Ατομική Διπλωματική Εργασία

Υλοποίηση και Απεικόνιση του Αλγόριθμου Ford Fulkerson με Χρήση του Πλαισίου JavaFX

Στέλιος Ευαγόρου

Πανεπιστήμιο Κύπρου

Τμήμα Πληροφορικής

Μάϊος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

Τμήμα Πληροφορικής

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΑΠΕΙΚΟΝΙΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ FORD FULKERSON ΜΕ ΧΡΗΣΗ ΤΟΥ ΠΛΑΙΣΙΟΥ JAVAFX

Στέλιος Ευαγόρου

Επιβλέπων Καθηγητής

Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάϊος 2022

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου Δρ. Γεωργίου Χρύση, Καθηγητής στο Τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου, για την πολύτιμη καθοδήγηση, ενθάρρυνση και υποστήριξη κατά τη διάρκεια αυτού του έργου. Η συμβολή και τα σχόλια του ήταν πολύ σημαντικά για την ολοκλήρωση και τη συγγραφή αυτής της διπλωματικής. Επίσης θα ήθελα να ευχαριστήσω τους φίλους μου, τους συμφοιτητές μου αλλά και τους φοιτητές του ΕΠΛ 236 για την ανατροφοδότηση που μου έδωσαν για την λειτουργικότητα της εφαρμογής.

Περίληψη

Σε αυτή την Διπλωματική Εργασία αναλύθηκε και υλοποιήθηκε ο αλγόριθμος Ford Fulkerson σε οπτική αναπαράσταση με χρήση του εργαλείου JavaFX. Το πρόγραμμα που υλοποιήθηκε έχει σκοπό να βοηθήσει στην εκμάθηση του αλγόριθμου Ford Fulkerson. Ο χρήστης της εφαρμογής θα έχει την δυνατότητα να αλληλοεπιδράσει με την εφαρμογή και να σχεδιάσει τα δικά του δίκτυα ροής και να τρέξει τον αλγόριθμο πάνω τους έτσι ώστε να δει και να κατανοήσει πως λειτουργεί ο αλγόριθμος. Επομένως με μια δυσδιάστατη αναπαράσταση και με χρήση animation αλλά και καθοδηγώντας τον χρήστη για την σωστή σχεδίαση ενός δικτύου ροής, η κατανόηση του αλγορίθμου γίνεται πολύ πιο εύκολη ακόμη και για άτομα που δεν έχουν προηγούμενες γνώσεις στην πληροφορική.

Αρχικά σε αυτή την διπλωματική εργασία περιγράφονται τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της Διπλωματικής εργασίας. Στη συνέχεια εξηγείται ο αλγόριθμος Ford Fulkerson και ακολούθως γίνεται μια λεπτομερής περιγραφή των λειτουργιών που υποστηρίζει η εφαρμογή αλλά επίσης δίνονται και λεπτομέρειες υλοποίησης σε αλγοριθμικό επίπεδο για κάθε λειτουργία που υλοποιείται.

Ακολούθως αναφέρθηκαν κάποια προβλήματα που αντιμετωπίστηκαν κατά τη διάρκεια της υλοποίησης της διπλωματικής εργασίας και οι αντίστοιχες τους λύσεις. Τέλος αναφέρθηκαν κάποιες χρήσιμες μελλοντικές επεκτάσεις που θα μπορούσαν να γίνουν.

Περιεχόμενα

Λίστα Εικόνων

Κεφάλαιο 1

Εισαγωγή

Περιεχόμενα

1.1 Κίνητρο και Στόχος ΑΔΕ

Το κίνητρο και ο στόχος της ΑΔΕ είναι να γίνει υλοποίηση του αλγόριθμου Ford-Fulkerson κατά Edmonds-Karp [1,2] και πάνω σε αυτόν να κτιστεί ένα γραφικό περιβάλλον, που δεδομένου ενός προβλήματος αντιστοίχισης, ο χρήστης να σχεδιάζει ένα δίκτυο ροής είτε αυτοματοποιημένα είτε μη αυτοματοποιημένα αλληλοεπιδρώντας με ένα γραφικό περιβάλλον. Ακολούθως ο χρήστης θα έχει την δυνατότητα να βλέπει τον αλγόριθμο να τρέχει πάνω στο δίκτυο ροής, επίσης να βλέπει το αντίστοιχο υπολειπόμενο δίκτυο του δικτύου ροής, που προκύπτει από τις αλλαγές που έχει κάνει ο αλγόριθμος πάνω στο δίκτυο ροής, αλλά και το μονοπάτι επαύξησης που επιλέγεται κάθε φορά. Επίσης με αυτή την εφαρμογή οι χρήστες θα έχουν την δυνατότητα να μάθουν πότε ένα δίκτυο ροής είναι έγκυρο και επομένως να σχεδιάζουν ένα έγκυρο δίκτυο ροής αλλά και να μάθουν πως δημιουργείται ένα υπολειπόμενο δίκτυο δεδομένου ενός δικτύου ροής. Επομένως η εφαρμογή απευθύνεται κυρίως σε φοιτητές της Πληροφορικής αλλά και γενικά σε οποιονδήποτε ενδιαφέρεται να κατανοήσει τον αλγόριθμο Ford Fulkerson.

Η πτυχιακή αυτή έχει ως κύριο στόχο να βοηθήσει τους φοιτητές στο μάθημα ΕΠΛ 236: Αλγόριθμοι και Πολυπλοκότητα [1,2]. Ο αλγόριθμος αυτός αποτελεί ένα αρκετά σημαντικό κεφάλαιο του μαθήματος όπου παρουσιάζεται το πρόβλημα μέγιστης ροής, ο αλγόριθμος Ford-Fulkerson κατά Edmond-Karp, το θεώρημα μέγιστης ροής και ελάχιστης αποκοπής αλλά και ένα παράδειγμα εφαρμογής του αλγορίθμου. Όλα τα προαναφερθέντα θα έχει τη δυνατότητα ο χρήστης να τα δοκιμάσει και να τα δει στην πράξη, αλλά επίσης και να δημιουργήσει τα δικά του παραδείγματα εφαρμογής του αλγόριθμου.

Σημαντική είναι και η θεωρητική γνώση του αλγόριθμου Ford Fulkerson που είχε αποκτηθεί μέσω του μαθήματος ΕΠΛ 236: Αλγόριθμοι και Πολυπλοκότητα [1] κατά τη διάρκεια του Εαρινού εξαμήνου το 2020 και επομένως γνώριζα πολύ καλά το θεωρητικό κομμάτι του αλγόριθμου, πράγμα που βοήθησε στην υλοποίηση σε μεγάλο βαθμό.

Μετά την ολοκλήρωση της η εφαρμογή παρέχει στον χρήστη τις ακόλουθες δυνατότητες:

1 Διαδραστικό περιβάλλον όπου μπορεί να δημιουργήσει ένα ολοκληρωμένο δίκτυο ροής με δυνατότητες αποθήκευσης, φόρτωσης και διαγραφής αυτού του δικτύου.
2 Δημιουργία πραγματικών εφαρμογών του αλγορίθμου Ford Fulkerson με αυτοματοποιημένη δημιουργία δικτύου ροής.
3 Τρέξιμο Ford Fulkerson πάνω σε οποιοδήποτε δίκτυο ροής είτε σαν ολοκληρωμένο animation είτε τρέχοντας τον αλγόριθμο βήμα-βήμα.
4 Παροχή πληροφοριών για τα βήματα του αλγορίθμου κατά τη διάρκεια εκτέλεσης του αλγορίθμου.

1.2 Μεθοδολογία

Για την οπτική αναπαράσταση του αλγορίθμου ακολουθήθηκε η πιο κάτω μεθοδολογία.

Αρχικά έγινε μια μικρή έρευνα για την επιλογή του καταλληλότερου εργαλείου για δημιουργία μιας διαδραστικής εφαρμογής έτσι ώστε να καλύπτονται και να μπορούν να υλοποιηθούν οι ανάγκες της εφαρμογής. Από αυτή τη διαδικασία προέκυψε ότι το JavaFX [3] είναι το καταλληλότερο και αποδοτικότερο εργαλείο για την δημιουργία μιας δυσδιάστατης αναπαράστασης του αλγορίθμου. Το JavaFX είναι ένα εργαλείο ανοιχτού κώδικα, το οποίο ειδικεύεται στην δημιουργία εφαρμογών για υπολογιστές και ενσωματωμένων συστημάτων που είναι χτισμένα σε Java. Ακολούθως, έγινε μελέτη του JavaFX εργαλείου παρακολουθώντας διάφορα προγράμματα εκμάθησης, διαβάζοντας το έγγραφο οδηγιών [3] του JavaFX αλλά και βιβλία που εξηγούν πως δουλεύει το JavaFX [4] τόσο πρακτικά αλλά και θεωρητικά.

Στην συνέχεια έγινε μελέτη του αλγορίθμου Ford Fulkerson έτσι ώστε να κατανοήσω και να αποκτήσω το απαραίτητο θεωρητικό υπόβαθρο που είναι απαραίτητο για την υλοποίηση του. Επίσης πριν την υλοποίηση και τη γραφική αναπαράσταση του αλγορίθμου πάνω στον υπολογιστή, έγινε ο σχεδιασμός κάποιων πρωτότυπων πάνω στο χαρτί για καλύτερη κατανόηση του τι πρέπει να υλοποιηθεί αλλά επίσης και για την εύρεση του βέλτιστου πρωτότυπου της εφαρμογής. Αυτά τα πρωτότυπα παρατίθενται στο Κεφάλαιο 4. Ακολούθως υλοποιήθηκαν οι κατάλληλες διεπαφές χρησιμοποιώντας το εργαλείο JavaFX για την δημιουργία της γραφικής διεπαφής και ο αλγόριθμος Ford Fulkerson υλοποιήθηκε με χρήση της γλώσσας Java.

Ακολούθως έγινε μελέτη εφαρμογών με μέγιστες ροές και ελάχιστες αποκοπές σε γραφήματα. Πιο συγκεκριμένα μελετήθηκε μια πολύ βασική εφαρμογή, η οποία είναι το πρόβλημα του διμερούς ταιριάσματος πάνω σε διμερή γραφήματα. Το πρόβλημα αυτό είναι η εύρεση ενός ταιριάσματος στο G που να έχει το μεγαλύτερο δυνατό μέγεθος. Αυτό το πρόβλημα μπορεί να αναχθεί στο πρόβλημα της μέγιστης ροής και ελάχιστης αποκοπής και επομένως να επιλυθεί με τον αλγόριθμο Ford Fulkerson.

Αξίζει να σημειωθεί ότι σημαντική ήταν και η συμβολή του επιβλέποντα καθηγητή ό οποίος με καθοδήγησε τόσο στην μεθοδολογία που έπρεπε να ακολουθήσω αλλά επίσης μου παρείχε και αλγοριθμική βοήθεια όπου ήταν απαραίτητο. Επίσης σημαντική ήταν και η βοήθεια που πρόσφεραν άλλοι συμφοιτητές αλλά και φίλοι οι οποίοι έδιναν συνεχώς ανατροφοδότηση σε κάθε στάδιο υλοποίησης για πράγματα που αφορούσαν την εμφάνιση, την ευχρηστία αλλά επίσης και συμβουλές για νέες χρήσιμες προσθήκες. Η επιλογή και ο αριθμός των ατόμων έγινε έτσι ώστε οι μισοί να γνωρίζουν τον αλγόριθμο και οι άλλοι μισοί να μην τον γνωρίζουν έτσι ώστε η ανατροφοδότηση να καλύπτει και τις δυο ομάδες ατόμων. Στο Παράρτημα Α παρατίθεται και το ερωτηματολόγιο μαζί με τις απαντήσεις που ελήφθησαν.

1.3 Οργάνωση Κειμένου

Η υφιστάμενη διπλωματική εργασία αποτελείται από εφτά κεφάλαια.

Στο Κεφάλαιο 2 συνεχίζουμε με την περιγραφή της μεθοδολογίας αναφέροντας την προηγούμενη γνώση που είναι απαραίτητη για την υλοποίηση. Επιπρόσθετα εξηγούνται και τα εργαλεία τα οποία χρησιμοποιήθηκαν για την υλοποίηση της διπλωματικής εργασίας και τέλος, η επιπλέον μελέτη που χρειάστηκε έτσι ώστε να αποκτήσουμε τις επιπλέον γνώσεις που ήταν απαραίτητες για το πρακτικό κομμάτι της ΑΔΕ.

Στο Κεφάλαιο 3 δίνεται μια θεωρητική επεξήγηση του αλγόριθμου Ford Fulkerson και του προβλήματος που προσπαθεί να επιλύσει, δηλαδή το πρόβλημα της μέγιστης ροής και επίσης αναφέρονται και κάποιες πραγματικές εφαρμογές στον έξω κόσμο του προβλήματος της μέγιστης ροής.

Το Κεφάλαιο 4 περιγράφει την κύρια διεπαφή του προγράμματος. Αρχικά εξηγείται η περιγραφή της υλοποίησης της διεπαφής χρήστη, στη συνέχεια παρατίθενται τα πρωτότυπα που χρησιμοποιήθηκαν για τον σχεδιασμό της πραγματικής διεπαφής και επίσης εξηγείτε πως μπορεί να πλοηγείτε μέσα στην εφαρμογή ο χρήστης. Τέλος αναφέρονται διάφορα προβλήματα που αντιμετωπίστηκαν κατά τη δημιουργία των διεπαφών.

Στο Κεφάλαιο 5 εξηγούνται οι λεπτομέρειες υλοποίησης για κάθε λειτουργεία/κουμπί που έχει η εφαρμογή μας. Στη συνέχεια περιγράφουμε τις διαδικασίες δημιουργίας του υπολειπόμενου γραφήματος, πως υπολογίζεται η ελάχιστη αποκοπή και τέλος πως υλοποιούνται τα animations και πως ρυθμίζεται η ταχύτητα τους στην JavaFX.

Στο Κεφάλαιο 6 περιγράφεται η υλοποίηση της διεπαφής της επιλογής “εφαρμογή αλγορίθμου του Ford Fulkerson”. Πιο συγκεκριμένα εξηγούνται διάφορες λεπτομέρειες υλοποίησης για την αυτόματη δημιουργία ενός δικτύου ροής. Επίσης εξηγούνται διάφορα προβλήματα και αδυναμίες που έχει η επιλογή της εφαρμογής του Ford Fulkerson.

Τέλος, το Κεφάλαιο 7 συνοψίζει την πτυχιακή και παρουσιάζει τα συμπεράσματα μας, επίσης παρουσιάζονται διάφορα γενικά προβλήματα και τρόποι επίλυσης τους και ορισμένες μελλοντικές ενέργειες που θα μπορούσαν να καταστήσουν πιο καλή και βελτιωμένη την εργασία αυτή.

Κεφάλαιο 2

Υπόβαθρο

2.1 Προηγούμενη Εργασία

Για την διεκπεραίωση αυτής της διπλωματικής είναι απαραίτητο κάποιος να έχει ένα σημαντικό υπόβαθρο τόσο θεωρητικό αλλά και πρακτικό έτσι ώστε να μπορεί να την φέρει εις πέρας. Προηγούμενες γνώσεις που ήταν χρήσιμες:

- Γνώσεις προγραμματισμού στη γλώσσα Java (ΕΠΛ 131: Αρχές Προγραμματισμού [5]) αλλά και γνώσεις αντικειμενοστραφούς προγραμματισμού (ΕΠΛ 133: Αντικειμενοστρεφής Προγραμματισμός [6]) που βοήθησαν σε μεγάλο βαθμό στην διεκπεραίωση της ΑΔΕ, καθώς η γλώσσα υλοποίησης είναι η Java.

- Θεωρητική γνώση από το μάθημα ΕΠΛ 231: Δομές Δεδομένων και Αλγόριθμοι [7] αλλά και καλή γνώση του αλγόριθμου Ford Fulkerson από το μάθημα ΕΠΛ 236 Αλγόριθμοι και Πολυπλοκότητα [1,2].

Χρησιμοποιήθηκαν επίσης και πιο παλιές διπλωματικές εργασίες με παρόμοιες θεματικές οι οποίες βοήθησαν κυρίως στην δομή της παρούσας διπλωματικής. Την περισσότερη επιρροή είχε η διπλωματική του φοιτητή Νικόλα Τζώρτζη με την διπλωματική εργασία "Απεικόνιση Βασικών Αλγορίθμων με τη Χρήση της Μηχανής Ηλεκτρονικών Παιχνιδιών Unity 3D" [15].

2.2 Περιγραφή Εργαλείων

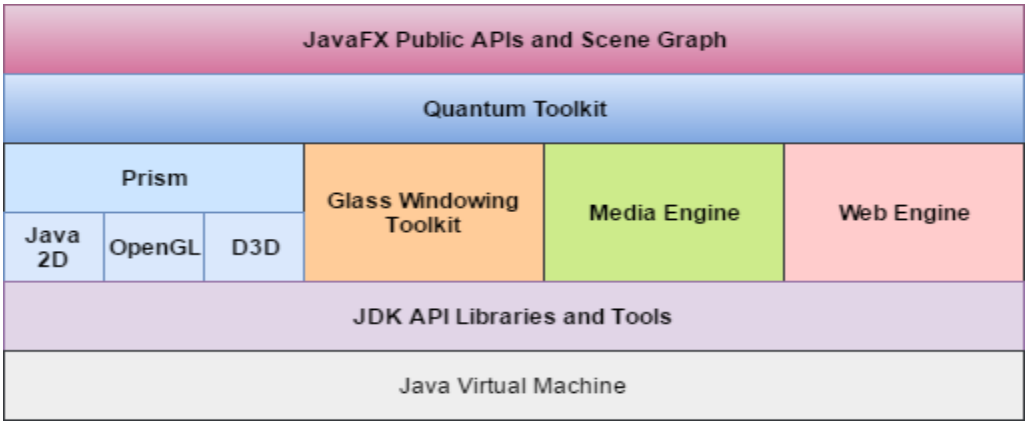
2.2.1 Ανασκόπηση JavaFX και FXML

Για να έχουμε μια ολιστική άποψη για το FXML και της χρήσης του στην JavaFX, επιβάλλεται να δούμε πώς ξεκίνησε η JavaFX και ποια είναι τα θεμέλια της. Τον Μάϊο του 2007, η εταιρεία Sun Microsystems ανακοίνωσε μια νέα προσθήκη για την γλώσσα Java, το JavaFX. Ο κύριος σκοπός του JavaFX ήταν να απλοποιήσει και να κάνει πιο γρήγορη την δημιουργία και την ανάπτυξη διαδραστικών εφαρμογών. Όπως είχε πει και ο Josh Marinacci, μηχανικός λογισμικού στην Sun "JavaFX is sort of a code word for reinventing client Java and fixing the sins of the past" [8].

Στο παρόν στάδιο , το JavaFX 17.0.1 θεωρείται ως μια από τις κορυφαίες σύγχρονες πλατφόρμες για πλούσιες διαδραστικές εφαρμογές και επόμενος είναι αναπόσπαστο κομμάτι της Java, αφού διευκολύνει τόσο την ανάπτυξη αλλά και την εγκατάσταση των διαδραστικών εφαρμογών [9].

2.2.1.1 Αρχιτεκτονική JavaFX

Το ακόλουθο διάγραμμα δείχνει την αρχιτεκτονική του JavaFX API και τα συστατικά τα οποία υποστηρίζουν το JavaFX API.



Σχήμα 2.1 – Αρχιτεκτονική JavaFX [21]

Το JavaFX είναι κατασκευασμένο με μια αρχιτεκτονική επιπέδων [16] από συστατικά τα οποία στηρίζουν την διεπαφή του χρήστη. Όπως φαίνεται από το Σχήμα 2.1, το ανώτερο επίπεδο παρέχει ένα πλήρες σύνολο από APIs που διευκολύνουν την δημιουργία πλούσιων και όμορφων γραφικών διεπαφών.

Στο ανώτερο επίπεδο παρέχεται επίσης και το Scene Graph που είναι ένα ιεραρχικό δέντρο κόμβων που περιλαμβάνει όλα τα στοιχεία σε μια διεπαφή αλλά και τις σχέσεις των στοιχείων. Σε αυτό το γράφημα κάθε κόμβος έχει έναν μόνο γονέα και μηδέν ή περισσότερα παιδία. Επίσης, κάθε κόμβος έχει το δικό του “ID”, “Style Class”,”Opacity transformations” και τέλος το δικό του Event Handler. Στα πιο κάτω επίπεδα τα στοιχεία γραφικών (Prism και Quantum Toolkit), Glass, Web και Media παρέχουν βελτιώσεις που μειώνουν τον χρόνο εκτέλεσης.

Τέλος το Java Virtual Machine βρίσκεται στο κάτω άκρο της αρχιτεκτονικής. Το JVM παρέχει JRE για την εκτέλεση των εφαρμογών JavaFX και επίσης χειρίζεται διάφορες εργασίες όπως διαχείριση της στοίβας, φόρτωσης και αποθήκευσης μεταβλητών, διακλαδώσεις,

αριθμητικές πράξεις, επιστροφή μεθόδων, μετατροπές τύπων, διαχείριση εξαιρέσεων και συγχρονισμό [10].

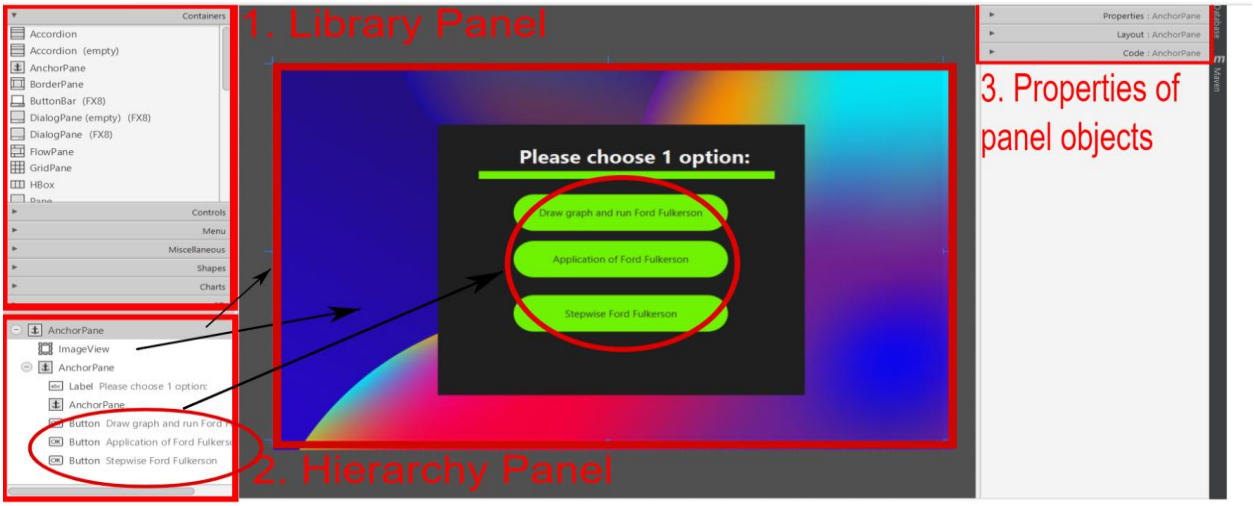
2.2.1.2 Χαρακτηριστικά του JavaFX

Το JavaFX έχει τα ακόλουθα χαρακτηριστικά:

- FXML και Scene Builder: Το JavaFX Scene Builder είναι ένα εργαλείο που σου επιτρέπει την σχεδίαση διεπαφών για JavaFX εφαρμογές γρήγορα και εύκολα κάνοντας μεταφορά και απόθεση των αντικειμένων - σχημάτων του JavaFX. Η τελική διεπαφή που προκύπτει είναι ένα ξεχωριστό FXML αρχείο το οποίο δεν έχει καμία σχέση με την λογική της εφαρμογής. Ακολουθεί επομένως την λογική MVC [17] και γίνεται διαχωρισμός της διεπαφής με την λογική της εφαρμογής.
- 3D Graphics Features: Το JavaFX πλέον υποστηρίζει σχήματα όπως 3D κουτιά, κυλίνδρους και σφαίρες.
- CSS: Χρησιμοποιείται για να βελτιώσει το εμφάνιση των JavaFX εφαρμογών.
- Canvas API: Το Canvas API επιτρέπει την σχεδίαση διεπαφής απευθείας πάνω στο JavaFX scene.
- Self-Contained Application Deployment Model: Δημιουργία αυτόνομων πακέτων τα οποία έχουν όλες τις απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή αλλά και ένα αντίγραφο του Java και JavaFX runtime environment.
- WebView: Είναι ένα εργαλείο διαδικτύου που χρησιμοποιά την WebKitHTML τεχνολογία για να κάνει δυνατή την ενσωμάτωση ιστοσελίδων μέσα σε μια JavaFX εφαρμογή. Επίσης το Javascript που τρέχει μέσα στο WebView μπορεί να καλέσει API της Java, και API της Java μπορεί να καλέσει JavaScript που τρέχει μέσα στο WebView.

