National Technical University of Athens

School of Electrical
and Computer Engineering

Algorithms & Complexity

Semester 7 - Flow L

Analytic Assignment 3
Graphs

Dimitris Dimos - 031 17 165

Athens
January, 2021

Περιεχόμενα

1 Ραντεβού μετά το Lockdown

1.1 Διατύπωση Αλγορίθμου

Το πρόβλημα μοιάζει με το κλασσικό πρόβλημα συντομότερου μονοπατιού, το οποίο μπορεί να λυθεί με μία εκτέλεση BFS. Διαφοροποιείται ως προς το ότι σε κάθε στιγμή οι ακμές αλλάζουν κατεύθυνση. Για να λύσουμε, λοιπόν, το νέο πρόβλημα θα χρησιμοποιήσουμε μια παραλλαγή του BFS. Για αρχή, θεωρούμε αναπαράσταση γράφου με λίστες γειτνίασης και διατυπώνουμε ορισμένες προσθήκες.

- Κάθε κόμβος v της λίστας γειτόνων μιας κορυφής p, θα διατηρεί μια δυαδική μεταβλητή $dir(p, v)$, η οποία είναι 0 αν η ακμή σύνδεσης p και v είναι $p \longrightarrow v$, αλλιώς 1.
- Κάθε κορυφή που μπαίνει στην ουρά διατηρεί μαζί της μια δυαδική μεταβλητή $wait(u)$ η οποία θα είναι 1 αν θα χρειαστεί να περιμένουμε μία στιγμή σε αυτή, αλλιώς 0.
- Σε κάθε κορυφή v θα αντιστοιχίσουμε τον ελάχιστο χρόνο $time(v)$ μετάβασης από την s σε αυτή.

Ύστερα, προχωρούμε στη διατύπωση του αλγορίθμου:

Αλγόριθμος: Randez - vous

- Αρχικά, διασχίζουμε τον γράφο με χρήση BFS, ο οποίος τροποποιείται ως ακολούθως:
  1. Αναθέτουμε $time(s) \leftarrow 0$ και βάζουμε την s στην ουρά με $wait(s) = 0$
  2. Βγάζουμε την κεφαλή της ουράς, έστω v, και:
     - Αν $time(v) + wait(v) =$ άρτιος, τότε όλες οι ακμές θεωρούνται ανεστραμμένες μέχρι να τελειώσει η εξέταση της v και να αρχίσει η εξέταση της νέας κεφαλής.
     - Κάθε αμαρκάριστο γείτονα n της v με σύνδεση $v \longrightarrow n$ τον τοποθετούμε στην ουρά με $wait(n) = 0$, τον μαρκάρουμε και αναθέτουμε $time(n) = time(v) + wait(v) + 1$.
     - Αν $wait(v) = 0$ και υπάρχει αμαρκάριστος γείτονας n της v με σύνδεση $v \longleftarrow n$, τότε επανατοποθετούμε την v στην ουρά με $wait(v) = 1$.
  3. Επαναλαμβάνουμε το βήμα 2 μέχρι να γίνει η ανάθεση χρόνου στην $time(t)$.
  4. Αποθηκεύουμε το αποτέλεσμα: $t_{even} \leftarrow time(t)$
  5. Επαναλαμβάνουμε τον τροποποιημένο BFS από το βήμα 1, με τη διαφορά ότι αρχικά αναθέτουμε $time(s) \leftarrow 1$.
  6. Αποθηκεύουμε το νέο αποτέλεσμα: $t_{odd} \leftarrow time(t)$
- Ύστερα, για να έχουμε αποδεκτό αποτέλεσμα, πρέπει να πληρούται τουλάχιστον ένα από τα παρακάτω για κάποιο $x \in \mathbb{N}$:

$$x \;+\; t_{even} \;\leq\; T \quad or \quad x \;+\; t_{odd} \;\leq\; T$$

  - $max_{even} \leftarrow$ ο μέγιστος άρτιος φυσικός $x \leq T - t_{even}$
  - $max_{odd} \leftarrow$ ο μέγιστος περιττός φυσικός $x \leq T - t_{odd}$

  Τέλος, επιστρέφεται ο μέγιστος εκ των max_{even} και max_{odd}.

Αν στο τελευταίο βήμα του αλγορίθμου δεν πληρούται καμία εκ των δύο συνθηκών, τότε αυτό σημαίνει πως δεν μπορούμε να φτάσουμε έγκαιρα στην t και ο αλγόριθμος επιστρέφει -1. Επιπλέον, έχουμε θεωρήσει ότι ο γράφος είναι συνεκτικός, αφού αναπαριστά δίκτυο συγκοινωνιών.

1.2 Εξήγηση Αλγορίθμου - Ορθότητα

Ο αλγόριθμος υπολογίζει για κάθε κορυφή τον ελάχιστο χρόνο μετάβασης από την s σε αυτή. Σε κάθε βήμα (που αντιπροσωπεύει μια χρονική στιγμή) μπορούμε να πάμε από μια κορυφή σε κάποιο γείτονα αν οι κατευθύνσεις των ακμών το επιτρέπουν ή, αν όχι, να περιμένουμε μια στιγμή για να αντιστραφούν. Έτσι, ευρισκόμενοι σε κάποια κορυφή αναθέτουμε χρόνους στους γείτονες που έχουμε πρόσβαση: χρόνος(γείτονα) $=$ χρόνος(πρόσβασης στον πατέρα) $+$ χρόνος(αναμονής μέχρι αλλαγή κατευθύνσεων) $+ 1$. Σε όσους δεν έχουμε πρόσβαση, περιμένουμε μία στιγμή κι, ύστερα, μεταβαίνουμε.

Τα παραπάνω συνοδεύουν την προσθήκη γείτονα στην ουρά. Αν μπορούμε να πάμε σε αυτόν, τότε τον τοποθετούμε στην ουρά αφού ανανεώσουμε τον χρόνο του. Αν δεν μπορούμε, τότε ξαναβάζουμε την εξεταζόμενη κορυφή στην ουρά με χρόνο αναμονής $wait = 1$, ο οποίος θα προστεθεί στον συνολικό χρόνο πρόσβασης στους υπολοιπόμενους (μη προσβάσιμους αμέσως πριν) γείτονες.

Έτσι, εξασφαλίζουμε ότι όταν εξαντλήσουμε τις κορυφές με χρόνο y, τότε όλες οι υπόλοιπες που περιμένουν να εξεταστούν θα έχουν χρόνο μεγαλύτερο από y. **Αυτό σημαίνει ότι αν βρούμε μια κορυφή κατά τη διάσχιση, τότε ο χρόνος που της αναθέτουμε είναι και ο μικρότερος δυνατός**.

Η πρώτη εκτέλεση του αλγορίθμου μας επιστρέφει τον ελάχιστο χρόνο που χρειαζόμαστε για να μεταβούμε στην t με δεδομένο ότι ξεκινήσαμε σε μια άρτια στιγμή. Δηλαδή, έστω ότι ξεκινάμε την στιγμή 0 και κάνουμε x στιγμές για να φτάσουμε στην t. Τότε αν ξεκινήσουμε τη στιγμή $2y$, επειδή ο γράφος εκκίνησης θα είναι ίδιος (αφού η αρχική τιμή του $time(s) = 2y$ είναι άρτια και, συνεπώς, το μοτίβο εκτέλεσης του τροποποιημένου BFS είναι ίδιο) ο χρόνος που θα χρειαστούμε είναι $2y + x$.

Αυτό, όμως αλλάζει αν ξεκινήσουμε μια περιττή στιγμή, αφού ο γράφος εκκίνησης είναι διαφορετικός. Γι' αυτό ξανατρέχουμε τον τροποποιημένο BFS και με $time(s) = 1$. Έτσι, προκύπτουν δύο αποτελέσματα, ένα για εκκίνηση σε άρτια κι ένα σε περιττή στιγμή.

Εν τέλει, κάθε ένα απο τα δύο αποτελέσματα (μπορεί και μόνο ένα, ή και κανένα αν δεν μπορούμε να φτάσουμε έγκαιρα) δίνει έναν μέγιστο φυσικό (τελευτάιο βήμα αλγορίθμου). Η αργότερη στιγμή που μπορούμε να ξεκινήσουμε θα είναι ο μεγαλύτερος εκ των δύο μεγίστων φυσικών.

1.3 Υπολογιστική Πολυπλοκότητα

Η πολυπλοκότητα του αλγορίθμου οφείλεται στους BFS που εκτελούνται. Εφόσον και οι δύο ακολουθούν ακριβώς τα ίδια βήματα, ασυμπτωτικά μας ενδιαφέρει η πολυπλοκότητα τους ενός. Ο τροποποιημένος BFS, λοιπόν, εκτελεί τα βήματα του κλασσικού BFS, αλλά λόγω της επανατοποθέτησης κορυφών στην ουρά, οι κορυφές που εν τέλει τοποθετούνται στην ουρά είναι στη χειρότερη: όσοι ανήκουν στον γράφο εισόδου $+$ $\#$(ακμών που δεν επιτρέπουν άμεση μετάβαση). Ακόμη, οι ακμές είναι σαν να αυξήθηκαν κι αυτές κατά $\#$(ακμών που δεν επιτρέπουν άμεση μετάβαση), αφού τόσες φορές θα πρέπει να "ξαναεπιχειρήσουμε" να μεταβουμε στις κορυφές-γείτονες. Τελικώς, έχουμε:

$$\boxed{O\big((n+m)+(m+m)\big) = O(|V|+|E|)}$$

Η πολυπλοκότητα είναι γραμμική ως προς το μήκος της εισόδου.

