

Προσεγγιστικός υπολογισμός διαμέτρου σε κατευθυνόμενους γράφους

Ευθυμιάδης Χρήστος

Διπλωματική Εργασία

Επιβλέπων: Γεωργιάδης Λουκάς

Ιωάννινα, Ιούνιος, 2025



**ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**

**DEPARTMENT OF COMPUTER SCIENCE &
ENGINEERING
UNIVERSITY OF IOANNINA**

Περίληψη

Οι γράφοι χρησιμοποιούνται πολύ συχνά για την μοντελοποίηση πραγματικών προβλημάτων και καταστάσεων. Η θεωρία γράφων μπορεί να βρει πάρα πολλές εφαρμογές στην Πληροφορική, στη Μηχανική, στη Χημεία, στην Κοινωνιολογία κλπ. Τα τελευταία χρόνια μάλιστα, με την εξάπλωση των μέσων Κοινωνικής Δικτύωσης (Social Media Networks), οι γράφοι χρησιμοποιούνται συχνά για την αναπαράσταση των χρηστών και των σχέσεων που έχουν αυτοί μεταξύ τους.

Είναι σαφές επομένως από τα πιο πάνω ότι είναι σημαντικό να γνωρίζουμε τη διάμετρο ενός γράφου, αφού αποτελεί μία από τις βασικότερες μετρικές του και μπορεί να δώσει χρήσιμες πληροφορίες για τη δομή ενός δικτύου. Ο ακριβής υπολογισμός της διαμέτρου για γράφους μεγάλης κλίμακας απαιτεί πάρα πολύ χρόνο, με αποτέλεσμα να έχει πρακτικό ενδιαφέρον ο γρήγορος υπολογισμός μιας προσέγγισης της διαμέτρου.

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι να παρουσιάσει και να υλοποιήσει έναν 7/4-προσεγγιστικό αλγόριθμο υπολογισμού της διαμέτρου ενός κατευθυνόμενου γράφου [Abboud et al., 2023]. Η υλοποίηση του αλγορίθμου έχει γίνει σε γλώσσα προγραμματισμού Python, έτσι ώστε να μπορεί να εκτελεστεί εύκολα σε οποιοδήποτε υπολογιστικό σύστημα. Η απόδοση του αλγορίθμου είναι καλύτερη για την περίπτωση αραιών γράφων και είναι της τάξης του $n^{1.5-\varepsilon}$ με παράγοντα προσέγγισης μικρότερο από 2.

Λέξεις Κλειδιά: Διάμετρος γράφου, προσεγγιστικός αλγόριθμος υπολογισμού διαμέτρου

Abstract

Graphs are frequently used to model real-world problems and situations. Graph theory can find numerous applications in Computer Science, Engineering, Chemistry, Sociology, etc. In recent years, with the proliferation of Social Media Networks, graphs are often used to represent users and the relationships between them.

It is therefore clear from the above that it is important to know the diameter of a graph, as it is one of its most fundamental metrics and can provide useful information about the geometry of a network. Calculating the diameter for graphs corresponding to massive data volumes is not at all easy, and its exact computation requires a great deal of time.

The purpose of this thesis is to present and implement a $7/4$ -approximation algorithm for computing the diameter of a directed graph [Abboud et al., 2023]. The algorithm has been implemented in the Python programming language, so that it can be easily executed on any computing system. The algorithm performs better in the case of sparse graphs and has a time complexity of the order of $n^{1.5-\epsilon}$, with an approximation factor of less than 2.

Keywords: Graph diameter, diameter approximation algorithm

Περιεχόμενα

1. Εισαγωγή	8
1.1 Οι γράφοι για την αναπαράσταση δεδομένων	8
1.2 Διάμετρος γράφου	10
1.3 Σκοπός και στόχοι της εργασίας	12
2. Υπολογισμός διαμέτρου ενός γράφου	13
2.1 Βασικές έννοιες από τη θεωρία Γράφων	13
2.1.1 Αναπαράσταση γράφου	15
2.2 Χαρακτηριστικά και μετρικές γράφων	19
2.2.1 Εκκεντρότητα και Διάμετρος	19
2.3 Αλγόριθμοι υπολογισμού της διαμέτρου ενός γράφου.....	22
2.3.1 Κατά Πλάτος Αναζήτηση σε γράφο χωρίς βάρη (Breadth First Search – BFS)	23
2.3.2 Αλγόριθμος Dijkstra	28
3. Προσεγγιστικός αλγόριθμος υπολογισμού διαμέτρου.....	31
3.1 Θεωρητικό υπόβαθρο	31
3.2 Παρουσίαση του προβλήματος και σχετική έρευνα.....	33
3.3 Προσεγγιστικός αλγόριθμος υπολογισμού διαμέτρου.....	36
3.3.1 Συμβολισμοί και προαπαιτούμενα.....	36
3.3.2 Περιγραφή του αλγορίθμου	37
3.3.3 Διαδικασία κατασκευής γράφου μέγιστου βαθμού 3	41
4. Υλοποίηση του προσεγγιστικού αλγορίθμου	44
4.1 Τρόπος υλοποίησης	44
4.2 Περιβάλλον ανάπτυξης.....	46
4.3 Βιβλιοθήκες της Python.....	49
4.4 Παρουσίαση της εφαρμογής.....	50
4.5 Δοκιμή της εφαρμογής.....	58
4.6 Πειραματική μελέτη	61
5. Επίλογος και συμπεράσματα	63
6. Βιβλιογραφικές Αναφορές.....	65
Παράρτημα	68

Ευρετήριο Σχημάτων

Σχήμα 1: Κατευθυνόμενος γράφος $G=(V,E)$	16
Σχήμα 2: Πίνακας γειτνίασης κατευθυνόμενου γράφου.....	16
Σχήμα 3: Αναπαράσταση γράφου με λίστα γειτνίασης	17
Σχήμα 4: Σταθμισμένος γράφος.....	18
Σχήμα 5: Πίνακας γειτνίασης σταθμισμένου γράφου.....	18
Σχήμα 6: Έσω και έξω-εκκεντρότητες κορυφών κατευθυνόμενου γράφου	22
Σχήμα 7: Αναζήτηση κατά Πλάτος (BFS) ξεκινώντας από την κορυφή a.....	25
Σχήμα 8: Αναζήτηση κατά Πλάτος (BFS) ξεκινώντας από την κορυφή c.....	26
Σχήμα 9: Γράφος ίδιας διαμέτρου με μέγιστο βαθμό κορυφών ίσο με 3	42
Σχήμα 10: Πίνακας γειτνίασης νέου γράφου G'	43

Ευρετήριο Εικόνων

Εικόνα 1: Κώδικας Python για την αναζήτηση κατά Πλάτος.....	27
Εικόνα 2: Κώδικας Python για τον αλγόριθμο Dijkstra	30
Εικόνα 3: Λίστα με τις 10 πιο δημοφιλείς γλώσσες προγραμματισμού (TIOBE).....	46
Εικόνα 4: Λίστα με τα πιο δημοφιλή περιβάλλοντα προγραμματισμού (PYPL)	47
Εικόνα 5: Το περιβάλλον του PyCharm	48
Εικόνα 6: Δημιουργία γράφου ίδιας διαμέτρου με μέγιστο βαθμό κορυφών 3	51
Εικόνα 7: Κώδικας για τον υπολογισμό έσω και έξω-εκκεντρότητας.....	52
Εικόνα 8: Δημιουργία συνόλου B^{out}	53
Εικόνα 9: Δημιουργία τυχαίου δείγματος κορυφών	53
Εικόνα 10: Δημιουργία συνόλου B^{out+}	54
Εικόνα 11: Υπολογισμός συνόλου B^{in}	55
Εικόνα 12: Υπολογισμός συνόλου B^{in+}	55
Εικόνα 13: Κώδικας για τα σύνολα A^{in} και A^{out}	57
Εικόνα 14: Δυαδική αναζήτησης κατά την εκτέλεση του αλγορίθμου A_D	58
Εικόνα 15: Κώδικας για τη δημιουργία τυχαίου κατευθυνόμενου γράφου.....	59
Εικόνα 16: Παράδειγμα εκτέλεσης προγράμματος για γράφο με 10 κορυφές	60
Εικόνα 17: Παράδειγμα εκτέλεσης προγράμματος για γράφο με $n=20$ κορυφές	61

1. Εισαγωγή

1.1 Οι γράφοι για την αναπαράσταση δεδομένων

Ένας γράφος ή γράφημα αναφέρεται στα διακριτά μαθηματικά ότι αποτελεί μία αφηρημένη αναπαράσταση ενός συνόλου δεδομένων-στοιχείων. Κάποια από αυτά τα ζεύγη στοιχείων συνδέονται μεταξύ τους με κάποιον τρόπο, ο οποίος υποδηλώνει κάποιου είδους εξάρτηση μεταξύ τους. Τα στοιχεία που διασυνδέονται ονομάζονται κορυφές ή κόμβοι ενώ οι δεσμοί που υπάρχουν μεταξύ τους ονομάζονται ακμές. Οι ακμές μπορεί να είναι κατευθυνόμενες (ασύμμετρες) ή μη κατευθυνόμενες (συμμετρικές).

Οι γράφοι χρησιμοποιούνται πολύ συχνά για την μοντελοποίηση πραγματικών προβλημάτων και καταστάσεων. Η θεωρία γράφων μπορεί να βρει πάρα πολλές εφαρμογές στην Πληροφορική, στη Μηχανική, στη Χημεία, στην Κοινωνιολογία κλπ. Τα τελευταία χρόνια μάλιστα, με την εξάπλωση των μέσων Κοινωνικής Δικτύωσης (Social Media Networks) οι γράφοι χρησιμοποιούνται συχνά για την αναπαράσταση των χρηστών και των σχέσεων που έχουν αυτοί μεταξύ τους. Με την μελέτη των γράφων αυτών μπορούν να εξαχθούν χρήσιμα συμπεράσματα, όπως για παράδειγμα η εύρεση των χρηστών που επηρεάζουν πολλούς άλλους χρήστες, οι συνήθειες των χρηστών κτλ. Η ανάλυση των διαφόρων χαρακτηριστικών επομένως ενός γράφου είναι συχνά πολύ χρήσιμη και έχουν αναπτυχθεί διάφοροι αλγόριθμοι για την ανακάλυψη των χαρακτηριστικών αυτών. Τα χαρακτηριστικά αυτά αφορούν τις αποστάσεις ανάμεσα στις κορυφές του γράφου και τις μετρικές που αφορούν τη δομή ενός γράφου, όπως είναι η εκκεντρότητα των κόμβων, η διάμετρος και η ακτίνα του γράφου κτλ.

Η αναπαράσταση κάποιων δεδομένων σε μορφή γράφου είναι συχνά πολύ χρήσιμη και έχει πολλές εφαρμογές, επειδή ένας γράφος δίνει τη δυνατότητα να αποδοθούν

κάποιες σχέσεις και διασυνδέσεις με φυσικό και ευέλικτο τρόπο. Κάποιες από τις βασικές χρήσεις της αναπαράστασης δεδομένων σε μορφή γράφου παρουσιάζονται πιο κάτω:

- Αναπαράσταση σχέσης ανάμεσα σε οντότητες (entities). Ένας γράφος είναι μία ιδανική δομή για να αναπαραστήσει με φυσικό τρόπο ποιες οντότητες συνδέονται με κάποιον τρόπο μεταξύ τους. Αυτό μπορεί να έχει διάφορες εφαρμογές σε χώρους όπως τα Κοινωνικά Δίκτυα (Social Media), δημιουργία γενεαλογικών δέντρων, αναπαράσταση δικτύων συνεργασίας ανάμεσα σε επιχειρήσεις κλπ.
- Εύρεση συντομότερης διαδρομής. Σε πολλές περιπτώσεις είναι πολύ χρήσιμη η δυνατότητα εύρεσης της συντομότερης διαδρομής ανάμεσα σε 2 κόμβους. Η αναπαράσταση των δεδομένων σε μορφή γράφου βοηθά πάρα πολύ, αφού είναι δυνατή στη συνέχεια η χρήση ενός αλγορίθμου εύρεσης συντομότερης διαδρομής. Μία από τις πολύ σημαντικές εφαρμογές είναι στην περίπτωση της δρομολόγησης πακέτων στο Διαδίκτυο, όπου εφαρμόζονται αλγόριθμοι εύρεσης συντομότερης διαδρομής, όπως ο αλγόριθμος Dijkstra και ο αλγόριθμος Bellman-Ford για πρωτόκολλα όπως το OSPF και το RIP. Υπάρχουν φυσικά και πάρα πολλές άλλες εφαρμογές όπως στην παράδοση πακέτων, τροφοδοτική αλυσίδα (Logistics), πλοήγηση κλπ.
- Μοντελοποίηση πολύπλοκων συστημάτων. Υπάρχουν πολύπλοκα συστήματα που αφορούν μοντέλα από τομείς όπως η Βιολογία, η Οικονομία κλπ. τα οποία μπορούν να αναπαρασταθούν με φυσικό τρόπο με τη μορφή γράφου. Η μοντελοποίηση της επικοινωνίας σε συστήματα Κοινωνικής Δικτύωσης μπορεί επίσης να αναπαρασταθεί με τη μορφή γράφων.
- Σημασιολογικά Δίκτυα. Ο σημασιολογικός Ιστός περιλαμβάνει τεχνολογίες που επιτρέπουν στα δεδομένα του Web να έχουν κάποια «σημασία» και να μπορούν να ερμηνευτούν από υπολογιστές. Η αναπαράσταση των στοιχείων γίνεται με τη μορφή γράφων, όπου οι πληροφορίες συνδέονται μεταξύ τους με κάποιες σχέσεις.
- Συσταδοποίηση (clustering). Με τη συσταδοποίηση γίνεται χωρισμός ενός

συνόλου δεδομένων σε ομάδες (συστάδες) με τέτοιον τρόπο ώστε τα αντικείμενα που μοιάζουν μεταξύ τους να βρίσκονται στην ίδια ομάδα. Με την αναπαράσταση δεδομένων σε μορφή γράφων, μπορούν να εφαρμοστούν αλγόριθμοι συσταδοποίησης, όπως και άλλοι αλγόριθμοι της Τεχνητής Νοημοσύνης.

1.2 Διάμετρος γράφου

Η διάμετρος ενός γράφου αποτελεί μία από τις βασικές μετρικές ενός γράφου, όπως περιγράφεται αναλυτικότερα στο επόμενο κεφάλαιο. Η διάμετρος ενός γράφου αντιστοιχεί στη μεγαλύτερη απόσταση (μήκος συντομότερης διαδρομής) μεταξύ οποιωνδήποτε 2 κορυφών του γράφου. Με άλλα λόγια, από όλες τις συντομότερες αποστάσεις ανάμεσα σε ζεύγη κορυφών του γράφου, η διάμετρος είναι η μέγιστη από αυτές τις συντομότερες αποστάσεις. Η διάμετρος αποτελεί δηλαδή μία σημαντική μετρική ενός γράφου γιατί εκφράζει το μέγεθος του γράφου σε συνδυασμό με τη συνδεσιμότητά του και τη «συνοχή» του. Περιγράφει με άλλα λόγια, πόσο μακριά μπορεί να βρίσκονται μεταξύ τους 2 κορυφές στην χειρότερη περίπτωση.

Η εύρεση της διαμέτρου ενός γράφου είναι πολλές φορές χρήσιμη και μπορεί να έχει διάφορες εφαρμογές, όπως:

- Μέτρηση αποδοτικότητας δικτύου υπολογιστών. Η διάμετρος του δικτύου δίνει ένα μέτρο για το πόσα βήματα ανάμεσα σε δρομολογητές θα πρέπει να κάνουν τα πακέτα για να φτάσουν από έναν υπολογιστή σε έναν άλλον στο χειρότερο σενάριο. Η διάμετρος επομένως βοηθά στη μέτρηση της αποδοτικότητας του δικτύου.
- Κοινωνικά Δίκτυα (Social Media). Με τον υπολογισμό της διαμέτρου είναι δυνατή η μέτρηση της «συνοχής» και της «πυκνότητας» ενός δικτύου.
- Μέτρηση της απόδοσης αλγορίθμων. Υπάρχουν αλγόριθμοι οι οποίοι δεν είναι καθόλου αποδοτικοί σε γράφους που έχουν μεγάλη διάμετρο.

Επομένως, είναι χρήσιμο σε τέτοιες περιπτώσεις να γνωρίζουμε τη διάμετρο ενός δικτύου, έτσι ώστε να επιλέξουμε τον αλγόριθμο που θα πρέπει να χρησιμοποιήσουμε.

- Ανάλυση της δομής ενός γράφου. Ένας γράφος μπορεί να χρησιμοποιηθεί για την αναπαράσταση πολύπλοκων μοντέλων και η γνώση της διαμέτρου δίνει σημαντικές πληροφορίες για τη γεωμετρία του γράφου και τη διασυνδεσιμότητα ανάμεσα στους κόμβους.

Είναι σαφές επομένως από τα πιο πάνω ότι είναι σημαντικό να γνωρίζουμε τη διάμετρο ενός γράφου, αφού αποτελεί μία από τις βασικότερες μετρικές του και μπορεί να δώσει χρήσιμες πληροφορίες για τη δομή ενός δικτύου. Ο υπολογισμός της διαμέτρου για γράφους που αντιστοιχούν σε τεράστιο όγκο δεδομένων δεν είναι καθόλου εύκολος και ο ακριβής υπολογισμός της απαιτεί πάρα πολύ χρόνο. Υπάρχουν φυσικά αλγόριθμοι που μπορούν να εφαρμοστούν, όπως είναι η κατά Πλάτος αναζήτηση, ο αλγόριθμος του Dijkstra κλπ., όμως η πολυπλοκότητα χρόνου για τους αλγορίθμους αυτούς είναι απαγορευτική όταν το μέγεθος του γράφου είναι πολύ μεγάλο. Για τον λόγο αυτόν, όπως θα δούμε στη συνέχεια, έχουν αναπτυχθεί προσεγγιστικοί αλγόριθμοι για τον υπολογισμό της διαμέτρου ενός γράφου.

Η σημαντικότητα του υπολογισμού της διαμέτρου ενός γράφου και οι δυσκολίες που παρουσιάζει, όπως και η πληθώρα των εφαρμογών που έχει η θεωρία γράφων σε πάρα πολλούς τομείς, αποτέλεσαν το σημαντικότερο κίνητρο για τη δημιουργία της παρούσας εργασίας.

1.3 Σκοπός και στόχοι της εργασίας

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι να παρουσιάσει και να υλοποιήσει έναν προσεγγιστικό αλγόριθμο υπολογισμού της διαμέτρου ενός κατευθυνόμενου γράφου. Η υλοποίηση του αλγορίθμου έχει γίνει σε γλώσσα προγραμματισμού Python, έτσι ώστε να μπορεί να εκτελεστεί εύκολα σε οποιοδήποτε υπολογιστικό σύστημα. Η απόδοση του αλγορίθμου είναι καλύτερη για την περίπτωση αραιών γράφων και είναι της τάξης του $n^{1.5-\varepsilon}$ με παράγοντα προσέγγισης μικρότερο από 2.

