Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Αλγόριθμοι και Πολυπλοκότητα
4η Σειρά Γραπτών Ασκήσεων

Όνομα: Παναγιώτης Κωστοπαναγιώτης
Α.Μ: 03115196

Άσκηση 1. Προτάσεις Φίλων

Αρχικά θα μετασχηματίσουμε λίγο την σχέση που μας δίνεται προκειμένου να μπορέσουμε να εφαρμόσουμε κάποιον γνωστό αλγόριθμο. Έχουμε:

$$t(p) = \prod_{q=0}^{l-1} t(q, q+1) \geq \beta_l \iff ln(\prod_{q=0}^{l-1} t(q, q+1) \geq ln(\beta_l) \iff \sum_{q=0}^{l-1} ln(t(q, q+1)) \geq ln(\beta_l) \ (1)$$

Όμως, έχουμε $t(q), \beta_l < 1 \ \forall q, l$. Συνεπώς $ln(t(q)), ln(\beta_l) < 0$.

Πολλαπλασιάζουμε την σχέση (1) με -1. Λαμβάνουμε:

$(1) \iff \sum_{q=0}^{l-1} -ln(t(q, q+1)) \leq -ln(\beta_l) \ (2)$. Θέτουμε $T(i,j) = -ln(t(i,j)) > 0$ και $\alpha_l = -ln(\beta_l)$. Για τις συνδέσεις με $t(i,j) = 0$, θέτουμε $T(i,j) = \infty$.

Έτσι η σχέση (2) γίνεται: $\sum_{q=0}^{l-1} T(q, q+1) \leq \alpha_l. \ (3)$

Θεωρούμε έναν γράφο με βάρη ακμών $T(i,j)$. Το ζητούμενό μας, όπως προκύπτει από την σχέση (3) είναι, για τον δεδομένο κόμβο i, να βρούμε το σύνολο των κόμβων για τους οποίους υπάρχει μονοπάτι μεγέθους l, για κάποιο $l \leq K$, για το οποίο ισχύει η σχέση (3).

Επομένως για τον κόμβο i υπολογίζουμε την εξής ποσότητα, για κάθε κόμβο που απέχει μέχρι και K από αυτόν. Θεωρούμε $D[u, k]$ ως την ελάχιστη απόσταση στον γράφο που κατασκευάσαμε από τον κόμβο i προς τον κόμβο u με ακριβώς k ακμές. Για τον εν λόγω υπολογισμό εφαρμόζουμε μια διαδικασία παρόμοια με εκείνη της άσκησης 4. Κατά βάση, τρέχουμε τον αλγόριθμο του Dijkstra με αρχική κορυφή i, κρατώντας στο state μας και το πόσες ακμές έχουμε διασχίσει στο μονοπάτι που βρισκόμαστε. Περισσότερες λεπτομέρειες για την συγκεκριμένη διαδικασία υπάρχουν στην λύση της άσκησης 4(α).

Πλέον, έχοντας υπολογίσει τον πίνακα D εξερευνούμε όλους τους κόμβους μέχρι και απόσταση K από τον κόμβο i, με μια BFS για παράδειγμα. Για κάθε κόμβο u ελέγχουμε αν υπάρχει $k \leq K$ τέτοιο ώστε $D[u, k] \leq \alpha_k$. Αν βρούμε κάποιο k για το οποίο ισχύει η εν λόγω σχέση, συμπεριλαμβάνουμε τον κόμβο u στην απάντηση. Αυτό ολοκληρώνει την περιγραφή του αλγορίθμου μας.

Ορθότητα: Ο μετασχηματισμός του ζητούμενου $\prod_{q=0}^{l-1} t(q, q+1) \geq \beta_l$ στο πιο βολικό $\sum_{q=0}^{l-1} T(q, q+1) \leq \alpha_l$ είναι σχετικά προφανής, αφού για να τον εξάγουμε

χρησιμοποιήσαμε απλά κάποιες αλγεβρικές πράξεις. Προκειμένου να βρούμε τους κόμβους για τους οποίους υπάρχει κάποιο l για το οποίο ισχύει η σχέση μας αρκεί να βρούμε τα συντομότερα μονοπάτια μήκους $k \; \forall \; k \leq K$, αφού είναι προφανές πως η ποσότητα που μας ενδιαφέρει αποτελεί βάρος μονοπατιού στον γράφο που κατασκευάσαμε.

Δεδομένου πως υπολογίσαμε τον πίνακα D σωστά (η απόδειξη ορθότητας είναι ακριβώς ίδια με εκείνη της 4(α)) για κάθε κόμβο σε απόσταση το πολύ K ψάχνουμε για κάποιο μήκος μονοπατιού για το οποίο το ελάχιστο μονοπάτι αυτού του μήκους ικανοποιεί την σχέση μας (μας ενδιαφέρει το ελάχιστο μονοπάτι αφού έχουμε άνω φράγμα στο συνολικό βάρος). Βάσει του παραπάνω ελέγχου αποφασίζουμε για το αν θα εισάγουμε τον κάθε κόμβο στο σύνολο των προτάσεών μας.

Πολυπλοκότητα: Ο γράφος μας έχει n κόμβους και m ακμές. Στον γράφο αυτόν τρέχουμε έναν Dijkstra με αρχική κορυφή i και γεμίζουμε έναν πίνακα αποστάσεων μεγέθους $O(nk)$ η πολυπλοκότητα αυτού του σκέλους είναι $O(nK + mlogn)$. Έπειτα, τρέχουμε μια BFS που μας βρίσκει όλους τους κόμβους σε απόσταση μέχρι και K και σε κάθε κόμβο κάνουμε έναν έλεγχο πολυπλοκότητας $O(K)$. Η πολυπλοκότητα αυτού του βήματος είναι επομένως $O(nK + m)$. Συνολικά, έχουμε πολυπλοκότητα $O(nK + mlogn)$. Αν θεωρήσουμε πως το K είναι $O(1)$ λαμβάνουμε πολυπλοκότητα $O(n + mlogn)$ ενώ αν θεωρήσουμε πως το K είναι $O(logn)$ λαμβάνουμε πολυπλοκότητα $O((n + m)logn)$.

Άσκηση 2. Αποφεύγοντας την Συμφόρηση

Στο συγκεκριμένο πρόβλημα, θα μας είναι χρήσιμο το να μπορούμε να ελέγχουμε αποδοτικά, αν μια αυθαίρετη ακμή (u,v) ανήκει σε κάποιο συντομότερο μονοπάτι από την κορυφή s στην κορυφή t. Βάσει της αρχής της βελτιστότητας προκύπτει το εξής αποτέλεσμα:

Λήμμα: Μια ακμή $(u, v) \in E$ ανήκει σε συντομότερο μονοπάτι από την κορυφή s στην κορυφή t αν και μόνο αν ισχύει $D(s, t) = D(s, u) + w(u, v) + D(v, t)$.

Σύμφωνα με το παραπάνω εκτελούμε το εξής στάδιο pre-computation το οποίο θα μας επιτρέπει να απαντάμε για κάθε ακμή σε σταθερό χρόνο το αν ανήκει σε συντομότερο μονοπάτι. Τρέχουμε έναν Dijkstra από την κορυφή s στον γράφο G και κρατάμε τις συντομότερες αποστάσεις σε έναν πίνακα $dist_1[u] = D(s, u)$ καθώς και έναν Dijkstra στον G^T και κρατάμε τις συντομότερες αποστάσεις σε έναν πίνακα $dist_2[u] = D(t, u) \; \forall \; u \in V$. Πλέον, μπορούμε να ελέγξουμε την συνθήκη του λήμματος σε σταθερό χρόνο για κάθε ακμή.

Τώρα κάνουμε το εξής, για κάθε ακμή ελέγχουμε αν ανήκει σε συντομότερο μονοπάτι από τον κόμβο s στον κόμβο t και αν ανήκει την σβήνουμε από τον γράφο. Για την ακρίβεια, κατασκευάζουμε έναν δεύτερο γράφο G' στον οποίο εισάγουμε μόνο ακμές για τις οποίες δεν ισχύει η συνθήκη του λήμματος. Πλέον, ο γράφος μας G' περιέχει μόνο "χρήσιμες" ακμές, αφού όσες ακμές ανήκουν σε κάποιο συντομότερο μονοπάτι s, t δεν μας επιτρέπεται να τις χρησιμοποιήσουμε. Επομένως, αρκεί να τρέξουμε έναν Dijkstra από την κορυφή s στον γράφο G' και να βρούμε την ελάχιστη απόσταση ως την κορυφή t, αυτό είναι και το ελάχιστο δυνατό.

Πολυπλοκότητα: Στον αλγόριθμό μας τρέχουμε συνολικά 3 αλγορίθμους του Dijkstra και κατασκευάζουμε έναν νεό γράφο διασχίζοντας όλες τις ακμές του αρχικού. Δεδομένου ότι η κατασκευή του νέου γράφου γίνεται σε γραμμικό χρόνο η συνολική πολυπλοκότητα του αλγορίθμου μας είναι $O(E + VlogV)$, όση δηλαδή και του Dijkstra.

Άσκηση 3. Τροποποίηση του Αλγορίθμου του Dijkstra

(α) Θα αλλάξουμε την δομή δεδομένων με την οποία εισάγουμε κορυφές. Αντί για Min-Heap θα χρησιμοποιήσουμε έναν απλό πίνακα μεγέθος nC. Ο πίνακας αυτός, ας τον πούμε A, θα κρατάει σε κάθε θέση του τις κορυφές που απέχουν μια συγκεκριμένη απόσταση από την αρχική κορυφή. Συγκεκριμένα, ο πίνακας σε κάθε θέση, έστω i, του θα έχει ένα stack που θα περιέχει όλες τις κορυφές που απέχουν ακριβώς i από την αρχική κορυφή s.

Η υλοποίηση του αλγορίθμου έχει ως εξής. Ξεκινάμε έναν δείκτη από την αρχή του πίνακα, κάθε φορά που θέλουμε να βγάλουμε την κορυφή που απέχει την μικρότερη απόσταση προχωράμε τον δείκτη μέχρι να βρεθούμε σε κάποια θέση που περιέχει κορυφές. Βγάζουμε την πρώτη και εκτελούμε την διαδικασία relax για τους γείτονες της, εισάγοντας όσους χρειάζεται στις αντίστοιχες θέσεις μνήμης. Η διαδικασία αυτή λειτουργεί αφού επισκεπτόμαστε τους κόμβους σε αύξουσα σειρά απόστασης και συνεπώς δεν υπάρχει περίπτωση να κινηθούμε ποτέ αριστερά στον πίνακα A.

Η πολυπλοκότητα των λετουργιών μας είναι $O(C)$ amortized χρόνος για κάθε κορυφή, αφού συνολικά θα διανύσουμε όλον τον πίνακα A στην χειρότερη περίπτωση και $O(1)$ χρόνο για κάθε εισαγωγή. Η συνολική μας πολυπλοκότητα είναι συνεπώς $O(nC + m)$.

(β) Θέλουμε μια δομή η οποία να μας δίνει την δυνατότητα να βρίσκουμε αποδοτικά το ελάχιστο στοιχείο, για την ακρίβεια θέλουμε να υποστηρίζει arg_{min} λειτουργία καθώς και εισαγωγή στοιχείων μέσα στην δομή.

Θα χρησιμοποιήσουμε ένα Segment Tree. Κάθε φύλλο του δέντρου θα κρατάει ένα stack με τους κόμβους που έχουν απόσταση από την αρχική κορυφή όση υποδεικνύει η τιμή του φύλλου του δέντρου. Συνολικά το Segment Tree έχει έυρος $n * 2^C$ (ωστόσο το μέγεθός του είναι μόλις $O(mlogn)$, περισσότερα εδώ) συνεπώς κάθε εισαγωγή στοιχείου, καθώς και η εύρεση του arg_{min} γίνεται σε χρόνο $O(logn + C)$. Ο τρόπος με τον οποίο γίνεται αυτό είναι ακριβώς ο ίδιος με την υποστήριξη Order-Statistics σε ένα απλό BST, η διαφορά είναι πως με το Segment Tree έχουμε την δυνατότητα με μια ελάχιστη αύξηση του χώρου να υποστηρίζουμε αυθαίρετα εύρυ τιμών.

Ορθότητα: Ο αλγόριθμος που περιγράψαμε είναι ακριβώς ο ίδιος με τον αρχικό αλγόριθμο του Dijkstra, με την διαφορά ότι η δομή που χρησιμοποιούμε αντί για Priority Queue είναι ένα Segment Tree. Δεδομένου ότι το Segment Tree λειτουργεί ορθά, η ορθότητα του αλγορίθμου μας είναι τετριμμένη.

Πολυπλοκότητα: Για κάθε εισαγωγή στοιχείου έχουμε χρόνο $O(logn + C)$, το ίδιο και για την εύρεση του ελάχιστου, καθώς και του κόμβου που έχει το ελάχιστο. Συνεπώς, η συνολική πολυπλοκότητα του αλγορίθμου μας είναι $O((n+m)(logn + C)) = O((n+m)C + (n+m)logn)$.

- Πηγές: http://wcipeg.com/wiki/Segment_tree , https://kartikkukreja.wordpress.com/2014/11/09/a-simple-approach-to-segment-trees/

Άσκηση 4: Συντομότερα Μονοπάτια με Δύο Κριτήρια

(α) Θα εφαρμόσουμε μια παραλλαγή του αλγορίθμου του Dijkstra. Για κάθε κορυφή στην οποία καταλήγουμε, δεν θα κρατάμε απλά την απόστασή της από την αρχική, αλλά την ελάχιστη απόσταση δεδομένου ότι έχουμε διασχίσει έναν δεδομένο αριθμό από ακμές τύπου 1. Συγκεκριμένα, έστω $D[u, k]$ η ελάχιστη δυνατή απόσταση από την κορυφή s εως την κορυφή u με κόστος ακριβώς k. Η τελικά μας απάντηση, τότε θα είναι το $min_{k<=K}(D[t, k])$.

Προκειμένου να το πετύχουμε αυτό, εκτελούμε τον αλγόριθμο του Dijkstra από την κορυφή s. Για να μπορέσουμε να υπολογίσουμε την τιμή που θέλουμε, θα αλλάξουμε ελαφρώς την διαδικασία relax. Συγκεκριμένα, έστω πως είμαστε σε μια κορυφή u με κόστος ακριβώς k και κοιτάμε τους γείτονες της κορυφής, έστω πως κοιτάμε την γειτονική κορυφή v με βάρος ακμής $w_{u,v}$ και κόστος $c_{u,v} \in \{0,1\}$ η διαδικασία $relax(u, v)$ γίνεται ως εξής:

$relax(u, v)$:

    $if(D[v, k + c_{u,v}] > D[u, k] + w_{u,v})$:

$D[\,v,\ k\ +\ c_{u,v}\,]\ =\ D[\,u,\ k\,]\ +\ w_{u,v}$

push to priority queue $(\,D[\,v,\ k\ +\ c_{u,v}],\ v\,)$

Στην ουσία, ελέγχουμε αν μπορούμε να μεταβούμε στην επόμενη έγκυρη κατάσταση πληρώνοντας βάρος $w_{u,\,v}$.Αν η εν λόγω μετάβαση μας μειώνει το κόστος την αναβαθμίζουμε και την εισάγουμε στην ουρά προτεραιότητας.

<u>*Ορθότητα:*</u> Θα μπορούσαμε να αιτιολογήσουμε την ορθότητα του παραπάνω αλγορίθμου με κάποιο επιχείρημα ως προς την ορθότητα της αναδρομικής σχέσης η οποία μοιάζει με σχέση δυναμικού προγραμματισμού. Σε αυτή την περίπτωση θα κάναμε μια επαγωγή ως προς το πλήθος των κόμβων που έχουμε ήδη επεξεργαστεί με κόστος $\leq K$. Έτσι θα είχαμε την απόδειξη της αναδρομικής σχέσης.

Ωστόσο, θα κάνουμε μια αναγωγή του προβλήματος στο κλασικό πρόβλημα Single Source Shortest Path. Θεωρούμε πως έχουμε Κ διαφορετικά αντίγραφα του αρχικού γράφου. Το $i-$οστό αντίγραφο του γράφου περιέχει κόμβους στους οποίους έχουμε φτάσει με *ακριβώς* κόστος i. Έτσι, αν κάθε κόμβος αναπαρίσταται από ένα ζεύγος $(\,u,\,i\,)$όπου u ο αριθμός του και i ο γράφος στον οποίο ανήκει. Κατασκευάζουμε τον γράφο ως εξής, για κάθε ακμή $(\,u,\,v,\,w_{u,\,v},\,c_{u,\,v}\,)$ και για κάθε αντίγραφο i, βάζουμε μια ακμή από τον κόμβο $(\,u,\,i\,)$ στον κόμβο $(\,v,\,i\,)$ αν $c_{u,v}\ =\ 0$, εφ' όσον δεν έχουμε επιπλέον κόστος παραμένουμε στο ίδιο αντίγραφο, ενώ αν $c_{u,v}=\ 1$ βάζουμε μια ακμή από τον κόμβο $(\,u,\,i\,)$ στον κόμβο $(\,v,\,i\ +\ 1\,)$. Προφανώς, οι προαναφερθείσες ακμές θα έχουν βάρος $w_{u,\,v}$ Πλέον, ενδιαφερόμαστε για την ελάχιστη απόσταση από τον κόμβο $(\,s,\,0\,)$ στον κόμβο $(\,t,\,i\,)$ για κάποιο $i\ \leq K$. Είναι προφανές πως η αναγωγή που κάναμε στο Shortest Path, επιλύεται από τον αλγόριθμο που γράψαμε παραπάνω αφού μιλάμε απλά για Dijkstra σε έναν "ενισχυμένο γράφο".

<u>*Πολυπλοκότητα:*</u> Στον αλγόριθμό μας γεμίζουμε έναν πίνακα αποστάσεων με VK θέσεις, κατά βάση είναι σαν να έχουμε VK κόμβους στον γράφο μας. Η πολυπλοκότητα δηλαδή είναι, από Dijkstra, $O(\,VK\ +\ ElogE\,)$. Εφ' όσον έχουμε $K\ \leq V$ η πολυπλοκότητά μας είναι $O(\,V^2\ +\ ElogE\,)$.

(β) Εφαρμόζουμε ακριβώς τον ίδιο αλγόριθμο, απλά τώρα, εφ' όσον τα κόστη μας μπορεί να είναι οποιοιδήποτε θετικοί ακέραιοι πρέπει να προσέξουμε τις διαστάσεις του πίνακά μας. Εφ' όσον το κόστος φράσσεται από το $V\ *C_{max}$ κατασκευάζουμε τον πίνακά μας $D[\,u,\,k\,]$ διαστάσεων $V x\,V\,*\,C_{max}$. Κατά τ' άλλα η διαδικασία που ακολουθούμε είναι ακριβώς η ίδια.

<u>*Πολυπλοκότητα:*</u> Η πολυπλοκότητα του αλγορίθμου μας είναι, με αντίστοιχη αιτιολόγηση όπως στο ερώτημα (α), $O(\,V^2C_{max}\ +\ ElogE\,)$ η οποία όμως δεν είναι πολυωνυμική, αφού εξαρτάται από μια παράμετρο που είναι εκθετική ως προς το

μέγεθος της εισόδου, το C_{max}. Ο αλγόριθμος μας είναι ψευδοπολυωνυμικός και πρακτικά, για μικρές τιμές του C_{max} είναι αρκετά αποδοτικός.

Άσκηση 5: Διαφημίσεις στο Διαδίκτυο

Θα επιλύσουμε το πρόβλημα μετασχηματίζοντας το στην εύρεση μέγιστης ροής σε κατάλληλα διαμορφωμένο δίκτυο.

Σχηματίζουμε έναν διμερή γράφο, του οποίου το σύνολο κόμβων V_1 αναπαριστά τις εταιρίες, ενώ το σύνολο κόμβων V_2 αναπαριστά τους πελάτες. Στον γράφο αυτό βάζουμε μια κατευθυνόμενη ακμή από τον κόμβο $u \in V_1$ στον κόμβο $v \in V_2$ βάρους 1 αν και μόνο αν το σύνολο στο οποίο ανήκει ο πελάτης v είναι υπερσύνολο του συνόλου S_u, δηλαδή του συνόλου που απαιτεί η εταιρία u. Επιπλέον εισάγουμε έναν κόμβο sink s τον οποίο συνδέουμε με όλες τις εταιρίες. Συγκεκριμένα δημιουργούμε τις κατευθυνόμενες ακμές (s, u_i) βάρους $c_{u_i} \forall\, u_i \in V_1$. Στην συνέχεια, δημιουργούμε έναν κόμβο t σαν sink με τον οποίο συνδέουμε όλους τους πελάτες. Συγκεκριμένα, δημιουργούμε τις κατευθυνόμενες ακμές βάρους 1 $(v_i, t)\ \forall v_i \in V_2$. Τώρα, τρέχουμε τον αλγόριθμο Ford-Fulkerson στο δίκτυο που δημιουργήσαμε. Αν η μέγιστη ροή ισούται με ακριβώς n, όπου n το πλήθος των πελατών η απάντηση είναι ΝΑΙ, αλλιώς η απάντηση είναι ΟΧΙ. Προκειμένου να ανακτήσουμε το "πρόγραμμα" των διαφημίσεων, κοιτάζουμε το Residual Network και επιλέγουμε τις ακμές από εταιρίες σε πελάτες που έχουν Flow.

Ορθότητα: Αρχικά, σημειώνουμε ότι το πλήθος των διαφημίσεων στο δίκτυο που κατασκευάσαμε είναι μια ροή στο δίκτυο. Το πλήθος των διαφημίσεων, αν θεωρήσουμε ότι πηγάζει από την κορυφή s έχει όλες τις ιδιότητες των ροών (Capacity Constraints, Flow Conservation).

Επιπλέον, Οφείλουμε να εξασφαλίσουμε τα εξής προκειμένου να αποδείξουμε την ορθότητα του αλγορίθμου μας:

1. Κάθε εταιρία μπορεί να στείλει διαφημίσεις σε έγκυρους πελάτες και μόνο σε αυτούς.

   Απόδειξη: Στον δίκτυό μας εισάγουμε ακμές μόνο από εταιρίες σε πελάτες που πληρούν τα κριτήρια που ορίζουν τα σύνολα S των εταιριών. Συνεπώς δεν υπάρχει σύνδεση μεταξύ εταιριών και μη-έγκυρων πελατών.

2. Κάθε εταιρία, έστω i μπορεί να στείλει το πολύ c_i διαφημίσεις.

   Απόδειξη: Πράγματι, με την εισαγωγή βάρους c_i σε κάθε ακμή (s, i)

περιορίζουμε το πλήθος των διαφημίσεων που μπορεί να στείλει κάθε εταιρία σε c_i. Ουσιαστικά, εφ' όσον το πλήθος των διαφημίσεων έχει αντίκρυσμα σε ροή στο δίκτυό μας, περιορίζουμε την μέγιστη ροή που μπορεί να περάσει από κάθε εταιρία i σε c_i.

3. Κάθε εταιρία μπορεί να στείλει στον κάθε πελάτη το πολύ μια διαφήμιση και κάθε πελάτης μπορεί να "δει" το πολύ 1 διαφήμιση.

   Απόδειξη: Το πρώτο εξασφαλίζεται μέσω του βάρους 1 στις ακμές μεταξύ κόμβων στα δύο ανεξάρτητα σύνολα V_1, V_2. Το δεύτερο σκέλος εξασφαλίζεται μέσω του βάρους 1 στις ακμές από τους πελάτες στον κόμβο t(sink). Πρακτικά, αυτό το κομμάτι της κατασκευής δεν διαφέρει καθόλου από την κλασική κατασκεύη για την αναγωγή του Maximum Bipartite Matching σε Max Flow.

Εφ' όσον εξασφαλίσαμε τα παραπάνω, διατυπώνουμε το πρόβλημά μας ως εξής. Υπάρχει ροή μεγέθους n στο δίκτυο; Αν υπάρχει κάποια τέτοια ροή, σίγουρα θα είναι η μέγιστη αφού η ροή του δικτύου φράσσεται από το n(αφού στο sink καταλήγουν n ακμές με χωρητικότητα 1). Συνεπώς ελέγχοντας αν η μέγιστη ροή είναι ίση με n μπορούμε να αποφανθούμε σχετικά με την απάντηση του προβλήματος.

Πολυπλοκότητα: Ο αλγόριθμος Ford-Fulkerson έχει πολυπλοκότητα $O(maxF * E)$, στην χειρότερη περίπτωση έχουμε $E = O(nm)$ και $maxF = n$. Οπότε το τελευταίο στάδιο του αλγορίθμου μας έχει πολυπλοκότητα $O(n^2m)$. Επιπλέον, έχουμε και το στάδιο της κατασκευής του γράφου, για κάθε ζεύγος κόμβων θέλουμε χρόνο $O(k)$ προκειμένου να ελέγξουμε αν πρέπει να βάλουμε ακμή. Συνεπώς, η συνολική πολυπλοκότητα του αλγορίθμου μας είναι $O(n^2m + nmk)$.

Άσκηση 6. Αναγωγές και NP-Πληρότητα

Όλα τα προβλήματα ανήκουν στο NP αφού δεδομένου μιας λύσης μπορούμε να κατασκευάσουμε verifier πολυωνυμικού χρόνου που την ελέγχει. Παρακάτω θα αποδείξουμε ότι είναι και NP-Hard.

3-Διαμέριση: Θα δείξουμε ότι Άθροισμα Υποσυνόλου $<_P$ 3-Διαμέριση. Έστω πίνακας ακεραίων $A = \{\alpha_1, \alpha_2, \dots, \alpha_N\}$ καθώς και ένας αριθμός "στόχος" T. Προκειμένου να πετύχουμε αναγωγή πολυωνυμικού χρόνου από το άθροισμα Υποσυνόλου στην 3-Διαμέριση εισάγουμε 3 νέα στοιχεία, έστω $\alpha_{N+1} = 2\sigma - T$, $\alpha_{N+2} = \sigma + T$ και $\alpha_{N+3} = 2\sigma$, όπου σ είναι το άθροισμα των N πρώτων στοιχείων του πίνακα A. Το άθροισμα όλων των στοιχείων του πίνακα, μετά την εισαγωγή των 3 νέων στοιχείων, είναι 6σ. Θα δείξουμε ότι υπάρχει 3-Διαμέριση στον νέο

πίνακα, αν και μόνο αν υπάρχει υποσύνολο που αθροίζει στο Τ στον αρχικό πίνακα.

- Ευθύ: Έστω πως υπάρχει 3-Διαμέριση στον νέο πίνακα. Τότε, τα 3 νέα στοιχεία που εισάγαμε δεν μπορούν να ανήκουν ανα 2 στο ίδιο σύνολο. Επομένως κάθε ένα από αυτά τα στοιχεία θα ανήκουν το καθένα σε χωριστό σύνολο της διαμέρισης και όλα τα υπόλοιπα στοιχεία της διαμέρισης θα προέρθουν από τον αρχικό πίνακα. Εφ' όσον κάθε σύνολο θα έχει άθροισμα 2σ το τρίτο στοιχείο της διαμέρισης θα περιέχει ακριβώς ένα στοιχείο, το $\alpha_{N+2} = 2\sigma$. Τα 2 που μένουν, εφ' όσον η διαμέριση υπάρχει θα συμπληρωθούν με στοιχεία του αρχικού πίνακα, το πρώτο θα συμπληρωθεί με ένα υποσύνολο που αθροίζει σε Τ και το άλλο με ένα υποσύνολο που αθροιζει σε σ - Τ. Επομένως υπάρχει υποσύνολο αθροίσματος Τ δεδομένου ότι υπάρχει 3-Διαμέριση.

- Αντίστροφο: Έστω πως υπάρχει υποσύνολο αθροίσματος Τ στον αρχικό πίνακα. Τότε, όπως κάναμε παραπάνω, το εισάγουμε στο πρώτο σύνολο της διαμέρισης και παίρνουμε το συμπληρωματικό του, στον αρχικό πίνακα, και το εισάγουμε στο δεύτερο σύνολο της διαμέρισης. Όπως και πριν δεν πειράζουμε το τρίτο σύνολο. Έτσι καταλήξαμε σε 2 σύνολα, το καθένα με άθροισμα 2σ. Αυτό ολοκληρώνει και την απόδειξη.

Συνεπώς, έχουμε πετύχει μια αναγωγή πολυωνυμικού χρόνου από το Άθροισμα Υποσυνόλου στην 3-Διαμέριση. Δεδομένου πως το Άθροισμα Υποσυνόλου είναι NP-Hard, συμπεραίνουμε πως και η 3-Διαμέριση είναι NP-Hard.

<u>Επιλογή Ανεξάρτητων Υποσυνόλων:</u> Θα δείξουμε ότι το πρόβλημα είναι NP-Hard ανάγοντας σε αυτό το 3D-Matching. Θα δείξουμε δηλαδή ότι 3DM $<_P$ ΕΑΥ.

Έστω ένα τυχαίο instance του 3D-Matching, παίρνουμε τις έγκυρες τριάδες και παράγουμε από αυτές το σύνολο S, θέτουμε $k = N$, όπου Ν ο πληθάριθμος των συνόλων X, Y, Z. Η απλή αυτή αναγωγή μας ανάγει το 3DM στην επιλογή ανεξάρτητων υποσυνόλων και είναι σαφώς πολυωνυμικού χρόνου.

Επομένως, καταλήγουμε πως η Επιλογή Ανεξάρτητων Υποσυνόλων είναι NP-Hard.