμε topological sorting στο DAG της εισόδου και λαμβάνουμε την τοπολογική διάταξη του γράφου. Πλέον λοιπόν, θα αναφερόμαστε στους κόμβους με την θέση τους στην τοπολογική διάταξη.

Θέλουμε να υπολογίσουμε για κάθε κόμβο  $u$  την τιμή  $\text{min\_cost}(u)$ , σαν βασική περίπτωση έχουμε ότι  $\text{min\_cost}(0) = c(0)$ , αφού ο πρώτος κόμβος στην τοπολογική σειρά έχει out degree 0. Έπειτα, για κάθε κόμβο  $i$  έχουμε  $\text{mincost}(i) = \min(c(i), \min_{u \in \text{adj}_i}(\text{mincost}(u)))$ . Στο τέλος, ο αλγόριθμός μας θα έχει υπολογίσει την τιμή  $\text{min\_cost}$  για κάθε κόμβο.

Ορθότητα: Για κάθε κόμβο  $u$  η μικρότερη τιμή είτε θα είναι  $c(u)$  είτε θα βρίσκεται σε κάποιον κόμβο πιο πίσω στην τοπολογική διάταξη, ο οποίος θα είναι προσπελάσιμος από τον κόμβο  $u$ . Συνεπώς, η ελάχιστη τιμή θα είναι το ελάχιστο από την  $c(u)$  και της ελάχιστης τιμής των κόμβων που συνδέονται με τον κόμβο  $u$ , αφού επεξεργαζόμαστε τους κόμβους σε τοπολογική διάταξη. Εξ' ού και η παραπάνω αναδρομική σχέση.

Πολυπλοκότητα: Στον αλγόριθμό μας τρέχουμε μια DFS και έπειτα μια γραμμική διαδικασία. Συνεπώς η συνολική πολυπλοκότητα είναι  $O(n + m)$ .

(β) Θα κάνουμε αναγωγή του προβλήματός μας στο αντίστοιχο πρόβλημα DAG που λύσαμε στο ερώτημα (α). Εφαρμόζουμε τον αλγόριθμο Kosaraju (i) και υπολογίζουμε τις ισχυρά συνεκτικές συνιστώσες του γράφου. Για κάθε μια από αυτές υπολογίζουμε την ελάχιστη τιμή  $c(u)$  για κάποιον κόμβο  $u$  της συνιστώσας και αντικαθιστούμε την κάθε συνιστώσα με έναν super-node με κόστος το ελάχιστο της συνιστώσας. Έπειτα, εφαρμόζουμε τον αλγόριθμο (α) αφού το γράφημα που προκύπτει είναι DAG.

Ορθότητα: Ο αλγόριθμός μας κατασκευάζει για κάθε SCC έναν νέο κόμβο με τιμή το ελάχιστο εντός της SCC. Έπειτα, το γράφημα που προκύπτει είναι DAG, αφού αν δεν ήταν ο κύκλος που θα υπήρχε θα ήταν ένα SCC. Συνεπώς το πρόβλημα ανάγεται σε εκείνο του ερωτήματος (α).

Πολυπλοκότητα: Ο αλγόριθμος Kosaraju τρέχει σε γραμμικό χρόνο και όλα τα υπόλοιπα βήματα του αλγορίθμου είναι επίσης γραμμικού χρόνου. Συνεπώς ο αλγόριθμός μας συνολικά έχει πολυπλοκότητα  $\Theta(n + m)$ .

- Πηγές: (i) Kosaraju algorithm: <http://lcm.cs.iisc.ernet.in/dsa/node171.html>

## Άσκηση 3: Ανάλυση Ασφάλειας

Αρχικά, χωρίζουμε τις τριάδες με βάση το  $t$  και επεξεργαζόμαστε ξεχωριστά τις τριάδες με ίδιο  $t$ , προφανώς σε αύξουσα σειρά ως προς το  $t$ . Έστω πως κάποια στιγμή, έστω  $t_i$ , έχουμε επεξεργαστεί όλες τις τριάδες με  $t < t_i$  και έχουμε χρωματίσει κόκκινους όσους υπολογιστές έχουν μολυνθεί κάποια στιγμή  $t < t_i$ .

Για τους υπολογιστές που επικοινωνούν την χρονική στιγμή  $t_i$  δημιουργούμε έναν γράφο βάζοντας μια μη κατευθυνόμενη ακμή μεταξύ των υπολογιστών που επικοινωνούν. Πλέον, έχουμε δημιουργήσει έναν γράφο με κάποιο πλήθος συνεκτικών συνιστώσων. Για μια συνεκτική συνιστώσα ισχύει το εξής: Οι υπολογιστές της συνιστώσας μολύνονται όλοι την χρονική στιγμή  $t_i$  αν και μόνο αν υπάρχει κάποιο κόκκινος κόμβος σε αυτή την συνιστώσα. Αυτό μπορεί να βρεθεί ένυκλα με έναν οποιονδήποτε αλγόριθμο διάσχισης (π.χ DFS). Εκτελώντας το παραπάνω για κάθε χρονική στιγμή έχουμε την τελική μας λύση.

