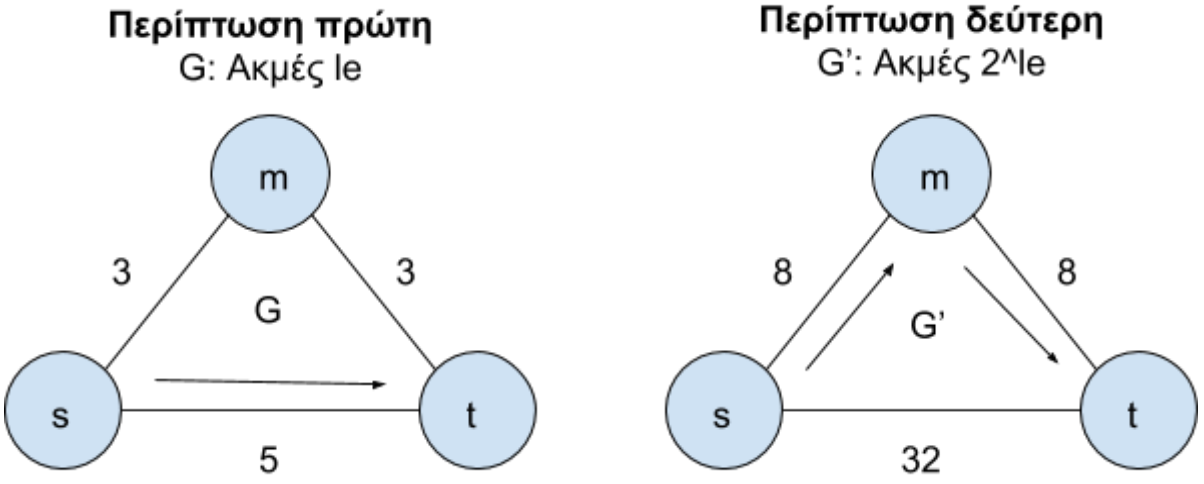
ΕΚΠΑ, Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Αλγόριθμοι και Πολυπλοκότητα
Τμήμα Αρτιων Εαρινό Εξάμηνο 2019-2020 Εργασία 2
Ιωάννης Ροβιθάκης sdi1800164

1. α) Η πρώτη πρόταση είναι **σωστή** διότι οι αλγόριθμοι οι οποίοι υπολογίζουν το/τα ΕΕΔ ενός γράφου βασίζονται στη **σχετική σειρά των μηκών και όχι στις τιμές τους**. Χωρίς Βλάβη της Γενικότητας (ΧΒΓ), αν χρησιμοποιήσουμε τον αλγόριθμο του **Kruskal** αρχικά για τον γράφο με μήκη l_e, στη φάση της ταξινόμησης προκύπτει μια σειρά μεταξύ των ακμών $l_1 \leq l_2 \leq \ldots \leq l_n$. Ο αλγόριθμος του Kruskal για να σχηματίσει ένα ΕΕΔ παίρνει μία μία σε αύξουσα σειρά τις ακμές αυτές. Συνεπώς, αφού η συνάρτηση **f(x) =** 2^x είναι γνησίως αύξουσα, η σχετική σειρά των ακμών διατηρείται $2^{l1} \leq 2^{l2} \leq \ldots \leq 2^{ln}$. Τελικά, αφού ο αλγόριθμος δεν ενδιαφέρεται για τιμές αλλα για τη σχετική σειρά των ακμών, η οποία μένει ίδια, ο αλγόριθμος θα πάρει τις ακμές με την ίδια σειρά και συνεπώς θα καταλήξει στο ίδιο ΕΕΔ.

(**Σημείωση:** δεν μας ενοχλούν πιθανές ισότητες ακμών, απλά οδηγούν συνήθως σε περισσότερα ΕΕΔ, ΧΒΓ θα υπάρχει αντιστοιχία μεταξύ των αναλογών δέντρων με l_e και αυτων με 2^{le})

β) Η δεύτερη πρόταση είναι **λανθασμένη**. Ένα αντιπαράδειγμα είναι το εξής:



Στον Γράφο αυτόν με κλήση ΧΒΓ Dijkstra για εύρεση ελάχιστου μονοπατιού έστω από τον κόμβο s στον κόμβο t, στην πρώτη περίπτωση ελάχιστο μονοπάτι P θα είναι το st ενώ στη δεύτερη παρατηρούμε ότι το ελάχιστο μονοπάτι πλέον είναι το smt.

(Τα σχήματα σχεδιάστηκαν με το ενσωματωμένο εργαλείο σχεδίου των Google Docs)

2. Στην άσκηση αυτή ένας καλός τρόπος να καταλήξουμε με το ελάχιστο τελικό κόστος είναι **σε κάθε νέο βήμα/ένωση πλακών χρυσού να ενώνουμε τα δύο διαθέσιμα κομμάτια με το ελάχιστο δυνατό κόστος**, ώστε το μεγάλο κόστος των "βαριών" κομματιών να μας επιβαρύνει παρα μόνο στα τελευταία βήματα αντί να προστεθεί νωρίς και να αυξάνει σημαντικά το κόστος (= Το άπληστο κριτήριο που χρειαζόμαστε).