2.2.2 Ανασκόπηση του JavaFX Scene Builder

Το JavaFX Scene Builder είναι ένα εργαλείο που παρέχει ένα περιβάλλον που μας επιτρέπει την σχεδίαση διεπαφών χρήστη για εφαρμογές JavaFX εύκολα και γρήγορα χωρίς την ανάγκη για ανάπτυξη κώδικα. Χρησιμοποιώντας αυτό το εργαλείο, καθώς δημιουργούμε την διεπαφή χρήστη, παράγεται αυτόματα και ο FXML κώδικας. Λόγω της απλής διαισθητικής διεπαφής του, το JavaFX Scene Builder επιτρέπει ακόμη και σε μη προγραμματιστές να δημιουργήσουν γρήγορα και εύκολα διαδραστικές εφαρμογές.



Σχήμα 2.2 – Scene Builder Overview

Στο Σχήμα 2.2 μπορεί να δει κάποιος πως φαίνεται το εργαλείο Scene Builder. Στο σημείο 1 μπορούμε να δούμε τα διάφορα συστατικά που υποστηρίζονται από το εργαλείο JavaFX και τα οποία μπορεί ο χρήστης να κάνει μεταφορά και απόθεση πάνω στη σκηνή. Στο σημείο 2, δηλαδή το hierarchy panel, μπορούμε να δούμε ποια συστατικά είναι ήδη πάνω στη σκηνή, στο πιο πάνω παράδειγμα έχουμε ήδη τοποθετημένα τα εξής components. Το AnchorPane είναι ένα “δοχείο” που μπορείς να τοποθετείς μέσα του και άλλα συστατικά. Το ImageView είναι ένα συστατικό που είναι ουσιαστικά ένα σημείο το οποίο σου κράτα χώρο για να τοποθετήσεις εκεί μια φωτογραφία για το φόντο. Τέλος μπορούμε να δούμε κυκλωμένα τα 3 κουμπιά τα οποία τοποθετήθηκαν στην σκηνή. Στο σημείο 3, φαίνονται διάφορες πληροφορίες και ιδιότητες για τα συστατικά που είναι ήδη τοποθετημένα πάνω

στην σκηνή. Για να δει κάποιος τις ιδιότητες ενός συστατικού πρέπει πρώτα να κάνει κλικ πάνω στο συστατικό που θέλει να δει τις ιδιότητες του.

Το εργαλείο Scene Builder στην συνέχεια θα παραγάγει τον πιο κάτω κώδικα FXML για την πιο πάνω διεπαφή αυτοματοποιημένα, όπως φαίνεται και στο πιο κάτω σχήμα.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="484.0" prefWidth="768.0" xmlns="http://javafx.com/javaf
   <children>
      <ImageView fitHeight="498.0" fitWidth="782.0">
         <image>
            <Image url="@../../../non_gradient.jpg" />
         </image>
      </ImageView>
      <AnchorPane layoutX="189.0" layoutY="78.0" prefHeight="349.0" prefWidth="421.0" style="-fx-background-color: #212121;" AnchorPane.bottomAnchor="57.0" AnchorPan
         <children>
            <Label alignment="TOP_CENTER" contentDisplay="CENTER" layoutX="64.0" layoutY="24.0" prefHeight="35.0" prefWidth="394.0" text="Please choose 1 option:" te
               <font>
                  <Font name="System Bold" size="24.0" />
               </font>
            </Label>
            <AnchorPane layoutX="47.0" layoutY="60.0" prefHeight="10.0" prefWidth="312.0" style="-fx-background-color: #76ff03;" AnchorPane.leftAnchor="47.0" AnchorP
            <Button fx:id="draw_graph_button" layoutX="87.0" layoutY="87.0" mnemonicParsing="false" prefHeight="46.0" prefWidth="218.0" style="-fx-background-color:
            <Button fx:id="application_button" layoutX="87.0" layoutY="154.0" mnemonicParsing="false" prefHeight="46.0" prefWidth="218.0" style="-fx-background-color
            <Button fx:id="stepwise_button" layoutX="87.0" layoutY="164.0" mnemonicParsing="false" prefHeight="46.0" prefWidth="218.0" style="-fx-background-color: #
         </children>
      </AnchorPane>
   </children>
```

Σχήμα 2.3 – Παραγόμενος FXML κώδικας

2.2.2.1 Ομάδα Στόχος

Οι ομάδες στόχοι του JavaFX Scene Builder είναι οι πιο κάτω:

- Java Developers: Τους δίνει την δυνατότητα να δημιουργήσουν γρήγορα ένα πρωτότυπο της εφαρμογής και να υλοποιήσουν την λογική της εφαρμογής ξεχωριστά από τη διεπαφή.
- Σχεδιαστές: Τους επιτρέπει να δημιουργήσουν πρωτότυπα εφαρμογών χωρίς να χρειάζεται να έχουν οποιαδήποτε προηγούμενη προγραμματιστική γνώση. Μπορούν επίσης να εμπλουτίσουν όσο θέλουν τις εφαρμογές τους χρησιμοποιώντας CSS παρέχοντας τους με αυτό τον τρόπο ένα ολοκληρωμένο εργαλείο για την δημιουργία πρωτοτύπων.

2.2.2.2 Κύρια Χαρακτηριστικά

Το JavaFX Scene Builder συμπεριλαμβάνει τα ακόλουθα χαρακτηριστικά:

- Μια διεπαφή που σου δίνει τη δυνατότητα για μεταφορά και απόθεση στοιχείων πάνω στη σκηνή επιτρέποντας σου επομένως να δημιουργήσεις γρήγορα και εύκολα μια γραφική διεπαφή χωρίς την ανάγκη ανάπτυξης κώδικα. Επιπρόσθετα ο χρήστης μπορεί να προσθέσει, να συνδυάσει και να αλλάξει τα JavaFX GUI κουμπιά μέσω του JavaFX Scene Builder.
- Εύκολη ενσωμάτωση του εργαλείου Scene Builder με οποιοδήποτε Java IDE αφού το Scene Builder είναι ένα αυτόνομο εργαλείο ανάπτυξης.
- Αυτόματη παραγωγή FXML κώδικα
- Προσθήκη δικών μας GUI αντικειμένων: Επιτρέπει στον χρήστη να δημιουργήσει και να τροποποιήσει διάφορα GUI αντικείμενα.
- CSS: Υποστήριξη CSS χρησιμοποιώντας το εργαλείο Scene Builder

2.2.3 Ανασκόπηση του εργαλείου Maven

Το Maven είναι ένα εργαλείο διαχείρισης έργων λογισμικού που χρησιμοποιείται κυρίως για έργα που βασίζονται σε Java. Χρησιμοποιείται όμως και για τη διαχείριση έργων σε άλλες γλώσσες προγραμματισμού όπως C# , Scala και Ruby. Το Maven επιτρέπει στον προγραμματιστή να αυτοματοποιήσει τον χειρισμό της δημιουργίας της αρχικής μορφής φακέλου αλλά επίσης και να μειώσει σημαντικά τον αριθμό βημάτων που χρειάζονται για να χτίσει κάποιος το πρόγραμμα του, απλοποιώντας με αυτό τον τρόπο την διαδικασία σε μεγάλο βαθμό.

2.2.3.1 Στόχοι του Εργαλείου Maven

Ο κυριότερος στόχος του Maven είναι να επιτρέψει στον προγραμματιστή να χτίσει το πρόγραμμα του με όλες τις απαραίτητες εξαρτήσεις εύκολα και γρήγορα. Για να το πετύχει αυτό, το Maven αυτοματοποιεί τις πιο κάτω διαδικασίες [18]:

- Αυτοματοποίηση του χτισίματος ενός προγράμματος.
- Δίνει ένα ομοιόμορφο τρόπο κατασκευής για διαφορετικά προγράμματα, αφού όλα τα προγράμματα που χτίζονται με Maven περιγράφονται από ένα είδος αρχείου τύπου POM.
- Ενθαρρύνει την χρησιμοποίηση καλύτερων τακτικών προγραμματισμού.
- Διαχείριση εξαρτήσεων συμπεριλαμβανομένης και αυτόματης ενημέρωσης των πακέτων.

2.2.4 Αναγκαία Επιπλέον Μελέτη

Για την εκμάθηση του εργαλείου JavaFX ήταν απαραίτητη η παρακολούθηση εκπαιδευτικού υλικού τόσο από το εγχειρίδιο χρήσης [3] ,από βιβλία [4] αλλά και διάφορες πηγές στο διαδίκτυο.

- JavaFX Tutorials [12]:
    - Creating a Basic Window
    - Handle User Events
    - Switching Scenes
    - Creating Alert Boxes
    - Communicating Between Windows
    - GridPane
    - Rotating 3D Objects with Keyboard Input
    - Controlling objects with Mouse
    - Zooming with Mouse Scroll

- Mastering JavaFX 10 - Build Advanced and Visually Stunning Java Applications by Sergey Grinev, (Εξήγα σε περισσότερο βάθος τις θεωρητικές έννοιες της JavaFX) [13]

Τα πιο πάνω βοηθήματα ήταν πολύ σημαντικά έτσι ώστε να μπορέσω να υλοποιήσω μια διαδραστική εφαρμογή όπου ο χρήστης θα μπορεί να σχεδιάζει τα δικά του δίκτυα ροής και να τρέχει τον αλγόριθμο Ford Fulkerson πάνω τους.

Κεφάλαιο 3

Περιγραφή Αλγορίθμου Ford Fulkerson και Ανάλυση Απαιτήσεων

3.1. Πρόβλημα Μέγιστης Ροής και Αλγόριθμος Ford Fulkerson

Τα γραφήματα είναι μια από τις πιο σημαντικές δομές στην Επιστήμη των Υπολογιστών και χρησιμοποιούνται για να μοντελοποιήσουν διάφορα προβλήματα. Η διπλωματική αυτή μελετάει το πρόβλημα της Μέγιστης Ροής [1,2]. Το πρόβλημα αυτό ασχολείται με την εύρεση της μέγιστης εφικτής ροής που μπορεί να περάσει μέσω ενός δικτύου ροής χρησιμοποιώντας τον αλγόριθμο Ford Fulkerson. Ένας πιο απλοϊκός ορισμός του πιο πάνω θα μπορούσε να είναι ο εξής : "Έστω ένα δίκτυο μεταφοράς υγρών όπου οι ακμές είναι σωληνώσεις και οι κόμβοι είναι συνδέσεις μεταξύ των σωληνώσεων. Ποια είναι η μέγιστη

ποσότητα νερού που μπορεί να κατευθυνθεί από ένα δεδομένο κόμβο προέλευσης σε ένα δεδομένο κόμβο απόληξης [2];

3.1.1. Περιγραφή

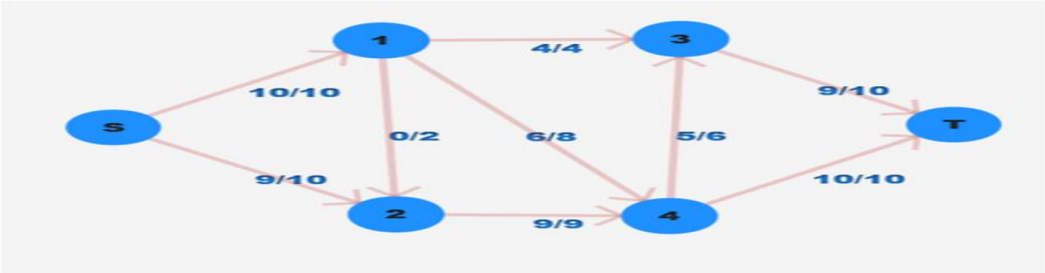
Για να εξηγήσουμε τον αλγόριθμο Ford Fulkerson θα εξηγήσουμε τρεις πολύ σημαντικές έννοιες στις οποίες στηρίζεται ο αλγόριθμος αυτός, οι οποίες είναι τα Δίκτυα Ροής, το Υπολειπόμενο Δίκτυο και η διαδρομή επαύξησης σε ένα υπολειπόμενο δίκτυο.

3.1.1.1 Δίκτυο Ροής

Θα εξετάσουμε γραφήματα που ονομάζονται δίκτυα ροής, όπου ροή σε αυτά τα γραφήματα θεωρείται μια αφηρημένη οντότητα που παράγεται στους κόμβους προέλευσης, μεταφέρεται μέσω των ακμών του δικτύου ροής και καταλήγει στους κόμβους απόληξης. Πάμε όμως να δούμε κάποιους πιο τυπικούς ορισμούς για τη ροή και το πρόβλημα της Μέγιστης Ροής δεδομένου ενός δικτύου ροής.

Ένα *δίκτυο ροής* ορίζεται ως ένα κατευθυνόμενο γράφημα G = (V , E) με τα ακόλουθα χαρακτηριστικά:

- Κάθε ακμή *e* έχει μια χωρητικότητα, η οποία είναι είτε ένας θετικός ακέραιος, είτε ένας πραγματικός αριθμός και την οποία συμβολίζουμε ως c_e. Αυτός ο αριθμός συμβολίζει ποια είναι η μέγιστη ροή που μπορεί να μεταφερθεί μέσω της ακμής *e*.
- Κάθε ακμή *e* έχει μια ροή, η οποία συσχετίζεται με ένα θετικό ακέραιο ή πραγματικό αριθμό και την οποία συμβολίζουμε ως f_e. Αυτός ο αριθμός δείχνει πόση ροή μπορεί να πέρνα μέσω αυτής της ακμής.
- Υπάρχει μόνο ένας κόμβος προέλευσης s, όπου $s \in V$
- Υπάρχει μόνο ένας κόμβος απόληξης t, όπου $t \in V$



Σχήμα 3.1 – Παράδειγμα δικτύου ροής

Στο Σχήμα 3.1 μπορούμε να δούμε ένα παράδειγμα ενός δικτύου ροής. Οι κόμβοι είναι αριθμημένοι και οι κόμβοι προέλευσης και απόληξης έχουν σαν όνομα το “S” και το “T”. Επιπρόσθετα, οι ακμές σχετίζονται με ροή/χωρητικότητα, για παράδειγμα η ακμή e=(1,4) έχει f_e=6 και χωρητικότητα c_e= 8.

Αφού είδαμε τι είναι ένα δίκτυο ροής πάμε τώρα να δούμε τι είναι ένα δίκτυο ροής και ποιο είναι το πρόβλημα της μέγιστης ροής :

Ροή:

Μια συνάρτηση $f:E \rightarrow R^+$ που αντιστοιχίζει κάθε ακμή σε ένα μη αρνητικό, πραγματικό αριθμό έτσι ώστε να ισχύουν πάντα πιο κάτω συνθήκες:

i. Συνθήκη χωρητικότητας: $\forall e \in E, 0 <= f(e) <= c_e$.
ii. Συνθήκη διατήρησης: $\forall v \in V \backslash \{s,t\}$,

$$\sum_{e \text{ προς την } v} f(e) = \sum_{e \text{ από την } v} f(e)$$

iii. Τιμή ροής στο Δίκτυο Ροής (Δ_F): $|f| = \sum_{e \text{ από την } s} f(e)$

Πρόβλημα Μέγιστης Ροής Δεδομένου ενός δικτύου ροής:

Με δεδομένο ένα δίκτυο ροής, βρείτε μια ροή με τη μέγιστη δυνατή τιμή.

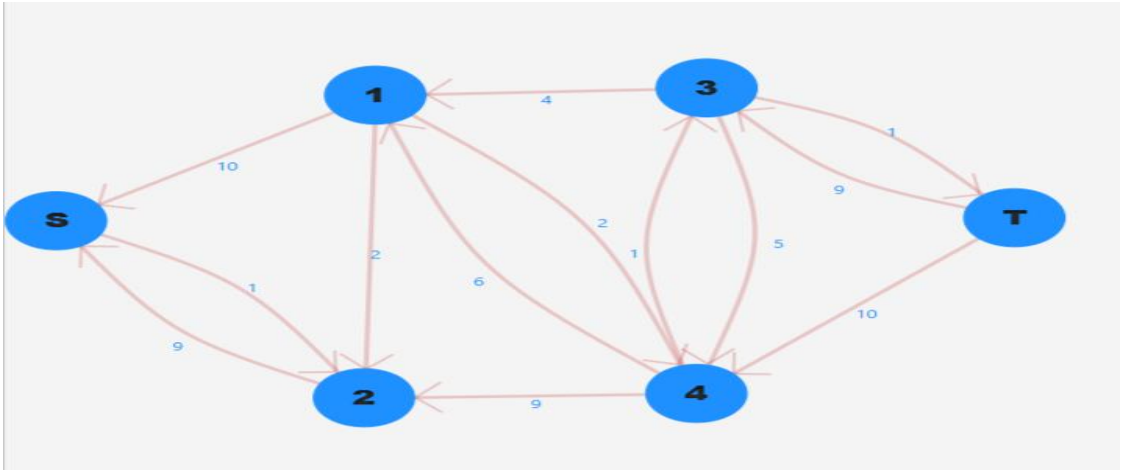
Παρατήρηση: Η μέγιστη ροή στο δίκτυο ροής δεν ξεπερνά πότε το άθροισμα των χωρητικοτήτων των ακμών που εξέρχονται από την s. Δηλαδή ισχύει πάντοτε ότι

$$|f| \leq \sum_{e \text{ από την } s} Χωρητικότητα(e)$$

3.1.1.2 Υπολειπόμενο Δίκτυο

Ακόμη μια σημαντική έννοια στον αλγόριθμο Ford Fulkerson είναι το υπολειπόμενο δίκτυο. Το υπολειπόμενο δίκτυο G_f ορίζεται ως εξής:

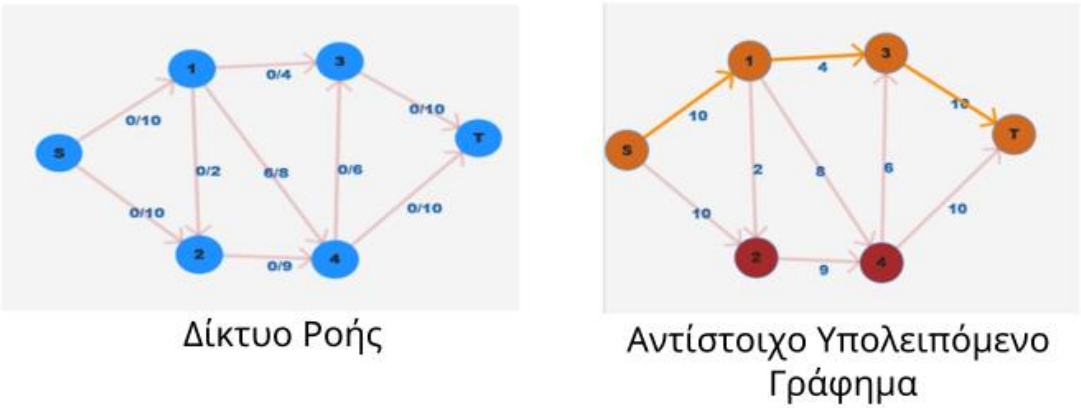
1. Το σύνολο των κόμβων στο υπολειπόμενο δίκτυο (G_f) είναι το ίδιο με το σύνολο των κόμβων του Δικτύου Ροής (G).
2. Για κάθε ακμή e= (u, v) μέσα στο Δίκτυο ροής G και έστω f_e η ροή αυτής της ακμής όπου $f_e < c_e$, προσθέτουμε στο G_f (υπολειπόμενο δίκτυο) μια *ευθύδρομη* ακμή e=(u , v) με χωρητικότητα c_e-f_e.
3. Για κάθε ακμή e = (u, v) μέσα στο Δίκτυο ροής G και έστω f_e η ροή της ακμής, όπου $f_e > 0$, προσθέτουμε στο G_f (υπολειπόμενο δίκτυο) μια *ανάδρομη* ακμή e=(v , u) με χωρητικότητα f_e.



Σχήμα 3.2: Υπολειπόμενο δίκτυο του Σχήματος 3.1

3.1.1.3 Διαδρομή Επαύξησης σε Υπολειπόμενο Δίκτυο

Αφού εξηγήσαμε τι είναι ένα δίκτυο ροής και τι είναι ένα υπολειπόμενο δίκτυο, μπορούμε πλέον να εξηγήσουμε με ακρίβεια πως προωθούμε ροή από τη πηγή (κόμβο “s”), προς την απόληξη (κόμβο “t”). Έστω ότι η P είναι μια απλή διαδρομή που αρχίζει από τη πηγή και καταλήγει στην απόληξη στο υπολειπόμενο δίκτυο G_f. Απλή διαδρομή σημαίνει ότι η διαδρομή P δεν επισκέπτεται κανένα κόμβο περισσότερες από μία φορές, δηλαδή δεν υπάρχουν κύκλοι(η διαδρομή αυτή επιλέγεται αυθαίρετα στην απλή εκδοχή του αλγορίθμου -- στο Κεφάλαιο 3.1.2 θα μελετηθούν οι βέλτιστες τεχνικές εύρεσης διαδρομής επαύξησης). Ορίζουμε ως *σημείο συμφόρησης*, την ελάχιστη υπολειπόμενη χωρητικότητα οποιασδήποτε ακμής της διαδρομής P, πάνω στο G_f.



Σχήμα 3.3: Ένα δίκτυο ροής και το αντίστοιχο υπολειπόμενο δίκτυο

Όπως φαίνεται και από το πιο πάνω σχήμα, η τονισμένη διαδρομή (με πορτοκαλί χρώμα) είναι μια *διαδρομή επαύξησης* στο πιο πάνω υπολειπόμενο δίκτυο (Residual Graph). Η διαδρομή επαύξησης είναι η εξής:

Διαδρομή Επαύξησης = {Ακμή(S,1), Ακμή(1,3), Ακμή(3,T)}

Σε αυτές ακριβώς τις ακμές θα αυξηθεί η ροή στο δίκτυο ροής. Το σημείο συμφόρησης της επιλεγμένης διαδρομής είναι το πιο κάτω :

Συμφόρηση = min {Χωρητικότητα(S,1), Χωρητικότητα (1,3), Χωρητικότητα (3,T)} = 4

3.1.1.4 Διαδικασία Επαύξησης σε Υπολειπόμενο Δίκτυο

Αφού εξηγήσαμε τι είναι η διαδρομή επαύξησης πάμε να εξηγήσουμε πως την χρησιμοποιούμε για να αυξηθεί η ροή στο δίκτυο ροής. Η διαδικασία επαύξησης βασίζεται στον πιο κάτω ψευδοκώδικα

Ψευδοκώδικας Διαδικασίας Επαύξησης [1]:

```
augment (f,P)
  Έστω ότι b=bottleneck (P, f)
  For όλες τις ακμές (u,v) που ανήκουν στην P
    If e=(u,v) είναι ευθύδρομη ακμή then
      αύξησε το f(e) στο G κατά b
    Else (το (u,v) είναι ανάδρομη ακμή, και έστω e=(v,u))
      μείωσε το f(e) στο G κατά b
    End If
  End For
  Return(f)
```

Με λόγια, αυτό που κάνει ο πιο πάνω ψευδοκώδικας είναι ότι για κάθε ακμή πάνω στην διαδρομή επαύξησης ελέγχει αν αυτή η ακμή είναι ευθύδρομη, δηλαδή αν υπάρχει η αντίστοιχη ακμή στο δίκτυο ροής, αν είναι *ευθύδρομη* τότε αυξάνεται η ροή της κατά "Bottleneck" μονάδες. Αν είναι *ανάδρομη* ακμή, δηλαδή υπάρχει η αντίστοιχη ακμή αλλά η φορά της είναι αντίθετη, ή με πιο απλά λόγια αντί να υπάρχει η ακμή e=(u,v) στο δίκτυο ροής, υπάρχει η e=(v,u). Σε αυτή την περίπτωση μειώνουμε την ροή της ακμής e κατά "Bottleneck" μονάδες.

3.1.1.5 Αλγόριθμος Ford Fulkerson:

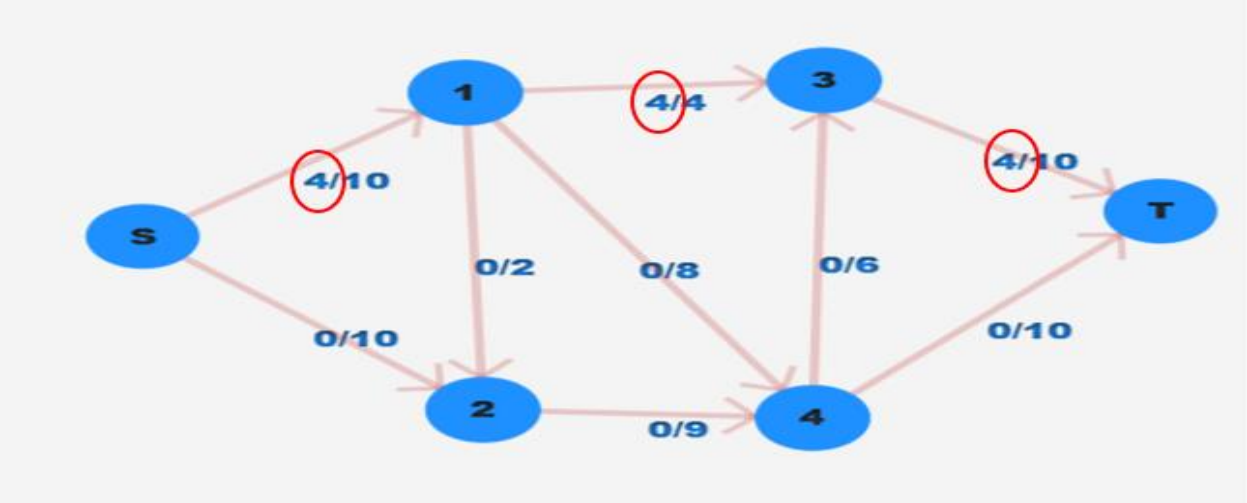
Αφού εξηγήσαμε την διαδικασία επαύξησης πάμε να εξηγήσουμε πως χρησιμοποιείται από τον αλγόριθμο Ford Fulkerson για την εύρεση της μέγιστης ροής. Ο αλγόριθμος Ford Fulkerson βασίζεται στον πιο κάτω ψευδοκώδικα.

Ψευδοκώδικας Ford Fulkerson [1]:

```
Ford Fulkerson
  Αρχικά f(e)=0 για όλα τα e του G
  While υπάρχει μια διαδρομή s-t στο υπολειπόμενο γράφημα Gf
    Έστω ότι η P είναι μια απλή διαδρομή s-t στο Gf
    f ' = augment(f, P)
    Ενημέρωσε το f σε f '
    Ενημέρωσε το υπολειπόμενο γράφημα Gf σε Gf '
  End While
  Return f
```

Όπως μπορούμε να δούμε η βασική ιδέα της μεθόδου Ford Fulkerson είναι η συνεχής εύρεση μιας διαδρομής επαύξησης πάνω στο υπολειπόμενο δίκτυο. Στη συνέχεια αυξάνεται η ροή πάνω στο δίκτυο ροής χρησιμοποιώντας την διαδικασία επαύξησης. Μετά, ενημερώνεται το υπολειπόμενο δίκτυο ανάλογα με τις αλλαγές που έγιναν. Τέλος, η διαδικασία αυτή εκτελείται επαναληπτικά μέχρι να μην υπάρχει άλλη διαδρομή πάνω στο υπολειπόμενο δίκτυο.

Στο πιο κάτω σχήμα μπορούμε να δούμε το αποτέλεσμα του τρεξίματος της πρώτης επανάληψης του αλγόριθμου Ford Fulkerson. Όπως μπορούμε να δούμε οι ακμές που ανήκουν στο μονοπάτι επαύξησης που περιγράψαμε στην Κεφάλαιο 3.1.1.4 είναι *ευθύδρομες* και επομένως θα αυξηθεί η ροή τους κατά "Στένωση" μονάδες. (όπου στην συγκεκριμένη περίπτωση Στένωση=4).



Σχήμα 3.4: Αύξηση ροής στο δίκτυο ροής μετά την διαδικασία επαύξησης που είδαμε στο Σχήμα 3.3

Θεωρητική Ανάλυση – Τερματισμός :

Η πολυπλοκότητα του αλγορίθμου για να επιλύσει το πρόβλημα Εύρεσης Μέγιστης Ροής σε ένα Δίκτυο Ροής είναι *O(m*C)* όπου m = |E| (αριθμός ακμών του Δικτύου Ροής) και C = $\sum_{e \text{ από την } s} Χωρητικότητα(e)$.

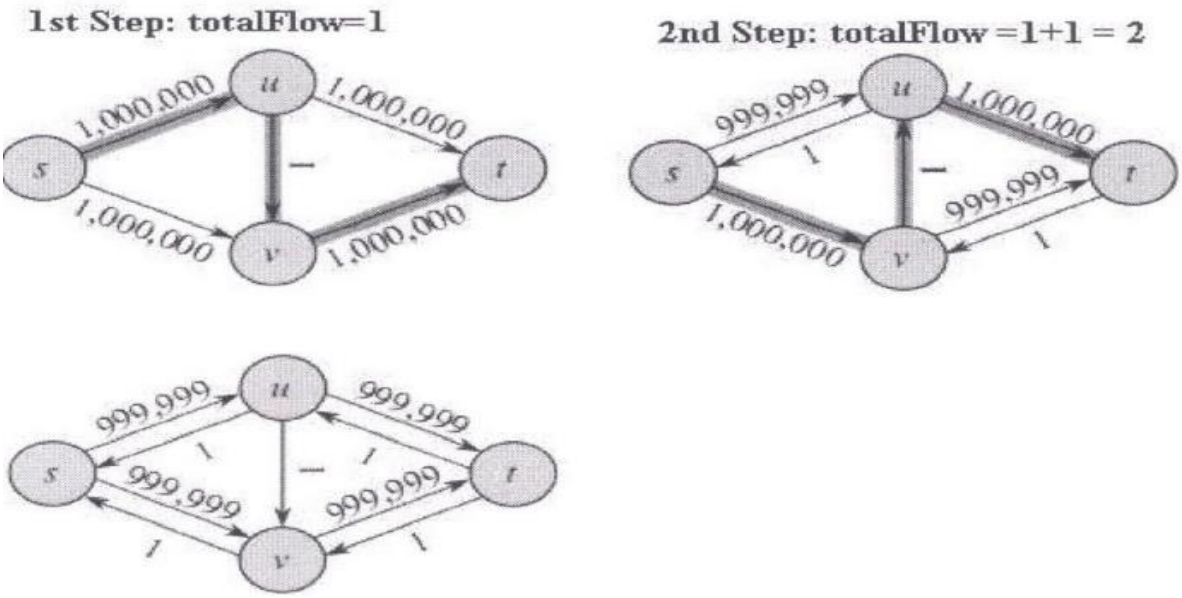
3.1.2. Ford Fulkerson κατά Edmonds Karp [1,2]

Ο αλγόριθμος Edmonds Karp χρησιμοποιεί παρόμοια τεχνική με τον αλγόριθμο Ford Fulkerson, όπου δημιουργεί διαδρομές επαύξησης σε ένα υπολειπόμενο δίκτυο έτσι ώστε να σταλεί η μέγιστη δυνατή ροή από την πηγή στην απόληξη. Η εκδοχή αυτή του αλγορίθμου διαφέρει σε ένα σημείο, στον τρόπο επιλογής του μονοπατιού επαύξησης. Ενώ ο αλγόριθμος που περιγράψαμε προηγουμένως επέλεγε τα μονοπάτια επαύξησης με ένα αυθαίρετο τρόπο, αυτή η εκδοχή του αλγορίθμου επιλέγει πρώτα τις μικρότερες διαδρομές που υπάρχουν στο Δίκτυο Ροής. Η εύρεση τέτοιων διαδρομών επιτυγχάνεται χρησιμοποιώντας αλγόριθμους όπως τον Breadth First Search ή Depth First Search [7].

Θεωρητική Ανάλυση – Τερματισμός :

Η πολυπλοκότητα του αλγορίθμου για να επιλύσει το πρόβλημα Εύρεσης Μέγιστης Ροής σε ένα Δίκτυο Ροής είναι $O(m^2*N)$ όπου m = |E| (αριθμός ακμών του Δικτύου Ροής) και N = |V| (αριθμός κόμβων του Δικτύου Ροής).

Όπως βλέπουμε οι χρόνοι πολυπλοκότητας των δυο εκδοχών διαφέρουν καθώς η πολυπλοκότητα εξαρτάται από τον τρόπο με τον οποίο επιλέγετε η διαδρομή επαύξησης. Εάν αυτή η διαδρομή επιλεγεί άπληστα τότε ο αλγόριθμος ενδέχεται να μην τερματίσει. Ακολουθεί το πιο κάτω παράδειγμα.



Σχήμα 3.5: Παράδειγμα δικτύου ροής όπου ο απλοϊκός αλγόριθμος είναι πολύ αργός [2]

Από το Σχήμα 3.5 μπορούμε να δούμε πως αν το μονοπάτι επιλέγεται άπληστα υπάρχει η πιθανότητα να επιλέγονται τα μονοπάτια s->u->v->t ή s->v->u->t. Αν επιλέγονται μόνο αυτά όμως τότε θα έχουμε 2.000.000 διαδρομές. Αν όμως επιλεχτούν τα μονοπάτια s->u->t στο 1ο βήμα και s->->t στο 2ο βήμα, τότε παίρνουμε τη μέγιστη ροή σε 2 μόνο διαδρομές!

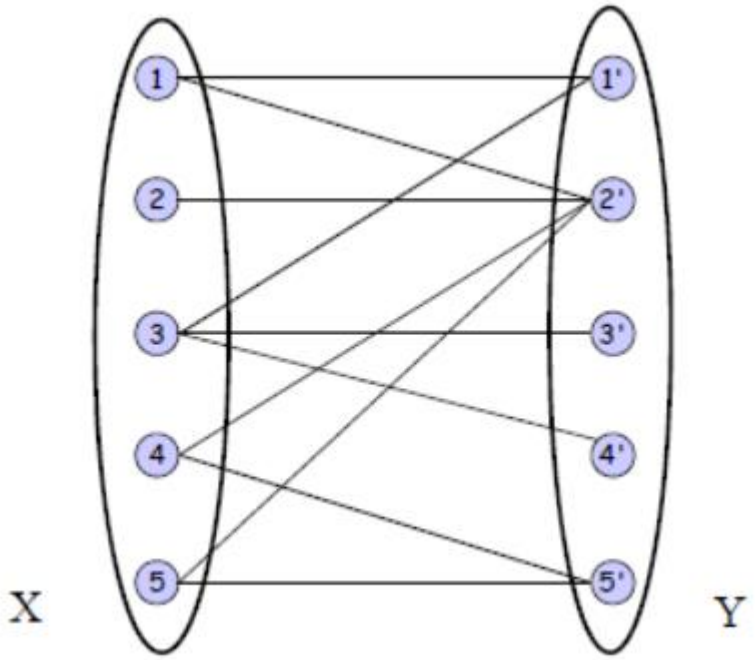
3.1.3. Εφαρμογές Αλγορίθμου

Το πρόβλημα Εύρεσης Μέγιστης Ροής σε ένα γράφο δεν είναι μόνο ένα θεωρητικό πρόβλημα αλλά έχει πολλές εφαρμογές στην πραγματική ζωή. Πολλοί επιστήμονες έχουν ασχοληθεί με την κατασκευή αλγορίθμων επίλυσης αυτού του προβλήματος για αυτό ακριβώς τον λόγο. Η επίλυση αυτού του προβλήματος σε μικρό χρόνο είναι δυνατό να αποτελέσει την λύση για μεγάλου μήκους προβλήματα. Αρχικά μια γενική εφαρμογή του αλγορίθμου είναι η αντιστοίχιση οντοτήτων. Ακόμη ένα πρόβλημα στο οποίο βρίσκει εφαρμογή ο αλγόριθμος είναι ο χρονοπρογραμματισμός αεροπορικών δρομολογίων [20]. Όπως όλοι μπορούμε να αντιληφθούμε αυτό είναι ένα πρόβλημα που όχι μόνο απαιτεί μια σωστή λύση αλλά απαιτεί και μια γρήγορη απάντηση. Μια άλλη σημαντική εφαρμογή είναι η διοχέτευση νερού από κάποια πηγή σε ολόκληρη την χώρα με τον καλύτερο εφικτό τρόπο [19], ώστε να μην γίνεται σπατάλη του νερού αλλά και να φτάνει το νερό στον προορισμό του που είναι στην συγκεκριμένη περίπτωση οι διάφορες για παράδειγμα πόλεις. Ακόμα η επίλυση του προβλήματος Εύρεσης Μέγιστης Ροής σε ένα Δίκτυο Ροής μπορεί να δώσει λύση στα προβλήματα των Διμερών Ταιριασμάτων, Δίκτυα Μεταφοράς, Συνδεσιμότητα Δικτύων κ.α. Βλέπουμε λοιπόν ότι το πρόβλημα Εύρεσης Μέγιστης Ροής σε ένα δίκτυο είναι ένα αρκετά διαδεδομένο πρόβλημα σε πολλούς τομείς της ζωής μας χωρίς εμείς να το αντιλαμβανόμαστε.

Επομένως στα πλαίσια της διπλωματικής υλοποιήθηκε και μια εφαρμογή αντιστοίχισης οντοτήτων όπου ο χρήστης μπορεί να δημιουργήσει εύκολα και γρήγορα δικές του αντιστοιχήσεις (Διμερή Ταιριάσματα) για οποιοδήποτε πρόβλημα το οποίο μπορεί να αναχθεί στο πρόβλημα του Διμερούς Ταιριάσματος. Με αυτή την επιλογή θα σχεδιάζεται αυτόματα ένας διμερής γράφος με το σύνολο των οντοτήτων που ο χρήστης επιθυμεί και στη συνέχεια θα επιλύνεται αυτό το πρόβλημα με τον αλγόριθμο Ford Fulkerson.

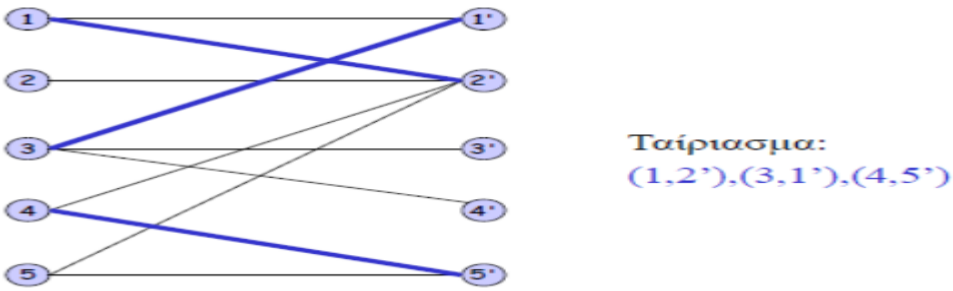
Ακολουθούν κάποιοι σημαντικοί ορισμοί οι οποίοι θα βοηθήσουν στην καλύτερη κατανόηση για το τι υλοποιήθηκε στο Κεφάλαιο 6 και γιατί.

Διμερής Γράφος [1,2]: Ένας μη κατευθυνόμενος γράφος G=(V,E) με την ιδιότητα ότι το V μπορεί να διαμεριστεί ως V=XUY έτσι ώστε για κάθε ακμή (u,v) ∈ E: u ∈ X και v ∈ Y.



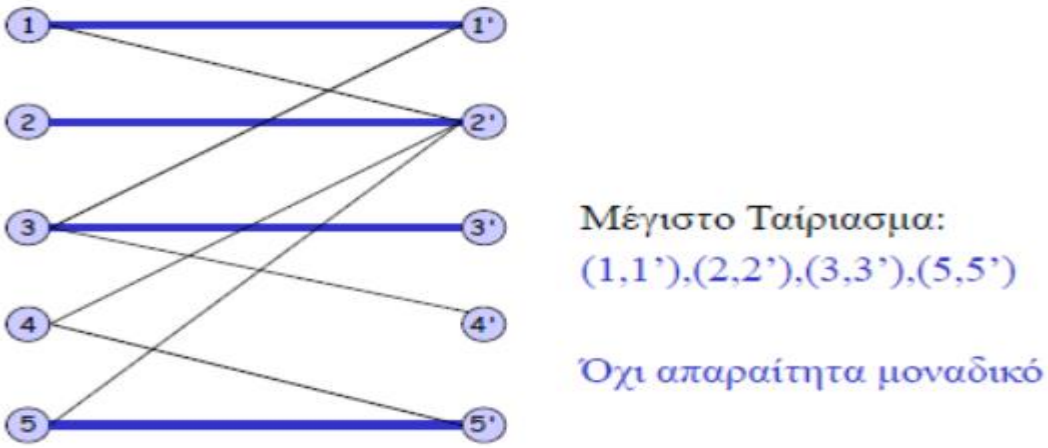
Σχήμα 3.6: Παράδειγμα Διμερή Γράφου [1]

Ταίριασμα [1]: Ένα υποσύνολο M ⊆ E έτσι ώστε κάθε κορυφή u ∈ V να εμφανίζεται το πολύ σε μια ακμή του M. Με πιο απλά λόγια ένα ταίριασμα είναι ένα σύνολο με ξένες ακμές μεταξύ τους.



Σχήμα 3.7: Παράδειγμα Ταιριάσματος [1]

<u>Πρόβλημα του Διμερούς Ταιριάσματος [1]:</u> Για ένα Διμερή Γράφημα, να βρεθεί ένα ταίριασμα στο G με το μέγιστος δυνατό μέγεθος



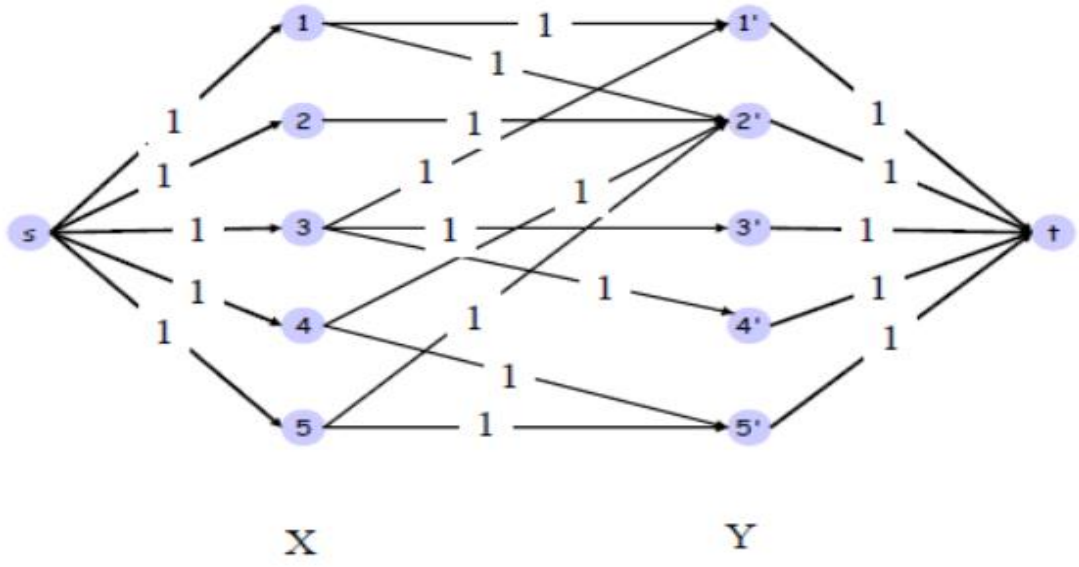
Σχήμα 3.8: Παράδειγμα μέγιστου Διμερούς Ταιριάσματος [1]

Πάμε τώρα να δείξουμε πως μπορεί να αναχθεί το πρόβλημα του Διμερούς Ταιριάσματος στο πρόβλημα της μέγιστης ροής.

<u>Αναγωγή στο Πρόβλημα της Μέγιστης Ροής [1]:</u>

- Ξεκινούμε από μια είσοδο G = (XUY, E) του Διμερούς Ταιριάσματος και κατασκευάζουμε από αυτήν ένα δίκτυο ροής G΄ ως εξής:
    - Κατευθύνουμε όλες τις ακμές του G από το X προς το Y (Δηλαδή τις μετατρέπει σε κατευθυνόμενες ακμές).
    - Προσθέτουμε μια κορυφή s (πηγή) και ακμές (s,x) προς όλες τις κορυφές x ∈ X
    - Προσθέτουμε μια κορυφή t (προορισμός) και ακμές (y,t) προς όλες τις κορυφές y ∈ Y
    - Αποδίδουμε χωρητικότητα 1 σε κάθε ακμή.

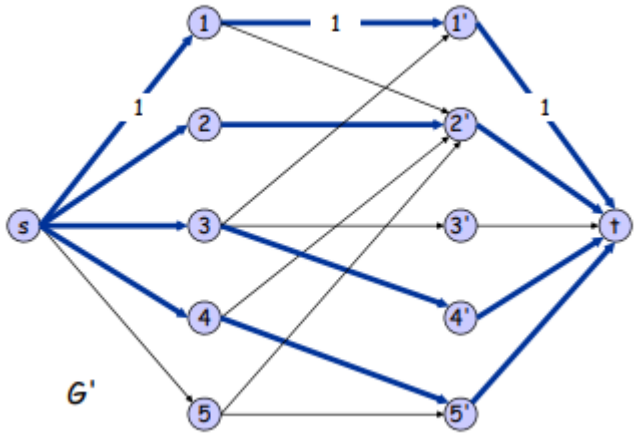
Παράδειγμα Αναγωγής:



Σχήμα 3.9: Παράδειγμα Αναγωγής του Διμερή Γράφου του σχήματος 3.6 [1]

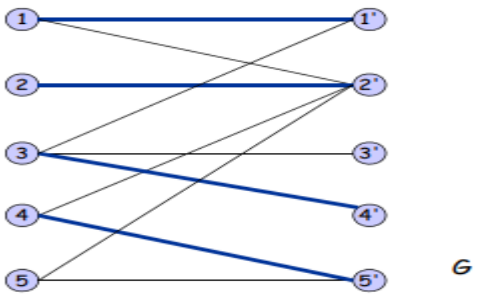
Όπως μπορούμε να δούμε προστέθηκαν 2 νέοι κόμβοι, η πηγή και η απόληξη όπου συνδέθηκαν με όλους τους κόμβους του X και Y συνόλου αντίστοιχα. Επίσης μπορούμε να δούμε ότι οι ακμές είναι πλέον κατευθυνόμενες σε αντίθεση με πριν.

Αφού δείξαμε πως ανάγουμε το διμερή γράφο, έστω G, σε ένα δίκτυο ροής, έστω G', πάμε να δούμε πως η εύρεση της μέγιστης ροής, μας επιλύει το πρόβλημα του διμερούς ταιριάσματος, δηλαδή μας δίνει ένα μέγιστο ταίριασμα. Όπως γνωρίζουμε ο αλγόριθμος Ford Fulkerson βρίσκει τη μέγιστη ροή σε ένα δίκτυο ροής. Τρέχοντας επομένως τον αλγόριθμο πάνω στο δίκτυο ροής του σχήματος 3.9 προκύπτει η πιο κάτω ροή.



Σχήμα 3.10: Ροή που προκύπτει από τρέξιμο Ford Fulkerson στο δίκτυο ροής του σχήματος 3.9 [1]

Όπως μπορούμε να δούμε οι ακμές που έχουν ροή μεταξύ του συνόλου X και Y είναι οι ακμές (1,1'), (2,2'), (3,4'), (4,5'). Αυτές οι ακμές μεταξύ του συνόλου X και Y που έχουν ροή ένα, μας δίνουν και την λύση στο πρόβλημα του διμερούς ταιριάσματος, δηλαδή αυτό το σύνολο των ακμών αποτελεί και το μέγιστο ταίριασμα.



Σχήμα 3.11: Μέγιστο ταίριασμα σε διμερή γράφο που προκύπτει από την εύρεση μέγιστης ροής στο γράφο του σχήματος 3.10

Κεφάλαιο 4

Κύρια Διεπαφή Προγράμματος

4.1. Περιγραφή Τρόπου Υλοποίησης

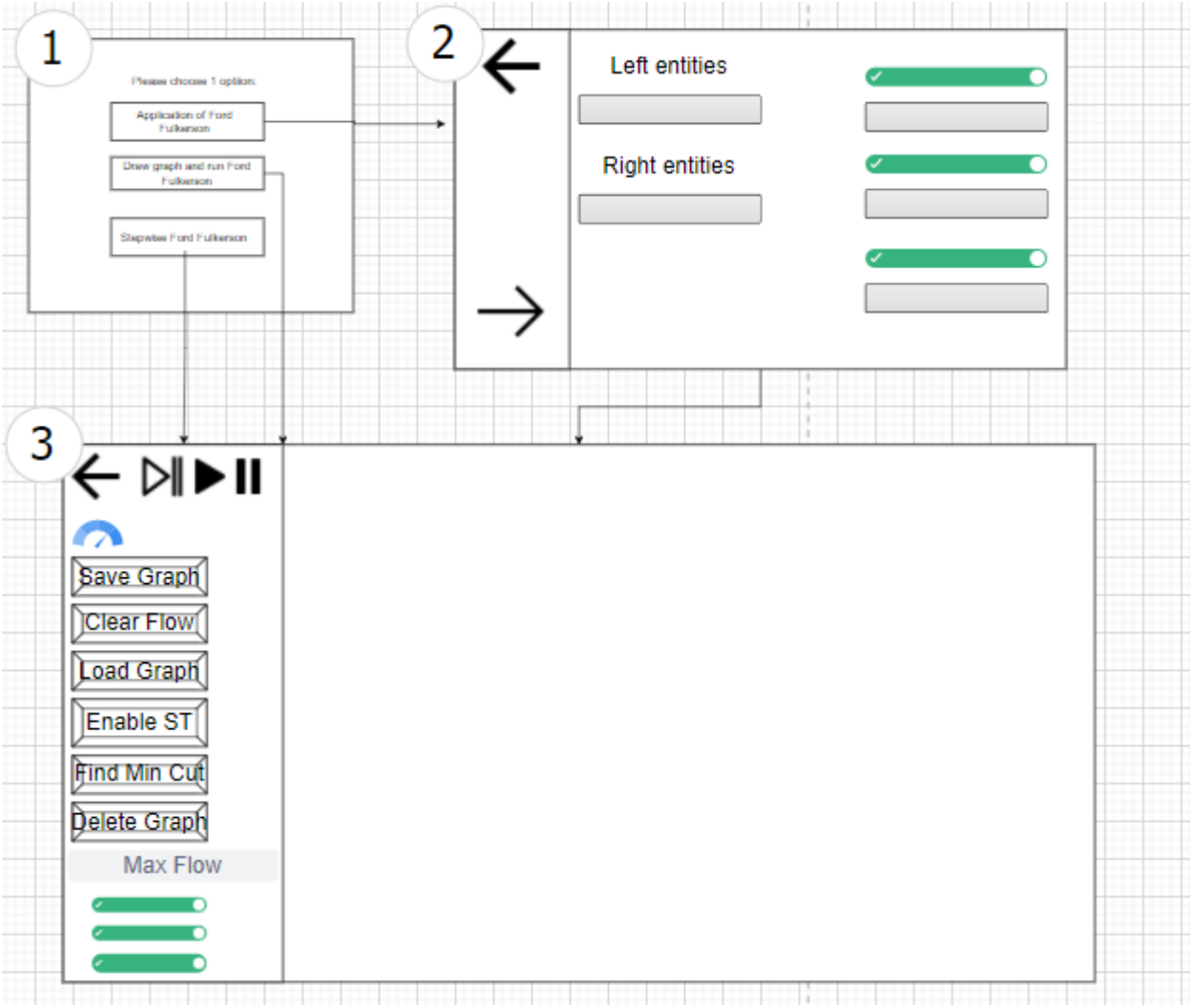
Για την δημιουργία των διεπαφών του προγράμματος, το εργαλείο JavaFX προσφέρει αρκετά βοηθητικά και εξειδικευμένα εργαλεία που μας δίνουν την δυνατότητα να σχεδιάζουμε και να υλοποιούμε εύκολα, γρήγορα και αποτελεσματικά την διεπαφή.

Αρχικά η πρώτη μορφή των διεπαφών σχεδιαζόταν σε ένα εργαλείο δημιουργίας πρωτοτύπων, πιο συγκεκριμένα το εργαλείο draw.io [14], και άλλαζε συνεχώς μέχρι να βρεθεί η πιο εύχρηστη και φιλική προς τον χρήστη διεπαφή για τους σκοπούς του προγράμματος. Αφού φτάναμε σε μια διεπαφή με την οποία νιώθαμε ότι ο χρήστης θα μπορούσε να αλληλοεπιδράσει εύκολα και ευχάριστα προχωρούσαμε στην σχεδίαση της με το εργαλείο Scene Builder το οποίο όπως εξηγήσαμε προηγουμένως παράγει FXML κώδικα ο οποίος μπορεί να χρησιμοποιηθεί από την JavaFX

Αφού υλοποιούνταν οι διεπαφές με το Scene Builder περνάμε ανατροφοδότηση από μια ομάδα ατόμων έτσι ώστε να πάρουμε εισηγήσεις και σχόλια για τις διεπαφές. Με αυτό τον τρόπο είμασταν σίγουροι ότι οι διεπαφές ήταν φιλικές και ευχάριστες προς τον χρήστη πρώτου προχωρήσουμε με την υλοποίηση της λογικής και της λειτουργικότητας των διεπαφών.

4.2. Σχεδιασμός και Υλοποίηση

Αρχικά ξεκινήσαμε σχεδιάζοντας ένα πρωτότυπο για το πως θα φαίνεται η εφαρμογή και πως θα μπορεί να αλληλοεπιδρά ο χρήστης μαζί της. Στο Σχήμα 4.1 μπορεί κάποιος να δει πως σχεδιάστηκε η εφαρμογή στα πρώτα στάδια της με τη βοήθεια του εργαλείου draw.io [14].

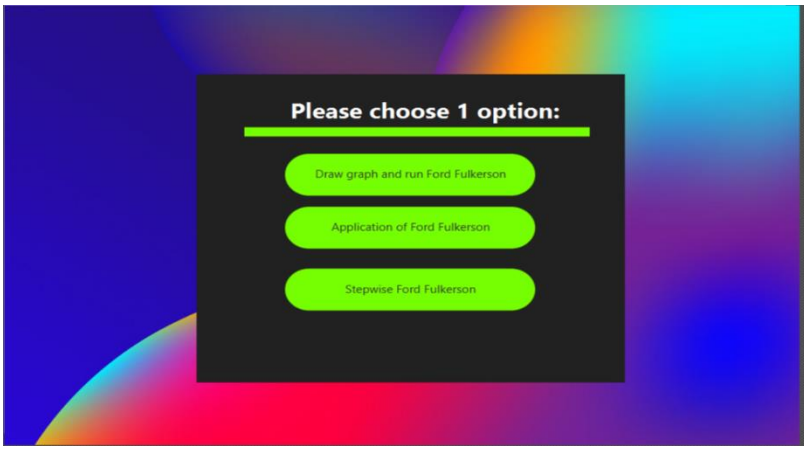


Σχήμα 4.1: Πρωτότυπο της διεπαφής της εφαρμογής Ford Fulkerson

Από το πιο πάνω σχήμα μπορούμε να δούμε ότι ο χρήστης αρχικά θα συνδέεται με την αρχική οθόνη (οθόνη 1) όπου θα έχει 3 επιλογές. Είτε θα επιλέγει το “Application of Ford

Fulkerson” και ακολούθως θα πηγαίνει στην επομένη οθόνη (οθόνη 2) που θα μπορεί να επιλέγει πόσες οντότητες θέλει στα αριστερά, πόσες οντότητες θέλει στα δεξιά και αν θέλει να τις ενώσει και αφού το κάνει αυτό θα πηγαίνει στην κύρια διεπαφή του προγράμματος (οθόνη 3) όπου και θα μπορεί να τρέξει τον αλγόριθμο στο δίκτυο ροής που θα έχει ήδη σχεδιαστεί αυτόματα. Αν ο χρήστης είχε επιλέξει στην πρώτη οθόνη είτε “Stepwise Ford Fulkerson “ είτε “Draw graph and run Ford Fulkerson” θα πηγαίνει απευθείας στην οθόνη 3 όπου και θα μπορεί να σχεδιάσει τα δικό του δίκτυο ροής και να τρέξει τον αλγόριθμο Ford Fulkerson πάνω σε αυτό.

Χρησιμοποιώντας το πιο πάνω πρωτότυπο προχωρήσαμε στην υλοποίηση του χρησιμοποιώντας το εργαλείο Scene Builder το οποίο θα μας παραγάγει τις πιο πάνω διεπαφές σε FXML κώδικα, το οποίο FXML μπορεί να φορτωθεί και να χρησιμοποιηθεί αυτόματα από την JavaFX. Λόγω του ότι οι διεπαφές μας δεν έχουν κάποια σχέση με τη λογική του προγράμματος οι αλλαγές μπορούσαν να γίνονται συνεχώς χωρίς να επηρεάζεται η υφιστάμενη υλοποίηση. Όπως θα δούμε στη συνέχεια οι πιο πάνω διεπαφές (που δείξαμε με draw.io) έχουν υλοποιηθεί ακριβώς όπως είχαν σχεδιαστεί και στα αρχικά πρωτότυπα (Σχήμα 4.1).



Σχήμα 4.2: Αρχική οθόνη της εφαρμογής

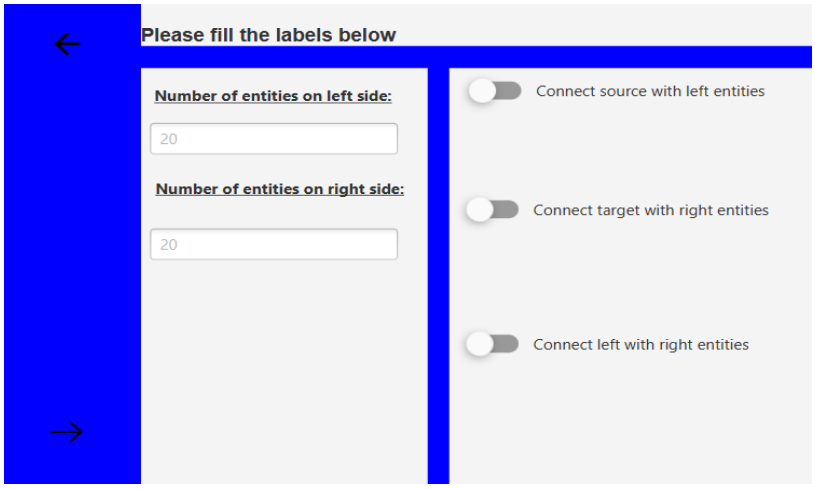
Όπως μπορούμε να δούμε η πρώτη εικόνα που βλέπει ο χρήστης έχει 3 κουμπιά τα οποία εξηγηθήκαν προηγουμένως . Η λειτουργικότητα που παρέχουν αυτά τα κουμπιά είναι μετάβαση από μια οθόνη σε μια άλλη. Η λειτουργικότητα αυτή υλοποιήθηκε χρησιμοποιώντας την γλώσσα Java.

Ακολουθεί ένα κομμάτι κώδικα το οποίο δείχνει πως επιτεύχθηκε η μετάβαση από τη πρώτη στη τρίτη οθόνη για το πρώτο κουμπί και πολύ παρόμοια λογική ισχύει και για τα υπόλοιπα κουμπιά.

```
FXMLLoader loader = new
FXMLLoader(getClass().getResource("/leftScreen.fxml"));
loader.load();

AnchorPane anchorPane = loader.getRoot();
ScrollPane scrollPane= (ScrollPane)
anchorPane.getChildren().get(0);
BorderPane root = new BorderPane();

LeftScreenController controller= loader.getController();

root.setLeft(scrollPane);
root.setCenter(graph.getScrollPane());

Scene scene = new Scene(root, 1024, 768);
addCss(scene);
primaryStage.setScene(scene);
primaryStage.show();
```

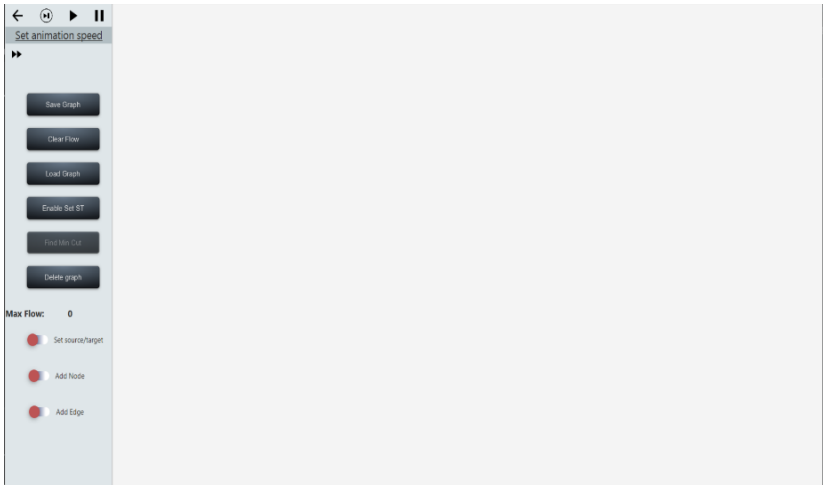
Απόσπασμα υλοποίησης μετάβασης από την πρώτη στην τρίτη οθόνη

Στη συνέχεια αν ο χρήστης επιλέξει την επιλογή “Application of Ford Fulkerson”μεταφέρεται στην πιο κάτω οθόνη.



Σχήμα 4.3: Οθόνη πληροφοριών για την επιλογή “Ford Fulkerson Application”

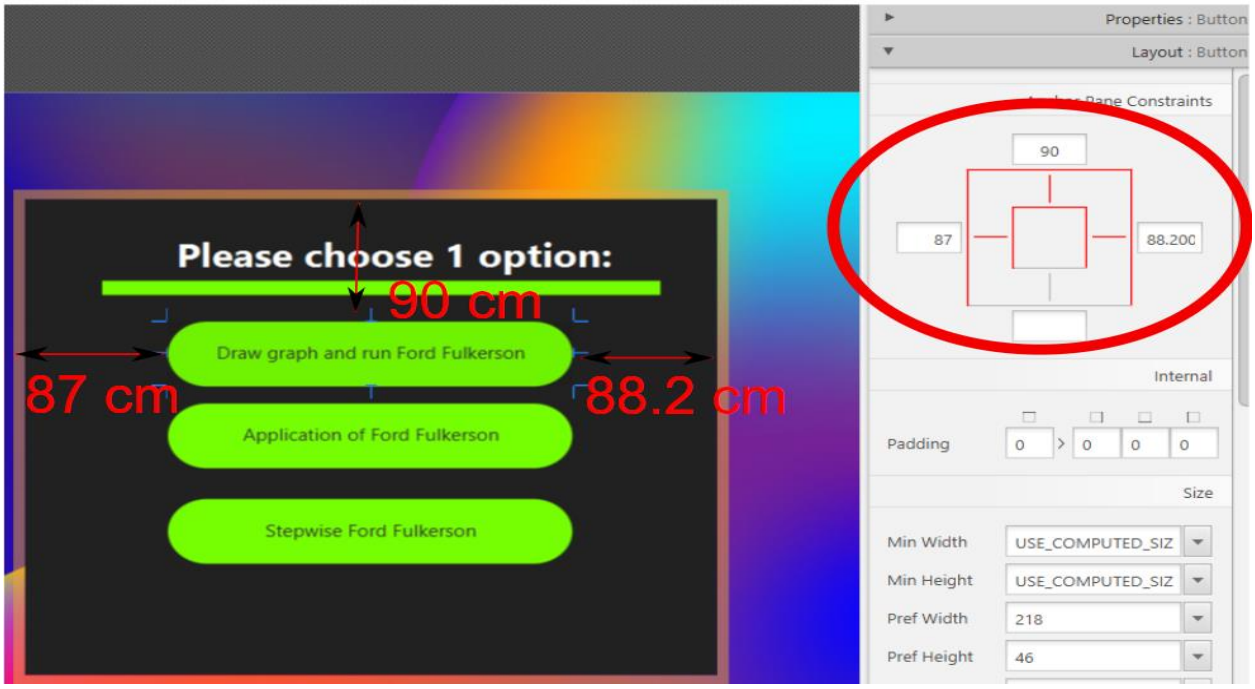
Η πιο πάνω οθόνη θα εξηγηθεί αναλυτικά στο Κεφάλαιο 6, αλλά περιληπτικά, στην πιο πάνω οθόνη ο χρήστης δηλώνει κάποιες πληροφορίες για το δίκτυο ροής και αυτό σχεδιάζεται αυτόματα. Ακολούθως πατώντας το “right button” μεταβαίνει στην τρίτη οθόνη όπου και βρίσκεται όλη η λογική του αλγορίθμου. Στην τρίτη οθόνη βρίσκεται η διεπαφή του αλγόριθμου Ford Fulkerson και αυτή η οθόνη θα εξηγηθεί στο επόμενο κεφάλαιο πολύ πιο αναλυτικά διότι εδώ βρίσκεται όλη η λογική και η κύρια λειτουργικότητα της εφαρμογής.



Σχήμα 4.4: Κύρια διεπαφή της εφαρμογής μας

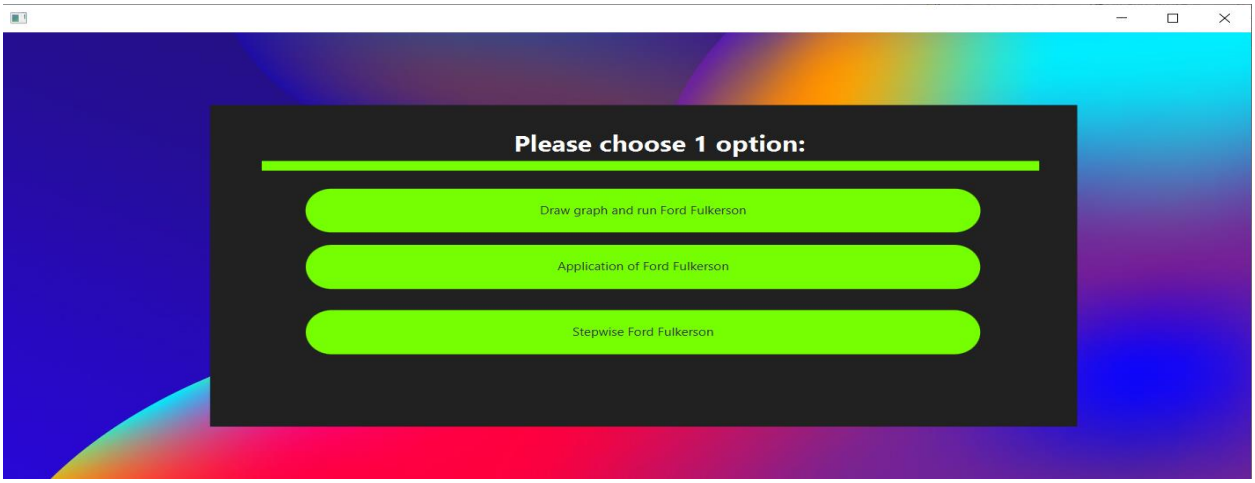
4.3. Προβλήματα και Διευκρινίσεις

Κατά τη διάρκεια της υλοποίησης των πιο πάνω διεπαφών παρουσιάστηκαν κάποια προβλήματα τα οποία είχαν να κάνουν κυρίως με την αποκρισιμότητα της εφαρμογής μας. Ιδανικά θα θέλαμε τα μεγέθη των διάφορων συστατικών όπως για παράδειγμα κουμπιά, ετικέτες(“labels”) κτλ, να προσαρμόζονται ανάλογα με το μέγεθος του παραθύρου στο οποίο βρίσκονται. Ευτυχώς για εμάς η λύση αυτού του προβλήματος ήταν αρκετά εύκολη χάρη στο Scene Builder το οποίο μας επιτρέπει να το κάνουμε αυτό γρήγορα και αυτόματα. Επιλέγοντας επομένως το συστατικό που θέλουμε στο Scene Builder και ακολούθως επιλέγοντας τη διάταξη του συστατικό αυτού(layout όπως αναφέρεται στο scene builder) μπορούμε να προσαρμόσουμε πόσο μπορεί να απέχει από το αντικείμενο στο οποίο βρίσκεται μέσα. Όπως φαίνεται και στο πιο κάτω σχήμα μπορούμε να καθορίσουμε ακριβώς πόσα cm μπορεί να απέχει το 1^{ο} κουμπί από το αντικείμενο στο οποίο βρίσκεται μέσα, αλλάζοντας τα Anchor Pane Constraints που υπάρχουν σε κάθε αντικείμενο που δημιουργείτε σε ένα καμβά. Σε αυτό το παράδειγμα ορίσαμε να απέχει πάντα 87 cm από τα αριστερά, 90 από πάνω και 88 cm από δεξιά.



Σχήμα 4.5: Δημιουργία Διαδραστικών κουμπιών με το εργαλείο Scene Builder

Με αυτό τον τρόπο το μέγεθος του κουμπιού προσαρμόζεται αναλόγως με το μέγεθος του αντικειμένου στο οποίο ανήκει. Ακολουθεί παράδειγμα όπου το μέγεθος του αντικειμένου μεγαλώνει (δηλαδή το μέγεθος του παραθύρου της εφαρμογής στην πιο κάτω περίπτωση), αυτό έχει σαν αποτέλεσμα να προσαρμόζεται και το μέγεθος του κουμπιού. Οι αποστάσεις όμως στα αριστερά και στα δεξιά μένουν όπως έχουν δηλωθεί πιο πάνω. Παρόμοια λογική ακολουθήθηκε και στις άλλες οθόνες έτσι ώστε να είναι διαδραστικές.



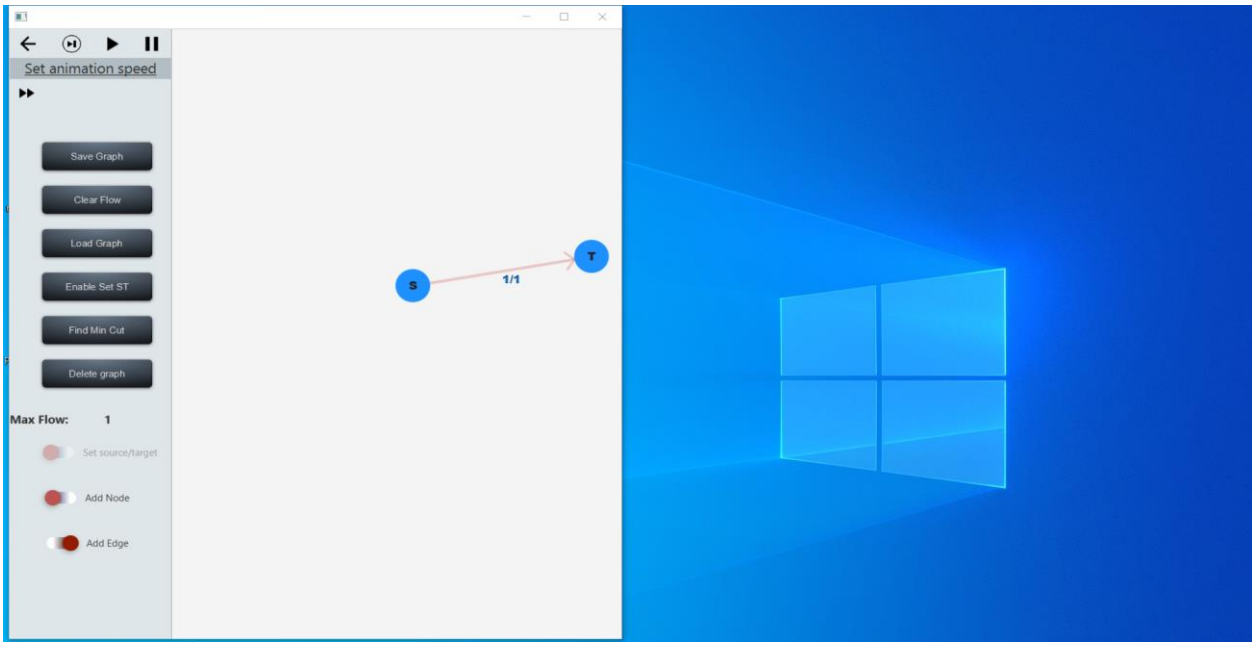
Σχήμα 4.6: Αλλαγή μεγέθους κουμπιών λόγω αλλαγής μεγέθους οθόνης

Ακόμη ένα πρόβλημα που συναντήσαμε ήταν ποια θα ήταν η αρχική θέση ενός παράθυρου μόλις αυτό εμφανιστεί. Για παράδειγμα, θέλαμε κάποια παράθυρα να ευθυγραμμίζονται στην πάνω αριστερή γωνιά της οθόνης του υπολογιστή μας και να καταλαμβάνουν την μισή οθόνη του χρήστη. Για να το επιτύχουμε αυτό χρησιμοποιήσαμε την μέθοδο της JavaFX, **Screen.getPrimary().getVisualBounds()**. Αυτό που κάνει αυτή η μέθοδος είναι ότι δημιουργεί ένα αντικείμενο με τις διαστάσεις της οθόνης του υπολογιστή που τρέχει η εφαρμογή και χρησιμοποιώντας αυτό το αντικείμενο, με τον πιο κάτω τρόπο, μπορούμε να καθορίσουμε την αρχική θέση του παραθύρου αλλά και το μέγεθος του.

```
Rectangle2D
visualBounds=Screen.getPrimary().getVisualBounds();
primaryStage.setX(visualBounds.getMinX());
primaryStage.setY(visualBounds.getMinY());
primaryStage.setHeight(visualBounds.getHeight());
primaryStage.setWidth(visualBounds.getWidth()/2);
```

Απόσπασμα καθορισμού μεγέθους και τοποθεσίας του παραθύρου ανάλογα με το μέγεθος της οθόνης του υπολογιστή που τρέχει η εφαρμογή

Όπως μπορούμε να δούμε και από το πιο πάνω κομμάτι κώδικα θέτουμε το παράθυρο (primary stage) με το setX() και setY() να ξεκινά στο πάνω αριστερό άκρο της οθόνης και ακολούθως με το setHeight() του δίνουμε ύψος ακριβώς όσο και του ύψος της οθόνης της συσκευής μας και πλάτος το μισό της οθόνης της συσκευής μας. Το αποτέλεσμα του πιο πάνω κώδικα είναι το πιο κάτω όπου μπορούμε να δούμε ότι όντως το παράθυρο καταλαμβάνει ακριβώς τη μίση μας οθόνη.



Σχήμα 4.7 – Δημιουργία Παραθύρου που έχει πλάτος το μισό της οθόνης. Αποτέλεσμα κώδικα καθορισμού μεγέθους και τοποθεσίας του παραθύρου

Όλα αυτά που ανάφερα σε αυτή το κεφάλαιο, θα εφαρμόζονται και θα χρησιμοποιούνται σε ολόκληρη την εφαρμογή. Όλες οι οθόνες θα λειτουργούν με παρόμοιους τρόπους και παρόμοιες τεχνικές τόσο για την διαχείριση της αρχικής θέσης μιας οθόνης αλλά και για την δημιουργία οθονών των οποίων τα στοιχεία τους προσαρμόζονται ανάλογα με το μέγεθος της οθόνης

Κεφάλαιο 5

Υλοποίηση Διεπαφής Αλγόριθμου Ford Fulkerson

5.1. Εισαγωγή

Σε αυτή το κεφάλαιο θα εξηγήσουμε και θα περιγράψουμε σε πιο πρακτικό βαθμό τις λειτουργίες της εφαρμογής μας αλλά και τον τρόπο υλοποίησης τους. Με αυτό τον τρόπο ο αναγνώστης θα είναι σε θέση να γνωρίζει τί κάνει το κάθε κουμπί , τον τρόπο υλοποίησης του αλλά και τι ακριβώς υποστηρίζει η εφαρμογή μας. Επομένως αρχικά θα εξηγήσουμε την λειτουργεία κάθε κουμπιού στην διεπαφή μας , στη συνέχεια θα εξηγήσουμε πώς δημιουργείται και παρουσιάζεται το υπολειπόμενο δίκτυο στον χρήστη, πώς υλοποιήθηκαν τα κινούμενα σχέδια, τα οποία δείχνουν τα μονοπάτια επαύξησης αλλά και τις ακμές που προστίθεται η ροή και τέλος τον τρόπο με τον οποίο βρίσκουμε και παρουσιάζουμε την ελάχιστη αποκοπή.

5.2. Δημιουργία Δίκτυού Ροής μέσω Διεπαφής - Λεπτομέρειες Υλοποίησης

5.2.1. Περιγραφή Υλοποίησης Διαδικασίας Προσθήκης Κόμβου – Κουμπί Add Node

Για την δημιουργία μιας διαδραστικής εφαρμογής πρέπει να δίνεται η δυνατότητα στον χρήστη να δημιουργήσει τους κόμβους του δίκτυού ροής κάνοντας κλικ απλά πάνω σε κάποιο σημείο της σκηνής/οθόνης. Επομένως κάθε φορά που κάνει κλικ ο χρήστης πρέπει να πιάνουμε το κλικ πάνω στην οθόνη. Για να επιτευχθεί αυτό έπρεπε να πάρουμε το αντικείμενο της οθόνης και να του προσθέσουμε ένα “event handler”, δηλαδή μια μέθοδο που καλείται οποτεδήποτε υπάρξει κάποιο κλικ πάνω στην οθόνη. Πιο κάτω μπορούμε να δούμε πως πάνω στο αντικείμενο της οθόνης, το οποίο παίρνουμε με την μέθοδο getScrollPane(), προσθέτουμε ένα “event handler” που κάνει capture τα mouse clicks και όποτε υπάρχει κάποιο click πάνω στην οθόνη εκτελεί την μέθοδο addNode(…).

```
graph.getScrollPane().addEventHandler(MouseEvent.MOUSE_CLICKE
D,mouseEvent ->{

addNode(mouseEvent,root.getLeft().getBoundsInParent().getWidt
h(),scene);

});
```

Απόσπασμα προσθήκης event handler πάνω στην οθόνη

Πάμε τώρα να δούμε πώς λειτουργεί η μέθοδος addNode() η οποία είναι υπεύθυνη τόσο για την προσθήκη ενός κόμβου οπτικά πάνω στην οθόνη, αλλά και αλγοριθμικά για την προσθήκη του κόμβου πάνω στη δομή του γράφου.

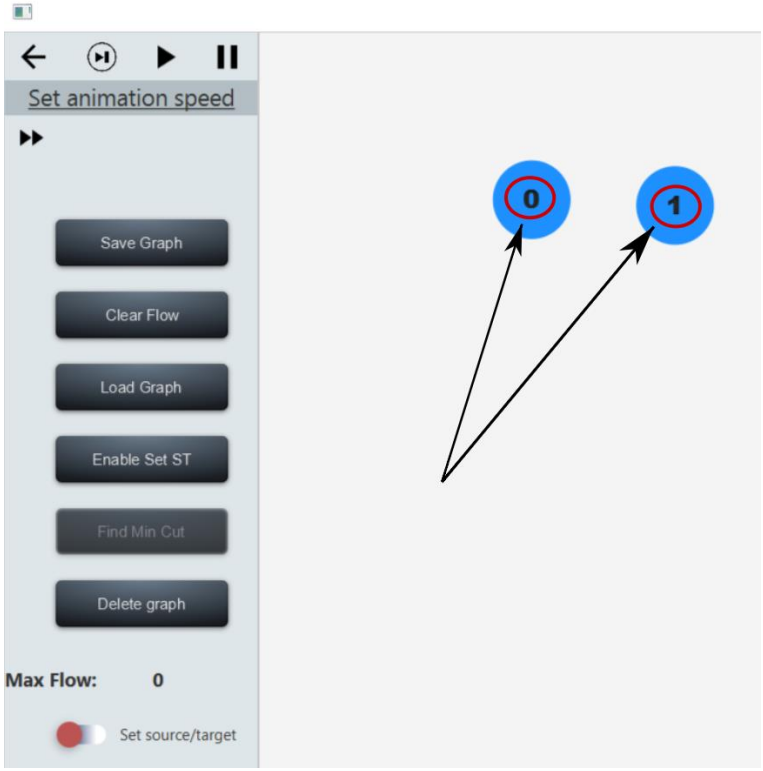
```
private void addNode (MouseEvent mouseEvent, double
widthOfLeftPane, Scene scene) {

    double scale = graph.scrollPane.getScaleValue();
    double coordX = mouseEvent.getSceneX() / scale ;
    double coordY = mouseEvent.getSceneY() / scale ;

graph.getModel().addCell((nodeID++)+"",CellType.RECTANGLE,coo
rdX,coordY,scene);

                                }
```

Απόσπασμα προσθήκης κόμβου πάνω στην οθόνη

Για την προσθήκη του κόμβου πάνω στην οθόνη, ακριβώς στο σημείο που κάνει κλικ ο χρήστης, αρχικά πρέπει να βρούμε την κλίμακα της οθόνης μας, δηλαδή το πόσο ζουμαρισμένη είναι η οθόνη μας. Ακολούθως πρέπει να βρούμε τις συντεταγμένες του σημείου που έκανε κλικ ο χρήστης, αυτό επιτυγχάνεται εύκολα χρησιμοποιώντας το αντικείμενο MouseEvent που μας παρέχει η JavaFX, και να διαιρέσουμε αυτές τις συντεταγμένες με την κλίμακα της οθόνης μας που βρήκαμε προηγούμενος. Έχοντας πλέον

τις σωστές συντεταγμένες που πρέπει να προστεθεί ο νέος κόμβος, δημιουργούμε σε αυτό το σημείο ένα αντικείμενο κύκλου που παρέχεται από την JavaFX και αυτός ο κύκλος θα συμβολίζει τον νέο κόμβο.

Επιπρόσθετα κάθε νέος κόμβος έχει και μια ταυτότητα, η ταυτότητα αυτή είναι ουσιαστικά ένας αύξων αριθμός. Στο πιο πάνω κομμάτι κώδικα η ταυτότητα του κόμβου δίνεται στην $4^{η}$ γραμμή (nodeID++). Θέλαμε αυτή η ταυτότητα να βρίσκεται στο κέντρο του κύκλου που αντιστοιχεί και να παραμένει σε αυτό το σημείο ακόμα και αν ο χρήστης μεταφέρει τον κόμβο σε άλλο σημείο. Δηλαδή θέλαμε το πιο κάτω αποτέλεσμα, οι αριθμοί που βρίσκονται κυκλωμένοι να βρίσκονται στο κέντρο του node και να παραμένουν εκεί.



Σχήμα 5.1 – Ταυτότητα κόμβου στο κέντρο του κόμβου

Για να καταφέρουμε το πιο πάνω, δημιουργήσαμε μια ετικέτα την οποία όμως έπρεπε να τοποθετήσουμε στο κέντρο του κύκλου. Το κεντράρισμα της ετικέτας έγινε με μια ισχυρή μέθοδο της JavaFX που ονομάζεται bind(). Όταν δυο αντικείμενα γίνουν bind() μεταξύ τους,

οι αλλαγές που γίνονται στο ένα αντικείμενο εφαρμόζονται αυτόματα και στο άλλο. Επομένως το μόνο που είχα να κάνω για να κεντράρω την ετικέτα στο κέντρο του κύκλου, ήταν να ενώσω(με την μέθοδο bind()) τις συντεταγμένες X και Y της ετικέτας με τις συντεταγμένες που αντιστοιχούν στο κέντρο του κύκλου. Ακολουθεί ένα κομμάτι κώδικα το οποίο δείχνει πως επιτεύχθηκε η υλοποίηση της πιο πάνω διαδικασίας.

```
label_id=new Label(id);
label_id.layoutXProperty().bind(circle.centerXProperty().subt
ract(radiusSize/4)
label_id.layoutYProperty().bind(circle.centerYProperty().subt
ract(radiusSize/2)
```

Απόσπασμα προσθήκης της ετικέτας (ID Label) πάνω στον κόμβο

5.2.1.1. Περιγραφή υλοποίησης Μεταφοράς και Απόθεσης του Κόμβου

Για να κάνουμε την εφαρμογή μας όσο πιο διαδραστική και εύχρηστη μπορούσαμε, έπρεπε να δίνουμε την δυνατότητα στον χρήστη να μπορεί να μεταφέρει τους κόμβους που προσθέτει. Για την υλοποίηση αυτής της λειτουργείας εφαρμόσαμε τα τρια πιο κάτω απλά βήματα για κάθε κόμβο που προσθέτουμε.

1. Καταγραφή της αρχικής τοποθεσίας του κόμβου χρησιμοποιώντας τις ιδιότητες “LayoutX” και “LayoutY” που έχει κάθε αντικείμενο του JavaFX.
2. Μέτρηση της συνολικής κίνησης του ποντικιού καθώς σύρεται ο κόμβος, εφαρμόζοντας αυτή την κίνηση στον κόμβο.
3. Πραγματοποίηση αλλαγών στον κόμβο όταν απελευθερωθεί το ποντίκι.

Ακολουθούν τρία κομμάτια κώδικα που υλοποιούν τα πιο πάνω βήματα και καλούνται κάθε φορά που μεταφέρουμε ένα κόμβο.

Υλοποίηση 1ου βήματος - Καταγραφή της αρχικής τοποθεσίας του κόμβου :

```
node.setOnMousePressed(onMousePressedEventHandler);
public void handle(MouseEvent event) {
    CircleCell node = (CircleCell) event.getSource();
    if (node.isResidualCell())
        return;
    double scale = graph.getScale();
    dragContext.x = node.getBoundsInParent().getMinX() *
scale - event.getScreenX()+30;
    dragContext.y = node.getBoundsInParent().getMinY() *
scale - event.getScreenY()+30;
}
```

Απόσπασμα καταγραφής της αρχικής τοποθεσίας του κόμβου

Υλοποίηση 2ου βήματος – Μέτρηση της συνολικής κίνησης του ποντικιού καθώς σύρεται ο κόμβος:

```
node.setOnMouseDragged(onMouseDraggedEventHandler);

public void handle(MouseEvent event) {
        double scale = graph.getScale();
        Node node = (Node) event.getSource();
        double 44ffset = event.getScreenX() + dragContext.x-
(1-scale)*20;
        double 44ffset = event.getScreenY() + dragContext.y-
(1-scale)*20;
}
```

Απόσπασμα μέτρησης συνολικής κίνησης του ποντικιού

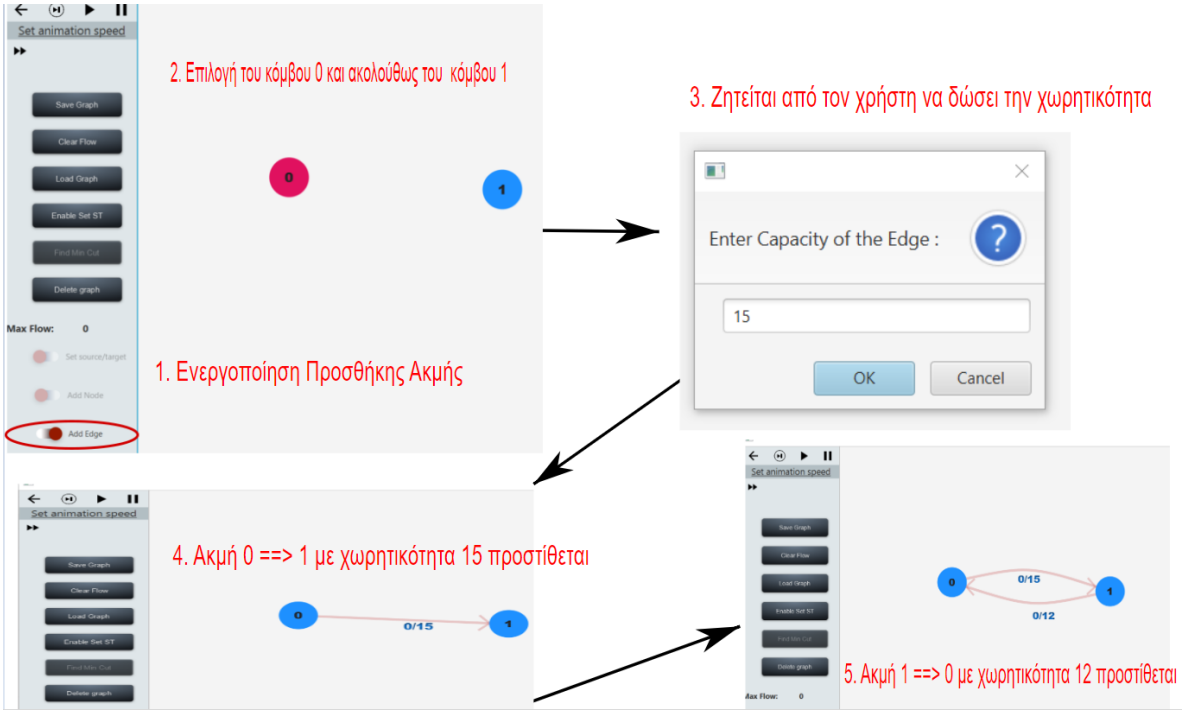
Υλοποίηση 3ου βήματος - Πραγματοποίηση αλλαγών στον κόμβο όταν απελευθερωθεί το ποντίκι.

```
node.setOnMouseReleased(onMouseReleasedEventHandler);

public void handle(MouseEvent event) {
        //adjust the offset in case we are zoomed
        offsetX /= scale;
        offsetY /= scale;
        node.relocate(offsetX, offsetY);
}
```

Απόσπασμα πραγματοποίησης αλλαγών στον κόμβο όταν απελευθερωθεί το ποντίκι

5.2.2. Περιγραφή Υλοποίησης Διαδικασίας Προσθήκης Ακμής – Κουμπί Add Edge

Ο χρήστης έχει την δυνατότητα να προσθέσει κατευθυνόμενες ακμές. Για να ενεργοποιήσει αυτή τη δυνατότητα πρέπει να επιλέξει το συρόμενο κουμπί “Add Edge”. Ακολούθως θα πρέπει να επιλέξει δυο κόμβους Α και Β για να προσθέσει μια ακμή μεταξύ τους. Αν επιλέξει πρώτα τον Α και μετά τον Β τότε προστίθεται μια ακμή Α->Β , αλλιώς αν επιλέξει τον Β και μετά τον Α τότε προστίθεται μια ακμή από τον Β->Α. Η διαδικασία που μόλις ανάφερα φαίνεται και στο πιο κάτω σχήμα.



Σχήμα 5.2 – Διαδικασία προσθήκης ακμής

Τώρα θα δείξουμε κάποιες λεπτομέρειες υλοποίησής οι οποίες έχουν να κάνουμε με την προσθήκη ακμής μεταξύ δυο κόμβων. Όταν ο χρήστης προσθέτει ακμές μεταξύ δυο κόμβων Α και Β, υπάρχουν δυο περιπτώσεις:

1. Είτε υπάρχει είδη μια ακμή από Α προς Β ή Β προς Α.
2. Είτε δεν υπάρχει καμία ακμή μεταξύ των δυο κόμβων.

Αρχικά και στις δυο περιπτώσεις προτού δημιουργηθεί μια νέα ακμή, εμφανίζεται ένα pop up παράθυρο το οποίο ρώτα τον χρήστη ποιο θα είναι το μέγεθος της ακμής(βήμα 3 στο πιο πάνω σχήμα). Η υλοποίηση για το pop up παράθυρο είναι η πιο κάτω.

```
TextInputDialog dialog = new TextInputDialog("1");
dialog.setHeaderText("Enter Capacity of the Edge :");
Optional<String> result = dialog.showAndWait();
if (result.isPresent()) {
    result=checkNumberFormat(result,dialog);
    weight.setText(flow+"/"+result.get());
    setCapacity(Integer.parseInt(result.get()));
} else {
    weight.setText("/1");
    setCapacity(1);
}
```

Απόσπασμα Υλοποίησης Pop Up παραθύρου

Ουσιαστικά δημιουργείται ένα αντικείμενο TextInputDialog και με την συνάρτηση showAndWait() εμφανίζεται το pop up παράθυρο που βλέπουμε στο πιο πάνω σχήμα (στο Σχήμα 5.2), στο βήμα 3.

Τώρα όσο αφορά την δεύτερηπερίπτωση , δηλαδή όταν δεν υπάρχει ήδη κάποια ακμή μεταξύ δυο κόμβων δημιουργείται η ακμή που φαίνεται στο βήμα 4 του σχήματος 5.2, δηλαδή μια ευθεία γραμμή με ένα τόξο συνδεδεμένο στο τέλος της.

Πάμε τώρα να εξηγήσουμε πώς δημιουργείται αυτή η γραμμή και πως την συνδέουμε με τους 2 κόμβους έτσι ώστε να μείνει συνδεδεμένη μαζί τους ακόμη και όταν αυτοί οι κόμβοι μετακινούνται από τον χρήστη.

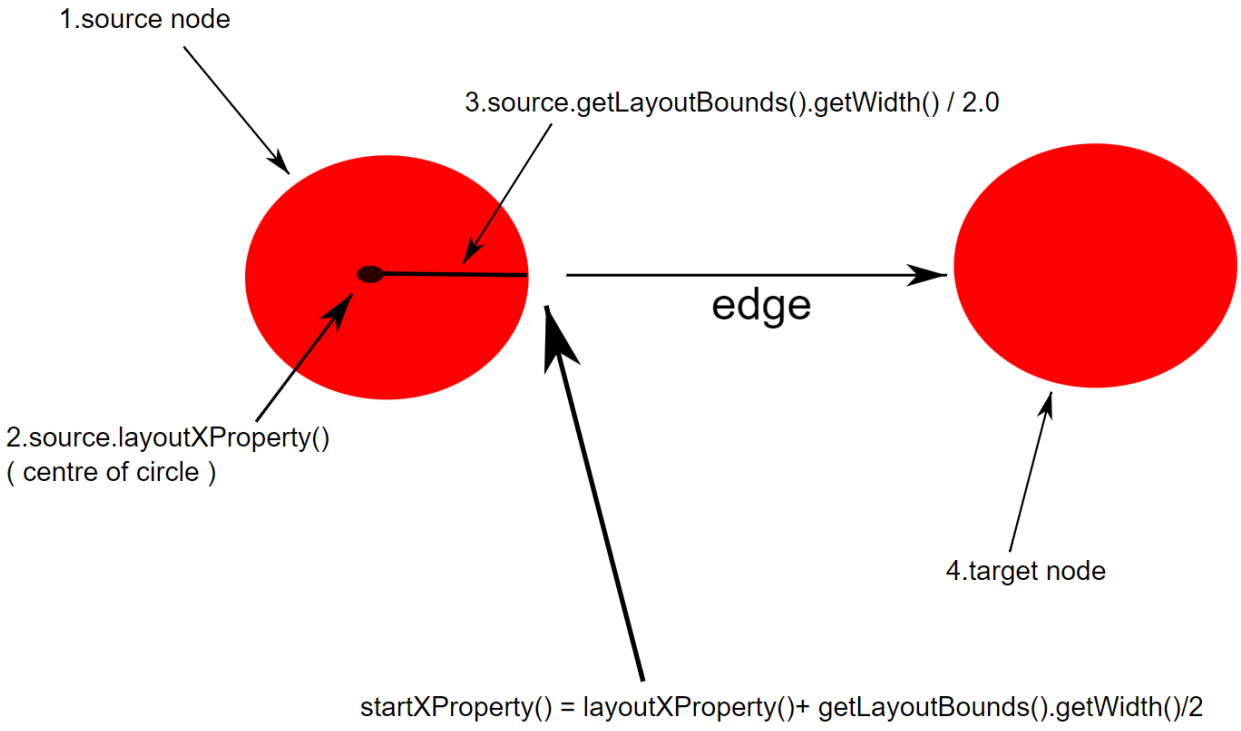
```
line = new MyLine((CircleCell) target);
line.startXProperty().bind( source.layoutXProperty().add(source.getLayoutBounds().getWidth() / 2.0 ));
line.startYProperty().bind( source.layoutYProperty().add(source.getLayoutBounds().getHeight() / 2.0 ));
line.endXProperty().bind( target.layoutXProperty().add( target.getLayoutBounds().getWidth() / 2.0 ));
line.endYProperty().bind( target.layoutYProperty().add( target.getLayoutBounds().getHeight() / 2.0 ));
```

Απόσπασμα δημιουργίας γραμμής που θα συμβολίζει την ακμή

Όπως φαίνεται και από το πιο πάνω απόσπασμα κώδικα, ενώνουμε την αρχική X και Y τοποθεσία της γραμμής (startXProperty και startYProperty) με τον κόμβο από τον οποίο ξεκινά η ακμή. Ουσιαστικά λέμε ότι η συντεταγμένη X για την γραμμή , το startXProperty()

της δηλαδή, θα είναι το ίδιο με το κέντρο του κύκλου συν το μήκος της μισής ακτίνας του κύκλου. Η ίδια λογική ισχύει και για τις συντεταγμένες που συνδέονται με τον 2ο κόμβο. Επομένως με αυτό τον τρόπο συνδέσαμε μια ακμή με τους κόμβους που αντιστοιχούν σε αυτή την ακμή.

Ακολουθεί και ένα σχήμα το οποίο περιγράφει αναλυτικά πως υπολογίστηκαν οι συντεταγμένες της ακμής και σχετίζεται με τον πιο πάνω κώδικα για καλύτερη κατανόηση.



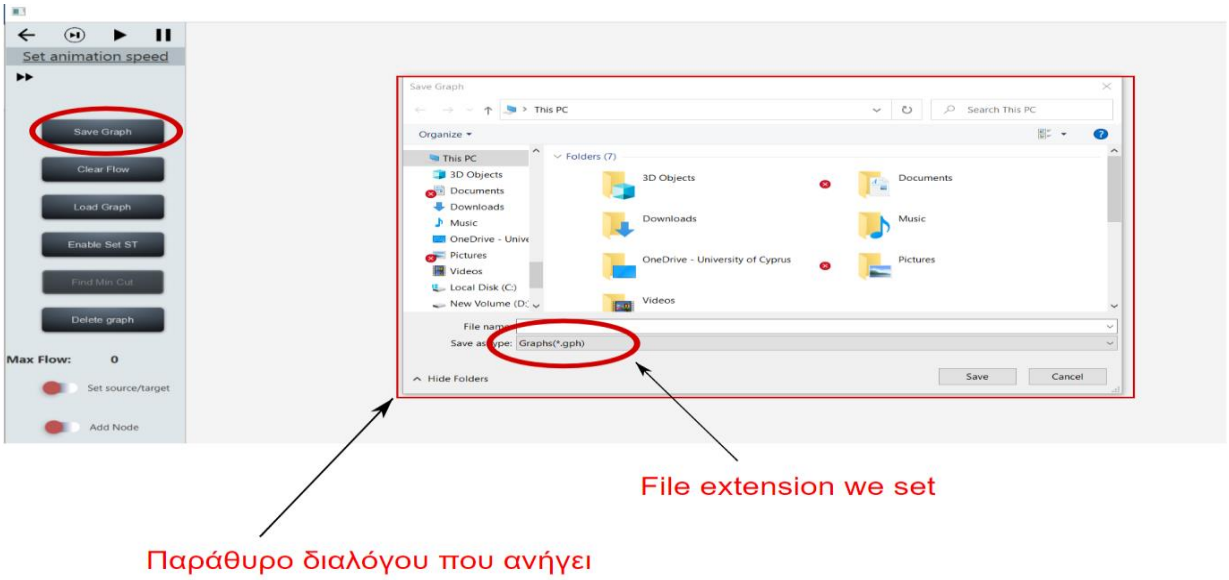
Σχήμα 5.3 – Επεξήγηση Υπολογισμού συντεταγμένων ακμής

Στην περίπτωση που υπάρχει είδη ακμή μεταξύ δυο κόμβων αλλάζει πλέον το σχήμα των ακμών και γίνεται πιο κυρτό όπως φαίνεται και στο βήμα 5 στο Σχήμα 5.2. Αυτό συμβαίνει έτσι ώστε οι χωρητικότητες των δυο ακμών να μην συγκρούονται και να είναι ευδιάκριτες. Η υλοποίηση για την κυρτή ακμή ήταν ακριβώς η ίδια με αυτήν που περιγράψαμε πιο πάνω, το μόνο που άλλαξε στην υλοποίησή είναι ότι αντί να δημιουργείτε το object “Line” στο JavaFX δημιουργείται το object “CubicCurve”.

5.2.3. Περιγραφή Υλοποίησης Κουμπιού Save Graph – Load Graph

Η εφαρμογή μας έχει ως στόχο να παρέχει λειτουργίες και να κάνει την εφαρμογή μας ευχάριστη και εύχρηστη. Επομένως υλοποιήσαμε την λειτουργία Save Graph έτσι ώστε ο χρήστης να μπορεί να αποθηκεύει τους γράφους που δημιουργεί για να μην επαναλαμβάνει τις κινήσεις του και επίσης να γλιτώνει χρόνο σε περίπτωση που θέλει να δημιουργήσει ξανά τον ίδιο γράφο.

Η λειτουργεία του κουμπιού έχει ως εξής, όταν το πατά ο χρήστης το κουμπί θα ανοίγει ένα παράθυρο διαλόγου, ακολούθως θα μπορεί να ψάξει μέσα στο σύστημα αρχείων του, το σημείο που θέλει να αποθηκεύσει τον γράφο και ακολούθως θα πληκτρολογεί το όνομα με το οποίο θέλει να αποθηκεύσει το αρχείο.



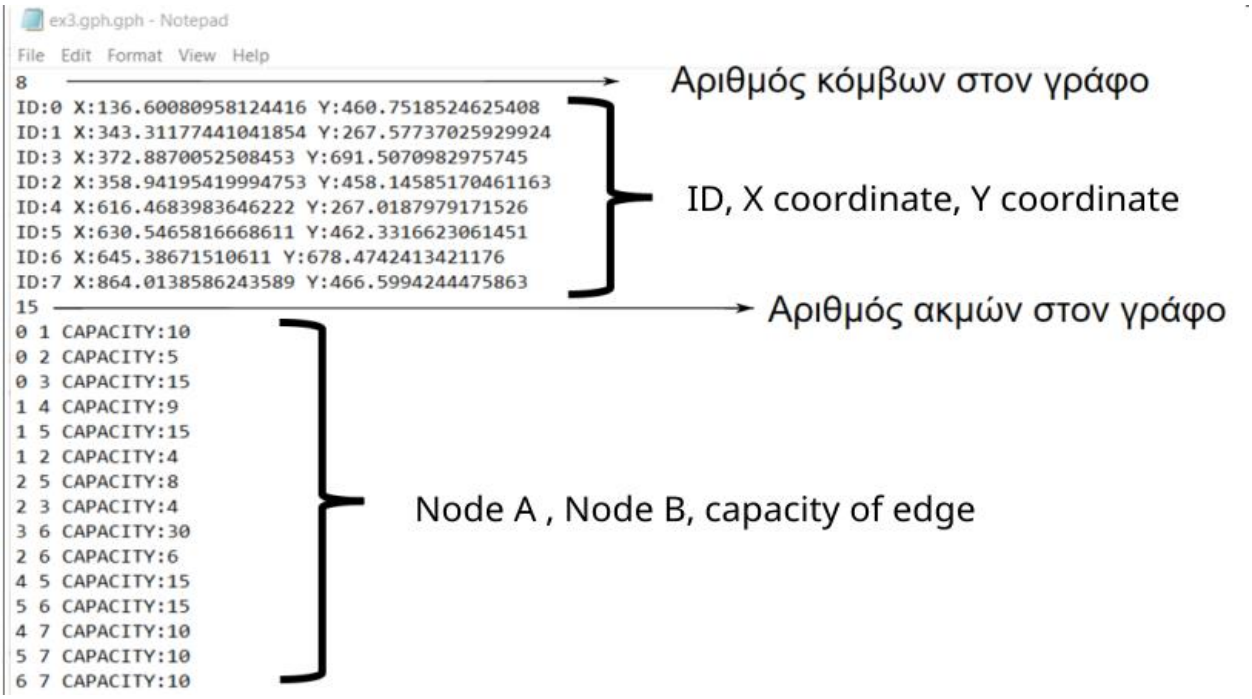
Σχήμα 5.4 – Διαδικασία αποθήκευσης γράφου

<u>Η υλοποίηση της πιο πάνω λειτουργίας επιτεύχθηκε με τον πιο κάτω τρόπο</u> :

```
FileChooser saveWindow = new FileChooser();
FileChooser.ExtensionFilter extFilter =
new FileChooser.ExtensionFilter("Graphs(*.gph)", "*.gph");
saveWindow.getExtensionFilters().add(extFilter);
File sfile = saveWindow.showSaveDialog(primaryStage);
```

Απόσπασμα Υλοποίησης Save Graph Button

Όπως φαίνεται και από το πιο πάνω κομμάτι κώδικα, χρησιμοποιώντας το αντικείμενο FileChooser(), μπορούμε να ανοίξουμε ένα παράθυρο διαλόγου, να του θέσουμε το extension του File που θα δημιουργηθεί και τέλος να εμφανίσουμε ένα παράθυρο διαλόγου όπως φαίνεται και στο πιο πάνω σχήμα με τη μέθοδο showSaveDialog(). Αυτή η μέθοδος αναμένει να επιλέξουμε τον χώρο που θα φυλαχτεί το αρχείο αλλά και να δώσουμε το όνομα του αρχείου και στη συνέχεια, θα επιστραφεί ένας δείκτης αρχείου πάνω στο αρχείο που δημιουργείτε έτσι ώστε να μπορούμε να γράφουμε πάνω σε αυτό. Ακολούθως αποθηκεύουμε στο αρχείο αυτό, τον γράφο μας με την εξής δομή.



Σχήμα 5.5 – Μορφή αρχείου που αποθηκεύεται το δίκτυο ροής

Όπως μπορούμε να δούμε στην πρώτη γραμμή έχουμε τον αριθμό των κόμβων, έστω αυτός ο αριθμός είναι ο X, οι επόμενες X γραμμές αντιστοιχούν στους κόμβους, πιο συγκεκριμένα όσο αφορά τις γραμμές που αντιστοιχούν στους κόμβους ο $1^{ος}$ αριθμός είναι η ταυτότητα του κόμβου, και το X και Y είναι οι συντεταγμένες του κόμβου πάνω στη σκηνή. Ακολουθεί ο αριθμός των ακμών, έστω αυτός ο αριθμός Z, οι επόμενες Z γραμμές αντιστοιχούν στις κατευθυνόμενες ακμές, όπου αν μια γραμμή είναι για παράδειγμα "0 1 CAPACITY:10" συμβολίζει ότι υπάρχει ακμή από τον κόμβο με ταυτότητα 0 στον κόμβο με ταυτότητα 1 με χωρητικότητα 10.

Η υλοποίηση του κουμπιού "Load Graph" ακολουθεί ακριβώς την ίδια προσέγγιση με αυτή που μόλις περιγράψαμε για το κουμπί "Save Graph". Δηλαδή και πάλι εμφανίζουμε ένα παράθυρου διαλόγου στον χρήστη που τον αφήνει να ψάξει μέσα στο σύστημα αρχείων του, να βρει το αρχείο που θέλει να ανοίξει και να του επιστραφεί ένας δείκτης αρχείου πάνω στο αρχείο που επέλεξε. Αξίζει να σημειωθεί ότι το αρχείο που θα ανοίξει ο χρήστης πρέπει να έχει την μορφή αρχείου που περιγράψαμε προηγούμενος. Λόγω του ότι το πιο πάνω αρχείο έχει όλες τις απαραίτητες πληροφορίες για την δημιουργία ενός γράφου, μπορεί να τον ξαναδημιουργήσει και να είναι ακριβώς ίδιος με αυτόν που είχε αποθηκεύσει.

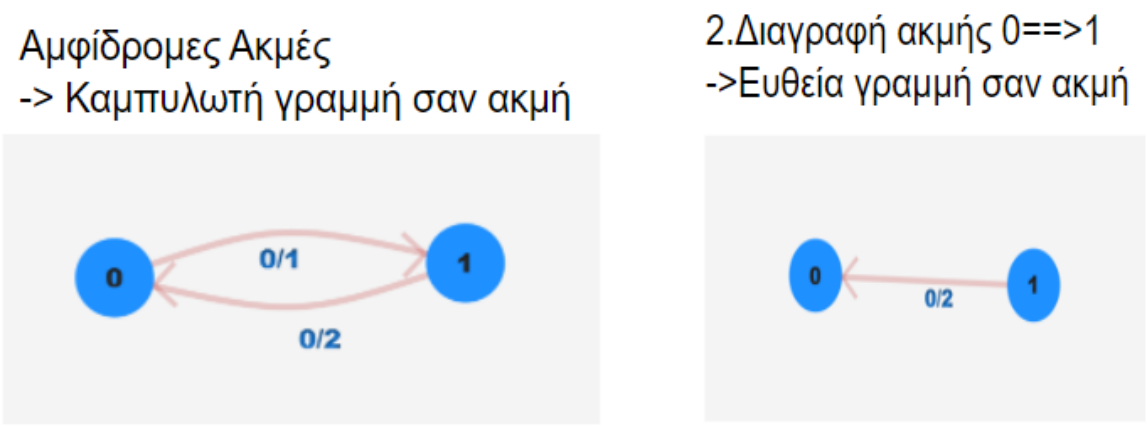
5.2.4. Περιγραφή Υλοποίησης Κουμπιού Delete Graph

Ακόμη μια λειτουργία που παρέχει η εφαρμογή μας είναι η διαγραφή όλου του γράφου , δηλαδή κάθε κόμβου και κάθε ακμής που υπάρχει πάνω στην οθόνη. Για την υλοποίηση αυτής της λειτουργίας έπρεπε πρώτα να υλοποιηθούν ακόμη δυο λειτουργίες:

1. Η διαγραφή ακμής
2. Η διαγραφή κόμβου

Η διαγραφή ακμής ήταν λίγο πιο περίπλοκη από την διαγραφή κόμβου διότι θέλαμε να έχουμε το πιο κάτω αποτέλεσμα που φαίνεται στο σχήμα 5.6. Δηλαδή όταν έχουμε δυο

ακμές μεταξύ δυο κόμβων οι ακμές να έχουν κυρτό σχήμα. Ενώ όταν διαγράφουμε μια ακμή και έχουμε ως αποτέλεσμα να μένει μόνο μια ακμή, αυτή να έχει σαν σχήμα μια ευθεία γραμμή.



Σχήμα 5.6 – Αλλαγή μορφής ακμών καθώς διαγράφουμε ακμές

Αρά για να έχουμε το πιο πάνω επιθυμητό αποτέλεσμα πρέπει όταν διαγράφεται μια από τις δυο ακμές να αντικαταστείτε το σχήμα της αντίστροφης ακμής με μια ίσια γραμμή, ακριβώς όπως και στο πιο πάνω σχήμα.

```
public void removeEdge(Edge edge){

    Cell source=edge.source;
    Cell target=edge.target;
    /** Since edge is deleted the 2 nodes are no longer
neighbors */
    source.children.remove(target);
    target.parents.remove(source);
    /** Check if inverse edge exist, if it does then the
CubicCurveLine is                deleted and it is replaced by
a straight Line*/
    Edge inverseEdge=getEdgeByUniqueID(target,source);
    if (inverseEdge!=null){


inverseEdge.getChildren().removeAll(inverseEdge.cubicCurveLine);

        MyLine line=new MyLine((CircleCell) edge.source);
        bindLine(line,target,source);
        bind_label_to_line(target,source,inverseEdge.getWeight());

        inverseEdge.line=line;

inverseEdge.getChildren().addAll(line,line.getAttachedArrow()
);
    }
    /** The other edge is just deleted.  */
    removedEdges.add(edge);
    addedEdges.remove(edge);
    edgesHashSet.remove(edge.getUniqueID());
}
```

Απόσπασμα Διαγραφής Ακμής

Όπως φαίνεται και στον πιο πάνω κώδικα αρχικά ελέγχουμε αν υπάρχει ακμή και από την άλλη κατεύθυνση. Αν υπάρχει διαγράφουμε το “CubicCurveLine” και το αντικαθιστούμε με μια ίσια γραμμή - ακμή, που δημιουργούμε εκείνη την στιγμή με την διαδικασία που εξηγήσαμε στο Κεφάλαιο 5.2.3.

Η διαγραφή κόμβου από την άλλη ήταν μια σχετικά απλή διαδικασία διότι έχοντας την ταυτότητα του κόμβου που θέλαμε να διαγράψουμε, έπρεπε να βρούμε και να κάνουμε delete τον κόμβο από την δομή που τον αποθηκεύσαμε (συνδεδεμένη λίστα). Αυτό επιτεύχθηκε με το πιο κάτω απόσπασμα κώδικα.

```
removedCells.add(cell);
addedCells.remove(cell);
cellMap.remove(cell.getCellId());
```

Απόσπασμα Διαγραφής Κόμβου “cell”

Επιπρόσθετα, αν διαγραφτεί κάποιος κόμβος που είχε προσαρτημένη κάποια ακμή πάνω του, τότε έπρεπε και η ακμή αυτή να διαγραφτεί. Επομένως έπρεπε να βρούμε αν υπάρχει κάποια ακμή προσαρτημένη σε έναν κόμβο και τότε να την διαγράψουμε.

```
for (Edge edge:allEdges){
    if (edge.contains(cell))
        removeEdge(edge);
}
```

Απόσπασμα Διαγραφής Ακμής που είναι προσαρτημένη στον κόμβο “cell”

5.2.5. Περιγραφή Υλοποίησης Κουμπιού Clear Flow

Ακόμη μια λειτουργία που παρέχει η εφαρμογή μας είναι ο μηδενισμός της ροής. Πολλές φορές μπορεί ο χρήστης να θέλει μηδενίσει τη ροή που υπάρχει στο δίκτυο ροής αφότου έτρεξε επιτυχώς τον αλγόριθμο Ford Fulkerson έτσι ώστε να μπορέσει τον ξανατρέξει ή μπορεί να θέλει να σταματήσει τον αλγόριθμο σαν τρέχει και να καθαρίσει την ροή που πρόσθεσε μέχρι στιγμής. Για αυτό τον λόγο υλοποιήσαμε το κουμπί Clear Flow.

```
public static void clearFlow(Graph graph,boolean
resStageOpened){

    for (Edge edge:graph.getModel().getAllEdges())
        edge.setFlow(0);

    for (Thread thread:BFS_Algorithm.threads)
        thread.stop();
}
```

Απόσπασμα Μηδενισμού Ροής – Μέθοδος Κουμπιού Clear Flow

Όπως φαίνεται και από το πιο πάνω κομμάτι κώδικα μηδενίζουμε την ροή κάθε ακμής και επίσης ελέγχουμε αν κάποιο thread τρέχει τον αλγόριθμο και το σταματούμε.

5.2.6. Περιγραφή Υλοποίησης Κουμπιού Back

Για να μπορεί ο χρήστης να μεταβαίνει στην σκηνή που ήταν προηγουμένως, υλοποιήθηκε το “Back Button”. Η υλοποίηση αυτού του κουμπιού ήταν σχετικά απλή. Έπρεπε απλά να αποθηκεύουμε σε μια δομή την προηγούμενη οθόνη που θα μας μεταφέρει το κουμπί “Back”.

```
primaryStage.close()
primaryStage.setScene(initialScreen);
primaryStage.setWidth(600);
primaryStage.setHeight(400);
primaryStage.show();
```

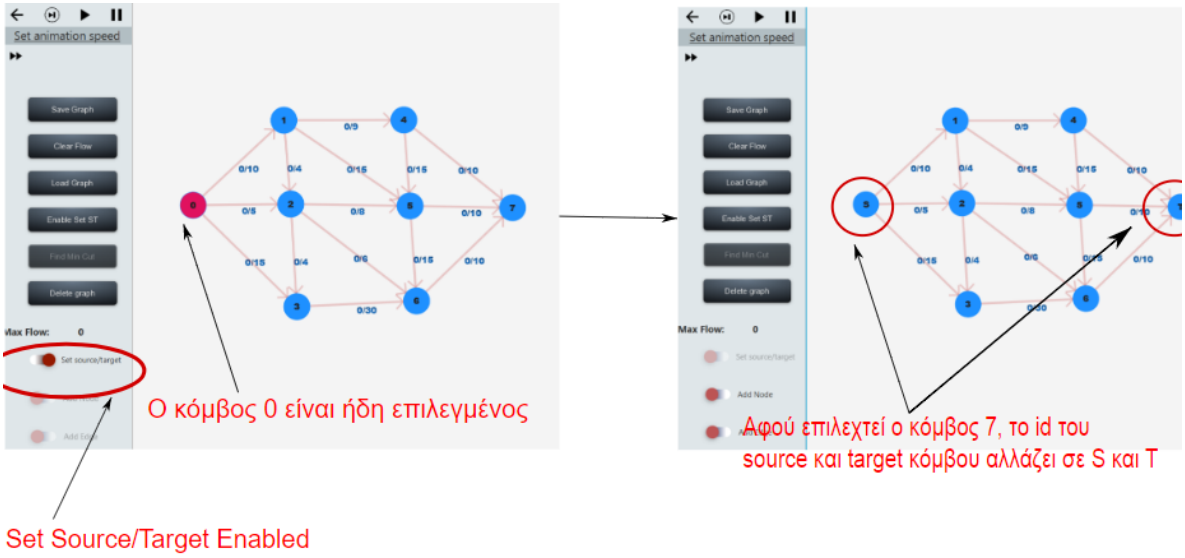
Απόσπασμα Υλοποίησης του κουμπιού Back :

Όπως βλέπουμε και από το πιο πάνω απόσπασμα αρχικά κλείνουμε την οθόνη που είναι ανοικτή, αλλάζουμε την σκηνή με αυτήν της προηγούμενης οθόνης, προσαρμόζουμε το ύψος και πλάτος που θέλουμε και στη συνέχεια εμφανίζουμε την νέα οθόνη.

5.2.7. Περιγραφή Υλοποίησης Κουμπιού Set Source/Target

Σε αυτή το κεφάλαιο θα περιγράψουμε τον ρόλο του συρόμενου κουμπιού “Set Source/Target”. Για να μπορεί κάποιος να δημιουργήσει ένα έγκυρο δίκτυο ροής πρέπει να μπορεί να θέσει κάποιους κόμβους ως πηγή και απόληξη σε ένα γράφημα. Το πρόγραμμα δίνει στον χρήστη τη δυνατότητα να θέσει τους κόμβους που αυτός επιθυμεί ως πηγή και απόληξη.

Επομένως όταν πατήσει αυτό το κουμπί πρέπει στη συνέχεια να κάνει κλικ πάνω σε δύο κόμβους. Ο 1ος κόμβος που θα κάνει κλικ είναι η πηγή και ο 2ος κόμβος που θα κάνει κλικ είναι η απόληξη. Ακολουθεί και εικονική αναπαράσταση της λειτουργίας που μόλις περιέγραψα.



Σχήμα 5.7 - Διαδικασία Προσθήκης κόμβων Source και Target

Η διαδικασία που μόλις περιέγραψα επιτυγχάνεται αλγοριθμικά σε μορφή ψευδοκώδικα με τον ακόλουθο τρόπο.

- Αν το toggle button source/target είναι ενεργοποιημένο και αν επιλεχτεί ένας κόμβος που δεν είναι ήδη επιλεγμένος αλλάζει το χρώμα του και αυξάνεται ο αριθμός των επιλεγμένων κόμβων. Αν όμως ο κόμβος που επιλέχτηκε είναι ήδη επιλεγμένος τότε απλά κάνε αποεπιλέγει(deselect) αυτό τον κόμβο.
- Αν ο αριθμός των επιλεγμένων κόμβων είναι ίσος με 2 τότε θέτει τον 1^{ο} επιλεγμένο κόμβο με “S” , δηλαδή τον θέτει ως την πηγή , και τον 2^{ο} επιλεγμένο κόμβο ως την απόληξη και στη συνέχεια απενεργοποιείται το toggle button “source/target”.

Ακολουθεί και η πραγματική υλοποίηση του πιο πάνω ψευδοκώδικα.

```
if (Main.add_source_targetToggleBtn.isSelected() &&
event.getButton()==MouseButton.PRIMARY &&
graph.getModel().allCells.size()>=2){


    if (!node.isSelected()){
        if (numberOfSelectedNodes <=1) {

            selectedCells.add(node);
            numberOfSelectedNodes++;


            if (numberOfSelectedNodes ==2){
                selectedCells.get(0).setLabel_id("S");
                selectedCells.get(0).set_is_start_node(true);
                selectedCells.get(1).setLabel_id("T");
                selectedCells.get(1).set_is_end_node(true);

                source_target_selected=true;
Main.add_source_targetToggleBtn.setDisable(true);

Main.add_source_targetToggleBtn.setSelected(false);
               unSelectNodes();
            }else {
                selectNode(node);
            }
        }
    }
}
```

Απόσπασμα Υλοποίησης Λειτουργίας Set Source – Target Button

5.2.8. Περιγραφή Υλοποίησης Μεγέθυνσης και Σμίκρυνσης με Mouse Scrolling

Σε αυτή το κεφάλαιο θα περιγράψουμε πως μπορεί ο χρήστης να μεγεθύνει και να σμικρύνει την οθόνη με τον τροχό κύλισης του ποντικιού του. Για να το επιτύχουμε αυτό προσθέσαμε ένα event handler πάνω στην οθόνη ο οποίος λαμβάνει την κίνηση του τροχού κύλισης του ποντικιού. Όποτε ο χρήστης κυλά τον τροχό του ποντικιού του προς τα μέσα τότε γίνεται zoom in η οθόνη και όποτε κυλά τον τροχό του ποντικιού προς τα έξω γίνεται zoom out.

```
private class ZoomHandler implements
EventHandler<ScrollEvent> {
    @Override
    public void handle(ScrollEvent scrollEvent) {
        /** if getDeltaY() less than zero
         * then user scrolled out , therefore zoom out.
         * if getDeltaY() greater than zero
         * then user scrolled in , therefore zoom in.
         */
        if (scrollEvent.getDeltaY() < 0) {
                scaleValue -= delta;
            } else {
                scaleValue += delta;
            }
            zoomTo(scaleValue);
            scrollEvent.consume();
    }
}
```

Απόσπασμα Υλοποίησης zoom in και zoom out όταν γίνεται scroll in και scroll out αντίστοιχα

5.3. Δημιουργία Υπολειπόμενου Δικτύου - Λεπτομέρειες Υλοποίησης

Λόγω του ότι ο κύριος σκοπός της εφαρμογής μας ήταν εκπαιδευτικός, το τρέξιμο του Ford Fulkerson μόνο πάνω στο δίκτυο ροής δεν ήταν αρκετό. Για αυτό το λόγο δώσαμε τη δυνατότητα να φαίνεται σε μια δεύτερη οθόνη το υπολειπόμενο δίκτυο, το οποίο θα ενημερώνεται αυτόματα ανάλογα με τις αλλαγές της ροής στο δίκτυο ροής με σκοπό την καλύτερη κατανόηση του χρήστη.

Η διαδικασία δημιουργίας ενός υπολειπόμενου δικτύου επιτυγχάνεται αλγοριθμικά σε μορφή ψευδοκώδικα με τον ακόλουθο τρόπο (Θυμηθείτε το Κεφάλαιο 3):

1. Δημιουργία μιας νέας σκηνής.
2. Προσθήκη των κόμβων του Δικτύου Ροής G στο υπολειπόμενο δίκτυο στη νέα σκηνή που δημιουργήσαμε προηγούμενος.
3. Για κάθε ακμή e= (u, v) μέσα στο Δίκτυο ροής G και έστω f_e η ροή αυτής της ακμής όπου $f_e < c_e$, προσθέτουμε στο G_f μια *ευθύδρομη* ακμή e=(u, v) με χωρητικότητα c_e-f_e.
4. Για κάθε ακμή e= (u, v) μέσα στο Δίκτυο ροής G και έστω f_e η ροή της της ακμής όπου $f_e > 0$, προσθέτουμε στο G_f μια *ανάδρομη* ακμή e=(v , u) με χωρητικότητα f_e.

Ακολουθεί και η πραγματική υλοποίηση του πιο πάνω ψευδοκώδικα:

Βήμα 1ο – Δημιουργία νέας σκηνής:

```
Scene residualScene = new Scene(root, scene.getWidth(),
scene.getHeight());
```

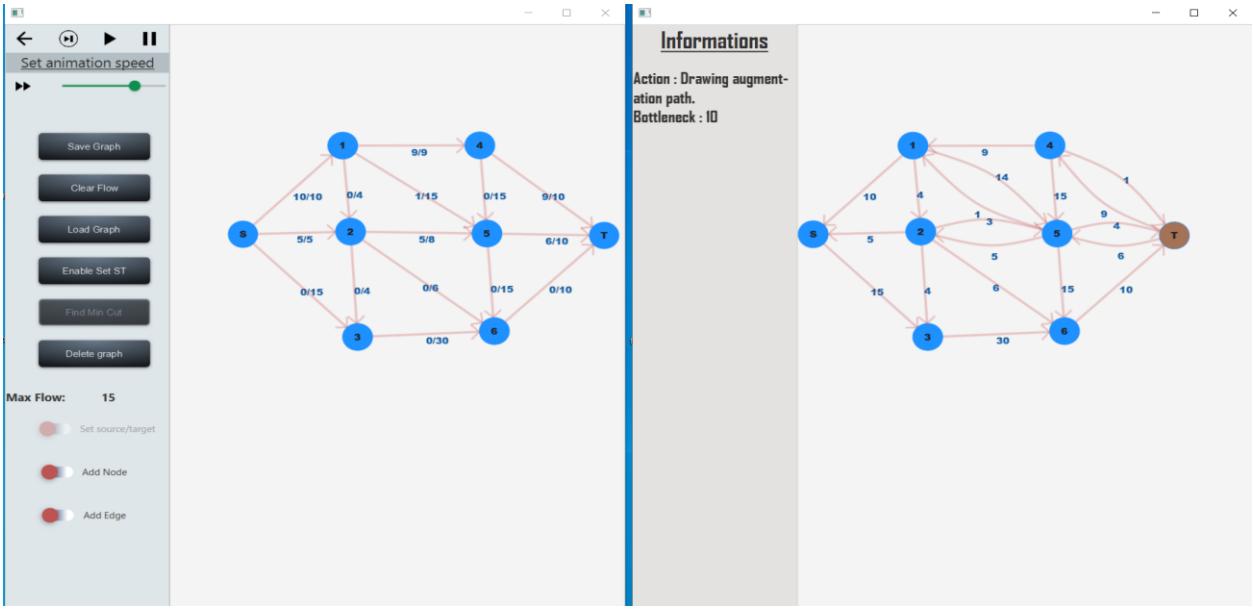
Βήμα 2ο – Προσθήκη κόμβων στο υπολειπόμενο δίκτυο:

```
for (Cell cell:graph.getModel().getAllCells()) {

double x = cell.getLayoutX();
double y = cell.getLayoutY();

residual.getModel().addCellResidual(cell.getCellId(),x, y,
newScene);
}
```

Πιο πάνω μπορούμε να δούμε πως οι κόμβοι που προσθέτουμε στο υπολειπόμενο δίκτυο είναι οι ίδιοι ακριβώς κόμβοι με αυτούς του δικτύου ροής (δηλαδή το graph.getModel().getAllCells()). Επίσης αξίζει να αναφερθεί πως οι συντεταγμένες των κόμβων του υπολειπόμενου δικτύου είναι ακριβώς οι ίδιες με αυτές του δικτύου ροής. Αυτός είναι και ο λόγος που στον πιο πάνω κώδικα αποθηκεύουμε αυτές τις συντεταγμένες στις μεταβλητές x και y. Επομένως, έχοντας πλέον τις συντεταγμένες μπορούμε εύκολα να προσθέσουμε ένα κόμβο στη νέα σκηνή χρησιμοποιώντας τον τρόπο που περιγράψαμε στο Κεφάλαιο 5.2.2.

Βήμα 3ο - 4ο – Προσθήκη ακμών στο υπολειπόμενο δίκτυο:

```
/** Για κάθε ακμή πάνω στο δίκτυο ροής*/
 for (Edge edge:graph.getModel().getAllEdges()){
     /** Προσθήκη Εμφυδρομής ακμής*/
     if (edge.getFlow()<edge.getCapacity()){
         Cell
source=residual.getModel().getCellByID(edge.getSource().getCe
llId());
         Cell
target=residual.getModel().getCellByID(edge.getTarget().getCe
llId());
         residual.addEdge(source,target,edge.getCapacity()-
edge.getFlow());
     }


     /** Προσθήκη Ανάδρομης ακμής*/
     if (edge.getFlow()>0){
         Cell
source=residual.getModel().getCellByID(edge.getSource().getCe
llId());
         Cell
target=residual.getModel().getCellByID(edge.getTarget().getCe
llId());
         residual.removeSelectedNodes(source,target);
         residual.addEdge(target,source,edge.getFlow());;
     }
 }
```



Σχήμα 5.8 – Παράδειγμα δημιουργίας υπολειπόμενου δικτύου

Στο πιο πάνω σχήμα, και πιο συγκεκριμένα στο 2ο παράθυρο μπορούμε να δούμε και το αποτέλεσμα της διαδικασίας δημιουργίας υπολειπόμενου δικτύου που μόλις περιέγραψα.

5.4. Έλεγχος Εγκυρότητας Δικτύου Ροής

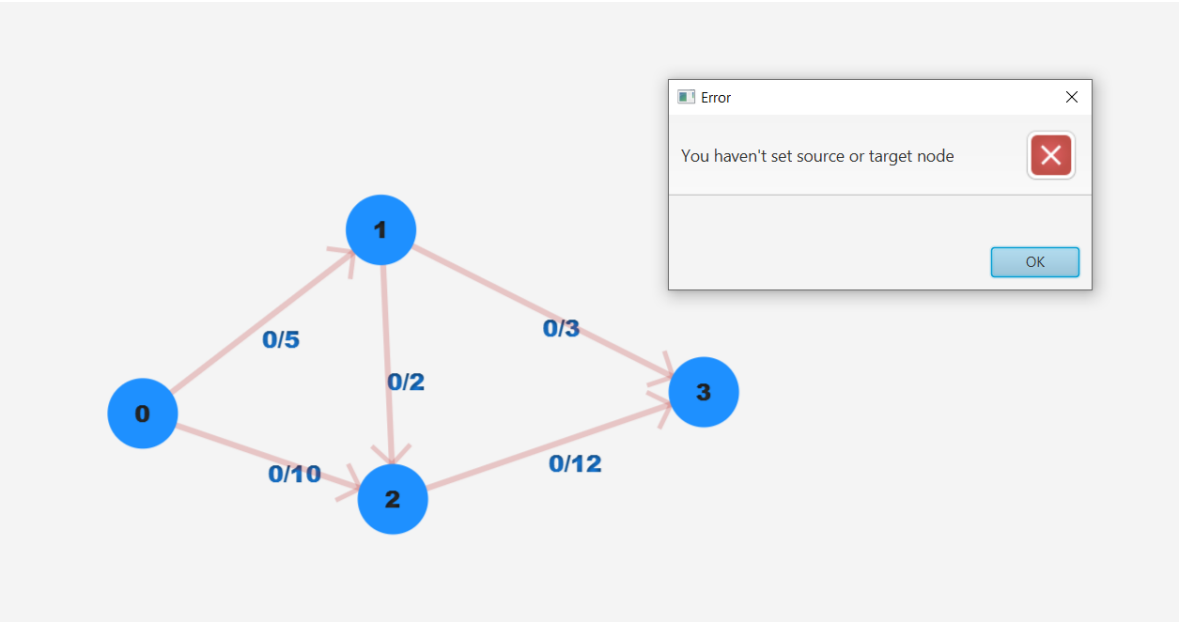
Κάθε φορά που ο χρήστης σχεδιάζει ένα δίκτυο ροής , πριν τρέξει τον αλγόριθμο γίνεται έλεγχος εγκυρότητας, δηλαδή ελέγχεται ότι το δίκτυο ροής που σχεδιάστηκε είναι ένα έγκυρο δίκτυο ροής. Οι έλεγχοι που γίνονται είναι οι πιο κάτω:

1. Έλεγχος ότι υπάρχει πηγή και απόληξη στο γράφημα.
2. Έλεγχος ότι υπάρχει μονοπάτι από την πηγή στην απόληξη
3. Έλεγχος ότι κάθε κόμβος έχει τουλάχιστον μια εισερχόμενη ή εξερχόμενη ακμή

Υλοποίηση 1ου ελέγχου:

```
for(Cell cell:graph.getModel().getAllCells()){
    CircleCell cell1=(CircleCell) cell;
    if (cell1.getLabel_id().equals("S"))
        foundSource=true;
    if (cell1.getLabel_id().equals("T"))
        foundTarget=true;
}
if (!foundSource && !foundTarget) {
    Alert alert=new Alert(Alert.AlertType.ERROR);
    alert.setHeaderText("You haven't set source or target
node");
    alert.show();

    return false;
}
```

Με αυτό τον τρόπο αν ο χρήστης προσπαθήσει να τρέξει τον αλγόριθμο σε ένα γράφημα που δεν έχει πηγή ή απόληξη τότε θα πάρει το πιο κάτω μήνυμα.

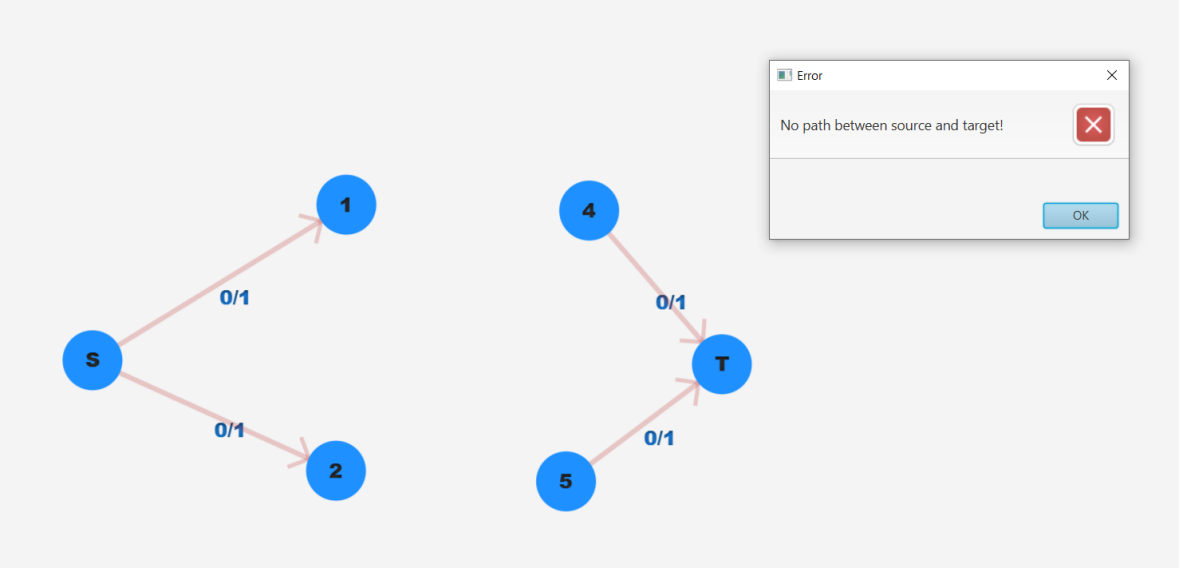


Σχήμα 5.9 – Μήνυμα λάθους για μη καθορισμό πηγής ή απόληξης

Υλοποίηση 2ου ελέγχου:

```
if(!BFS_Algorithm.run_BFS(graph.getSource(),graph.getTarget()
,graph)){
    Alert alert=new Alert(Alert.AlertType.ERROR);
    alert.setHeaderText("No path between source and
target!");
    alert.show();
    graph.unmarkNodes();
    return;
}
```

Ουσιαστικά τρέχει τον αλγόριθμο BFS πάνω στο δίκτυο ροής και αν δεν βρεθεί μονοπάτι από την πηγή στην απόληξη τότε εμφανίζεται το πιο κάτω μήνυμα.

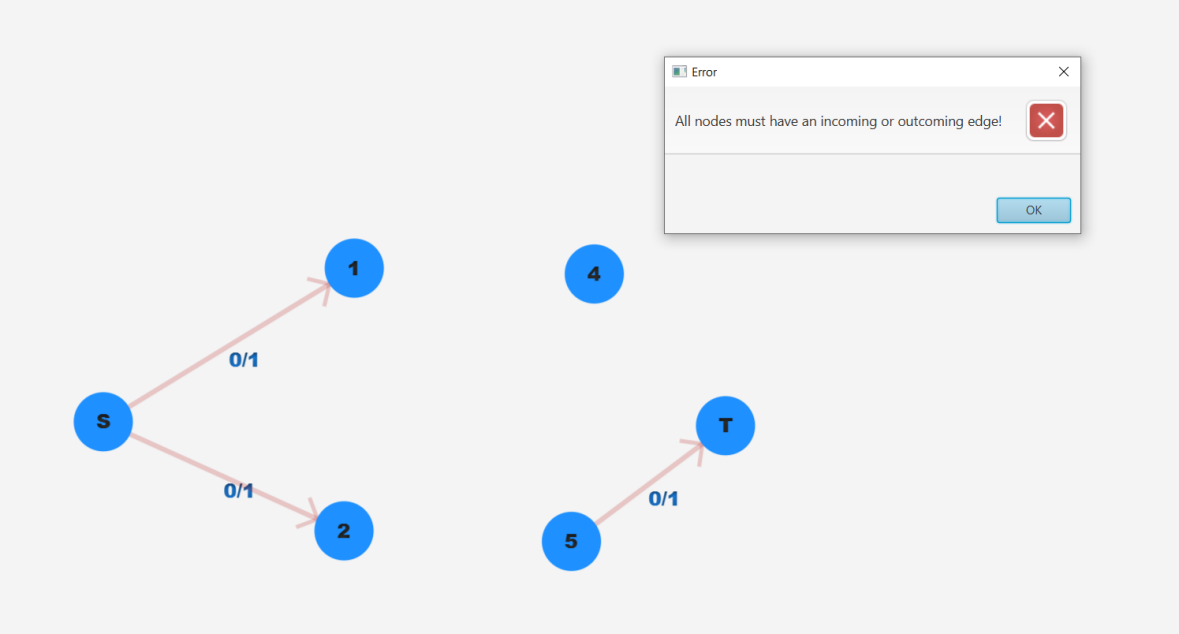


Σχήμα 5.10 – Μήνυμα λάθος για μονοπάτι μεταξύ πηγής και απόληξης

Υλοποίηση 3ου ελέγχου :

```
for (Cell cell:graph.getModel().getAllCells()){
    boolean foundEdge=false;
    String id=cell.getCellId();
    for (Edge edge:graph.getModel().getAllEdges()){
        if (edge.getSource().getCellId().equals(id) ||
edge.getTarget().getCellId().equals(id))
            foundEdge=true;
    }
    if (!foundEdge){
        Alert alert=new Alert(Alert.AlertType.ERROR);
        alert.initOwner(primaryStage);
        alert.setHeaderText("All nodes must have an incoming
or outcoming edge!");
        if(controller.playButton!=null)
            controller.playButton.setDisable(false);
        if(controller.playFastButton!=null)
            controller.playFastButton.setDisable(false);
        alert.show();
        return;
    }
}
```

Εδώ ελέγχουμε αν όλοι οι κόμβοι πάνω στο δίκτυο ροής που σχεδίασε ο χρήστης έχουν τουλάχιστον μια εισερχόμενη ή εξερχόμενη ακμή. Αν όχι τότε τυπώνεται το πιο κάτω μήνυμα.



Σχήμα 5.11 – Μήνυμα λάθους για κόμβο χωρίς εισερχόμενη ή εξερχόμενη ακμή.

5.5. Αναπαράσταση και Εκτέλεση Αλγορίθμου με χρήση Κινούμενων Σχεδίων

Γενικά, η κίνηση ενός αντικειμένου, συνεπάγεται τη δημιουργία ψευδαίσθησης της κίνησης του αντικειμένου μέσω γρήγορης εναλλαγής πολλαπλών εικόνων. Στο JavaFX ένα αντικείμενο μπορεί να κινείται αλλάζοντας τις ιδιότητες του με την πάροδο του χρόνου, δίνοντας με αυτό τον τρόπο την ψευδαίσθηση της κίνησης του αντικειμένου. Το JavaFX παρέχει ένα πακέτο κλάσεων με το όνομα javafx.animation. Αυτό το πακέτο περιέχει κλάσεις που χρησιμοποιούνται για την κίνηση των κόμβων. Επομένως με το JavaFX μπορούμε να δημιουργήσουμε animations ή αλλιώς μεταβάσεις όπως τα ονομάζει η JavaFX όπως το FillTransition, Parallel Transition και Sequential Transition. Τα πιο πάνω είναι και τα Transitions που χρησιμοποιήθηκαν στο προγραμματιστικό κομμάτι για τους σκοπούς της διπλωματικής.

Για να κάνουμε ένα αντικείμενο στο JavaFX να μετακινείτε πρέπει να ακολουθήσουμε τα πιο κάτω βήματα.

- Αρχικοποίηση του αντίστοιχου αντικειμένου μετάβασης (π.χ FillTransition, Stroke Transition).
- Θέτουμε τις ιδιότητες του αντικειμένου μετάβασης.
- Παίζουμε το Animation χρησιμοποιώντας την μέθοδο play() της κλάσης Animation().

Σε αυτή το κεφάλαιο θα περιγράψω τις μεταβάσεις που χρησιμοποιήθηκαν στην διπλωματική εργασία.

FillTransition Animation:

Η Transition Class δημιουργεί ένα κινούμενο σχέδιο, το οποίο αλλάζει τη γέμιση (δηλαδή το χρώμα) ενός σχήματος σε μια καθορισμένη διάρκεια. Αυτό ήταν ιδιαίτερα χρήσιμο για την διπλωματική μου διότι μια από τις ανάγκες που είχα στην διπλωματική

εργασία ήταν ο χρωματισμός τόσο των κόμβων αλλά και των ακμών του γράφου με χρήση κινούμενων σχεδίων, έτσι ώστε να μπορεί να σχεδιάζεται το μονοπάτι επαύξησης.

```
Circle u=(Circle)node.getView();
FillTransition ft = new FillTransition(Duration.millis(time),
u);
ft.play();
```

Απόσπασμα υλοποίησης του FillTransition :

Το πιο πάνω κόμματι κώδικα δείχνει πως σχεδιάζουμε μεμονωμένα ένα οποιοδήποτε κόμβο με χρήση animation και στη συνέχεια πως παίζουμε αυτό το animation. Τι γίνεται όμως αν θέλουμε να σχεδιάσουμε πολλαπλούς κόμβους ακολουθώντας μια λογική σειρά, π.χ πρώτα να σχεδιάσουμε τον κόμβο Α, μετά τον κόμβο Β κοκ. Κάποιος θα μπορούσε να εισηγηθεί την πιο κάτω διαισθητική προσέγγιση .

```
FillTransition ft1 = new
FillTransition(Duration.millis(time), u);
FillTransition ft2 = new
FillTransition(Duration.millis(time), u);
ft1.play();
ft2.play();
```

Το πρόβλημα όμως είναι ότι η μέθοδος play() της κλάσης animation είναι ασύγχρονή. Δηλαδή η μέθοδος play() δημιουργεί ένα καινούργιο νήμα το οποίο εκτελεί το ft1.play() και η διεργασία συνεχίζει στην εκτέλεση της επόμενης εντολής που είναι η ft2.play(). Αυτό έχει σαν αποτέλεσμα τα δυο animation να εκτελούνται απροσδιόριστα (λογικό διότι δεν μπορούμε να γνωρίζουμε πότε ένα thread θα τρέξει). Αυτό το πρόβλημα μπορεί να επιλυθεί με δυο τρόπους. Είτε να κάνουμε την ασύγχρονο κλήση σύγχρονη, δηλαδή πρώτα να τελειώνει το 1^{ο} animation και μετά να καλείτε η επόμενη play() μέθοδος, δηλαδή πρώτα να τελειώνει το ft1.play() και μετά να καλείται το ft2.play(). Είτε να χρησιμοποιήσουμε ένα

άλλο είδος μετάβασης που μας παρέχει η JavaFX και ονομάζεται SequentialTransition. Αυτό το transition παίζει μια λίστα από κινούμενα σχέδια(Animations) διαδοχικά. Δηλαδή παίζει το 1ο κινούμενο σχέδιο τελειώνει μαζί του, μετά παίζει το 2ο κοκ.