Ορθότητα: Θα αιτιολογήσουμε την ορθότητα του αλγορίθμου μας επαγωγικά ως προς τον χρόνο  $t$ . Έστω πως για κάποια χρονική στιγμή  $t$  έχουμε υπολογίσει τους χρόνους που μολύνεται κάθε

υπολογιστής και έχουμε χρωματίσει κόκκινους τους υπολογιστές που έχουν μολυνθεί μέχρι και την χρονική στιγμή t. Κατασκευάζοντας τον γράφο για τους υπολογιστές που επικοινωνούν την επόμενη χρονική στιγμή από την t, βρίσκουμε αν αυτοί μολύνονται ελέγχοντας αν εκείνη την χρονική στιγμή επικοινωνούν με κάποιον υπολογιστή που έχει ήδη μολυνθεί. Συνεπώς, στο τέλος της επεξεργασίας γιατην χρονική στιγμή μετά την t θα έχουμε υπολογίσει ορθά τους χρόνους μόλυνσης των υπολογιστών που μολύνονται εκείνη την χρονική στιγμή.

**Πολυπλοκότητα:** Συνολικά, μέσα από τις DFS που τρέχουμε, διατρέχουμε όλους τους κόμβους και όλες τις ακμές από 1 φορά. Συνεπώς η συνολική πολυπλοκότητα του αλγορίθμου μας είναι  $\Theta(n+m)$ , όπου n, m το πλήθος των κόμβων και των ακμών του γράφου αντίστοιχα.

#### Άσκηση 4. Το Σύνολο των Συνδετικών Δέντρων

---

α) Αφού αφαιρέσουμε την μη κοινή ακμή e από το δέντρο  $T_1$  τότε το δέντρο αποσυνδέεται και δημιουργείται μια τομή. Το δέντρο  $T_2$  πρέπει με κάποιο τρόπο να συνδέει τα 2 σύνολα κόμβων της τομής με κάποια ακμή e' != e, αφού η e αποτελεί μη κοινή ακμή των 2 δέντρων. Συνεπώς το  $T_1 \cap e \cup e'$  αποτελεί συνδετικό δέντρο.

Μπορούμε να αναπτύξουμε αλγόριθμο γραμμικού χρόνου για την εύρεση μια τέτοιας ακμής. Όπως αναφέραμε, η αφαίρεση της ακμής από το δέντρο  $T_1$  μας οδηγεί στην δημιουργία 2 υπόδεντρων. Εφαρμόζοντας 1 DFS κατηγοριοποιούμε τους κόμβους ως προς το σε ποιό υπόδεντρο ανήκουν. Έπειτα, διατρέχουμε τις ακμές του δέντρου  $T_2$  μέχρι να βρούμε μια ακμή που να μας πηγαίνει από το ένα σύνολο κόμβων στο άλλο. Η συνολική πολυπλοκότητα της λύσης μας είναι  $O(V + E)$ .

β) Θα δείξουμε ότι, για κάθε ζεύγος δέντρων  $T_1, T_2$  με  $|T_1 \setminus T_2| = n$  υπάρχει μονοπάτι μεγέθους n στον γράφο H και άρα o H είναι συνδεδεμένος.

Εφαρμόζουμε επαγωγή στην απόστασή τους n:

**Επαγωγική βάση:** Αν  $n = 1$  τότε όπως δείξαμε στο ερώτημα (α) ο ισχυρισμός μας ισχύει.

**Επαγωγικό βήμα:** Έστω πως ο ισχυρισμός μας ισχύει για  $|T_1 \setminus T_2| = 1, 2, \dots, n - 1$ . Θα δείξουμε πως ισχύει και για  $|T_1 \setminus T_2| = n$ . Επιλέγουμε μια ακμή e του δέντρου  $T_1$  που δεν ανήκει στο  $T_2$  και την αφαιρούμε. Λόγω του ερωτήματος α) θα υπάρχει ακμή e' που ανήκει στο  $T_2$  και δεν ανήκει στο  $T_1$  που να γεφυρώνει την τομή που δημιουργήθηκε από την αφαίρεση της ακμής e. Σχηματίζουμε λοιπόν το δέντρο  $T_3 = T_1 \cap e \cup e'$  που αποτελεί γείτονα του δέντρου  $T_1$  και είναι  $|T_3 \setminus T_2| = n - 1$ . Συνεπώς η απόστασή του με το  $T_2$ , λόγω της επαγωγικής υπόθεσης, θα είναι  $n - 1$  και συνεπώς αφού είναι γείτονας του  $T_1$  στο H θα έχουμε  $d(T_1, T_2) = n$ .

