Υλοποίηση Αλγορίθμων για AT-free Γραφήματα

Δημήτριος Σίντος

Διπλωματική Εργασία

Επιβλέπων: Λεωνίδας Παληός

Ιωάννινα, Φεβρουάριος, 2024

ΤΜΉΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΉΣ
ΠΑΝΕΠΙΣΤΉΜΙΟ ΙΩΑΝΝΊΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου κ. Λεωνίδα Παληό, για την άριστη συνεργασία μας και τις εύστοχες παρατηρήσεις του σε κάθε βήμα και δυσκολία της εργασίας αυτής. Επίσης, ιδιαίτερα ευχαριστώ τους γονείς και συγγενείς μου για την συνεχή τους υποστήριξη και καθοδήγηση, καθώς και τους συμφοιτητές και φίλους μου για το ενδιαφέρον και την συμπαράστασή τους.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Δημήτριος Σίντος, Δίπλωμα, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Φεβρουάριος 2024.
Υλοποίηση Αλγορίθμων για AT-free Γραφήματα.
Επιβλέπων: Λεωνίδας Παληός, Καθηγητής.

Αστεροειδής τριάδα (AT σύντομα) είναι ένα σύνολο τριών κορυφών ενός γραφήματος τέτοιο ώστε να υπάρχει μονοπάτι μεταξύ οποιωνδήποτε δύο από αυτές αποφεύγοντας τη γειτονιά της τρίτης. Τα γραφήματα που δεν περιέχουν αστεροειδή τριάδα ονομάζονται AT-free. Η κατηγορία των AT-free γραφημάτων είναι ένας τύπος γραφήματος για τον οποίο πολλά προβλήματα που είναι NP-πλήρη σε γενικότερα γραφήματα μπορούν να λυθούν σε πολυωνυμικό χρόνο.

Σε αυτή τη διπλωματική εργασία έχουμε υλοποιήσει τρεις αλγορίθμους πολυωνυμικού χρόνου για τα AT-free γραφήματα.

Συγκεκριμένα, τον αλγόριθμο υπολογισμού μέγιστου ανεξάρτητου συνόλου των Hajo Broersma, Ton Kloks, Dieter Kratsch, και Haiko Müller[2], τον αλγόριθμο υπολογισμού Ελάχιστου Κυρίαρχου Συνόλου του Dieter Kratsch[19] και τον αλγόριθμο για το πρόβλημα του 3-Χρωματισμού του Juraj Stacho[17].

ABSTRACT

Dimitrios Sintos, Diploma, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, February 2024.
Implementation of Algorithms for AT-free Graphs.
Advisor: Leonidas Palios, Professor.

Asteroidal triple (AT for short) is a set of three vertices of a graph such that there is a path between any two of them avoiding the neighborhood of the third. Graphs that do not contain an asteroidal triple are called AT-free. The class of AT-free graphs is a type of graph for which many problems that are NP-complete in general graphs can be solved in polynomial time.

In this thesis, we have implemented three polynomial-time algorithms for AT-free graphs.

Specifically, the algorithm for computing the maximum independent set of Hajo Broersma, Ton Kloks, Dieter Kratsch, and Haiko Müller[2], the algorithm for computing the Minimum Dominating Set of Dieter Kratsch[2] and the algorithm for the 3-Coloring problem of Juraj Stacho[17].

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Η αστεροειδής τριάδα (asteroidal triple) εισήχθη το 1962 για να χαρακτηρίσουν τα interval γραφήματα ως εκείνα τα χορδωτά γραφήματα που δεν περιέχουν μια αστεροειδή τριάδα[20]. Αποτελούν μια μεγάλη κατηγορία γραφημάτων που περιλαμβάνει interval, permutation, trapezoid, και cocomparability γραφήματα. Στη παρούσα εργασία παρουσιάζουμε την υλοποίηση τριών αλγορίθμων για τα AT-free γραφήματα που τρέχουν σε πολυωνυμικό χρόνο. Τον αλγόριθμο των Broersma, H., Kloks, T., Kratsch, D. και Müller, H.[2] για τον υπολογισμό του μέγιστου ανεξάρτητου συνόλου, με χρονική πολυπλοκότητα $O(n^4)$, τον αλγόριθμο του Dieter Kratsch[19] για τον υπολογισμό του ελάχιστου κυρίαρχου συνόλου, με χρονική πολυπλοκότητα $O(n^6)$ και τον αλγόριθμο του Juraj Stacho[17] για την επίλυση του προβλήματος του 3-χρωματισμού, με χρονική πολυπλοκότητα $O(n^2m)$

1.1 Βασικοί Ορισμοί

Ακολουθούν βασικοί ορισμοί που απαιτούνται για την παρούσα εργασία.

Ορισμός 1.1. *Γράφημα (Graph)* είναι μια δομή που αποτελείται από ένα σύνολο κορυφών που συνδέονται μεταξύ τους με ένα σύνολο ακμών και το συμβολίζουμε με $G = (V, E)$, όπου V και E είναι τα σύνολα των κορυφών και των ακμών αντίστοιχα.

Ορισμός 1.2. *Μονοπάτι (Path)* μεταξύ δύο κορυφών σε ένα γράφημα ονομάζεται μια ακολουθία διαφορετικών κορυφών, όπου κάθε κορυφή της ακολουθίας συνδέεται με την επόμενή της μέσω ακμής.

Ορισμός 1.3. Έστω γράφημα G. Λέμε ότι ένα σύνολο κορυφών $S \subseteq V(G)$ είναι *ανεξάρτητο σύνολο* του G αν κανένα ζεύγος κορυφών από το S δεν είναι ακμή του G.

Ορισμός 1.4. Ο *αριθμός ανεξαρτησίας* $\alpha(G)$ ενός γραφήματος G, είναι το μέγιστο πλήθος ενός ανεξάρτητου συνόλου του G.

Ορισμός 1.5. Ένα *κυρίαρχο σύνολο* ενός γραφήματος G είναι ένα υποσύνολο κορυφών D τέτοιο ώστε κάθε κορυφή του G είτε ανήκει στο D είτε είναι γειτονική με μία κορυφή που ανήκει στο D.

Ορισμός 1.6. Ο *αριθμός κυριαρχίας* $\gamma(G)$ είναι ο αριθμός των κορυφών στο μικρότερο σύνολο κυριαρχίας του G.

Ορισμός 1.7. Ο *χρωματικός αριθμός* ενός γραφήματος G είναι το ελάχιστο πλήθος χρωμάτων που απαιτούνται για να χρωματιστούν οι κορυφές του G έτσι ώστε να μην υπάρχουν δύο γειτονικές κορυφές με το ίδιο χρώμα.

Ορισμός 1.8. Το *πρόβλημα του 3-χρωματισμού* αφορά εάν ένα γραφήμα G μπορεί να χρωματιστεί χρησιμοποιώντας μόνο τρία χρώματα, έτσι ώστε να μην υπάρχουν δύο γειτονικές κορυφές με το ίδιο χρώμα.

Ορισμός 1.9. Μια *αστεροειδής τριάδα* σε ένα γράφημα αποτελείται από τρεις μη γειτονικές κορυφές, έτσι ώστε να υπάρχει μονοπάτι μεταξύ κάθε ζευγαριού από αυτές που αποφεύγει την κλειστή γειτονιά της τρίτης.

Ορισμός 1.10. Ένα γράφημα ονομάζεται *AT-free* εάν δεν περιέχει καμία αστεροειδής τριάδα.

Ορισμός 1.11. Ένας γράφημα G είναι *ισχυρά συνεκτικό* αν δεν υπάρχει κανένας κόμβος του οποίου η αφαίρεση (μαζί με τις αντίστοιχες ακμές) να χωρίζει το γράφημα σε δύο ή περισσότερες συνεκτικές συνιστώσες.

1.2 Αντικείμενο της Διπλωματικής Εργασίας

Στην παρούσα διπλωματική εργασία, μελετούμε και υλοποιούμε τρεις αλγορίθμους για την επίλυση προβλημάτων σε AT-free γραφήματα.

Ειδικότερα, μελετάμε τον αλγόριθμο των Hajo Broersma , Ton Kloks , Dieter Kratsch , και Haiko Müller[2] για τον υπολογισμό του ανεξάρτητου αριθμού, τον αλγόριθμο του Dieter Kratsch[19] για τον εντοπισμό ελάχιστου κυρίαρχου συνόλου και τον αλγόριθμο του Juraj Stacho[17] για την αντιμετώπιση του προβλήματος του 3-χρωματισμού σε AT-free γραφήματα.

Ο σκοπός αυτής της εργασίας είναι η βαθύτερη κατανόηση της δομής και των αλγοριθμικών ιδιοτήτων των AT-free γραφημάτων, μέσω της εφαρμογής και επικύρωσης αυτών των αλγορίθμων. Όλοι αυτοί οι αλγόριθμοι επιτυγχάνουν το αποτέλεσμα τους σε πολυωνυμικό χρόνο.

1.3 Σχετικά Ερευνητικά Αποτελέσματα

Η εύρεση της αλγοριθμικής πολυπλοκότητας των ανεξάρτητων συνόλων σε AT-free γραφήματα αποτελεί μια σημαντική πρόκληση. Αν και τα ανεξάρτητα σύνολα είναι ένα κλασσικό NP-πλήρες πρόβλημα, τα AT-free γραφήματα παρουσιάζουν μοναδικές προκλήσεις. Σε αντίθεση με άλλες υποκατηγορίες, όπως τα cocomparability γραφήματα, τα γραφήματα AT-free δεν είναι τέλεια. Ως εκ τούτου, οι πολυωνυμικοί αλγόριθμοι που αναπτύχθηκαν για τέλεια γραφήματα, όπως αυτοί των Grötschel, Lovász και Schrijver[23], δεν εφαρμόζονται σε αυτή την περίπτωση.

Για τον υπολογισμό ενός ελάχιστου κυρίαρχου συνόλου έχουν σχεδιαστεί αλγόριθμοι πολυωνυμικού χρόνου για πολλές κατηγορίες γράφων (βλ.[27, 7]). Για παράδειγμα, υπάρχουν αποδοτικοί αλγόριθμοι για τον υπολογισμό ενός ελάχιστου κυρίαρχου συνόλου για τις ακόλουθες κατηγορίες γραφημάτων: interval graphs [22], strongly chordal graphs[12, 21] , cographs[4], permutation graphs[5, 9, 26, 26], k-polygon graphs [8], cocomparability graphs [15, 18], circular-arc graphs [24] and dually chordal graphs[1]. Ιδιαίτερα ενδιαφέρον είναι ο αλγόριθμος των Breu και Kirkpatrick σχετικά με μια υποκατηγορία των AT-free γραφημάτων. Έχουν δώσει αλγορίθμους $O(nm^2)$ για τον υπολογισμό ελάχιστου κυρίαρχου συνόλου και ενός ολικού ελάχιστου κυρίαρχου συνόλου στα cocomparability γραφήματα[15]

Το πρόβλημα του χρωματισμού είναι ένα από τα πρώτα προβλήματα που γνωρίζουμε ότι είναι NP-hard[11]. Αυτό ισχύει και για ιδικές κλάσεις γραφημάτων όπως τα planar graphs[10], line graphs[16], regular graphs[6] ή ακόμα και για σταθερό αριθμό k χρωμάτων (γνωστό και ώς το πρόβλημα του 3-χρωματισμού)[10]. Αντί-

θετα, για σενάρια με δύο ή λιγότερα χρώματα, το πρόβλημα επιλύεται σε πολυωνυμικό χρόνο. Αυτό ισχύει επίσης για ορισμένες κατηγορίες γραφημάτων με μοναδικές ιδιότητες, όπως τα interval graphs [13], chordal graphs [13], comparability graphs [13], και γενικότερα για τέλεια γραφήματα[14].

1.4 Δομή της Διπλωματικής Εργασίας

Η διπλωματική εργασία αναπτύσσεται σε τρία κεφάλαια. Στο κεφάλαιο 2 θα μελετήσουμε του τρεις αλγορίθμους. Στο επόμενο κεφάλαιο 3, παρουσιάζεται η λεπτομερής αναπαράσταση των δεδομένων, οι χρήσιμες δομές δεδομένων και η υλοποίηση κάθε αλγορίθμου. Στο τελευταίο κεφάλαιο 4 γίνεται σύνοψη των ευρημάτων, αναφέρονται δυνατές βελτιώσεις και προτείνεται πώς η υπάρχουσα υλοποίηση θα μπορούσε να τροποποιηθεί για την εκπόνηση πιστοποιητικών που επιβεβαιώνουν την ακρίβεια των αποτελεσμάτων.

ΚΕΦΑΛΑΙΟ 2

Οι Αλγόριθμοι

Σε αυτό το κεφάλαιο θα μελετήσουμε με λεπτομέρεια όλους τους αλγορίθμους που υλοποιήσαμε για τα AT-free γραφήματα. Για κάθε αλγόριθμο θα δώσουμε τον συμβολισμό και τα λήμματα που χρειάζονται για την περιγραφή του. Αμέσως μετά, θα εξηγήσουμε την πολυπλοκότητά του και συγκεκριμένα βήματά του, που θεωρήσαμε πιο ιδιαίτερα. Τέλος θα δώσουμε και ένα απλό παράδειγμα επίλυσης του κάθε αλγορίθμου.

Διευκρινίζουμε ότι δεν αποδεικνύουμε την ορθότητα του κάθε αλγορίθμου που χρησιμοποιούμε. Οι αποδείξεις αυτές βρίσκονται στις αντίστοιχες αναφορές([2], [19], [17])

2.1 Υπολογισμός Μέγιστου Ανεξάρτητου Συνόλου

Συμβολίζουμε τον αριθμό των κορυφών ενός γραφήματος $G = (V, E)$ με n και τον αριθμό των ακμών με m. Υπενθυμίζουμε ότι ένα ανεξάρτητο σύνολο σε ένα γράφημα G είναι ένα σύνολο από ζεύγη με μη γειτονικές κορυφές. Ο αριθμός ανεξαρτησίας ενός γραφήματος G, συμβολίζεται ως $\alpha(G)$ είναι το μέγεθος του μεγαλύτερου ανεξάρτητου συνόλου στο γράφημα. Οι βασικές δομικές ιδιότητες που πρέπει να αναλύσουμε πριν την περιγραφή του αλγορίθμου είναι τα $Components$ και τα $Intervals$.

Σε ένα AT-free γράφημα G όπου το x και το y είναι δύο ξεχωριστές μη γειτονικές κορυφές του G. Συμβολίζουμε με $C^x(y)$ το $Component$ του $G - N[x]$ όπου εμπεριέχεται το y, και με $r(x)$ τον αριθμό των $Components$ του $G - N[x]$.

Ορισμός 2.1. Μια κορυφή $z \in V \setminus \{x, y\}$ βρίσκεται μεταξύ των x και y εάν οι x

και z βρίσκονται σε ένα $Component$ του $G - N[y]$ και τα y και z βρίσκονται σε ένα $Component$ του $G - N[x]$.

Ορισμός 2.2. Το διάστημα $I = I(x, y)$ του G είναι το σύνολο όλων των κορυφών του G που βρίσκονται μεταξύ x και y. Συνεπώς, $I(x, y) = C_x(y) \cap C_y(x)$.

Ο αλγόριθμος των Broersma, H., Kloks, T., Kratsch, D. και Müller, H. καθορίζει τον αριθμό ανεξαρτησίας κάθε $Component$ και κάθε $Interval$ χρησιμοποιώντας τις σχέσεις που δίνονται στα Λήμματα 2.1, 2.2 και 2.3.

Λήμμα 2.1. *Έστω ότι $G = (V, E)$ είναι οποιαδήποτε γράφημα. Τότε*

$$\alpha(G) = 1 + \max_{x \in V} \left(\sum_{i=1}^{r(x)} \alpha(C_i^x) \right)$$

όπου $C_1^x, C_2^x, ..., C_{r(x)}^x$ τα $Compontes$ του $G - N[x]$.

Λήμμα 2.2. *Έστω ότι $G = (V, E)$ είναι ένα AT-free γράφημα. Έστω $x \in V$ και έστω C^x ένα $Component$ του $G - N[x]$. Τότε*

$$\alpha(C^x) = 1 + \max_{y \in C^x} \left(\alpha(I(x, y)) + \sum_i \alpha(D_i^y) \right)$$

όπου τα D_i^y είναι $Components$ του $G - N[y]$ που εμπεριέχονται στο C^x.

Λήμμα 2.3. *Έστω ότι $G = (V, E)$ είναι ένα AT-free γράφημα. Έστω $I = I(x, y)$ είναι ένα $Interval$ του G. Αν $I = \emptyset$ τότε $\alpha(I) = 0$. Αλλιώς*

$$\alpha(I) = 1 + \max_{s \in I} \left(\alpha(I(x, s)) + \alpha(I(s, y)) + \sum_i \alpha(C_i^s) \right)$$

όπου τα C_i^s είναι $Components$ του $G - N[s]$ που εμπεριέχονται στο $I(x, y)$.

Σε αυτό το σημείο μπορούμε να δώσουμε τον αλγόριθμο για τον υπολογισμό του αριθμού ανεξαρτησίας $\alpha(G)$ για AT-free γραφήματα, ο οποίος είναι βασισμένος στον δυναμικό προγραμματισμό.

Αλγόριθμος 2.1 Αλγόριθμος υπολογισμού αριθμού ανεξαρτησίας σε AT-free γραφήματα

Είσοδος: Ένα AT-free γράφημα G.

Έξοδος: Αριθμός ανεξαρτησίας $\alpha(G)$

1: Για κάθε $x \in V$, υπολόγισε όλα τα $Components$ $C_1^x, C_2^x, \ldots, C_{r(x)}^x$
2: Για κάθε ζευγάρι μη γειτονικών κορυφών x και y, υπολόγισε το $Interval$ $I(x, y)$.
3: Ταξινόμησε όλα τα $Components$ και τα $Intervals$ με βάση το μη-αύξοντα αριθμό κορυφών.
4: Υπολόγισε τα $\alpha(C)$ και $\alpha(I)$ για κάθε $Components$ C και κάθε $Interval$ I με τη σειρά του Βήματος 3.
5: Υπολόγισε το $\alpha(G)$.

Ορισμός 2.3. Υπάρχει αλγόριθμος χρόνου $O(n^4)$ για τον υπολογισμό του ανεξάρτητου αριθμού ενός AT-free γραφήματος.

Για την πολυπλοκότητα του αλγορίθμου μελετάμε κάθε βήμα ξεχωριστά. Το πρώτο βήμα μπορεί να υλοποιηθεί σε χρόνο $O(n(n + m))$ χρησιμοποιώντας έναν γραμμικό αλγόριθμο για τον υπολογισμό των $Components$.

Το βήμα 2 υπολογίζει $Intervals$ για τις μη γειτονικές κορυφές x και y, χρησιμοποιώντας την τομή των συνιστωσών $C_x(y)$ και $C_y(x)$. Η διαδικασία, που εκτελείται σε χρόνο O(n) για κάθε $Interval$, οδηγεί σε συνολική χρονική πολυπλοκότητα $O(n^3)$. Η υλοποίηση αξιοποιεί ένα λεξικό με αντικείμενα της κλάσης $Interval$, παρέχοντας έναν αποτελεσματικό τρόπο διαχείρισης και υπολογισμού εντός το πολύ n^2 $Component$ και $Intervals$.

Με την χρήση του Bucket sort το βήμα 3 υλοποιείται σε $O(n^2)$.

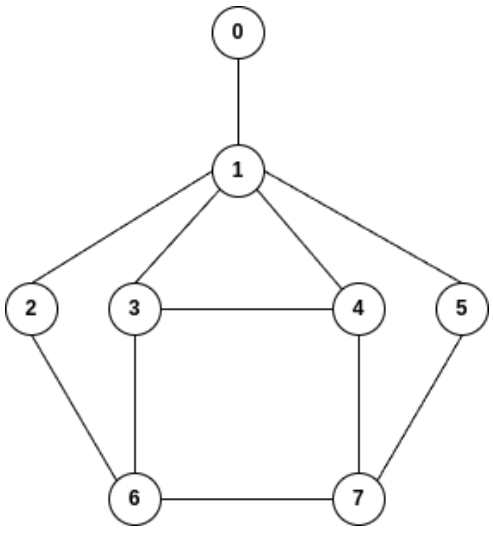
Το σημείο φραγμού για τη χρονική πολυπλοκότητα του αλγορίθμου μας είναι το βήμα 4 αφού εκτελείται σε $O(n^4)$. Για κάθε $Component$ C_x του $G - N[x]$ και μια κορυφή $y \in C_x$, ο αλγόριθμος πρέπει να υπολογίσει τα $Components$ του $G - N[y]$ που περιέχονται στην C_x. Αυτό γίνεται σε χρόνο $O(|C_x|)$ για σταθερές κορυφές x και y στο C_x, με αποτέλεσμα ο συνολικός χρόνος να είναι $O(n^3)$ για όλα τα $Compoents$. Θεωρώντας ένα $Interval$ $I(x, y)$ και μια κορυφή $s \in I$, ο αλγόριθμος πρέπει να αθροίσει τους αριθμούς ανεξαρτησίας των $Components$ C_s του $G - N[s]$ που περιέχονται στο I. Η λειτουργία αυτή απαιτεί χρόνο $O(|I(x, y)|)$ για ένα σταθερό $I(x, y)$ και μια κορυφή $s \in I$. Ο συνολικός χρόνος για τον υπολογισμό του $\alpha(I)$ για

όλα τα $Intervals$ είναι $O(n^3)$.

Είναι σαφές ότι το βήμα 5 μπορεί να γίνει σε χρόνο $O(n^2)$. Έτσι ο χρόνος εκτέλεσης του αλγορίθμου μας είναι $O(n^4)$.

Ο αλγόριθμος έτσι όπως παρουσιάζεται από τους Broersma, H., Kloks, T., Kratsch, D. και Müller, H υπολογίζει τον αριθμό ανεξαρτησίας ενός AT-free γραφήματος. Έχουμε επεκτείνει την υλοποίηση του αλγορίθμου έτσι ώστε στο βήμα 5 ο αλγόριθμος να επιστρέφει και ένα μέγιστο ανεξάρτητο σύνολο. Ο τρόπος με τον οποίο το πετύχαμε αυτό είναι προσθέτοντας μια επιπλέον δομή αποθήκευσης, στην περίπτωση μας ένα λεξικό. Κάθε φόρα που υπολογίζαμε το μέγιστο alpha είτε για τα $Intervals$ 2.3, είτε για τα $Components$ 2.2 αποθηκεύαμε στο λεξικό όλες τις ακμές από τα $Intervals$ και $Components$ που χρησιμοποιήθηκαν για να υπολογιστεί το μέγιστο alpha. Με αυτόν τον τρόπο στο τελευταίο βήμα είμαστε σε θέσει να επιστρέψουμε όλους τους κόμβους που χρειάστηκαν για τον υπολογισμό των alpha κάθε $Component$ του αθροίσματος 2.1 καθώς επίσης και το x που αντιστοιχεί στο 1 του τύπου.

Παρακάτω παραθέτουμε ένα παράδειγμα εκτέλεσης του αλγορίθμου για το AT-free γράφημα 2.1.



Σχήμα 2.1: Ένα AT-free γράφημα με 8 κόμβους

Μετά την εκτέλεση του πρώτου βήματος τα $Components$ που υπολογίσαμε φαίνονται στον πίνακα 2.1

Πίνακας 2.1: Components του γραφήματος μετά την εκτέλεση του πρώτου βήματος

Component	**Σύνολο κορυφών**	**Alpha**
C_1^1	{6, 7}	-
C_1^4	{2, 6}	-
C_2^4	{0}	-
C_3^4	{5}	-
C_1^3	{2}	-
C_2^3	{7, 5}	-
C_3^3	{0}	-
C_1^2	{3, 4, 5, 7}	-
C_2^2	{0}	-
C_1^0	{4, 3, 2, 6, 7, 5}	-
C_1^6	{1, 4, 0, 5}	-
C_1^7	{1, 3, 2, 0}	-
C_1^5	{3, 4, 6, 2}	-
C_2^5	{0}	-

Για το δεύτερο βήμα του αλγορίθμου, παίρνοντας όλους τους συνδυασμούς μη γειτονικών κορυφών για το παρών γράφημα. Τα $Intervals$ που προκύπτουν φαίνονται στον παρακάτω πίνακα2.2.

Πίνακας 2.2: Intervals του γραφήματος μετά την εκτέλεση του δεύτερου βήματος

Intervals	**Σύνολο κορυφών**	**Alpha**
I(2, 5)	{3, 4}	-
I(2, 7)	{3}	-
I(5, 2)	{3, 4}	-
I(5, 6)	{4}	-
I(0, 7)	{2, 3}	-
I(0, 6)	{4, 5}	-
I(7, 2)	{3}	-
I(7, 0)	{2, 3}	-
I(6, 5)	{4}	-
I(6, 0)	{4, 5}	-

Όπως φαίνεται και στους πίνακες 2.3 και 2.4,μετά το τρίτο βήμα τα *Components* και *Intervals* ταξινομούνται με βάση το πλήθος των κορυφών τους, από το μικρότερο στο μεγαλύτερο.