2 Προγραμματίζοντας την Αντίδραση

2.1 Διασαφήνιση Ορισμένων Στοιχείων

Έστω f η ζητούμενη κορυφή, η οποία απέχει $dist(v_i)$ από την αρχική κορυφή v_i. Ισχυριζόμαστε ότι οι αποστάσεις $dist(v_i), \forall i \in \{1, \cdots, k\}$ θα πρέπει ανά δύο να διαφέρουν το πολύ κατά ένα. Δηλαδή, θα πρέπει κάποιες αποστάσεις να ισούνται με y και οι υπόλοιπες με $y+1$.

Απόδειξη

Έστω ότι για ένα ζεύγος αρχικών κορυφών κορυφών v_i και v_j είναι $dist(v_i) = x$ και $dist(v_j) = y$, με $x > y+1$. Τότε ο ελάχιστος χρόνος που χρειάζεται για να φτάσουν τα αντίστοιχα σωματίδια στην κορυφή f είναι $max\{x, y\} = x$ (y χρόνος για να φτάσει πρώτα το j, όπου ύστερα "περιμένει", και $x-y$ χρόνος ακόμα για να φτάσει το i). Τότε, όμως υπάρχει μια κορυφή f' που βρίσκεται σε απόσταση $dist'(v_i) = \left\lfloor \frac{x+y}{2} \right\rfloor$ από την v_i και $dist'(v_j) = \left\lceil \frac{x+y}{2} \right\rceil$ από την v_j (Γενικά: $\left\lceil \frac{x+y}{2} \right\rceil = \left\lfloor \frac{x+y}{2} \right\rfloor$ ή $\left\lceil \frac{x+y}{2} \right\rceil = \left\lfloor \frac{x+y}{2} \right\rfloor + 1$). Ο χρόνος άφηξης των σωματιδίων στην f' είναι $max\{\left\lfloor \frac{x+y}{2} \right\rfloor, \left\lceil \frac{x+y}{2} \right\rceil\} = \left\lceil \frac{x+y}{2} \right\rceil < x$. Άρα η κορυφή f' φτάνεται γρηγορότερα και από τα δύο σωματίδια σε σχέση με το f, άρα είναι προτιμότερη $\Longrightarrow$ Άτοπο.

Άρα, ο αλγόριθμος που θα σχεδιάσουμε πρέπει να εντοπίζει την κορυφή f που απέχει από κάποιες αρχικές κορυφές σωματιδίων απόσταση x και από τις υπόλοιπες $x+1$.

2.2 Διατύπωση Αλγορίθμου

Καταρχάς, θεωρούμε ότι ο γράφος αναπαρίσταται με λίστες γειτνίασης. Ύστερα, το ζητούμενό μας είναι να βρούμε μια κορυφή που απέχει από όλες τις αρχικές κορυφές v_i απόσταση x ή $x+1$. Για να γίνει αυτό, θα χρησιμοποιήσουμε μια τροποποιημένη εκδοχή του BFS αλγορίθμου. Για την εκτέλεση του BFS θα πρέπει:

- να έχουμε έναν δυαδικό πίνακα $visited[k][n]$ στον οποίο αποθηκεύουμε 1 στη θέση $visited[i][j]$ αν από την αρχική κορυφή v_i έχουμε φτάσει στην κορυφή j και 0 αν όχι.
- κάθε κορυφή του γράφου να διαθέτει έναν μετρητή, ο οποίος μετρά το πλήθος των σωματιδίων που έχουν φτάσει σε αυτή μέχρι στιγμής.
- να έχουμε έναν πίνακα $parent[i][j]$ ο οποίος αποθηκεύει τον πατέρα της κορυφής j, όταν έχουμε φτάσει σε αυτή μέσω μονοπατιού που ξεκίνησε από την κορυφή v_i.

Ο αλγόριθμος παρατίθεται εντός χαρακτηριστικού πλαισίου.

2.3 Εξήγηση Αλγορίθμου - Ορθότητα

Εφόσον αποδείξαμε ότι το ζητούμενο σημείο θα έχει απόσταση x από κάποιες αρχικές κορυφές και $x+1$ από κάποιες άλλες, θα πρέπει (για να το κάνουμε αποδοτικά) να αναζητούμε το σημείο αυτό "επεκτεινόμενοι" κάθε φορά κατά ίση απόσταση από τις αρχικές κορυφές. Γι'αυτό κάνουμε παράλληλα BFS από όλες τις κορυφές v_i. Έτσί, κάθε φορά που κάποιο από τα επεκτεινόμενα μονοπάτια φτάσει σε κάποια κορυφή "τη μαρκάρει με τον αριθμό i". Σε αυτή την κορυφή δεν θα ξαναφτάσει ποτέ το ίδιο μονοπάτι, γιατί "πηγαίνουμε" μόνο σε αμαρκάριστες κορυφές. Έτσι, από κάθε μονοπάτι μπορούμε να φτάσουμε σε κάθε κορυφή μία φορά. Αυξάνοντας τον μετρητή κάθε κορυφής που βάζουμε στην ουρά (ο μετρητής αυξάνεται μόνο μία φορά από κάθε ένα εκ των k μονοπατιών) εξασφαλίζουμε ότι όταν ο μετρητής αυτός γίνει k τότε φτάσανε στην κορυφή του όλα τα μονοπάτια. Ακόμη, επειδή από κάθε μέτωπο αναζήτησης (μέτωπο αναζήτησης i: το μέτωπο αναζήτησης ξεκινώντας από την κορυφή v_i) πηγαίνουμε με τη σειρά ένα βημα κάθε φορά, έχουμε εξασφαλίσει ότι σε κάθε στιγμή η απόσταση κάθε μετώπου από την αρχική κορυφή της είναι πάντοτε ίση ή μεγαλύτερη κατά ένα με όλες τις άλλες αποστάσεις. Άρα, όταν φτάσουν όλα τα μέτωπα στην f (όπου ο μετρητής της θα γίνει πρώτος από όλους ίσος με k) θα έχουμε αποστάσεις x ή $x+1$ από όλες τις κορυφές v_i, άρα η f είναι η ζητούμενη κορυφή. Έτσι, έχουμε τον μικρότερο δυνατό χρόνο συνάντησης (τον x, αλλα δεν χρειάζεται να επιστρέφεται) και το μονοπάτι που πρέπει κάθε σωματίδιο να ακολουθήσει είναι η αντίστοιχη λίστα από τις επιστρεφόμενες.

Αλγόριθμος: Chick - a - Boom

1. Έχουμε αρχικοποιημένους τους μετρητές των κορυφών του γράφου στο 0.
2. Αυξάνουμε τον μετρητή της κορυφής v_i και τοποθετούμε στην ουρά το ζεύγος (v_i, i). Επαναλαμβάνουμε για κάθε άλλη κορυφή όπου βρίσκεται σωματίδιο.
3. Βγάζουμε την κεφαλή της ουράς, έστω (p, i). Η p είναι η κορυφή p στην οποία έχουμε φτάσει ξεκινωντας από την αρχική κορυφή v_i.
4. Αν ο μετρητής έχει τιμή ίση με k, τότε αποθήκευσε την p ως f και προχώρα στο βήμα 7.
5. Για κάθε αμαρκάριστο γείτονα της p, έστω j:
   - θέτουμε $visited[i][j] = 1$,
   - αυξάνουμε τον μετρητή του,
   - θέτουμε $parent[i][j] = p$
   - τοποθετούμε στην ουρά το ζεύγος (j, i)
6. Επαναλαμβάνουμε από το βήμα 3.
7. Αποθήκευσε σε k λίστες τα μονοπάτια μέσω των οποίων φτάσαμε στην κορυφή f, μέσω backtracking, χρησιμοποιώντας τον πίνακα $parent$. Για παράδειγμα, το μονοπάτι μέσω του οποίου φτάσαμε στην f ξεκινώντας από την v_i θα είναι: $\Big[v_i, \quad \cdots, \quad parent\big[i\big]\big[parent[i][f]\big], \quad parent[i][f], \quad f\Big]$
8. Επίστρεψε ως έξοδο την κορυφή f και τις k λίστες.

2.4 Υπολογιστική Πολυπλοκότητα

Ο ασυμπτωτικός χρόνος που χρειάζεται ο αλγόριθμός μας εξαρτάται από την εκτέλεση του τροποποιημένου BFS. Ο BFS στην χειρότερη περίπτωση θα πρέπει να διασχίσει για κάθε κορυφή v_i ολόκληρο τον γράφο. Άρα, έχουμε πολυπλοκότητα $O\Big(k(n+m)\Big)$. Ωστόσο, θεωρούμε πως το πλήθος n των κορυφών είναι σημαντικά μεγαλύτερο από το πλήθος των κορυφών k. Άρα, τον κυρίαρχο ρόλο στον ασυμπτωτικό χρόνο του αλγορίθμου παίζουν οι κορυφές και οι ακμές. Συνεπώς, η πολυπλοκότητα είναι γραμμική ως προς το μήκος της εισόδου, δηλαδή:

$$\boxed{O(n+m) = O(|V| + |E|)}$$

3 Ένας παράξενος περίπατος

3.1 Μερικά Στοιχεία για την Λύση

Προτού προχωρήσουμε στη διατύπωση αλγορίθμου, θα πρέπει να διατυπώσουμε μερικά μαθηματικά στοιχεία που θα φανούν χρήσιμα.