Συνεπώς ο γράφος H είναι συνδεδεμένος.

#### Άσκηση 5. Μοναδικότητα Ελάχιστου Συνδετικού Δέντρου

---

(α) Το παραδειγμα φαίνεται παρακάτω:

$G = (V, E) = (\{1, 2, 3, 4\}, \{(1, 2, 2), (2, 3, 3), (1, 3, 4), (3, 4, 2)\})$ . Το μοναδικό ΕΣΔ είναι το  $(1, 2, 2)$ ,  $(2, 3, 3)$ ,  $(3, 4, 2)$  που αποτελείται από 2 ακμές βάρους 2.

(β) Το αντίστροφο του ισχυρισμού μας δεν ισχύει λόγω του παραδειγματος που φαίνεται παρακάτω:

$G = (V, E) = (\{1, 2, 3, 4\}, \{(1, 2, 2), (2, 3, 3), (4, 3, 2), (1, 4, 4)\})$ . Το μοναδικό ΕΣΔ είναι το  $(1, 2, 2)$ ,  $(2, 3, 3)$ ,  $(3, 4, 2)$  παρόλο που η τομή  $(1, 4)$ ,  $(2, 3)$  διασχίζεται από 2 ακμές βάρους 2.

(γ) Έστω  $c_2$  το κόστος του 2ου ΕΣΔ και  $c_1$  το κόστος του ΕΣΔ του γράφου G. Ο γράφος G

έχει μοναδικό ΕΣΔ αν και μόνο αν  $c_2 > c_1$ .

Απόδειξη: Θα αποδείξουμε ότι η συνθήκη που περιγράψαμε είναι ικανή και αναγκαία

- Ευθύ: Έστω πως ο γράφος έχει μοναδικό ΕΣΔ. Τότε, το δεύτερο ΕΣΔ δεν μπορεί να είναι ίσο σε αξία με το ΕΣΔ, αφού τότε το ΕΣΔ δεν θα ήταν μοναδικό.
- Αντίστροφο: Έστω πως  $c_2 > c_1$ . Τότε, το ΕΣΔ είναι μοναδικό αφού αν δεν ήταν το δεύτερο ΕΣΔ θα είχε ίση αξία με το αρχικό και άρα θα ήταν  $c_2 = c_1$ .

(δ) Προκειμένου να αποφανθούμε το αν το γράφημα μας έχει μοναδικό ΕΣΔ, αρκεί να υπολογίσουμε το δεύτερο ΕΣΔ και να ελέγξουμε το κόστος του σε σχέση με το αρχικό ΕΣΔ, όπως δείξαμε και στο ερώτημα (γ). Προκειμένου να το κάνουμε αυτό θα αναπτύξουμε έναν αλγόριθμο υπολογισμού του δεύτερου ΕΣΔ.

Αρχικά θα δείξουμε ότι το δεύτερο ΕΣΔ θα διαφέρει από το αρχικό ΕΣΔ κατά ακριβώς μια ακμή. Παρακάτω θα συμβολίζουμε με  $w(e)$  το βάρος μας αυθαίρετης ακμής  $e$ .

Απόδειξη: Έστω πως το δεύτερο ΕΣΔ διαφέρει κατά 2 ακμές από το αρχικό ΕΣΔ. Έστω δηλαδή ότι θα έπρεπε να αφαιρέσουμε τις ακμές  $e_1, e_2$  και να εισάγουμε τις ακμές  $e_1', e_2'$ . Έστω  $T$  το κόστος του ΕΣΔ. Με βάση την παραπάνω τροποποίηση λαμβάνουμε το νέο κόστος  $T_1 = T - w(e_1) - w(e_2) + w(e_1') + w(e_2')$ . Προφανώς από τις ιδιότητες του ΕΣΔ έχουμε  $w(e_1) \leq w(e_1')$  και  $w(e_2) \leq w(e_2')$ . Ωστόσο, μπορούμε να παράξουμε ένα δέντρο  $T_2$  με  $T_1 \geq T_2 \geq T$  αν δεν αφαιρέσουμε κάποια από τις ακμές  $e_1, e_2$ . Συνεπώς το  $T_1$  δεν μπορεί να είναι το δεύτερο ΕΣΔ.

