Γεώργιος Δημόπουλος

Α.Μ. 2964

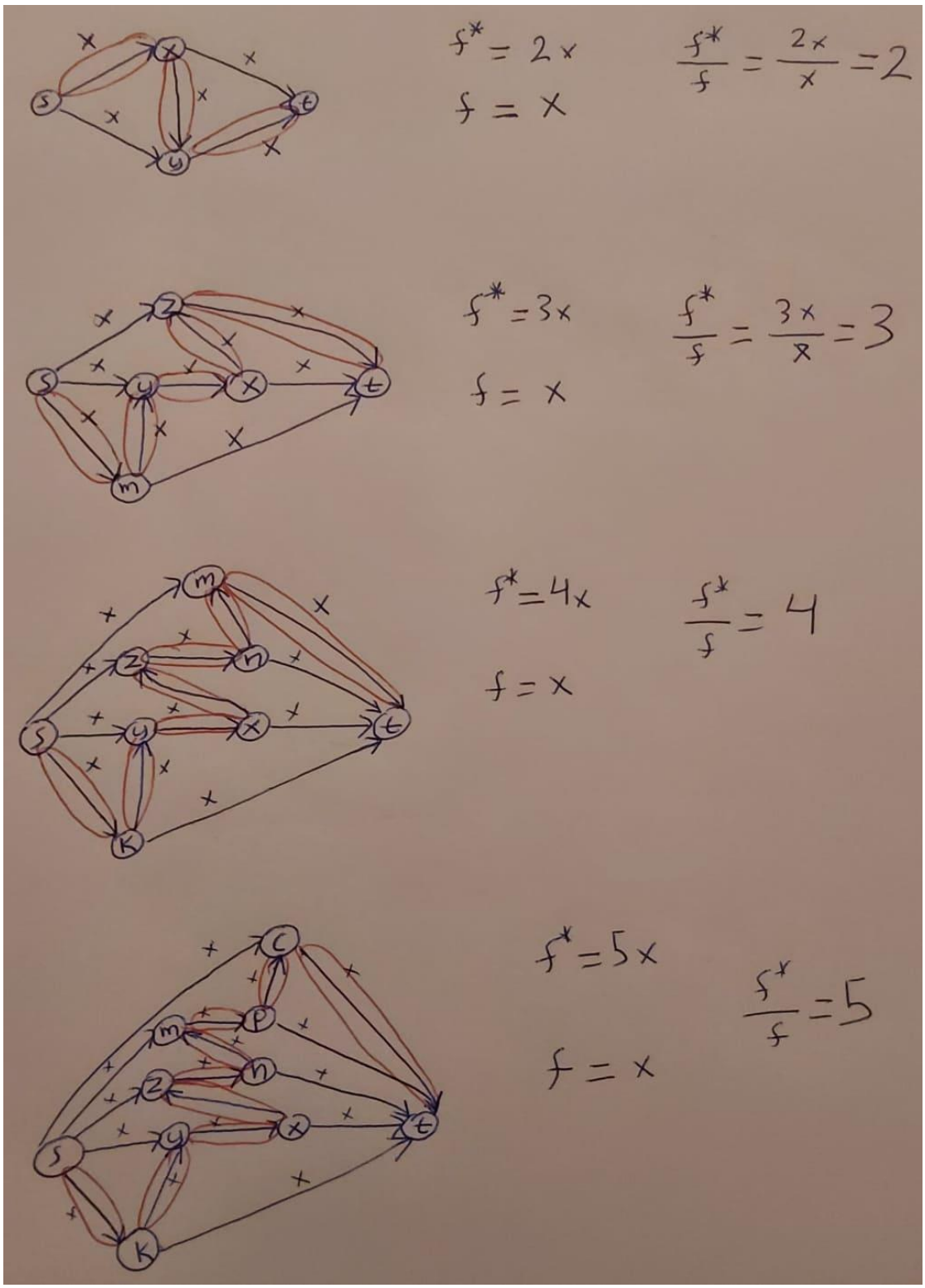
Άσκηση 1 (Αντισταθμιστική ανάλυση δομής FIFO ουράς)