1. Ο GCD n αριθμών είναι μικρότερος ή ίσος με τον μικρότερο εξ αυτών.
2. Αν έχουμε τον GCD των αριθμών a και b, έστω $G(a, b) = \delta$, τότε ο GCD των α και β με έναν τρίτο αριθμό c είναι: $GCD(a, b, c) = GCD(\delta, c)$.
3. Αναδρομικά, για n αριθμούς ο GCD τους είναι ο $GCD(n_1, n_2, ..., n) = GCD(\delta_{n-1}, n_n)$, οπου δ_{n-1} : ο GCD των $n - 1$ εξ αυτών.

Τα παραπάνω είναι γνωστά θεωρήματα των μαθηματικών και δεν χρειάζονται απόδειξη στην παρούσα εργασία.

3.2 Μετασχηματισμός Προβλήματος

Η πρώτη μας παρατήρηση είναι ότι ο γράφος δεν είναι απαραίτητα DAG, επομένως αποτελείται από ισχυρά συνεκτικές συνιστώσες (όπως κάθε κατευθυνόμενος γράφος), εκ των οποίων ενδέχεται κάποιες να περιέχουν περισσότερες από 1 κορυφές.

Αυτό σημαίνει ότι αν ο βέλτιστος περίπατος (του οποίου το κόστος ψάχνουμε) διέρχεται από μία τουλάχιστον κορυφή μιας ΙΣΣ, τότε υπάρχει περίπατος που διέρχεται από ολόκληρη την ΙΣΣ και έχει ίδιο κόστος με το βέλτιστο μονοπάτι (είναι δηλαδή κι αυτός βέλτιστος).

<u>Απόδειξη</u>

Εστω βέλτιστος περίπατος $best = (v_1, v_2, \cdots, v_s, \cdots, v_k)$ και $v_s \in \{$ΙΣΣ που συνίσταται από τουλάχιστον 2 κορυφές$\}$. Τότε ο περίπατος $(v_1, v_2, \cdots, v_s,$ (διάσχιση όλων των κορυφών τις ΙΣΣ)$, v_k)$ αποτελεί υπερσύνολο του $best$ και λόγω του μαθηματικού δεδομένου (1), θα έχει κόστος μικρότερο ή ίσο του $best$. Είναι δηλαδή κι αυτός βέλτιστος.

Επομένως, στην προσπάθειά μας να βρούμε το βέλτιστο κόστος, μπορούμε να θεωρήσουμε ότι οποιοσδήποτε υποψήφιος περίπατος περνά από κορυφή ΙΣΣ, θα τη διασχίζει ολόκληρη, χωρίς να χάσουμε ποτέ τη βέλτιστη λύση.

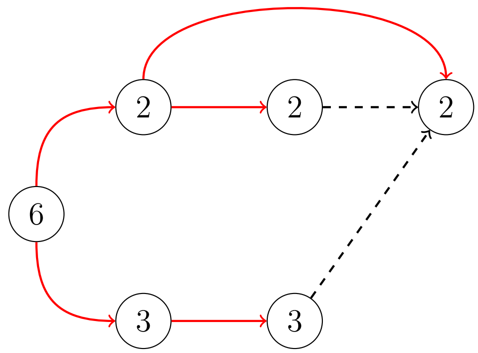
Για να απλοποιήσουμε, λοιπόν, τη δομή του γράφου, μπορούμε να θεωρήσουμε τις ΙΣΣ ως ενιαίες κορυφές με κόστος τον GCD των κορυφών που την συναποτελούν. Οι κορυφές αυτές έχουν ακμές εισόδου, όλες τις ακμές που προσέπιπταν πάνω σε οποιαδήποτε από τις κορυφές μέλη της ΙΣΣ και ακμές εξόδου όλες τις ακμές που εξέρχονταν από οποιαδήποτε από τις κορυφές μέλη της ΙΣΣ.

Η μελέτη μας από εδώ και πέρα αφορά τον μετασχηματισμένο γράφο, του οποίου όλες οι ΙΣΣ αποτελούνται από μία κορυφή $\implies$ δεν υπάρχουν κύκλοι $\implies$ ο νέος ισοδύναμος ως προς το ζητούμενο γράφος είναι DAG.

Στη συνέχεια, προσπαθούμε να βρούμε μια λύση που να περιλαμβάνει μία μόνο διάσχιση του γράφου με κάποιο αλγόριθμο διάσχισης που έχει γραμμικό χρόνο εκτέλεσης ως προς το μήκος της εισόδου (όπως πχ ο BFS). Όμως, βλέπουμε ότι αλγόριθμοι σαν τον BFS προκειμένου να πετύχουν την γραμμική πολυπλοκότητα, μαρκάρουν τους εξερευνημένους κόμβους, ώστε να μην τους ξαναεπισκεφτούν. Αυτό ενέχει τον κίνδυνο να φτάσουμε σε έναν κόμβο μέσω ενός μονοπατιού, "απαγορεύοντας" να φτάσουμε ξανά σε αυτόν μέσω άλλου μονοπατιού που ενδέχεται να έχει εν τέλει χαμηλότερο κόστος.

Δεν κάνουμε λόγο για την κορυφή εκκίνησης του αλγορίθμου διάσχισης, καθώς ο παραπάνω κίνδυνος ισχύει πάντοντε.

Ένα απλό παράδειγμα παρατίθεται στη συνέχεια, στο οποίο έχουμε υποθέσει ότι ο BFS ξεκινάει από την κορυφή που βρίσκεται ανώτερα στην τοπολογική ταξινόμηση του γράφου.



Εν ολίγοις, για να επιλέξουμε *ντετερμινιστικά, μέσω αλγορίθμου διάσχισης,* τον καλύτερο τρόπο για να πάμε σε μια κορυφή θα πρέπει να έχουμε πληροφορία για όλα το μονοπάτια που καταλήγουν σε αυτή.

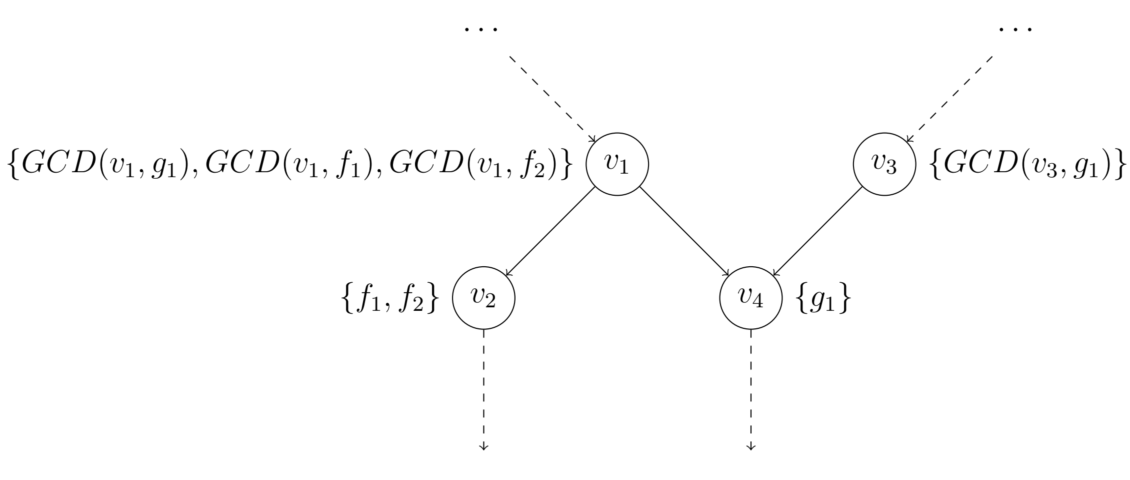
Θα ακολουθήσουμε, λοιπόν, διαφορετική στρατηγική.

3.3 Διατύπωση Αλγορίθμου

Ο γράφος στον οποίο δουλεύουμε έχει τη μορφή DAG. Αυτό σημαίνει ότι μπορούμε να ταξινομήσουμε τοπολογικά τις κορυφές του σε χρόνο $O(n+m)$.

Υπάρχει μονοπάτι που είναι βέλτιστο (έχει δηλαδή βέλτιστο κόστος), και ξεκινάει από κορυφή που είναι $source$ (καμία ακμή δεν προσπίπτει σε αυτή), η οποία σίγουρα υπάρχει αφού έχουμε DAG. Κι αυτό, γιατί αν έχουμε ένα μονοπάτι με βέλτιστο κόστος που δεν περνάει από $source$, τότε "προεκτείνοντάς" το προς τις πίσω κορυφές της τοπολογικής ταξινόμησης (αν δεν υπάρχουν τότε η κορυφή εκκίνησης του μονοπατιού είναι $source$), φτάνουμε σε $source$ κι εφόσον οι κορυφές του νέου μονοπατιού είναι υπερσύνολο του βέλτιστου, λόγω της μαθηματικής πρότασης (1), το κόστος θα είναι ίσο ή μικρότερο, άρα βέλτιστο.

Επιπλέον, παρατηρούμε ότι αν γνωρίζουμε (με κάποιο τρόπο) τα κόστη όλων των μονοπατιών που ξεκινούν από μια κορυφή, τότε μπορούμε εύκολα να υπολογίσουμε τα κόστη των μονοπατιών που ξεκινούν μια κορυφή πίσω. Σχηματικά, εννοούμε το εξής:



Έστω ότι τα κόστη των μονοπατιών που ξεκινούν από την κορυφή v_4 είναι τα στοιχεία του συνόλου $\{g_1\}$. Τότε μπορούμε για κάθε γονέα της v_4 να βρούμε τα κόστη των μονοπατιών που ξεκινάνε από αυτούς, όπως στο σχήμα.