Συνεπώς, στον αλγόριθμό μας αρκεί να υπολογίσουμε ένα αρχικό ΕΣΔ και έπειτα να εισάγουμε δοκιμάζουμε κάθε ακμή που δεν ανήκει στο ΕΣΔ. Προφανώς, η εισαγωγή μας νέας ακμής στο ΕΣΔ δημιουργεί κύκλο και άρα πρέπει να αφαιρέσουμε την μεγαλύτερη ακμή του κύκλου πριν αυτής που δοκιμάζουμε για εισαγωγή. Συνεπώς, ο αλγόριθμός μας θα μοιάζει κάπως έτσι:

Αλγόριθμος: Second-MST

Input:  $G(V, E)$

Output: cost of second mst

```

 $T = MST(G)$ 
 $cst = cost(T)$ 
 $rnd\_cost = \infty$ 
 $\forall e = (u, v) \in E - T$ 
 $A = max_{edge}(u, v)$ 
 $rnd\_cost = min(rnd\_cost, cst - A + w(e))$ 

```

Όλες οι λειτουργίες του αλγορίθμου μας είναι προφανείς ως προς την υλοποίησή τους εκτός από εκείνη της εύρεσης της ακμής μεγίστου βάρους στο μονοπάτι  $(u, v)$ .

Μια αρχική ιδέα για την υλοποίηση της παραπάνω λειτουργίας είναι κάθε φορά που δοκιμάζουμε μια νέα ακμή να τρέχουμε μια dfs από τον κόμβο  $u$  και να βρίσκουμε την ακμή μέγιστου βάρους στο μονοπάτι προς τον  $v$ . Στην χειρότερη περίπτωση βέβαια η διαδικασία αυτή θα πάρει χρόνο  $O(V)$  και συνεπώς καταλήγουμε με έναν αλγόριθμο πολυπλοκότητας  $O(ElogE + EV)$ .

Μπορούμε να βελτιώσουμε την πολυπλοκότητα του αλγορίθμου μας κάποιο precomputation. Για κάθε ζεύγος κόμβων θέλουμε να υπολογίσουμε μια τιμή  $max[u, v]$  που θα είναι η μέγιστη ακμή στο μονοπάτι  $(u, v)$  του ΕΣΔ. Προκειμένου να το κάνουμε αυτό για κάθε κόμβο  $u$  τρέχουμε μια DFS και διατηρούμε κάθε φορά την μέγιστη τιμή που έχουμε συναντήσει και αναβαθμίζουμε την τιμή του  $max[u, i]$  για κάθε  $i$  που συναντάμε κατά την εκτέλεση της DFS. Πλέον ο υπολογισμός του  $max_{edge}(u, v)$  γίνεται σε στάθερό χρόνο με  $O(V^2)$  προεπεξεργασία.

Καταλήγουμε λοιπόν σε έναν αλγόριθμο πολυπλοκότητας  $O(V^2 + E \log E)$ .

Μπορούμε να βελτιώσουμε κι άλλο την πολυπλοκότητα του αλγορίθμου μας. Έστω ότι οιζώνουμε το ΕΣΔ από έναν αυθαίρετο κόμβο, έστω τον κόμβο 1. Για κάθε κόμβο  $u$  θα θέλαμε να υπολογίσουμε την μέγιστη ακμή που εμφανίζεται στο μονοπάτι καθώς ανεβαίνουμε από τον κόμβο προς την  $q$ . Συγκεκριμένα, ας συμβολίσουμε ως  $M[u, k]$  ως την μέγιστη ακμή που εμφανίζεται στο μονοπάτι από τον κόμβο  $u$  μέχρι τον πρόγονό που απέχει  $2^k$  ακμές από τον  $u$  καθώς ανεβαίνουμε προς την  $q$ . Ο υπολογισμός της παραπάνω ποσότητας μπορεί να γίνει με έναν απλό αλγόριθμο δυναμικού προγραμματισμού σε χρόνο  $O(V \log V)$  συνολικά. Έπειτα, για κάθε ακμή  $(u, v)$  που δοκιμάζουμε, διασπάμε το μονοπάτι  $(u, v)$  σε 2 μονοπάτια, το  $(u, A)$  και το  $(v, A)$  όπου ο  $A$  είναι ο ελάχιστος κοινός πρόγονος των κορυφών  $u, v$ . Έπειτα, αρκεί να βρούμε την μέγιστη ακμή στα αντίστοιχα μονοπάτια και να πάρουμε το μέγιστο εκ των δύο. Ο υπολογισμός όλων των μεγεθών γίνεται με απλό αλγόριθμο σε χρόνο  $O(\log V)$ . Καταλήγουμε λοιπόν σε έναν αλγόριθμο πολυπλοκότητας  $O(E \log E)$ , όση δηλαδή της εύρεσης του αρχικού ΕΣΔ. Περισσότερα για τον αλγόριθμο υπολογισμού του LCA και των μεγίστων μπορεί να βρεθεί στο άρθρο που παρατίθεται στις πηγές.

- Πηγές: RMQ and LCA: <https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/>