Πίνακας 2.3: Ταξινόμηση Components με βάση το σύνολο κορυφών

Components	**Σύνολο Κορυφών**	**Alpha**
C_2^4	{0}	-
C_3^4	{5}	-
C_1^3	{2}	-
C_3^3	{0}	-
C_2^2	{0}	-
C_2^5	{0}	-
C_1^1	{6, 7}	-
C_1^4	{2, 6}	-
C_2^3	{7, 5}	-
C_1^2	{3, 4, 5, 7}	-
C_1^6	{1, 4, 0, 5}	-
C_1^7	{1, 3, 2, 0}	-
C_1^5	{3, 4, 6, 2}	-
C_1^0	{4, 3, 2, 6, 7, 5}	-

Πίνακας 2.4: Ταξινόμηση Intervals με βάση το σύνολο κορυφών

Intervals	**Σύνολο Κορυφών**	**Alpha**
I(2, 7)	{3}	-
I(5, 6)	{4}	-
I(7, 2)	{3}	-
I(6, 5)	{4}	-
I(2, 5)	{3, 4}	-
I(5, 2)	{3, 4}	-
I(0, 7)	{2, 3}	-
I(0, 6)	{4, 5}	-
I(7, 0)	{2, 3}	-
I(6, 0)	{4, 5}	-

Χρησιμοποιώντας τα λήμματα 2.1, 2.2 και εφαρμόζοντας τεχνικές δυναμικού προγραμματισμού, μετά την εκτέλεση του βήματος τέσσερα έχουν υπολογιστεί τα alpha όλων των *Components* και *Intervals*. Τα αποτελέσματα φαίνονται στους πίνακες 2.5 και 2.6

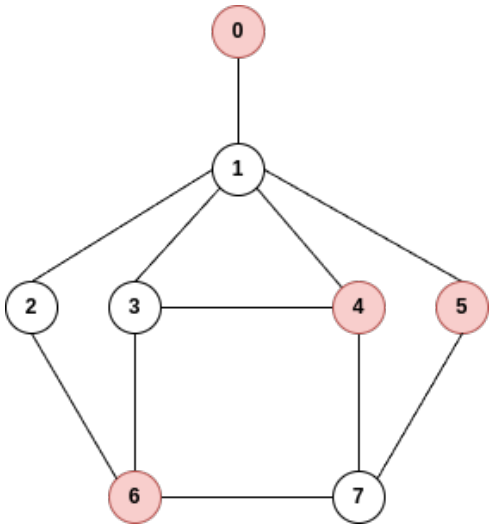
Πίνακας 2.5: Πίνακας Components μετά την εκτέλεση του βήματος 4 του αλγορίθμου

Components	**Σύνολο Κορυφών**	**Alpha**
C_2^4	{0}	1
C_3^4	{5}	1
C_1^3	{2}	1
C_3^3	{0}	1
C_2^2	{0}	1
C_2^5	{0}	1
C_1^1	{6, 7}	1
C_1^4	{2, 6}	1
C_2^3	{7, 5}	1
C_1^2	{3, 4, 5, 7}	2
C_1^6	{1, 4, 0, 5}	3
C_1^7	{1, 3, 2, 0}	3
C_1^5	{3, 4, 6, 2}	2
C_1^0	{4, 3, 2, 6, 7, 5}	3

Πίνακας 2.6: Πίνακας Intervals μετά την εκτέλεση του βήματος 4 του αλγορίθμου

Intervals	**Σύνολο Κορυφών**	**Alpha**
I(2, 7)	{3}	1
I(5, 6)	{4}	1
I(7, 2)	{3}	1
I(6, 5)	{4}	1
I(2, 5)	{3, 4}	1
I(5, 2)	{3, 4}	1
I(0, 7)	{2, 3}	2
I(0, 6)	{4, 5}	2
I(7, 0)	{2, 3}	2
I(6, 0)	{4, 5}	2

Στο τελευταίο βήμα με την εφαρμογή του λήμματος 2.1, είναι προφανές ότι ο αριθμός ανεξαρτησίας για το παρόν γράφημα είναι τέσσερα. Αυτό μπορεί, στο συγκεκριμένο παράδειγμα, να προκύψει από διάφορες κορυφές. Στο παρακάτω σχήμα φαίνεται ένα μέγιστο ανεξάρτητο σύνολο που έχει προκύψει από τον κόμβο 4.



Σχήμα 2.2: Μέγιστο Ανεξάρτητο Σύνολο σε AT-free γράφημα

2.2 Υπολογισμός Ελάχιστου Κυρίαρχου Συνόλου

Θα αναλύσουμε πεπερασμένα, μη-κατευθυνόμενα γραφήματα. Έστω $G = (V, E)$ ένα γράφημα. Ορίζουμε n ως τον αριθμό των κορυφών του G και $G[W]$ ως το υπογράφημα που επάγεται από το $W \subseteq V$. Η γειτονιά $N(v) = \{w \in V : \{v, w\} \in E\}$ αποτελείται από τις κορυφές που συνδέονται με το $v \in V$. Το $N[v] = N(v) \cup \{v\}$ είναι η κλειστή γειτονιά του $u \in V$, ενώ το $N[W] = \bigcup_{w \in W} N[w]$ αποτελεί την γειτονιά που εκτείνεται από το W.

Μια ακολουθία κορυφών ενός γραφήματος $G = (V, E)$ αποτελεί ένα μονοπάτι $P = (u = x_0, x_1, \ldots, x_k = u)$ στο G, όπου $\{x_i, x_{i+1}\} \in E$ για κάθε $i \in \{0, 1, \ldots, k-1\}$. Το μήκος k είναι η απόσταση ενός μονοπατιού $P = (u = x_0, x_1, \ldots, x_k = u)$.

Η απόσταση μεταξύ δύο κορυφών u και v στο G, συμβολίζεται ως $d_G(u, v)$ και ορίζεται ως το ελάχιστο μήκος ενός μονοπατιού μεταξύ των u και v. Η διάμετρος ενός γραφήματος G ορίζεται ως $diam(G) = \max\{d_G(u, v) : u, v \in V\}$.

Ένα υποσύνολο $D \subseteq V$ είναι ένα κυρίαρχο σύνολο του $G = (V, E)$ αν για κάθε κορυφή $u \in V \setminus D$ υπάρχει μια κορυφή $v \in D$ τέτοια ώστε $\{u, v\} \in E$, δηλαδή $N[D] = V$. Συμβολίζουμε το ελάχιστο κυρίαρχο σύνολο ενός γραφήματος G με $\gamma(G)$.

Ορισμός 2.4. Ένα μονοπάτι $P = (x_0, x_1, \ldots, x_d = y)$ είναι ένα κυρίαρχο συντομότερο μονοπάτι (DSP εν συντομία) ενός γραφήματος $G = (V, E)$, αν $dG(x, y) = d$ και $x_0, x_1, \ldots, x_d$ είναι ένα κυριάρχο σύνολο του G.

Θεώρημα 2.1. *Έστω $G = (V, E)$ ένα γράφημα με DSP $P = (x = x_0, x_1, \ldots, x_d = y)$. Έστω $H_0 = x, H_1 = N(x), \ldots, H_i = w \in V : D_G(x, w) = i, \ldots, H_l = w \in V : d_G(x, w) = l$ είναι τα BFS-επίπεδα του x. Τότε υπάρχει ένα ελάχιστης πληθικότητας κυρίαρχο σύνολο D του G τέτοιο ώστε*

$$\bigwedge_{i \in \{0,1,\ldots,\ell\}} \bigwedge_{j \in \{0,1,\ldots,\ell-i\}} \left| D \cap \bigcup_{s=i}^{i+j} H_s \right| \leq j + 4$$

Θεώρημα 2.2. *Έστω $G = (V, E)$ ένα συνδεδεμένο γράφημα χωρίς AT. Υπάρχει μια κορυφή $x \in V$ που μπορεί να προσδιοριστεί σε γραμμικό χρόνο με την ακόλουθη ιδιότητα. Έστω $H_0, H_1, \ldots, H_l$ είναι τα BFS-επίπεδα του x. Τότε υπάρχει ένα ελάχιστης πληθικότητας κυρίαρχο σύνολο D του G τέτοιο ώστε*

$$\bigwedge_{i\in\{0,1,\ldots,\ell\}} \bigwedge_{j\in\{0,1,\ldots,\ell-i\}} \left| D \cap \bigcup_{s=i}^{i+j} H_s \right| \leq j+3$$

Η βασική ιδέα του αλγορίθμου μας είναι ο υπολογισμός ενός κυρίαρχου συνόλου μέσω δυναμικού προγραμματισμού με τη χρήση των επιπέδων ενός δέντρου BFS. Ας εξετάσουμε ορισμένες λεπτομέρειες. Για εμάς, μια υποεπίλυση είναι ένα σύνολο $\mathbf{S} \subseteq \bigcup_{j=0}^{i-1} H_j$, που επιλέγεται κατά τη διάρκεια του δυναμικού προγραμματισμού μέχρι ένα σταθερό επίπεδο $i-1 \in \{1,2,\ldots,l-1\}$. Για να συλλέξουμε τις σχετικές πληροφορίες οποιασδήποτε υποεπίλυσης S, επιλέγουμε να αποθηκεύσουμε το υποσύνολο των κορυφών των υπολύσεων που ανήκουν στα δύο τελευταία επίπεδα, δηλαδή $S \cap (H_{i-2} \cup H_{i-1})$. Το άνω όριο για τον μέγιστο αριθμό κορυφών ενός ελάχιστου κυρίαρχου συνόλου που μπορεί να έχει σε οποιαδήποτε τρία διαδοχικά επίπεδα BFS είναι ζωτικής σημασίας για τον χρόνο εκτέλεσης του αλγορίθμου. Έχουμε δείξει ότι αυτός ο αριθμός είναι το πολύ 6 για γραφήματα με DSP 2.1 και το πολύ 5 για συνδεδεμένα γραφήματα χωρίς AT 2.2.

Θα παρουσιάσουμε τον αλγόριθμο $mcds_w(G)$, όπου w είναι ένας σταθερός θετικός ακέραιος, που υπολογίζει ένα κυρίαρχο σύνολο για ένα συνδεδεμένο γράφημα G. Αυτός ο αλγόριθμος θα μπορούσε να εφαρμοστεί και σε γενικά γραφήματα ως ένας απλός ευρετικός αλγόριθμος. Ωστόσο, η συμπεριφορά αυτού του αλγορίθμου μπορεί να είναι πολύ κακή. Για παράδειγμα, εάν, για όλα τα κυρίαρχα σύνολα D γραφήματος εισόδου $G=(V,E)$ και για όλες τις κορυφές x του G, υπάρχουν τρία διαδοχικά επίπεδα BFS του x έτσι ώστε το S να έχει περισσότερες από w κορυφές σε αυτά τα τρία επίπεδα, τότε η έξοδος του $mcds_w(G)$ είναι απλώς το τετριμμένο κυρίαρχο σύνολο V. Εάν το γράφημα εισόδου G έχει μια κορυφή x και ένα κυρίαρχο σύνολο ελάχιστης πληθικότητας D έτσι ώστε το πολύ w κορυφές του D να ανήκουν σε οποιαδήποτε τρία διαδοχικά επίπεδα BFS, τότε η $mcds_w(G)$ εξάγει ένα ελάχιστο κυρίαρχο σύνολο του G.

Αλγόριθμος 2.2 Αλγόριθμος υπολογισμού ελάχιστου κυρίαρχου συνόλου σε AT-free γραφήματα $mcds_w$

Είσοδος: Ένα AT-free γράφημα $G = (V, E)$.

Έξοδος: Ελάχιστο κυρίαρχο σύνολο $D \subseteq V$

Αρχικοποίησε το $D := V$;
for όλα τα $x \in V$ **do**
  Υπολόγισε τα επίπεδα BFS του κόμβου x:
  $H_0 = \{x\}$; $H_1 = N(x)$; ...; $H_l = \{u \in V : d_G(x, u) = l\}$;
  $i := 1$;
  Αρχικοποίησε την ουρά A_1 ώστε να περιέχει μια διατεταγμένη τριάδα $(S, S, \text{val}(S))$ για όλα τα μη κενά υποσύνολα S του $N[x]$ που ικανοποιούν το $\text{val}(S) := |S| \leq w$;
  while $A_i \neq \emptyset$ **and** $i < l$ **do**
    $i := i + 1$;
    for όλες τις τριάδες $(S, S', \text{val}(S'))$ στην ουρά A_{i-1} **do**
      for κάθε $U \subseteq H_i$ με $|S \cup U| \leq w$ **do**
        if $N[S \cup U] \supseteq H_{i-1}$ **then**
          $R := (S \cup U) \setminus H_{i-2}$;
          $R' := S' \cup U$;
          $\text{val}(R') := \text{val}(S') + |U|$;
          if δεν υπάρχει καμία τριάδα στην A_i με πρώτη εγγραφή R **then**
            πρόσθεσε το $(R, R', \text{val}(R'))$ στην ουρά A_i;
          end if
          if υπάρχει μια τριάδα $(P, P', \text{val}(P'))$ in A_i όπου $P = R$ and $\text{val}(R') < \text{val}(P')$ **then**
            αντικατέστησε το $(P, P', \text{val}(P'))$ μέσα στην A_i με $(R, R', \text{val}(R'))$;
          end if
        end if
      end for
    end for
  end while
  Μεταξύ όλων των τριάδων $(S, S', \text{val}(S'))$ στην ουρά A_l που ικανοποιούν $H_l \subseteq N[S]$, προσδιορίστε μία με ελάχιστη τιμή $\text{val}(S')$, έστω $(B, B', \text{val}(B'))$;
  if $\text{val}(B') < |D|$ **then**
    $D := B'$;
  end if
end for

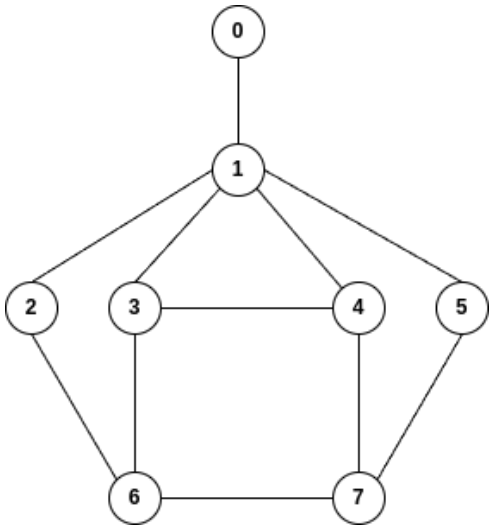
Θεώρημα 2.3. *Ο αλγόριθμος $mcds_w(G)$ υπολογίζει σε χρόνο $O(n^{w+2})$ ένα ελάχιστο κυρίαρχο σύνολο ενός συνδεδεμένου γραφήματος $G = (V, E)$ αν το G έχει ελάχιστο κυρίαρχο σύνολο D και μια κορυφή $x \in V$ έτσι ώστε το πολύ w κορυφές του D να ανήκουν σε οποιαδήποτε τρία διαδοχικά επίπεδα BFS του x.*

Θεώρημα 2.4. *Υπάρχει ένας αλγόριθμος χρόνου $O(n^7)$ για τον υπολογισμό ενός ελάχιστου κυρίαρχου συνόλου οποιουδήποτε γραφήματος για το οποίο κάθε συνδεδεμένη συνιστώσα έχει ένα κυρίαρχο συντομότερο μονοπάτι.*

Κατ' αναλογία, τα θεωρήματα 2.2 και 2.4 υποδηλώνουν ότι ο αλγόριθμος $mcds_5(G)$ υπολογίζει για ένα δοθέν συνδεδεμένο γράφημα AT-free ένα ελάχιστο κυρίαρχο σύνολο. Επιπλέον, αρκεί να υπολογίσουμε ένα κυρίαρχο ζεύγος (x, y) και στη συνέχεια να ελέγξουμε τα επίπεδα BFS της κορυφής x μόνο. Έτσι, λαμβάνουμε

Θεώρημα 2.5. *Υπάρχει ένας αλγόριθμος χρόνου $O(n^6)$ για τον υπολογισμό ελάχιστου κυρίαρχου συνόλου για οποιοδήποτε δεδομένο γράφημα χωρίς AT.*

Παρακάτω παραθέτουμε ένα παράδειγμα εκτέλεσης του αλγορίθμου για τον κόμβο 1 του AT-free γραφήματος 2.3.



Σχήμα 2.3: Ένα AT-free γράφημα με 8 κόμβους

Πίνακας 2.7: Επίπεδα BFS του κόμβου 1

Επίπεδα	**Κορυφές**
0	$\{1\}$
1	$\{0, 2, 3, 4, 5\}$
2	$\{6, 7\}$

Πίνακας 2.8: Αρχικοποίηση Ουράς A_1

S	**S'**	**val(S)**
$\{4\}$	$\{4\}$	1
$\{3\}$	$\{3\}$	1
$\{2\}$	$\{2\}$	1
$\{0\}$	$\{0\}$	1
$\{5\}$	$\{5\}$	1
$\{1\}$	$\{1\}$	1
$\{4,3\}$	$\{4,3\}$	2
$\{4,2\}$	$\{4,2\}$	2
$\{4,0\}$	$\{4,0\}$	2
$\{4,5\}$	$\{4,5\}$	2
$\{4,1\}$	$\{4,1\}$	2
$\{3,2\}$	$\{3,2\}$	2
$\{3,0\}$	$\{3,0\}$	2
$\{3,5\}$	$\{3,5\}$	2
$\{3,1\}$	$\{3,1\}$	2
$\{2,0\}$	$\{2,0\}$	2
$\{5,2\}$	$\{5,2\}$	2
$\{2,1\}$	$\{2,1\}$	2
$\{5,0\}$	$\{5,0\}$	2
$\{1,0\}$	$\{1,0\}$	2
$\{5,1\}$	$\{5,1\}$	2
$\{4,3,2\}$	$\{4,3,2\}$	3
$\{4,3,0\}$	$\{4,3,0\}$	3
$\{4,3,5\}$	$\{4,3,5\}$	3
$\{4,3,1\}$	$\{4,3,1\}$	3
$\{4,2,0\}$	$\{4,2,0\}$	3
$\{4,5,2\}$	$\{4,5,2\}$	3
$\{4,2,1\}$	$\{4,2,1\}$	3
$\{4,5,0\}$	$\{4,5,0\}$	3
$\{4,1,0\}$	$\{4,1,0\}$	3
$\{4,5,1\}$	$\{4,5,1\}$	3

S	S'	val(S)
$\{3,2,0\}$	$\{3,2,0\}$	3
$\{5,3,2\}$	$\{5,3,2\}$	3
$\{3,2,1\}$	$\{3,2,1\}$	3
$\{3,5,0\}$	$\{3,5,0\}$	3
$\{3,1,0\}$	$\{3,1,0\}$	3
$\{3,5,1\}$	$\{3,5,1\}$	3
$\{5,2,0\}$	$\{5,2,0\}$	3
$\{2,1,0\}$	$\{2,1,0\}$	3
$\{5,2,1\}$	$\{5,2,1\}$	3
$\{5,1,0\}$	$\{5,1,0\}$	3
$\{4,3,2,0\}$	$\{4,3,2,0\}$	4
$\{4,3,2,5\}$	$\{4,3,2,5\}$	4
$\{4,3,2,1\}$	$\{4,3,2,1\}$	4
$\{4,3,5,0\}$	$\{4,3,5,0\}$	4
$\{4,3,1,0\}$	$\{4,3,1,0\}$	4
$\{4,3,5,1\}$	$\{4,3,5,1\}$	4
$\{4,5,2,0\}$	$\{4,5,2,0\}$	4
$\{4,2,1,0\}$	$\{4,2,1,0\}$	4
$\{4,5,2,1\}$	$\{4,5,2,1\}$	4
$\{4,5,1,0\}$	$\{4,5,1,0\}$	4
$\{5,3,2,0\}$	$\{5,3,2,0\}$	4
$\{3,2,1,0\}$	$\{3,2,1,0\}$	4
$\{5,3,2,1\}$	$\{5,3,2,1\}$	4
$\{3,5,1,0\}$	$\{3,5,1,0\}$	4
$\{5,2,1,0\}$	$\{5,2,1,0\}$	4
$\{4,2,5,3,0\}$	$\{4,2,5,3,0\}$	5
$\{4,2,3,0,1\}$	$\{4,2,3,0,1\}$	5
$\{4,2,5,3,1\}$	$\{4,2,5,3,1\}$	5
$\{4,5,3,0,1\}$	$\{4,5,3,0,1\}$	5
$\{4,2,5,0,1\}$	$\{4,2,5,0,1\}$	5
$\{2,5,3,0,1\}$	$\{2,5,3,0,1\}$	5

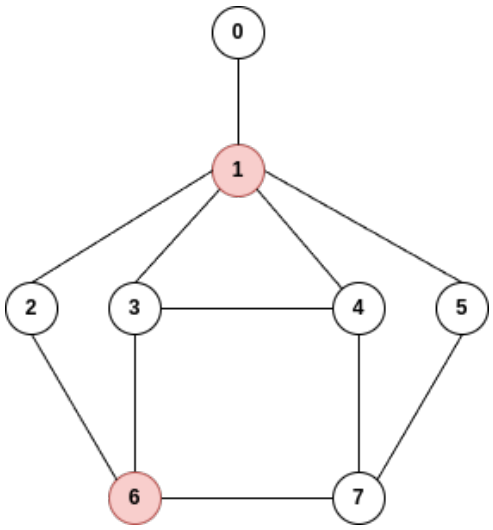
Πίνακας 2.9: Ουρά A_1 μετά την εκτέλεση του δυναμικού αλγορίθμου

S	**S'**	**val(S)**
$\{6,7,0\}$	$\{7,6,0\}$	3
$\{6\}$	$\{6,1\}$	2
$\{7\}$	$\{7,1\}$	2
$\{6,7\}$	$\{7,6,1\}$	3
$\{4,6,7,0\}$	$\{4,7,6,0\}$	4
$\{4,6\}$	$\{4,6,1\}$	3
$\{4,7\}$	$\{4,7,1\}$	3
$\{4,6,7\}$	$\{4,7,6,1\}$	4
$\{6,3,7,0\}$	$\{7,3,6,0\}$	4
$\{3,6\}$	$\{3,6,1\}$	3
$\{3,7\}$	$\{3,7,1\}$	3
$\{6,3,7\}$	$\{7,3,6,1\}$	4
$\{6,2,7,0\}$	$\{7,2,6,0\}$	4
$\{2,6\}$	$\{2,6,1\}$	3
$\{2,7\}$	$\{2,7,1\}$	3
$\{6,2,7\}$	$\{7,2,6,1\}$	4
$\{6,5,7,0\}$	$\{7,5,6,0\}$	4
$\{6,0\}$	$\{1,6,0\}$	3
$\{7,0\}$	$\{1,7,0\}$	3
$\{5,6\}$	$\{5,6,1\}$	3
$\{5,7\}$	$\{5,7,1\}$	3
$\{6,5,7\}$	$\{7,5,6,1\}$	4
$\{4,6,3,7,0\}$	$\{4,3,6,0,7\}$	5
$\{4,3,6\}$	$\{4,3,6,1\}$	4
$\{4,3,7\}$	$\{4,3,7,1\}$	4
$\{4,3,6,7\}$	$\{4,3,6,7,1\}$	5
$\{4,2,7,0\}$	$\{4,2,7,0\}$	4
$\{4,2,6,7,0\}$	$\{4,2,6,0,7\}$	5
$\{4,2,6\}$	$\{4,2,6,1\}$	4
$\{4,2,7\}$	$\{4,2,7,1\}$	4
$\{4,2,6,7\}$	$\{4,2,6,7,1\}$	5

S	S'	val(S)
$\{4,5,6,0\}$	$\{4,5,6,0\}$	4
$\{4,6,5,7,0\}$	$\{4,6,0,7,5\}$	5
$\{4,6,0\}$	$\{4,1,6,0\}$	4
$\{4,7,0\}$	$\{4,1,7,0\}$	4
$\{4,5,6\}$	$\{4,5,6,1\}$	4
$\{4,5,7\}$	$\{4,5,7,1\}$	4
$\{4,5,6,7\}$	$\{4,6,7,5,1\}$	5
$\{3,2,7,0\}$	$\{3,2,7,0\}$	4
$\{2,6,3,7,0\}$	$\{3,2,6,0,7\}$	5
$\{3,2,6\}$	$\{3,2,6,1\}$	4
$\{3,2,7\}$	$\{3,2,7,1\}$	4
$\{3,2,6,7\}$	$\{3,2,6,7,1\}$	5
$\{3,5,6,0\}$	$\{3,5,6,0\}$	4
$\{6,5,3,7,0\}$	$\{3,6,0,7,5\}$	5
$\{3,6,0\}$	$\{3,1,6,0\}$	4
$\{3,7,0\}$	$\{3,1,7,0\}$	4
$\{3,5,6\}$	$\{3,5,6,1\}$	4
$\{3,5,7\}$	$\{3,5,7,1\}$	4
$\{3,5,6,7\}$	$\{3,6,7,5,1\}$	5
$\{2,6,5,7,0\}$	$\{2,6,0,7,5\}$	5
$\{2,6,0\}$	$\{2,1,6,0\}$	4
$\{2,7,0\}$	$\{2,1,7,0\}$	4
$\{5,2,6\}$	$\{5,2,6,1\}$	4
$\{5,2,7\}$	$\{5,2,7,1\}$	4
$\{5,2,6,7\}$	$\{2,6,7,5,1\}$	5
$\{5,6,0\}$	$\{5,1,6,0\}$	4
$\{5,7,0\}$	$\{5,1,7,0\}$	4
$\{4,2,3,7,0\}$	$\{4,3,2,7,0\}$	5
$\{4,3,2,6\}$	$\{4,3,2,6,1\}$	5
$\{4,3,2,7\}$	$\{4,3,2,7,1\}$	5
$\{4,6,5,3,0\}$	$\{4,3,6,0,5\}$	5
$\{4,3,6,0\}$	$\{4,3,6,0,1\}$	5
$\{4,3,7,0\}$	$\{4,3,7,0,1\}$	5

S	S'	val(S)
$\{4,3,5,6\}$	$\{4,3,6,5,1\}$	5
$\{4,3,5,7\}$	$\{4,3,7,5,1\}$	5
$\{4,2,6,5,0\}$	$\{4,2,6,0,5\}$	5
$\{4,2,5,7,0\}$	$\{4,2,7,0,5\}$	5
$\{4,2,6,0\}$	$\{4,2,6,0,1\}$	5
$\{4,5,2,6\}$	$\{4,2,6,5,1\}$	5
$\{4,5,2,7\}$	$\{4,2,7,5,1\}$	5
$\{4,5,7,0\}$	$\{4,7,0,5,1\}$	5
$\{2,6,5,3,0\}$	$\{3,2,6,0,5\}$	5
$\{2,5,3,7,0\}$	$\{3,2,7,0,5\}$	5
$\{3,2,6,0\}$	$\{3,2,6,0,1\}$	5
$\{5,3,2,6\}$	$\{3,2,6,5,1\}$	5
$\{5,3,2,7\}$	$\{3,2,7,5,1\}$	5
$\{3,5,7,0\}$	$\{3,7,0,5,1\}$	5
$\{5,2,6,0\}$	$\{2,6,0,5,1\}$	5
$\{5,2,7,0\}$	$\{2,7,0,5,1\}$	5

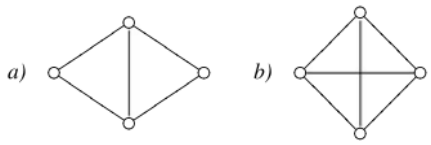
Από τη γραμμή 25 του αλγορίθμου 2.2 προκύπτει ότι η τετράδα με το ελάχιστο $val(S')$ είναι η $(\{6\},\{6,1\},2)$, αφού ισχύει και η συνθήκη $H_l \subseteq N[S]$ ($\{6,7\} \subseteq \{6,2,7,3\}$). Άρα, το ελάχιστο κυρίαρχο σύνολο είναι το $\{6,1\}$.



Σχήμα 2.4: Ελάχιστο Κυρίαρχο Σύνολο σε AT-free γράφημα

2.3 Το Πρόβλημα του 3-Χρωματισμού

Για το Πρόβλημα του 3-Χρωματισμού θεωρούμε ότι ένα γράφημα είναι πάντα απλό, μη κατευθυνόμενο και χωρίς βρόχους. Για μια κορυφή v ενός γραφήματος G, συμβολίζουμε με $N_G(v)$ το σύνολο των κορυφών που είναι γειτονικές με την v στο G και γράφουμε $N_G[v] = N_G(v) \cup \{v\}$. Αποσύρουμε τον δείκτη G και γράφουμε $N(v)$ και $N[v]$ όποτε αυτό είναι σαφές από τα συμφραζόμενα. Για $X \subseteq V(G)$, γράφουμε $G[X]$ για το υπογράφημα του G που επάγεται από το X, και γράφουμε $G - X$ για το υπογράφημα του G που επάγεται από το $V(G) \setminus X$. Ένα σύνολο $X \subseteq V(G)$ είναι σταθερό αν το $G[X]$ δεν περιέχει ακμές. Το K_n δηλώνει το πλήρες γράφημα (δηλαδή το γράφημα με όλες τις πιθανές ακμές) σε n κορυφές, και το διαμάντι είναι το (μοναδικό) γράφημα με 4 κορυφές με 5 ακμές 2.5.



Σχήμα 2.5: **(α)** Διαμάντι, **(β)** K_4

Γράφουμε G/S για το γράφημα που λαμβάνουμε από το G με τη σύμπτυξη όλων των κορυφών του S σε μία μόνο κορυφή. Δηλαδή,

$V(G/s) = (V(G) \setminus S) \cup \{s\}$ where $s \notin V(G)$,

$E(G/s) = \{xy \in E(G) \mid x, y \notin S\} \cup \{sy \mid xy \in E(G) \wedge x \in S \wedge y \notin S\}$

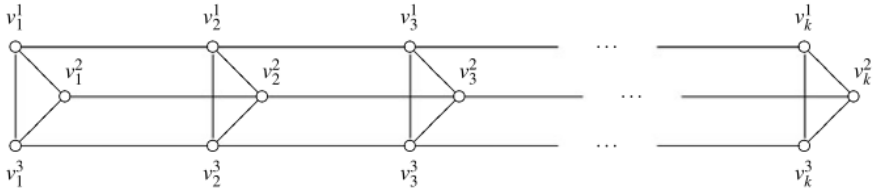
Ένα σύνολο $S \subseteq V(G)$ αποσυνδέει τις κορυφές a και b στο G αν a και b ανήκουν σε διαφορετικές συνδεδεμένες συνιστώσες του $G - S$. Ορίζουμε το S ως σύνολο αποκοπής του G αν αποσυνδέει ορισμένες κορυφές a και b. Επιπλέον, το S ονομάζεται ελάχιστο διαχωριστικό (*minimal separator*) του G αν υπάρχουν κορυφές a και b τέτοιες ώστε το S να αποσυνδέει τις a και b, αλλά κανένα υποσύνολο του S να μην μπορεί να τις αποσυνδέσει. Ένα σημείο αποκοπής (*cutpoint*) του G είναι μια κορυφή v τέτοια ώστε το σύνολο $\{v\}$ να λειτουργεί ως σύνολο αποκοπής. Ένα μπλοκ του G είναι ένα μέγιστο συνδεδεμένο επαγόμενο υπογράφημα του G που δεν έχει σημεία αποκοπής.

Παρουσιάζουμε τα απαραίτητα Θεωρήματα και Λήμματα για την υλοποίηση του αλγορίθμου.

Θεώρημα 2.6. *Υπάρχει ένας αλγόριθμος χρόνου $O(n^2m)$ για να αποφασίσουμε, δεδομένου ενός AT-free γραφήματος G με n κορυφές και m ακμές, αν το G είναι ή όχι 3-χρωματίσιμο και να κατασκευάσει επίσης ένα 3-χρωματισμό G, αν υπάρχει.*

Θεώρημα 2.7. *Έστω G ένα γράφημα AT-free με τουλάχιστον τρεις κόμβους και χωρίς κανένα επαγόμενο διαμάντι ή K_4.*

1. *G είναι τριγωνική λωρίδα (triangular strip) 2.6, ή*
2. *G περιέχει ένα σταθερό cutset.*



Σχήμα 2.6: Τριγωνική Λωρίδα (*triangular strip*) τάξης k

Θεώρημα 2.8. *Κάθε γράφημα G που είναι AT-free και δεν περιέχει επαγόμενο διαμάντι και K_4 είναι 3-χρωματίσιμο. Επιπλέον, αν το G περιέχει ένα ελάχιστο σταθερό διαχωριστικό S, τότε υπάρχει ένας 3-χρωματισμός του G στον οποίο όλοι οι κόμβοι του S έχουν τον ίδιο χρωματισμό.*

Θεώρημα 2.9. *Αν οι κόμβοι u, v είναι γειτονικοί σε ένα γράφημα G που είναι AT-free και S είναι ένα μέγιστο σταθερό σύνολο στο $N(u) \cap N(v)$, τότε το G/S είναι AT-free. Επιπλέον, το G είναι 3-χρωματίσιμο αν και μόνο αν το G/S είναι.*

Λήμμα 2.4. *Αν το G είναι AT-free και το S είναι ένα ελάχιστο διαχωριστικό σύνολο, τότε το G/S είναι επίσης AT-free.*

Λήμμα 2.5. *Έστω G ένα ισχυρά συνεκτικό γράφημα γράφημα AT-free χωρίς επαγόμενο διαμάντι και K_4. Τότε κάθε κόμβος του G βρίσκεται το πολύ σε ένα τρίγωνο.*

Λήμμα 2.6. *Αν το S είναι ένα ελάχιστο σταθερό διαχωριστικό ενός γραφήματος G που είναι AT-free, τότε υπάρχει ένας κόμβος $x \in V(G)$ με $N(x) \supseteq S$.*

Λήμμα 2.7. *Αν το S είναι ένα σταθερό διαχωριστικό ενός συνεκτικού γραφήματος G, και $S \subseteq S$ είναι ένα σταθερό σύνολο, τότε το S είναι επίσης ένα σταθερό διαχωριστικό του G.*

Λήμμα 2.8. *Ένα σύνολο $S \subseteq V(G)$ είναι ένα ελάχιστο διαχωριστικό ενός γραφήματος G με $|S| \geq 2$ αν και μόνο αν υπάρχει ένα μπλοκ B του G τέτοιο ώστε το S να είναι ένα ελάχιστο διαχωριστικό του B.*

Για την απόδειξη του Θεωρήματος 2.6 πρέπει:

1. Μειώσουμε το πρόβλημα σε γραφήματα AT-free χωρίς επαγόμενα διαμάντια,
2. Αα αποσυνθέσουμε κάθε AT-free γράφημα χωρίς επαγόμενα διαμάντια και χωρίς K_4 σε τριγωνικές λωρίδες χρησιμοποιώντας σταθερά διαχωριστικά, και
3. Να συμπτύξουμε ελάχιστα σταθερά διαχωριστικά χωρίς να αλλάξουμε την απάντηση στο πρόβλημα.

Αυτό μειώνει το πρόβλημα σε γραφήματα τα οποία κάθε μπλοκ είναι μια τριγωνική λωρίδα ή έχει το πολύ δύο κόμβους. Σε αυτά τα γραφήματα είναι εύκολο να κατασκευάσουμε τον 3-χρωματισμό τους. Αν σε οποιοδήποτε στάδιο συναντήσουμε ένα K_4, δηλώνουμε ότι ο γράφημα δεν είναι 3-χρωματίσιμο.

Παρουσιάζουμε ένα περίγραμμα αλγορίθμου που προκύπτει από αυτό που έχουμε αναφέρει:

Αλγόριθμος 2.3 Αλγόριθμος επίλυσης του προβλήματος 3-χρωματισμού για AT-free γραφήματα

Είσοδος: Ένα AT-free γράφημα G.

Έξοδος: Ένα 3-χρωματισμό του G ή "Το G δεν είναι 3-χρωματίσιμο".

1: **if** το G περιέχει ένα K_4 **then**
2: **return** "Το G δεν είναι 3-χρωματίσιμο"
3: **end if**
4: /* Τώρα το G δεν περιέχει K_4 */
5: **if** το G περιέχει γειτονικές κορυφές u, v με $|N(u) \cap N(v)| \geq 2$ **then**
6: **return** Αναδρομικά βρες τον 3-χρωματισμό του $G/N(u) \cap N(v)$
7: **end if**
8: /* Τώρα το G δεν περιέχει επαγόμενα διαμάντια και κανένα K_4 */
9: **if** το G περιέχει έναν ελάχιστο σταθερό διαχωριστή S με $|S| \geq 2$ **then**
10: **return** Αναδρομικά βρες τον 3-χρωματισμό του G/S
11: **end if**
12: /* Τώρα κάθε τμήμα του G είναι είτε τριγωνική λωρίδα είτε έχει το πολύ 2 κορυφές */
13: **return** Κατασκεύασε τον 3-χρωματισμό του G

Η χρονική πολυπλοκότητα του παρόν αλγορίθμου είναι $O(n^2m)$. Για να το αποδείξουμε αυτό αρκεί να δείξουμε πώς μπορούμε να υλοποιήσουμε τις γραμμές 1,5,9 και 13 του αλγορίθμου.

Για την γραμμή 1 του αλγορίθμου η διαδικασία έχει ως εξής: Εάν η γραμμή 1 εκτελείται για πρώτη φορά, ελέγχουμε εάν το γράφημα G περιέχει ένα πλήρες γράφημα με τέσσερις κορυφές (K_4) σε χρόνο πολυπλοκότητας $O(m^2) = O(n^2 \cdot m)$ εξετάζοντας όλα τα πιθανά ζεύγη μη συνδεδεμένων ακμών στο G. Εάν η γραμμή 1 επιτυγχάνεται μέσω αναδρομής στο G/S, όπου s είναι η κορυφή του G/S που προκύπτει από τη σύμπτυξη του S, τότε χρειάζεται μόνο να ελέγξουμε εάν η γειτονιά του s στο G/S περιέχει ένα τρίγωνο. Αυτός ο έλεγχος μπορεί να γίνει σε χρόνο $O(n \cdot m)$ εξετάζοντας κάθε ζεύγος κορυφής-ακμής του G. Αρκεί για να επαληθεύσουμε ότι το G/S δεν περιέχει ένα K_4, καθώς πριν από τη σύμπτυξη του S, το γράφημα G θεωρήσαμε ότι δεν περιέχει K_4 (φτάσαμε τουλάχιστον στη γραμμή 5 πριν από την αναδρομική κλήση).

Η διαδικασία ελέγχου για την γραμμή 5 υλοποιείται με πολυπλοκότητα $O(nm)$. Διατρέχουμε κάθε ακμή uv του G και κατασκευάζουμε την τομή των γειτονιών των u και v, που συμβολίζεται ως $N(u) \cap N(v)$, εξερευνώντας τις γειτονιές των u και v ξεχωριστά. Αυτή η εξερεύνηση γίνεται σε χρόνο $O(n)$ για κάθε κορυφή, με αποτέλεσμα η συνολική χρονική πολυπλοκότητα να είναι $O(nm)$.

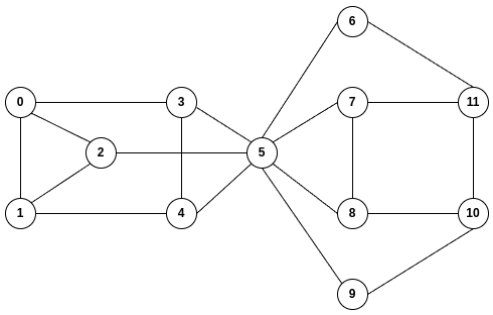
Για να υλοποιήσουμε τον έλεγχο στη γραμμή 9 με χρονική πολυπλοκότητα $O(nm)$, χρησιμοποιούμε το Λήμμα 2.8, το οποίο δηλώνει ότι ο έλεγχος κάθε μπλοκ B του G για έναν ελάχιστο σταθερό διαχωριστή είναι επαρκής. Σύμφωνα με το Λήμμα 2.6, αν υπάρχει ένας τέτοιος διαχωριστής S του B, είναι επίσης ένα σταθερό cutset του B, υπονοώντας την ύπαρξη μιας κορυφής x με $N_B(x)$ που περιέχει το S. Με $|V(B)| \geq 3$ και $|S| \geq 2$, όπως προκύπτει από το Λήμμα 2.8, το B είναι ισχυρά συνεκτικό γράφημα και δεν περιέχει κανένα διαμάντι ή K_4, όντας ένας επαγόμενο υπογράφα του G. Επομένως, το $N_B(x)$ περιέχει το πολύ δύο μέγιστα σταθερά σύνολα, εκ των οποίων το ένα περιέχει το S, καθορίζοντας το ως σταθερό cutset του B από το Λήμμα 2.7.

Για να προχωρήσουμε, υπολογίζουμε πρώτα τα μπλοκ που προκύπτουν από τον αλγόριθμο "Biconnectivity" του Robert Tarjan [28] στο G σε χρόνο $O(n + m)$. Στη συνέχεια, επαναλαμβάνουμε για κάθε κορυφή x στο $V(G)$ και κάθε μπλοκ B του G που περιέχει το x με τουλάχιστον τρεις κορυφές. Για κάθε τέτοιο συνδυασμό, ελέγχουμε αν το $N_B(x)$, το $N_B(x) \setminus \{u\}$, ή το $N_B(x) \setminus \{v\}$ είναι ένα σταθερό cutset του B, όπου uv (αν υπάρχει) είναι η μοναδική ακμή στο $G[N_B(x)]$. Αυτή η δοκιμή μπορεί να ολοκληρωθεί σε χρόνο $O(|V(B)|+|E(B)|)$ χρησιμοποιώντας τον αλγόριθμο αναζήτησης "Depth-first search". Δεδομένου ότι $|V(B)| \leq |E(B)|$ για κάθε μπλοκ B του G, το άθροισμα σε όλες τις επιλογές του B δίνει πολυπλοκότητα $O(m)$. Έτσι, λαμβάνοντας υπόψη όλες τις επιλογές του x, η συνολική πολυπλοκότητα είναι $O(nm)$. Αν βρεθεί ένα σταθερό cutset S κάποιου μπλοκ B, το ανάγουμε σε ένα ελάχιστο σταθερό διαχωριστικό (minimal stable seperator) του B αφαιρώντας επαναληπτικά κορυφές του S και ελέγχοντας αν το σύνολο που προκύπτει παραμένει cutset του B. Αυτή η διαδικασία διαρκεί επίσης $O(nm)$ χρόνο, καθώς κάθε κορυφή του S χρειάζεται να ελεγχθεί μόνο μία φορά. Σύμφωνα με το Λήμμα 2.8, το σύνολο S που προκύπτει είναι ένας ελάχιστος διαχωριστής του G και ικανοποιεί την σχέση $|S| \geq 2$.

Τέλος για την κατασκευή ενός 3-χρωματισμού του γραφήματος G στην γραμμή 13, αν το G είναι τριγωνική λωρίδα, μπορούμε να 3-χρωματίσουμε το G σε χρόνο $O(m)$ αφαιρώντας επαναληπτικά τρίγωνα σε κορυφές βαθμού 3. Αν η G δεν εί-

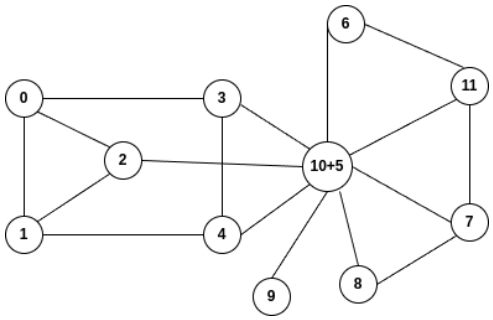
ναι τριγωνική λωρίδα, αλλά τα μπλοκ της G είναι τριγωνικές λωρίδες ή έχουν το πολύ δύο κορυφές, μπορούμε να κατασκευάσουμε ξανά τα μπλοκ του G σε χρόνο $O(n+m)$ χρησιμοποιώντας ξανά τον αλγόριθμο του Robert Tarjan [28]. Στη συνέχεια, 3-χρωματίζουμε όλα τα μπλοκ σε χρόνο $O(n+m)$ εφαρμόζοντας το προηγούμενο επιχείρημα σε κάθε μπλοκ. Τέλος, χρησιμοποιώντας το δέντρο που σχηματίζουν τα μπλοκ του G, μπορούμε να λάβουμε έναν 3-χρωματισμό του G σε χρόνο $O(n)$ συνδυάζοντας τους 3-χρωματισμούς των μπλοκ και αντιμεταθέτοντας τα χρώματα αν είναι απαραίτητο για να ταιριάζουν στα σημεία αποκοπής του G. Αυτή η προσέγγιση οδηγεί σε μια υλοποίηση της γραμμής 13 κατά $O(n+m)$.

Παρακάτω παραθέτουμε ένα παράδειγμα εκτέλεσης του αλγορίθμου για το AT-free γράφημα 2.7



Σχήμα 2.7: Ένα AT-free γράφημα με 12 κόμβους

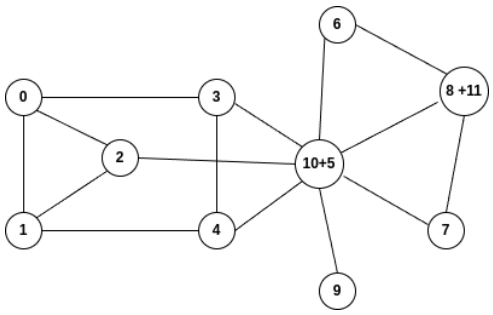
Το γράφημα περνάει τον έλεγχο της γραμμής 1 και της γραμμής 5 αφού δεν περιέχει κάποιο K_4 αλλά και ούτε κάποιο διαμάντι. Ο αλγόριθμος σταματάει στην γραμμή 9 αφού στο γράφημα εντοπίζεται ένας ελάχιστος σταθερός διαχωριστής $S = \{10, 5\}$. Ο αλγόριθμος συμπτύσσει τις ακμές του S και ο αλγόριθμος καλείται ξανά αναδρομικά στο νέο γράφημα 2.8



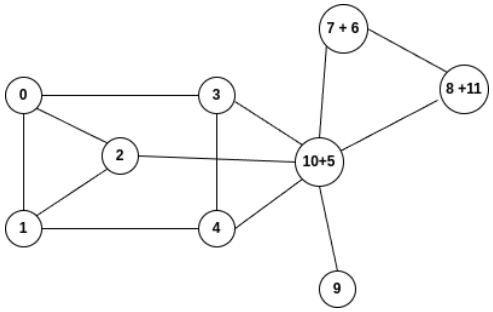
Σχήμα 2.8: Το γράφημα μετά την σύμπτυξη των κορυφών 10,5

Όπως είναι αντιληπτό από το παραπάνω γράφημα, από την σύμπτυξη των ακμών

10,5 το γράφημα έχει δημιουργήσει δύο διαμάντια. Στα παρακάτω δύο γραφήματα φαίνεται πως ο αλγόριθμος 2.3 στην γραμμή 5 εξαλείφει τα διαμάντια συμπτύσσοντας τις κορυφές u, v για τις οποίες ισχύει $|N(u) \cap N(v)| \geq 2$.



Σχήμα 2.9: Το γράφημα μετά την σύμπτυξη των κορυφών 8,11



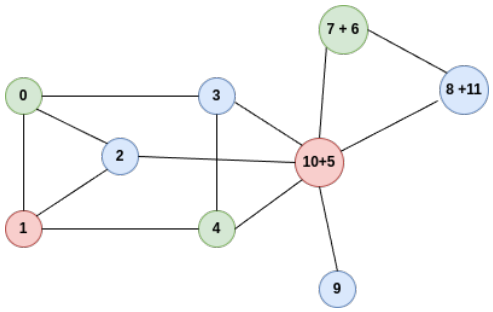
Σχήμα 2.10: Το γράφημα μετά την σύμπτυξη των κορυφών 7,6

Το γράφημα μετά την σύμπτυξη των ακμών 7,6 φτάνει στην γραμμή 13 του αλγορίθμου2.3. Το γράφημα για να μπορέσει να χρωματιστεί πρέπει να χωριστεί σε μπλοκς. Τα μπλοκς του γράφηματος 2.10 φαίνονται στον παρακάτω πίνακα με την σειρά που θα χρωματιστούν.

Μπλοκ	Κορυφές του Μπλοκ
1	$\{1, 10+5, 2, 3, 4, 0\}$
0	$\{8+11, 7+6, 10+5\}$
2	$\{9\}$

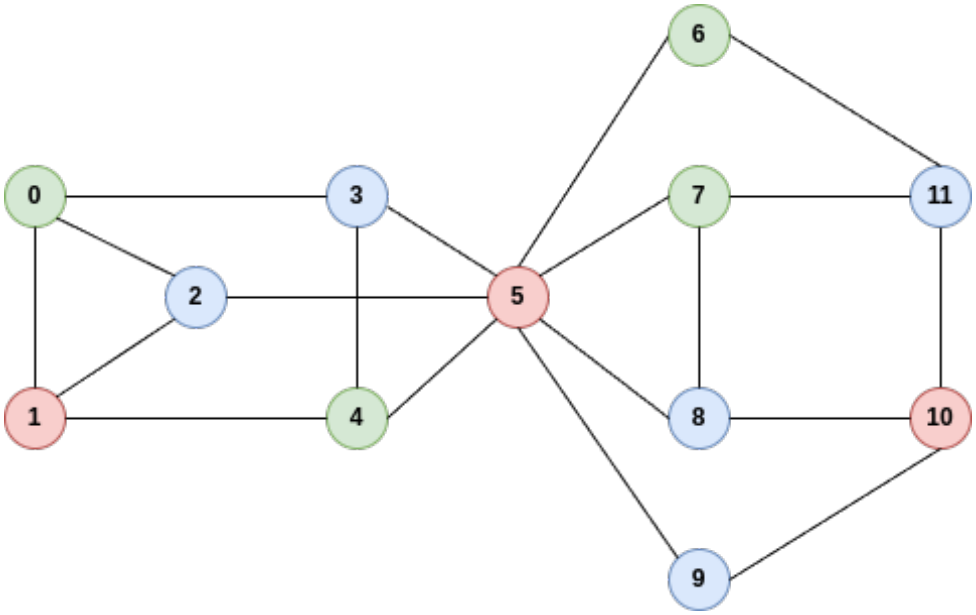
Πίνακας 2.10: Πίνακας με τα μπλοκς του γραφήματος

Χρωματίζουμε εναλλάξ τους κόμβους του κάθε μπλοκ με τα χρώματα κόκκινο, πράσινο και μπλε.



Σχήμα 2.11: Το συμπτυγμένο" γράφημα μετά τον 3-χρωματισμό

Από το Θεώρημα 2.8, γνωρίζουμε ότι οι συμπτυγμένοι κόμβοι στο αρχικό γράφημα (βλ. Σχήμα 2.7) θα έχουν το ίδιο χρώμα. Για παράδειγμα, οι κόμβοι 10 και 5 θα είναι κόκκινοι.



Σχήμα 2.12: Το γράφημα μετά το τέλος του αλγορίθμου

ΚΕΦΑΛΑΙΟ 3

Η Υλοποίηση

Υλοποιήσαμε όλους τους αλγορίθμους που αναφέραμε στο Κεφάλαιο 2 και σχεδιάσαμε μια γραφική διαπροσωπεία για την οπτικοποίηση της εισόδου και των αποτελεσμάτων. Η υλοποίηση έγινε στη γλώσσα προγραμματισμού **Python 3** και η οπτικοποίηση με την βιβλιοθήκη **PyVis**[25]

3.1 Οργάνωση Κώδικα

Η γενική αρχιτεκτονική για την επίλυση του κάθε αλγορίθμου είναι η ίδια σε κάθε περίπτωση. Πιο συγκεκριμένα η υλοποίηση οργανώνεται στα αρχεία:

- `main.py` : η αρχή του εκτελέσιμου προγράμματος.
- `graph.py` : η διευθύνουσα κλάση του γραφήματος.
- `polynomial_time_algorithm.py` : η διευθύνουσα κλάση για την υλοποίηση του αλγορίθμου.

Επιπλέον αρχεία υλοποιούν κλάσεις οι οποίες είναι απαραίτητες για την υλοποίηση των επιμέρους αλγορίθμων. Αναλυτικά θα τις αναλύσουμε στις αντίστοιχες ενότητες.

3.2 Είσοδος - Έξοδος

Κάθε αλγόριθμος που έχουμε υλοποιήσει παίρνει σαν είσοδο ένα AT-free γράφημα. Η αναπαράσταση του γραφήματος έχει γίνει με την κλάση Graph. Έχουμε επιλέξει ο χρήστης μπορεί να ορίσει ένα γράφημα σε αρχείο JSON καθώς η διαχείριση του είναι αρκετά εύκολη από γλώσσες προγραμματισμού όπως η **Python**.

Το αρχείο JSON έχει τα κλειδιά "num_of_vertices" και "num_of_edges" τα οποία δηλώνουν τον αριθμό των κόμβος του γραφήματος και ακμών αντίστοιχα. Οι τιμές που παίρνουν είναι θετική ακέραιοι $\mathbb{N}$. Επιπλέον υπάρχει το κλειδί "edges" το οποίο έχει όλες τις ακμές του γραφήματος. Η δήλωση των ακμών γίνεται με έναν δισδιάστατο πίνακα που περιέχει ζεύγη ακεραίων που υποδηλώνουν τους συνδεδεμένους κόμβους.

Το αρχείο υφίσταται μια ανάλυση προκειμένου να επιβεβαιωθεί η απουσία AT (asteroidal triple). Αυτή η ανάλυση δεν είναι η βέλτιστη αλλά μας διασφαλίζει ότι το γράφημα είναι AT-free. Η κλάση για την ανάλυση του αρχείου:

graph_parser.py

```
import networkx as nx
import json
from graph import Graph
from itertools import combinations


class GraphParser:
    @staticmethod
    def parse_graph_from_file(file_path):
        with open(file_path, 'r') as file:
            data = json.load(file)

        num_of_vertices = data['num_of_vertices']
        num_of_edges = data['num_of_edges']
        edges = [tuple(edge) for edge in data['edges']] # Convert edge lists to tuples

        # Verify if we got the correct number of edges
        if len(edges) != num_of_edges:
            raise ValueError("The number of edges specified does not match the number of edges
                parsed.")

        if GraphParser.has_asteroidal_triple(num_of_vertices, edges):
```

```
            raise ValueError("The graph is no AT-free.")
        else:
            return Graph(num_of_vertices, num_of_edges, edges)

    @staticmethod
    def has_asteroidal_triple(num_of_vertices, edges):
        # Create a graph
        G = nx.Graph()
        G.add_nodes_from(range(num_of_vertices))
        G.add_edges_from(edges)

        # Generate all possible combinations of 3 vertices
        vertex_combinations = combinations(range(num_of_vertices), 3)

        # Check for each combination
        for v1, v2, v3 in vertex_combinations:
            if GraphParser.has_path_avoiding_neighbors(G, v1, v2, v3) \
                    and GraphParser.has_path_avoiding_neighbors(G, v2,v3,v1) \
                        and GraphParser.has_path_avoiding_neighbors(G, v3, v1, v2):
                print("Found an asteroidal triple: ", v1, v2, v3)
                return True
        return False

    @staticmethod
    def has_path_avoiding_neighbors(G, source, target, avoid):
        """
        Check if there is a path from source to target in graph G
        avoiding neighbors of avoid.
        """
        neighbors_to_avoid = set(G.neighbors(avoid))
        visited = set()
        queue = [source]

        while queue:
            node = queue.pop(0)
            visited.add(node)
            if node == target:
                return True
            for neighbor in G.neighbors(node):
                if neighbor != avoid and neighbor not in visited and neighbor not in queue and
                    neighbor not in neighbors_to_avoid:
```

```
                queue.append(neighbor)

    return False
```

Αν η ανάλυση του γραφήματος είναι επιτυχημένη, τότε δημιουργείται ένα αντικείμενο τύπου Graph. Παρακάτω παρουσιάζεται ο ορισμός του Graph, καθώς και τα πεδία που δημιουργούνται. Τα συγκεκριμένα πεδία είναι κοινά και για τους τρεις αλγορίθμους που έχουμε υλοποιήσει.

Ορισμός κλάσης Graph

```
class Graph:
        def __init__(self, num_of_vertices,  num_of_edges, edges, vertices=None):
                self.num_of_vertices = num_of_vertices
                self.num_of_edges = num_of_edges
                self.vertices = set(str(i) for i in range(num_of_vertices)) if vertices is
                     None else vertices
                self.edges = set((str(u), str(v)) for u, v in edges)
                self.adjacency_list = {str(vertex): set() for vertex in self.vertices}
                for edge in self.edges:
                        u, v = edge
                        self.adjacency_list[u].add(v)
                        self.adjacency_list[v].add(u)
```

Η έξοδος στον αλγόριθμο υπολογισμού μέγιστου ανεξάρτητου συνόλου είναι ο αριθμός ανεξαρτησίας του γραφήματος καθώς και ένα σετ από κόμβους που είναι ένα μέγιστο ανεξάρτητο σύνολο.

Ο αλγόριθμος υπολογισμού του ελάχιστου κυρίαρχου συνόλου επιστρέφει ένα σετ κόμβων που αποτελεί ένα ελάχιστο κυρίαρχο σύνολο.

Για το πρόβλημα του 3-χρωματισμού, η έξοδος του αλγορίθμου παρουσιάζεται σε μορφή λεξικού, όπου κάθε κόμβος του γραφήματος αντιστοιχεί σε ένα κλειδί, και η σχετική τιμή είναι το χρώμα που έχει ανατεθεί σε αυτόν τον κόμβο (π.χ., ”red”, ”green”, ”blue”).

Σε όλους τους αλγορίθμους εκτελούμε την συνάρτηση `show()` της κλάσης Graph. Η συνάρτηση επιστρέφει ένα αρχείο HTML το οποίο περιέχει την οπτικοποίηση του γραφήματος.

3.3 Δομές Δεδομένων

Οι δομές που χρησιμοποιήσαμε για την επίλυση των αλγορίθμων είναι δομές που μας παρέχει η γλώσσα προγραμματισμού **Python**.Η επιλογή των δομών έγινε με σκοπό τη διατήρηση της χωρικής και χρονικής πολυπλοκότητας για κάθε αλγόριθμο. Παρόλο που σε ορισμένες περιπτώσεις θα μπορούσαμε να επιτύχουμε μείωση της χωρικής πολυπλοκότητας εάν είχαμε επιλέξει τη δομή της λίστας αντί του λεξικού, προτιμήσαμε να διατηρήσουμε τη σαφήνεια και την ευανάγνωστη μορφή του κώδικά μας. Κατανοούμε ότι η κατανόηση του κώδικα είναι εξίσου σημαντική με τυχόν οικονομία σε χώρο που θα μπορούσαμε να επιτύχουμε.

Αρχικά η κλάση Graph 3.2 διαχειρίζεται τις παρακάτω δομές δεδομένων που είναι κοινές και για τους τρεις αλγορίθμους που έχουμε υλοποιήσει:

1. πλειάδα `tuple`: Οι πλειάδες (tuples) στην **Python** αποτελούν μια αμετάβλητη δομή δεδομένων που επιτρέπει την ομαδοποίηση στοιχείων με σειρά. Οι ακμές του γραφήματος αναπαρίστανται ως πλειάδες, δηλαδή ζεύγη ακεραίων αριθμών.

2. δομή συνόλων `set`: Τα σύνολα στην **Python** είναι δομές δεδομένων που αποθηκεύουν μοναδικά στοιχεία, χωρίς να διατηρούν τη σειρά εισαγωγής. Η μοναδικότητα επιτυγχάνεται μέσω της χρήσης hash tables, που επιτρέπουν γρήγορες λειτουργίες εισαγωγής, διαγραφής και αναζήτησης με χρόνο εκτέλεσης $O(1)$ σε μέσο όρο. Κατά τη διάρκεια της εισαγωγής, το στοιχείο υπολογίζει ένα hash value, το οποίο καθορίζει τη θέση του στο hash table. Κατά την αναζήτηση, το hash table χρησιμοποιείται για να εντοπίσει γρήγορα τη θέση του στοιχείου. Παρόλα αυτά, σε περιπτώσεις συγκρούσεων (hash collisions), όπου δύο στοιχεία έχουν το ίδιο hash value, η απόδοση μπορεί να μειωθεί, αυξάνοντας τον χρόνο αναζήτησης στο χειρότερο σενάριο σε $O(n)$. Χρησιμοποιούμε τα σύνολα για να ορίσουμε τους κόμβους και τις ακμές του γραφήματος.

3. λεξικό `dictionary`: Τα λεξικά στην **Python** είναι δομές δεδομένων που επιτρέπουν την αποθήκευση και την ανάκτηση δεδομένων με βάση τα κλειδιά. Κάθε κλειδί στο λεξικό είναι μοναδικό και συσχετίζεται με μια τιμή. Η κύρια ιδιότητα των λεξικών είναι η αποδοτική αναζήτηση, καθώς η πρόσβαση σε μια τιμή γίνεται σε σταθερό χρόνο $O(1)$ χάρη στη χρήση hash table. Τα λεξικά μπορούν να περιέχουν διάφορους τύπους δεδομένων ως τιμές, όπως

αριθμούς, συμβολοσειρές, λίστες, και ακόμη και άλλα λεξικά. Η ενημέρωση, εισαγωγή, και διαγραφή στοιχείων σε ένα λεξικό γίνεται με διαφορετικές μεθόδους, προσφέροντας μεγάλη ευελιξία στη διαχείριση τους. Χρησιμοποιούμε τα λεξικά για να ορίσουμε τις λίστες γειτνίασης. Κάθε κόμβος (κλειδί) είναι συσχετισμένος με ένα σύνολο από γειτονικούς κόμβους. Η επανάληψη μέσω των ακμών και η ενημέρωση των συνόλων γειτονικών κόμβων είναι γραμμική ως προς τον αριθμό των ακμών. Συνεπώς, ο συνολικός χρόνος εκτέλεσης για τη δημιουργία των λιστών γειτνίασης είναι $O(m)$, όπου m είναι ο αριθμός των ακμών που παρέχονται στη κλάση Graph.

Πιο συγκεκριμένα θα δούμε τις δομές δεδομένων που χρησιμοποιήσαμε για κάθε αλγόριθμο στις παρακάτω παραγράφους.

Ο Αλγόριθμος υπολογισμός μέγιστου ανεξάρτητου συνόλου 2.1 χρειάζεται και διαχειρίζεται επιπλέον τις παρακάτω δομές δεδομένων:

1. λεξικά: Η κλάση Graph αυτού το αλγορίθμου έχει επιπλέον μεταβλητές για την επίλυση του. Πιο συγκεκριμένα ένα λεξικό για τα $Components$ χρησιμοποιεί ως κλειδί μια πλειάδα, όπου το πρώτο στοιχείο αναφέρεται στον κόμβο προέλευσης, ενώ το δεύτερο στοιχείο (αριθμός i) καθορίζει τη μοναδικότητα του $Component$. Αυτό σημαίνει ότι για κάθε κόμβο, μπορεί να υπάρχουν πολλαπλά $Components$. Η χρήση μιας πλειάδας ως κλειδί επιτρέπει τον εύκολο και γρήγορο προσδιορισμό και ανάκτηση των $Components$ που αντιστοιχούν σε έναν συγκεκριμένο κόμβο. Επιπλέον χρησιμοποιούμε ένα λεξικό το οποίο έχει κλειδιά τους κόμβους x του γραφήματος και σαν τιμές τον αριθμό των $Componentes$ που έχουν στο $G - N[x]$. Αντίστοιχα για τα $Intervals$ ένα λεξικό με κλειδιά πλειάδες με μη γειτονικές ακμές. Ένα λεξικό που για όλες τις μη γειτονικές κορυφές x και y έχει ως τιμή ένας δείκτης $P(x, y)$ στο $Component$ του $C^x(y)$. Για την κλάση PolynomialTimeAlgorithm, απαιτούνται δύο επιπλέον λεξικά, τα οποία έχουν κλειδιά παρόμοια με τα λεξικά που διατηρούν τις πληροφορίες για τα $Components$ και τα $Intervals$. Τα νέα λεξικά, λειτουργούν ως αντίγραφα για την αποθήκευση των κόμβων που χρησιμοποιήθηκαν για τον υπολογισμό των μέγιστων τιμών από τους τύπους 2.2 και 2.3. Η χρήση αυτών των δομών είναι απαραίτητη προκειμένου ο αλγόριθμος να είναι σε θέση να επιστρέψει, εκτός από τον αριθμό ανεξαρτησίας, και το ανεξάρτητο σύνολο που προκύπτει από την εκτέλεση των εν λόγω τύπων.

Για τον Αλγόριθμο υπολογισμού ελάχιστου κυρίαρχου συνόλου 2.2 χρειάζεται η παρακάτω δομή δεδομένων:

1. λεξικά: Στην κλάση PolynomialTimeAlgorithm όπου υλοποιούμε τον αλγόριθμο έχουμε αναπαραστήσει την δομή H, η οποία διατηρεί τα BFS-levels του γραφήματος για κάποιον κόμβο $x \in V$, με ένα λεξικό. Το παίρνει ως κλειδί έναν θετικό ακέραιο που δηλώνει το επίπεδο του BFS και σαν τιμή ένα σύνολο `set` με τους κόμβους που υπάρχουν στο εκάστοτε επίπεδο.

   Επιπλέον ένα λεξικό για να ορίσουμε την δομή A όπου σαν κλειδί έχει τον αριθμό i σε κάθε βρόχο της εκτέλεση του αλγορίθμου και σαν τιμή έχει μια ουρά που περιέχει μια ταξινομημένη πλειάδα $(S, S, val(S))$.

Τέλος για την επίλυση του Πρόβληματος 3-Χρωματισμού χρησιμοποιήσαμε επιπλέον τις δομές:

1. λεξικό: Στην κλάση Graph χρειαστήκαμε μια δομή λεξικού για να αποθηκεύει τα $Blocks$ του γραφήματος.

2. σύνολο: Για τον χρωματισμό του γραφήματος χρειάστηκε να αποθηκεύσουμε σε ένα σύνολο τα $cutpoints$ του.

3.4 Υπολογισμός Ελάχιστου Κυρίαρχου Συνόλου

Στο παρόν κεφάλαιο, παρουσιάζουμε ενδεικτικές κλάσεις που αποτελούν το αλγοριθμικό τμήμα για τον υπολογισμό του Ελάχιστου Κυρίαρχου Συνόλου. Προχωρούμε στην παρουσίαση αυτών των κλάσεων με τη σειρά που καλούνται στο πλαίσιο του συνολικού αλγοριθμικού σχεδιασμού. Σημειώνουμε πως παραλείπουμε τις κλάσεις που υλοποιούν αλγορίθμους από άλλες εργασίες, όπως ο αλγόριθμος του Bucket sort.

3.4.1 Η Κλάση PolynomialTimeAlgorithm

Η κλάση `PolynomialTimeAlgorithm` αποτελεί την κύρια κλάση για τον υπολογισμό του μέγιστου ανεξάρτητου συνόλου.

polynomial_time_algorithm.py

```
from graph import Graph
from bucket_sort import BucketSort

from itertools import combinations



class PolynomialTimeAlgorithm:
    def __init__(self,graph : Graph):
        self.graph = graph

        self.max_interval_vertices_set = {}
        self.max_component_vertices_set = {}




    def computing_independent_set(self):
        # Step 1. For every x □V compute all components C1x , C2x , . . . , Cr(x)
        # Step 2. For every pair of nonadjacent vertices x and y compute the interval I(x, y).
        # Step 3. Sort all the components and intervals according to nondecreasing number of
        # vertices.
        # Step 4. Compute α(C) and α(I) for each component C and each interval I in the
        # order of Step 3.
        # Step 5. Compute α(G).

        # Step 1
        self.graph.compute_all_components()
        print("\nComponents:")
        for component in self.graph.components.values():
            print(component)

        print("self.graph.num_of_components:", self.graph.num_of_components)



        # Step 2
        self.graph.compute_all_intervals()
        print("\nIntervals:")
        for interval in self.graph.intervals.values():
            print(interval)
```

```
    # Step 3

    # Sort components
    bucket_sort_components = BucketSort(list(self.graph.components.keys()), key_function=
        lambda x: len(self.graph.components[x]))
    sorted_components = bucket_sort_components.sort()

    # Sort intervals
    bucket_sort_intervals = BucketSort(list(self.graph.intervals.keys()), key_function=
        lambda x: len(self.graph.intervals[x]))
    sorted_intervals = bucket_sort_intervals.sort()

    print("\nSorted components:", sorted_components, len(sorted_components), type(
        sorted_components))
    print("\nSorted intervals:", sorted_intervals, len(sorted_intervals), type(
        sorted_intervals))
    # Step 4
    for key in sorted_components:
        self.alpha_C(key)

    # Step 5
    print('\n\n\n')
    print("Graph intependent set number:", self.alpha_G())


def alpha_C(self,component_key):
    component = self.graph.components[component_key]
    print(component)
    if component.alpha is not None:
        return component.alpha

    x = component.x

    component_vertices = component.vertices

    max_alpha = 0
    self.max_component_vertices_set[component_key] = set()
    max_y = None
    max_D_y_components_vertices = set()
    for y in component_vertices: # y E Cx
```

```
            alpha_I_xy = self.alpha_I((x, y))

            D_y_components = self.compute_D_iy(y, component_vertices)

            alpha_D_sum = sum(self.alpha_C(D_iy) for D_iy in D_y_components)
            if max_alpha <= alpha_I_xy + alpha_D_sum:
                max_alpha = alpha_I_xy + alpha_D_sum
                max_y = y
                max_D_y_components_vertices = set()
                for D_iy in D_y_components:
                    max_D_y_components_vertices = max_D_y_components_vertices.union(self.
                        max_component_vertices_set[D_iy])


        alpha_value = 1 + max_alpha

        self.max_component_vertices_set[component_key] = max_D_y_components_vertices.union({
            max_y})
        if (x,max_y) in self.graph.intervals:
            self.max_component_vertices_set[component_key] = self.max_component_vertices_set[
                component_key].union(self.max_interval_vertices_set[(x,max_y)])
        print(f"self.max_component_vertices_set[{component_key}]:{self.
            max_component_vertices_set[component_key]}")

        component.alpha = alpha_value


        print(f'Computed Component,{component_key} Alpha: {self.graph.components[component_key]}
            ')

        return alpha_value

    def alpha_I(self, I):

        try:
            interval = self.graph.intervals[I]
            print(interval)
            if interval.alpha is not None:
                print(f"Interval ({I}) is already computed with alpha: {interval.alpha}")
                return interval.alpha
```

```
x = interval.x
y = interval.y
I_vertices = interval.vertices

max_alpha = 0
self.max_interval_vertices_set[I] = set()
max_s = None
max_C_s_components_vertices = set()
for s in I_vertices:
    alpha_I_xs = self.alpha_I((x, s))
    alpha_I_sy = self.alpha_I((s, y))
    C_s_components = self.compute_C_is(s, I_vertices)
    alpha_C_sum = sum(self.alpha_C(C_is) for C_is in C_s_components)
    # max_alpha = max(max_alpha, alpha_I_xs + alpha_I_sy + alpha_C_sum)
    if max_alpha <= alpha_I_xs + alpha_I_sy + alpha_C_sum:
        max_alpha = alpha_I_xs + alpha_I_sy + alpha_C_sum
        max_s = s
        max_C_s_components_vertices = set()
        for C_is in C_s_components:
            max_C_s_components_vertices = max_C_s_components_vertices.union(self.
                max_component_vertices_set[C_is])


alpha_value = 1 + max_alpha
print("interval alpha:", alpha_value)

print("max_C_s_components_vertices", max_C_s_components_vertices)
interval_x_s_vertices = self.graph.intervals[(x, max_s)].vertices if (x, max_s) in
    self.graph.intervals else set()
print("interval_x_s_vertices:", interval_x_s_vertices)
interval_s_y_vertices = self.graph.intervals[(max_s, y)].vertices if (max_s, y) in
    self.graph.intervals else set()
print("interval_s_y_vertices:", interval_s_y_vertices)
self.max_interval_vertices_set[I] = interval_x_s_vertices.union(interval_s_y_vertices
    ).union(max_C_s_components_vertices).union({max_s})
print(f"self.max_interval_vertices_set[{I}]: {self.max_interval_vertices_set[I]}")

interval.alpha = alpha_value


print(f'Computed Interval({I}) alpha:{self.graph.intervals[I]}')
```

```
            return alpha_value
        except:
            print("Interval", I, "not found")
            return 0

def compute_components_subset(self, vertex, target_vertices, num_components):
    """
    Computes the components of G - N[vertex] contained in the target set.

    :param vertex: The vertex whose neighborhood defines the components.
    :param target_vertices: The set of vertices that the component should be a subset of.
    :param num_components: The number of components to consider.
    :return: A list of component keys whose vertices are a subset of the target set.
    """
    computed_components = []

    for i in range(num_components):
        component_key = (vertex, i)
        component = self.graph.components[component_key]
        if component.vertices.issubset(target_vertices):
            computed_components.append(component_key)

    return computed_components

def compute_D_iy(self, y, Cx_vertices):
    """
    D_iy are the components of G - N[y] contained in Cx.
    """
    return self.compute_components_subset(y, Cx_vertices, self.graph.num_of_components[y])

def compute_C_is(self, s, I_vertices):
    """
    C_is are the components of G - N[s] contained in I.
    """
    return self.compute_components_subset(s, I_vertices, self.graph.num_of_components[s])

def alpha_G(self):
    """
    G is the graph
    """
```

```
        max_alpha = 0
        max_alpha_component = None
        for x in self.graph.vertices:
            alpha_C_sum = sum(self.alpha_C((x,i)) for i in range(self.graph.num_of_components[x])
                )
            if max_alpha < alpha_C_sum:
                max_alpha = alpha_C_sum
                max_alpha_component = x



        print("max_alpha_component:", max_alpha_component)
        independent_set = {max_alpha_component}
        for i in range(self.graph.num_of_components[max_alpha_component]):
            component_key = (max_alpha_component, i)
            independent_set = independent_set.union(self.max_component_vertices_set[component_key
                ])
        print("max independent set:", independent_set)
        self.graph.independent_set = independent_set
        # self.graph.show()
        return max_alpha + 1

    def run(self):
        return self.computing_independent_set()
```

Εξηγούμε αναλυτικά τις συναρτήσεις της κλάσης:

1. Κατασκευαστής (__init__): Δέχεται ένα αντικείμενο Graph και αρχικοποιεί τα χαρακτηριστικά της κλάσης, όπως τα σύνολα "max_interval_vertices_set" και "max_component_vertices_set".

2. Η συνάρτηση `computing_independent_set`: Είναι η κύρια συνάρτηση που εκτελεί τα βήματα του αλγορίθμου για τον υπολογισμό του μέγιστου ανεξάρτητου συνόλου.

3. Η συνάρτηση `alpha_C`: Υλοποιεί το λήμμα 2.3.

4. Η συνάρτηση `alpha_I`: Υλοποιεί το λήμμα 2.2.

5. Η συνάρτηση `alpha_G`: Υλοποιεί το λήμμα 2.1.

6. Η συνάρτηση `compute_components_subset` λαμβάνει ως είσοδο έναν κόμβο `vertex`, ένα σύνολο κορυφών `target_vertices`, και τον αριθμό των *Components* που πρέπει να εξεταστούν `num_components`. Επιστρέφει μια λίστα με τα κλειδιά των *Components* που ανήκουν στο σύνολο `target_vertices`. Αυτή η λειτουργία είναι κρίσιμη για τον υπολογισμό των ανεξάρτητων συνόλων στη βάση των Λημμάτων 2.2 και 2.3.

7. Η συνάρτηση `compute_D_iy` χρησιμοποιεί την `compute_components_subset` για να υπολογίσει τα *Components* `D_iy`, δηλαδή εκείνες που περιέχονται στο `Cx_vertices` και ανήκουν στη γειτονιά του `y`. Η είσοδος αυτής της συνάρτησης περιλαμβάνει τον κόμβο `y` και το υποσύνολο κορυφών `Cx_vertices`, ενώ η έξοδός της είναι η λίστα των *Components* `D_iy`.

8. Η συνάρτηση `compute_C_is` λειτουργεί παρόμοια με την `compute_D_iy`, αλλά χρησιμοποιείται για τον υπολογισμό των *Components* `C_is`, που περιέχονται στο `I_vertices` και ανήκουν στη γειτονιά του `s`. Η είσοδος περιλαμβάνει τον κόμβο `s` και το υποσύνολο κορυφών `I_vertices`, ενώ η έξοδός της είναι η λίστα των *Components* `C_is`.

3.4.2 Η Κλάση Graph

Η κλάση `Graph` αποτελεί τη βασική δομή για την παρουσίαση του AT-free γράφηματος, προσφέροντας τις απαραίτητες λειτουργίες για τον υπολογισμό του μέγιστου ανεξάρτητου συνόλου.

graph.py

```
from itertools import combinations
from pyvis.network import Network
from component import Component
from interval import Interval

import os

class Graph:
    show_count = 0 # Class-level variable to keep track of show calls

    def __init__(self, num_of_vertices, num_of_edges, edges, vertices=None):
        self.num_of_vertices = num_of_vertices
```

```
    self.num_of_edges = num_of_edges
    self.vertices = set(str(i) for i in range(num_of_vertices)) if vertices is None else
        vertices
    self.edges = set((str(u), str(v)) for u, v in edges)
    self.adjacency_list = {str(vertex): set() for vertex in self.vertices}
    for edge in self.edges:
        u, v = edge
        self.adjacency_list[u].add(v)
        self.adjacency_list[v].add(u)

    self.components = {}
    self.num_of_components = {}
    self.intervals = {}
    self.non_adjacent_vertices_component_pointer = {}

    self.independent_set = set()


def compute_all_components(self):
    for vertex in self.vertices:
        vertex_components = self.compute_components_of_vertex(vertex)
        self.num_of_components[vertex] = len(vertex_components)
        for i in range(len(vertex_components)):
            self.components[(vertex, i)] = Component(vertex, i, vertex_components[i])

def compute_components_of_vertex(self,vertex):
    components_vertices = self.vertices - self.closed_neighborhood(vertex)
    components = []
    visited = set()

    def dfs(node,component):
        visited.add(node)
        component.add(node)
        self.non_adjacent_vertices_component_pointer[(vertex, node)] = (vertex, len(
            components))
        for neighbor in self.adjacency_list[node]:
            if neighbor not in visited and neighbor in components_vertices:
                dfs(neighbor, component)

    for v in components_vertices:
        if v not in visited:
```

```
            component = set()
            dfs(v, component)
            components.append(component)

    return components

def closed_neighborhood(self, vertex):
    return {vertex}.union(self.adjacency_list[vertex])

def closed_neighborhood_of_set(self, vertex_set):
    closed_neighborhood = set()
    for vertex in vertex_set:
        closed_neighborhood = closed_neighborhood.union(self.closed_neighborhood(vertex))
    return closed_neighborhood

def compute_all_intervals(self):
    for x in self.vertices:
        for y in self.vertices:
            if x != y and y not in self.adjacency_list[x]:
                interval = self.compute_interval(x, y)
                if interval:
                    self.intervals[(x, y)] = Interval(x, y, interval)
    return self.intervals



def compute_interval(self, x, y):
    if y in self.adjacency_list[x]:
        raise ValueError("Vertices x and y are adjacent. Interval can only be computed for
            nonadjacent vertices.")

    Cx_y = None
    Cy_x = None

    Cx_y_pointer = self.non_adjacent_vertices_component_pointer[(x, y)]
    Cy_x_pointer = self.non_adjacent_vertices_component_pointer[(y, x)]

    Cx_y = self.components[Cx_y_pointer].vertices
    Cy_x = self.components[Cy_x_pointer].vertices

    interval = Cx_y.intersection(Cy_x) if Cx_y is not None and Cy_x is not None else None
    return interval
```

```
def show(self, graph_name='graph'):
    Graph.show_count += 1
    print("Showing graph:", Graph.show_count)
    net = Network(height="500px", width="100%", bgcolor="#222222", font_color="white")

    for vertex in self.vertices:
        if vertex in self.independent_set:
            net.add_node(vertex, color="red")
        else:
            net.add_node(vertex)



    for edge in self.edges:
        u, v = edge
        net.add_edge(u, v, color="white")

    script_dir = os.path.dirname(os.path.abspath(__file__))
    output_dir = os.path.join(script_dir, '..', 'output-graphs')
    file_name = os.path.join(output_dir, f"{graph_name}-{Graph.show_count}.html")
    net.show(file_name)
```

Εξηγούμε αναλυτικά τις συναρτήσεις της κλάσης:

1. Συνάρτηση `compute_all_components`: Υπολογίζει όλα τα $Components$ για κάθε κορυφή του γραφήματος και τις αποθηκεύει στην λίστα των $Component$.

2. Συνάρτηση `compute_components_of_vertex`: Χρησιμοποιείται για τον υπολογισμό των $Component$ μιας συγκεκριμένης κορυφής.

3. Συνάρτηση `closed_neighborhood`: Επιστρέφει την κλειστή γειτονιά μιας κορυφής.

4. Συνάρτηση `closed_neighborhood_of_set`: Υπολογίζει την κλειστή γειτονιά ενός συνόλου κορυφών.

5. Συνάρτηση `compute_all_intervals`: Υπολογίζει όλα τα $Intervals$ μεταξύ μη γειτονικών κορυφών.

6. Συνάρτηση `compute_interval`: Υπολογίζει το $Inteval$ μεταξύ δύο μη γειτονικών κορυφών.

7. Συνάρτηση show: Χρησιμοποιεί το πακέτο pyvis για να εμφανίσει τον γράφο σε μορφή διαδραστικού δικτύου.

3.4.3 Κλάσεις Component και Interval

component.py

```
class Component:
    def __init__(self, x, i, vertices):
        self.x = x
        self.i = i
        self.vertices = vertices
        self.alpha = None

    def __repr__(self):
        return f"Component(x={self.x}, i={self.i}, vertices={self.vertices}, alpha={self.alpha})
            "

    def __lt__(self, other):
        return len(self.vertices) < len(other.vertices)

    def __len__(self):
        return len(self.vertices)
```

interval.py

```
class Interval:
    def __init__(self, x, y, vertices):
        self.x = x
        self.y = y
        self.vertices = vertices
        self.alpha = None

    def __repr__(self):
        return f"Interval(x={self.x}, y={self.y}, vertices={self.vertices}, alpha={self.alpha})"

    def __lt__(self, other):
        return len(self.vertices) < len(other.vertices)

    def __len__(self):
        return len(self.vertices)
```

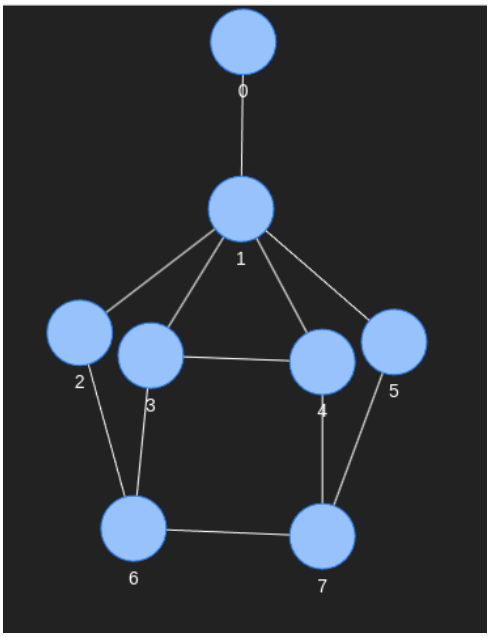
Οι κλάσεις `Component` και `Interval` λειτουργούν ως βοηθητικές δομές για την αναπαράσταση των $Components$ και των $Intervals$ αντίστοιχα στο AT-free γράφημα. Κάθε αντικείμενο αυτών των κλάσεων διαθέτει πληροφορίες σχετικά με τη θέση του στο γράφημα, τις κορυφές που περιέχει, καθώς και ένα πεδίο για την αποθήκευση του αριθμού ανεξαρτησίας alpha. Η υλοποίηση της μεθόδου `__lt__` επιτρέπει τη σωστή λειτουργία της ταξινόμησης, ενώ η μέθοδος `__len__` επιστρέφει τον αριθμό των κορυφών του αντικειμένου, βοηθώντας στη διευκόλυνση της σύγκρισης μεταξύ τους. Με αυτόν τον τρόπο, ο κώδικας γίνεται πιο δομημένος και ευανάγνωστος, ενισχύοντας την κατανόηση του αλγορίθμου.

3.4.4 Παραδείγματα Εκτέλεσης του Προγράμματος

Ως παράδειγμα δείχνουμε πως εκτελούμε τον αλγόριθμο υπολογισμού του μέγιστου ανεξάρτητου συνόλου στο ίδιο γράφημα με την Ενότητα 2.1.

Το γράφημα σε μορφή JSON

```
{
   "num_of_vertices" : 8,
   "num_of_edges" : 11,
   "edges" :
      [
         [0,1],
         [1,2],
         [1,5],
         [1,3],
         [1,4],
         [2,6],
         [3,6],
         [3,4],
         [4,7],
         [6,7],
         [5,7]
      ]
}
```



Σχήμα 3.1: Το γράφημα εισόδου του Αλγορίθμου σχεδιασμένο από την βιβλιοθήκη PyVis

Η εκτέλεση του Αλγορίθμου

```
Components:
Component(x=5, i=0, vertices={'6', '2', '4', '3'}, alpha=None)
Component(x=5, i=1, vertices={'0'}, alpha=None)
Component(x=7, i=0, vertices={'1', '2', '0', '3'}, alpha=None)
Component(x=2, i=0, vertices={'5', '4', '7', '3'}, alpha=None)
Component(x=2, i=1, vertices={'0'}, alpha=None)
Component(x=3, i=0, vertices={'5', '7'}, alpha=None)
Component(x=3, i=1, vertices={'2'}, alpha=None)
Component(x=3, i=2, vertices={'0'}, alpha=None)
Component(x=6, i=0, vertices={'5', '4', '0', '1'}, alpha=None)
Component(x=4, i=0, vertices={'5'}, alpha=None)
Component(x=4, i=1, vertices={'6', '2'}, alpha=None)
Component(x=4, i=2, vertices={'0'}, alpha=None)
Component(x=0, i=0, vertices={'5', '7', '2', '3', '6', '4'}, alpha=None)
Component(x=1, i=0, vertices={'6', '7'}, alpha=None)
self.graph.num_of_components: {'5': 2, '7': 1, '2': 2, '3': 3, '6': 1, '4': 3, '0': 1, '1': 1}

Intervals:
Interval(x=5, y=2, vertices={'4', '3'}, alpha=None)
Interval(x=5, y=6, vertices={'4'}, alpha=None)
```

```
Interval(x=7, y=2, vertices={'3'}, alpha=None)
Interval(x=7, y=0, vertices={'2', '3'}, alpha=None)
Interval(x=2, y=5, vertices={'4', '3'}, alpha=None)
Interval(x=2, y=7, vertices={'3'}, alpha=None)
Interval(x=6, y=5, vertices={'4'}, alpha=None)
Interval(x=6, y=0, vertices={'5', '4'}, alpha=None)
Interval(x=0, y=7, vertices={'2', '3'}, alpha=None)
Interval(x=0, y=6, vertices={'5', '4'}, alpha=None)

Sorted components: [('5', 1), ('2', 1), ('3', 1), ('3', 2), ('4', 0), ('4', 2), ('3', 0), ('4',
     1), ('1', 0), ('5', 0), ('7', 0), ('2', 0), ('6', 0), ('0', 0)] 14 <class 'list'>

Sorted intervals: [('5', '6'), ('7', '2'), ('2', '7'), ('6', '5'), ('5', '2'), ('7', '0'),
    ('2', '5'), ('6', '0'), ('0', '7'), ('0', '6')] 10 <class 'list'>

Dynamic programming part of the algorithm, computing alpha values for components and intervals:
Component(x=5, i=1, vertices={'0'}, alpha=None)
Interval ('5', '0') not found
Computed Component,('5', 1) Alpha: Component(x=5, i=1, vertices={'0'}, alpha=1)
Component(x=2, i=1, vertices={'0'}, alpha=None)
Interval ('2', '0') not found
Computed Component,('2', 1) Alpha: Component(x=2, i=1, vertices={'0'}, alpha=1)
Component(x=3, i=1, vertices={'2'}, alpha=None)
Interval ('3', '2') not found
Computed Component,('3', 1) Alpha: Component(x=3, i=1, vertices={'2'}, alpha=1)
Component(x=3, i=2, vertices={'0'}, alpha=None)
Interval ('3', '0') not found
Computed Component,('3', 2) Alpha: Component(x=3, i=2, vertices={'0'}, alpha=1)
Component(x=4, i=0, vertices={'5'}, alpha=None)
Interval ('4', '5') not found
Computed Component,('4', 0) Alpha: Component(x=4, i=0, vertices={'5'}, alpha=1)
Component(x=4, i=2, vertices={'0'}, alpha=None)
Interval ('4', '0') not found
Computed Component,('4', 2) Alpha: Component(x=4, i=2, vertices={'0'}, alpha=1)
Component(x=3, i=0, vertices={'5', '7'}, alpha=None)
Interval ('3', '5') not found
Interval ('3', '7') not found
Computed Component,('3', 0) Alpha: Component(x=3, i=0, vertices={'5', '7'}, alpha=1)
Component(x=4, i=1, vertices={'6', '2'}, alpha=None)
Interval ('4', '6') not found
Interval ('4', '2') not found
```

```
Computed Component,('4', 1) Alpha: Component(x=4, i=1, vertices={'6', '2'}, alpha=1)
Component(x=1, i=0, vertices={'6', '7'}, alpha=None)
Interval ('1', '6') not found
Interval ('1', '7') not found
Computed Component,('1', 0) Alpha: Component(x=1, i=0, vertices={'6', '7'}, alpha=1)
Component(x=5, i=0, vertices={'6', '2', '4', '3'}, alpha=None)
Interval(x=5, y=6, vertices={'4'}, alpha=None)
Interval ('5', '4') not found
Interval ('4', '6') not found
interval alpha: 1
Computed Interval(('5', '6')) alpha:Interval(x=5, y=6, vertices={'4'}, alpha=1)
Interval(x=5, y=2, vertices={'4', '3'}, alpha=None)
Interval ('5', '4') not found
Interval ('4', '2') not found
Interval ('5', '3') not found
Interval ('3', '2') not found
interval alpha: 1
Computed Interval(('5', '2')) alpha:Interval(x=5, y=2, vertices={'4', '3'}, alpha=1)
Interval ('5', '4') not found
Component(x=4, i=1, vertices={'6', '2'}, alpha=1)
Interval ('5', '3') not found
Component(x=3, i=1, vertices={'2'}, alpha=1)
Computed Component,('5', 0) Alpha: Component(x=5, i=0, vertices={'6', '2', '4', '3'}, alpha=2)
Component(x=7, i=0, vertices={'1', '2', '0', '3'}, alpha=None)
Interval ('7', '1') not found
Interval(x=7, y=2, vertices={'3'}, alpha=None)
Interval ('7', '3') not found
Interval ('3', '2') not found
interval alpha: 1
Computed Interval(('7', '2')) alpha:Interval(x=7, y=2, vertices={'3'}, alpha=1)
Component(x=2, i=1, vertices={'0'}, alpha=1)
Interval(x=7, y=0, vertices={'2', '3'}, alpha=None)
Interval(x=7, y=2, vertices={'3'}, alpha=1)
Interval (('7', '2')) is already computed with alpha: 1
Interval ('2', '0') not found
Interval ('7', '3') not found
Interval ('3', '0') not found
Component(x=3, i=1, vertices={'2'}, alpha=1)
interval alpha: 2
Computed Interval(('7', '0')) alpha:Interval(x=7, y=0, vertices={'2', '3'}, alpha=2)
Interval ('7', '3') not found
```

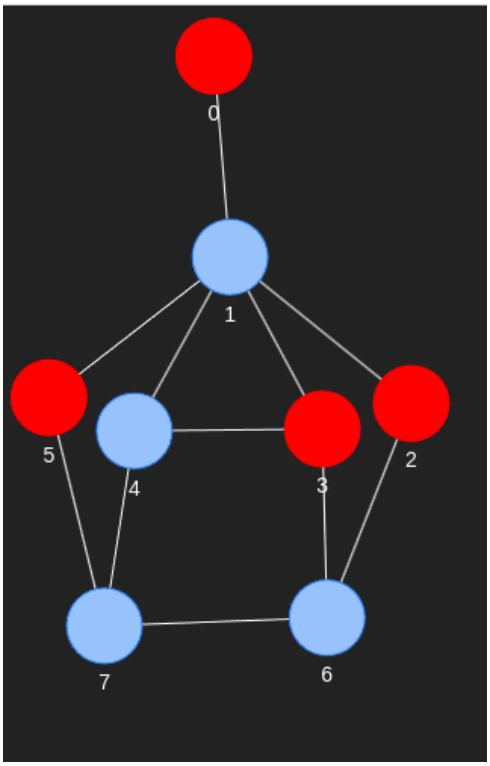
```
Component(x=3, i=1, vertices={'2'}, alpha=1)
Component(x=3, i=2, vertices={'0'}, alpha=1)
Computed Component,('7', 0) Alpha: Component(x=7, i=0, vertices={'1', '2', '0', '3'}, alpha=3)
Component(x=2, i=0, vertices={'5', '4', '7', '3'}, alpha=None)
Interval(x=2, y=5, vertices={'4', '3'}, alpha=None)
Interval ('2', '4') not found
Interval ('4', '5') not found
Interval ('2', '3') not found
Interval ('3', '5') not found
interval alpha: 1
Computed Interval(('2', '5')) alpha:Interval(x=2, y=5, vertices={'4', '3'}, alpha=1)
Interval ('2', '4') not found
Component(x=4, i=0, vertices={'5'}, alpha=1)
Interval(x=2, y=7, vertices={'3'}, alpha=None)
Interval ('2', '3') not found
Interval ('3', '7') not found
interval alpha: 1
Computed Interval(('2', '7')) alpha:Interval(x=2, y=7, vertices={'3'}, alpha=1)
Interval ('2', '3') not found
Component(x=3, i=0, vertices={'5', '7'}, alpha=1)
Computed Component,('2', 0) Alpha: Component(x=2, i=0, vertices={'5', '4', '7', '3'}, alpha=2)
Component(x=6, i=0, vertices={'5', '4', '0', '1'}, alpha=None)
Interval(x=6, y=5, vertices={'4'}, alpha=None)
Interval ('6', '4') not found
Interval ('4', '5') not found
interval alpha: 1
Computed Interval(('6', '5')) alpha:Interval(x=6, y=5, vertices={'4'}, alpha=1)
Component(x=5, i=1, vertices={'0'}, alpha=1)
Interval ('6', '4') not found
Component(x=4, i=0, vertices={'5'}, alpha=1)
Component(x=4, i=2, vertices={'0'}, alpha=1)
Interval(x=6, y=0, vertices={'5', '4'}, alpha=None)
Interval(x=6, y=5, vertices={'4'}, alpha=1)
Interval (('6', '5')) is already computed with alpha: 1
Interval ('5', '0') not found
Interval ('6', '4') not found
Interval ('4', '0') not found
Component(x=4, i=0, vertices={'5'}, alpha=1)
interval alpha: 2
Computed Interval(('6', '0')) alpha:Interval(x=6, y=0, vertices={'5', '4'}, alpha=2)
Interval ('6', '1') not found
```

```
Computed Component,('6', 0) Alpha: Component(x=6, i=0, vertices={'5', '4', '0', '1'}, alpha=3)
Component(x=0, i=0, vertices={'5', '7', '2', '3', '6', '4'}, alpha=None)
Interval ('0', '5') not found
Component(x=5, i=0, vertices={'6', '2', '4', '3'}, alpha=2)
Interval(x=0, y=7, vertices={'2', '3'}, alpha=None)
Interval ('0', '2') not found
Interval(x=2, y=7, vertices={'3'}, alpha=1)
Interval (('2', '7')) is already computed with alpha: 1
Interval ('0', '3') not found
Interval ('3', '7') not found
Component(x=3, i=1, vertices={'2'}, alpha=1)
interval alpha: 2
Computed Interval(('0', '7')) alpha:Interval(x=0, y=7, vertices={'2', '3'}, alpha=2)
Interval ('0', '2') not found
Component(x=2, i=0, vertices={'5', '4', '7', '3'}, alpha=2)
Interval ('0', '3') not found
Component(x=3, i=0, vertices={'5', '7'}, alpha=1)
Component(x=3, i=1, vertices={'2'}, alpha=1)
Interval(x=0, y=6, vertices={'5', '4'}, alpha=None)
Interval ('0', '5') not found
Interval(x=5, y=6, vertices={'4'}, alpha=1)
Interval (('5', '6')) is already computed with alpha: 1
Interval ('0', '4') not found
Interval ('4', '6') not found
Component(x=4, i=0, vertices={'5'}, alpha=1)
interval alpha: 2
Computed Interval(('0', '6')) alpha:Interval(x=0, y=6, vertices={'5', '4'}, alpha=2)
Interval ('0', '4') not found
Component(x=4, i=0, vertices={'5'}, alpha=1)
Component(x=4, i=1, vertices={'6', '2'}, alpha=1)
Computed Component,('0', 0) Alpha: Component(x=0, i=0, vertices={'5', '7', '2', '3', '6', '4'},
     alpha=3)




Component(x=5, i=0, vertices={'6', '2', '4', '3'}, alpha=2)
Component(x=5, i=1, vertices={'0'}, alpha=1)
Component(x=7, i=0, vertices={'1', '2', '0', '3'}, alpha=3)
Component(x=2, i=0, vertices={'5', '4', '7', '3'}, alpha=2)
Component(x=2, i=1, vertices={'0'}, alpha=1)
```

```
Component(x=3, i=0, vertices={'5', '7'}, alpha=1)
Component(x=3, i=1, vertices={'2'}, alpha=1)
Component(x=3, i=2, vertices={'0'}, alpha=1)
Component(x=6, i=0, vertices={'5', '4', '0', '1'}, alpha=3)
Component(x=4, i=0, vertices={'5'}, alpha=1)
Component(x=4, i=1, vertices={'6', '2'}, alpha=1)
Component(x=4, i=2, vertices={'0'}, alpha=1)
Component(x=0, i=0, vertices={'5', '7', '2', '3', '6', '4'}, alpha=3)
Component(x=1, i=0, vertices={'6', '7'}, alpha=1)
max_alpha_component: 5
max independent set: {'5', '2', '0', '3'}
Graph intependent set number: 4
Brute force: 4
```



Σχήμα 3.2: Το γράφημα εξόδου του Αλγορίθμου σχεδιασμένο από την βιβλιοθήκη PyVis

3.5 Υπολογισμός Μέγιστου Ανεξάρτητου Συνόλου

Σε αυτό το κεφάλαιο, παρουσιάζουμε συνοπτικά τις κλάσεις που αποτελούν το αλγοριθμικό τμήμα για τον υπολογισμό του μέγιστου ανεξάρτητου συνόλου. Προ-

χωρούμε στην παρουσίαση αυτών των κλάσεων με τη σειρά που χρησιμοποιούνται στο πλαίσιο του συνολικού αλγοριθμικού σχεδιασμού. Επισημαίνουμε ότι αγνοούμε τις κλάσεις που υλοποιούν αλγορίθμους από άλλες εργασίες, όπως ο υπολογισμός ενός ελάχιστου κυρίαρχου μονοπατιού [3].

3.5.1 Η Κλάση PolynomialTimeAlgorithm

polynomial_time_algorithm.py

```
from graph import Graph
from itertools import combinations



class PolynomialTimeAlgorithm:
    def __init__(self,graph : Graph, weight : int):
        self.graph = graph
        self.weight = weight



    def mcdsw(self):
        """
        Returns a minimum cardinality dominating set of the graph.
        """
        D = self.graph.vertices
        for vertex in self.graph.vertices:
            H = self.graph.bfs_levels(vertex)
            l = len(H) - 1
            i = 1

            print("\nBFS Levels of:",vertex,":\n" ,H)

            #Initialize the queue A1 to contain an ordred triple (S, S, val_S)
            #for all nonempty subets S of N[vertex] satisfying val_S:= |S| ≤w
            A1 = self.initialize_queue(vertex)
            print("Init queue of:",vertex,":\n" ,A1)
            A = { i : A1}

            while A[i] and i < l:
```

```
    i = i + 1
    A[i] = []

    for triple in A[i-1]:
        print(f"Triple in the queue A[{i-1}]:", triple)
        S = triple[0]
        S_accent = triple[1]
        val_S_accent = triple[2]
        #For all U H[i] such that |S U| ≤w
        for r in range(1, len(H[i]) + 1):
            for U in combinations(H[i], r):
                U = set(U)
                if len(S | U) <= self.weight:
                    closed_neighborhood_of_S_and_U = self.graph.
                        closed_neighborhood_of_set(S | U)
                    if closed_neighborhood_of_S_and_U.issuperset(H[i-1]):
                        R = ( S | U ) - H[i-2]
                        R_accent = S_accent | U
                        val_R_accent = val_S_accent + len(U)
                        print("R:", (R, R_accent, val_R_accent))
                        #IF there is no triple in Ai with first entry R
                        #THEN inser (R; R_accent ; val_R_accent) into Ai ;
                        if not any(triple[0] == R for triple in A[i]):
                            print(f"R is not in A[{i}] so we insert it into A[{i}")
                            A[i].append((R, R_accent, val_R_accent))
                        #IF there is a triple (P; P_accent ; val_P_accent) in Ai such
                            that P = R AND val_R_accent < val_P_accent
                        #THEN replace (P; P_accent ; val_P_accent) by (R; R_accent ;
                            val_R_accent) in Ai ;
                        for index, triple in enumerate(A[i]):
                            if triple[0] == R and val_R_accent < triple[2]:
                                print(f"R is in A[{i}] and val_R_accent < val_P_accent
                                    so we replace it in A[{i}]")
                                A[i][index] = (R, R_accent, val_R_accent)


print("\nA[l] after while loop:", A[l])
# Find the triple with minimum val(S') in A[l] that satisfies H[l] N[S]
optimal_triple = min((triple for triple in A[l] if set(H[l]).issubset(self.graph.
    closed_neighborhood_of_set(triple[0]))),
                key=lambda x: x[2], default=None)
```

```
            print("\noptimal_triple:", optimal_triple)

            if optimal_triple and optimal_triple[2] < len(D):
                D = optimal_triple[1] # Update D if a better solution is found
                print("########\nD is updated to:", D)
                print("########")

        print("THE DOMINATION SET IS:", D)
        self.graph.domination_set = D
        self.graph.show()

        return

    def initialize_queue(self, vertex):
        queue = []
        for subset in self.generate_subsets(vertex):
            queue.append((subset, subset, len(subset)))
        return queue

    def generate_subsets(self, vertex):
        closed_nbhd = self.graph.closed_neighborhood(vertex)
        subsets = []
        for r in range(1, self.weight + 1):
            for subset in combinations(closed_nbhd, r):
                subsets.append(set(subset))
        return subsets



    def run(self):
        return self.mcdsw()
```

Εξηγούμε αναλυτικά τις συναρτήσεις της κλάσης:

1. Κατασκευαστής (__init__): Δέχεται ένα αντικείμενο Graph και ένα ακέραιο weight, που υποδηλώνει το βάρος w αλγορίθμου mcdsw και αρχικοποιεί τα αντίστοιχα χαρακτηριστικά της κλάσης.

2. Η συνάρτηση `mcdsw`: Υλοποιεί τον αλγόριθμο $mcds_w$ 2.2.

3. Η συνάρτηση `initialize_queue`: Βοηθητική κλάση που αρχικοποιεί την ουρά

A_1.

4. Η συνάρτηση `generate_subsets`: Η συνάρτηση δημιουργήσει όλα τα υποσύνολα της κλειστής γειτονιάς ενός κόμβου σε μήκη από 1 έως το βάρος (weight) του έχουμε επιλέξει.

3.5.2 Η Κλάση Graph

graph.py

```
from itertools import combinations
from pyvis.network import Network


class Graph:
    show_count = 0 # Class-level variable to keep track of show calls

    def __init__(self, num_of_vertices, num_of_edges, edges, vertices=None):
        self.num_of_vertices = num_of_vertices
        self.num_of_edges = num_of_edges
        self.vertices = set(str(i) for i in range(num_of_vertices)) if vertices is None else
            vertices
        self.edges = set((str(u), str(v)) for u, v in edges)
        self.adjacency_list = {str(vertex): set() for vertex in self.vertices}
        for edge in self.edges:
            u, v = edge
            self.adjacency_list[u].add(v)
            self.adjacency_list[v].add(u)

        domination_set = set()



    def bfs_levels(self, source):
        source = str(source)

        visited = {vertex: False for vertex in self.vertices}
        level = {vertex: None for vertex in self.vertices}
        queue = []

        visited[source] = True
```

```
    level[source] = 0
    queue.append(source)

    levels = {}

    while queue:
        vertex = queue.pop(0)
        current_level = level[vertex]

        if current_level not in levels:
            levels[current_level] = set()
        levels[current_level].add(vertex)

        for neighbor in self.adjacency_list[vertex]:
            if not visited[neighbor]:
                queue.append(neighbor)
                visited[neighbor] = True
                level[neighbor] = current_level + 1

    return levels

def closed_neighborhood(self, vertex):
    return {vertex}.union(self.adjacency_list[vertex])

def closed_neighborhood_of_set(self, vertex_set):
    closed_neighborhood = set()
    for vertex in vertex_set:
        vertex_str = str(vertex)
        closed_neighborhood = closed_neighborhood.union(self.closed_neighborhood(vertex_str))
    return closed_neighborhood



def show(self, graph_name='graph'):
    Graph.show_count += 1
    print("Showing graph:", Graph.show_count)
    net = Network(height="500px", width="100%", bgcolor="#222222", font_color="white")

    for vertex in self.vertices:
        if vertex in self.domination_set:
            net.add_node(vertex, color="red")
        else:
```

```
            net.add_node(vertex)


    for edge in self.edges:
        u, v = edge
        net.add_edge(u, v, color="white")

    file_name = f"../output-graphs/{graph_name}-{Graph.show_count}.html"
    net.show(file_name)
```

Εξηγούμε αναλυτικά τις συναρτήσεις της κλάσης:

1. Μέθοδος `bfs_levels`: Υλοποιεί τον αλγόριθμο BFS (Breadth-First Search) για τον υπολογισμό των επιπέδων του γράφου από μια δεδομένη κορυφή.

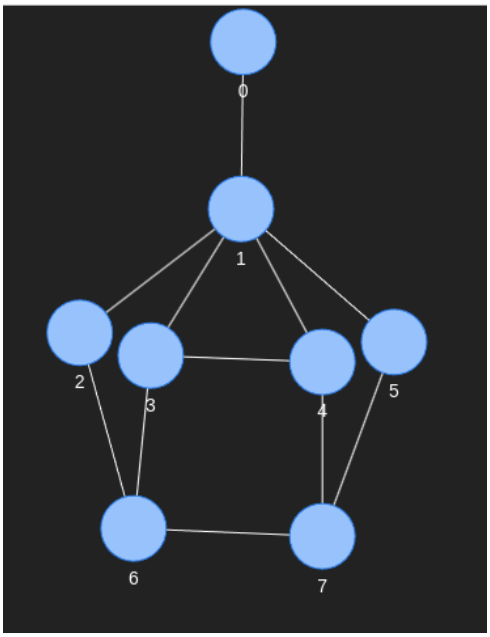
2. Μέθοδος `closed_neighborhood`: Επιστρέφει την κλειστή γειτονιά μιας κορυφής, δηλαδή την κορυφή και όλες τις γειτονικές της κορυφές.

3. Μέθοδος `closed_neighborhood_of_set`: Υπολογίζει την κλειστή γειτονιά ενός συνόλου κορυφών.

4. Μέθοδος `show`: Χρησιμοποιεί το πακέτο `pyvis` για να εμφανίσει το γράφημα σε μορφή διαδραστικού δικτύου.

3.5.3 Παραδείγματα Εκτέλεσης του Προγράμματος

Ως παράδειγμα δείχνουμε πως εκτελούμε τον αλγόριθμο υπολογισμού του ελάχιστου κυρίαρχου συνόλου στο ίδιο γράφημα με την Ενότητα 2.2.



Σχήμα 3.3: Το γράφημα εισόδου του Αλγορίθμου σχεδιασμένο από την βιβλιοθήκη PyVis

Η εκτέλεση του Αλγορίθμου

```
BFS Levels of: 1 :
 {0: {'1'}, 1: {'4', '2', '5', '3', '0'}, 2: {'6', '7'}}
Init queue of: 1 :
 [({'4'}, {'4'}, 1), ({'3'}, {'3'}, 1), ({'2'}, {'2'}, 1), ({'0'}, {'0'}, 1), ({'5'}, {'5'}, 1)
     , ({'1'}, {'1'}, 1), ({'4', '3'}, {'4', '3'}, 2), ({'4', '2'}, {'4', '2'}, 2), ({'4',
     '0'}, {'4', '0'}, 2), ({'4', '5'}, {'4', '5'}, 2), ({'4', '1'}, {'4', '1'}, 2), ({'3',
     '2'}, {'3', '2'}, 2), ({'3', '0'}, {'3', '0'}, 2), ({'3', '5'}, {'3', '5'}, 2), ({'3',
     '1'}, {'3', '1'}, 2), ({'2', '0'}, {'2', '0'}, 2), ({'5', '2'}, {'5', '2'}, 2), ({'2',
     '1'}, {'2', '1'}, 2), ({'5', '0'}, {'5', '0'}, 2), ({'1', '0'}, {'1', '0'}, 2), ({'5',
     '1'}, {'5', '1'}, 2), ({'4', '3', '2'}, {'4', '3', '2'}, 3), ({'4', '3', '0'}, {'4', '3',
     '0'}, 3), ({'4', '3', '5'}, {'4', '3', '5'}, 3), ({'4', '3', '1'}, {'4', '3', '1'}, 3),
     ({'4', '2', '0'}, {'4', '2', '0'}, 3), ({'4', '5', '2'}, {'4', '5', '2'}, 3), ({'4', '2',
     '1'}, {'4', '2', '1'}, 3), ({'4', '5', '0'}, {'4', '5', '0'}, 3), ({'4', '1', '0'}, {'4',
     '1', '0'}, 3), ({'4', '5', '1'}, {'4', '5', '1'}, 3), ({'3', '2', '0'}, {'3', '2', '0'},
     3), ({'5', '3', '2'}, {'5', '3', '2'}, 3), ({'3', '2', '1'}, {'3', '2', '1'}, 3), ({'3',
     '5', '0'}, {'3', '5', '0'}, 3), ({'3', '1', '0'}, {'3', '1', '0'}, 3), ({'3', '5', '1'},
     {'3', '5', '1'}, 3), ({'5', '2', '0'}, {'5', '2', '0'}, 3), ({'2', '1', '0'}, {'2', '1',
     '0'}, 3), ({'5', '2', '1'}, {'5', '2', '1'}, 3), ({'5', '1', '0'}, {'5', '1', '0'}, 3),
     ({'4', '3', '2', '0'}, {'4', '3', '2', '0'}, 4), ({'4', '3', '2', '5'}, {'4', '3', '2',
     '5'}, 4), ({'4', '3', '2', '1'}, {'4', '3', '2', '1'}, 4), ({'4', '3', '5', '0'}, {'4',
     '3', '5', '0'}, 4), ({'4', '3', '1', '0'}, {'4', '3', '1', '0'}, 4), ({'4', '3', '5',
```

```
     '1'}, {'4', '3', '5', '1'}, 4), ({'4', '5', '2', '0'}, {'4', '5', '2', '0'}, 4), ({'4',
     '2', '1', '0'}, {'4', '2', '1', '0'}, 4), ({'4', '5', '2', '1'}, {'4', '5', '2', '1'}, 4),
      ({'4', '5', '1', '0'}, {'4', '5', '1', '0'}, 4), ({'5', '3', '2', '0'}, {'5', '3', '2',
     '0'}, 4), ({'3', '2', '1', '0'}, {'3', '2', '1', '0'}, 4), ({'5', '3', '2', '1'}, {'5',
     '3', '2', '1'}, 4), ({'3', '5', '1', '0'}, {'3', '5', '1', '0'}, 4), ({'5', '2', '1',
     '0'}, {'5', '2', '1', '0'}, 4), ({'4', '2', '5', '3', '0'}, {'4', '2', '5', '3', '0'}, 5),
      ({'4', '2', '3', '0', '1'}, {'4', '2', '3', '0', '1'}, 5), ({'4', '2', '5', '3', '1'},
     {'4', '2', '5', '3', '1'}, 5), ({'4', '5', '3', '0', '1'}, {'4', '5', '3', '0', '1'}, 5),
     ({'4', '2', '5', '0', '1'}, {'4', '2', '5', '0', '1'}, 5), ({'2', '5', '3', '0', '1'},
     {'2', '5', '3', '0', '1'}, 5)]
Triple in the queue A[1]: ({'4'}, {'4'}, 1)
Triple in the queue A[1]: ({'3'}, {'3'}, 1)
Triple in the queue A[1]: ({'2'}, {'2'}, 1)
Triple in the queue A[1]: ({'0'}, {'0'}, 1)
R: ({'6', '7', '0'}, {'7', '6', '0'}, 3)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5'}, {'5'}, 1)
Triple in the queue A[1]: ({'1'}, {'1'}, 1)
R: ({'6'}, {'6', '1'}, 2)
R is not in A[2] so we insert it into A[2]
R: ({'7'}, {'7', '1'}, 2)
R is not in A[2] so we insert it into A[2]
R: ({'6', '7'}, {'7', '6', '1'}, 3)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3'}, {'4', '3'}, 2)
Triple in the queue A[1]: ({'4', '2'}, {'4', '2'}, 2)
Triple in the queue A[1]: ({'4', '0'}, {'4', '0'}, 2)
R: ({'4', '6', '7', '0'}, {'4', '7', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '5'}, {'4', '5'}, 2)
Triple in the queue A[1]: ({'4', '1'}, {'4', '1'}, 2)
R: ({'4', '6'}, {'4', '6', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'4', '7'}, {'4', '7', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'4', '6', '7'}, {'4', '7', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'3', '2'}, {'3', '2'}, 2)
Triple in the queue A[1]: ({'3', '0'}, {'3', '0'}, 2)
R: ({'6', '3', '7', '0'}, {'7', '3', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
```

```
Triple in the queue A[1]: ({'3', '5'}, {'3', '5'}, 2)
Triple in the queue A[1]: ({'3', '1'}, {'3', '1'}, 2)
R: ({'3', '6'}, {'3', '6', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'3', '7'}, {'3', '7', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'6', '3', '7'}, {'7', '3', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'2', '0'}, {'2', '0'}, 2)
R: ({'6', '2', '7', '0'}, {'7', '2', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '2'}, {'5', '2'}, 2)
Triple in the queue A[1]: ({'2', '1'}, {'2', '1'}, 2)
R: ({'2', '6'}, {'2', '6', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'2', '7'}, {'2', '7', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'6', '2', '7'}, {'7', '2', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '0'}, {'5', '0'}, 2)
R: ({'6', '5', '7', '0'}, {'7', '5', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'1', '0'}, {'1', '0'}, 2)
R: ({'6', '0'}, {'1', '6', '0'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'7', '0'}, {'1', '7', '0'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'6', '7', '0'}, {'7', '1', '6', '0'}, 4)
Triple in the queue A[1]: ({'5', '1'}, {'5', '1'}, 2)
R: ({'5', '6'}, {'5', '6', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'5', '7'}, {'5', '7', '1'}, 3)
R is not in A[2] so we insert it into A[2]
R: ({'6', '5', '7'}, {'7', '5', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3', '2'}, {'4', '3', '2'}, 3)
Triple in the queue A[1]: ({'4', '3', '0'}, {'4', '3', '0'}, 3)
R: ({'4', '6', '3', '7', '0'}, {'4', '3', '6', '0', '7'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3', '5'}, {'4', '3', '5'}, 3)
Triple in the queue A[1]: ({'4', '3', '1'}, {'4', '3', '1'}, 3)
```

```
R: ({'4', '3', '6'}, {'4', '3', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '3', '7'}, {'4', '3', '7', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '3', '6', '7'}, {'4', '3', '6', '7', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '2', '0'}, {'4', '2', '0'}, 3)
R: ({'4', '2', '7', '0'}, {'4', '2', '7', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '2', '6', '7', '0'}, {'4', '2', '6', '0', '7'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '5', '2'}, {'4', '5', '2'}, 3)
Triple in the queue A[1]: ({'4', '2', '1'}, {'4', '2', '1'}, 3)
R: ({'4', '2', '6'}, {'4', '2', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '2', '7'}, {'4', '2', '7', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '2', '6', '7'}, {'4', '2', '6', '7', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '5', '0'}, {'4', '5', '0'}, 3)
R: ({'4', '5', '6', '0'}, {'4', '5', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '6', '5', '7', '0'}, {'4', '6', '0', '7', '5'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '1', '0'}, {'4', '1', '0'}, 3)
R: ({'4', '6', '0'}, {'4', '1', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '7', '0'}, {'4', '1', '7', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '7', '6', '0'}, {'4', '6', '0', '7', '1'}, 5)
Triple in the queue A[1]: ({'4', '5', '1'}, {'4', '5', '1'}, 3)
R: ({'4', '5', '6'}, {'4', '5', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '5', '7'}, {'4', '5', '7', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'4', '5', '6', '7'}, {'4', '6', '7', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'3', '2', '0'}, {'3', '2', '0'}, 3)
R: ({'3', '2', '7', '0'}, {'3', '2', '7', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'2', '6', '3', '7', '0'}, {'3', '2', '6', '0', '7'}, 5)
```

```
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '3', '2'}, {'5', '3', '2'}, 3)
Triple in the queue A[1]: ({'3', '2', '1'}, {'3', '2', '1'}, 3)
R: ({'3', '2', '6'}, {'3', '2', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'3', '2', '7'}, {'3', '2', '7', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'3', '2', '6', '7'}, {'3', '2', '6', '7', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'3', '5', '0'}, {'3', '5', '0'}, 3)
R: ({'3', '5', '6', '0'}, {'3', '5', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'6', '5', '3', '7', '0'}, {'3', '6', '0', '7', '5'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'3', '1', '0'}, {'3', '1', '0'}, 3)
R: ({'3', '6', '0'}, {'3', '1', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'3', '7', '0'}, {'3', '1', '7', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'7', '3', '6', '0'}, {'3', '6', '0', '7', '1'}, 5)
Triple in the queue A[1]: ({'3', '5', '1'}, {'3', '5', '1'}, 3)
R: ({'3', '5', '6'}, {'3', '5', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'3', '5', '7'}, {'3', '5', '7', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'3', '5', '6', '7'}, {'3', '6', '7', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '2', '0'}, {'5', '2', '0'}, 3)
R: ({'2', '6', '5', '7', '0'}, {'2', '6', '0', '7', '5'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'2', '1', '0'}, {'2', '1', '0'}, 3)
R: ({'2', '6', '0'}, {'2', '1', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'2', '7', '0'}, {'2', '1', '7', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'7', '2', '6', '0'}, {'2', '6', '0', '7', '1'}, 5)
Triple in the queue A[1]: ({'5', '2', '1'}, {'5', '2', '1'}, 3)
R: ({'5', '2', '6'}, {'5', '2', '6', '1'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'5', '2', '7'}, {'5', '2', '7', '1'}, 4)
R is not in A[2] so we insert it into A[2]
```

```
R: ({'5', '2', '6', '7'}, {'2', '6', '7', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '1', '0'}, {'5', '1', '0'}, 3)
R: ({'5', '6', '0'}, {'5', '1', '6', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'5', '7', '0'}, {'5', '1', '7', '0'}, 4)
R is not in A[2] so we insert it into A[2]
R: ({'7', '5', '6', '0'}, {'6', '0', '7', '5', '1'}, 5)
Triple in the queue A[1]: ({'4', '3', '2', '0'}, {'4', '3', '2', '0'}, 4)
R: ({'4', '2', '3', '7', '0'}, {'4', '3', '2', '7', '0'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3', '2', '5'}, {'4', '3', '2', '5'}, 4)
Triple in the queue A[1]: ({'4', '3', '2', '1'}, {'4', '3', '2', '1'}, 4)
R: ({'4', '3', '2', '6'}, {'4', '3', '2', '6', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'4', '3', '2', '7'}, {'4', '3', '2', '7', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3', '5', '0'}, {'4', '3', '5', '0'}, 4)
R: ({'4', '6', '5', '3', '0'}, {'4', '3', '6', '0', '5'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3', '1', '0'}, {'4', '3', '1', '0'}, 4)
R: ({'4', '3', '6', '0'}, {'4', '3', '6', '0', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'4', '3', '7', '0'}, {'4', '3', '7', '0', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '3', '5', '1'}, {'4', '3', '5', '1'}, 4)
R: ({'4', '3', '5', '6'}, {'4', '3', '6', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'4', '3', '5', '7'}, {'4', '3', '7', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '5', '2', '0'}, {'4', '5', '2', '0'}, 4)
R: ({'4', '2', '6', '5', '0'}, {'4', '2', '6', '0', '5'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'4', '2', '5', '7', '0'}, {'4', '2', '7', '0', '5'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '2', '1', '0'}, {'4', '2', '1', '0'}, 4)
R: ({'4', '2', '6', '0'}, {'4', '2', '6', '0', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'4', '2', '7', '0'}, {'4', '2', '7', '0', '1'}, 5)
Triple in the queue A[1]: ({'4', '5', '2', '1'}, {'4', '5', '2', '1'}, 4)
R: ({'4', '5', '2', '6'}, {'4', '2', '6', '5', '1'}, 5)
```

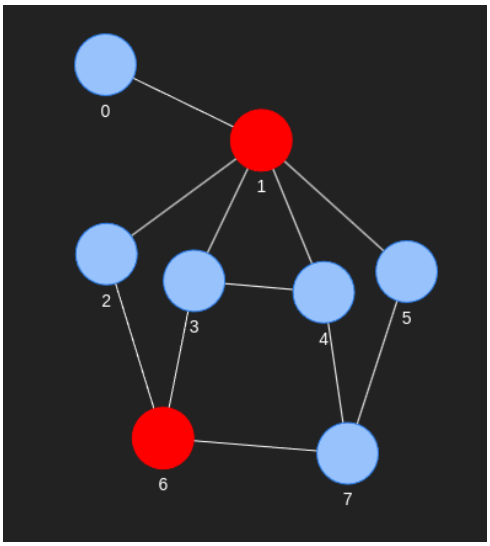
```
R is not in A[2] so we insert it into A[2]
R: ({'4', '5', '2', '7'}, {'4', '2', '7', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '5', '1', '0'}, {'4', '5', '1', '0'}, 4)
R: ({'4', '5', '6', '0'}, {'4', '6', '0', '5', '1'}, 5)
R: ({'4', '5', '7', '0'}, {'4', '7', '0', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '3', '2', '0'}, {'5', '3', '2', '0'}, 4)
R: ({'2', '6', '5', '3', '0'}, {'3', '2', '6', '0', '5'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'2', '5', '3', '7', '0'}, {'3', '2', '7', '0', '5'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'3', '2', '1', '0'}, {'3', '2', '1', '0'}, 4)
R: ({'3', '2', '6', '0'}, {'3', '2', '6', '0', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'3', '2', '7', '0'}, {'3', '2', '7', '0', '1'}, 5)
Triple in the queue A[1]: ({'5', '3', '2', '1'}, {'5', '3', '2', '1'}, 4)
R: ({'5', '3', '2', '6'}, {'3', '2', '6', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'5', '3', '2', '7'}, {'3', '2', '7', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'3', '5', '1', '0'}, {'3', '5', '1', '0'}, 4)
R: ({'3', '5', '6', '0'}, {'3', '6', '0', '5', '1'}, 5)
R: ({'3', '5', '7', '0'}, {'3', '7', '0', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'5', '2', '1', '0'}, {'5', '2', '1', '0'}, 4)
R: ({'5', '2', '6', '0'}, {'2', '6', '0', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
R: ({'5', '2', '7', '0'}, {'2', '7', '0', '5', '1'}, 5)
R is not in A[2] so we insert it into A[2]
Triple in the queue A[1]: ({'4', '2', '5', '3', '0'}, {'4', '2', '5', '3', '0'}, 5)
Triple in the queue A[1]: ({'4', '2', '3', '0', '1'}, {'4', '2', '3', '0', '1'}, 5)
Triple in the queue A[1]: ({'4', '2', '5', '3', '1'}, {'4', '2', '5', '3', '1'}, 5)
Triple in the queue A[1]: ({'4', '5', '3', '0', '1'}, {'4', '5', '3', '0', '1'}, 5)
Triple in the queue A[1]: ({'4', '2', '5', '0', '1'}, {'4', '2', '5', '0', '1'}, 5)
Triple in the queue A[1]: ({'2', '5', '3', '0', '1'}, {'2', '5', '3', '0', '1'}, 5)

A[l] after while loop: [({'6', '7', '0'}, {'7', '6', '0'}, 3), ({'6'}, {'6', '1'}, 2), ({'7'},
    {'7', '1'}, 2), ({'6', '7'}, {'7', '6', '1'}, 3), ({'4', '6', '7', '0'}, {'4', '7', '6',
    '0'}, 4), ({'4', '6'}, {'4', '6', '1'}, 3), ({'4', '7'}, {'4', '7', '1'}, 3), ({'4', '6',
    '7'}, {'4', '7', '6', '1'}, 4), ({'6', '3', '7', '0'}, {'7', '3', '6', '0'}, 4), ({'3',
```

```
    '6'}, {'3', '6', '1'}, 3), ({'3', '7'}, {'3', '7', '1'}, 3), ({'6', '3', '7'}, {'7', '3',
    '6', '1'}, 4), ({'6', '2', '7', '0'}, {'7', '2', '6', '0'}, 4), ({'2', '6'}, {'2', '6',
    '1'}, 3), ({'2', '7'}, {'2', '7', '1'}, 3), ({'6', '2', '7'}, {'7', '2', '6', '1'}, 4),
    ({'6', '5', '7', '0'}, {'7', '5', '6', '0'}, 4), ({'6', '0'}, {'1', '6', '0'}, 3), ({'7',
    '0'}, {'1', '7', '0'}, 3), ({'5', '6'}, {'5', '6', '1'}, 3), ({'5', '7'}, {'5', '7', '1'},
    3), ({'6', '5', '7'}, {'7', '5', '6', '1'}, 4), ({'4', '6', '3', '7', '0'}, {'4', '3', '6',
     '0', '7'}, 5), ({'4', '3', '6'}, {'4', '3', '6', '1'}, 4), ({'4', '3', '7'}, {'4', '3',
    '7', '1'}, 4), ({'4', '3', '6', '7'}, {'4', '3', '6', '7', '1'}, 5), ({'4', '2', '7', '0'},
     {'4', '2', '7', '0'}, 4), ({'4', '2', '6', '7', '0'}, {'4', '2', '6', '0', '7'}, 5),
    ({'4', '2', '6'}, {'4', '2', '6', '1'}, 4), ({'4', '2', '7'}, {'4', '2', '7', '1'}, 4),
    ({'4', '2', '6', '7'}, {'4', '2', '6', '7', '1'}, 5), ({'4', '5', '6', '0'}, {'4', '5',
    '6', '0'}, 4), ({'4', '6', '5', '7', '0'}, {'4', '6', '0', '7', '5'}, 5), ({'4', '6', '0'},
     {'4', '1', '6', '0'}, 4), ({'4', '7', '0'}, {'4', '1', '7', '0'}, 4), ({'4', '5', '6'},
    {'4', '5', '6', '1'}, 4), ({'4', '5', '7'}, {'4', '5', '7', '1'}, 4), ({'4', '5', '6',
    '7'}, {'4', '6', '7', '5', '1'}, 5), ({'3', '2', '7', '0'}, {'3', '2', '7', '0'}, 4),
    ({'2', '6', '3', '7', '0'}, {'3', '2', '6', '0', '7'}, 5), ({'3', '2', '6'}, {'3', '2',
    '6', '1'}, 4), ({'3', '2', '7'}, {'3', '2', '7', '1'}, 4), ({'3', '2', '6', '7'}, {'3',
    '2', '6', '7', '1'}, 5), ({'3', '5', '6', '0'}, {'3', '5', '6', '0'}, 4), ({'6', '5', '3',
    '7', '0'}, {'3', '6', '0', '7', '5'}, 5), ({'3', '6', '0'}, {'3', '1', '6', '0'}, 4),
    ({'3', '7', '0'}, {'3', '1', '7', '0'}, 4), ({'3', '5', '6'}, {'3', '5', '6', '1'}, 4),
    ({'3', '5', '7'}, {'3', '5', '7', '1'}, 4), ({'3', '5', '6', '7'}, {'3', '6', '7', '5',
    '1'}, 5), ({'2', '6', '5', '7', '0'}, {'2', '6', '0', '7', '5'}, 5), ({'2', '6', '0'},
    {'2', '1', '6', '0'}, 4), ({'2', '7', '0'}, {'2', '1', '7', '0'}, 4), ({'5', '2', '6'},
    {'5', '2', '6', '1'}, 4), ({'5', '2', '7'}, {'5', '2', '7', '1'}, 4), ({'5', '2', '6',
    '7'}, {'2', '6', '7', '5', '1'}, 5), ({'5', '6', '0'}, {'5', '1', '6', '0'}, 4), ({'5',
    '7', '0'}, {'5', '1', '7', '0'}, 4), ({'4', '2', '3', '7', '0'}, {'4', '3', '2', '7', '0'},
     5), ({'4', '3', '2', '6'}, {'4', '3', '2', '6', '1'}, 5), ({'4', '3', '2', '7'}, {'4',
    '3', '2', '7', '1'}, 5), ({'4', '6', '5', '3', '0'}, {'4', '3', '6', '0', '5'}, 5), ({'4',
    '3', '6', '0'}, {'4', '3', '6', '0', '1'}, 5), ({'4', '3', '7', '0'}, {'4', '3', '7', '0',
    '1'}, 5), ({'4', '3', '5', '6'}, {'4', '3', '6', '5', '1'}, 5), ({'4', '3', '5', '7'},
    {'4', '3', '7', '5', '1'}, 5), ({'4', '2', '6', '5', '0'}, {'4', '2', '6', '0', '5'}, 5),
    ({'4', '2', '5', '7', '0'}, {'4', '2', '7', '0', '5'}, 5), ({'4', '2', '6', '0'}, {'4',
    '2', '6', '0', '1'}, 5), ({'4', '5', '2', '6'}, {'4', '2', '6', '5', '1'}, 5), ({'4', '5',
    '2', '7'}, {'4', '2', '7', '5', '1'}, 5), ({'4', '5', '7', '0'}, {'4', '7', '0', '5', '1'},
     5), ({'2', '6', '5', '3', '0'}, {'3', '2', '6', '0', '5'}, 5), ({'2', '5', '3', '7', '0'},
     {'3', '2', '7', '0', '5'}, 5), ({'3', '2', '6', '0'}, {'3', '2', '6', '0', '1'}, 5),
    ({'5', '3', '2', '6'}, {'3', '2', '6', '5', '1'}, 5), ({'5', '3', '2', '7'}, {'3', '2',
    '7', '5', '1'}, 5), ({'3', '5', '7', '0'}, {'3', '7', '0', '5', '1'}, 5), ({'5', '2', '6',
    '0'}, {'2', '6', '0', '5', '1'}, 5), ({'5', '2', '7', '0'}, {'2', '7', '0', '5', '1'}, 5)]

optimal_triple: ({'6'}, {'6', '1'}, 2)
```

```
THE DOMINATION SET IS: {'6', '1'}
Showing graph: 1
brute_force_domintaion_set {'7', '1'}
brute_force_algorithm, minimum_domination_set_size: 2
Showing graph: 2
```



Σχήμα 3.4: Το γράφημα εξόδου του Αλγορίθμου σχεδιασμένο από την βιβλιοθήκη PyVis

3.6 Το Πρόβλημα του 3-Χρωματισμού

Στο παρόν κεφάλαιο, παρουσιάζουμε ενδεικτικές κλάσεις που αποτελούν το αλγοριθμικό τμήμα για την επίλυση του προβλήματος 3-χρωματισμού. Προχωρούμε στην παρουσίαση αυτών των κλάσεων με τη σειρά που καλούνται στο πλαίσιο του συνολικού αλγοριθμικού σχεδιασμού. Σημειώνουμε πως παραλείπουμε τις κλάσεις που υλοποιούν αλγορίθμους από άλλες εργασίες, όπως ο Αλγόριθμος "Biconnectivity" του Robert Tarjan [28].

3.6.1 Η Κλάση PolynomialTimeAlgorithm

polynomial_time_algorithm.py

```
from utilities import *
from graph import Graph
from tarjans_biconnectivity import TarjansBiconnectivity
```

```
from three_colouring import ThreeColouring
from itertools import combinations

class PolynomialTimeAlgorithm:
    def __init__(self,graph: Graph):
        self.graph = graph
        self.is_recursive_call = False
        self.init_graph = graph.copy()
        self.contracted_vertex = None

    def three_colouring(self):
        try:
            line_1_result = self.line_1_check()
            if line_1_result:
                print("Line 1 check failed. There is a K4 in the graph so it is not 3-colourable.
                    ")
                return False

            line_9_condition, vertices_to_contract = self.line_9_check()
            if line_9_condition:
                self.perform_contraction(vertices_to_contract)
                return self.three_colouring()

            line_13_condition,minimal_stable_separator = self.line_13_check()
            if line_13_condition:
                self.perform_contraction(minimal_stable_separator)
                return self.three_colouring()

            colouring = ThreeColouring(self.graph,self.init_graph)

            return colouring.construct_three_colouring()
        except Exception as e:
            print(f"An exception occurred: {e}")
            raise

    def line_1_check(self):
        if self.is_recursive_call:
            return self.graph.find_triangle_in_neighborhood(self.contracted_vertex)
        else:
            return self.graph.find_K4()
```

```
def line_9_check(self):
    result = self.graph.find_diamond()
    if result is None:
        return False , None
    else:
        return True , result


def line_13_check(self):
    """
    this should run in O(n*m) time
    if G contains a minimal stable separator S with |S| ≥2 then
        Recursively find a 3-colouring of G/S
    """
    # Find the biconnected components of the graph
    biconnectivity_algorithm = TarjansBiconnectivity(self.graph)
    blocks, self.graph.cutpoints = biconnectivity_algorithm.find_biconnected_components()

    for block in blocks:
        print("block:", block)

    print("cutpoints:",self.graph.cutpoints)

    # delete old blocks from graph and add the new blocks
    self.graph.blocks = {}

    for i, block in enumerate(blocks):# i represents the block id
        self.graph.blocks[i] = block


    for vertex in self.graph.vertices:
        for block_id in self.graph.blocks:

            block = self.graph.blocks[block_id]
            if vertex in block.vertices and len(block.vertices) >= 3:
                cond , minimal_stable_separator = block.find_minimal_stable_separator(vertex)
                if cond:
                    return True, minimal_stable_separator

    return False, None
```

```
    def perform_contraction(self, vertices_to_contract):
        self.graph = self.graph.contract(vertices_to_contract)
        self.contracted_vertex = rename_vertices_to_contract(vertices_to_contract)
        self.is_recursive_call = True
        self.graph.show("contracted-graph")


    def run(self):
        return self.three_colouring()
```

Εξηγούμε τις συναρτήσεις της κλάσης:

1. Κατασκευαστής (__init__): Δέχεται ένα αντικείμενο Graph και αρχικοποιεί μερικές μεταβλητές που θα χρειαστούν για την σωστή λειτουργία του αλγορίθμου.

2. Η συνάρτηση `three_colouring`: Είναι η κύρια συνάρτηση που εκτελεί τα βήματα του αλγορίθμου για την επίλυση του προβλήματος 3-Χρωματισμού.

3. Η συνάρτηση `line_1_check`: Υλοποιεί τον έλεγχο της πρώτης γραμμής του αλγορίθμου.: Ελέγχει αν υπάρχει κύκλος μεγέθους 4 (K_4) στο G.

4. Η συνάρτηση `line_5_check`: Ελέγχει αν υπάρχει διαμάντι στον γράφημα.

5. Η συνάρτηση `line_9_check`: Ελέγχει αν υπάρχει ελάχιστο σταθερό διαχωριστής στο γράφημα.

6. Η συνάρτηση `perform_contraction`: Βοηθητική συνάρτηση που εκτελεί σύμπτυξη δύο κορυφών.

3.6.2 Η Κλάση Graph

graph.py

```
from utilities import *
from itertools import combinations
from pyvis.network import Network


class Graph:
    show_count = 0 # Class-level variable to keep track of show calls
```

```
def __init__(self, num_of_vertices, num_of_edges,edges,vertices = None):
    self.num_of_vertices = num_of_vertices
    self.num_of_edges = num_of_edges
    self.vertices = set(str(i) for i in range(num_of_vertices)) if vertices is None else
        vertices
    self.edges = set((str(u),str(v)) for u,v in edges)
    self.adjacency_list = { str(vertex): set() for vertex in self.vertices}
    for edge in self.edges:
        u, v = edge
        self.adjacency_list[u].add(v)
        self.adjacency_list[v].add(u)

    self.blocks = {}
    self.cutpoints = set()
    self.vertices_color = None



def is_triangle(self):
    if self.num_of_vertices == 3 or self.num_of_edges == 3:
        return True
    else:
        return False

def find_K4(self):
    """
    The function `find_K4` checks if a graph contains a complete graph K4.
    :return: a boolean value. It returns True if a K4 subgraph is found in the given graph,
        and False
    otherwise.
    """
    for edge in self.edges:
        u, v = edge
        disjoint_edges = set( (x,y) for (x,y) in self.edges if x not in (u,v) and y not in (u
            ,v))
        for across_edge in disjoint_edges:
            x, y = across_edge
            edges_found = 0
            if ( ( (x,u) in self.edges or (u,x) in self.edges )
                 and ( (y,v) in self.edges or (v,y) in self.edges )
                   and ( (x,v) in self.edges or (v,x) in self.edges )
                    and ( (y,u) in self.edges or (u,y) in self.edges ) ):
```

```
                return True
        return False



def find_triangle_in_neighborhood(self, contracted_vertex):
    """
    Test if the neighborhood of contracted_vertex in graph contains a triangle.
    This function is designed to be called when Line 1 is reached via recursion.
    This function should run in time O(n*m)

    :param graph: The graph instance
    :param contracted_vertex: The contracted vertex in the graph
    :return: True if the neighborhood of s contains a triangle, otherwise False
    """
    neighbors = self.adjacency_list[contracted_vertex]
    if len(neighbors) < 3:
        return False
    else:
        for sub_vertices in combinations(neighbors, 3):
            if all((u, v) in self.edges or (v, u) in self.edges for u, v in combinations(
                sub_vertices, 2)):
                return True

    return False

def find_diamond(self):
    """
    This function should run in time O(n*m)
    The function "find_diamond" searches for a diamond pattern in a graph by finding two
        nodes that
    have at least two common neighbors.
    :return: a set of common neighbors between two vertices in a graph. If there are at
        least two
    common neighbors, the function returns the set of common neighbors. If there are no
        common
    neighbors or less than two common neighbors, the function returns None.
    """
    for edge in self.edges:
        u, v = edge
        adj_u = self.adjacency_list[u]
```

```
            adj_v = self.adjacency_list[v]

            common_neighbors = adj_u & adj_v

            if len(common_neighbors) >= 2:
                return common_neighbors
        return None

    def contract(self, vertices_to_contract):
        """
        Contract a set of vertices into a single vertex.

        :param vertices_to_contract: set of vertices to contract
        :return: A new Graph instance with the vertices contracted.
        """

        #All vertices to contract will be replaced by a new vertex
        new_vertex = rename_vertices_to_contract(vertices_to_contract)
        new_vertices = [v for v in self.vertices if v not in vertices_to_contract] + [new_vertex
            ]

        # All edges incident to a vertex in vertices_to_contract will now be incident to
            new_vertex
        new_edges = []
        for u, v in self.edges:
            if u in vertices_to_contract and v in vertices_to_contract:
                # Ignore edges within the contracted set
                continue
            elif u in vertices_to_contract:
                new_edges.append((new_vertex, v))
            elif v in vertices_to_contract:
                new_edges.append((u, new_vertex))
            else:
                new_edges.append((u, v))


        #(self, num_of_vertices, num_of_edges,edges,vertices = None)
        return Graph(len(new_vertices), len(new_edges), new_edges, new_vertices)
```

```
def delete_vertices(self, vertices_to_delete):
    """
    Delete a set of vertices from the graph.

    :param vertices_to_delete: set of vertices to delete
    :return: A new Graph instance with the vertices deleted.
    """
    new_vertices = set(self.vertices - vertices_to_delete)
    print('new_vertices:',new_vertices)
    new_edges = set(e for e in self.edges if e[0] not in vertices_to_delete and e[1] not in
        vertices_to_delete)#TODO
    print("new_edges:",new_edges)
    return Graph(len(new_vertices), len(new_edges), new_edges, new_vertices)


def edge_exists(self, u, v):
    return (u, v) in self.edges or (v, u) in self.edges


def copy(self):
    return Graph(self.num_of_vertices, self.num_of_edges, self.edges, self.vertices)


def show(self,graph_name='graph'):
    Graph.show_count += 1
    print("Showing graph:",Graph.show_count)
    net = Network(height="500px", width="100%", bgcolor="#222222", font_color="white")

    for vertex in self.vertices:
        # If vertex colors are provided, use them; otherwise, default to None
        color = self.vertices_color.get(vertex) if isinstance(self.vertices_color, dict) else
            None
        net.add_node(vertex, color=color)


    for edge in self.edges:
        u, v = edge
        net.add_edge(u, v, color="white")

    file_name = f"../output-graphs/{graph_name}-{Graph.show_count}.html"
    net.show(file_name)
```

Εξηγούμε τις συναρτήσεις της κλάσης:

1. `is_triangle`: Ελέγχει εάν το γράφημα είναι τρίγωνο, δηλαδή αν έχει 3 κορυφές ή 3 ακμές.
2. `find_K4`: Ελέγχει εάν το γράφημα περιέχει ένα K_4.
3. `find_triangle_in_neighborhood`: Ελέγχει εάν η γειτονιά μιας συγκεκριμένης κορυφής περιέχει τρίγωνο. Αυτή η συνάρτηση καλείται για τον έλεγχο της γραμμής 1 του αλγορίθμου μετά από αναδρομή.
4. `find_diamond`: Η συνάρτηση ψάχνει να βρει άμα υπάρχει διαμάντι στο γράφημα, βρίσκοντας δύο κορυφές που έχουν τουλάχιστον δύο κοινούς γείτονες.
5. `contract`: Συμπτύσσει ένα σύνολο κορυφών σε μια μοναδική κορυφή.
6. `delete_vertices`: Διαγράφει ένα σύνολο κορυφών από το γρφημα.
7. `edge_exists`: Ελέγχει εάν μια ακμή υπάρχει στο γράφημα.
8. `copy`: Δημιουργεί αντίγραφο του γράφου.
9. `show`: Εμφανίζει το γράφημα χρησιμοποιώντας τη βιβλιοθήκη pyvis σε μορφή διαδραστικού δικτύου.

3.6.3 Η Κλάση Block

Η κλάση Block αναπαριστά τα μπλοκς του αλγορίθμου. Περιλαμβάνει μεθόδους για την εύρεση ελάχιστων σταθερών διαχωριστών και τον έλεγχο της συνδεσιμότητας

block.py

```
from utilities import *
from itertools import combinations
from graph import Graph

class Block(Graph):
    def __init__(self, vertices, edges):
        num_of_vertices = len(vertices)
        num_of_edges = len(edges)
        super().__init__(num_of_vertices, num_of_edges, edges, vertices)
```

```
def find_minimal_stable_separator(self, x):
    x_neighbors = self.adjacency_list[x]
    cond, stable_cutset = self.find_stable_cutset(x, x_neighbors)

    if cond:
        minimal_stable_separator = self.reduce_cutset_to_minimal_stable_separator(x,
            stable_cutset)
        return True, minimal_stable_separator
    else:
        return False, None

def find_stable_cutset(self, starting_vertex, S):
    unique_edge = None

    edge_found = False
    for u, v in combinations(S, 2):
        if (u, v) in self.edges or (v, u) in self.edges:
            if not edge_found: # Check if no edge was found previously
                unique_edge = (u, v)
                edge_found = True # Set the flag to indicate an edge has been found
            else:
                unique_edge = None # More than one edge found, so it's not unique
                break # Exit the loop as we are only interested in unique edge

    if unique_edge is None:
        #check if S is stable cutset of block
        block_without_S = self.delete_vertices(S)
        if not block_without_S.is_connected(starting_vertex):
            return True , S
        else:
            return False, None
    else:
        u, v = unique_edge
        S_without_u = S - {u}
        block_without_S_u = self.delete_vertices(S_without_u)

        if not block_without_S_u.is_connected(starting_vertex):
            return True, S_without_u

        S_without_v = S - {v}
        block_without_S_v = self.delete_vertices(S_without_v)
```

```
        if not block_without_S_v.is_connected(starting_vertex):
            return True, S_without_v

        return False, None

def reduce_cutset_to_minimal_stable_separator(self, starting_vertex, S):
    separator = S.copy()
    for u in separator:
        if len(S) >= 3:
            S_without_u = S - {u}
            block_without_S_u = self.delete_vertices(S_without_u)
            if not block_without_S_u.is_connected(starting_vertex):
                S = S_without_u.copy()
        else:
            break
    return S

def delete_vertices(self, vertices_to_delete):
    """
    Delete a set of vertices from the block.

    :param vertices_to_delete: set of vertices to delete
    :return: A new Block instance with the vertices deleted.
    """
    new_vertices = set(self.vertices - vertices_to_delete)
    new_edges = set(e for e in self.edges if e[0] not in vertices_to_delete and e[1] not in
        vertices_to_delete)

    return Block(new_vertices,new_edges)

def is_connected(self, start):
    if not self.vertices:
        return True
    visited = self.dfs(start)

    return len(visited) == len(self.vertices)

def dfs(self, start=None, visited=None):
    if start is None:
        start = next(iter(self.vertices))
    if visited is None:
```

```
        visited = set()
    visited.add(start)

    if not self.adjacency_list[start]:
        return visited

    for neighbor in self.adjacency_list[start]:
        if neighbor not in visited:
            self.dfs(neighbor, visited)
    return visited


def copy(self):
    return Block(self.vertices.copy(), self.edges.copy())
```

Εξηγούμε τις συναρτήσεις της κλάσης:

1. `find_minimal_stable_separator`: Αναζητά τον ελάχιστο σταθερό διαχωριστικό σε έναν κόμβο x.
2. `find_stable_cutset`: Ελέγχει εάν μια συγκεκριμένη γειτονιά του κόμβου x είναι ένα σταθερό διαχωριστικό.
3. `reduce_cutset_to_minimal_stable_separator`: Μειώνει ένα διαχωριστικό σύνολο σε ένα ελάχιστο σταθερό διαχωριστικό.
4. `delete_vertices`: Διαγράφει ένα σύνολο κορυφών από το μπλοκ.
5. `is_connected`: Ελέγχει εάν το μπλοκ είναι συνδεδεμένο.
6. `dfs`: Υλοποιεί τον αλγόριθμο του "Depth-first search" για να ελέγξει τη συνδεσιμότητα του μπλοκ.
7. `copy`: Δημιουργεί αντίγραφο του μπλοκ.

3.6.4 Η Κλάση ThreeColouring

Η κλάση ThreeColouring χρωματίζει με τρία χρώματα το συμπτυγμένο γράφημα που προκύπτει στην γραμμή 13 του αλγορίθμου 2.3.

three_colouring.py

```
from utilities import *
from block_cutpoint_tree import BlockCutpointTree
from itertools import combinations

class ThreeColouring:
    def __init__(self,graph, initial_graph):
        self.graph = graph
        self.initial_graph = initial_graph
        self.previous_triangle = []

    def construct_three_colouring(self):
        print("constructing three colouring...")
        self.graph.vertices_color = {vertex: None for vertex in self.graph.vertices}

        if len(self.graph.blocks) == 1:
            #Then the graph is a triangle or a triangular strip
            if self.graph.is_triangle():
                self.colour_triangle(self.graph.vertices)
            else:#graph is a triangular strip
                self.colour_trianglular_strip(self.graph.copy())
        else:
            #create block-cutpoint tree
            block_cutpoint_tree = BlockCutpointTree(self.graph.blocks, self.graph.cutpoints)
            block_cutpoint_tree.show()
            for block_id in self.graph.blocks:
                print("block id:",block_id, "block vertices:",self.graph.blocks[block_id].
                    vertices)

            next_block_id = block_cutpoint_tree.get_next_block()
            while next_block_id is not None:
                print("\nIm colouring block:",next_block_id)
                block = self.graph.blocks[next_block_id]
                print("block vertices:",block.vertices)
                if block.num_of_vertices < 3:
                    pass # we will colour this block later
                elif block.num_of_vertices == 3:
                    self.colour_triangle(block.vertices)
                else: # block is a triangular strip
                    triangular_strip = block.copy()
                    next_vertex = None
                    is_cutpoint = False
```

```
                for vertex in triangular_strip.vertices:
                    if vertex in self.graph.cutpoints and self.graph.vertices_color[vertex] is
                         not None:
                        next_vertex = vertex
                        is_cutpoint = True
                        break
                self.colour_trianglular_strip(triangular_strip,next_vertex, is_cutpoint)



            next_block_id = block_cutpoint_tree.get_next_block()
            print("next block id:",next_block_id)


        self.colour_remaining_vertices()

    self.graph.show("three colouring before expansion")


    initial_graph = self.colour_initial_graph()

    initial_graph.show("three-colouring")

def colour_triangle(self,triangle_vertices):

    cutpoint = None
    available_colors = {'red', 'green', 'blue'}

    if self.graph.cutpoints:
        for vertex in triangle_vertices:
            if vertex in self.graph.cutpoints and self.graph.vertices_color[vertex] is not
                None:
                available_colors.remove(self.graph.vertices_color[vertex])
                cutpoint = vertex
                break


    for vertex in triangle_vertices:
        if vertex != cutpoint:
            self.graph.vertices_color[vertex] = available_colors.pop()
```

```
def colour_trianglular_strip(self, triangular_strip, next_vertex=None, is_cutpoint=False):
    if triangular_strip.num_of_vertices == 6:
        # then the triangular strip is a prism
        self.colour_prism(triangular_strip, next_vertex)
        return

    if next_vertex is None:
        first_triangle, next_vertex = self.find_init_triangle_in_strip(triangular_strip)
        self.colour_triangle(first_triangle)
        self.previous_triangle = first_triangle
        triangular_strip = triangular_strip.delete_vertices(first_triangle)
        self.colour_trianglular_strip(triangular_strip, next_vertex)
    elif next_vertex is not None and next_vertex in self.graph.cutpoints:
        triangle, next_vertex = self.find_triangle_in_strip(next_vertex, triangular_strip)
        self.colour_triangle(triangle)
        self.previous_triangle = triangle
        triangular_strip = triangular_strip.delete_vertices(triangle)
        self.colour_trianglular_strip(triangular_strip, next_vertex)
    else:
        triangle, next_vertex = self.find_triangle_in_strip(next_vertex, triangular_strip)
        self.colour_triangle_of_trianglular_strip(triangle)
        self.previous_triangle = triangle
        triangular_strip = triangular_strip.delete_vertices(triangle)
        self.colour_trianglular_strip(triangular_strip, next_vertex)



def find_init_triangle_in_strip(self,triangular_strip):
    for vertex in triangular_strip.vertices:
        if len(triangular_strip.adjacency_list[vertex]) == 3:
            triangle,next_vertex = self.find_triangle_in_strip(vertex,triangular_strip)
            return triangle,next_vertex

def find_triangle_in_strip(self,vertex,triangular_strip):
    triangle = {vertex}
    next_vertex = None
    for neighbour in triangular_strip.adjacency_list[vertex]:
        if len(triangular_strip.adjacency_list[neighbour]) == 3:
            triangle.add(neighbour)

        else:
            next_vertex = neighbour
```

```
    return triangle, next_vertex


def colour_triangle_of_trianglular_strip(self,triangle):
    for vertex in triangle:
        for previous_vertex in self.previous_triangle:
            if previous_vertex in self.graph.adjacency_list[vertex]:
                previous_vertex_color = self.graph.vertices_color[previous_vertex]
                self.graph.vertices_color[vertex] = get_next_colour(previous_vertex_color)
                break

def colour_prism(self, prism, next_vertex):
    first_triangle, second_triangle = self.find_triangles_in_prism(prism)

    if next_vertex is None or next_vertex in first_triangle:
        self.colour_triangle(first_triangle)
        self.previous_triangle = first_triangle
        self.colour_triangle_of_trianglular_strip(second_triangle)
    else:
        self.colour_triangle_of_trianglular_strip(second_triangle)
        self.previous_triangle = second_triangle
        self.colour_triangle_of_trianglular_strip(first_triangle)


def find_triangles_in_prism(self,prism):
    first_triangle = set()
    second_triangle = set()
    prism_vertices = prism.vertices
    for vertices in combinations(prism_vertices,3):
       if self.graph.edge_exists(vertices[0],vertices[1]) and\
            self.graph.edge_exists(vertices[1],vertices[2]) and \
                self.graph.edge_exists(vertices[2],vertices[0]):
           first_triangle = set(vertices)
           break


    second_triangle = prism_vertices - first_triangle

    return first_triangle, second_triangle
```

```
def colour_remaining_vertices(self):
    none_coloured_vertices = [vertex for vertex in self.graph.vertices if self.graph.
        vertices_color[vertex] is None]
    print("none coloured vertices:",none_coloured_vertices)
    for vertex in none_coloured_vertices:
        for neighbour in self.graph.adjacency_list[vertex]:
            available_colors = {'red', 'green', 'blue'}
            if self.graph.vertices_color[neighbour] in available_colors:
                available_colors.remove(self.graph.vertices_color[neighbour])
        self.graph.vertices_color[vertex] = available_colors.pop()
```

```
def colour_initial_graph(self):
    self.initial_graph.vertices_color = {vertex: None for vertex in self.initial_graph.
        vertices}

    for vertex in self.graph.vertices:
        expanded_vertices = expand_contracted_vertices(vertex)
        for expanded_vertex in expanded_vertices:
            self.initial_graph.vertices_color[expanded_vertex] = self.graph.vertices_color[
                vertex]

    return self.initial_graph
```

Εξηγούμε τις συναρτήσεις της κλάσης:

1. `construct_three_colouring`: Είναι η βασική συνάρτηση χρωματισμού του γραφήματος. Αρχικά ελέγχει πόσα μπλοκς υπάρχουν στο γράφημα. Αν είναι περισσότερα από ένα υπολογίζει ένα δέντρο για με τα μπλοκς χρησιμοποιώντας τα cutpoints του γραφήματος.

2. `colour_triangle`: Χρωματίζει ένα τρίγωνο με τρία διαφορετικά χρώματα.

3. `colour_triangular_strip`: Χρωματίζει μια τριγωνική λωρίδα με τρία χρώματα.

4. `find_init_triangle_in_strip`: Εντοπίζει το αρχικό τρίγωνο σε μια τριγωνική λωρίδα.

5. `find_triangle_in_strip`: Βρίσκει ένα τρίγωνο σε μια τριγωνική λωρίδα, ξεκινώντας από μια δεδομένη κορυφή.

6. `colour_triangle_of_triangular_strip`: Χρωματίζει ένα τρίγωνο μιας τριγωνικής λωρίδας.

7. `colour_prism`: Χρωματίζει ένα πρίσμα, δηλαδή μια τριγωνικό λωρίδα τάξης 2.

8. `find_triangles_in_prism`: Εντοπίζει τα δύο τρίγωνα σε ένα πρίσμα.

9. `colour_remaining_vertices`: Χρωματίζει τις κορυφές του γραφήματος που δεν ανήκουν σε κάποιο μπλοκ, χρησιμοποιώντας τρία χρώματα και αποφεύγοντας τις συγκρούσεις χρωμάτων.

10. `colour_initial_graph`: Χρωματίζει το αρχικό γράφημα από το συμπαγές.

3.6.5 Η Κλάση BlockCutpointTree

Η κλάση BlockCutpointTree δημιουργεί μια δενδρική δομή όπου τα μπλοκ είναι κόμβοι και τα μπλοκ που μοιράζονται ένα σημείο αποκοπής συνδέονται με μια ακμή, περιέχει μεθόδους πλοήγησης και εμφάνισης του δέντρου.

block_cutpoint_tree.py

```
from graph import Graph
from pyvis.network import Network

class BlockCutpointTree:
    current_block = None

    def __init__(self, blocks, cutpoints):
        self.blocks = blocks
        self.cutpoints = cutpoints
        self.root = None
        self.tree = self.create_tree()
        self.visited = set()

    def create_tree(self):
        """
        For every block of the graph, create a node in the tree.
        If two blocks(nodes) contains the same cutpoint, then we connect them with an edge.
        """

        vertices = set()
```

```
    edges = set()
    for block_id in self.blocks:
        print("type(block_id)",type(block_id))
        vertices.add(block_id)

    for i in range(len(self.blocks)):
        for j in range(i+1,len(self.blocks)):
            for cutpoint in self.cutpoints:
                if cutpoint in self.blocks[i].vertices and cutpoint in self.blocks[j].
                    vertices:
                    print("(i,j)",(i,j))
                    edges.add((i,j))

    #vertices to string
    vertices = set(str(v) for v in vertices)

    return Graph(len(vertices),len(edges),edges, vertices)

def get_next_block(self):
    if self.current_block is None:
        # Find the first unvisited node to start
        for node in self.tree.vertices:
            if node not in self.visited:
                self.current_block = node
                break
        else:
            # No unvisited nodes left
            return None

    stack = [self.current_block]
    while stack:
        node = stack[-1]
        if node not in self.visited:
            self.visited.add(node)
            self.current_block = node
            return int(node)

        # Explore neighbors
        unvisited_neighbors = [neighbor for neighbor in self.tree.adjacency_list[str(node)]
            if neighbor not in self.visited]
        if unvisited_neighbors:
```

```
            stack.extend(unvisited_neighbors)
        else:
            stack.pop() # Backtrack

        # If the stack is empty and there are still unvisited nodes, continue from an
            unvisited node
        if not stack:
            for next_node in self.tree.vertices:
                if next_node not in self.visited:
                    stack.append(next_node)
                    break

    # If all nodes are visited, return None
    return None


def show(self):
    """
    Show the tree.
    """
    self.tree.show("block_cutpoint_tree")
```

Εξηγούμε τις συναρτήσεις της κλάσης:

1. `__init__`: Ο κατασκευαστής της κλάσης που αρχικοποιεί τις μεταβλητές blocks, cutpoints, root, tree και visited.

2. `create_tree`: Δημιουργεί το δέντρο απόφασης (Block-Cutpoint Tree) βασισμένο στα μπλοκ και τα cutpoints. Δημιουργεί έναν κόμβο για κάθε μπλοκ και συνδέει με ακμές τους κόμβους που αντιστοιχούν σε μπλοκ που περιέχουν το ίδιο cutpoint.

3. `get_next_block`: Επιστρέφει τον επόμενο μη επισκεπτημένο κόμβο του δέντρου Block-Cutpoint Tree. Χρησιμοποιείται για τον εντοπισμό επόμενου μπλοκ που πρέπει να εξεταστεί.

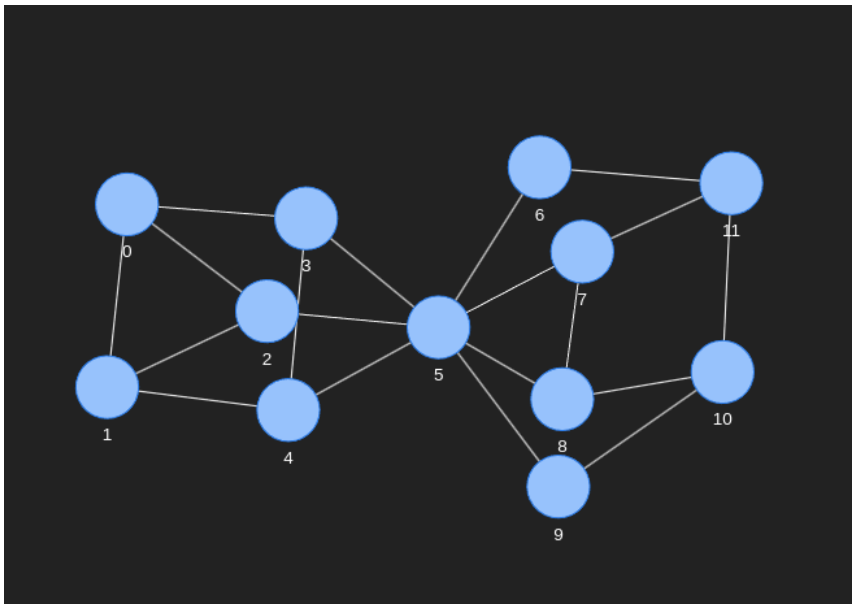
4. `show`: Εμφανίζει γραφικά το δέντρο Block-Cutpoint Tree.

3.6.6 Παραδείγματα Εκτέλεσης του Προγράμματος

Ως παράδειγμα δείχνουμε πως εκτελούμε τον αλγόριθμο 3-χρωματισμού στο ίδιο γράφημα με την Ενότητα 2.3.

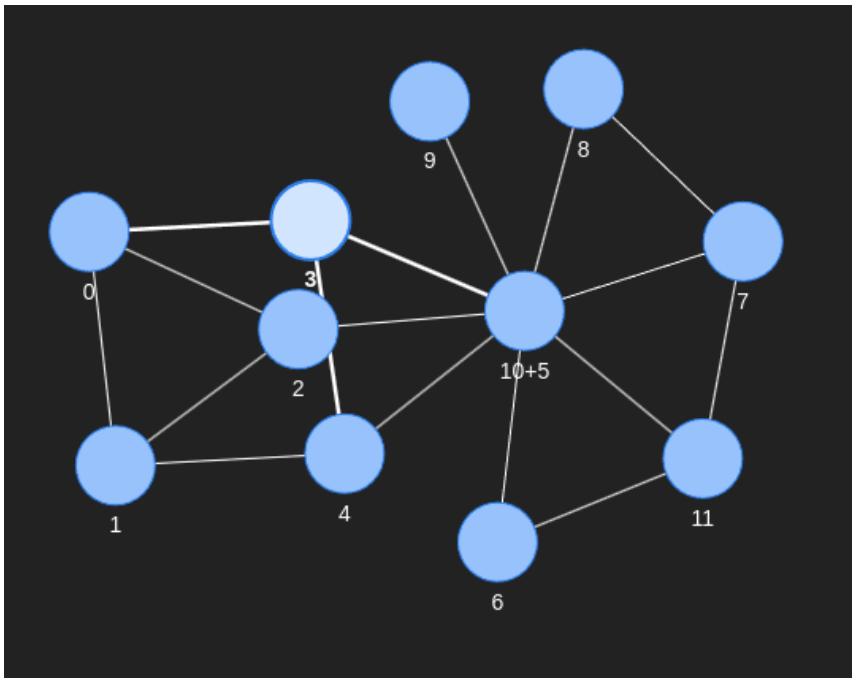
Το γράφημα σε μορφή JSOn

```
{
   "num_of_vertices" : 12,
   "num_of_edges" : 19,
   "edges" :
      [
         [0,1],
         [0,2],
         [2,1],
         [2,5],
         [0,3],
         [1,4],
         [3,4],
         [3,5],
         [5,4],
         [5,6],
         [5,9],
         [6,11],
         [9,10],
         [5,7],
         [5,8],
         [7,8],
         [7,11],
         [8,10],
         [11,10]
      ]
}
```

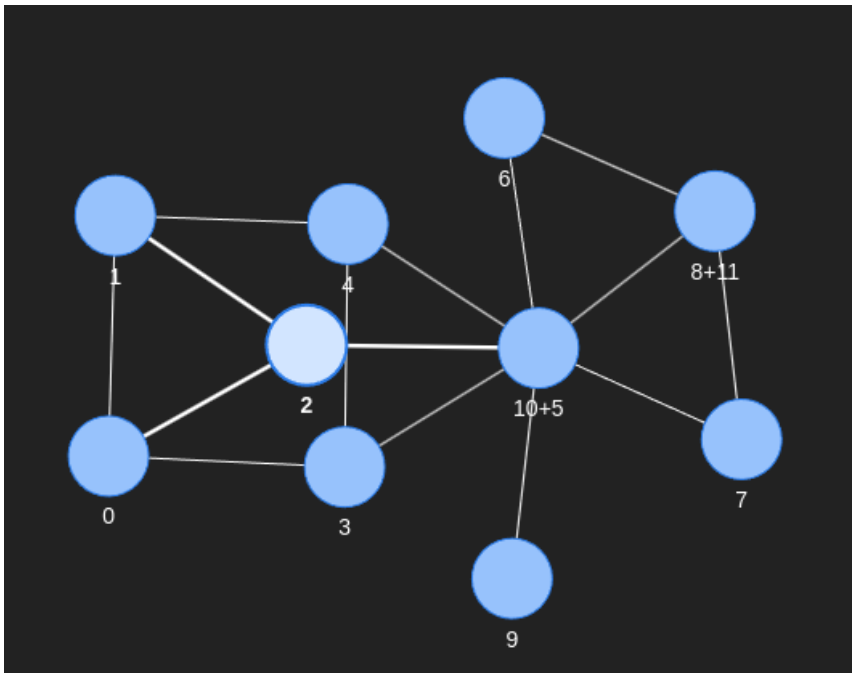


Σχήμα 3.5: Το γράφημα εισόδου του Αλγορίθμου σχεδιασμένο από την βιβλιοθήκη PyVis

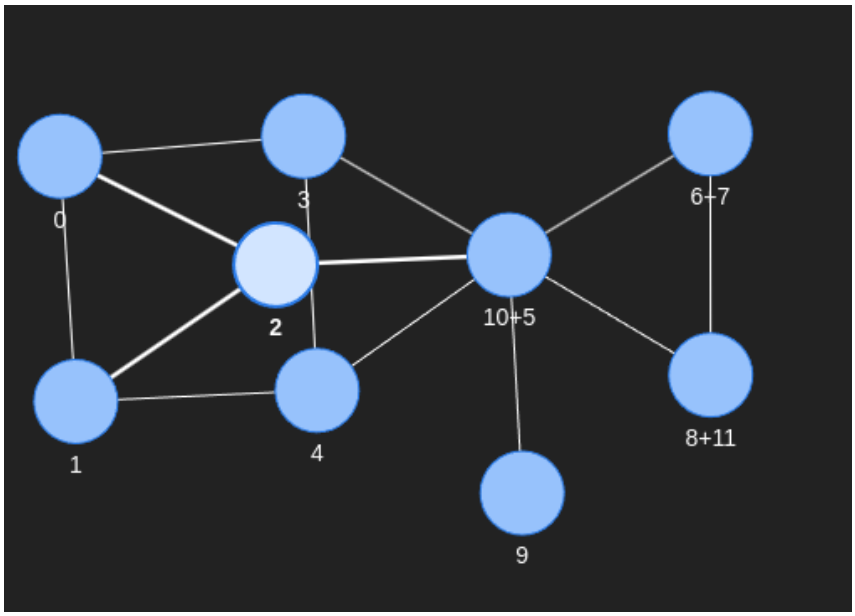
Παρακάτω δείχνουμε όλα τα στιγμιότυπα του γραφήματος κατά την διάρκεια εκτέλεσης του αλγορίθμου.



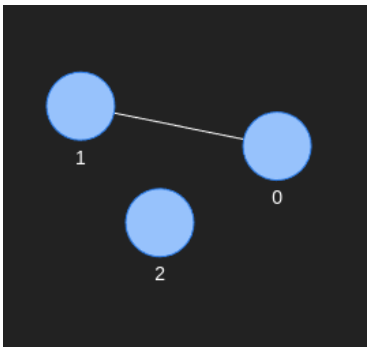
Σχήμα 3.6: Το γράφημα μετά την σύμπτυξη των κορυφών 10,5



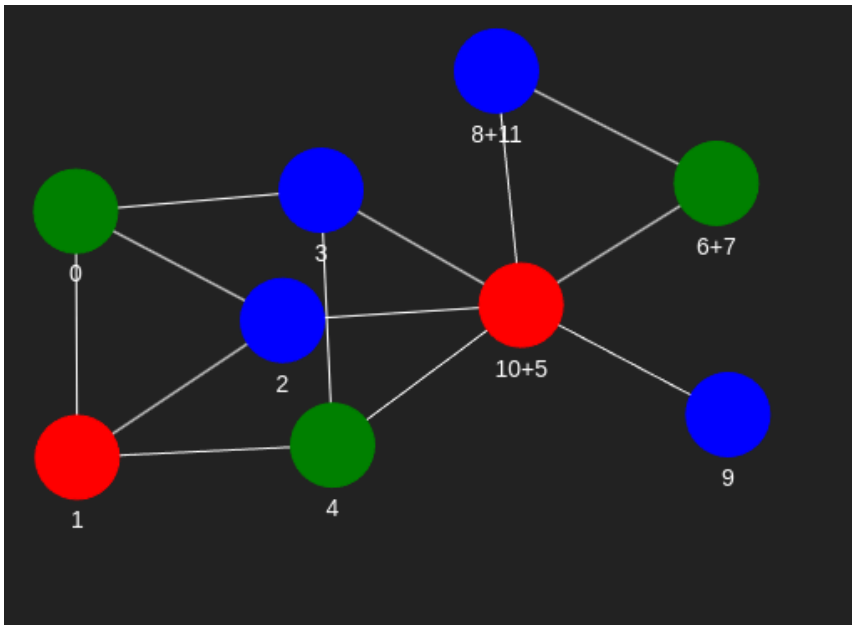
Σχήμα 3.7: Το γράφημα μετά την σύμπτυξη των κορυφών 8,11



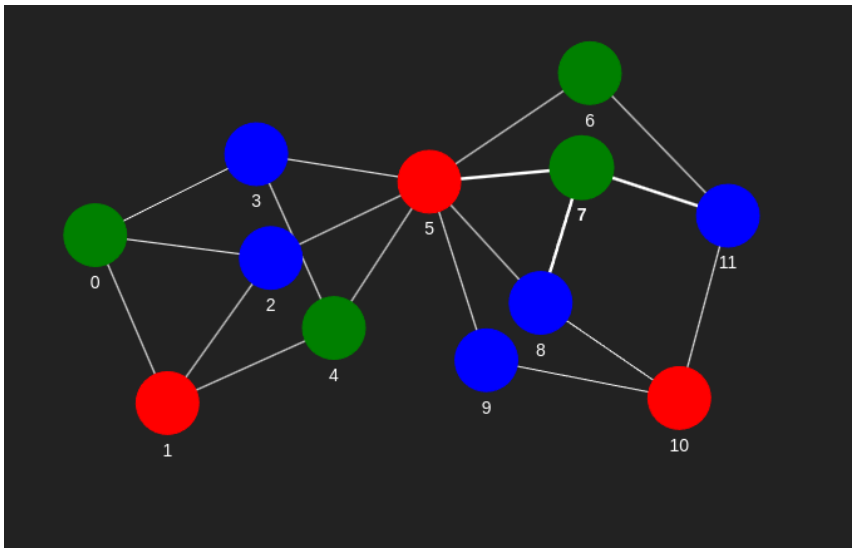
Σχήμα 3.8: Το γράφημα μετά την σύμπτυξη των κορυφών 7,6



Σχήμα 3.9: Block Cutpoint Tree { 0: {8+11,6+7,10+5}, 1: {1,10+5,2,3,4,0}, 2: {9}}



Σχήμα 3.10: Το ”συμπτυγμένο” γράφημα μετά τον 3-χρωματισμό



Σχήμα 3.11: Το γράφημα μετά το τέλος του αλγορίθμου

ΚΕΦΑΛΑΙΟ 4

Επίλογος

Στην παρούσα εργασία καταφέραμε να υλοποιήσουμε τρεις κλασικούς αλγορίθμους της θεωρίας γραφημάτων για τα AT-free γραφήματα. Συγκεκριμένα μελετήσαμε τον αλγόριθμο των Hajo Broersma , Ton Kloks , Dieter Kratsch , και Haiko Müller[2] για τον υπολογισμό του ανεξάρτητου αριθμού τον οποίο επεκτείναμε ώστε να επιστρέφει και ένα μέγιστο ανεξάρτητο σύνολο, τον αλγόριθμο του Dieter Kratsch[19] για τον εντοπισμό ελάχιστου κυρίαρχου συνόλου και τον αλγόριθμο του Juraj Stacho[17] για την αντιμετώπιση του προβλήματος του 3-χρωματισμού σε AT-free γραφήματα.

Για το Πρόβλημα του 3-Χρωματισμού χρειάστηκε να υλοποιήσουμε και τον αλγόριθμο "Biconnectivity" του Robert Tarjan [28].

Δεν έχουμε υλοποιήσει τον αλγόριθμο εντοπισμού ενός κυρίαρχου ζεύγους [3], που χρειάζεται ώστε ο αλγόριθμος του Dieter Kratsch[19], να έχει χρονική πολυπλοκότητα $O(n^6)$. Αυτό κάνει τον αλγόριθμο που έχουμε υλοποιήσει να τρέχει σε $O(n^7)$ καθώς χρειάζεται να γίνει ο έλεγχος για όλους τους κόμβους του γραφήματος.

Για τους αλγορίθμους υπολογισμού μέγιστου ανεξάρτητου συνόλου και ελάχιστου κυρίαρχου συνόλου έχουμε υλοποιήσει αλγόριθμους "Brute force" που ελέγχουν την ορθότητα των αλγορίθμων που έχουμε υλοποιήσει. Για το πρόβλημα του 3-χρωματισμού υλοποιήσαμε μια συνάρτηση που ελέγχει την γειτονιά κάθε κόμβου ώστε να μην έχει το ίδιο χρώμα.

Καταφέραμε να υλοποιήσουμε όλους τους αλγορίθμους, μέσα στα όρια της πολυπλοκότητας τους, όπως αυτή περιγράφεται στις αντίστοιχες αναφορές. Παρόλα αυτά δεν έχουμε υλοποιήσει πιστοποιητικά που ελέγχουν την χρονική και χωρική πολυπλοκότητα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Brandstädt, V.D. Chepoi, and F.F. Dragan. The algorithmic use of hypertree structure and maximum neighbourhood orderings. *Discrete Applied Mathematics*, 82(1-3):43–77, 1998.

[2] Kloks T. Kratsch D. Broersma, H. and H. Müller. Independent sets in asteroidal triple-free graphs. *SIAM Journal on Discrete Mathematics*, 12(2):276–287, 1999.

[3] D.G. Corneil, S. Olariu, and L. Stewart. A linear time algorithm to compute dominating pairs in asteroidal triple-free graphs. 1995.

[4] D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Appl. Math.*, 9:27–39, 1984.

[5] D.G. Corneil and L. Stewart. Dominating sets in perfect graphs. *Discrete Math.*, 86:145–164, 1990.

[6] D.P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are np-complete. *Discrete Math.*, 30:289–293, 1980.

[7] Johnson D.S. The np-completeness column: an ongoing guide. *J. Algorithms*, 5:147–160, 1984.

[8] E.S. Elmallah and L.K. Stewart. Independence and domination in polygon graphs. *Discrete Appl. Math.*, 44:65–77, 1993.

[9] M. Farber and M. Keil. Domination in permutation graphs. *J. Algorithms*, 6:309–321, 1985.

[10] Johnson D. Stockmeyer L. Garey, M.R. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1:237–267, 1976.

[11] Johnson D.S. Garey, M.R. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[12] Chang G.J. Labelling algorithms for domination problems in sunfree chordal graphs. *Discrete Appl. Math.*, 22:21–34, 1988.

[13] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. North-Holland, Amsterdam, 2nd edn edition, 2004.

[14] Lovász L. Schrijver A. Grötschel, M. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[15] D.G. Kirkpatrick H. Breu. Algorithms for dominating and steiner set problems in cocomparability graphs. manuscript, 1993.

[16] I. Holyer. The np-completeness of edge-coloring. *SIAM J. Comput.*, 10:718–720, 1981.

[17] 3-Colouring AT-Free Graphs in Polynomial Time. Juraj stacho. *Algorithmica*, 64(3):384–399, 2012.

[18] D. Kratsch and L. Stewart. Domination on cocomparability graphs. *SIAM J. Discrete Math.*, 6:400–417, 1993.

[19] Dieter Kratsch. Domination and total domination on asteroidal triple-free graphs. *Discrete Applied Mathematics*, 99(1-3):111–123, 2000.

[20] C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math*, 51(1):45–64, 1964.

[21] Farber M. Domination, independent domination, and duality in strongly chordal graphs. *Discrete Appl. Math.*, 7:115–130, 1984.

[22] Chang M.-S. Efficient algorithms for the domination problems on interval and circular-arc graphs. *SIAM Journal on Computing*, 27(6):1671–1694, 1998.

[23] Sau-I. Mertzios, G.B. and S. Zaks. The recognition of tolerance and bounded tolerance graphs. *SIAM Journal on Computing*, 40(5):1234–1257, 2011.

[24] Chang M.S. Efficient algorithms for the domination problems on interval and circular-arc graphs. *SIAM J. Comput.*, 27:1671–1694, 1998.

[25] pyvis Contributors. Interactive network visualizations — pyvis 0.1.3.1 documentation, n.d. Version 0.1.3.1.

[26] Liang-Y.D. Dhall S.K. Rhee, C. and S. Lakshmivarahan. An o(n + m)-time algorithm for finding a minimum-weight dominating set in a permutation graph. *SIAM J. Comput.*, 25:404–419, 1996.

[27] Arnborg S. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — a survey. *Bit Numerical Mathematics*, 25(1):1–23, 1985.

[28] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.