Ο αλγόριθμος που προτείνω είναι ο εξής:
i) Ταξινομούμε αρχικά τις ράβδους χρυσού σε **αύξουσα σειρά** ως προς το βάρος τους με κόστος **O(nlogn).**

ii) Σε κάθε νέα ένωση, **ενώνουμε τις 2 ράβδους που βρίσκονται στις 2 πρώτες θέσεις της ταξινομημένης λίστας**, δηλαδή αυτές με το μικρότερο δυνατό άθροισμα με κόστος **O(1).** (εννοείται ότι οι δύο ράβδοι που μολις ενώσαμε αφαιρούνται από τη λίστα με **O(1)**)

iii) Διατρέχουμε την λίστα των ράβδων (η οποία παραμένει ταξινομημένη) και **εισάγουμε την νέα ράβδο** που προέκυψε από την προηγούμενη ένωση στην **κατάλληλη θέση ώστε η λίστα να παραμείνει ταξινομημένη** και συνεπώς τα δύο μικρότερα στοιχεία της να βρίσκονται στις πρώτες δύο θέσεις, με κόστος **O(n)**.

iv) Επιστρέφουμε στο βήμα ii μέχρις ότου η **λίστα μας να έχει ένα μόνο στοιχείο**, οπότε και τελειώνει η εκτέλεση. (Η επανάληψη για μία λίστα η οποία έχει **n** στοιχεία/ράβδους θα γίνει **n-1** φορές)

Από τα παραπάνω προκύπτει η πολυπλοκότητα:
T(n) ~ nlogn + (n-1) * n => **T(n) = O(** n^2 **).**

(Δεν ζητείται αλλά απόδειξη βέλτιστου θα έκανα με την τεχνική "ο άπληστος μένει πάντα μπροστά-υπερτερεί σε κάθε βήμα", δεν μπόρεσα να σκεφτώ αλγόριθμο με καλύτερη πολυπλοκότητα + ο αλγόριθμος δουλεύει ακόμα και με ράβδους ιδιου βάρους)

3. Ο περιοριστικός όρος (bottleneck) του προβλήματος είναι το άρμεγμα των αγελάδων, αφού αναγκαστικά σε κάθε χρονική στιγμή μπορεί να αρμέγεται μόνο μια αγελάδα, δεν μπορούμε με κάποιο τρόπο να εξοικονομήσουμε χρόνο στο στάδιο αυτό. Οπότε θα προσπαθήσουμε να εξοικονομήσουμε χρόνο στα επόμενα δύο στάδια. Γνωρίζοντας ότι όλες οι αγελάδες θα περάσουν μια μια απο το πρώτο στάδιο, μπορούμε να βάλουμε πρώτες για άρμεγμα τις αγελάδες οι οποίες είναι οι πιο αργές στα επόμενα δύο στάδια, έτσι ώστε οι ταχύτερες (στα τελευταία 2 στάδια) αγελάδες που θα ακολουθήσουν να "τις προλάβουν", εξοικονομώντας ετσι χρόνο.

Συνεπώς ο αλγόριθμος που προτείνω είναι ο εξής:

1) Ταξινομούμε πρώτα την λίστα των αγελάδων σε φθίνουσα σειρά ως προς το άθροισμα των χρόνων φαγητού και νερού (ei + di).
2) Αρχίζουμε αρμέγοντας πρώτα την πιο αργή στο φαγητο και νερό αγελάδα όπως προέκυψε από την ταξινόμηση και συνεχίζουμε σε φθινουσα σειρα.

Η πολυπλοκότητα του συγκεκριμένου αλγορίθμου προκύπτει από το στάδιο της ταξινόμησης της λίστας, η οποία με έναν από τους γνωστούς αλγορίθμους ταξινόμησης είναι **O(nlogn).**

Για να αποδείξουμε ότι ο αλγόριθμος είναι βέλτιστος θα χρησιμοποιήσουμε τη μέθοδο της **αντιστροφής**:

Έστω η λύση-σειρά του αλγορίθμου μας **S** και μια βέλτιστη λύση-σειρά **S'.**
Η σειρά της λύσης μας είναι στην πράξη τα $e_i + d_i$ της κάθε αγελάδας ταξινομημένα σε φθίνουσα σειρά. Άρα, αν η S' είναι διαφορετική από την S τότε στην S' Θα υπάρχουν σε κάποιο σημείο δύο αγελάδες έστω α και β τέτοιες ώστε η α να ξεκινάει το άρμεγμα πριν από τη β αλλά να ισχύει $e_\alpha + d_\alpha < e_\beta + d_\beta$ (αλλιώς θα ήταν ταξινομημένη και η S' δηλαδή ίδια με την λύση μας).