Εν ολίγοις μπορούμε να βρούμε τα κόστη των μονοπατιών που ξεκινούν από μια κορυφή αν γνωρίζουμε τα κόστη των μονοπατιών που ξεκινούν από τα παιδιά τους.

Παράγουμε την αναδρομική σχέση:

$$cost\_set(u) = \bigcup_i \left\{ \bigcup_j \{gcd(g_{ij}, u)\} \right\}$$

όπου g_{ij} είναι ο GCD του $j-$οστού μονοπατιού που ξεκινάει από το $i-$οστό παιδί της u

Δηλαδή, παίρνουμε όλα τα κόστη μονοπατιών που ξεκινούν από κάθε παιδί της u, τα βάζουμε στην φόρμουλα $gcd(\_, u)$ κι ενώνουμε σε ένα σύνολο τα αποτελέσματα. Συνεχίζοντας την ίδια διαδικασία προς τα πίσω, υπολογίζουμε όλα τα πιθανά κόστη μονοπατιών που ξεκινούν από τα *sources* του DAG, δηλαδή όλα τα υποψήφια βέλτιστα κόστη.

Διατυπώνουμε, λοιπόν, αλγόριθμο που χρησιμοποιεί δυναμικό προγραμματισμό.

Αλγόριθμος: minimum GCD cost

1. Μετασχηματίζουμε τον γράφο σε DAG συγχωνεύοντας όλες τις ΙΣΣ σε μία νέα ενιαία κορυφή (διαφορετική για κάθε ΙΣΣ) με κόστος τον GCD των κοστών που συναποτελούσαν την ΙΣΣ.

   Οι ακμές που προσπίπτουν σε μια νέα ενιαία κορυφή είναι αυτές που προσέπιπταν σε κάποια κορυφή της ΙΣΣ και οι ακμές που εξέρχονται από μια ενιαία κορυφή είναι αυτές που έξρχονταν από κάποια κορυφή της ΙΣΣ.

2. Ταξινομούμε τοπολογικά τον DAG.

3. Εξετάζουμε τις κορυφές της τοπολογικής ταξινόμησης με ανάποδη σειρά.

   - Αν η εξεταζόμενη κορυφή δεν έχει παιδιά, αρχικοποιούμε το σύνολο κοστών της σε $\{c_i\}$, όπου c_i το κόστος της κορυφής.
   - Αλλιώς, έχοντας υπολογίσει πιο πριν τα σύνολα των παιδιών της (γιατί σίγουρα είναι χαμηλότερα στην ταξινόμηση και άρα τα εξετάσαμε νωρίτερα), με χρήση της άνω αναδρομικής σχέσης υπολογίζουμε το σύνολο κοστών της εξεταζόμενης κορυφής.

4. Επαναλαμβάνουμε το βήμα (3) μέχρι να εξετάσουμε όλες τις κορυφές της τοπολογικής ταξινόμησης.

5. Στο τέλος, έχουμε υπολογίσει το σύνολο κοστών για κάθε κορυφή.

6. Διατρέχουμε τα κόστη των συνόλων κάθε κορυφής κι επιστρέφουμε το μικρότερο.

3.4 Ορθότητα

Ο αλγόριθμος χρησιμοποιεί δυναμικό προγραμματισμό για να βρεί τα κόστη των μονοπατιών που ξεκινούν από κάθε κορυφή. Πηγαίνοντας από τις κατώτερες κορυφές της τοπολογικής ταξινόμησης προς τις ανώτερες, γνωρίζουμε ότι πρώτα θα εξερευνώνται κορυφές που είναι παιδιά των ανώτερων. Άρα, όταν επισκευτούμε μια κορυφή δεν γίνεται να μην έχουμε εξερευνήσει πρώτα τα παιδά της. Κι αφού έχουμε εξερευνήσει τα παιδιά της, έχουμε τα σύνολά κοστών τους και άρα, μέσω της αναδρομικής σχέσης, έχουμε και το σύνολο κοστών της εξεταζόμενης κορυφής. Συνεχίζοντας μέχρι το τέλος, έχουμε βρει τα σύνολα με τα κόστη των μονοπατιών που ξεκινούν από κάθε κορυφή. Το βέλτιστο μονοπάτι ξεκινάει από ένα $sourse$, αλλά δεν μπορούμε να ξέρουμε που είναι τα $sources$ στην ταξινόμηση. Άρα, εξετάζουμε όλα τα σύνολα και από τα κόστη που συναντάμε κρατάμε το μικρότερο.

3.5 Υπολογιστική Πολυπλοκότητα

Προκειμένου να μετατρέψουμε τον γράφο μας σε DAG πρεπει να εντοπίσουμε τις ΙΣΣ, σε χρόνο $O(n+m)$ και να τις συγχωνεύσουμε. Η συγνώνευση θα κοστίσει επίσης $O(n+m)$, αφού για κάθε κορυφή (την οποία εξετάζουμε μία και μόνο φορά) μιας ΙΣΣ εξετάζουμε τις ακμές που εισέρχονται κι εξέρχονται. Κάθε ακμή δεν μπορεί να εξεταστεί πάνω από δύο φορές.

Για το κομμάτι του δυναμικού προγραμματισμού, χρεωνόμαστε μία διάσχιση γράφου. Θεωρώντας ότι τα σύνολα με πιθανές λύσεις, καθώς "ανεβαίνουμε" στην ιεραρχία της τοπολογικής ταξινόμησης, όχι μόνο δεν αυξάνουν το πλήθος των στοιχείων τους, αλλά σε κάποια σημεία το μειώνουν (επειδή κάποιοι νέοι gcd που προκύπτουν είναι ίδιοι), οι πράξεις που γίνονται (ασυμπτωτικά) σε κάθε κορυφή δεν αυξάνονται. Άρα, χρεωνόμαστε $O(n+m)$ για τη διάσχιση του γράφου και $O(n+m)$ για την αναζήτηση του αποτελέσματος.

Άρα, η τελική πολυπλοκότητα είναι γραμμική ως προς το μήκος της εισόδου:

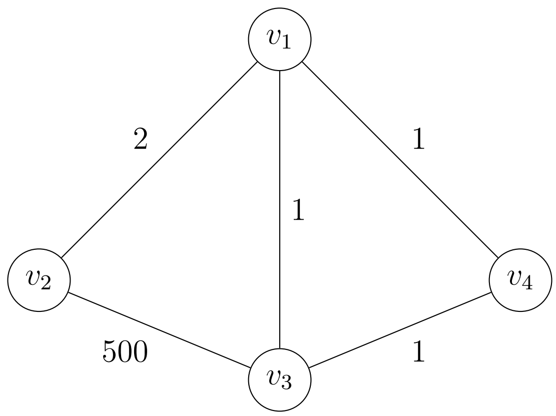
$$\boxed{O(n+m)}$$

4 Ελάχιστο Συνδετικό Δέντρο με Περιορισμούς

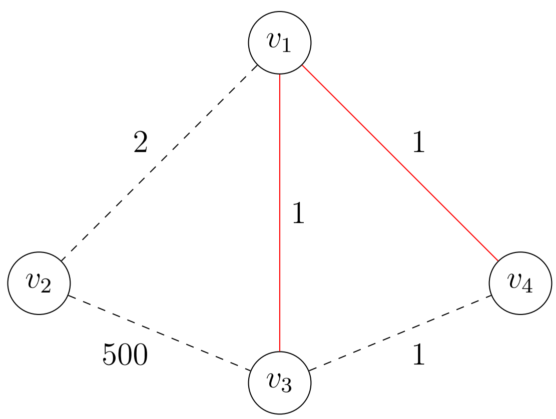
4.1 Ερώτημα (α)

Για να αποδείξουμε τη μη βελτιστότητα της άπληστης στρατηγικής, αρκεί να βρούμε ένα αντιπαράδειγμα στο οποίο η εφαρμογή της δεν οδηγεί στο επιθυμητό απότέλεσμα.

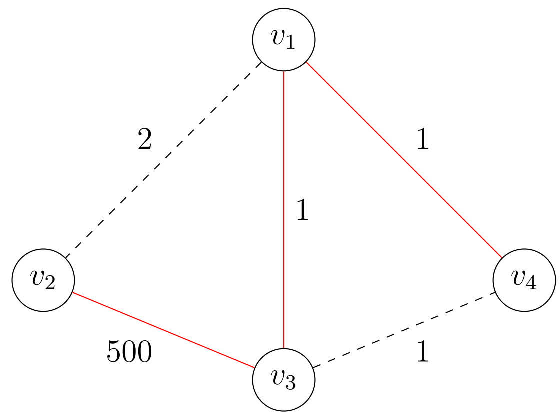
Έστω ότι έχουμε τον παρακάτω γράφο.