Ο χρόνος χειρότερης περίπτωσης για την λειτουργία **put=O(1),** επειδή αυτή εκτελείται σε σταθερό χρόνο. Απλά βάζει ένα στοιχείο στην ουρά. Ο χρόνος χειρότερης περίπτωσης για την λειτουργία get εξαρτάται από την στοίβα S2. Έτσι λοιπόν αν η στοίβα S2 είναι κενή και η στοίβα S1 έχει n στοιχεία, τότε ο χρόνος χειρότερης περίπτωσης για την **get=O(n).** Το κόστος χειρότερης περίπτωσης μιας ακολουθίας n λειτουργιών στην ουρά είναι **$O(n^2)$**. Αυτό όμως είναι ένα πολύ αυστηρό άνω φράγμα. Θα χρησιμοποιήσω την αθροιστική μέθοδο της αντισταθμιστικής ανάλυσης. Κάθε ακολουθία από n put και get ξεκινώντας από μια άδεια ουρά έχει χρονική πολυπλοκότητα **O(n)**. Αυτό ισχύει γιατί το πλήθος των get από την ουρά δεν μπορεί να υπερβαίνει το πλήθος των put στην ουρά. Το πλήθος των get είναι το πολύ όσο το πλήθος των put, δηλαδή μια λειτουργία pop είναι ακριβή μόνο μια φορά στις n λειτουργίες push. Η συνολική χρονική πολυπλοκότητα είναι: **O(n) για κάθε put λειτουργία + O(2n) για την πρώτη get λειτουργία + O(n-1) για κάθε άλλη get λειτουργία= O(n).** Έτσι λοιπόν με την χρήση της αντισταθμιστικής ανάλυσης προκύπτει ότι η χρονική πολυπλοκότητα κάθε λειτουργίας είναι **O(n/n) = O(1).**

Άσκηση 2 (Υπολογισμός μέγιστης ροής)

Όπως βλέπουμε στην παρακάτω εικόνα υπάρχουν γραφήματα στα οποία ένα αυξητικό μονοπάτι μπορεί να εμποδίσει την ροή από όλα τα άλλα τα μονοπάτια. Έτσι η βέλτιστη ροή θα διαφέρει πολύ σε σχέση με την μέγιστη ροή του δικτύου. Αυτό συμβαίνει επειδή δεν χρησιμοποιούμε το υπολειπόμενο δίκτυο και έτσι αυτό το μονοπάτι μπορεί να μπλοκάρει όλα τα άλλα διαθέσιμα μονοπάτια όπως φαίνεται στην παρακάτω εικόνα. Το μονοπάτι που βρίσκω κάνει το γράφημα μη συνεκτικό. Με τον ίδιο τρόπο απλά προσθέτοντας κατάλληλα 2 κόμβους στο γράφημα μπορούμε να αυξήσουμε την μέγιστη ροή αλλά η βέλτιστη ροή στην χειρότερη περίπτωση να παραμείνει ίδια. Έτσι παρατήρησα ότι ο λόγος **$|f^*|/|f|$=floor(N/2)** όπου N ο αριθμός των κόμβων του γραφήματος. Φυσικά και μπορούμε να έχουμε κόμβους που δεν επηρεάζουν το γράφημα και την μέγιστη και βέλτιστη ροή. Εγώ όμως κοιτάω πάντα το χειρότερο γράφημα και το χειρότερο αυξητικό μονοπάτι που μπλοκάρει όλα τα άλλα. *Έτσι βλέπουμε ότι ο λόγος μέγιστης ροής προς βέλτιστης ροής να είναι αυθαίρετα μεγάλος και για ακρίβεια όσοι οι κόμβοι του γραφήματος δια 2.* Να σημειώσω ότι με κόκκινο κύκλο φαίνεται το χειρότερο μονοπάτι στην παρακάτω εικόνα. (Μπορεί τα παρακάτω γραφήματα να έχουν περιττό αριθμό

κόμβων, αλλά ο επιπλέον κόμβος δεν θα επηρεάζει τον λόγο της μέγιστης προς τη βέλτιστη ροή).



Άσκηση 3 (Υπολογισμός ανεξάρτητων μονοπατιών)