Αν αντιστρέψουμε τη σειρά των δύο αυτών αγελάδων, τότε προφανώς η β θα τελειώνει όλο το πρόγραμμα ταχύτερα από ότι τελείωνε πριν αφου θα ξεκινάει νωρίτερα. Η α, ενώ θα τελειώνει πλέον το άρμεγμα όταν θα τελείωνε κανονικά το άρμεγμα η β (είναι δίπλα σε σειρά οπότε μοιράζονται χρόνο) , όμως αφού ισχύει ότι $e_\alpha + d_\alpha < e_\beta + d_\beta$ τελικά και η α θα τελειώσει όλο πρόγραμμα ταχύτερα. Συνεπώς με την αντιστροφή ο χρόνος της λύσης μειώνεται.

Συνεπώς μπορούμε να συνεχίσουμε τις αντιστροφές χωρίς αρνητική επίδραση στην πολυπλοκότητα (με κάθε αντιστροφή η τελικός χρόνος ολοκλήρωσης μειώνεται), μέχρι που θα φτάσουμε σε μια σειρά η οποία δεν έχει πλέον τέτοια στοιχεία α και β για τα οποία να ισχύει $e_\alpha + d_\alpha < e_\beta + d_\beta$ **.** Θα έχουμε λοιπόν ταξινομήσει τις αγελάδες μας σε φθίνουσα σειρά ως προς $e_i + d_i$ **,** δηλαδή θα έχουμε φτάσει στην λύση-σειρά του δικού μας αλγορίθμου S και η πολυπλοκότητα θα είναι πλέον η ελάχιστη δυνατή. Άρα ο S είναι τελικά βέλτιστος.

4. Αφού θέλουμε να βρούμε το μήκος του μικρότερου δυνατου κύκλου του γράφου (αν υπάρχει), στην πράξη **πρέπει να βρούμε όλους τους βέλτιστους κύκλους του γράφου, και να κρατήσουμε τον μικρότερο**.

Έστω ένας γράφος με **n** ακμές και **m** κορυφές:

i) Ξεκινάμε **θεωρώντας ότι ο γράφος μας δεν έχει κανέναν κύκλο** και χρησιμοποιούμε ως συμβολικό μήκος ελάχιστου κύκλου την τιμή 0 (0 γιατί είναι αδύνατο ένας κύκλος σε γράφο με θετικές ακμές να έχει κύκλο μήκους 0 ή μικρότερο΄).

ii) Για κάθε μία από τις κορυφές του γράφου, θέλουμε το ελάχιστο μονοπάτι από αυτή προς τον εαυτό της, χωρίς όμως να περάσουμε από την ίδια ακμή πάνω από μία φορά. Αυτό μπορεί να επιτευχθεί με τον εξή τρόπο:

Για κάθε μία ακμή του γράφου: (Δηλαδή **n** επαναλήψεις)
(Τις παίρνουμε αυθαίρετα, μια μονο φορά την κάθε μια, η σειρά δεν έχει σημασία)

- **Αφαιρούμε την ακμή** από το γράφο προσωρινά,
- Παίρνουμε τις **δύο κορυφές** τις οποίες ένωνε,
- **Βρίσκουμε το ελάχιστο μήκος μονοπατιού από την μία προς την άλλη**, χρησιμοποιώντας τον αλγόριθμο ελαχίστου μονοπατιού του **Dijkstra** (Μπορούμε αφού από υπόθεση ο γράφος έχει θετικά βάρη) (Dijkstra => **O(nlogm)**)
- Αν το μήκος που επέστρεψε ο Dijkstra δεν είναι 0, δηλαδή βρέθηκε μονοπάτι τότε στο μήκος που επέστρεψε ο Dijkstra προσθέτουμε και το μήκος της ακμής που αφαιρέσαμε ώστε να έχουμε το συνολικό μήκος του κύκλου και στη συνέχεια ελέγχουμε αν το μήκος που βρήκαμε είναι μικρότερο από το μήκος που έχουμε αποθηκευμένο ως **ελάχιστο** και αν χρειαστεί το **ενημερώνουμε**. (Αν έχουμε ως αποθηκευμένο ελάχιστο μήκος 0 => βρήκαμε τον 1ο κύκλο αρα απλά περνάμε το μήκος του κύκλου που έχουμε ως ελάχιστο)
- Αν ο Dijkstra μας επέστρεψε 0 τότε δεν υπάρχει κύκλος και δεν κάνουμε τίποτα.
- Ξαναβάζουμε πίσω την ακμή που αφαιρέσαμε στο γράφο και επαναλαμβάνουμε την διαδικασία για την επόμενη ακμή του γράφου.

iii) Τελικά αν έχουμε ακόμα ως ελάχιστο μήκος την τιμή 0 από την αρχή τότε δεν υπάρχει κύκλος στο γράφο μας, αλλιώς η μεταβλητή μας θα έχει το μήκος του συντομότερου μονοπατιού που ψάχνουμε.

Από τα παραπάνω βήματα ο αλγόριθμος αυτός έχει πολυπλοκότητα O(n * nlogm) δηλαδή **O(** n^2 **logm).**