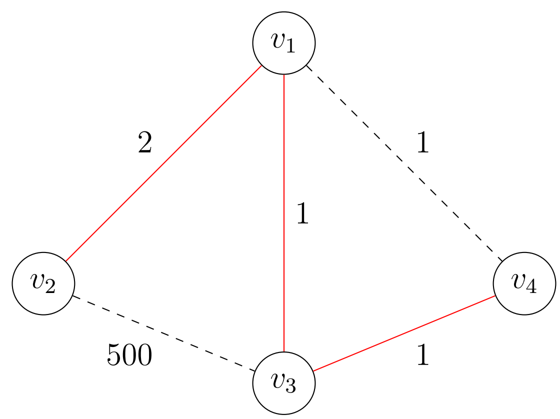
Για να σχηματίσουμε το $T^*(v_1, 2)$, τότε αν επιλέγαμε άπληστα, θα διαλέγαμε τις ακμές $v_1 - v_3$ και $v_1 - v_4$. Και το σχηματιζόμενο δέντρο μέχρι στιγμής θα ήταν:



Επομένως, θα αναγκαζόμασταν να επιλέξουμε την ακμή $v_2 - v_3$ για να καλύψουμε την κορυφή v_2 και το συνολικό κόστος του δέντρου θα ήταν 502.



Αν όμως η αρχική (απληστη) επιλογή μας αντικατασταθεί από την επιλογή των ακμών $v_1 - v_2$ και $v_1 - v_4$, τότε με την ακμή $v_4 - v_3$ θα καλύπταμε και την κορυφή v_3, έχοντας συνολικό κόστος 4.



Προφανώς, το δέντρο που προκύπτει με την μη άπληστη επιλογή είναι καλύτερο από το πρώτο, καθώς είναι βέλτιστο.

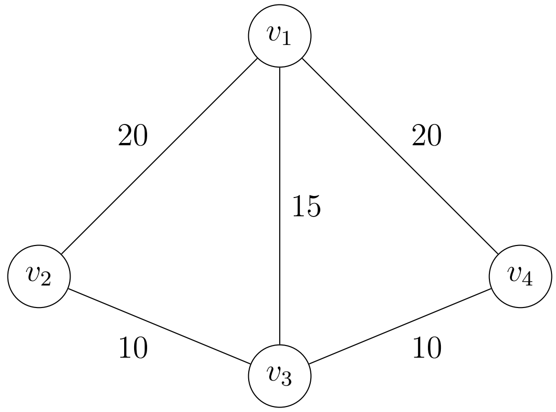
4.2 Ερώτημα (β)

4.2.1 Πρώτη Προσέγγιση

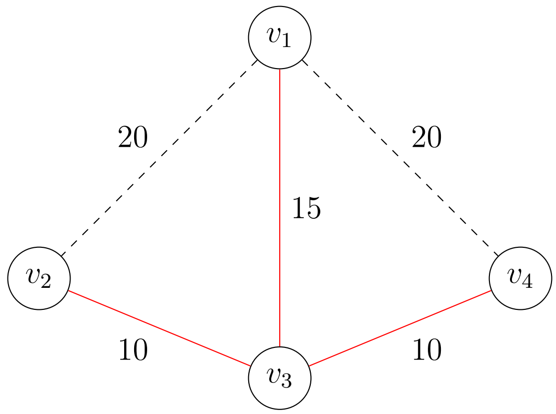
Μια αρκετά εμφανής λύση θα ήταν να επιλέγαμε ένα συνδυασμό k ακμών που προσπίπτουν στην s και στη συνέχεια, για τον υπόλοιπο γράφο να εκτελέσουμε τον αλγόριθμο $Kruskal$ για να επιλέξουμε τις υπόλοιπες ακμές. Για να δοκιμάσουμε όμως όλους τους συνδυασμούς $k-$άδων ακμών, θα πρέπει να εκτελέσμουμε τελικά $Kruskal$ τόσες φορές όσοι είναι οι συνδυασμοί των πιθανών $k-$άδων, δηλαδή $\Theta(\binom{n}{k})$ φορές. Θα προσπαθήσουμε να βρούμε, λοιπόν, κάτι πιο αποδοτικό.

4.3 Βασική Παρατήρηση

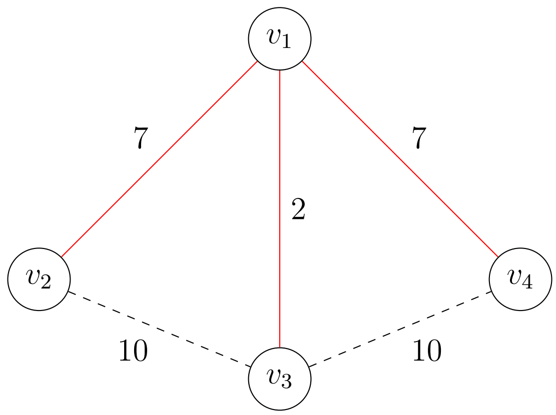
Τι είναι αυτό που κάνει μια ακμή να επιλεγεί έναντι μίας άλλης στον αλγόριθμο του $Kruskal$; Είναι το βάρος της σε σχέση με το βάρος των υπολοίπων ακμών. Αυτό σημαίνει ότι παρότι οι ακμές που θα επιλέξουμε για να φτιάξουμε το βέλτιστο $T^*(s,k)$ μπορεί να μην συμπεριλαμβάνονται όλες στο ΕΣΔ του γράφου (που δεν υποκειται σε περιορισμούς), αν είχαν βάρη αρκετά μικρότερα από αυτά που έχουν, θα ερχόντουσαν πιο κάτω στην ταξινόμηση των ακμών και θα επιλέγονταν νωρίτερα από τις άλλες ακμές που έχουν χρησιμοποιηθεί για να καλυφθεί ολόκληρος ο γράφος. Για παράδειγμα, έστω ότι έχουμε τον παρακάτω γράφο:



Ο $Kruskal$ θα παρήγαγε το παρακάτω ΕΣΔ.



Αν, όμως, μειώναμε το βάρος των ακμών $v_1 - v_2$, $v_1 - v_3$ και $v_1 - v_4$ κατά 13, τότε ο $Kruskal$ θα παρήγαγε το παρακάτω ΕΣΔ:



Άρα, αν απλώς οι ακμές από την s είχαν κατάλληλο βάρος, τότε θα επιλεγόντουσαν ακριβώς k από αυτές στο ΕΣΔ του $Kruskal$. Τότε, όμως, με δεδομένο ότι έχουν επιλεγεί αυτές οι k ακμές, οι υπόλοιπες, είναι ελαχίστου κόστους. Άρα και στο αρχικό μας γράφημα αν "καρφώσουμε" τις ίδιες ακμές, οι υπόλοιπες που θα επιλεγούν θα είναι αυτές που ελαχιστοποιούσαν το κόστος του δεύτερου γραφήματος.

Συνεπώς, θα πρέπει να βρούμε το κατά πόσο πρέπει αν αυξηθούν ή να μειωθούν τα κόστη των ακμών που προσπίπτουν στην s. Έστω f ο αριθμός που προθέτουμε σε όλες τις ακμές που προσπίπτουν στην s και T' το νέο ΕΣΔ. Αν το f είναι κατάλληλο, τότε $T' \equiv T^*(s, k)$.

Απόδειξη

Το να "καρφώσουμε" k ακμές στο ΕΣΔ ισοδυναμεί με το να της "μετακινήσουμε" στην κατάλληλη θέση στον ταξινομημένο πίνακα του Kruskal.
Το να τις μετακινήσουμε στην ταξινόμηση, όμως, ισοδυναμεί με το να τους αλλάξουμε το βάρος κατάλληλα. Με κατάλληλη (και ισοδύναμη) αύξηση όλων των ακμών της s, αυτές πηγαίνουν (ισαπέχοντας μεταξύ τους) πάνω ή κάτω στην κατάταξη.
Άρα με πρόσθεση ενός κατάλληλου αριθμού f στο βάρος όλων τους, μετακινούνται στην ταξινόμηση και k εξ αυτών επιλέγονται έναντι άλλων. Ανάλογα με το f:

- όσο αυξάνεται, τόσο ανεβαίνουν στην ταξινόμηση $\longrightarrow$ όλο και λιγότερες επιλέγονται
- όσο μειώνεται, τόσο κατεβαίνουν στην ταξινόμηση $\longrightarrow$ όλο και περισσότερες επιλέγονται

Άρα, ανάλογα με το f μπορούμε να πετύχουμε όποιο (δυνατό) k θέλουμε.
Τελικώς, αλλάζουμε το βάρος των ακμών της s κατά f (κατάλληλο) και το ΕΣΔ που δίνει ο *Kruskal* είναι ο ίδιο που θα έδινε, αν καρφώναμε εξαρχής τις k ίδιες ακμές που επιλέχθηκαν όταν είχαμε αύξηση κατα f.

Το πρόβλημα πλέον ανάγεται στον εντοπισμό του κατάλληλου f.

4.3.1 Διατύπωση Αλγορίθμου

Δεν θα βρούμε το f κάνοντας γραμμικη αναζήτηση. Κι αυτό, γιατί θα εκμεταλλευτούμε την μονοτονία που παρουσιάζει το f με το k (όπως αυτή φαίνεται στην παραπάνω απόδειξη). Παρατηρούμε ότι τα f και k είναι αντιστρόφως ανάλογα. Άρα, μπορούμε να βρούμε το f με binary search.