Εφόσον το γράφημα είναι μη κατευθυνόμενο και χωρίς βάρη στις ακμές θα θεωρήσω η κάθε ακμή έχει βάρος 1. Αυτό σημαίνει ότι η μέγιστη ροή θα φανερώνει και το πλήθος των διαφορετικών μονοπατιών που υπάρχουν από τον s στον t. Έτσι αν η μέγιστη ροή είναι μεγαλύτερη ή ίση του 2 τότε σίγουρα έχουμε 2 ή περισσότερα μονοπάτια. Αυτό συμβαίνει επειδή όλες οι ακμές μπορούν να έχουν μέγιστη ροή 1 και έτσι κανένα μονοπάτι δεν μπορεί να χωρέσει ροή μεγαλύτερη ή ίση του 1. Αυτό σημαίνει ότι όση είναι η μέγιστη ροή τόσο είναι και το πλήθος των διαφορετικών μονοπατιών. Το μη κατευθυνόμενο γράφημα θα το κάνω κατευθυνόμενο με τον απλό μετασχηματισμό που είχαμε δει στο πρώτο σετ ασκήσεων. Αυτό που θα κάνω για να βρω άμα έχω 2 ανεξάρτητα μονοπάτια είναι να εκτελέσω τον αλγόριθμο *Ford-Fulkerson*. Μπορώ να εκτελέσω αυτόν τον αλγόριθμο επειδή όλες οι ακμές δέχονται την ίδια ροή και ο αλγόριθμος μου θα τερματίσει σε χρόνο **O(E*f)** όπου E οι ακμές του γραφήματος και f η μέγιστη ροή. Οπότε επειδή θέλω να έχω μέγιστή ροή ίση με 2 (f=2) θα τρέξω τον αλγόριθμο για χρόνο **O(2E)**. Αν τώρα μου δώσει μέγιστη ροή ίση με 2 τότε έχουμε 2 διαφορετικά μονοπάτια, αλλιώς δεν υπάρχουν 2 διαφορετικά μονοπάτια και δεν θα προχωρήσω στον επόμενο αλγόριθμο. Ο αλγόριθμος αυτός όμως μπορεί να μην μου δώσει τα 2 ξεχωριστά μονοπάτια οπότε τώρα θα χρειαστεί να εφαρμόσω τον αλγόριθμο *BFS*, που τρέχει σε χρόνο **O(V+E)**. Για να εκτελέσω τον αλγόριθμο *BFS* ξέρω ότι έχω σίγουρα δύο διαφορετικά μονοπάτια. Οπότε τρέχω μια φορά τον *BFS* και μου δίνει το πρώτο μονοπάτι. Έπειτα διαγράφω τις ακμές που μου έδωσε ο *BFS* και τον ξανατρέχω χωρίς αυτές τις ακμές, ώστε να μου δώσει και το δεύτερο μονοπάτι. Επομένως ο αλγόριθμος του ερωτήματος 3α στην χειρότερη περίπτωση τρέχει σε χρόνο **O(2*E) = O(E)** ενώ ο αλγόριθμος του ερωτήματος 3β τρέχει στην χειρότερη περίπτωση σε χρόνο **O(2(V+E)) = O(V+E).**

Άσκηση 4 (Αντισταθμιστική ανάλυση δομής διατεταγμένων συνόλων)

Έστω ότι έχουμε δύο λίστες A και B. Αρχικά βλέπω ποια λίστα έχει το μεγαλύτερο min σε σταθερό χρόνο με την συνθήκη min(A)>min(B). Έστω ότι η συνθήκη ισχύει (χωρίς βλάβη της γενικότητας), κάνω search(B,min(A)), δηλαδή αναζητώ το μικρότερο στοιχείο της A στην λίστα B. Έστω ότι η search μου επιστρέψει την θέση k. Τότε κάνω split(B,B[k]). Έπειτα δημιουργούνται δύο λίστες έστω Γ για <=B[k] και Δ για >B[k]. Επίσης κάνω split(A,min(A)) και δημιουργούνται δύο λίστες Κ για <=min(A) και Λ για >min(A). Τώρα κάνω join(Γ,Κ) και το αποτέλεσμα της join το αποθηκεύω στην λίστα final. Πλέον επαναλαμβάνεται η ίδια διαδικασία. Βρίσκω το μεγαλύτερο min από τις λίστες Δ,Λ και κάνω split στα αντίστοιχα σημεία. Έπειτα κάνω join εκεί που χρειάζεται. Η μόνη διαφορά τώρα είναι ότι θα χρειαστώ 1 επιπλέον join. Το αποτέλεσμα που μου έβγαλε η πρώτη join θα το κάνω join με την λίστα final και θα αντικαταστήσω την final με το αποτέλεσμα της δεύτερης join. Η επανάληψη αυτή θα σταματήσει στην χειρότερη περίπτωση όταν τελειώσουν όλα τα στοιχεία μιας λίστας. Έστω ότι οι δύο λίστες είναι ίσες και έχουν n=size(A)=size(B) στοιχεία. Συνοψίζοντας χρειαζόμαστε για κάθε επανάληψη 1 search 2 split και 2 join. Άρα

συνολικά **O(5log n)=O(log n)**. Για n επαναλήψεις χρειαζόμαστε συνολικό χρόνο **O(nlog n)**.

Η παρακάτω εικόνα για το 4β ξέρω ότι είναι λάθος, αλλά προσπαθώ να σας δείξω πως πήγα να την προσεγγίσω: