Αλγόριθμοι
4η Σειρά Γραπτών Ασκήσεων

Ιωάννης Δάρας (03115018, el15018@central.ntua.gr, daras.giannhs@gmail.com)

"You probably know that arrogance, in computer science, is measured in nanodijkstras."
—Alan Kay

---

1 Άσκηση 1

1.1 Προεπεξεργασία

Αρχικά, παρατηρούμε ότι ο γράφος των χρηστών μας δίνεται σε μορφή πίνακα γειτνίασης. Μια πιο βολική αναπαράσταση είναι η μετατροπή του σε λίστα γειτνίασης. Έτσι, θα μπορέσουμε να εκμεταλλευτούμε και στην εκτίμηση της πολυπλοκότητας ότι υπάρχουν m ακμές μεταξύ των χρηστών. Για να μετατρέψουμε τον πίνακα γειτνίασης σε λίστα γειτνίασης έχουμε πολυπλοκότητα $O(n^2)$.

Στην ενότητα της διατύπωσης του αλγορίθμου θα δούμε ότι χρειαζόμαστε να τρέξουμε έναν αλγόριθμο εύρεσης ελάχιστων μονοπατιών. Παρατηρούμε όμως ότι το κόστος για δύο ακμές είναι το γινόμενο του κόστους των ακμών αντί για το άθροισμα τους και οι τιμές στις ακμές δεν είναι κόστη καθώς μικρή εμπιστοσύνη αντιστοιχεί σε μικρή τιμή. Με πιο αυστηρή μαθηματική ματιά, δεν βρισκόμαστε στον τροπικό ημιδακτύλιο $(\oplus \equiv min, \otimes \equiv +)$ για τον οποίο γνωρίζουμε αλγορίθμους εύρεσης συντομότερων μονοπατιών. Χρειάζόμαστε λοιπόν έναν μετασχηματισμό του γράφου ώστε να περάσουμε στον τροπικό ημιδακτύλιο. Ένας τέτοιος μετασχηματισμός είναι το $-log_{10}$ για τα βάρη. Με αυτόν τον μετασχηματισμό μεγάλες εμπιστοσύνες αντιστοιχούν σε μικρά ($0 + \epsilon$ για κάποιο θετικό ϵ) θετικά κόστη, μικρές εμπιστοσύνες αντιστοιχούν σε μεγάλα ($1 - \epsilon$ για κάποιο θετικό ϵ) θετικά κόστη. Ακόμη, η σύνθεση ακμών για σχηματισμό μονοπατιών γίνεται από πολλαπλασιασμός πρόσθεση αφού:

$$-\log_{10}(x_1 \cdot x_2) = (-\log_{10} x_1) + (-\log_{10} x_2)$$

Με βάση αυτές τις παρατηρήσεις, με τον μετασχηματισμό αυτό βρισκόμαστε στον τροπικό ημιδακτύλιο, για τον οποίο γνωρίζουμε αποδοτικούς αλγορίθμους εύρεσης ελάχιστων μονοπατιών.

1.2 Διατύπωση αλγορίθμου

Η βασική ιδέα είναι ότι για τον i χρήστη θα βρούμε τα συντομότερα μονοπάτια του προς κάθε χρήστη j χρησιμοποιώντας από 1 εώς k ακμές. Στη συνέχεια, για κάθε χρήστη j θα βρούμε το συντομότερο από αυτά τα μονοπάτια. Έστω ότι το μονοπάτι αυτό είναι το p_t, που ο συμβολισμός δηλώνει ότι το μονοπάτι αυτό χρησιμοποιεί ακριβώς t ακμές. Θα μετατρέψουμε το κόστος αυτού του μονοπατιού πίσω σε εμπιστοσύνη και θα δούμε αν είναι δεκτό μονοπάτι σύμφωνα με την μελέτη των ειδικών. Αν είναι, κάνουμε την αντίστοιχη πρόταση φιλίας. Αν δεν είναι, δοκιμάζουμε από τα συντομότερα μονοπάτια από τον i στον j που έχουμε υπολογίσει το συντομότερο που χρησιμοποιεί λιγότερες από l ακμές και συνεχίζουμε αναδρομικά μέχρι να μας τελειώσουν τα μονοπάτια ή μέχρι να γίνει η πρόταση φιλίας.

Για την εύρεση των ελαχίστων μονοπατιών που χρησιμοποιούν όλων των ελαχίστων μονοπατιών που χρησιμοποιούν από 0 εώς k ακμές ένας τρόπος να δουλέψουμε είναι με δυναμικό προγραμματισμό. Συγκεκριμένα, σχηματίζουμε έναν πίνακα $DP[n][k]$ που στη θέση του $DP[u][j]$ θα πρέπει να έχει το κόστος του ελάχιστου μονοπατιού για να φτάσουμε από το i στο u χρησιμοποιώντας ως j ακμές.

Η αναδρομική μας σχέση είναι:

$$DP[v][j+1] = min \begin{cases} DP[v][j] \\ min(\ DP[u][j] + w_{uv}\ \forall u : (u,v) \in E\ ) \end{cases} \quad , \qquad \forall v \in V, \forall j \in \{0, ..., k-2\}$$

Οι αρχικές συνθήκες είναι:

$$DP[t][0] = \begin{cases} 0, & t = i \\ \infty, & t \neq i \end{cases}$$

Στη συνέχεια, εφαρμόζουμε τη διαδικασία που περιγράψαμε. Για κάθε πιθανό φίλο, πρέπει να βρούμε, αν υπάρχει, αποδεκτό μονοπάτι. Ξεκινάμε από το $DP[u][k]$ που είναι το συντομότερο μονοπάτι που ξέρουμε. Βρίσκουμε το πλήθος των ακμών που αυτό χρησιμοποιεί κοιτώντας τον δείκτη j στον οποίο ισχύει 1η φορά ότι: $DP[u][k] \neq DP[u][j]$. Το πλήθος των ακμών που χρησιμοποιεί θα είναι $j+1$. Στη συνέχεια, το μετατρέπουμε σε εμπιστοσύνη με χρήση της σχέσης

$$\text{trust} = 10^{-DP[u][j+1]}$$

και το συγκρίνουμε με το β_{j+1}. Αν δεν είναι αποδεκτό, συνεχίζουμε αναδρομικά μέχρι να βρούμε κάποιο αποδεκτό ή να μας τελειώσουν τα μονοπάτια.

1.3 Ορθότητα αλγορίθμου

Αρχικά, θα αποδείξουμε ότι το ελάχιστο μονοπάτι στον δεύτερο γράφο είναι αυτό που αντιστοιχεί στην μέγιστη εμπιστοσύνη στον πρώτο γράφο. Θα δουλέψουμε με άτοπο. Έστω ότι στον 1ο γράφο το ελάχιστο μονοπάτι αποτελείται από τις ακμές: $\{e_1, e_2, ..., e_t\}$. Αντί αυτού, διαλέγεται ένα μονοπάτι που αποτελείται από τις ακμές: $\{e_1, e_2, ..., e'_t\}$ (υποθέτουμε οτί διαφέρουν μόνο στην τελευταία ακμή χωρίς βλάβη της γενικότητας). Οι ακμές αυτές θα γίνουν στον 2ο γράφο:

$$\{-\log_{10} e_1, -\log_{10} e_2, ..., -\log_{10} e_t\}, \qquad \{-\log_{10} e_1, -\log_{10} e_2, ..., -\log_{10} e'_t\}$$

Το μονοπάτια θα έχουν κόστη:

$$w_1 = -\log_{10}(e_1 \cdot e_2 \cdot ... \cdot e_t), \qquad w_2 = -\log_{10}(e_1 \cdot e_2 \cdot ... \cdot e'_t),$$

Για να επιλέχθηκε το 2ο αντί του 1ου σημαίνει ότι:

$$w_2 > w_1 \iff$$

$$-\log_{10}(e_1 \cdot e_2 \cdot ... \cdot e'_t) > -\log_{10}(e_1 \cdot e_2 \cdot ... \cdot e_t) \iff$$

$$\log_{10}\left(\frac{1}{e_1 \cdot e_2 \cdot ... \cdot e'_t}\right) > \log_{10}\left(\frac{1}{e_1 \cdot e_2 \cdot ... \cdot e_t}\right) \iff$$

$$e_1 \cdot e_2 \cdot ... \cdot e_t < e_1 \cdot e_2 \cdot ... \cdot e'_t$$

το οποίο είναι άτοπο αφού αυτό εκφράζει την συνολική εμπιστοσύνη των μονοπατιών.

Άρα, αποδείξαμε ότι δεδομένης της ορθότητας του αλγορίθμου υπολογισμού των συντομότερων μονοπατιών, ο αλγόριθμος μας υπολογίζει τα σωστά συντομότερα μονοπάτια στο νέο γράφο.

Η ορθότητα του αλγορίθμου υπολογισμού των συντομότερων μονοπατιών προκύπτει άμεσα από την ορθότητα του αλγορίθμου Bellman ford, καθώς στην DP μορφή του, είναι ο αλγόριθμος Bellman ford για τις πρώτες k επαναλήψεις.

Τέλος, πρέπει να αποδείξουμε την διάταξη που υποθέτει ο αλγόριθμος μας. Συγκεκριμένα, μετά τον υπολογισμό όλων των βέλτιστων μονοπατιών που χρησιμοποιούν ως t ακμές $\forall t \in \{2, ..., k\}$, εξετάζουμε τα μονοπάτια από τις περισσότερες ακμές προς τις λιγότερες. Αυτό σημαίνει ότι υποθέτουμε πως

$$DP[i][t] \geq DP[i][t-1]$$

. Αυτό προκύπτει άμεσα από τη σχέση:

$$DP[v][j+1] = min(DP[v][j], ...), \qquad \forall v \in V$$

Από τα παραπάνω προκύπτει άμεσα η ορθότητα του αλγορίθμου που περιγράψαμε.

1.4 Υπολογιστική πολυπλοκότητα

Από την αναδρομικη σχέση που περιγράψαμε προκύπτει ότι χρειαζόμαστε $O(kn^2)$ για να υπολογίσουμε το DP αν χρησιμοποιούμε πίνακα γειτνίασης για την εύρεση των γειτόνων. Στη συνέχεια, πρέπει πιθανώς να ψάξουμε όλο τον πίνακα για να βρούμε πιθανούς φίλους και για κάθε στοιχείο του πίνακα πιθανώς χρειαζόμαστε backtracking στο DP μεχρι και k θέσεις. Έτσι, η πράξη αυτή έχει πολυπλοκότητα $O(k^2n)$. Έτσι, η συνολική υπολογιστική πολυπλοκότητα είναι:

$$O(kn^2 + k^2n)$$

2 Άσκηση 2

Ο αλγόριθμος του Dijkstra υπάρχει σε πολλές διαφορετικές παραλλαγές. Οι διαφοροποιήσεις υπάρχουν στο πως υλοποιείται η priority queue που χρησιμοποιείται για να αποθηκεύονται με προτεραιότητες οι κορυφές που δεν έχουν ακόμα πλήρως επεκταθεί, δηλαδή οι κορυφές που ανήκουν στο ανοικτό μέτωπο.

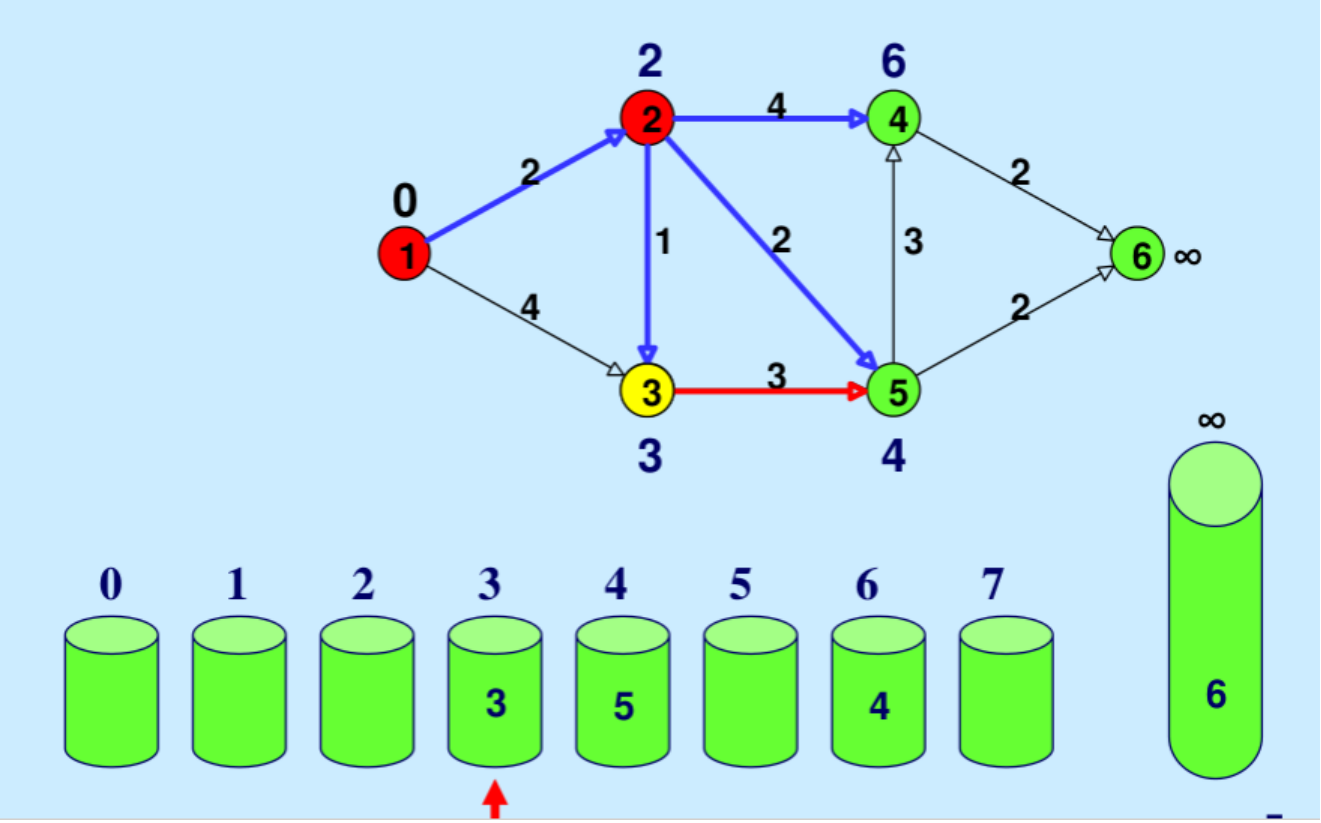
2.1 (α)

Στην άσκηση αυτή εκμεταλλευόμαστε το φράγμα που υπάρχει ως προς τις αξίες των κορυφών και υλοποιούμε την priotity queue με buckets. Συγκεκριμένα, αφού οι αξίες ανήκουν στο σύνολο: $\{0, 1, ..., C\}$ όσο τρέχουμε τον Dijkstra η μεγαλύτερη τιμή που μπορεί να πάρει ως εκτίμηση ελάχιστης απόστασης μια κορυφή είναι $(V-1) \cdot C$, δηλαδή η εκτίμηση ελάχιστης απόστασης να έχει προκύψει από ένα μονοπάτι V κορυφών που συνδέονται με ακμές βάρους C η κάθε μία με την επόμενη της.

Από αυτή την παρατήρηση, φτιάχνουμε $V \cdot C$ buckets που καθένα αντιπροσωπεύει μια εκτίμηση απόστασης: $0, 1, ..., (V-1) \cdot C$ αντίστοιχα.

Στον αλγόριθμο του Dijkstra διαλέγουμε κάθε φορά το πιο αριστερό μη άδειο bucket ως τρέχουσα κορυφή και την κάνουμε pop από το bucket. Στη συνέχεια, κάνουμε updates τους γείτονες πηγάινοντας στο bucket στο οποίο βρίσκονται, κάνοντας pop και τοποθετώντας τους στο κατάλληλο νέο bucket.

Ένα στιγμιότυπο από την εκτέλεση του αλγορίθμου θα ήταν το ακόλουθο:



Μπορούμε να υλοποιήσουμε τα buckets ως hash tables με keys τις αντίστοιχες κορυφές. Έτσι, το pop από το bucket και το insert σε νέο bucket, δηλαδή οι διαδικασίες που αφορούν τα updates των γειτόνων, είναι διαδικασίες που έχουν κόστος $O(1)$. Από την άλλη, για να βρούμε το πιο αριστερό άδειο bucket μπορεί να χρειαστεί να ελέγξουμε ενδιάμεσα buckets. Η μετακίνηση όμως είναι μονότονη από αριστερά προς τα δεξιά που σημαίνει ότι κάθε φορά δεν χρειάζεται να ελέγξουμε buckets που είχαμε ελέγξει στο παρελθόν. Έτσι, η συνολική πολυπλοκότητα που θα δώσουμε για τη μετακίνηση στα buckets είναι $O(V \cdot C)$.

Στην χειρότερη περίπτωση θα πρέπει να ψάξουμε όλα τα buckets και να κάνουμε ένα decrease key operation για κάθε ακμή. Για να περάσουμε από όλα τα buckets θέλουμε πολυπλοκότητα $O(V \cdot C)$ ενώ αφού το decrease key θέλει $O(1)$ άμα τα buckets υλοποιηθούν με hash tables, το να κάνουμε decrease key για όλες τις ακμές θέλει πολυπλοκότητα $O(E)$. Έτσι, η συνολική πολυπλοκότητα είναι $O(E + V \cdot C)$ ή με τους συμβολισμούς τις εκφώνησης:

$$O(m + nC)$$

2.2 (β)

Για να επιτύχουμε τη ζητούμενη πολυπλοκότητα πρέπει να περάσουμε από διάφορα λογικά βήματα, τα οποία εξετάζουμε αναλυτικά στη συνέχεια.

Αρχικά, δηλώνουμε ότι ως ουρά προτεραιότητας θα χρησιμοποιήσουμε ένα σωρό. Ακόμη, δηλώνουμε ότι στην ουρά προτεραιότητας δεν θα έχουμε κορυφές αλλά buckets από κορυφές καθένα από τα οποία αντιστοιχεί σε μία απόσταση που έχει αποδοθεί στις κορυφές όσο τρέχει ο Dijkstra.

Στη συνέχεια, θα αποδείξουμε ότι το πλήθος των κορυφών buckets που μπορούν να μπούν ταυτόχρονα μέσα στο heap δεν ξεπερνά το 2^C ή ισοδύναμα ότι δεν μπορούν να υπάρξουν συγχρόνως πάνω από 2^C διαφορετικές εκτιμήσεις για τις κορυφές όσο τρέχει ο Dijkstra.

Λήμμα 1. *Το πλήθος των διαφορετικών εκτιμήσεων που μπορεί να δώσει ανά πάσα στιγμή ο Dijkstra στις κορυφές του γράφου φράσσεται από το 2^C αν όλα τα βάρη των κορυφών φράσσονται από το 2^C.*

Απόδειξη.

Κάθε στιγμή στον αλγόριθμο του Dijkstra, $\exists a \in \mathcal{N}$ για τον οποίο όλες οι ακμές που απέχουν πάνω από a έχουν μπει στο κλειστό σύνολο. Οι κορυφές που περιέχονται στο priority Queue είναι αυτές που βρίσκονται σε απόσταση 1 ακμής από αυτές, καθώς αυτές είναι οι επόμενες που πρόκειται να γίνουν finalize. Άρα, οι αποστάσεις των κορυφών στο priority Queue είναι στο διάστημα $\{a+1, ..., a+2^C\}$. Οι διαφορετικές αποστάσεις στο διάστημα αυτό φράσσονται από το 2^C. □

Αφού δείξαμε ότι κάθε φορά υπάρχουν το πολύ 2^C διαφορετικές αποστάσεις στο priority queue, θα υπάρχουν και το πολύ 2^C διαφορετικά buckets που αντιστοιχούν σε αυτές τις αποστάσεις. Άρα, το priority queue θα έχει κάθε φορά το πολύ 2^C διαφορετικά στοιχεία.

Η αναζήτηση για το minimum σε σωρό έχει λογαριθμικό κόστος, άρα $O(log_2 2^C) = O(C)$. Επίσης, η πράξη του decrease key αντιστοιχεί στο να βρούμε το αντίστοιχο bucket να πάρουμε ένα στοιχείο του και αφού υπολογίσουμε την νέα τιμή του σε O(1) να βρούμε το νέο bucket και να το βάλουμε (αν το παλιό bucket άδειασε το πετάμε). Δηλαδή, αντιστοιχεί σε 2 αναζητήσεις και 1 delete στο σωρό που σημαίνει $O(C)$.

Η πολυπλοκότητα του Dijkstra όμως είναι:

$$O(m \cdot T_{dk} + n \cdot T_{em}), \quad T_{dk}: \text{ χρόνος decrease key}, \quad T_{em}: \text{ χρόνος extract minimum}$$

το οποίο όπως εξηγήσαμε γράφεται για την περίπτωση μας:

$$O((m+n)C)$$

3 Άσκηση 3

3.1 (α)

3.1.1 Διατύπωση αλγορίθμου

Πριν διατυπώσουμε τον αλγόριθμο μας, είναι χρήσιμο να διατυπώσουμε μια αναδρομική σχέση για το πρόβλημα που θα μας βοηθήσει να καταστρώσουμε το σχέδιο αντιμετώπισης του. Έχουμε ότι:

$$OPT(v, c) = min\big(OPT(u, c - c_{uv}) + w_{uv}, \quad \forall u : (u, v) \in E\big)$$

Αυτή η σχέση αποτελεί μια γενίκευση της έννοιας του συντομότερου μονοπατιού που αποθηκεύει και το κόστος για την εύρεση του. Αν θέσουμε όλα τα κόστη $c_{ij} = 0$ αυτό που παίρνουμε είναι η γνωστή μας σχέση με βάση την οποία κάνουμε τα updates στον Dijkstra. Αυτή λοιπόν η σχέση μας δίνει το πώς θα κάνουμε τα updates σε έναν γενικευμένο Dijkstra που δεν εκτελείται πάνω στις κορυφές του αρχικού γράφου αλλά πάνω σε states κορυφών.

Πιο συγκεκριμένα, θα θεωρήσουμε ότι ο Dijkstra διατηρεί δύο δομές, το ανοικτό μέτωπο και το κλειστό σύνολο. Το ανοικτό μέτωπο είναι ένα σύνολο με προτεραιότητες γενικευμένων κορυφών ως προς το μήκος, οι οποίες κορυφές δεν έχουν επεκταθεί πλήρως. Το κλειστό σύνολο είναι ένα σύνολο γενικευμένων κορυφών που έχουν επεκταθεί πλήρως. Με τον όρο γενικευμένες κορυφές εννοούμε κορυφές που κωδικοποιούν δύο πληροφορίες: το σε ποιά κορυφή του αρχικού γράφου αντιστοιχούν καθώς και το κόστος τους.

Κάθε φορά διαλέγουμε από το ανοικτό μέτωπο το ποιά κορυφή θα χρησιμοποιηθεί με βάση τη διάταξη που ορίζουν οι προτεραιότητες. Στη συνέχεια για αυτή την γενικευμένη κορυφή πραγματοποιούμε τα updates όπως ορίζει η αναδρομική σχέση. Συγκεκριμένα, βρίσκουμε τις γειτονικές της κορυφές στον αρχικό γράφο. Αν οι γενικευμένες κορυφές που προκύπτουν από τους γείτονες της βάζοντας το κόστος της κορυφής που εξετάζουμε + το κόστος της σύνδεσης, δεν υπάρχουν στο Q, τότε τις προσθέτουμε με τις κατάλληλες προτεραιότητες. Αν πάλι υπάρχουν στο Q τότε κάνουμε τα απαραίτητα updates.

Μόλις τελειώσει η εκτέλεση του Dijkstra κοιτάσουμε το κλειστό σύνολο και κρατάμε από αυτό μόνο τις γενικευμένες κορυφές που αντιστοιχούν στην κορυφή t. Διατάσσουμε ως προς τις προτεραιότητες τις κορυφές που απέμειναν. Εξετάζουμε μία μία αυτές τις γενικευμένες κορυφές από την μικρότερη προτεραιότητα (μικρότερο μήκος συντομότερου μονοπατιού) στη μεγαλύτερη. Η πρώτη από αυτές που θα βρούμε που δεν παραβιάζει το κριτήριο του κόστους είναι η λύση στο πρόβλημα μας.

3.1.2 Απόδειξη ορθότητας

Η ορθότητα του αλγορίθμου μας προκύπτει ως μια επέκταση της ορθότητας του αλγορίθμου του Dijkstra. Συγκεκριμένα, από τον τρόπο που κάνουμε τα updates γνωρίζουμε κάθε φορά ότι κάθε διαφορετικό κόστος που μπορεί να προκύψει από μονοπάτια κορυφών θα σχηματιστεί αφού οι κορυφές που αντιστοιχούν σε αυτό θα μπουν στο ανοικτό μέτωπο Q. Αφού κάθε φορά βάζουμε όλα τα προκύπτοντα κόστη που δεν υπάρχουν στο Q, ακόμα και αν αντιστοιχούν σε μεγαλύτερη προτεραιότητα, γνωρίζουμε ότι δεν θα χάσουμε κάποιο συνδυασμό κόστους και προτεραιότητας.

Πέρα από την πληρότητα των συνδυασμών που εξασφαλίζει ο αλγόριθμος μας, το γεγονός ότι οι διάφορες κορυφές γίνονται updated με τον τρόπο που ορίζει ο Dijkstra, διασφαλίζει ότι οι προκύπτοντες συνδυασμοί είναι βέλτιστοι, ότι δηλαδή για συγκεκριμένο κόστος αυτό είναι το συντομότερο μονοπάτι που μπορεί να επιτευχθεί.

Τέλος, ορθή είναι και η επιλογή του καλύτερου μονοπατιού από το κλειστό σύνολο, αφού ξεκινάμε από τα συντομότερα μονοπάτια με τα μεγάλα πιθανώς κόστη και όσο δεν τα καταφέρνουμε να ικανοποιήσουμε τους περιορισμούς καταφεύγουμε σε μεγαλύτερα μονοπάτια με μικρότερα κόστη.

3.1.3 Ανάλυση πολυπλοκότητας

Αφού η διάταξη είναι μονοδιάστατη ο χρόνος T_{em} παραμένει ίδιας πολυπλοκότητας με τον απλό Dijkstra που γνωρίζουμε ως προς το μέγεθος του ανοικτού μετώπου Q. Αν χρησιμοποιηθεί Fibonacci heap η πράξη αυτή αντιστοιχεί σε $(log|V|')$, όπου |V|' το πλήθος των κορυφών που θα μπούν στο ανοικτό σύνολο.

Ακόμη, και το decrease key γίνεται μόνο ως προς τη μία διάσταση, αυτή της προτεραιότητας, άρα διατηρεί την πολυπλοκότητα του με τον παραδοσιακό Dijkstra ως προς το μέγεθος του ανοικτού μετώπου Q. Σε Fibonacci heap η πράξη αυτή γίνεται σε O(1).

Πρακτικά, η υλοποίηση μας αντιστοιχεί σε έναν Dijkstra με augmented ακμές και κορυφές. Η πολυπλοκότητα του Dijkstra όπως ξέρουμε είναι:

$$O(|E|' \cdot T_{dk} + |V|' \cdot T_{em}), \quad T_{dk}: \text{ χρόνος decrease key}, \quad T_{em}: \text{ χρόνος extract minimum}$$

Με όσα παρατηρήσαμε γράφεται:

$$O(|E|' + |V|' log|V|')$$

Πρέπει λοιπόν να προσδιορίσουμε το πλήθος των ακμών και των κορυφών.

Αφού κάθε ακμή έχει κόστος το πολύ 1, δεν υπάρχει περίπτωση να ξεπεράσουμε το κόστος $|E|$ αφού κάθε μονοπάτι θα περιέχει το πολύ $|E|$ ακμές του αρχικού γράφου. Έτσι, οι κορυφές πολλαπλασιάζονται με έναν παράγοντα $|E|$, δηλαδή:

$$|V|' = |V| \cdot |E|$$

Ακόμη, και οι ακμές πολλαπλασιάζονται με τον ίδιο παράγοντα καθώς στον augmented γράφο η γειτνίαση διατηρείται. Άρα:

$$|E|' = |E| \cdot |E| = |E|^2$$

Άρα, η συνολική πολυπλοκότητα είναι:

$$O(|E|^2 + |V||E|log(|V||E|))$$

Η πολυπλοκότητα των υπολοίπων πράξεων παραλείπεται ως πολύ μικρή συγκρινόμενη με την προαναφερθείσα.

3.2 (β)

Στο ερώτημα αυτό δεν αλλάζουμε τον αλγόριθμο μας, συνεπώς αφού για την ορθότητα δεν κάναμε καμία υπόθεση για την φύση των βαρών, η ορθότητα μένει ανεπηρέαστη.

3.2.1 Ανάλυση πολυπλοκότητας

Αυτό που πραγματικά αλλάζει είναι η πολυπλοκότητα του αλγορίθμου μας. Ο λόγος είναι ότι ο αριθμός των κορυφών δεν φράσσεται από κάποιο εξαρχής γνωστό αριθμό για το πρόβλημα αλλά εξαρτάται κάθε φορά από τη φύση των βαρών. Συγκεκριμένα, το μόνο φράγμα που γνωρίζουμε για το πλήθος των κορυφών (δηλαδή για τα διαφορετικά πιθανά κόστη που μπορούμε να συναντήσουμε) είναι:

$$C = W \cdot |E|$$

όπου W το μεγαλύτερο κόστος στο πρόβλημα.

Με την ανάλυση αυτή, ο αλγόριθμος μας έχει πολυπλοκότητα:

$$O(|E| \cdot C + |V| \cdot C \log(|V| \cdot C))$$

Ο αλγόριθμος αυτός είναι **ψεύδοπολυωνυμικός**, αφού το C μπορεί να γίνει απείρως μεγάλο.

4 Άσκηση 4

4.1 Κατασκευή γράφου

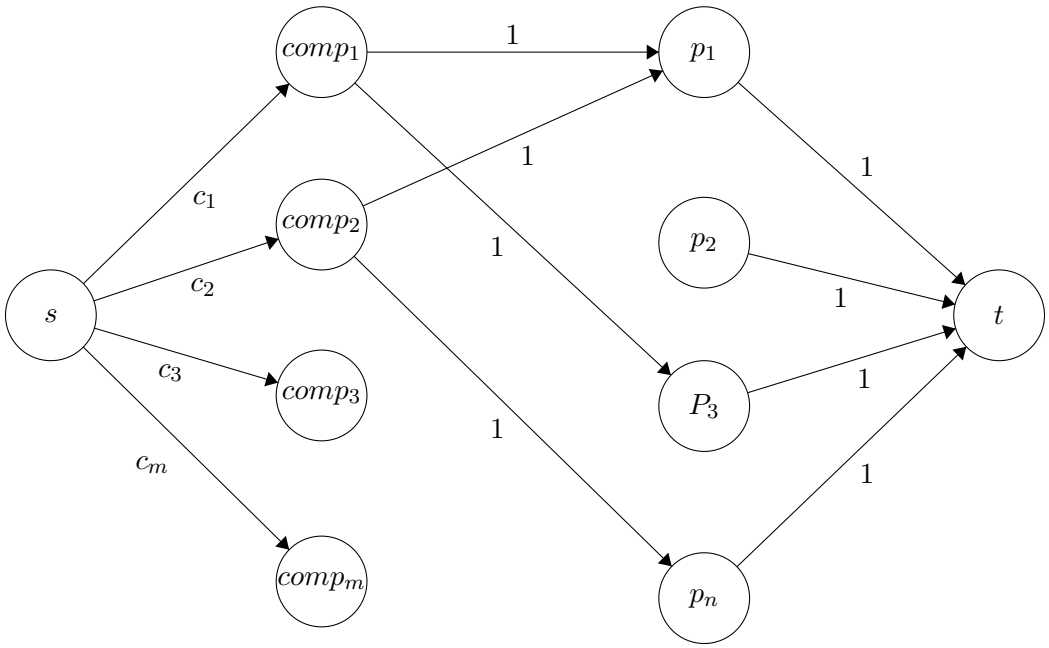
Η συγκεκριμένη άσκηση αφορά τη μοντελοποίηση του προβλήματος σε γράφο και την επίλυση του προβλήματος μέγιστης ροής σε αυτόν.

Το πρόβλημα που περιγράφεται στην εκφώνηση είναι ένα πρόβλημα ροής πολλών πηγών (διαφορετικές εταιρείες) με περιορισμό όμως στη δυνατότητα παραγωγής ροής από κάθε πηγή (χρήματα c_i). Για τη μοντελοποίηση του προβλήματος αυτού εισάγεται ένας νέος κόμβος s που μπορεί να στέλνει σε κάθε μία πηγή ξεχωριστά ροή μικρότερη ή ίση από το capacity της κάθε πηγής (εταιρείας).

Στο επόμενο στάδιο εισάγονται κόμβοι για κάθε άνθρωπο και οι εταιρείες συνδέονται με τους ανθρώπους που μπορούν να συνδεθούν, πληροφορία που δίνεται από τα σύνολα ενδιαφέροντος S_i. Αφού κάθε άνθρωπος πρέπει να αφαιρεί μια μονάδα από το c_i της εταιρείας i, το capacity όλων αυτών των συνδέσεων είναι 1.

Τέλος, όλοι οι άνθρωποι συνδέονται με έναν κόσμο προορισμού t με το capacity όλων των συνδέσεων να είναι στο 1, ώστε να ικανοποιηθεί η απαίτηση κάθε άνθρωπος να παίρνει το πολύ 1 διαφήμιση.

Όσα περιγράφηκαν φαίνονται σχηματικά στον ακόλουθο γράφο:



4.2 Διατύπωση αλγορίθμου

Αφου το πρόβλημα μοντελοποιηθεί σε γράφο με τον τρόπο που περιγράφηκε στην παραπάνω ενότητα, η επίλυση του ανάγεται στην επίλυση του προβλήματος της μέγιστης ροής. Συγκεκριμένα, χρησιμοποιείται κάποιος αλγόριθμος εύρεσης μέγιστης ροής σε γράφο, εδώ θα χρησιμοποιηθεί ο Edmonds-Karp και αν η ροή που βγαίνει από το s (ή ισοδύναμα η ροή που καταλήγει στο t) είναι n, σημαίνει ότι όλοι οι άνθρωποι είδαν από μια διαφήμιση.

Αν κάτι τέτοιο είναι εφικτό, τότε ο αλγόριθμος εύκολα αποφασίζει ποιός άνθρωπος είδε διαφήμιση ποιάς εταιρείας, βλέποντας ποιές από τις ακμές που συνδέουν εταιρείες με ανθρώπους έχουν ροή 1.

4.3 Διατύπωση ορθότητας

Η ορθότητα του αλγορίθμου Edmonds-Karp εγγυάται ότι στο γράφημα θα βρεθεί η μέγιστη ροή. Κάθε εταιρεία μπορεί να δώσει μόνο μέχρι c_i ροή λόγω των συνδέσεων με την πηγή, οπότε καλύπτεται η πρώτη προδιαγραφή. Αυτή η ροή μπορεί να διοχετευθεί μόνο σε ανθρώπους του συνόλου ενδιαφέροντος της καθώς έτσι κατασκευάστηκε ο γράφος, οπότε καλύπτεται και η 2η προδιαγραφή. Ένας άνθρωπος δεν μπορεί να δει παραπάνω από μία διαφημίσεις αφού το capacity των εισερχόμενων συνδέσεων στους ανθρώπους είναι 1, οπότε καλύπτεται και η 3η προδιαγραφή. Η απόφαση επίλυσης του αρχικού προβλήματος από την επίλυση του προβλήματος μέγιστης ροής σε γράφο παίρνεται από την συνολική ροή στο t, δηλαδή από το πόσοι άνθρωποι πήραν ροή (αφού κάθε άνθρωπος δίνει ως 1 μονάδα ροής στο t), συνεπώς το κριτήριο είναι σύμφωνο με όσα ζητήθηκαν στην εκφώνηση.

Από τα παραπάνω προκύπτει η ορθότητα του αλγορίθμου.

4.4 Χρονική πολυπλοκότητα

Η χρονική πολυπλοκότητα είναι το max της πολυπλοκότητας κατασκευής του γράφου και της πολυπλοκότητας του αλγορίθμου Edmonds-Karp για την εύρεση της μέγιστης ροής σε αυτόν και του backtracking για την αντιστοίχηση ανθρώπων σε εταιρείες.

4.4.1 Πολυπλοκότητα κατασκευής γράφου

Ο γράφος αναπαρίσταται με λίστα γειτνίασης. Άρα, η πολυπλοκότητα κατασκευής του είναι η πολυπλοκότητα της προσθήκης ακμών. Έχουμε συνολικά m εταιρείες που κάθε μία συνδέεται δυνητικά με n ανθρώπους. Άρα, από εδώ προκύπτουν $n \cdot m$ ακμές. Ακόμη, υπάρχουν άλλες n ακμές για τη σύνδεση των ανθρώπων με τον προορισμό και άλλες m ακμές για τη σύνδεση της πηγής με τις εταιρείες. Άρα συνολικά, έχουμε $\Theta(n \cdot m)$ ακμές, και αντίστοιχα ίδια πολυπλοκότητα για την κατασκευή του γράφου.

Σημείωση: Συνολικά για τον γράφο έχουμε:

$$|V| = \Theta(n + m)$$

$$|E| = \Theta(n \cdot m)$$

4.4.2 Πολυπλοκότητα Edmonds-Karp

Ο αλγόριθμος Edmonds-Karp έχει πολυπλοκότητα:

$$O(|V| \cdot |E|^2) = O(n^3 m^3)$$

4.4.3 Πολυπλοκότητα Backtracking

Για το backtracking, εξετάζονται απλώς οι ακμές οπότε έχουμε πολυπλοκότητα:

$$O(|E|) = O(n \cdot m)$$

4.4.4 Συνολική πολυπλοκότητα

Από τα παραπάνω προκύπτει ότι η συνολική πολυπλοκότητα είναι:

$$O(|V| \cdot |E|^2) = O(n^3 m^3)$$

5 Άσκηση 5

Έστω ότι ζητείται να αποδειχθεί ότι ένα πρόβλημα $A \in \mathbf{NPC}$. Αν είναι γνωστό ότι ένα πρόβλημα $B \in \mathbf{NPC}$ τότε η εργασία οργανώνεται ως εξής:

1. Γίνεται η υπόθεση ότι το πρόβλημα Α μπορεί να λυθεί "εύκολα" (σε πολυωνυμικό χρόνο)
2. Το πρόβλημα Β μετασχηματίζεται με κατάλληλο τρόπο σε ένα στιγμιότυπο εισόδου για το πρόβλημα Α που άμα λυθεί, δίνει την λύση για το Β
3. Αφού το Α λύνεται γρήγορα (πολυωνυμικά), τότε και το Β λύνεται γρήγορα λόγω του 2.
4. Κάτι τέτοιο δεν ισχύει, αφού $B \in \mathbf{NPC}$ άρα το Α είναι τουλάχιστον το ίδιο δύσκολο με το Β.
5. Αν δειχθεί ότι η λύση του Α επαληθεύεται σε πολυωνυμικό χρόνο τότε $A \in \mathbf{NPC}$.

5.1 3-Διαμέριση

Έστω ότι είναι δυνατό να λύσουμε σε πολυωνυμικό χρόνο το πρόβλημα της 3 διαμέρισης. Θα δείξουμε ότι τότε θα είναι δυνατό να λύσουμε σε πολυωνυμικό χρόνο το πρόβλημα της 2-διαμέρισης, το οποίο γνωρίζουμε ότι είναι **NPC** για να οδηγηθούμε σε άτοπο.

Έστω ότι μας δίνεται ένα σύνολο Α και θέλουμε να δούμε αν υπάρχει 2-διαμέριση για το σύνολο αυτό. Βρίσκουμε το άθροισμα του συνόλου αυτού, έστω s. Έτσι, θέλουμε εμείς να διαμερίσουμε το σύνολο σε δύο υποσύνολα αθροίσματος $\frac{s}{2}$.

Προσθέτουμε στο σύνολο Α έναν ακόμα αριθμό, φτιάχνοντας το σύνολο Α', τον αριθμό $\frac{s}{2}$. Έτσι, το σύνολο Α' θα έχει άθροισμα:

$$s' = s + \frac{s}{2} = \frac{3s}{2}$$

Δίνουμε το Α' για επίλυση στο πρόβλημα της 3-Διαμέρισης. Αναμένουμε να ελέγξει αν μπορεί να σπάσει σε τρία υποσύνολα αθροίσματος $\frac{s}{2}$ το καθένα. Αφού το κάθε υποσύνολο θα πρέπει να έχει άθροισμα $\frac{s}{2}$, το στοιχείο που προσθέσαμε θα πρέπει να μπει σε ένα υποσύνολο μόνο του. Η ένωση των άλλων δύο ξένων υποσυνόλων θα μας δίνει το Α και καθένα από αυτά θα έχει άθροισμα $\frac{s}{2}$. Άρα, βρήκαμε τρόπο να σπάσουμε σε πολυωνυμικό χρόνο το Α σε δύο ξένα υποσύνολα με άθροισμα $\frac{s}{2}$ το καθένα. Αφού το πρόβλημα αυτό είναι **NPC** οδηγούμαστε σε άτοπο.

Άρα, το πρόβλημα της 3-Διαμέρισης είναι τουλάχιστον όσο δύσκολο όσο το πρόβλημα της 2-Διαμέρισης. Όμως μια 3-Διαμέριση ελέγχεται σε χρόνο $O(n)$, δηλαδή πολυωνυμικό, άρα το πρόβλημα είναι **NPC**.

5.2 Άθροισμα υποσυνόλου κατά προσέγγιση

5.3 Κύκλος Hamilton κατά Προσέγγιση

Έστω ότι είναι δυνατό να βρεθεί πολυωνυμικά αν υπάρχει κύκλος Hamilton κατά προσέγγιση. Έχουμε ένα γράφο, έστω $G(V, E)$, για τον οποίο στόχος είναι να βρούμε αν υπάρχει κύκλος Hamilton. Μετασχηματίζουμε τον γράφο $G(V, E)$ στον γράφο $G'(V', E')$ ώστε να διατηρηθούν όλες οι κορυφές και όλες οι ακμές του γράφου G και να προστεθεί για κάθε κορυφή του γράφου G μια κορυφή και μια ακμή που συνδέει την νέα κορυφή με την παλιά.

Με τον μετασχηματισμό αυτό, λύνουμε το πρόβλημα του κύκλου Hamilton κατά προσέγγιση για τον νέο γράφο που σχηματίστηκε. Αν το πρόβλημα λύνεται, γνωρίζουμε ότι από τη λύση αφαιρόντας τις extra ακμές και τις κορυφές που προστέθηκαν θα πάρουμε κύκλο Hamilton αφού "αναγκαστικά" στον κύκλο Hamilton κατά προσέγγιση για κάθε κορυφή του γράφου G η μια φορά θα σπαταληθεί για να περάσει ο κύκλος από τις extra κορυφές καθώς ο μόνος τρόπος να περάσει είναι μέσα από τις παλιές κορυφές που τους αντιστοιχούν.

Αντίστοιχα, αν το πρόβλημα δεν λύνεται τότε γνωρίζουμε ότι δεν υπάρχει κύκλος Hamilton στον αρχικό γράφο καθώς η ελαστικότητα του προβλήματος του κύκλου Hamilton κατά προσέγγιση ώστε να περνάμε από κάθε κορυφή μέχρι και 2 φορές μας επιτρέπει να βρούμε κύκλο Hamilton αν υπάρχει.

Δείξαμε ότι το πρόβλημα του κύκλου Hamilton κατά προσέγγιση είναι τουλάχιστον το ίδιο δύσκολο με το πρόβλημα του κύκλου Hamilton. Όμως μια λύση του επαληθέυεται σε πολυωνυμικό χρόνο, άρα ανήκει στην NP κλάση και αφού είναι το τουλάχιστον το ίδιο δύσκολο με ένα NP complete τότε θα είναι και αυτό NP Complete.

5.4 Ικανοποιησιμότητα με περιορισμούς

Αρχικά θα αποδείξουμε ότι αν μπορούμε να λύσουμε το πρόβλημα 4-SAT, τότε μπορούμε να λύσουμε και το πρόβλημα 3-SAT.

Έστω ότι μας δίνεται ένα στιγμιότυπο εκφώνησης ενός 3-SAT προβλήματος της μορφής:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge ...(x_k \vee x_{k+1} \vee x_{k+2})$$

Σημειώνουμε ότι x_i, x_j πιθανώς ταυτίζονται ή το ένα είναι άρνηση του άλλου. Σε κάθε όρο συμπληρώνουμε ένα $\vee\neg x_i$ όπου το x_i να είναι ο πρώτος επιμέρους όρος κάθε ευρύτερου όρου που συμμετέχει στην τομή. Έστω $\neg x_p, x_{p+1}, ..., x_{p+k}$ οι επιμέρους όροι των οποίων προσθέσαμε την άρνηση. Προσθέτουμε ακόμη τους εξής όρους:

$$(\neg x_p \vee x_{p+1} \vee x_{p+2} \vee x_{p+3}), \quad (\neg x_{p+1} \vee x_p \vee x_{p+2} \vee x_{p+3}), ...$$

για κάθε τετράδα. Αν δεν υπάρχουν διαθέσιμοι όροι για να συμπληρωθεί τετράδα τους φτιάχνουμε ως εξής:

$$x_1 \vee \neg x_1 \vee x_1 \vee x_1$$

Δίνουμε το μετασχηματισμένο στιγμιότυπο για επίλυση προς το πρόβλημα 4-SAT με περιορισμούς. Ο μόνος τρόπος να λύνεται το δεύτερο και να μην λύνεται το πρώτο είναι κάθε όρος του δεύτερου να είναι αληθής. Με τον τρόπο που διαλέξαμε να σχηματίσουμε τους όρους του στιγμιοτύπου κάτι τέτοιο δεν συμβαίνει. Επίσης, αποτρέπουμε την περίπτωση να λυθεί το δεύτερο και να μη λύνεται το πρώτο καθώς οι περιορισμοί του προβλήματος θα αποτύγχουν σε μια τέτοια εκδοχή στους τελευταίους όρους που προσθέσαμε.

Άρα, το 4-SAT με περιορισμούς πρόβλημα αν λύνεται μας δίνει λύση του 3-SAT που ξέρουμε ότι είναι **NPC**, δηλαδή το 4-SAT με περιορισμούς πρόβλημα είναι τουλάχιστον το ίδιο δύσκολο με το 3-SAT. Τα δύο προβλήματα είναι όμως ακριβώς ίδιας δυσκολίας αφού και για το $4-SAT$ με περιορισμούς μια λύση του επαληθεύεται σε πολυωνυμικό χρόνο και άρα ανήκει στην κλάση NP.

5.5 Επιλογή ανεξάρτητων υποσυνόλων

Θα δείξουμε ότι άμα μπορούμε να σπάσουμε σε k ξένα υποσύνολα τότε μπορούμε να λύσουμε και το 3DM πρόβλημα που γνωρίζουμε ότι είναι **NPC**.

Έστω ένας τριμερής γράφος που αποτελείται από τα ξένα σύνολα X, Y, Z για τον οποίο επιχειρούμε να βρούμε το μεγαλύτερο $3DM$. Παίρνουμε m ανεξάρτητα υποσύνολα πλειάδων από το σύνολο των πλειάδων του τριμερούς γραφήματος. Λύνουμε το πρόβλημα της επιλογής ανεξάρτητων υποσυνόλων για $k = |X| = |Y| = |Z|$ και έτσι παίρνουμε το τρισδιάστατο perfect matching.

Δείξαμε ότι η επιλογή ανεξάρτητων υποσυνόλων είναι τουλάχιστον το ίδιο δύσκολη με το $3DM$. Όμως, μια λύση του προβλήματος αυτού επαληθεύεται σε πολυωνυμικό χρόνο άρα ανήκει στην κλάση NP και συνεπώς και τα δύο προβλήματα είναι **NPC**.

Αναφορές

[1] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[2] Shigang Chen and Klara Nahrstedt. On finding multi-constrained paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on*, volume 2, pages 874–879. IEEE, 1998.

[3] Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Etats-Unis Informaticien, and Edsger Wybe Dijkstra. *A discipline of programming*, volume 1. prentice-hall Englewood Cliffs, 1976.

[4] Ronald A Howard. Dynamic programming. *Management Science*, 12(5):317–348, 1966.

[5] Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.