Οι επιμέρους στόχοι της εργασίας είναι:

- Περιγραφή του προβλήματος υπολογισμού της διαμέτρου ενός γράφου, για γράφους που αναπαριστούν πολύ μεγάλο όγκο δεδομένων.
- Παρουσίαση της σχετικής έρευνας που έχει πραγματοποιηθεί για τον προσεγγιστικό υπολογισμό της διαμέτρου ενός γράφου.
- Παρουσίαση ενός προσεγγιστικού αλγορίθμου υπολογισμού της διαμέτρου ενός γράφου με πολυπλοκότητα της τάξης του $n^{1.5-\varepsilon}$ με παράγοντα προσέγγισης μικρότερο από 2.
- Υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού Python.
- Πειραματική εκτέλεση του αλγορίθμου για δίκτυα διαφόρων μεγεθών και διαφόρων τιμών πυκνότητας.
- Παρουσίαση των αποτελεσμάτων που προέκυψαν και εξαγωγή χρήσιμων συμπερασμάτων.

Στο Κεφάλαιο 2 παρουσιάζεται το βασικό θεωρητικό υπόβαθρο για το πρόβλημα υπολογισμού της διαμέτρου ενός γράφου. Στο Κεφάλαιο 3 παρουσιάζεται η σχετική έρευνα που έχει πραγματοποιηθεί. Στο Κεφάλαιο 4 περιγράφεται ένας προσεγγιστικός αλγόριθμος για τον υπολογισμό της διαμέτρου ενός γράφου και παρουσιάζεται η υλοποίησή του γλώσσα Python. Τέλος, στο Κεφάλαιο 5 παρουσιάζεται ο επίλογος και εξάγονται κάποια χρήσιμα συμπεράσματα.

2. Υπολογισμός διαμέτρου ενός γράφου

2.1 Βασικές έννοιες από τη θεωρία Γράφων

Γράφος ή **γράφημα** (graph) $G=(V,E)$ είναι μια αναπαράσταση ενός συνόλου στοιχείων, τα οποία ονομάζονται **κορυφές** (vertices) ή **κόμβοι** (nodes), κάποια εκ των οποίων συνδέονται μεταξύ τους με δεσμούς, οι οποίοι ονομάζονται **ακμές** (edges). Το σύνολο των κορυφών συμβολίζεται με V και το σύνολο των ακμών με E . Το πλήθος των κορυφών ενός γράφου συμβολίζεται συνήθως με $|V|$ ή n και ονομάζεται **τάξη** του γράφου. Αντίστοιχα, το πλήθος των ακμών ενός γράφου συμβολίζεται συνήθως με $|E|$ ή m και ονομάζεται **μέγεθος** του γράφου.

Μία ακμή που συνδέει μία κορυφή με τον εαυτό της ονομάζεται βρόχος ή ανακύκλωση (loop). Δύο ή περισσότερες ακμές που ενώνουν το ίδιο ζευγάρι κορυφών ονομάζονται παράλληλες. Ένας γράφος ο οποίος δεν περιλαμβάνει βρόχους και παράλληλες ακμές, ονομάζεται **απλός γράφος**.

Οι ακμές ενός γράφου μπορεί να είναι κατευθυνόμενες (τόξα) ή μη-κατευθυνόμενες και επομένως ένας γράφος μπορεί να είναι **κατευθυνόμενος** (directed) ή αντίστοιχα μη κατευθυνόμενος (undirected). Στους μη κατευθυνόμενους γράφους θεωρούμε ότι για κάθε ακμή (u, v) υπάρχει και ακμή (v, u) στον γράφο, ενώ στους κατευθυνόμενους γράφους δεν ισχύει απαραίτητα κάτι τέτοιο.

Μονοπάτι (path) από έναν κόμβο σε έναν άλλο ενός γράφου, είναι μία ακολουθία από κόμβους του γράφου, όπου κάθε κόμβος της ακολουθίας συνδέεται με ακμή με τον επόμενο του. Το πλήθος των ακμών της ακολουθίας αυτής ονομάζεται και **μήκος** του μονοπατιού. Ένα μονοπάτι ενός γράφου ονομάζεται **απλό μονοπάτι** αν κάθε κόμβος εμφανίζεται το πολύ μία φορά σε αυτό. Ένα μονοπάτι μπορεί επίσης να χαρακτηριστεί ως κατευθυνόμενο ή μη κατευθυνόμενο, ανάλογα με το αν

πρόκειται για κατευθυνόμενο ή όχι γράφο. Ένα μονοπάτι για το οποίο η πρώτη του κορυφή είναι ίδια με την τελευταία κορυφή, δηλαδή που ξεκινάει και τελειώνει στην ίδια κορυφή, ονομάζεται **κύκλος**.

Μια **συνιστώσα** (component) ενός απλού γράφου είναι ένα υποσύνολο κορυφών V' του V , για το οποίο ισχύει ότι για κάθε ζεύγος κορυφών του V' , υπάρχει κάποιο μονοπάτι που τις συνδέει. Ένας γράφος καλείται **συνεκτικός** ή συνδεδεμένος (connected) εάν για κάθε ζευγάρι κορυφών, υπάρχει τουλάχιστον ένα μονοπάτι που συνδέει τις κορυφές αυτές. Ένας συνεκτικός γράφος αποτελείται από μία μόνο συνιστώσα, ενώ ένας μη συνεκτικός γράφος έχει περισσότερες από μία. Αν πρόκειται για έναν κατευθυνόμενο γράφο και υπάρχει μονοπάτι ανάμεσα σε οποιοδήποτε ζευγάρι κορυφών, τότε ο γράφος ονομάζεται **ισχυρά συνεκτικός**.

Σε κάθε ακμή e ενός γράφου μπορεί να αντιστοιχεί ένας αριθμός $w(e)$ ο οποίος ονομάζεται **βάρος** (weight). Οι τιμές των βαρών σε πραγματικές συνθήκες θα μπορούσαν να αντιπροσωπεύουν μεγέθη όπως το κόστος, το μήκος, το κέρδος κτλ. της μετάβασης από μία κορυφή του γράφου σε μία άλλη. Ένας τέτοιος γράφος ονομάζεται **σταθμισμένος** γράφος (weighted graph) ή απλά γράφος με βάρη.

Ένας γράφος στον οποίο η κάθε κορυφή συνδέεται με κάθε άλλη κορυφή του γράφου, ονομάζεται **πλήρης** γράφος ή κλίκα. Το πλήθος των ακμών m ενός πλήρους γράφου είναι ίσο με

$$m = \frac{n \cdot (n - 1)}{2}$$

Επιπλέον, αν ένας γράφος έχει σχετικά λίγες ακμές, τότε ονομάζεται **αραιός** γράφος (sparse graph), ενώ αντίθετα αν έχει πολλές σχετικά ακμές, ονομάζεται **πυκνός** γράφος (dense graph). Ως **πυκνότητα** $\rho(G)$ ενός γράφου G ορίζεται το πηλίκο

$$\rho(G) = \frac{m}{n^2} = \frac{|E|}{|V|^2}$$

Σε έναν αραιό γράφο το πλήθος των ακμών είναι της τάξης του $O(n)$, ενώ σε έναν πυκνό γράφο είναι τάξης $O(n^2)$. Η πυκνότητα ενός γράφου είναι ένας πολύ σημαντικός παράγοντας και επηρεάζει συχνά σε μεγάλο βαθμό την απόδοση ενός

αλγορίθμου που εφαρμόζεται στον γράφο.

Ως **βαθμός** (degree) d_u μίας κορυφής u ορίζεται το πλήθος των ακμών που έχουν το ένα τους άκρο σε αυτήν την κορυφή. Σύμφωνα το *λήμμα της Χειραψίας*, το άθροισμα όλων των βαθμών των κορυφών u_i ενός γράφου, ισούται με το διπλάσιο του πλήθους των ακμών του γράφου, δηλαδή

$$\sum_{i=1}^n d(u_i) = 2m$$

Για κατευθυνόμενους γράφους ορίζεται και ο **έσω βαθμός** (in-degree) d_u^- που αντιστοιχεί στο πλήθος των ακμών που καταλήγουν στην κορυφή u , όπως και ο **έξω βαθμός** (out-degree) d_u^+ που αντιστοιχεί στο πλήθος των ακμών που ξεκινούν από την κορυφή u .

2.1.1 Αναπαράσταση γράφου

Για την αναπαράσταση και αποθήκευση ενός γράφου μπορεί να χρησιμοποιηθεί ένας **πίνακας γειτνίασης** (adjacency matrix) A , ο οποίος περιλαμβάνει μία γραμμή και μία στήλη για κάθε κόμβο του γράφου και σε κάθε θέση του πίνακα έχουμε 0 ή 1 ανάλογα με το αν συνδέονται με ακμή οι αντίστοιχες κορυφές που αντιστοιχούν στη γραμμή και τη στήλη του πίνακα. Επομένως, ισχύει:

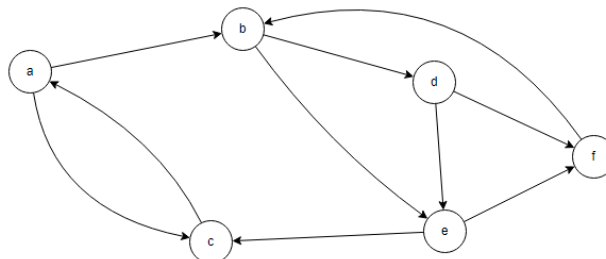
$$A_{ij} = \begin{cases} 1, & \text{αν υπάρχει ακμή } (i, j) \\ 0, & \text{διαφορετικά} \end{cases}$$

Ο χώρος που καταλαμβάνει ένας πίνακας γειτνίασης είναι τάξης $O(n^2)$, όπου n το πλήθος των κορυφών του γράφου. Πολλαπλασιάζοντας μάλιστα τον πίνακα γειτνίασης A με τον εαυτό του, προκύπτει πίνακας τα στοιχεία (i, j) του οποίου αντιστοιχούν στο πλήθος των μονοπατιών συγκεκριμένου μήκους που υπάρχουν στον γράφο και ξεκινούν από την κορυφή i καταλήγοντας στην κορυφή j . Με άλλα λόγια, το στοιχείο A_{ij}^l είναι ίσο με το πλήθος των μονοπατιών μήκους l τα οποία ξεκινούν από την κορυφή i και καταλήγουν στην κορυφή j (επιτρέπεται επανάληψη

κορυφών στα μονοπάτια αυτά).

Για παράδειγμα, για τον κατευθυνόμενο γράφο στο Σχήμα 1 **Error! Reference source not found.** αντιστοιχεί ο πίνακας γειτνίασης που εμφανίζεται στο **Σχήμα 2** **Error! Reference source not found.**. Στην περίπτωση μη κατευθυνόμενου γράφου, ο πίνακας γειτνίασης είναι συμμετρικός και είναι ίσος με τον ανάστροφό του, δηλαδή

$$A = A^T$$



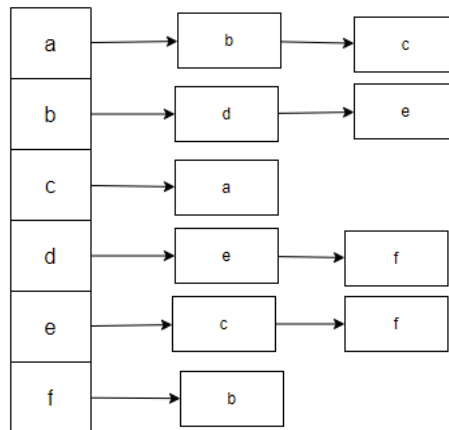
Σχήμα 1: Κατευθυνόμενος γράφος $G=(V,E)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Σχήμα 2: Πίνακας γειτνίασης κατευθυνόμενου γράφου

Ένας γράφος μπορεί εναλλακτικά να αναπαρασταθεί με τη βοήθεια μίας λίστας γειτνίασης, όπως φαίνεται στο Σχήμα 3, όπου εμφανίζεται η λίστα γειτνίασης του κατευθυνόμενου γράφου στο Σχήμα 1. Για κάθε μία από τις κορυφές v του γράφου, υπάρχει μία συνδεδεμένη λίστα η οποία περιλαμβάνει τους κόμβους με τους οποίους συνδέεται η κορυφή v . Κάθε μία από τις δύο αναπαραστάσεις που

περιγράφηκαν έχει τα δικά της πλεονεκτήματα και μειονεκτήματα, τα οποία μάλιστα επηρεάζονται από την πυκνότητα του αναπαριστώμενου γράφου. Το μεγάλο πλεονέκτημα της αναπαράστασης με τη βοήθεια πίνακα γειτνίασης είναι ότι είναι δυνατόν σε σταθερό χρόνο $O(1)$ να ελεγχθεί εάν δύο κορυφές συνδέονται μεταξύ τους με ακμή. Η αντίστοιχη λειτουργία στην περίπτωση της λίστας γειτνίασης απαιτεί γραμμικό χρόνο $O(n)$, όπου n το πλήθος των κορυφών του γράφου. Ο τρόπος αναπαράστασης που χρησιμοποιείται για τις ανάγκες των αλγορίθμων της παρούσας πτυχιακής εργασίας είναι η αναπαράσταση των γράφων με τη βοήθεια πίνακα γειτνίασης, λόγω του συγκεκριμένου πλεονεκτήματος του τρόπου αυτού αναπαράστασης.

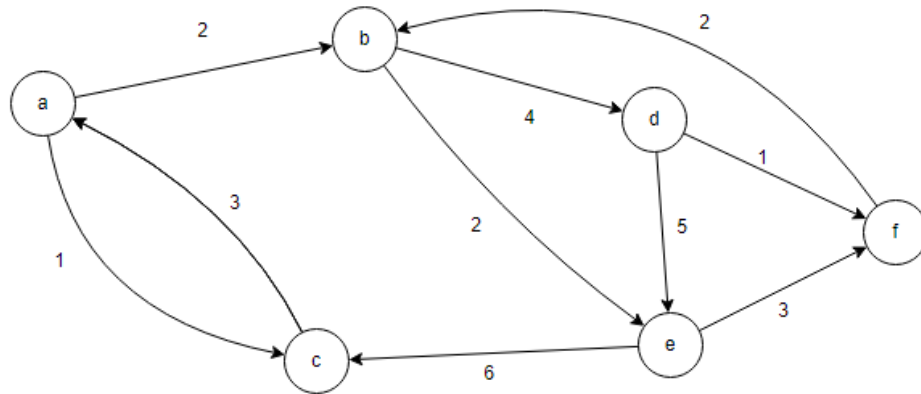


Σχήμα 3: Αναπαράσταση γράφου με λίστα γειτνίασης

Στην περίπτωση σταθμισμένου γράφου, υπάρχουν βάρη στις ακμές του γράφου αυτού και κατά την αναπαράσταση του γράφου με τη βοήθεια πίνακα γειτνίασης, θα πρέπει να αποθηκευτούν τα βάρη αυτά. Για παράδειγμα, ο σταθμισμένος γράφος που εμφανίζεται στο Σχήμα 4, μπορεί να αναπαρασταθεί με τον πίνακα γειτνίασης στο **Error! Reference source not found..** Για τα ζευγάρια κορυφών οι οποίες δεν συνδέονται με ακμή, τότε μπορούμε να θεωρήσουμε ότι το αντίστοιχο βάρος είναι άπειρο και άρα στις αντίστοιχες θέσεις του πίνακα αποθηκεύεται η τιμή που αντιπροσωπεύει το άπειρο.

Με άλλα λόγια, θεωρούμε ότι ισχύει:

$$A_{ij} = \begin{cases} w_{ij}, & \text{αν υπάρχει ακμή } (i, j) \text{ με βάρος } w_{ij} \\ \infty, & \text{διαφορετικά} \end{cases}$$



Σχήμα 4: Σταθμισμένος γράφος

$$A = \begin{bmatrix} 0 & 2 & 1 & \infty & \infty & \infty \\ \infty & 0 & \infty & 4 & 2 & \infty \\ 3 & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 5 & 1 \\ \infty & \infty & 6 & \infty & 0 & 3 \\ \infty & 2 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Σχήμα 5: Πίνακας γειτνίασης σταθμισμένου γράφου

2.2 Χαρακτηριστικά και μετρικές γράφων

Η ανάλυση των διαφόρων χαρακτηριστικών επομένως ενός γράφου είναι συχνά πολύ χρήσιμη και έχουν αναπτυχθεί διάφοροι αλγόριθμοι για την ανακάλυψη των χαρακτηριστικών αυτών. Τέτοια χαρακτηριστικά αφορούν τις αποστάσεις ανάμεσα στις κορυφές του γράφου και τις μετρικές που αφορούν τη δομή ενός γράφου, όπως είναι η εκκεντρότητα των κόμβων, η διάμετρος και η ακτίνα του γράφου κτλ. Κάποιες από αυτές τις μετρικές θα παρουσιαστούν στη συνέχεια.

Ως **συντομότερο μονοπάτι** (shortest path) ανάμεσα σε δύο κορυφές, ονομάζεται το μονοπάτι ελαχίστου μήκους που συνδέει τις δύο αυτές κορυφές μεταξύ τους. Το μήκος του συντομότερου μονοπατιού ανάμεσα σε δύο κορυφές u και v ενός γράφου, καλείται **απόσταση** ανάμεσα στις δύο κορυφές και συμβολίζεται με $\text{dist}(u,v)$. Για την απόσταση ανάμεσα στις κορυφές ενός γράφου, ισχύει πάντοτε η τριγωνική ανισότητα, δηλαδή για κάθε τριάδα κορυφών u , v και w του γράφου ισχύει

$$\text{dist}(u, v) + \text{dist}(v, w) \geq \text{dist}(u, w), \forall u, v, w \in V$$

Υπάρχουν διάφοροι αλγόριθμοι για τον υπολογισμό της μικρότερης απόστασης και των συντομότερων μονοπατιών ανάμεσα σε 2 κορυφές ενός γράφου. Οι υπολογισμοί αυτοί είναι χρήσιμοι για διάφορες μετρικές ενός γράφου, όπως η εκκεντρότητα και η διάμετρος που περιγράφονται πιο κάτω. Κάποιοι από αυτούς τους αλγόριθμους θα παρουσιαστούν στη συνέχεια.

2.2.1 Εκκεντρότητα και Διάμετρος

Ως **εκκεντρότητα** ή εκκεντρικότητα $e(v)$ μιας κορυφής v ενός μη κατευθυνόμενου γράφου $G=(V,E)$, ορίζεται η απόσταση από την κορυφή v προς την πλέον απομακρυσμένη κορυφή του γράφου. Με άλλα λόγια, εκκεντρότητα μίας κορυφής είναι η μέγιστη από τις αποστάσεις της κορυφής αυτής προς τις άλλες κορυφές του γράφου. Είναι:

$$e(v) = \max \{ \text{dist}(u, v), \forall u \in V \}$$

Αν έχουμε έναν μη συνεκτικό γράφο τότε θεωρούμε ότι έχει άπειρη ή απροσδιόριστη εκκεντρότητα.