Αλγόριθμος - Restricted MST

1. Κάνοντας binary search στο $[-e_{min}(s), E]$, όπου E : το μέγιστο βάρος ακμής του γράφου και $e_{min}(s)$: το ελάχιστο βάρος ακμής που προσπίπτει στην s, αναζητούμε το κατάλληλο f.
2. Για κάθε f που δοκιμάζουμε, βρίσκουμε - με Kruskal - το MST που προκύπτει αν αυξήσουμε τις ακμές που προσπίπτουν στην s κατά f.
3. Αν επιλεγούν στο MST, λιγότερες από k ακμές, ψάχνουμε για f μικρότερο από το επιλεγμένο. Αν επιλεγούν στο MST, περισσότερες από k ακμές, ψάχνουμε για f μεγαλύτερο από το επιλεγμένο.
4. Όταν οι επιλεγόμενες ακμές είναι k, σταματάμε γιατί εντοπίσαμε το $T' \equiv T^*(s, k)$

4.3.2 Ορθότητα

Η ορθότητα του αλγορίθμου αποδεικνύεται από την παράγραφο "Βασική Προσέγγιση" και από την απόδειξη περί ισοδυναμίας των T' και $T^*(s,k)$.

Προσθέτουμε σε αυτό το σημείο, ότι η αναζήτηση του κατάλληλου f έγινε στο διάστημα $[-e_{min}(s), E]$, καθώς αν αυξηθούν όλες ο ακμές της s κατά E, τότε σίγουρα θα επιλεγεί μόνο μία από αυτές, ώστε να καλυφθεί η s, ενώ αν μειωθούν όλες κατά τη μέγιστη δυνατή διαφορά (που δεν δημιουργεί αρνητικές ακμές), τότε θα επιλεγούν όλες.

4.3.3 Υπολογιστική Πολυπλοκότητα

- Για την δυαδική αναζήτηση έχουμε πολυπλοκότητα $O\Big(E_{max}log(E_{max})\Big)$
- Για κάθε τιμή του f που δοκιμάζουμε, βρίσκουμε το MST με $Kruskal$, άρα έχουμε πολυπλοκότητα $O(mlogm)$.

Άρα, συνολικά έχουμε πολυπλοκότητα:

$$\boxed{O\Big(m \cdot E_{max} \cdot logm \cdot log(E_{max})\Big)}$$

5 Αλγόριθμος Borůvka

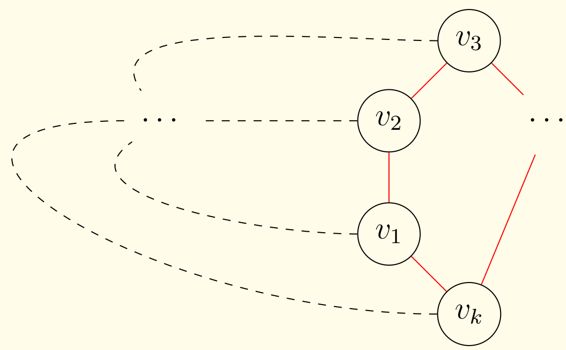
5.1 Ερώτημα α: Απόδειξη Ορθότητας Borůvka

5.1.1 (i)

Θέση προς απόδειξη: Ο αλγόριθμος Borůvka καταλήγει πάντα σε ένα συνδετικό δέντρο T.

Απόδειξη

Θα υποθέσουμε την άρνηση του ζητουμένου και θα καταλήξουμε σε άτοπο.
Έστω ότι σε κάποια φάση του αλγορίθμου, σχηματίζεται κύκλος που εμπλέκει τις συνεκτικές συνιστώσες $v_1, v_2, \cdots, v_k$. Παραθέτουμε και σχηματικά την αφηρημένη εκδοχή του σχηματιζόμενου κύκλου:



Έστω ότι από τις ακμές του κύκλου, πρώτα επιλέγεται η (v_1, v_2) από τη συνιστώσα v_1, χωρίς βλάβη της γενικότητας. Κάποια στιγμή η συνιστώσα v_2 θα πρέπει να επιλέξει ακμή. Έχει επιλεγεί ήδη η (v_1, v_2), από την επιλογή της v_1. Για να επιλέξει τότε η v_2 την (v_2, v_3) θα πρέπει να ισχύει $w(v_1, v_2) > w(v_2, v_3)$. Για τον εντελώς αντίστοιχο λόγο θα πρέπει αν η v_3 επιλέξει την (v_3, v_4), τότε ισχύει $w(v_2, v_3) > w(v_3, v_4)$.
Χωρίς να είναι απαραίτητο οι ακμές να επιλέγονται με τη σειρά, σε οποιοδήποτε στιγμή αν κάποια συνιστώσα επιλέξει ακμή, έστω e', του κύκλου διαφορετική από αυτή της γειτόνισσάς της, έστω e, τότε θα πρέπει $w(e') < w(e)$.

Άρα, εν τέλει, θα ισχύει η σχέση: $w(v_1, v_2) > w(v_2, v_3) > w(v_3, v_4) > \cdots > w(v_{k-1}, v_k)$. Για τον ίδιο λόγο με προηγουμένως, προκειμένου να κλείσει ο κύκλος, δηλαδή η v_k να επιλέξει την (v_k, v_1) θα πρέπει να ισχύει $w(v_{k-1}, v_k) > w(v_k, v_1)$. Άρα, τελικά:

$w(v_1, v_2) > w(v_2, v_3) > w(v_3, v_4) > \cdots > w(v_{k-1}, v_k) > w(v_k, v_1) \Longrightarrow w(v_1, v_2) > w(v_k, v_1)$.
Αυτό σημαίνει, όμως, ότι η επιλογή της ελαχίστου βάρους ακμής που προσπίπτει στην v_1 θα έπρεπε να είναι η (v_1, v_k) και όχι η $(v_1, v_2) \Longrightarrow$ Άτοπο.

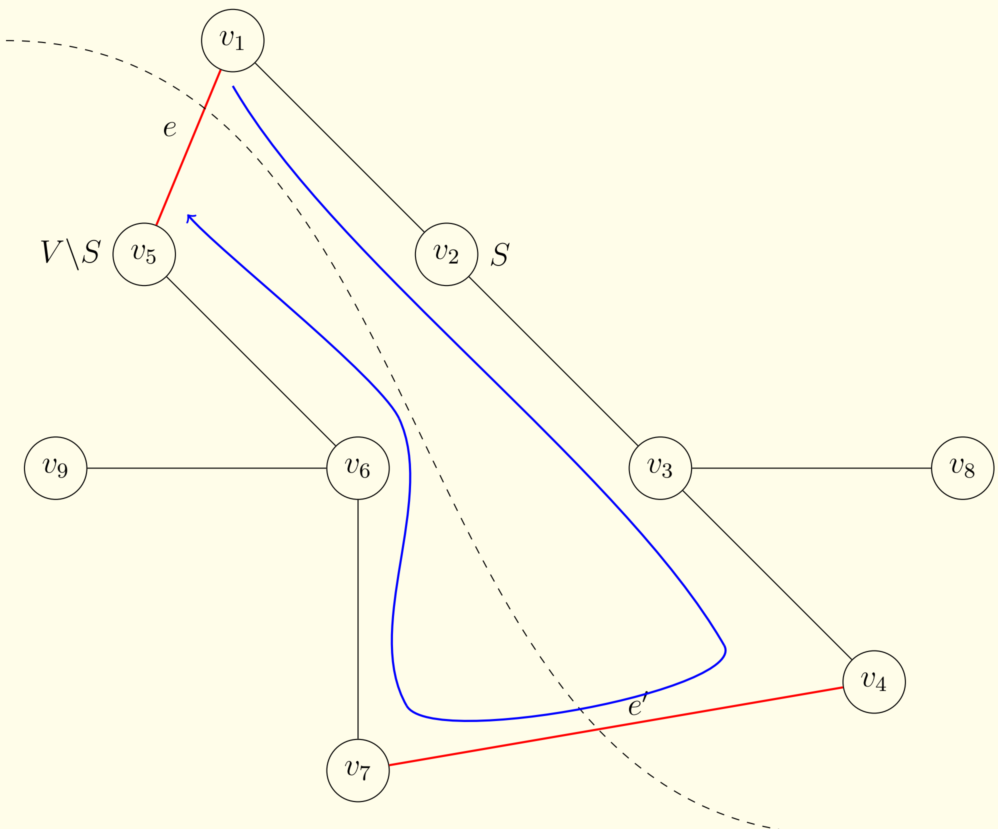
Άρα, δεν δύναται να υπάρξει κύκλος και, γι'αυτό, ο αλγόριθμος Borůvka καταλήγει πάντα σε ένα συνδετικό δέντρο T.

5.1.2 (ii)

Θέση προς απόδειξη: Το συνδετικό δέντρο T στο οποίο καταλήγει ο αλγόριθμος Borůvka είναι πάντα ελαχίστου βάρους.

Απόδειξη

Θα υποθέσουμε την άρνηση του ζητουμένου και θα καταλήξουμε σε άτοπο. Υποθέτουμε ότι ο αλγόριθμος Borůvka δεν παράγει πάντοντε ελάχιστου κόστους δέντρο. Αυτό σημαίνει, ότι υπάρχει MST που δεν χρησιμοποιεί μόνο τις ελαχίστου κόστους προσπίπτουσες ακμές σε κάθε κορυφή. Δηλαδή, εκεί που θα επέλεγε ο Borůvka μια ακμή ως την καλύτερη για μια κορυφή, υπάρχει MST που θα διάλεγε μια άλλη στη θέση της.
Θεωρούμε την τομή του MST, έστω $(S, V \setminus S)$, η οποία γεφυρώνεται από την ακμή e'. Η e' έχει επιλεγεί στη θέση μιας άλλης που θα επέλεγε ο Borůvka. Αυτό σημαίνει ότι υπάρχει τουλάχιστον μία ακμή που έχει μικρότερο βάρος από την e (πρώτα και κύρια αυτή που θα επέλεγε ο Borůvka έναντι αυτής).



Κάθε τομή που χωρίζει το MST σε δύο συνεκτικές συνιστώσες γεφυρώνεται από μοναδική ακμή (ως γνωστόν). Αν λοιπόν προσθέσουμε στο MST την ακμή e που έχει μικρότερο βάρος από την e' (η οποία υπάρχει όπως δείξαμε πριν) τότε κλείνει κύκλος που αναγκαστικά διέρχεται από την e'. Αφαιρώντας την e' τότε ο κύκλος "σπάει" και η τομή που γεφυρωνόταν με την e', πλέον γεφυρώνεται από την e, διατηρώντας την ιδιότητα του MST να είναι δέντρο και με κόστος μικρότερο του προηγούμενου, αφού η νέα ακμή e έχει μικρότερο κόστος από την παλιά. Αφού το νέο MST έχει κόστος μικρότερο του παλιού, τότε το παλιό δεν ήταν MST $\implies$ Άτοπο. Άρα, ο αλγόριθμος Borůvka παράγει πάντοντε ελάχιστου κόστους δέντρο.

5.2 Ερώτημα β: Υλοποίηση Αλγορίθμου με Πολυπλοκότητα $O(mlogn)$

Ο αλγόριθμος του Borůvka αποτελείται από τα εξής βήματα:

1. Αρχικοποιούμε όλες τις κορυφές ως ανεξάρτητα σύνολα.
2. Αρχικοποιούμε το MST ως κενό.
3. Επαναλαμβάνουμε μέχρι να έχουμε μοναδικό σύνολο. Για κάθε σύνολο:
   - Βρίσκουμε την πιο φθηνή ακμή που το συνδέει με ένα διαφορετικό σύνολο
   - Προσθέτουμε αυτή την ακμή στο MST.
4. Επιστρέφουμε το MST.

Παραθέτουμε, στη συνέχεια, μια αναλυτική υλοποίηση του αλγορίθμου με χρήση $Union - Find$. Επισημαίνουμε ότι παρακάτω, ο όρος "ανεξάρτητα σύνολα" χρησιμοποιείται με την έννοια ότι τα σύνολα στα οποία αναφέρεται δεν έχουν καμία ακμή που να ξεκινάει από το ένα και να καταλήγει στο άλλο. Δηλαδή, ανεξάρτητο σύνολο = συνεκτική συνιστώσα = δέντρο.

Υλοποίηση Borůvka σε $O(mlogn)$

1. Θεωρούμε ότι ο γράφος αναπαρίσταται ως σύνολο ακμών.
2. Αρχικοποιούμε τον αριθμό ανεξάρτητων συνόλων σε $|V|$.
3. Δημιουργούμε έναν πίνακα $cheapest$ που διαθέτει στη θέση i την φθηνότερη ακμή που επιλέγει το σύνολο i.
4. Επαναλαμβάνουμε μέχρι ο αριθμός συνόλων να γίνει 1.
5. Αρχικοποιούμε τις θέσεις του πίνακα $cheapest$ σε -1, στην αρχή της επανάληψης.
6. Στη συνέχεια, διατρέχουμε τη λίστα με τις ακμές και για κάθε ακμή e:
   - Βρίσκουμε με $find()$ την κορυφή αντιπρόσωπο s_1 του συνόλου στο οποίο ανήκει το ένα άκρο της e και την κορυφή αντιπρόσωπο s_2 του συνόλου στο οποίο ανήκει το άλλο.
   - Αν οι δύο κορυφές αντιπρόσωποι ταυτίζονται, τότε η e ή ανήκει ήδη στο MST ή κλείνει κύκλο, οπότε την αγνοούμε.
   - Διαφορετικά:
     - αν η e έχει βάρος μικρότερο από το βάρος της ακμής με αριθμό το $cheapest[s_1]$ ή το $cheapest[s_1]$ είναι ακόμα αρχικοποιημένο σε -1, τότε ανανεώνουμε το $cheapest[s_1]$ με τον αριθμό της e. Εντελώς το ίδιο και για το $cheapest[s_2]$.
7. Ύστερα, για κάθε σύνολο s:
   - Αν το $cheapest[s]$ έχει τεθεί σε δείκτη ακμής $i \neq -1$, βρίσκουμε τους αντιπροσώπους των συνόλων που "ακουμπάει" η ακμή με δείκτη i.
   - Αν δεν ταυτίζονται, τότε προσθέτουμε την ακμή i στο MST. Ενώνουμε με $union()$ τα δύο σύνολα σε ένα.
   - Μειώνουμε τον αριθμό των ανεξάρτητων συνόλων. Επαναλαμβάνουμε από το βήμα (4).

Εφόσον σε κάθε επανάληψη, ο αριθμός των συνιστωσών μειώνεται τουλάχιστον στο μισό χρεωνόμαστε έναν παράγοντα $O(logn)$.

Ύστερα, διατρέχουμε κάθε φορά όλες τις ακμές και για κάθε ακμή εκτελούμε κάποιες απλές λειτουργίες εκ των οποίων μία είναι η $find()$. Ύστερα διατρέχουμε όλα τα σύνολα κι εκτελούμε union() για να ενώσουμε αυτά που ενώνονται με τις ακμές που έχουμε επιλέξει. Χρεωνόμαστε δηλαδή με $O(m)$.

Συνολική πολυπλοκότητα: $O(mlogn)$.

5.3 Ερώτημα γ: Βελτίωση πολυπλοκότητας σε $O(mloglogn)$

- Πολυπλοκότητα Borůvka: $O(mlogn)$
- Πολυπλοκότητα Prim (υλοποίηση με Fibonacci Heap): $O(nlogn)$

Παρατηρούμε ότι οι δύο πολυπλοκότητες διαφέρουν ως προς τον πολυωνυμικό τους παράγοντα. Με λίγα λίγα, αν το m είναι σημαντικά μικρότερο του n τότε θα συνέφερε η εύρεση του MST με Borůvka αντί για Prim. Φυσικά, ισχύει και το αντίστροφο. Αυτό μπορούμε να το εκμεταλλευτούμε.

Καθώς ο αλγόριθμος του Borůvka τρέχει, το n διαρκώς μειώνεται, ενώ το m παραμένει σταθερό. Άρα, ύστερα από ένα κατάλληλο σημείο θα συνέφερε να σταματήσουμε να εκτελούμε τον Borůvka, γιατί πλέον το m είναι σημαντικά μεγαλύτερο του n. Σταματάμε, λοιπόν, τον Borůvka και συνεχίζουμε την εύρεση του MST με χρήση του Prim.

Εκτελούμε, λοιπόν, k φορές Borůvka και συνεχίζουμε με Prim. Θα έχουμε, τότε:

$$O\Big(k \cdot m\Big) + O\Big(\frac{n}{2^k}log(\frac{n}{2^k})\Big) = O\Big(k \cdot m + \frac{n}{2^k}log(\frac{n}{2^k})\Big)$$

Για $k = loglogn$, έχουμε:

$$O\Big(loglogn \cdot m + m + \frac{n}{2^{loglogn}}log(\frac{n}{2^{loglogn}})\Big) =$$

$$O\Big(loglogn \cdot m + \frac{n}{logn}log(\frac{n}{logn})\Big) =$$

$$O\Big(loglogn \cdot m + n(1 - \frac{loglogn}{logn})\Big) =$$

$$O\Big(loglogn \cdot m + n\Big) =$$

$$\boxed{O\Big(mloglogn\Big)}$$

5.4 Ερώτημα δ: Ειδική Κλάση Γράφων - Βελτίωση Πολυπλοκότητας σε $O(n)$

Για την ειδική περίπτωση της κλάσσης C μπορούμε να εργαστούμεμε παραλλαγμένο τον Borůvka ως εξής:

1. Εισάγουμε τον γράφο εισόδου G

2. Δημιουργούμε έναν προσωρινό γράφο G' που περιέχει τα ίδια σύνολα κόμβων. Για κάθε κόμβο v, ο G' περιέχει την φθηνότερη ακμή που προσπίπτει στον v.
3. Συμπτύσουμε τις ακμές του G' στον G: βρίσκουμε τις συνεκτικές συνιστώσες του G' και επαναριθμούμε κάθε κόμβο του G με τον αριθμό της συνεκτικής συνιστώσας όπου αυτός ανήκει.
4. Καθαρίζουμε τον γράφο
5. Επαναλαμβάνουμε μέχρι να μην μείνουν ακμές

Εξηγούμε στη συνέχεια το κάθε βήμα εξάγοντας την πολυπλοκότητά του.

1. Εισάγουμε τον γράφο $\longrightarrow O(n)$
2. Βρίσκουμε τις ακμές που θα επέλεγε ο Borůvka όπως εξηγούμε σε προηγούμενη παράγραφο $\longrightarrow O(m)$
3. Η σύμπτηξη έχει πολυπλοκότητα που οφείλεται στην διαγραφή των παράλληλων ακμών πλην της φθηνότερης (το μόνο πρόβλημα που προκύπτει, αφού λόγω του ότι τα βάρη είναι διακεκριμένα δεν μπορεί να προκύψει κύκλος). Κάνουμε, λοιπόν, την σύμπτυξη και καθαρίζουμε μετά τον γράφο για να ξαναγίνει απλός.
4. Ο καθαρισμός αφαιρεί τις παράλληλες ακμές και κρατά τη μικρότερη. Αυτό θα γίνει ως εξής $O(n+m)$:
   - Ταξινομούμε όλες τις ακμές του γράφου λεξικογραφικά για να φέρουμε μαζί τις παράλληλες ακμές.
   - Διατρέχουμε τη λίστα ακμών σειριακά και διαγράφουμε τις αχρείαστες ακμές και επανυπολογίζουμε τους βαθμούς των κορυφών. Διαγράφουμε τις κορυφές με μηδενικό βαθμό.
5. Επαναλαμβάνουμε μέχρι να μην μείνουν άλλες ακμές.