Η κορυφή με την ελάχιστη εκκεντρότητα λέγεται **κεντρική** κορυφή και η ελάχιστη αυτή εκκεντρότητα καλείται **ακτίνα** $rad(G)$ του γράφου. Δηλαδή είναι:

$$rad(G) = \min \{ecc(u), \forall u \in V\}$$

Ένας πλήρης γράφος επομένως έχει ακτίνα ίση με τη μονάδα. Το σύνολο των κεντρικών κορυφών, δηλαδή το σύνολο των κορυφών με εκκεντρότητα ίση με την ακτίνα του γράφου λέγεται **κέντρο** $center(G)$ του γράφου G .

Η **διάμετρος** $diam(G)$ ενός γράφου ορίζεται ως η μέγιστη εκκεντρότητα των κορυφών του γράφου, δηλαδή

$$diam(G) = \max \{e(u), \forall u \in V\}$$

Αν έχουμε έναν μη συνεκτικό γράφο τότε θεωρούμε ότι έχει άπειρη ή απροσδιόριστη διάμετρο.

Οι κορυφές οι οποίες έχουν εκκεντρότητα ίση με τη διάμετρο του γράφου, ονομάζονται **απόκεντρες** κορυφές και το σύνολο των κορυφών αυτών λέγεται **απόκεντρο** του γράφου G και συμβολίζεται με $far(G)$.

Δύο κορυφές μάλιστα u και v , οι οποίες απέχουν μεταξύ τους απόσταση ίση με τη διάμετρο του γράφου, δηλαδή για τις οποίες ισχύει

$$dist(u, v) = diam(G)$$

αποκαλούνται **αντιδιαμετρικές** κορυφές.

Για όλους τους γράφους G ισχύει η διπλή ανισότητα

$$rad(G) \leq diam(G) \leq 2 \cdot rad(G)$$

Επίσης, ισχύει ότι το κέντρο και το απόκεντρο ενός γράφου G , είτε ταυτίζονται, είτε είναι ξένα μεταξύ τους σύνολα

$$\text{Για κάθε γράφο } G \left\{ \begin{array}{l} center(G) = far(G) \\ \text{ή } center(G) \cap far(G) = \emptyset \end{array} \right.$$

Για την περίπτωση κατευθυνόμενου γράφου, τότε ορίζεται η **έσω-εκκεντρότητα** (in-eccentricity) $e^-(v)$ ως η απόσταση από την πλέον απομακρυσμένη κορυφή u του γράφου μέχρι την κορυφή v . Δηλαδή είναι:

$$e^-(v) = \max \{dist(u, v), \forall u \in V\}$$

Όμοια, ορίζεται η **έξω-εκκεντρότητα** (out-eccentricity) $e^+(v)$ ως η απόσταση από την κορυφή v μέχρι την πλέον απομακρυσμένη από αυτή κορυφή u του γράφου. Δηλαδή είναι:

$$e^+(v) = \max \{dist(v, u), \forall u \in V\}$$

Στην περίπτωση αυτή η διάμετρος ορίζεται ως η τιμή της μέγιστης έσω-εκκεντρότητας ή ισοδύναμα ίση με την τιμή της μέγιστης έξω-εκκεντρότητας. Δηλαδή είναι:

$$diam(G) = \max\{e^-(v), \forall v \in V\} = \max\{e^+(v), \forall v \in V\}$$

Για παράδειγμα, για τον γράφο στο Σχήμα 1, έχουμε τις εκκεντρότητες που εμφανίζονται στο **Σχήμα 6**. Η διάμετρος επομένως του κατευθυνόμενου αυτού γράφου είναι ίση με την μέγιστη έσω (έξω) εκκεντρότητα και ισούται με 4.

Η διάμετρος ενός γράφου αποτελεί συχνά μία πολύ σημαντική παράμετρος και μπορεί να οδηγήσει σε χρήσιμα συμπεράσματα, δεδομένου ότι η διάμετρος αντιστοιχεί στην μεγαλύτερη απόσταση ανάμεσα στις κορυφές του γράφου. Η γνώση της διαμέτρου μπορεί να δώσει πληροφορίες που αφορούν τη συνδεσιμότητα και τη δομή του γράφου και συχνά ο υπολογισμός της διαμέτρου ενός γράφου κρίνεται απαραίτητος κατά τη μελέτη δικτύων που αφορούν τις σχέσεις ανάμεσα σε οντότητες του Διαδικτύου ή γενικότερα δικτύων του «πραγματικού κόσμου».

Κορυφή	Έσω-εκκεντρότητα	Έξω-εκκεντρότητα
a	4	3
b	2	3
c	3	4
d	3	3
e	3	3
f	4	4

Σχήμα 6: Έσω και έξω-εκκεντρότητες κορυφών κατευθυνόμενου γράφου

2.3 Αλγόριθμοι υπολογισμού της διαμέτρου ενός γράφου

Η διάμετρος ενός γράφου G , όπως είδαμε στις προηγούμενες παραγράφους, είναι η απόσταση ανάμεσα στις 2 πιο απομακρυσμένες κορυφές του γράφου. Επομένως, για τον υπολογισμό της διαμέτρου ενός γράφου G , θα πρέπει να υπολογιστεί για όλες τις κορυφές του γράφου η απόσταση από την πιο απομακρυσμένη από αυτήν κορυφή. Στη συνέχεια, θα πρέπει να υπολογιστεί η μέγιστη από αυτές τις αποστάσεις. Ο γρήγορος υπολογισμός της ακριβούς διαμέτρου ενός γράφου επομένως, εξαρτάται από την ταχύτητα υπολογισμού των αποστάσεων που απέχουν μεταξύ τους οι κορυφές του γράφου.

Υπάρχουν διάφοροι αλγόριθμοι οι οποίοι μπορούν να χρησιμοποιηθούν για την εύρεση της διαμέτρου ενός γράφου. Κάθε ένας από αυτούς τους αλγόριθμους μπορεί να έχει τα δικά του μειονεκτήματα και πλεονεκτήματα και τη δική του πολυπλοκότητα χρόνου. Η επίδοση ενός τέτοιου αλγορίθμου επηρεάζεται συχνά από τον τύπο και την αραιότητα του γραφήματος.

Κάποιοι από τους διάσημους αλγόριθμους που θα μπορούσαν να χρησιμοποιηθούν για τον ακριβή υπολογισμό της διαμέτρου ενός γράφου είναι οι πιο κάτω:

- Αναζήτηση κατά Πλάτος (Breadth First Search – BFS).
- Αλγόριθμος Dijkstra
- Αλγόριθμος Floyd-Warsall
- Αλγόριθμος Bellman-Ford

Υπάρχουν επίσης κάποιοι προσεγγιστικοί αλγόριθμοι για τον υπολογισμό της διαμέτρου ενός μεγάλου γράφου, όπου η διάμετρος του γράφου υπολογίζεται με κάποια προσέγγιση η οποία θεωρούμε ότι είναι ικανοποιητική, ανάλογα με την περίπτωση. Στις επόμενες παραγράφους θα παρουσιαστούν κάποιες εξελίξεις στο πρόβλημα του προσεγγιστικού υπολογισμού της διαμέτρου ενός γράφου και θα υλοποιηθεί ένας τέτοιος αλγόριθμος. Για τις ανάγκες της εργασίας υλοποιήθηκαν οι αλγόριθμοι κατά Πλάτος και ο αλγόριθμος Dijkstra, όπως παρουσιάζεται στις επόμενες παραγράφους.

2.3.1 Κατά Πλάτος Αναζήτηση σε γράφο χωρίς βάρη (Breadth First Search – BFS)

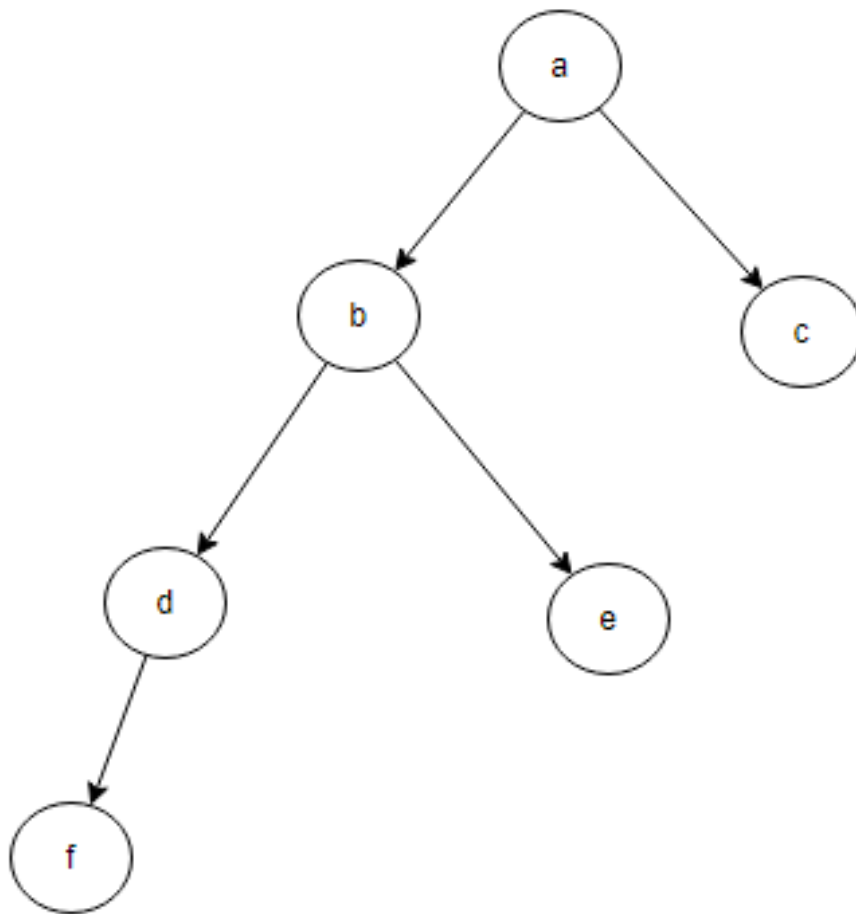
Για την περίπτωση ενός μη σταθμισμένου γράφου, οι ακμές του γράφου δεν έχουν κάποιο βάρος και μπορεί επομένως να υπολογιστεί η διάμετρος του με εφαρμογή Αναζήτησης κατά Πλάτος (Breadth First Search – BFS). Κατά την εφαρμογή του αλγορίθμου BFS, μπορεί να σχηματιστεί το δέντρο της αναζήτησης και η εκκεντρότητα της κορυφής από την οποία ξεκίνησε η αναζήτηση, αντιστοιχεί στο ύψος αυτού του δέντρου.. Με άλλα λόγια, κατά την εμφάνιση μίας νέας κορυφής του γράφου, θα πρέπει να καταχωρείται το βάθος της κορυφής αυτής στο δέντρο αναζήτησης, δηλαδή η απόσταση του αντίστοιχου κόμβου από την ρίζα του δέντρου. Η μεγαλύτερη από τις αποστάσεις αυτές είναι ουσιαστικά η διάμετρος του γράφου.

Για τον γράφο στο Σχήμα 1 για παράδειγμα, η αναζήτηση κατά πλάτος αν ξεκινώντας από την κορυφή a, δίνει το δέντρο αναζήτησης που εμφανίζεται στο

Σχήμα 7, ενώ αν ξεκινήσουμε από την κορυφή c έχουμε το δέντρο που εμφανίζεται στο **Σχήμα 8**. Στην πρώτη περίπτωση το δέντρο έχει ύψος ίσο με 3 και άρα η πιο απομακρυσμένη κορυφή από την κορυφή a απέχει απόσταση 3. Αντίστοιχα, από το δεύτερο δέντρο προκύπτει ότι η πιο απομακρυσμένη κορυφή από την κορυφή c απέχει απόσταση 4. Αν εφαρμόσουμε Αναζήτηση κατά Πλάτος για όλες τις κορυφές του γράφου, προκύπτει ότι η μέγιστη απόσταση είναι 4 και άρα η διάμετρος του γράφου είναι 4. Μπορεί επομένως να υπολογιστεί η διάμετρος ενός μη σταθμισμένου γράφου εφαρμόζοντας για κάθε κορυφή του Αναζήτηση κατά Πλάτος (BFS).

Κατά την εφαρμογή του αλγορίθμου BFS, θα πρέπει να εξεταστούν όλες οι κορυφές του γράφου μία φορά η κάθε μία και να εξεταστούν επίσης και όλες οι ακμές του γράφου, επίσης μία φορά η κάθε μία. Επομένως, η πολυπλοκότητα της αναζήτησης BFS είναι $O(|V|+|E|)$, όπου V είναι το σύνολο των κορυφών και E το σύνολο των ακμών ή αλλιώς $O(n+m)$, αν θεωρήσουμε ότι n είναι το πλήθος των κορυφών και m το πλήθος των ακμών του γράφου.

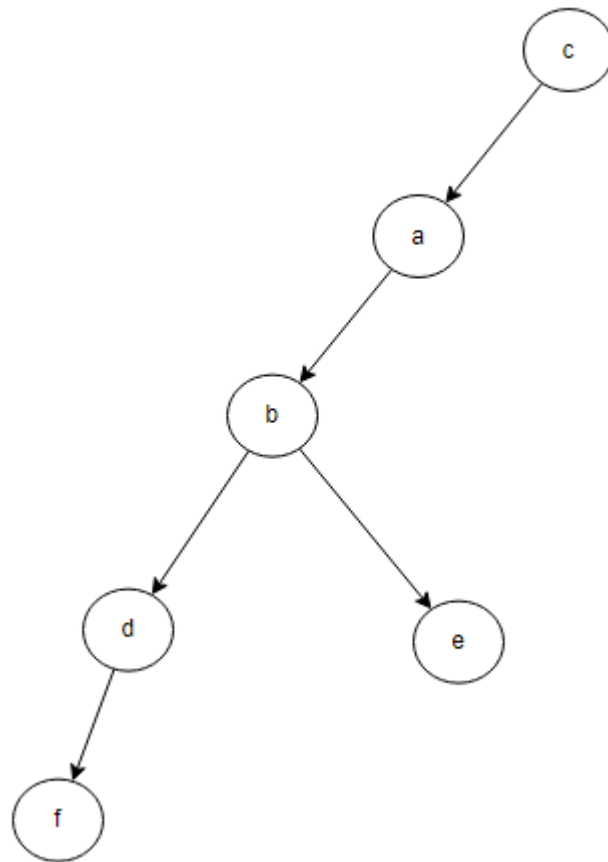
Η πολυπλοκότητα επομένως είναι γραμμική ως προς τη διάταξη του προβλήματος (πλήθος κορυφών και ακμών). Επειδή όμως θα πρέπει να εφαρμοστεί ο αλγόριθμος BFS για κάθε μία από τις κορυφές του γράφου ώστε να υπολογιστεί η διάμετρος, προκύπτει ότι η πολυπλοκότητα για την εύρεση της διαμέτρου όταν εφαρμόζεται Αναζήτηση κατά Πλάτος είναι της τάξης του $O(n^2 + nm)$. Για την εύρεση της μεγαλύτερης από τις αποστάσεις που προκύπτουν απαιτείται χρόνος $O(n)$, αλλά αυτό δεν επηρεάζει τη συνολική πολυπλοκότητα του αλγορίθμου.



Σχήμα 7: Αναζήτηση κατά Πλάτος (BFS) ξεκινώντας από την κορυφή a

Ο αλγόριθμος της αναζήτησης κατά πλάτος αναπτύχθηκε σε γλώσσα προγραμματισμού Python στα πλαίσια της εργασίας, έτσι ώστε να είναι μπορούν να αξιολογηθεί η ακρίβεια των αποτελεσμάτων του προσεγγιστικού αλγόριθμου που περιγράφεται στη συνέχεια.

Ο κώδικας Python για την εφαρμογή του αλγορίθμου αναζήτησης κατά πλάτος δίνεται στην **Εικόνα 1**.



Σχήμα 8: Αναζήτηση κατά Πλάτος (BFS) ξεκινώντας από την κορυφή c

```

# Συνάρτηση η οποία εφαρμόζει αναζήτηση κατά πλάτος σε έναν γράφο χωρίς βάρη
# ξεκινώντας από συγκεκριμένη κορυφή του γράφου
# A είναι ο πίνακας γειτνίασης του γράφου
# n είναι το πλήθος των κορυφών
# node είναι η κορυφή από την οποία ξεκινά η αναζήτηση
# Η συνάρτηση επιστρέφει την απόσταση από την πιο απομακρυσμένη κορυφή
def bfs(A, n, node):
    # Καταχώρηση των κόμβων με τις αντίστοιχες αποστάσεις από τη ρίζα του δέντρου
    # που απεικονίζει την αναζήτηση BFS
    list_of_nodes = {node:0}
    # Λίστα με τους κόμβους που εξετάζουμε
    open_nodes = [node]
    # Όσο υπάρχουν κόμβοι που είναι υπό-εξέταση
    while len(open_nodes) > 0:
        # Παίρνουμε τον επόμενο κόμβο από την αρχή της λίστας
        current_node = open_nodes[0]
        # Τον αφαιρούμε από τη λίστα
        open_nodes.pop(0)
        for i in range(n):
            # προσθέτουμε τους γείτονές του στη λίστα
            if A[current_node, i] > 0:
                # Αν δεν έχουν ήδη εξεταστεί
                if (i not in open_nodes) and (i not in list_of_nodes.keys()):
                    open_nodes.append(i)
                    list_of_nodes[i] = list_of_nodes[current_node] + 1
        # Αν ο γράφος είναι μη συνεκτικός έχουμε άπειρη διάμετρο
        if len(list_of_nodes) < n:
            return math.inf
    else:
        return max(list_of_nodes.values())

```

Εικόνα 1: Κώδικας Python για την αναζήτηση κατά Πλάτος

2.3.2 Αλγόριθμος Dijkstra

Ο αλγόριθμος Dijkstra είναι ένας πολύ δημοφιλής άπληστος αλγόριθμος που βοηθά στην εύρεση των συντομότερων μονοπατιών από μια κορυφή ενός σταθμισμένου γράφου, η οποία θεωρείται ως πηγή (source), σε όλες τις άλλες κορυφές του γράφου. Ο αλγόριθμος Dijkstra μπορεί να λειτουργήσει επομένως σε γράφους όπου η κάθε ακμή έχει κάποιο βάρος, είτε πρόκειται για κατευθυνόμενο είτε για μη κατευθυνόμενο γράφο. Δεν μπορεί να λειτουργήσει αν υπάρχουν αρνητικά βάρη σε κάποιες ακμές. Στην περίπτωση αυτή, μπορεί να χρησιμοποιηθεί εναλλακτικά ο αλγόριθμος Bellman-Ford, ο οποίος επίσης υπολογίζει τα συντομότερα μονοπάτια από μία κορυφή προς όλες τις άλλες. Αν έχουμε μη σταθμισμένο γράφο, τότε μπορούμε να θεωρήσουμε ότι όλες οι ακμές έχουν βάρος ίσο με τη μονάδα και να εκτελέσουμε τον αλγόριθμο του Dijkstra.

Για την εύρεση της διαμέτρου ενός γράφου με τη βοήθεια του αλγορίθμου του Dijkstra, θα πρέπει να εφαρμοστεί ο αλγόριθμος επαναληπτικά σε κάθε μία από τις κορυφές του γράφου, ώστε να υπολογιστούν τα κόστη των συντομότερων μονοπατιών από κάθε κορυφή προς κάθε άλλη και να κρατήσουμε ως διάμετρο το μεγαλύτερο από αυτά τα κόστη.

Ο αλγόριθμος του Dijkstra εκτελεί τα πιο κάτω βήματα:

1. Βάζουμε σε κάθε κορυφή μία ετικέτα, έστω d . Στην πηγή (source) δίνουμε ετικέτα 0 και σε όλες τις άλλες κορυφές ετικέτα άπειρο.
2. Θεωρούμε το σύνολο S , το οποίο είναι αρχικά κενό και περιλαμβάνει τις κορυφές που έχουν την τελική τιμή για την ετικέτα τους.
3. Επιλέγουμε την κορυφή u που δεν είναι στο σύνολο S και έχει την μικρότερη ετικέτα.
4. Προσθέτουμε αυτήν την κορυφή u στο S και για όλες τις κορυφές v που δεν είναι στο S και είναι γείτονες της u , υπολογίζουμε το άθροισμα $d[u] + c(u, v)$, όπου $c(u, v)$ το βάρος της ακμής από την κορυφή u στην v .
5. Αν είναι $d[v] > d[u] + c(u, v)$, τότε θέτουμε $d[v] = d[u] + c(u, v)$, δηλαδή αλλάζει τιμή η ετικέτα της v . Διαφορετικά κρατάει την προηγούμενη ετικέτα.
6. Επαναλαμβάνουμε την προηγούμενη διαδικασία για όσο υπάρχουν κορυφές που δεν ανήκουν στο σύνολο S .

Οι τελικές τιμές για τις ετικέτες των κορυφών του γράφου, αντιστοιχούν στα κόστη για τα συντομότερα μονοπάτια από την αρχική κορυφή u προς όλες τις άλλες κορυφές του γράφου. Αν επαναληφθεί η πιο πάνω διαδικασία για όλες τις κορυφές του γράφου, τότε μπορεί να υπολογιστεί εύκολα η ακριβής τιμή για τη διάμετρο του γράφου.

Αυτό ισχύει θεωρώντας ότι δεν χρησιμοποιούμε κάποια ουρά προτεραιότητας για να βρίσκουμε την κορυφή με την ελάχιστη ετικέτα στο βήμα 3. Με δυαδικό σωρό, η πολυπλοκότητα γίνεται $O(m \log n)$ και με σωρό Fibonacci $O(m + n \log n)$. Σε περιπτώσεις που γίνει χρήση πιο σύνθετων δομών δεδομένων η πολυπλοκότητα του αλγορίθμου μπορεί να μειωθεί περισσότερο. Επειδή όμως θα πρέπει να εφαρμοστεί ο αλγόριθμος Dijkstra για κάθε μία από τις κορυφές του γράφου ώστε να υπολογιστεί η διάμετρος, προκύπτει ότι η πολυπλοκότητα για την εύρεση της διαμέτρου είναι της τάξης του $O(mn + n^2 \log n)$.

Βλέπουμε επομένως ότι η πολυπλοκότητα της αναζήτησης κατά πλάτος είναι μικρότερη από αυτήν όπου χρησιμοποιείται ο αλγόριθμος Dijkstra. Όμως, για την περίπτωση του αλγορίθμου κατά Πλάτος θεωρήσαμε ότι ο γράφος δεν είναι σταθμισμένος, ενώ για την περίπτωση του αλγορίθμου του Dijkstra είχαμε σταθμισμένο γράφο.

Ο κώδικας Python που αναπτύχθηκε για την υλοποίηση του αλγορίθμου Dijkstra εμφανίζεται στην **Εικόνα 2**. Ο κώδικας κάνει χρήση μίας δομής ουράς προτεραιότητας, έτσι ώστε να επιλέγει σε κάθε επανάληψη την κορυφή που έχει την μικρότερη ετικέτα.

```

# Αλγόριθμος Dijkstra για τον υπολογισμό συντομότερων αποστάσεων
# από την κορυφή source σε όλες τις άλλες κορυφές του γράφου
# ο γράφος έχει πίνακα γειτνίασης A, n: πλήθος κορυφών γράφου
def shortest_distances_from_source(source, A, n):
    # πίνακας αποστάσεων
    distances = {node: float("inf") for node in range(n)}
    distances[source] = 0
    # Ουρά προτεραιότητας
    pq = [(0, source)]
    heapify(pq)
    # Σύνολο κόμβων που έχουμε επισκεφθεί ήδη
    visited = set()
    while pq: # Όσο η ουρά δεν είναι άδεια
        current_distance, current_node = heappop(pq)
        # Αν έχουμε ήδη επισκεφθεί τον κόμβο
        if current_node in visited:
            continue
        # Αν δεν τον έχουμε επισκεφθεί τον προσθέτουμε στην ουρά
        visited.add(current_node)
        # Ελέγχουμε τις γειτονικές κορυφές
        for j in range(n):
            if(A[current_node,j] < math.inf):
                # νέα απόσταση μέσω του άλλου κόμβου
                new_distance = current_distance + A[current_node,j]
                # Αν η νέα απόσταση είναι μικρότερη από την προηγούμενη
                # ενημερώνουμε την ουρά
                if new_distance < distances[j]:
                    distances[j] = new_distance
                    heappush(pq, (new_distance, j))
    return distances

```

Εικόνα 2: Κώδικας Python για τον αλγόριθμο Dijkstra

3. Προσεγγιστικός αλγόριθμος υπολογισμού διαμέτρου

3.1 Θεωρητικό υπόβαθρο

Ο υπολογισμός της διαμέτρου ενός γράφου αποτελεί, όπως είδαμε στις προηγούμενες παραγράφους, ένα από τα θεμελιώδη προβλήματα και έχει πολλές πρακτικές εφαρμογές σε διάφορους τομείς. Ειδικά για την περίπτωση των μη κατευθυνόμενων γραφημάτων, η θεωρητική υπολογιστική δυσκολία για τον ακριβή υπολογισμό της διαμέτρου ενός γραφήματος έχει μελετηθεί σε πολύ μεγάλο βαθμό. Για τα κατευθυνόμενα γραφήματα όμως υπάρχουν επιπλέον δυσκολίες, αφού είναι σημαντική στην περίπτωση αυτή η κατεύθυνση της κάθε ακμής. Στην περίπτωση αυτή, για ένα ζεύγος κορυφών u και v , υπάρχει η απόσταση για τη μετάβαση από την u στην v , όπως και η απόσταση για την αντίστροφη διαδρομή, δηλαδή από την v στην u . Σε κάποιες περιπτώσεις είναι σημαντική η συνολική απόσταση από την u στην v και αντίστροφα (roundtrip).

Με τη χρήση σύνθετων δομών, ο αλγόριθμος Dijkstra που περιγράφηκε στις προηγούμενες παραγράφους, απαιτεί χρόνο της τάξης $O(m+n \cdot \log n)$ για κάθε κορυφή που εφαρμόζεται, όπου n το πλήθος των κορυφών και m το πλήθος των ακμών του γράφου. Άρα αν χρησιμοποιηθεί για την εύρεση της διαμέτρου, θα πρέπει να εκτελεστεί n φορές και άρα η πολυπλοκότητα θα είναι τάξης $O(m \cdot n + n^2 \cdot \log n)$, δηλαδή $\tilde{O}(n^2)$ για αραιά γραφήματα. Ο συμβολισμός \tilde{O} χρησιμοποιείται μερικές φορές για να αγνοήσουμε τον λογαριθμικό παράγοντα και να επικεντρωθούμε στο πολυωνυμικό μέρος της πολυπλοκότητας.

Επομένως, υπολογίζοντας τις συντομότερες διαδρομές για όλα τα δυνατά ζευγάρια (All-Pairs Shortest Paths – APSP), ο απαιτούμενος χρόνος είναι της τάξης του $\tilde{O}(n^2)$. Έχει αποδειχθεί ότι αν κάποιος αλγόριθμος μπορέσει να υπολογίσει τη

διάμετρο ενός γράφου με n κορυφές σε χρόνο $O(n^{2-\epsilon})$ για κάποιο θετικό $\epsilon > 0$, τότε παραβιάζεται η καλά εδραιωμένη Υπόθεση Ισχυρού Εκθετικού Χρόνου (Strong Exponential Time Hypothesis), γνωστή ως SETH (Roditty & Vassilevska-Williams, 2013, Williams, 2005). Η υπόθεση SETH αφορά το πρόβλημα k -SAT και λέει ότι για κάθε θετικό $\epsilon > 0$, υπάρχει τιμή του k , έτσι ώστε το πρόβλημα k -SAT δεν μπορεί να λυθεί σε χρόνο $O(2^{(1-\epsilon)n})$, δηλαδή σε χρόνο μικρότερο από $O(2^n)$.

Ο χρόνος της τάξης του $\tilde{O}(n^2)$ που απαιτείται για τον ακριβή υπολογισμό της διαμέτρου ενός γράφου θεωρείται αρκετά μεγάλος, όταν το πλήθος n των κορυφών αυξάνεται σημαντικά, δηλαδή για τις περιπτώσεις όπου έχουμε μεγάλο όγκο δεδομένων. Για τον λόγο αυτόν έχουν προταθεί διάφοροι προσεγγιστικοί αλγόριθμοι. Ένας προσεγγιστικός αλγόριθμος προσπαθεί να βρει λύση σε περιπτώσεις όπου η εύρεση της ακριβούς λύσης μπορεί να είναι πολύ χρονοβόρο ή αδύνατον να υπολογιστεί. Αντί να δώσει με απόλυτη ακρίβεια τη βέλτιστη λύση, ένας προσεγγιστικός αλγόριθμος δίνει μια λύση η οποία θεωρείται ότι είναι σχετικά κοντά στην ακριβή λύση, αλλά σε πολύ μικρότερο χρόνο. Η ακρίβεια της λύσης που προσφέρει ένας προσεγγιστικός αλγόριθμος, είναι ένας πολύ σημαντικός παράγοντας για να αξιολογήσουμε τον αλγόριθμο, ανάλογα πάντοτε με το πεδίο εφαρμογής του. Για παράδειγμα, ένας 2-προσεγγιστικός αλγόριθμος για τον υπολογισμό της διαμέτρου ενός γράφου, εγγυάται ότι η διάμετρος που υπολογίζει θα είναι το πολύ διπλάσια από την ακριβή διάμετρο. Με άλλα λόγια, αν π.χ. η πραγματική διάμετρος είναι 10, τότε ένας 2-προσεγγιστικός αλγόριθμος θα δώσει ως απάντηση το πολύ 20, όχι όμως περισσότερο.

Έχουν προταθεί διάφοροι άλλοι προσεγγιστικοί αλγόριθμοι για τον υπολογισμό της διαμέτρου ενός γράφου. Έχουν προταθεί 2-προσεγγιστικοί αλγόριθμοι χρόνου $\tilde{O}(n)$, οι οποίοι βασίζονται σε αλγόριθμους εύρεσης συντομότερων διαδρομών από μία κορυφή, όπως είναι ο Dijkstra (Single Source Shortest Paths - SSSP). Επίσης, υπάρχει και 3/2-προσεγγιστικός αλγόριθμος τάξης $\tilde{O}(n^{1.5})$ (Chechik, 2014). Για μη κατευθυνόμενους γράφους επομένως μπορούν να χρησιμοποιηθούν οι πιο πάνω προσεγγιστικοί αλγόριθμοι, ανάλογα με την επιθυμητή ακρίβεια.

3.2 Παρουσίαση του προβλήματος και σχετική έρευνα

Οι αλγόριθμοι που περιγράφηκαν στην προηγούμενη παράγραφο αναφέρονται κυρίως στην περίπτωση των μη κατευθυνόμενων γράφων, ενώ αυτοί που αναφέρονται σε μη κατευθυνόμενους γράφους είναι πολύ λιγότεροι (Cairo et al, 2016). Αποτελεί ανοικτό ερώτημα επομένως αν υπάρχει η δυνατότητα ανάπτυξης περισσότερων προσεγγιστικών αλγορίθμων για κατευθυνόμενους γράφους, όπως αντίστοιχα υπάρχουν για μη κατευθυνόμενους (Rubinstein & Vassilevska-Williams, 2019).

Για τη βελτίωση της απόδοσης των προσεγγιστικών αλγορίθμων μπορεί να γίνει χρήση τεχνικών γρήγορου πολλαπλασιασμού πινάκων (Fast Matrix Multiplication). Ενώ ο παραδοσιακός πολλαπλασιασμός πινάκων διάστασης $n \times n$ απαιτεί χρόνο της τάξης του $O(n^3)$, έχουν προταθεί αλγόριθμοι οι οποίοι επιτυγχάνουν καλύτερη απόδοση. Ένας διάσημος τέτοιος αλγόριθμος είναι ο αλγόριθμος του Strassen, ο οποίος είναι ένας Διαίρει και Βασίλευε (Divide and Conquer) αλγόριθμος που επιτυγχάνει πολλαπλασιασμό πινάκων σε χρόνο $O(n^{2.81})$. Αυτό το όριο έχει μειωθεί αρκετά από άλλους αλγόριθμους, όπως ο αλγόριθμος Coppersmith-Winograd που έχει πολυπλοκότητα $O(n^{2.376})$. Η καλύτερη απόδοση για αλγόριθμο πολλαπλασιασμού πινάκων είναι θεωρητικά της τάξης του $O(n^{2.371339})$, αν και στην πράξη τέτοιοι αλγόριθμοι δεν χρησιμοποιούνται συχνά γιατί εμφανίζουν άλλες πρακτικές δυσκολίες, λόγω προβλημάτων που προκύπτουν από τη χρήσης πολύ μεγάλων σταθερών, σύνθετων δομών δεδομένων κτλ. (Alman et al., 2024).

Υπάρχουν ήδη προσεγγιστικοί αλγόριθμοι υπολογισμού της διαμέτρου ενός γράφου οι οποίοι κάνουν χρήση γρήγορου πολλαπλασιασμού πινάκων, αλλά αναφέρονται σε πυκνούς γράφους (Alon et al., 1997, Dalirrooyfard M. & Kaufmann, 2021). Κανένας από τους αλγορίθμους αυτούς όμως δεν κατάφερε να "σπάσει" το όριο του $O(n^{1.5})$, κάτι που έχει απομείνει σαν ένα ακόμη ανοικτό πρόβλημα.

Μπορεί όμως να αποδειχθεί το πιο κάτω θεώρημα (Abboud et al., 2023):

Θεώρημα-1: Έστω $k = 2^{t+2}$ για κάποιον μη αρνητικό ακέραιο $t \geq 0$. Για κάθε $\varepsilon > 0$ (που πιθανόν εξαρτάται από το πλήθος m των ακμών του γράφου), υπάρχει κάποιος τυχαιοκρατικός $2 - \frac{1}{k} + \varepsilon$ -προσεγγιστικός αλγόριθμος υπολογισμού της διαμέτρου ενός κατευθυνόμενο σταθμισμένου γράφου με πολυπλοκότητα χρόνου $\tilde{O}(m^{1+\alpha}/\varepsilon)$, όπου είναι

$$\alpha = \frac{2 \left(\frac{2}{\omega-1} \right)^t - \frac{(\omega-1)^2}{2}}{\left(\frac{2}{\omega-1} \right)^t (7-\omega) - \frac{\omega^2-1}{2}}$$

Ο συντελεστής $2 \leq \omega \leq 2.37286$ του θεωρήματος αναφέρεται στον εκθέτη του γρήγορου πολλαπλασιασμού πινάκων (Alman & Vassilevska-Williams, 2021). Ο πολλαπλασιασμός δύο πινάκων A και B διαστάσεων $a \times b$ και $b \times a$ αντίστοιχα, όπου κάθε ένας από τους πίνακες έχει το πολύ ac μη μηδενικά στοιχεία, μπορεί να εκτελεστεί κάνοντας χρήση πολλαπλασιασμού αραιών πινάκων σε χρόνο τάξης $O \left(ac \cdot a^{\frac{\omega-1}{2}} \right)$ (Kaplan et al. 2006).

Ο πολλαπλασιασμός αραιών πινάκων χρησιμοποιείται ώστε να ελαττώσει την πολυπλοκότητα του πολλαπλασιασμού, αφού αν τα περισσότερα στοιχεία του πίνακα είναι μηδενικά, τότε πολλές πράξεις μπορούν να παραλειφθούν. Οι αλγόριθμοι πολλαπλασιασμού αραιών πινάκων έχουν συνήθως διπλό στόχο:

- Τη βελτίωση της απόδοσης, έτσι ώστε να απαιτείται μικρότερος υπολογιστικός χρόνος.
- Μείωση των απαιτήσεων μνήμης, αφού η αποθήκευση πολλών μηδενικών μπορεί να γίνει ευκολότερα.

Ο αλγόριθμος που προτείνεται στη συνέχεια και βασίζεται στο πιο πάνω θεώρημα, μπορεί να χρησιμοποιηθεί σε αραιούς κατευθυνόμενους γράφους (Abboud et al. 2023). Ο αλγόριθμος αυτός είναι $7/4$ -προσεγγιστικός με πολυπλοκότητα χρόνου ίση με $O(m^{1.4575})$, όπου m το πλήθος των ακμών του γράφου και για $\alpha = \frac{\omega+1}{\omega+5} \leq 0.4575$.

Αποδεικνύεται επίσης το πιο κάτω θεώρημα (Abboud et al., 2023):

Θεώρημα-2: Έστω $\alpha = \frac{\omega+1}{\omega+5}$. Υπάρχει τυχαιοκρατικός 7/4-προσεγγιστικός αλγόριθμος υπολογισμού της διαμέτρου ενός κατευθυνόμενου γράφου χωρίς βάρη σε χρόνο $\tilde{O}(m^{1+\alpha})$

Ένας τέτοιος αλγόριθμος που αναφέρει το θεώρημα-2 περιγράφεται αναλυτικά στις επόμενες παραγράφους.

Για κατευθυνόμενους γράφους, όπως αναφέρθηκε προηγουμένως, η εύρεσης της διαμέτρου συχνά είναι πιο πολύπλοκη, αφού γενικά για ένα ζεύγος κορυφών u και v του γράφου ισχύει $d(u,v) \neq d(v,u)$. Στην περίπτωση αυτή μπορεί να μας ενδιαφέρει το άθροισμα των δύο αυτών αποστάσεων, δηλαδή το $\tilde{d}(u,v) = d(u,v) + d(v,u)$, γνωστό ως απόσταση roundtrip (Cowen & Wagner, 1999). Ο αλγόριθμος υπολογισμού των συντομότερων μονοπατιών για όλα τα δυνατά ζεύγη κορυφών (All-Pairs Shortest Path – APSP) και ο 2-προσεγγιστικός αλγόριθμος SSSP μπορούν να χρησιμοποιηθούν και για την περίπτωση των roundtrip αποστάσεων. Δεν έχουν όμως αναπτυχθεί ακόμη προσεγγιστικοί αλγόριθμοι με καλύτερη απόδοση (Abboud et al., 2016). Το πρόβλημα αυτό αποτέλεσε ένα από τα βασικά θέματα συζήτησης στη διάρκεια του “Fine-Grained Approximation Algorithms and Complexity Workshop” στο Betimono το 2019, αν και δεν υπήρξαν θεαματικά αποτελέσματα.

Έχει αποδειχθεί όμως ότι οποιοσδήποτε 5/3-ε προσεγγιστικός αλγόριθμος απαιτεί $O(n^{2-O(1)})$ χρόνο (Abboud et al., 2023). Η απόδειξη αυτή βασίζεται σε αναγωγή από το πρόβλημα που είναι γνωστό ως All-nodes k-Cycle για κατευθυνόμενους γράφους.

Εκτός από τον κλασικό υπολογισμό της διαμέτρου ενός γράφου και τον υπολογισμό της διαμέτρου με βάση τις roundtrip αποστάσεις, έχει παρουσιαστεί και ο όρος της min-διαμέτρου (Dalirrooyfard et al., 2019). Στην περίπτωση αυτή η απόσταση ανάμεσα σε δύο κορυφές u και v ενός κατευθυνόμενου γράφου ορίζεται ως η ελάχιστη των αποστάσεων ανάμεσα στις δύο κατευθύνσεις, δηλαδή η $\min\{d(u,v), d(v,u)\}$. Το πρόβλημα αυτό είναι πιο δύσκολο να λυθεί ακόμη και από την περίπτωση υπολογισμού των roundtrip αποστάσεων και δεν έχει παρουσιαστεί ούτε καν 2-προσεγγιστικός αλγόριθμος ο οποίος να έχει πολυπλοκότητα μικρότερη

από $O(n^2)$. Το αντίστοιχο πρόβλημα max-διαμέτρου, όπου ως απόσταση ανάμεσα σε δύο κορυφές u και v ενός κατευθυνόμενου γράφου θεωρείται η $\min\{d(u,v), d(v,u)\}$, θεωρείται πρόβλημα θεωρητικά ισοδύναμο με το κλασικό πρόβλημα υπολογισμού διαμέτρου κατευθυνόμενου γράφου (Abboud et al., 2023).

Στη συνέχεια του κεφαλαίου θα παρουσιαστούν αναλυτικά τα βήματα του αλγορίθμου που αναφέρεται στο θεώρημα-2, ενώ στο επόμενο κεφάλαιο θα παρουσιαστεί η υλοποίηση του συγκεκριμένου αλγορίθμου σε γλώσσα προγραμματισμού Python.

3.3 Προσεγγιστικός αλγόριθμος υπολογισμού διαμέτρου

3.3.1 Συμβολισμοί και προαπαιτούμενα

Για μία κορυφή v ενός γράφου συμβολίζουμε με $\deg(v)$ το βαθμό της κορυφής αυτής, δηλαδή το πλήθος των ακμών που έχουν το ένα τους άκρο στην κορυφή v . Για έναν μη αρνητικό πραγματικό αριθμό $r \geq 0$, συμβολίζουμε με $B_r^{in}(v)$, το σύνολο των κορυφών u από τις οποίες απέχει η v απόσταση το πολύ r .

Επομένως είναι:

$$B_r^{in}(v) = \{u: d(u, v) \leq r\}$$

Το σύνολο $B_r^{in}(v)$ λέγεται και έσω-σφαίρα γύρω από την κορυφή v ακτίνας r (in-ball of radius r around v).

Όμοια, συμβολίζουμε με $B_r^{out}(v)$, το σύνολο των κορυφών u οι οποίες απέχουν απόσταση από τη v το πολύ r .

Επομένως είναι:

$$B_r^{out}(v) = \{u: d(v, u) \leq r\}$$

Το σύνολο $B_r^{out}(v)$ λέγεται και έξω-σφαίρα γύρω από την κορυφή v ακτίνας r (out-ball of radius r around v).

Συμβολίζουμε επίσης με $B_r^{in+}(v)$ το σύνολο των κορυφών u του γράφου οι οποίες ανήκουν στο σύνολο $B_r^{in}(v)$, ή είναι έσω-γείτονες με κάποια κορυφή του $B_r^{in}(v)$.

Δηλαδή είναι:

$$B_r^{in+}(v) = B_r^{in}(v) \cup \{u: \text{υπάρχει } w \in B_r^{in}(v) \text{ και υπάρχει ακμή } (u, w)\}$$

Όμοια, συμβολίζουμε με $B_r^{out+}(v)$ το σύνολο των κορυφών u του γράφου οι οποίες ανήκουν στο σύνολο $B_r^{out}(v)$, ή είναι έξω-γείτονες με κάποια κορυφή του $B_r^{out}(v)$.

Δηλαδή είναι:

$$B_r^{out+}(v) = B_r^{out}(v) \cup \{u: \text{υπάρχει } w \in B_r^{out}(v) \text{ και υπάρχει ακμή } (w, u)\}$$

Επίσης, συμβολίζουμε με ω τον συντελεστή πολλαπλασιασμού πινάκων και θεωρούμε ότι είναι:

$$\omega \leq 2.371339$$

Θεωρούμε επίσης ότι είναι

$$\alpha = \frac{\omega + 1}{\omega + 5} \leq 0.4575$$

Τέλος, σημειώνεται ότι όλοι οι λογάριθμοι του αλγόριθμου που παρουσιάζεται στη συνέχεια είναι με βάση την σταθερά e , δηλαδή έχουμε φυσικούς λογαρίθμους, εκτός αν δηλωθεί κάποια άλλη βάση.

3.3.2 Περιγραφή του αλγορίθμου

Θα παρουσιαστούν στην παράγραφο αυτή τα βήματα του 7/4-προσεγγιστικού αλγόριθμου υπολογισμού της διαμέτρου για έναν κατευθυνόμενο μη σταθμισμένο γράφο $G(V, E)$ με $n=|V|$ κορυφές και $m=|E|$ ακμές. Η πολυπλοκότητα του αλγορίθμου είναι $O(m^{1.4575})$, όπου m το πλήθος των ακμών του γράφου (Abboud, 2023).

Ο αλγόριθμος A_D θεωρούμε ότι παίρνει σαν είσοδο έναν θετικό ακέραιο D και εκτελεί τα πιο κάτω βήματα:

- Αρχικά δημιουργούμε έναν νέο γράφο G' ο οποίος περιλαμβάνει $2m$ κορυφές, όπου καμία κορυφή δεν έχει βαθμό μεγαλύτερο από 3. Ο νέος γράφος G' έχει την ίδια διάμετρο με τον αρχικό γράφο G . Ο αλγόριθμος δημιουργίας του νέου γράφου αντικαθιστά κάθε κορυφή v του G με έναν κύκλο των $\deg(v)$ κορυφών που συνδέονται μεταξύ τους με ακμή μηδενικού βάρους. Ο γράφος G' επομένως είναι σταθμισμένος, αφού κάποιες ακμές θεωρούμε ότι έχουν βάρος ίσο με 1, ενώ άλλες έχουν μηδενικό βάρος. Στα επόμενα βήματα, ο αλγόριθμος λειτουργεί στον νέο γράφο G' . Η διαδικασία κατασκευής του νέου γράφου G' , περιγράφεται στην επόμενη παράγραφο.
- Επιλέγουμε τυχαίο δείγμα από $4m^a \log m$ ομοιόμορφα κατανεμημένες κορυφές και υπολογίζουμε για κάθε μία από τις κορυφές αυτές την έσω και την έξω-εκκεντρότητά τους. Αν κάποια από τις κορυφές αυτές έχει έσω ή έξω-εκκεντρότητα τουλάχιστον $4D/7$, τότε ο αλγόριθμος αποδέχεται (accept).
- Για κάθε μία από τις κορυφές v :
 - υπολογίζεται το σύνολο $B_{D/7}^{out}(v)$.
 - Εάν ισχύει $|B_{D/7}^{out}(v)| \leq m^a$ για κάποια κορυφή v , τότε υπολογίζεται το σύνολο $B_{D/7}^{out+}(v)$ και ελέγχεται εάν για κάποια από τις κορυφές u του $B_{D/7}^{out+}(v)$ η εκκεντρότητα είναι τουλάχιστον $4D/7$. Εάν υπάρχει τέτοια κορυφή u τότε ο αλγόριθμος αποδέχεται (accept).
- Για κάθε μία από τις κορυφές v :
 - υπολογίζεται το σύνολο $B_{D/7}^{in}(v)$.
 - Εάν ισχύει $|B_{D/7}^{in}(v)| \leq m^a$ για κάποια κορυφή v , τότε υπολογίζεται το σύνολο $B_{D/7}^{in+}(v)$ και ελέγχεται εάν για κάποια από τις κορυφές u του συνόλου $B_{D/7}^{in+}(v)$ η εκκεντρότητα είναι τουλάχιστον $4D/7$. Εάν υπάρχει τέτοια κορυφή u , τότε ο αλγόριθμος αποδέχεται (accept).
- Επιλέγουμε τυχαίο δείγμα από $4m^{1-a} \log m$ ομοιόμορφα κατανεμημένες κορυφές και τις τοποθετούμε στο σύνολο \hat{S} .

Έστω:

- $S^{\text{out}} = \{s \in \hat{S} : |B_{2D/7}^{\text{out}}(s)| \leq m^{1-a}\}$ και
- $S^{\text{in}} = \{s \in \hat{S} : |B_{2D/7}^{\text{in}}(s)| \leq m^{1-a}\}.$

Υπολογίζουμε τα σύνολα:

- $B_{2D/7}^{\text{out}}(s)$ και $B_{2D/7}^{\text{out}+}(s)$ για κάθε $s \in S^{\text{out}}$
- $B_{2D/7}^{\text{in}}(s)$ και $B_{2D/7}^{\text{in}+}(s)$ για κάθε $s \in S^{\text{in}}.$
- Δημιουργούμε τους πίνακες:
 - A^{out} διάστασης $|S^{\text{out}}| \times n$, ο οποίος έχει μία γραμμή για κάθε μία κορυφή s του συνόλου S^{out} και μία στήλη για κάθε μία από τις κορυφές v του γράφου. Θέτουμε:

$$A^{\text{out}}[s, v] = \begin{cases} 1, & \text{αν } v \in B_{2D/7}^{\text{out}}(s) \\ 0, & \text{διαφορετικά} \end{cases}$$

- A^{in} διάστασης $n \times |S^{\text{in}}|$, ο οποίος έχει μία γραμμή για κάθε μία από τις κορυφές v του γράφου και μία στήλη για κάθε κορυφή s του συνόλου $S^{\text{in}}.$ Θέτουμε:

$$A^{\text{in}}[v, s] = \begin{cases} 1, & \text{αν } \left\lfloor \frac{4D}{7} \right\rfloor = 2 \left\lfloor \frac{2D}{7} \right\rfloor \text{ και } v \in B_{2D/7}^{\text{in}}(s) \\ 1, & \text{αν } \left\lfloor \frac{4D}{7} \right\rfloor \neq 2 \left\lfloor \frac{2D}{7} \right\rfloor \text{ και } v \in B_{2D/7}^{\text{in}+}(s) \\ 0 & \text{διαφορετικά} \end{cases}$$

Στη συνέχεια ο αλγόριθμος:

- υπολογίζει το γινόμενο $A^{\text{out}} \cdot A^{\text{in}}$ διάστασης $|S^{\text{out}}| \times |S^{\text{in}}|$ κάνοντας χρήση πολλαπλασιασμού αραιών πινάκων (sparse matrix multiplication).
- αν το γινόμενο $A^{\text{out}} \cdot A^{\text{in}}$ περιλαμβάνει τουλάχιστον ένα μηδενικό, τότε ο αλγόριθμος A_D αποδέχεται (reject), διαφορετικά απορρίπτει (reject).

Όπως προκύπτει εύκολα από την περιγραφή των βημάτων του αλγορίθμου, για είσοδο έναν θετικό ακέραιο D , ο αλγόριθμος A_D είτε αποδέχεται (accept), είτε απορρίπτει (reject).

Για μικρές τιμές του D ο αλγόριθμος αποδέχεται, αλλά κάποια στιγμή όταν μεγαλώσει αρκετά η τιμή D , ο αλγόριθμος απορρίπτει και η τιμή αυτή αποτελεί μία προσεγγιστική τιμή για τη διάμετρο του γράφου. Θα μπορούσαν επομένως να δοκιμάζονται διάφορες τιμές του D , αυξάνοντας κάθε φορά την τιμή του D κατά 1, μέχρι να προκύψει μία τιμή του D για την οποία ο αλγόριθμος απορρίπτει. Στην περίπτωση αυτή έχει υπολογιστεί μία 7/4-προσέγγιση της διαμέτρου του αρχικού γράφου G .

Για να είναι πιο αποδοτική η πιο πάνω διαδικασία, θα μπορούσε εναλλακτικά να εφαρμοστεί δυαδική αναζήτηση με τον πιο κάτω τρόπο:

- Έστω το D^* η πραγματική διάμετρος του γράφου.
- Θέτουμε αρχικά $hi=n$ και $lo=0$, όπου n το πλήθος κορυφών του γράφου.
- Επαναλαμβάνουμε τα πιο κάτω βήματα για όσο είναι $hi-lo>1$.
 - Θέτουμε $mid = \left\lfloor \frac{hi+lo}{2} \right\rfloor$.
 - Εκτελούμε τον αλγόριθμο A_D για $D=mid$.
 - Εάν ο αλγόριθμος A_D αποδεχτεί (accept) τότε θέτουμε $lo=mid$, διαφορετικά θέτουμε $hi=mid$.
- Όταν τερματίσει ο βρόχος, επιστρέφουμε σαν αποτέλεσμα το lo

Σε κάθε βήμα της πιο πάνω διαδικασίας πάντοτε ισχύει

$$hi \geq D^* - 1$$

και

$$lo \leq \frac{7D^*}{4}$$

Η τιμή lo που επιστρέφει ο αλγόριθμος αποτελεί μία 7/4-προσέγγιση της πραγματικής διαμέτρου D^* .

3.3.3 Διαδικασία κατασκευής γράφου μέγιστου βαθμού 3

Για κάθε έναν γράφο $G=(V,E)$, μπορεί να κατασκευαστεί ένας γράφος $G'=(V',E')$ ο οποίος έχει την ίδια διάμετρο αλλά ταυτόχρονα καμία από τις κορυφές του δεν έχει βαθμό μεγαλύτερη από 3. Για να γίνει αυτό, αρκεί να θεωρήσουμε ότι στον νέο γράφο G' έχουμε μία κορυφή για κάθε μία από τις ακμές του αρχικού γράφου G . Με άλλα λόγια ισχύει $|V'| = 2|E|$, αφού σε οποιονδήποτε γράφο, το άθροισμα των βαθμών όλων των κορυφών ισούται με το διπλάσιο του πλήθους των ακμών (*λήμμα της Χειραψίας*). Στον γράφο G' , οι κορυφές που προκύπτουν από μία κορυφή του G , θεωρούμε ότι συνδέονται μεταξύ τους με ακμή μηδενικού βάρους, ενώ όλες οι υπόλοιπες ακμές θεωρούμε ότι έχουν βάρος ίσο με τη μονάδα. Με τον τρόπο αυτόν, είναι εύκολο να αποδειχθεί ότι ο νέος αυτός γράφος G' έχει διάμετρο ίση με τη διάμετρο του G , δηλαδή ισχύει

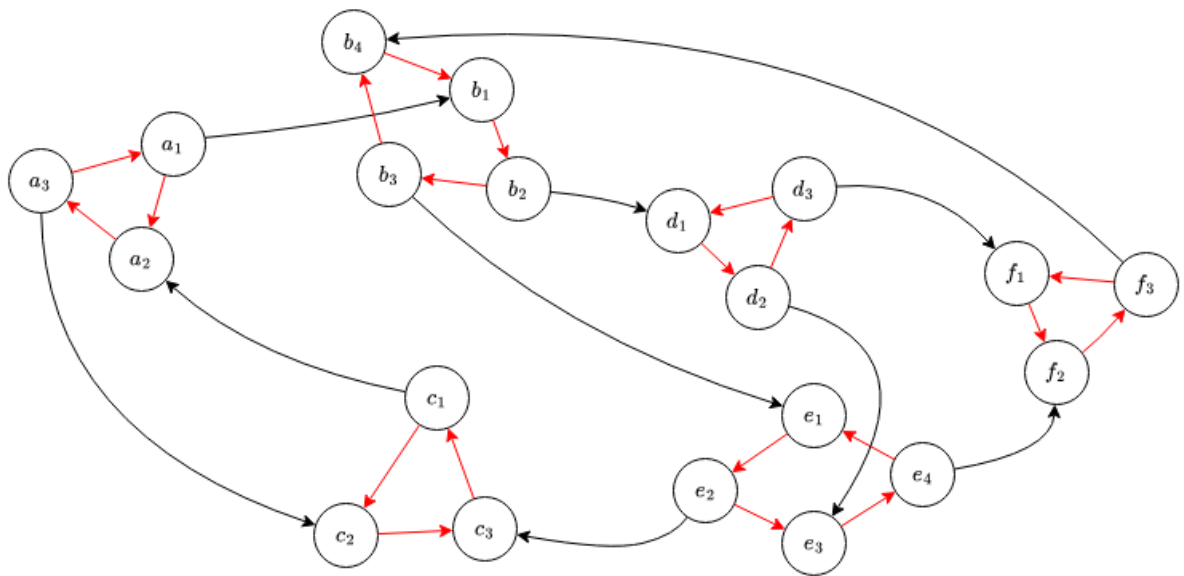
$$diam(G') = diam(G)$$

Για παράδειγμα, για τον κατευθυνόμενο γράφο στο Σχήμα 1, προκύπτει ο γράφος που εμφανίζεται στο **Σχήμα 9**. Στον νέο γράφο G' θεωρούμε ότι οι κόκκινες ακμές έχουν βάρος ίσο με 0, ενώ οι ακμές με μαύρο χρώμα έχουν βάρος ίσο με 1. Για παράδειγμα, η κορυφή b στον αρχικό γράφο είχε βαθμό ίσο με 4 και στο νέο γράφο έχουμε τις κορυφές b_1, b_2, b_3 και b_4 , οι οποίες σχηματίζουν κατευθυνόμενο κύκλο με ακμές μηδενικού βάρους. Στον νέο γράφο G' καμία κορυφή δεν έχει βαθμό μεγαλύτερο από 3 και η διάμετρος του νέου γράφου G' , είναι ίση με τη διάμετρο του αρχικού γράφου G , δεδομένου ότι οι ακμές που προστέθηκαν έχουν μηδενικό βάρος (ακμές με κόκκινο χρώμα).

Για τον αρχικό γράφο G θεωρούμε ότι έχουμε τον πίνακα γειτνίασης που εμφανίζεται στο Σχήμα 2. Για τον νέο αυτόν γράφο G' , δεδομένου ότι υπάρχουν ακμές με μηδενικό βάρος, θεωρούμε ότι ο πίνακας περιλαμβάνει τιμή ∞ για κάθε ζευγάρι κορυφών που δεν συνδέονται μεταξύ τους με ακμή. Θα έχουμε επομένως τον πίνακα γειτνίασης ο οποίος εμφανίζεται στο **Σχήμα 10**.

Ο νέος πίνακας γειτνίασης είναι διάστασης 20×20 , αφού το άθροισμα των βαθμών όλων των κορυφών του γράφου G (Σχήμα 1) είναι ίσο με 20. Παρατηρούμε ότι κάθε γραμμή και κάθε στήλη του νέου πίνακα περιλαμβάνει 2 μηδενικά, αφού κάθε

κορυφή του νέου γράφου συμμετέχει σε κατευθυνόμενο κύκλο με ακμές μηδενικού βάρους. Επίσης, σε κάθε γραμμή και στήλη περιλαμβάνεται ένας το πολύ άστος, αφού κάθε κορυφή συνδέεται με μία άλλη κορυφή με τη βοήθεια ακμής βάρους ίσου με 1.



Σχήμα 9: Γράφος ίδιας διαμέτρου με μέγιστο βαθμό κορυφών ίσο με 3

Η υλοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού Python και κάποιες δοκιμαστικές εκτελέσεις παρουσιάζονται στο επόμενο κεφάλαιο.

0	0	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	0	0	∞	∞	∞	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	0	0	∞	∞	∞	∞	1	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	0	0	∞	∞	∞	∞	∞	∞	1	∞	∞	∞	∞	∞
∞	∞	∞	0	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	1	∞	∞	∞	∞	∞	0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	0	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞	0	∞	∞	∞	∞	1	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0
∞	∞	∞	∞	∞	∞	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	0

Σχήμα 10: Πίνακας γειτνίασης νέου γράφου G'

4. Υλοποίηση του προσεγγιστικού αλγορίθμου

4.1 Τρόπος υλοποίησης

Η υλοποίηση του προσεγγιστικού αλγορίθμου που περιγράφηκε στο προηγούμενο κεφάλαιο έγινε σε γλώσσα προγραμματισμού Python. Η Python είναι μία πολύ δημοφιλής γλώσσα προγραμματισμού κατά τα τελευταία χρόνια, λόγω της απλότητάς και της φιλικότητάς της. Δημιουργήθηκε από τον Guido van Rossum και κυκλοφόρησε το 1991. Είναι γλώσσα προγραμματισμού γενικού σκοπού, μπορεί να χρησιμοποιηθεί δηλαδή για την ανάπτυξη διαφόρων τύπων εφαρμογών, τόσο από αρχάριους προγραμματιστές, όσο και από έμπειρους επαγγελματίες. Διαθέτει πλήθος έτοιμων βιβλιοθηκών, καθώς και μία μεγάλη κοινότητα η οποία υποστηρίζει την εξέλιξη της γλώσσας και βοηθά τους προγραμματιστές στην ανάπτυξη των εφαρμογών.

Η python είναι διερμηνευόμενη γλώσσα (interpreted language). Οι εντολές εκτελούνται δηλαδή η μία μετά την άλλη, χωρίς να έχει γίνει προηγουμένως μεταγλώττιση του προγράμματος, όπως γίνεται με γλώσσες όπως είναι η C και η C++. Κατά την πρώτη εκτέλεση όμως του προγράμματος, στο παρασκήνιο ο διερμηνευτής μεταφράζει τον πηγαίο κώδικα (source code) της Python και δημιουργεί αρχεία τα οποία περιλαμβάνουν τα Bytecodes της εφαρμογής (αρχεία μορφής .pyc στον φάκελο __pycache__), τα οποία χρησιμοποιούνται για γρήγορες μελλοντικές εκτελέσεις. Ο κώδικας Bytecode αποτελεί μία ενδιάμεση μορφή ανάμεσα στην αρχική μορφή των εντολών και την εκτελέσιμη μορφή. Τα Bytecodes εκτελούνται από τον διερμηνευτή γραμμή προς γραμμή, ο οποίος λειτουργεί σαν εικονική μηχανή Python (Python Virtual Machine – PVM) γραμμή προς γραμμή, κάτι αντίστοιχο με την Java Virtual Machine (JVM). Σε αρκετά











μεγάλο βαθμό, η διαδικασία αυτή μοιάζει με τον τρόπο λειτουργίας της Java, αν και οι δύο γλώσσες διαφέρουν στον τρόπο δημιουργίας του ενδιαμέσου κώδικα.

Η Python παρουσιάζει διάφορα χαρακτηριστικά τα οποία την έχουν κάνει μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού, όπως:

- Απλή σύνταξη.
- Ευκολία εκμάθησης.
- Μεγάλη και πολύ ενεργή κοινότητα.
- Ανεξαρτησία από την πλατφόρμα. Ένα πρόγραμμα Python μπορεί να εκτελεστεί χωρίς κάποιες αλλαγές σε πρακτικά οποιοδήποτε λειτουργικό σύστημα.
- Πολλές έτοιμες βιβλιοθήκες.

Σύμφωνα με τον δείκτη TIOBE (The Information is Only Brilliant Because it is Evidently True), η γλώσσα Python κατατάσσεται στην κορυφή της λίστας των περισσότερο δημοφιλών γλωσσών προγραμματισμού για τον Απρίλιο του 2025 (TIOBE, 2025). Μάλιστα, η Python βρίσκονταν στην ίδια θέση κατά το προηγούμενο έτος, παρουσιάζοντας επίσης αυξητικές τάσεις, όπως φαίνεται στην **Εικόνα 3**. Ο δείκτης TIOBE προσφέρει μια ένδειξη της δημοτικότητας μιας γλώσσας προγραμματισμού και χρησιμοποιείται συχνά για να παρακολουθήσει τις τάσεις και τις εξελίξεις στον χώρο της δημοφιλίας των γλωσσών προγραμματισμού. Ο δείκτης δεν αποτελεί φυσικά μία μέτρηση της ποιότητας μίας γλώσσας ούτε αποτελεί κάποια απόλυτη και μη αμφισβητήσιμη μέτρηση. Δείχνει απλά τη δημοφιλία της με βάση τις αναζητήσεις που έχουν γίνει στο Διαδίκτυο.

Τα χαρακτηριστικά της γλώσσας προγραμματισμού Python και η δημοφιλία της γλώσσας, οδήγησαν στην απόφαση για την υλοποίηση του αλγορίθμου με τη βοήθεια της γλώσσας αυτής.

Apr 2025	Apr 2024	Change	Programming Language		Ratings	Change
1	1			Python	23.08%	+6.67%
2	3	▲		C++	10.33%	+0.56%
3	2	▼		C	9.94%	-0.27%
4	4			Java	9.63%	+0.69%
5	5			C#	4.39%	-2.37%
6	6			JavaScript	3.71%	+0.82%
7	7			Go	3.02%	+1.17%
8	8			Visual Basic	2.94%	+1.24%
9	11	▲		Delphi/Object Pascal	2.53%	+1.06%
10	9	▼		SQL	2.19%	+0.57%

Εικόνα 3: Λίστα με τις 10 πιο δημοφιλείς γλώσσες προγραμματισμού (TIOBE)

4.2 Περιβάλλον ανάπτυξης

Υπάρχουν διάφορα Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης (Integrated Development Environments – IDEs) για την γλώσσα προγραμματισμού Python, όπως είναι το IDLE, το PyCharm, το Spyder, το Visual Studio Code, το Jupyter κλπ. Κάποια από αυτά έχουν αναπτυχθεί ειδικά για την Python, ενώ άλλα υποστηρίζουν και άλλες γλώσσες προγραμματισμού, πέρα από την Python. Επιπρόσθετα, υπάρχουν και διάφορα Online περιβάλλοντα για την ανάπτυξη προγραμμάτων Python. Για την υλοποίηση του αλγορίθμου επιλέχθηκε το περιβάλλον PyCharm.

Το PyCharm είναι ένα από τα πιο δημοφιλή περιβάλλοντα προγραμματισμού, όπως φαίνεται στην **Εικόνα 4** (PYPL, 2025). Η λίστα που εμφανίζεται στην εικόνα αυτή δείχνει το ποσοστό χρήστης για περιβάλλοντα ανάπτυξης εφαρμογών, όπως καταγράφεται από τον δείκτη PYPL (PopularitY of Programming Language Index).

Ο δείκτης αυτός καταγράφει τη δημοφιλή με βάση τις αναζητήσεις στο Διαδίκτυο για υλικό και εκπαιδευτικό υλικό (tutorials) που πραγματοποιούν οι χρήστες.

Worldwide, Apr 2025 :				
Rank	Change	IDE	Share	1-year trend
1		Visual Studio	28.48 %	+1.3 %
2		Visual Studio Code	15.27 %	+1.7 %
3	↑↑	Android Studio	11.2 %	+1.3 %
4		pyCharm	10.53 %	-0.4 %
5	↓↓	Eclipse	10.06 %	-1.7 %
6		IntelliJ	7.22 %	-0.3 %
7		NetBeans	3.57 %	-0.4 %
8		Xcode	2.88 %	-0.2 %
9		RStudio	2.86 %	-0.0 %
10		Sublime Text	2.34 %	-0.2 %
11		Atom	1.9 %	-0.5 %
12		Code::Blocks	1.54 %	-0.1 %

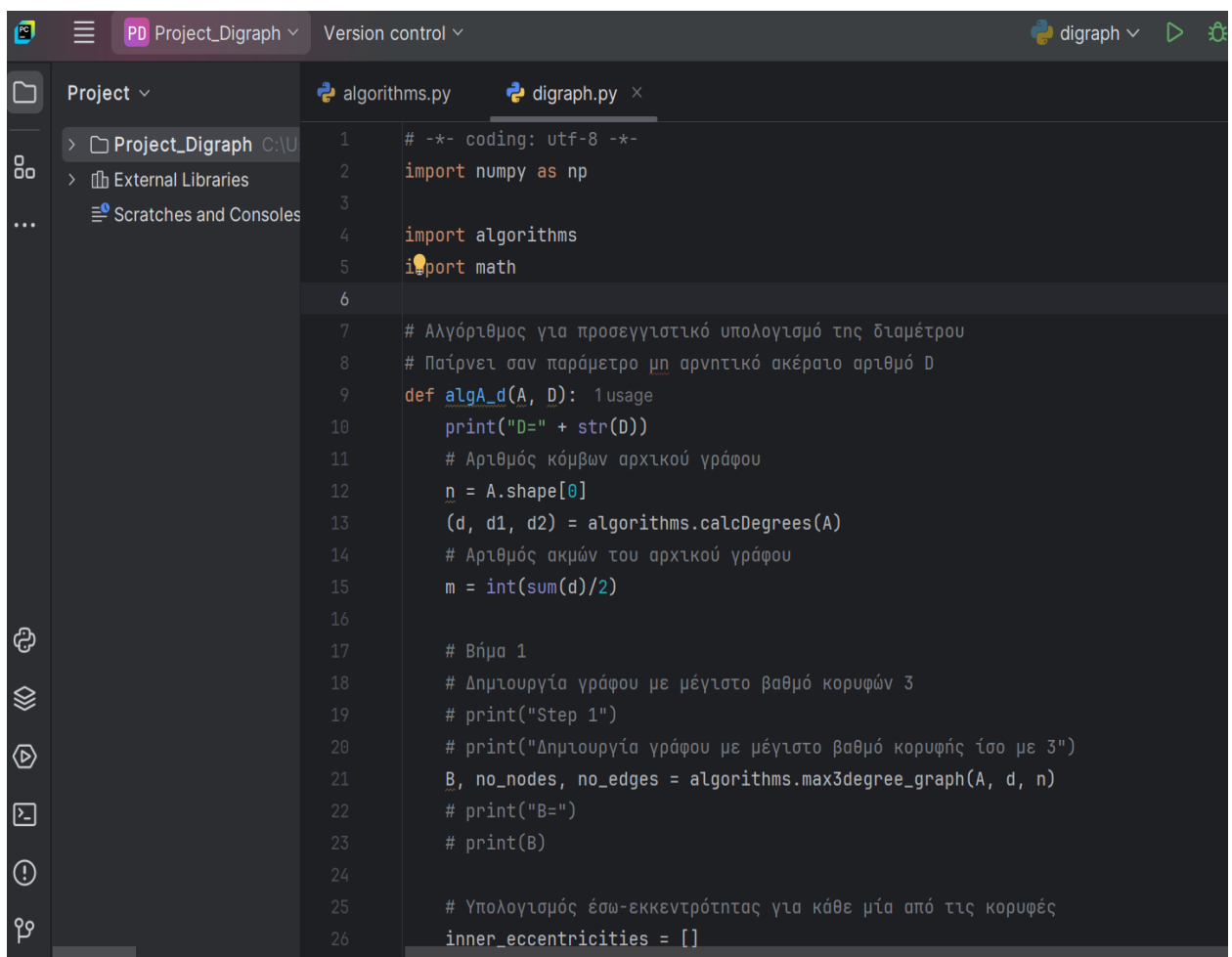
Εικόνα 4: Λίστα με τα πιο δημοφιλή περιβάλλοντα προγραμματισμού (PYPL)

Το PyCharm είναι ένα πολύ δημοφιλές περιβάλλον ανάπτυξης εφαρμογών, ειδικά σχεδιασμένο για τη δημιουργία κώδικα σε γλώσσα Python. Το περιβάλλον αναπτύχθηκε από την εταιρεία JetBrains και κάποια από τα χαρακτηριστικά που κάνουν δημοφιλή την εφαρμογή είναι:

- Έξυπνη συμπλήρωση κώδικα (Intelligent Code Completion)
- Έλεγχος του κώδικα σε πραγματικό χρόνο (on-the-fly analysis of the source code), εμφανίζοντας άμεσα λάθη και προειδοποιήσεις.
- Εύκολες τροποποιήσεις στον κώδικα (Code Refactoring).
- Debugger για την εκσφαλμάτωση του κώδικα.

- Εργαλεία για έλεγχο (testing)
- Δυνατότητες παραμετροποίησης του περιβάλλοντος.

Η εταιρεία JetBrains που αναπτύσσει το περιβάλλον προσφέρει την έκδοση PyCharm Pro και την έκδοση PyCharm Community Edition (CE). Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε η έκδοση PyCharm Community Edition έκδοση 2024.2.2, όπως φαίνεται στην **Εικόνα 5**.



Εικόνα 5: Το περιβάλλον του PyCharm

4.3 Βιβλιοθήκες της Python

Οι βιβλιοθήκες της Python αποτελούν τμήματα έτοιμου επαναχρησιμοποιήσιμου κώδικα, ο οποίος έχει γραφτεί από κάποιους άλλους προγραμματιστές και είναι διαθέσιμος για την χρήση τους κατά την ανάπτυξη των εφαρμογών. Με την χρήση έτοιμων βιβλιοθηκών μειώνεται ο απαιτούμενος χρόνος για την ανάπτυξη μίας εφαρμογής, ενώ μειώνεται ταυτόχρονα η πιθανότητα εισαγωγής προγραμματιστικών σφαλμάτων (bugs) στον κώδικα, δεδομένου ότι συνήθως ο κώδικας που περιλαμβάνεται στις βιβλιοθήκες έχει ήδη ελεγχθεί διεξοδικά για πιθανά σφάλματα.

Η Python υποστηρίζει πλήθος βιβλιοθηκών, οι οποίες αφορούν διάφορους χώρους, όπως είναι η ανάλυση δεδομένων, η αριθμητική ανάλυση, η μηχανική μάθηση, η τεχνητή νοημοσύνη, η δημιουργία γραφικής διεπαφής (Graphical User Interface – GUI), η δικτύωση (networking), η επεξεργασία εικόνας (image processing) κλπ. Δεν υπάρχει κάποιος επίσης κεντρικός κατάλογος για τις βιβλιοθήκες της Python και συνεχώς δημιουργούνται νέες βιβλιοθήκες. Μία βιβλιοθήκη μπορεί να συμπεριληφθεί στον κώδικα μίας εφαρμογής Python με τη βοήθεια της εντολής “import”.

Στο αποθετήριο PyPI (Python Package Index) περιλαμβάνονται πολλά πακέτα Python) τα οποία μοιράζονται ανάμεσα στην κοινότητα των προγραμματιστών Python και μπορούν να επαναχρησιμοποιηθούν από οποιονδήποτε προγραμματιστή. Δεν είναι όλες οι βιβλιοθήκες της Python καλά συντηρημένες και οργανωμένες, οι πιο δημοφιλείς όμως από αυτές είναι αξιόπιστες.

Οι βιβλιοθήκες της Python οι οποίες χρησιμοποιήθηκαν κατά την υλοποίηση του αλγορίθμου ήταν οι πιο κάτω:

- **NumPy** (Numerical Python). Είναι μία από τις βασικές βιβλιοθήκες της Python και μπορεί να χρησιμοποιηθεί για την εκτέλεση πολύπλοκων αριθμητικών υπολογισμών και για απευθείας πράξεις ανάμεσα σε πολυδιάστατους πίνακες. Υποστηρίζει και πολλές άλλες μαθηματικές λειτουργίες από χώρους όπως η γραμμική Άλγεβρα, η Στατιστική, διανυσματικές συναρτήσεις κλπ. Η βιβλιοθήκη NumPy χρησιμοποιήθηκε κατά την ανάπτυξη της εφαρμογής για την αναπαράσταση γράφων με τη

μορφή πίνακα γειτνίασης (adjacency matrix), δηλαδή δισδιάστατων πινάκων που αποτυπώνουν τις συνδέσεις ανάμεσα στις κορυφές ενός γράφου.

- **math.** Η βιβλιοθήκη `math` προσφέρει μαθηματικές συναρτήσεις και χρησιμοποιήθηκε για τον λόγο αυτόν κατά την ανάπτυξη της εφαρμογής (λογαριθμικές συναρτήσεις κλπ.)
- **random.** Η βιβλιοθήκη `random` επιτρέπει την εύκολη παραγωγή τυχαίων αριθμών. Χρησιμοποιήθηκε για την τυχαία επιλογή με ομοιόμορφο τρόπο κάποιων από τις κορυφές του γράφου, όπως περιγράφεται από τον αλγόριθμο.
- **heapq.** Η βιβλιοθήκη `heapq` δίνει τη δυνατότητα χρήση ουράς προτεραιότητας (Priority Queue) από την εφαρμογή. Η χρήση τέτοιας ουράς είναι χρήσιμη κατά την ανάπτυξη αλγορίθμων, όπως είναι ο αλγόριθμος του Dijkstra, έτσι ώστε να επιλέγεται σε κάθε επανάληψη η κατάλληλη κορυφή του γράφου.

4.4 Παρουσίαση της εφαρμογής

Η εφαρμογή υλοποιεί τα βήματα του αλγορίθμου, όπως αυτά περιγράφηκαν στις προηγούμενες ενότητες. Πιο κάτω περιγράφονται κάποια από τα τμήματα του κώδικα που αναπτύχθηκαν για την υλοποίηση των βημάτων του αλγορίθμου.

Αρχικά δημιουργείται ένας γράφος ο οποίος έχει την ίδια διάμετρο με τον αρχικό γράφο, αλλά καμία από τις κορυφές δεν έχει βαθμό μεγαλύτερο από 3. Αυτό έχει γίνει με τη βοήθεια της συνάρτησης "`max3degree`" που εμφανίζεται στην **Εικόνα 3**. Η συνάρτηση χρησιμοποιεί και άλλες βοηθητικές συναρτήσεις, οι οποίες δεν εμφανίζονται στην εικόνα.

```

# A είναι ο πίνακας γειτνίασης του γράφου
# d είναι το διάνυσμα βαθμών των κορυφών του γράφου
# n είναι το πλήθος των κορυφών
# το πλήθος κορυφών του νέου γράφου είναι ίσο με nodes
# το πλήθος ακμών (τόξων) του νέου γράφου είναι ίσο με edges
def max3degree_graph(A, d, n):
    # Αθροισμα βαθμών που είναι ίσο με το πλήθος κορυφών του νέου γράφου
    nodes = sum(d)
    # Αρχικά δημιουργούμε πίνακα με τιμές ίσες με άπειρο
    B = np.ones((nodes, nodes), dtype=int)
    B = math.inf*B
    # Ελέγχουμε όλες τις ακμές του αρχικού πίνακα γειτνίασης
    for i in range(n):
        for j in range(n):
            if A[i,j] == 1:
                # Εύρεση κατάλληλης θέσης στον νέο πίνακα γειτνίασης
                k = getNext(B, d, nodes, i, i)
                l = getNext(B, d, nodes, k, j)
                # Τοποθέτηση 1 (ακμής) στον νέο πίνακα γειτνίασης
                B[k,l] = 1
    # Δημιουργία των ακμών μηδενικού βάρους
    for i in range(n):
        # Εύρεση σωστής θέσης
        pos = 0
        for k in range(i):
            pos += d[k]
        for j in range(d[i]):
            for k in range(d[i]):
                # Κάθε νέα (τεχνητή) κορυφή συνδέεται με ακμή με την επόμενη
                if (k == (j+1)%d[i]):
                    B[pos+j,pos+k] = 0
    # Η κύρια διαγώνιος περιλαμβάνει μηδενικά
    for i in range(nodes):
        B[i,i] = 0
    # Το πλήθος των ακμών
    edges = 3 * nodes / 2
    # Επιστροφή του νέου πίνακα γειτνίασης και του πλήθους κορυφών
    return B, nodes, edges

```

Εικόνα 6: Δημιουργία γράφου ίδιας διαμέτρου με μέγιστο βαθμό κορυφών 3

```

# Συνάρτηση που επιστρέφει την έσω-εκκεντρότητα της κορυφής node
# για έναν γράφο με πίνακα γειτνίασης A και πλήθος κορυφών n
def inner_eccentricity(node, A, n):
    distances = shortest_distances_to_target(node, A, n)
    return max(distances.values())

# Συνάρτηση που επιστρέφει την έξω-εκκεντρότητα της κορυφής node
# για έναν γράφο με πίνακα γειτνίασης A και πλήθος κορυφών n
def outer_eccentricity(node, A, n):
    distances = shortest_distances_from_source(node, A, n)
    return max(distances.values())

```

Εικόνα 7: Κώδικας για τον υπολογισμό έσω και έξω-εκκεντρότητας

Για το βήμα 2 απαιτείται ο υπολογισμός της έσω και της έξω-εκκεντρότητας κάποιων κορυφών που επιλέγονται ως τυχαίο δείγμα. Ο υπολογισμός αυτός γίνεται με τον κώδικα στην **Εικόνα 7**. Οι συναρτήσεις που εμφανίζονται στην εικόνα κάνουν χρήση των συναρτήσεων "shortest_distance_to_target" και "shortest_distance_from_source" οι οποίες κάνουν χρήση του αλγόριθμου Dijkstra για τον υπολογισμό των συντομότερων αποφάσεων, όπως περιγράφηκε σε προηγούμενη παράγραφο. Στο βήμα αυτό επίσης θα πρέπει να επιλεγούν $4m^a \log m$ κορυφές του γράφου με τυχαίο τρόπο. Αυτό γίνεται με χρήση της βιβλιοθήκης random, όπως φαίνεται στον κώδικα στην **Εικόνα 9**.

Στο βήμα-3 του αλγορίθμου απαιτείται ο υπολογισμός των συνόλων $B_{D/7}^{out}$ και $B_{D/7}^{out+}$ για κάποιες από τις κορυφές του γράφου, κάτι που μπορεί να γίνει με τη βοήθεια της αναζήτησης κατά Πλάτος (Breadth-first search), όπως φαίνεται στην **Εικόνα 8** και στην **Εικόνα 10** αντίστοιχα. Αν το πλήθος $|B_{D/7}^{out}|$ των κορυφών στο σύνολο $B_{D/7}^{out}$ είναι μικρότερο ή ίσο από m^a , τότε ελέγχεται αν υπάρχει κορυφή του συνόλου $B_{D/7}^{out+}$ με εκκεντρότητα $\frac{4D}{7}$, έτσι ώστε ο αλγόριθμος να αποδεχθεί.

```

# Συνάρτηση που παίρνει μία λίστα με κορυφές list
# και επιστρέφει λίστα με k από αυτές επιλεγμένες τυχαία
def choose_random(list, k):
    list1 = []
    for i in range(k):
        x = random.choice(list)
        list1.append(x)

    return list1

```

Εικόνα 9: Δημιουργία τυχαίου δείγματος κορυφών

```

# Υπολογισμός πλήθους κορυφών οι οποίες απέχουν από την κορυφή v
# απόσταση το πολύ r με τη βοήθεια αναζήτησης κατά πλάτος
def find_Bout1(A, n, v, r):
    # Υπολογισμός συντομότερων αποστάσεων από την κορυφή v
    set_of_nodes = set()
    for i in range(n):
        d = bfs1(A, n, v, i)
        if(d <= r):
            set_of_nodes.add(d)
    return set_of_nodes, len(set_of_nodes)

```

Εικόνα 8: Δημιουργία συνόλου B^{out}

```

# Υπολογισμός πλήθους κορυφών οι οποίες απέχουν από την κορυφή v
# απόσταση το πολύ r με τη βοήθεια αναζήτησης κατά πλάτος
def find_Bout_plus1(A, n, v, r):
    # Υπολογισμός συντομότερων αποστάσεων από την κορυφή v
    set_of_nodes = set()
    for i in range(n):
        d = bfs1(A, n, v, i)
        if(d <= r):
            set_of_nodes.add(d)
    # προσθέτουμε και τους γείτονες αν η απόσταση είναι ίση με r
    if (d == r):
        for u in range(n):
            if (A[v, u] == 1):
                set_of_nodes.add(u)
    return set_of_nodes, len(set_of_nodes)

```

Εικόνα 10: Δημιουργία συνόλου B^{out+}

Όμοια, στο βήμα-4 του αλγορίθμου απαιτείται ο υπολογισμός των συνόλων $B_{D/7}^{in}$ και $B_{D/7}^{in+}$ για κάποιες από τις κορυφές του γράφου, όπως φαίνεται στον κώδικα στην **Εικόνα 11** και στην **Εικόνα 12** αντίστοιχα. Αν το πλήθος $|B_{D/7}^{in}|$ των κορυφών στο σύνολο $B_{D/7}^{in}$ είναι μικρότερο ή ίσο από m^a , τότε ελέγχεται αν υπάρχει κορυφή του συνόλου $B_{D/7}^{in+}$ με εκκεντρότητα $\frac{4D}{7}$, έτσι ώστε ο αλγόριθμος να αποδεχθεί.

Στο βήμα-5 δημιουργείται νέο τυχαίο δείγμα με $4m^{1-a} \log m$ κορυφές ώστε να δημιουργηθεί το σύνολο \hat{S} . Υπολογίζονται επίσης τα σύνολα

- $S^{out} = \{s \in \hat{S}: |B_{2D/7}^{out}(s)| \leq m^{1-a}\}$ και
- $S^{in} = \{s \in \hat{S}: |B_{2D/7}^{in}(s)| \leq m^{1-a}\}.$

Στη συνέχεια υπολογίζονται, με τρόπο παρόμοιο με το προηγούμενο βήμα, τα σύνολα:

- $B_{2D/7}^{out}(s)$ και $B_{2D/7}^{out+}(s)$ για κάθε $s \in S^{out}$
- $B_{2D/7}^{in}(s)$ και $B_{2D/7}^{in+}(s)$ για κάθε $s \in S^{in}.$

```

# Υπολογισμός πλήθους κορυφών u για τις οποίες η απόσταση προς την κορυφή v
# το πολύ r
def find_Bin1(A, n, v, r):
    # Υπολογισμός συντομότερων αποστάσεων από την κορυφή v
    set_of_nodes = set()
    for i in range(n):
        d = bfs1(A, n, i, v)
        if(d <= r):
            set_of_nodes.add(d)
    return set_of_nodes, len(set_of_nodes)

```

Εικόνα 11: Υπολογισμός συνόλου B^{in}

```

# Συνάρτηση που επιστρέφει τις κορυφές του συνόλου Bin μίας κορυφής v
# μαζί με αυτές που γειτονεύουν μαζί τους
def find_Bin_plus1(A, n, v, r):
    # Υπολογισμός συντομότερων αποστάσεων από την κορυφή v
    set_of_nodes = set()
    for i in range(n):
        d = bfs1(A, n, i, v)
        if(d <= r):
            set_of_nodes.add(d)
    # προσθέτουμε και τους γείτονες αν η απόσταση είναι ίση με r
    if(d == r):
        for u in range(n):
            if (A[u, v] == 1):
                set_of_nodes.add(u)
    return set_of_nodes, len(set_of_nodes)

```

Εικόνα 12: Υπολογισμός συνόλου B^{in+}

Στο βήμα-6 του αλγορίθμου δημιουργούνται τους πίνακες:

- A^{out} διάστασης $|S^{out}| \times n$, με τη βοήθεια του τύπου:

$$A^{out}[s, v] \begin{cases} 1, & \text{αν } v \in B_{2D/7}^{out}(s) \\ 0, & \text{διαφορετικά} \end{cases}$$

- A^{in} διάστασης $n \times |S^{in}|$, με τη βοήθεια του τύπου:

$$A^{in}[v, s] \begin{cases} 1, & \text{αν } \left\lfloor \frac{4D}{7} \right\rfloor = 2 \left\lfloor \frac{2D}{7} \right\rfloor \text{ και } v \in B_{2D/7}^{in}(s) \\ 1, & \text{αν } \left\lfloor \frac{4D}{7} \right\rfloor \neq 2 \left\lfloor \frac{2D}{7} \right\rfloor \text{ και } v \in B_{2D/7}^{in+}(s) \\ 0 & \text{διαφορετικά} \end{cases}$$

Στην **Εικόνα 13** εμφανίζεται ο κώδικας για τη δημιουργία των συνόλων A^{out} και A^{in} . Στη συνέχεια υπολογίζεται το γινόμενο $A^{out} \cdot A^{in}$. Ο πολλαπλασιασμός αυτός μπορεί να γίνει με χρήση αλγορίθμων πολλαπλασιασμού αραιών πινάκων, έτσι να μπορεί να εκτελεστεί γρηγορότερα. Ένα το γινόμενο των πινάκων περιλαμβάνει μηδενικά στοιχεία, τότε ο αλγόριθμος αποδέχεται, διαφορετικά ο αλγόριθμος απορρίπτει για τη συγκεκριμένη τιμή του D.

Ο αλγόριθμος θα πρέπει να εκτελεστεί για διάφορες τιμές του D, κάτι που μπορεί να γίνει με τη βοήθεια δυαδικής αναζήτησης (binary search), όπως περιγράφηκε στις προηγούμενες παραγράφους. Ο κώδικας για την εφαρμογή της δυαδικής αναζήτησης μέχρις ότου να βρεθεί η επιθυμητή τιμή για το D, η οποία αντιστοιχεί στην προσέγγιση της διαμέτρου που επιστρέφει ο αλγόριθμος, εμφανίζεται στην **Εικόνα 14**. Ο αλγόριθμος επιστρέφει την τελική τιμή της μεταβλητής lo, όπως προκύπτει από τον κώδικα της δυαδικής αναζήτησης. Ο κώδικας ο οποίος εκτελεί τα βήματα του αλγορίθμου A_D , υπάρχει στο παράρτημα.

```

# Δημιουργία πίνακα Aout
Aout = np.zeros(shape=(len(Sout), no_nodes), dtype=int)
for s in Sout:
    Bout, count = algorithms.find_Bout1(B, no_nodes, s, 2*D/7)
    for v in range(no_nodes):
        if v in Bout:
            Aout[Sout.index(s),v] = 1

# Δημιουργία πίνακα Ain
Ain = np.zeros(shape=(no_nodes, len(Sin)), dtype=int)
if int(4*D/7) == 2*int(2*D/7):
    for s in Sin:
        Bin, count = algorithms.find_Bin1(B, no_nodes, s, 2 * D / 7)
        for v in range(no_nodes):
            if v in Bin:
                Ain[v, Sin.index(s)] = 1
else:
    for s in Sin:
        Bin_plus, count = algorithms.find_Bin_plus1(B, no_nodes, s, 2 * D / 7)
        for v in range(no_nodes):
            if v in Bin_plus:
                Ain[v, Sin.index(s)] = 1

```

Εικόνα 13: Κώδικας για τα σύνολα A^{in} και A^{out}

```

lo = 0
hi = n
while hi-lo > 1:
    mid = int((lo+hi)/2)
    accept = algA_d(A, mid)
    if accept:
        lo = mid
    else:
        hi = mid

```

Εικόνα 14: Δυναδική αναζήτηση κατά την εκτέλεση του αλγορίθμου A_D

4.5 Δοκιμή της εφαρμογής

Για τη δοκιμή της εφαρμογής για διάφορα δίκτυα αναπτύχθηκε κώδικας σε Python ο οποίος δημιουργεί με τυχαίο τρόπο έναν κατευθυνόμενο γράφο, ορίζοντας το επιθυμητό πλήθος κορυφών και τον επιθυμητό βαθμό πυκνότητας του γράφου. Ο κώδικας για τη δημιουργία ενός τυχαίου κατευθυνόμενου γράφου εμφανίζεται στην **Εικόνα 15**. Η συνάρτηση "create_directed_graph" παίρνει σαν παραμέτρους το επιθυμητό πλήθος n των κορυφών του γράφου και την επιθυμητή πυκνότητα p , σαν έναν πραγματικό αριθμό μεταξύ 0 και 1. Για παράδειγμα, εάν θέσουμε $p=0.3$, τότε η πιθανότητα δημιουργίας ακμής ανάμεσα σε 2 κορυφές είναι ίση με 0.3, δηλαδή 30%. Το πρόγραμμα μετά την εκτέλεση του επιστρέφει την προσεγγιστική τιμή της διαμέτρου του γράφου, όπως προκύπτει από τον αλγόριθμο που περιγράφηκε στις προηγούμενες παραγράφους. Σε περίπτωση που ο γράφος που δημιουργείται είναι μη συνδεδετικός, τότε η διάμετρος του θεωρείται ότι είναι άπειρη και ο αλγόριθμος επιστρέφει ανάλογο μήνυμα.

```

# Συνάρτηση που δημιουργεί έναν πίνακα nxn ως πίνακα γειτνίασης
# ενός κατευθυνόμενου γράφου
def create_directed_graph(n, p):
    A = np.zeros(shape=(n, n), dtype=int)
    for i in range(n):
        A[i,i] = 0
        for j in range(n):
            if i != j:
                x = random.uniform(0, 1)
                if x<p:
                    A[i,j] = 1
    return A

```

Εικόνα 15: Κώδικας για τη δημιουργία τυχαίου κατευθυνόμενου γράφου

Ένα παράδειγμα από την εκτέλεση του προγράμματος για τιμές $n=10$ και $p=0.3$ εμφανίζεται στην **Εικόνα 16**. Ο γράφος εμφανίζεται στην εικόνα σε μορφή πίνακα γειτνίασης και στη συνέχεια εμφανίζονται οι διάφορες τιμές του D κατά την εφαρμογή της δυαδική αναζήτησης και την επαναληπτική εκτέλεση του αλγορίθμου, όπως περιγράφηκε εκτενώς στις προηγούμενες παραγράφους. Αντίστοιχα, ένα παράδειγμα εκτέλεσης του προγράμματος για έναν γράφο με $n=20$ κορυφές και πιθανότητα δημιουργίας ακμής ίση με $p=0.2$, εμφανίζεται στην **Εικόνα 17**.

```
A=
[[0 0 0 0 0 0 1 0 0 1]
 [0 0 0 1 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 1 1 1 0 1 1]
 [0 1 1 0 0 0 0 1 0 0]
 [0 0 1 0 0 0 1 0 1 0]
 [0 0 0 1 1 0 0 0 0 1]
 [0 0 0 1 0 0 0 0 1 0]
 [0 0 0 0 1 1 1 0 0 1]
 [0 1 1 0 0 1 0 0 1 0]]
D=5
D=7
D=8
diam=7
```

Εικόνα 16: Παράδειγμα εκτέλεσης προγράμματος για γράφο με 10 κορυφές

Από τις δοκιμές της εφαρμογής οι οποίες πραγματοποιήθηκαν προκύπτει ότι ο αλγόριθμος λειτουργεί ικανοποιητικά και δίνει μία καλή προσέγγιση της διαμέτρου ενός κατευθυνόμενου γράφου χωρίς βάρη, μέσα στα επιθυμητά όρια, αφού η υπολογιζόμενη διάμετρος δεν είναι μεγαλύτερη σε καμία περίπτωση από τα $7/4$ της διαμέτρου του κάθε γράφου.

```

A=
[[0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1]
 [0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0]
 [0 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1 0 1]
 [0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1]
 [0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1]
 [1 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0 0 1 1 0]
 [0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0]
 [1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1]
 [1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 1 0 0 0]]
D=10
D=5
D=7
D=8
diam=7

```

Εικόνα 17: Παράδειγμα εκτέλεσης προγράμματος για γράφο με $n=20$ κορυφές

4.6 Πειραματική μελέτη

Στο πλαίσιο της παρούσας πτυχιακής εργασίας πραγματοποιήθηκε πειραματική μελέτη για τον έλεγχο της ακρίβειας των αποτελεσμάτων του προσεγγιστικού αλγορίθμου. Ο στόχος της πειραματικής αυτής μελέτης ήταν να γίνει σύγκριση της

πραγματικής διαμέτρου κάποιων γράφων σε σχέση με τη διάμετρο που υπολογίζει ο προσεγγιστικός αλγόριθμος ιδιαίτερα για μικρού και μεσαίου μεγέθους γράφους. Για τις ανάγκες της πειραματικής μελέτης δημιουργήθηκαν τυχαίοι κατευθυνόμενοι γράφοι με πλήθος κορυφών από 5 έως 30, με τη βοήθεια της συνάρτησης η οποία περιγράφηκε πιο πάνω (graph generator). Για κάθε τιμή του πλήθους κορυφών κατασκευάστηκαν πολλαπλά δείγματα (10 γράφοι ανά πλήθος κορυφών), έτσι ώστε να επιτευχθεί η απαραίτητη αξιοπιστία και στη συνέχεια υπολογίστηκε η μέση διάμετρος για την κάθε περίπτωση. Για την κάθε περίπτωση υπολογίστηκε η πραγματική διάμετρος με τη βοήθεια του αλγόριθμου της Κατά Πλάτος αναζήτησης και έγινε σύγκριση με τη διάμετρο που υπολογίζει ο προσεγγιστικός αλγόριθμος. Η απόλυτη τιμή του σφάλματος, για την κάθε κατηγορία γράφων, υπολογίστηκε με τη βοήθεια του τύπου:

$$\text{Σφάλμα} = \frac{|[\text{Προσεγγιστική Διάμετρος}] - [\text{Πραγματική Διάμετρος}]|}{[\text{Πραγματική Διάμετρος}]}$$

Ο πιο κάτω πίνακας δείχνει τα αποτελέσματα της πειραματικής μελέτης:

Πλήθος κορυφών	Προσεγγιστική διάμετρος	Πραγματική διάμετρος	Σφάλμα (%)
5	3,1	2,5	19,35
10	5,5	4,1	25,45
15	7,3	5,4	26,03
20	8,9	6,6	25,84
30	10,7	7,9	26,17
40	12,6	9,3	26,19

Τα αποτελέσματα της πειραματικής μελέτης δείχνουν ότι ο προσεγγιστικός αλγόριθμος λειτουργεί αρκετά καλά και το σφάλμα είναι στα επιτρεπτά όρια.

5. Επίλογος και συμπεράσματα

Η χρήση γράφων για την μοντελοποίηση πραγματικών προβλημάτων είναι πολύ εκτεταμένη. Η θεωρία γράφων μπορεί να βρει πάρα πολλές εφαρμογές στην Πληροφορική, στη Μηχανική, στη Χημεία, στην Κοινωνιολογία κλπ. Τα τελευταία χρόνια μάλιστα, με την εξάπλωση των μέσων Κοινωνικής Δικτύωσης (Social Media Networks), οι γράφοι χρησιμοποιούνται συχνά για την αναπαράσταση των χρηστών και των σχέσεων που έχουν αυτοί μεταξύ τους. Η ύπαρξη μεγάλων κοινοτήτων στο Διαδίκτυο και τον Παγκόσμιο Ιστό κατά τα τελευταία χρόνια, ανοίγει νέα πεδία χρήσης, αφού η αναπαράσταση της επικοινωνίας των μελών των κοινοτήτων αυτών μπορεί να γίνει με τη βοήθεια των γράφων. Είναι πολύ σημαντική επομένως η ύπαρξη αποδοτικών αλγορίθμων οι οποίοι θα μπορούν να επεξεργάζονται γράφους και θα μπορούν να υπολογίσουν κάποιες από τις πιο σημαντικές μετρικές τους.

Η διάμετρος ενός γράφου αποτελεί μία από τις σημαντικότερες μετρικές, γιατί εκφράζει το μέγεθος του γράφου σε συνδυασμό με τη συνδεσιμότητά του και τη «συνοχή» του. Περιγράφει με άλλα λόγια, πόσο μακριά μπορεί να βρίσκονται μεταξύ τους 2 κορυφές στην χειρότερη περίπτωση. Η εύρεση της διαμέτρου ενός γράφου είναι πολλές φορές χρήσιμη και μπορεί να έχει διάφορες χρήσιμες εφαρμογές.

Είναι σαφές επομένως από τα πιο πάνω ότι είναι σημαντικό να γνωρίζουμε τη διάμετρο ενός γράφου, αφού μπορεί να δώσει χρήσιμες πληροφορίες για τη γεωμετρία ενός δικτύου. Ο υπολογισμός της διαμέτρου για γράφους που αντιστοιχούν σε τεράστιο όγκο δεδομένων δεν είναι καθόλου εύκολος και ο ακριβής υπολογισμός της απαιτεί πάρα πολύ χρόνο. Υπάρχουν αρκετοί αλγόριθμοι υπολογισμού της διαμέτρου ενός γράφου, όμως οι περισσότεροι από αυτούς εξετάζουν μόνο την περίπτωση μη κατευθυνόμενων γράφων και η απόδοσή τους δεν είναι καλή για την περίπτωση γράφων με πολύ μεγάλο πλήθος κορυφών.

Η παρούσα πτυχιακή εργασία παρουσιάζει την έρευνα που έχει γίνει για τον υπολογισμό της διαμέτρου ενός γράφου και παρουσιάζει έναν $7/4$ -προσεγγιστικό αλγόριθμο υπολογισμού της διαμέτρου ενός κατευθυνόμενου γράφου. Η απόδοση του αλγορίθμου είναι καλύτερη για την περίπτωση αραιών γράφων και είναι της

τάξης του $n^{1.5-\varepsilon}$ με παράγοντα προσέγγισης μικρότερο από 2. Μετά από την αναλυτική παρουσίαση των βημάτων του αλγορίθμου, ακολούθησε περιγραφή της υλοποίησης του με τη βοήθεια της γλώσσας προγραμματισμού Python. Για τη δοκιμή του αλγορίθμου δημιουργήθηκε ένα σύνολο τυχαίων γράφων με διαφορετικό πλήθος κορυφών και διαφορετική πυκνότητα και στη συνέχεια εφαρμόστηκε ο προσεγγιστικός αλγόριθμος στο σύνολο αυτό των γράφων. Ο αλγόριθμος κατάφερε σε όλες τις περιπτώσεις να υπολογίσει τη διάμετρο των γράφων, μέσα στα επιτρεπτά προσεγγιστικά όρια. Επομένως, μπορούμε να βγάλουμε το συμπέρασμα ότι η χρήση προσεγγιστικών αλγορίθμων για την επίλυση δύσκολων προβλημάτων, όπως είναι ο υπολογισμός της διαμέτρου για έναν πολύ μεγάλο γράφο, μπορεί να βοηθήσει στην ανάπτυξη χρήσιμων και αποδοτικών εφαρμογών.

6. Βιβλιογραφικές Αναφορές

- Abboud, A., Dalirrooyfard, M., Li, R. & Vassilevska-Williams, V., (2023). *On Diameter Approximation in Directed Graphs*. 31st Annual European Symposium on Algorithms (ESA 2023), arXiv preprint arXiv:2307.07583, 2023, arxiv.org.
- Abboud, A., Vassilevska Williams, V. & Wang, J. R. (2016) *Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs*. In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 377–391, 2016.
- Alman, J., Duan, R., Vassilevska Williams, V., Xu, Y., Xu, Z. & Zhou, R. (2024). *More Asymmetry Yields Faster Matrix Multiplication*. Cornell University, arXiv:2404.16349
- Alman, J. & Vassilevska Williams, V. (2021). *A refined laser method and faster matrix multiplication*. In Daniel Marx, editor, Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021, pages 522–539. SIAM, 2021.
- Alon, N., Galil, Z. & O. Margalit (1997). *On the exponent of the all pairs shortest path problem*. Journal of Comput. Syst. Sci., 54(2):255–262, 1997.
- Cairo, M., Grossi, R. & Rizzi, R. (2016) *New bounds for approximating extremal distances in undirected graphs*. In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 363–376, 2016.

- Chechik, S., Larkin, D. H., Roditty, L., Schoenebeck, G., Tarjan, R. E. & Vassilevska-Williams, V. (2014). *Better approximation algorithms for the graph diameter*. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 1041–1052, 2014.
- Cowen L. & Wagner, C.G. (1999) *Compact roundtrip routing for digraphs*. In Robert Endre Tarjan and Tandy J. Warnow, editors, Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA, pages 885–886. ACM/SIAM, 1999
- Dalirrooyfard M. & Kaufmann, J. (2021). *Approximation algorithms for mindistance problems in dags*. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference), volume 198 of LIPIcs, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021
- Dalirrooyfard, M., Vassilevska-Williams, V., Vyas, N., Wein, N., Xu, Y. & Yu, Y. (2019). *Approximation algorithms for min-distance problems*. In 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.
- Kaplan, H., Sharir, M. & Verbin, E. (2006) *Colored intersection searching via sparse rectangular matrix multiplication*. In Proceedings of the twenty-second annual symposium on Computational geometry, pages 52–60, 2006.
- Lincoln, A., Vassilevska-Williams, V. & Williams, R. R. (2018). *Tight hardness for shortest cycles and paths in sparse graphs*. In Artur Czumaj, editor, Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, pages 1236–1252. SIAM, 2018.
- Probst Gutenberg, M., Vassilevska-Williams, V. & Wein, N. (2020) *New algorithms and hardness for incremental single-source shortest paths in*

- directed graphs*. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, pages 153–166, 2020.
- PYPL (2025). *The Top IDE Index is created by analyzing how often IDEs' download page are searched on Google*. Διαθέσιμο στον σύνδεσμο: <https://pypl.github.io/IDE.html>
- Roditty, L. & Vassilevska-Williams, V. (2013). *Fast approximation algorithms for the diameter and radius of sparse graphs*. In Proceedings of the 45th annual ACM symposium on Symposium on theory of computing, STOC '13, pages 515–524, New York, NY, USA, 2013. ACM.
- Takes, F. W. & Kusters, W. A. (2011) *Determining the diameter of small world networks*. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, pages 1191–1196, 2011.
- Rubinstein, A. & Vassilevska-Williams (2019). *Seth vs approximation*. ACM SIGACT News, 50(4):57–76, 2019.
- TIOBE, (2025). *TIOBE Index for April 2025. April Headline: Former top 20 players Kotlin, Ruby and Swift have a hard time*. Διαθέσιμο στον σύνδεσμο: <https://www.tiobe.com/tiobe-index/>
- Williams, R. (2005). *A new algorithm for optimal 2-constraint satisfaction and its implications*. Theoretical Computer Science, 348(2-3):357–365, 2005.

Παράρτημα

Κώδικας για τον αλγόριθμο A_D :

```
def algA_d(A, D):  
    print("D=" + str(D))  
    # Αριθμός κόμβων αρχικού γράφου  
    n = A.shape[0]  
    (d, d1, d2) = algorithms.calcDegrees(A)  
    # Αριθμός ακμών του αρχικού γράφου  
    m = int(sum(d)/2)  
  
    # Βήμα 1  
    # Δημιουργία γράφου με μέγιστο βαθμό κορυφών 3  
    # print("Step 1")  
    # print("Δημιουργία γράφου με μέγιστο βαθμό κορυφής ίσο με 3")  
    B, no_nodes, no_edges = algorithms.max3degree_graph(A, d, n)  
    # print("B=")  
    # print(B)  
  
    # Υπολογισμός έσω-εκκεντρότητας για κάθε μία από τις κορυφές  
    inner_eccentricities = []  
    # print("**Εσω-Εκκεντρότητα")  
    for node in range(no_nodes):  
        e = algorithms.inner_eccentricity(node, B, no_nodes)  
        inner_eccentricities.append(e)
```

```

# print(inner_eccentricities)

# Υπολογισμός έξω-εκκεντρότητας για κάθε μία από τις κορυφές
outer_eccentricities = []
# print("Έξω-Εκκεντρότητα")
for node in range(no_nodes):
    e = algorithms.outer_eccentricity(node, B, no_nodes)
    outer_eccentricities.append(e)
# print(outer_eccentricities)

if math.inf in inner_eccentricities or math.inf in outer_eccentricities:
    return None

# Παράμετροι α και ω
# ω = fast matrix multiplication exponent
omega = 2.37286
alpha = (omega+1)/(omega+5)
# print("alpha="+str(alpha))
# Η λίστα με όλες τις κορυφές
list_of_nodes = [x for x in range(no_nodes)]
# print("List of nodes=" + str(list_of_nodes))
# print("Πλήθος κορυφών στον νέο γράφο:" + str(no_nodes))
# Βήμα 2
# Επιλέγουμε δείγμα με  $4m^{\alpha \log m}$  κορυφές
# print("Step 2")
sample_size = int(4*m**alpha*math.log10(m))
### sample_size = int(4 * m ** alpha * math.log(m))
# print("sample_size=" + str(sample_size))
# Επιλέγουμε με τυχαίο τρόπο κάποιες κορυφές
selected = algorithms.choose_random(list_of_nodes, sample_size)

```

```

# print(selected)

# Αν κάποια από τις επιλεγμένες κορυφές έχει έσω ή έξω εκκεντρότητα >=
4D/7
# ο αλγόριθμος αποδέχεται
limit = 4*D/7

# print("4D/7 = "+str(limit))
# print("Έλεγχος αν η εσωτερική ή εξωτερική εκκεντρότητα μίας κορυφής του
δείγματος είναι τουλάχιστον 4D/7")
for i in selected:
    # print("i="+str(i))
    # print("inner eccentricity=" + str(inner_eccentricities[i]))

    if inner_eccentricities[i] >= limit or outer_eccentricities[i] >= limit:
        # print("Βρέθηκε τέτοια κορυφή")
        # print("Accept: Step 2")
        return True
# print("Δεν βρέθηκε τέτοια κορυφή")
# Βήμα 3
# Ελέγχουμε αν υπάρχει κορυφή v όπου το πλήθος κορυφών στο Bout(v) για
απόσταση
# D/7 είναι <= m^a
# για κάθε μία κορυφή
# print("Step 3")
list_of_nodes = [x for x in range(no_nodes)]
# print("m**alpha="+str(m**alpha))
# print("Για κάθε κόμβο υπολογίζουμε το πλήθος των κόμβων στο Bout για
D/7")

```

```

# print("Αν για κάποιον κόμβο είναι  $\leq m^a$  και στο Bout+ αν υπάρχει κάποιος
κόμβος έχει εκκεντρότητα τουλάχιστον  $4D/7$ ")
# print("τότε ο αλγόριθμος αποδέχεται")
for v in list_of_nodes:
    Bout_set, count = algorithms.find_Bout1(B, no_nodes, v, D/7)
    # print("COUNT for "+str(v)+"="+str(count))
    if count  $\leq m^{**alpha}$ :
        # Αν υπάρχει τέτοια κορυφή, τότε ελέγχουμε αν υπάρχει κορυφή στο
        σύνολο
        # Bout_plus με εκκεντρότητα τουλάχιστον  $4D/7$ 
        # Βρίσκουμε το σύνολο Bout_plus
        Bout_plus, count = algorithms.find_Bout_plus1(B, no_nodes, v, D/7)
        for node in Bout_plus:
            if inner_eccentricities[node]  $\geq$  limit or outer_eccentricities[node]  $\geq$ 
limit:
                # print("Βρέθηκε τέτοια κορυφή")
                # print("Accept: Step 3")
                return True
        # print("Δεν βρέθηκε τέτοια κορυφή")

# Βήμα 4
# Ελέγχουμε αν υπάρχει κορυφή v όπου το πλήθος κορυφών στο Bin(v) για
απόσταση
# D/7 είναι  $\leq m^a$ 
# για κάθε μία κορυφή
# list_of_nodes = [x for x in range(no_nodes)]
# print("Step 4")
# print("Για κάθε κόμβο υπολογίζουμε το πλήθος των κόμβων στο Bin για D/7")
# print("Αν για κάποιον κόμβο είναι  $\leq m^a$  και στο Bout+ αν υπάρχει κάποιος
κόμβος έχει εκκεντρότητα τουλάχιστον  $4D/7$ ")

```

```

# print("τότε ο αλγόριθμος αποδέχεται")
for v in list_of_nodes:
    Bin_set, count = algorithms.find_Bin1(B, no_nodes, v, D/7)
    if count <= m**alpha:
        # Αν υπάρχει τέτοια κορυφή, τότε ελέγχουμε αν υπάρχει κορυφή στο
        # σύνολο
        # Bin_plus με εκκεντρότητα τουλάχιστον 4D/7
        # Βρίσκουμε το σύνολο Bin_plus
        Bin_plus, count = algorithms.find_Bin_plus1(B, no_nodes, v, D/7)
        for node in Bin_plus:
            if inner_eccentricities[node] >= limit or outer_eccentricities[node] >=
limit:
                # print("Βρέθηκε τέτοια κορυφή")
                # print("Accept: Step 4")
                return True
        # print("Δεν βρέθηκε τέτοια κορυφή")

# Βήμα 5
# Παίρνουμε δείγμα S_v με  $4m^{(1-\alpha)}\log m$  κορυφές
# print("Step 5")

sample_size = int(4*m**(1-alpha)*math.log10(m))
#### sample_size = int(4 * m ** (1 - alpha) * math.log(m))
# print("Sample size = " + str(sample_size))
# Η λίστα με όλες τις κορυφές
# list_of_nodes = [x for x in range(no_nodes)]
S_v = algorithms.choose_random(list_of_nodes, sample_size)
list_of_nodes = [x for x in range(no_nodes)]
# print(list_of_nodes)
# print("S_v=")

```

```

# print(S_v)
Sout = []
# print("m**(1-a)="+str(m**(1-alpha)))
# print("2*D/7="+str(2*D/7))
for s in S_v:
    # print(s)
    Bout, count = algorithms.find_Bout1(B, no_nodes, s, 2*D/7)
    # print("count="+str(count))
    if count <= m**(1-alpha):
        Sout.append(s)
# print("Sout="+str(Sout))

Sin = []
for s in S_v:
    # print(s)
    Bin, count = algorithms.find_Bin1(B, no_nodes, s, 2*D/7)
    # print(count)
    if count <= m**(1-alpha):
        Sin.append(s)
# print("Sin="+str(Sin))

# Βήμα 6
# Δημιουργία πίνακα Aout
Aout = np.zeros(shape=(len(Sout), no_nodes), dtype=int)
for s in Sout:
    Bout, count = algorithms.find_Bout1(B, no_nodes, s, 2*D/7)
    for v in range(no_nodes):
        if v in Bout:
            Aout[Sout.index(s),v] = 1

```

```

# Δημιουργία πίνακα Ain
Ain = np.zeros(shape=(no_nodes, len(Sin)), dtype=int)
if int(4*D/7) == 2*int(2*D/7):
    for s in Sin:
        Bin, count = algorithms.find_Bin1(B, no_nodes, s, 2 * D / 7)
        for v in range(no_nodes):
            if v in Bin:
                Ain[v, Sin.index(s)] = 1
else:
    for s in Sin:
        Bin_plus, count = algorithms.find_Bin_plus1(B, no_nodes, s, 2 * D / 7)
        for v in range(no_nodes):
            if v in Bin_plus:
                Ain[v, Sin.index(s)] = 1

product = np.matmul(Aout, Ain)
# print("product=")
# print(product)
if 0 in product:
    return True
else:
    return False

```