Τέλος, σε κάθε επανάληψη μένουν οι μισές κορυφές, άρα και, λόγω της ιδιότητας της κλάσης που μελετάμε, οι μισές ακμές. Επομένως, σε κάθε φάση του αλγορίθμου εκτελούνται λειτουργίες που είναι γραμμικές ως προς το πλήθος των κορυφών της φάσης (αφού $O(n) = O(m)$). Άρα, οι συνολικές λειτουργίες που θα εκτελεστούν είναι:

$$O(n) + O(\frac{n}{2}) + O(\frac{n}{4}) + O(\frac{n}{8}) + \cdots =$$

$$\boxed{\sum_i O(\frac{n}{2^i}) = O(n)}$$

6 Το Σύνολο των Συνδετικών Δέντρων (ΚΤ 4.27 και ΚΤ 4.28)

6.1 Ερώτημα (α)

Άπόδειξη

Έστω $e \in T_1 \backslash T_2$. Τότε, αν προσθέσουμε την e στο T_2 κλείνει κύκλος. Αυτός ο κύκλος όμως πρέπει να περιέχει μια ακμή e' που να μην περιέχεται στο T_1. Διαφορετικά, όλες οι ακμές του κύκλου θα ανήκουν στο T_1, που είναι άτοπο αφού το T_1 είναι δέντρο. Άρα $\forall e \in T_1 \backslash T_2 \;\; \exists e' : e' \in T_2 \backslash T_1$. Άρα, αν αφαιρέσουμε από το T_1 την e και προσθέσουμε την e' (που μόλις δείξαμε ότι σίγουρα υπάρχει), τότε θα έχουμε έναν γράφο με ίσο αριθμό ακμών με πριν, δηλαδή $n - 1 \Longrightarrow$ (συνδετικό) δέντρο.

Αλγόριθμος υπολογισμού της e'

1. Τα συνδετικά δέντρα είναι του G. Οπότε διατηρούμε δύο δυαδικούς πίνακες A_1 και A_2, που αντιστοιχούν στα T_1 και T_2, αντίστοιχα. Κάθε θέση i του πίνακα A_1 περιέχει 1 αν η ακμή i περιέχεται στο T_1, αλλιώς 0. Ομοίως και ο A_2 για το T_2.
2. Εκτελούμε μια φορά σε $O(n+m)$ τον DFS για να βρούμε μονοπάτι που ενώνει τις κορυφές στις οποίες "ακουμπάει" η e.
3. Για κάθε ακμή εκ των ακμών του μονοπατιού, ελέγχουμε αν ανήκει στο T_2. Ο κάθε έλεγχος γίνεται με χρήση των πινάκων, άρα κοστίζει $O(1)$.
4. Όταν βρούμε μια ακμή που δεν ανήκει στο T_2, τότε επιστρέφουμε αυτήν ως e'.

Συνολικό κόστος αλγορίθμου:

$$\boxed{O(n+m) + m \cdot O(1) = O(n+m) = O(n),}$$

αφού η διάσχιση γίνεται πάνω σε δέντρο, άρα ισχύει m = n - 1. Άρα η συνολική πολυπλοκότητα είναι γραμμική ως προς το μήκος της εισόδου.

6.2 Ερώτημα (β)

Αν δείξουμε ότι η απόσταση των κορυφών του H που αντιστοιχούν στα T_1 και T_2 είναι $|T_1 \backslash T_2|$, τότε ο H θα πρέπει να είναι συνεκτικός, διαφορετικά η απόσταση μεταξύ κορυφών που ανήκουν σε διαφορετικές συνιστώσες θα ήταν άπειρη, που θα σήμαινε $|T_1 \backslash T_2| = \infty \implies$ Άτοπο.

Απόδειξη $d(T_1, T_2) = |T_1 \backslash T_2|$

Θα δείξουμε ότι $d(T_1, T_2) = |T_1 \backslash T_2| = n$ με επαγωγή στο n.

Βάση: Για $n = 1$ είναι δεδομένο ότι $d(T_1, T_2) = |T_1 \backslash T_2| = 1$

Υπόθεση: Έστω ότι $d(T_1, T_2) = |T_1 \backslash T_2| = n, \forall n \leq n_0$

Βήμα:

- Έστω $|T_1 \backslash T_2| = n_0 + 1$. Αυτό σημαίνει ότι τα δέντρα T_1 και T_2 διαφέρουν κατά $n_0 + 1$ ακμές. Λόγω του ερωτήματος (α) $\exists T_1'$ που να είναι συνδετικό δέντρο και να προκύπτει αν από το T_1 αφαιρέσουμε μια ακμή και του προθέσουμε μια ακμή του $T_2 \backslash T_1$. Τότε το T_1' διαφέρει από το T_2 κατά n_0 ακμές. Δηλαδή, $|T_1' \backslash T_2| = n_0$, που από την επαγωγική υπόθεση συνεπάγεται ότι:

$$\left.\begin{array}{rcl} d(T_1', T_2) & = & n_0 \\ while \quad d(T_1, T_1') & = & 1 \end{array}\right\} \implies d(T_1, T_2) \leq n_0 + 1 \quad (1)$$

  Ακόμη, από επαγωγική υπόθεση έχουμε ότι αν $d(T_1, T_2) \leq n_0 \implies |T_1 \backslash T_2| \leq n_0$, που είναι άτοπο.

  Άρα, θα πρέπει $d(T_1, T_2) > n_0 \quad (2)$.

  Από (1) και (2) $\implies d(T_1, T_2) = n_0 + 1 = |T_1 \backslash T_2|$.

- Έστω $d(T_1, T_2) = n_0 + 1$. Τότε στο μονοπάτι μήκους $n_0 + 1$, υπάρχει μια κορυφή - γείτονας του T_2 που απέχει 1 από το T_2 και n_0 από το T_1.

  Δηλαδή, $\exists T_2' : d(T_1, T_2') = n_0$ και $d(T_2', T_2) = 1$.

  Λόγω της επαγωγικής υπόθεσης τότε συνεπάγεται ότι: $|T_1 \backslash T_2'| = n_0$ και $|T_2' \backslash T_2| = 1$

  Αυτό σημαινει ότι $|T_1 \backslash T_2| = n_0 \pm 1$. Αν $|T_1 \backslash T_2| = n_0 - 1$, τότε από επαγωγική υπόθεση, θα έπρεπε να ήταν $d(T_1, T_2) = n_0 - 1 \implies$ άτοπο.

  Άρα: $|T_1 \backslash T_2| = n_0 + 1 = d(T_1, T_2)$.

Ύστερα, για να υπολογίσουμε το συντομότερο μονοπάτι από το T_1 στο T_2, δηλαδή να βρούμε με τη σειρά τα ενδιάμεσα δέντρα μέσω των οποίων το T_1 μετατρέπεται στο T_2, ακολουθούμε τον παρακάτω αλγόριθμο:

Αλγόριθμος: Συντομότερο μονοπάτι $T_1 \longrightarrow T_2$

1. Βρίσκουμε τις ακμές του T_2 που δεν ανήκουν στο T_1 και τις αποθηκεύουμε σε μια λίστα $transform$. Αυτό γίνεται διατρέχοντας τις ακμές του T_2 κι ελέγχοντας τον πίνακα A_1 (δυαδικός και περιέχει 1 στη θέση i αν το T_1 περιέχει την $i-$οστή ακμή, αλλιώς 0). Κόστος: $O\big(m(T_2)\big) = O(n)$
2. Για κάθε ακμή e της $transform$, βρίσκουμε μια ακμή e' που ανήκει στο T_1 και όχι στο T_2. Αυτό θα γίνει με τον αλγόριθμο του ερωτήματος (α), με κόστος $O(n)$ για κάθε εύρεση.
3. Προσθέτουμε την e και αφαιρούμε την e' από το T_1. Κόστος $O(1)$.
4. Επαναλαμβάνουμε από το βήμα (2) για κάθε νέο δέντρο T_1' που προκύπτει ($|T_2 \backslash T_1|$ επαναλήψεις).

   Συνολική πολυπλοκότητα: $O(n + n \cdot |T_2 \backslash T_1|) = O(n \cdot |T_2 \backslash T_1|)$

Ο αλγόριθμος είναι ορθός, καθώς λόγω του ότι τα T_1 και T_2 διαφέρουν κατά $|T_2 \backslash T_1|$ ακμές, το ένα μπορεί να μετατραπεί στο άλλο σε $|T_2 \backslash T_1|$ βήματα, αφού σε κάθε βήμα διαφέρουν σε μία λιγότερη. Δείξαμε, όμως προηγουμένως ότι σε $|T_2 \backslash T_1|$ τουλάχιστον βήματα μπορούμε να φτάσουμε στο T_2. Άρα, αφού εμείς φτάνουμε σε ακριβώς $|T_2 \backslash T_1|$, το μονοπάτι μέσω του οποίου το καταφέραμε είναι ένα από τα συντομότερα.