```
SequentialTransition sequential_transition=new
SequentialTransition();
for (Cell vertexV=resTarget;
!vertexV.getCellId().equals(resSource.getCellId());
vertexV=vertexV.getPreviousNodeInPath()){



    Circle node=(Circle)vertexV.getView();
    FillTransition ft = new
FillTransition(Duration.millis(time), node);
    ft.setToValue(Color.CHOCOLATE);
    sequential_transition.getChildren().add(ft);


    Cell vertexU=vertexV.getPreviousNodeInPath();
    Edge
edge=residual.getModel().getEdgeByUniqueID(vertexU,vertexV);
    MyLine line=(MyLine)vertexV.getView();
    FillTransition ft = new
FillTransition(Duration.millis(time), line);
    ft.setToValue(Color.CHOCOLATE);
    sequential_transition.getChildren().add(ft);
}
sequential_transition.play();
```

Απόσπασμα υλοποίησης Animation για διαδρομή επαύξησης με Sequential Transition.

Η sequential transition μέθοδος όμως αντιμετωπίζει το πρόβλημα που αναφέραμε προηγούμενος. Η play() μέθοδος της είναι ασύγχρονη. Επομένως ναι μεν θα παίξει τα διάφορες μεταβάσεις/κινούμενα σχέδια που θα της δοθούν ακολουθιακά αλλά η play() μέθοδος δεν θα περιμένει να τελειώσουν τα κινούμενα σχέδια και μετά να συνεχίσει την εκτέλεση του προγράμματος και αυτό αποτελούσε πρόβλημα στην περίπτωση μας, διότι μόλις τελείωναν τα κινούμενα σχέδια θέλαμε να κάνουμε κάποιες αλλαγές πάνω στο δίκτυο ροής. Επομένως έπρεπε να βρεθεί κάποιος τρόπος να κάνω την ασύγχρονη κλήση σύγχρονη.

Η διαδικασία μετατροπής της ασύγχρονης κλήσης σε σύγχρονη επιτυγχάνεται αλγοριθμικά σε μορφή ψευδοκώδικα με τον ακόλουθο τρόπο:

- Δημιουργία ενός καινούργιου νήματος που θα του δίνεται ως είσοδος ένα κινούμενο σχέδιο και θα κάνει τα τρια πιο κάτω βήματα.
- Θέσε ένα event handler στο animation που θα τρέξεις ο οποίος θα ενεργοποιείται μόλις τελειώσει το κινούμενο σχέδιο. Αυτό που κάνει η “handle” μέθοδος στο πιο κάτω απόσπασμα κώδικα, είναι ότι μόλις τελειώσει το κινούμενο σχέδιο καλεί την αρχική “handle” μέθοδο του κινούμενο σχεδίου και ακολούθως ξυπνά το νήμα που την καλεί με την μέθοδο notify().

```
final Thread currentThread = Thread.currentThread();
final EventHandler<ActionEvent> originalOnFinished =
animation.setOnFinished(new EventHandler<ActionEvent>(){
    public void handle(ActionEvent event) {
        if (originalOnFinished != null) {
            originalOnFinished.handle(event);
        }
        synchronized (currentThread) {
            currentThread.notify();
        }
    }
});
```

- Τρέξε το animation δημιουργώντας ένα καινούργιο νήμα που θα αναλάβει την εκτέλεση του animation

```
Platform.runLater(new Runnable() {

    @Override
    public void run() {
        animation.play();
    }
});
```

- Ανάστειλε την λειτουργεία του νήματος μέχρι κάποιο να το ξυπνήσει. Αυτό θα συμβεί με την μέθοδο notify() του 1ου bullet μόλις τελειώσει η εκτέλεση του animation από το 2ο νήμα που δημιουργήθηκε στο bullet 2.

```
synchronized (currentThread) {
    try {
        currentThread.wait();
    } catch (InterruptedException ex) {
        //somebody interrupted me, OK
    }
}
```

Αρά με αυτό τον τρόπο καταφέραμε να κάνουμε μια ασύγχρονη κλήση , δηλαδή το παίξιμο του κινουμένου σχεδίου σύγχρονο. Αξίζει να αναφέρουμε ότι το πιο πάνω ήταν από τα πιο σημαντικά βήματα στην επιτυχή ολόκληρη της διπλωματικής εργασίας διότι μας επέτρεψε να αφήσουμε την λογική του κώδικα ακολουθιακή και να κρατήσουμε την υλοποίηση μας απλή χωρίς να χρειαστεί ασχοληθούμε με συγχρονισμό νημάτων.

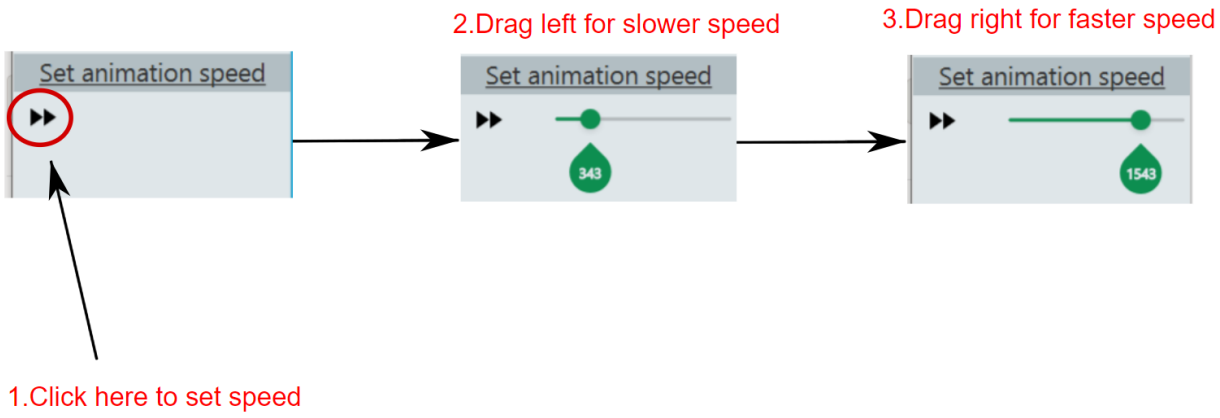
5.5.1. Περιγραφή Υλοποίησης Ρύθμισης Ταχύτητας του Animation

Όπως είδαμε προηγούμενος όταν ο χρήστης δημιουργεί κινούμενα σχέδια χρειάζεται να δώσει και την ταχύτητα με την οποία θα εκτελεστούν όπως φαίνεται και στο τονισμένο κομμάτι του πιο κάτω κώδικα.

Απόσπασμα Δημιουργίας Animation:

```
FillTransition ft = new FillTransition( Duration.millis(time) , u);
ft.play();
```

Για την υλοποίηση αυτής της απαίτησης δημιουργήσαμε το πιο κάτω κουμπί με το οποίο μπορούμε να ρυθμίσουμε την ταχύτητα.



Σχήμα 5.12 – Διαδικασία Ρύθμισης της ταχύτητας

Αρχικά πατώντας το κουμπί θα εμφανιστεί μια μπάρα. Όσο πιο αριστερά σύρουμε την μπάλα στη μπάρα τόσο πιο αργό γίνεται το κινούμενο σχέδιο και όσο πιο δεξιά την σύρουμε τόσο πιο γρήγορο γίνεται. Για να ενημερώνεται η ταχύτητα οποτεδήποτε ο χρήστης αλλάξει την ταχύτητα μέσω του κουμπιού, προσθέσαμε ένα event listener που ενεργοποιείται οποτεδήποτε υπάρξει κάποια αλλαγή πάνω στο κουμπί ολίσθησης(“slide button”) και στη συνέχεια ενημερώνει αμέσως την μεταβλητή time, η οποία μεταβλητή είναι μια global μεταβλητή και χρησιμοποιείται για τις ταχύτητες όλων των Animation.

Επεξήγηση Διαδικασίας Αλλαγής Ταχύτητας Animation μέσω του Κουμπιού:

Προσθήκη Event Listener πάνω στο κουμπί:

```
slider.valueProperty().addListener(changed);
```

Μέθοδος changed που καλείται όταν υπάρξουν αλλαγές στο κουμπί slider:

```
public void changed(ObservableValue observable, Object
oldValue, Object newValue) {
    int temp = (int) slider.getValue();

    if (temp > 1000) {
        int diff = temp - 1000;
        temp = 1000;
        temp -= diff;
        temp += 10;
    } else if (temp < 1000) {
        int diff = 1000 - temp;
        temp = 1000;
        temp += diff;
        temp -= 10;
    }
    BFS_Algorithm.setTime(temp);
    BFS_Algorithm_Stepwise.setTime(temp);

}
```

Όπως μπορούμε να δούμε και από την πιο πάνω μέθοδο όσο πιο μεγάλη είναι η τιμή του slider τόσο πιο μικρή θα είναι η τιμή της μεταβλητής temp και επομένως πιο μεγάλη και η ταχύτητα του animation. Επίσης μπορούμε να δούμε πως στο τέλος ενημερώνεται και η global μεταβλητή time μέσω των μεθόδων setTime().

5.5.2. Περιγραφή Υλοποίησης του Κουμπιού Play

Σε αυτή το κεφάλαιο θα εξηγηθεί σε λεπτομέρεια τι γίνεται στο παρασκήνιο όταν ο χρήστης πατήσει το κουμπί “play” για να ξεκινήσει το κινούμενο σχέδιο του Ford Fulkerson.

Αφού ο χρήστης σχεδιάσει ένα έγκυρο δίκτυο ροής μπορεί να πατήσει το κουμπί “play” για να ξεκινήσει να τρέχει ο αλγόριθμος Ford Fulkerson. Τα βήματα που εκτελούνται στο παρασκήνιο είναι τα πιο κάτω:

1. Δημιουργία υπολειπόμενου γραφήματος βάση υπάρχοντος δικτύου ροής όπως εξηγήθηκε στο Κεφάλαιο 5.3.
2. Εύρεση μονοπατιού επαύξησης πάνω στο υπολειπόμενο δίκτυο. Αν δεν υπάρχει μονοπάτι τότε τέλειωσε ο αλγόριθμος και βρέθηκε η μέγιστη ροή.
3. Αύξηση ροής στο δίκτυο ροής βάση της διαδρομής επαύξησης που βρέθηκε στο βήμα 2.
4. Πήγαινε στο βήμα 1

Σημείωση: Τα πιο πάνω βήματα εκτελούνται επαναληπτικά μέχρι το τέλος του αλγορίθμου.

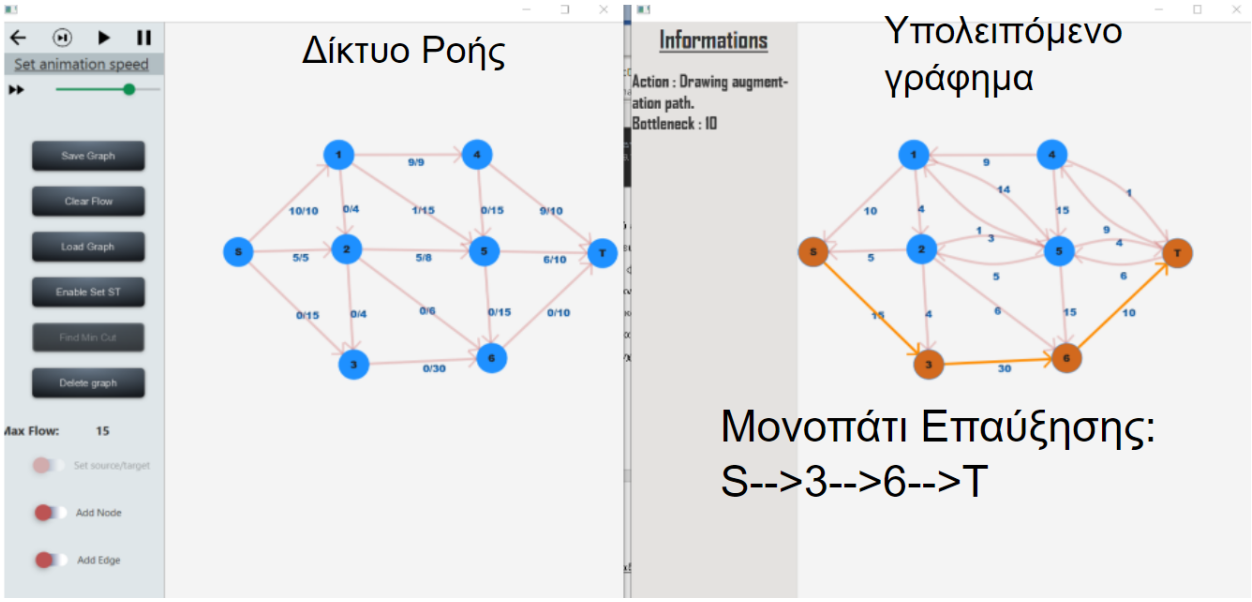
Η υλοποίηση του 1^{ου} βήματος εξηγήθηκε αναλυτικά στο Κεφάλαιο 5.3 και για αυτό παραλείπονται οι λεπτομέρειες υλοποίησης της σε αυτό το σημείο.

Βήμα 2ο - Εύρεση μονοπατιού επαύξησης πάνω στο υπολειπόμενο δίκτυο

```
public static boolean run_BFS(Cell source, Cell target,Graph
graph) {

    LinkedList<Cell> queue = new LinkedList<Cell>();
    queue.add(source);
    source.markVisited();
    SequentialTransition monopati_epavksisis=new
SequentialTransition();
    /** Run BFS on residual graph*/
    while(!queue.isEmpty()) {
        Cell temp = queue.poll();
        if (temp.getCellId().equals(target.getCellId())) {
            break;
        }
        if (temp.is_start_node() ){
            Circle u=(Circle)temp.getView();


        }
        for(Cell n : temp.children) {
            if(!n.isVisited()) {
                n.markVisited();
                n.setPreviousNodeInPath(temp);
                queue.add(n);
            }
        }
    }
    /** Find augmentation Path */
    Cell node=target;
    while (node!=null){
```

```
        Circle u=(Circle)node.getView();
        FillTransition ft = new
FillTransition(Duration.millis(time), u);
        ft.setToValue(Color.CHOCOLATE);
        monopati_epavksisis.getChildren().add(ft);
        node=node.getPreviousNodeInPath();
    }
    animationLinkedList.add(monopati_epavksisis);
    playAnimationAndWaitForFinish(monopati_epavksisis);
    return source.isVisited();

}
```

Όπως μπορούμε να δούμε η εύρεση του μονοπατιού επαύξησης είναι μια σχετικά απλή διαδικασία. Αρχικά τρέχουμε τον αλγόριθμο BFS πάνω στο υπολειπόμενο δίκτυο. Στη συνέχεια ξεκινώντας από τον κόμβο απόληξης πηγαίνουμε προς τα πίσω μέχρι να φτάσουμε στον κόμβο πηγή. Οι κόμβοι που συναντούμε σε αυτή την φάση είναι οι κόμβοι που ανήκουν στο μονοπάτι επαύξησης, αυτός είναι και ο λόγος που δημιουργείται ένα FillTransition για κάθε κόμβο που συναντούμε. Κάθε FillTransition προστίθεται σε ένα SequentialTransition έτσι ώστε να δημιουργηθεί το κινούμενο σχέδιο για το μονοπάτι επαύξησης. Στη συνέχεια παίζουμε το animation σύγχρονα όπως εξηγήσαμε στο Κεφάλαιο 5.4. Όπως μπορούμε να δούμε και από το πιο κάτω σχήμα το μονοπάτι επαύξησης που βρίσκουμε από το βήμα 2, και φαίνεται με πορτοκαλί χρώμα είναι το S-->3-->6-->T.



Σχήμα 5.13: Δίκτυο ροής και το αντίστοιχο υπολειπόμενο γράφημα με το μονοπάτι επαύξησης

Σημείωση: Η διαδικασία runBFS(….) που είδαμε στο βήμα 2 επιστρέφει στο τέλος αν ο κόμβος source είναι “visited”, η με άλλα λόγια αν υπάρχει μονοπάτι μεταξύ της πηγής και της απόληξης. Αν υπάρχει επιστρέφει “true”, αλλιώς επιστρέφει “false”. Ο λόγος που επιστρέφει true αν υπάρχει μονοπάτι στο υπολειπόμενο δίκτυο είναι διότι αυτή η μέθοδος είναι η συνθήκη για τον τερματισμό του βρόγχου που εκτελείται ο αλγόριθμος Ford Fulkerson όπως φαίνεται και από τον πιο κάτω ψευδοκώδικα:

```
while (run_BFS(source_node,target_node,residual_graph)) {
        execute_step_three(..)
}
```

Ψευδοκώδικας που εκτελείτε κάθε φορά που θα πατηθεί το κουμπί play.

Βήμα 3ο - Αύξηση ροής στο δίκτυο ροής βάση της διαδρομής επαύξησης:

Αφότου βρεθεί μια διαδρομή επαύξησης πάνω στο υπολειπόμενο δίκτυο, μένει να αυξηθεί η ροή πάνω στο δίκτυο ροής βάση της ελάχιστης υπολειπόμενης χωρητικότητας της διαδρομής επαύξησης. Στο πιο πάνω μονοπάτι επαύξησης η ελάχιστη υπολειπόμενη χωρητικότητα είναι 10=min{10,30,15}.

Όπως φαίνεται και από το πιο πάνω σχήμα οι ακμές του μονοπατιού επαύξησης πάνω στο υπολειπόμενο δίκτυο είναι οι S->3,3->6 και 6->T. Σε αυτές ακριβώς τις ακμές θα αυξηθεί η ροή πάνω στο δίκτυο ροής κατά 10 μονάδες.

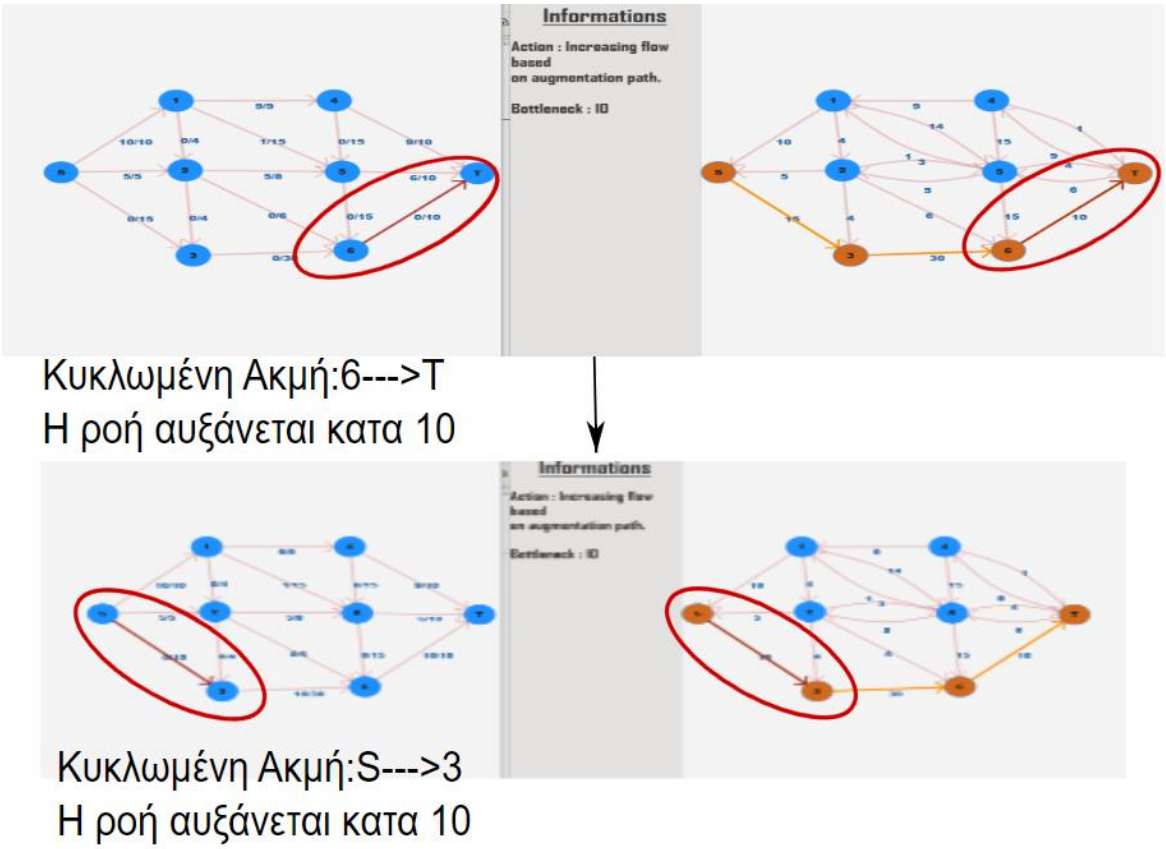
Για την υλοποίηση της επαύξησης χρησιμοποιήθηκε ο πιο κάτω ψευδοκώδικας.

```
augment(f, P)
  Let b = bottleneck(P, f)
  For each edge (u, v) ∈ P
    If e = (u, v) is a forward edge then
      increase f(e) in G by b

    Else ((u, v) is a backward edge, and let e = (v, u))
      decrease f(e) in G by b
    Endif
  Endfor
  Return(f)
```

Σχήμα 5.14 – Ψευδοκώδικας Ford Fulkerson

Ουσιαστικά αυτό που κάνει, είναι ότι για κάθε ακμή **e=(u,v)** πάνω στο μονοπάτι επαύξησης του υπολειπόμενου δικτύου, ελέγχει αν η ακμή είναι *ευθύδρομη* (δηλαδή υπάρχει η ίδια ακμή **e=(u,v)** στο δίκτυο ροής) και αν είναι *ευθύδρομη* προσθέτει bottleneck μονάδες ροής στην ακμή στο δίκτυο ροής, διαφορετικά αν είναι *ανάδρομη*, δηλαδή δεν υπάρχει η ακμή

e=(u,v) πάνω στο δίκτυο ροής , αφαιρεί την ελάχιστη υπολειπόμενη χωρητικότητα από την ακμή **e=(v,u)**. Η αλγοριθμική υλοποίηση της πιο πάνω διαδικασίας είναι αρκετά απλή και επομένως δεν θα δείξω την υλοποίηση της σε πραγματικό κώδικα.



Σχήμα 5.15 - Διαδικασία Επαύξησης στο Δίκτυο ροής

Η διαδικασία επαύξησης στην εφαρμογή μας φαίνεται στο πιο πάνω σχήμα. Όπως μπορούμε να δούμε όταν μια ακμή στο δίκτυο ροής πρόκειται να αυξηθεί η ροή της, τονίζεται το χρώμα τόσο της ακμή πάνω δίκτυο ροής αλλά και στο υπολειπόμενο δίκτυο. Θέλουμε το κινούμενο σχέδιο του τονισμού του χρώματος των 2 ακμών να γίνεται παράλληλα. Το παράλληλο κινούμενο σχέδιο υλοποιήθηκε χρησιμοποιώντας το ParallelTransition του JavaFX το οποίο αναλαμβάνει να εκτελέσει παράλληλα τα δυο κινούμενα σχέδια που θα του δοθούν. Η υλοποίηση είναι αρκετά παρόμοια με την υλοποίηση του SequentialTransition που εξηγήθηκε στο Κεφάλαιο 5.4.

```
/** Διαδικασία Επαύξησης */
for (Cell vertexV=resTarget;!vertexV.getCellId().equals(resSource.getCellId());vertexV=vertexV.getPreviousNodeInPath()){
    Cell vertexU=vertexV.getPreviousNodeInPath();
    Edge res_edge=residual.getModel().getEdgeByUniqueID(vertexU,vertexV);

    if (residual.is_forward_edge(res_edge)){
        Edge flow_edge=graph.getModel().getEdgeByUniqueID(res_edge.getSource(),res_edge.getTarget());
        SequentialTransition temp=new SequentialTransition();
        ParallelTransition parallelTransition;
        temp.onFinishedProperty();
        int x=bottleneckFlow;
        parallelTransition=new ParallelTransition();
        GraphUtil.addEdgeLineTransition(time,flow_edge,parallelTransition,Color.DARKRED);
        if (res_edge.line!=null)
            GraphUtil.addEdgeLineTransition(time,res_edge,parallelTransition,Color.DARKRED);
        if (res_edge.cubicCurveLine!=null)
            GraphUtil.addEdgeCubicTransition(time,res_edge,parallelTransition,Color.DARKRED);
        temp.getChildren().add(parallelTransition);
        parallelTransition=new ParallelTransition();
        GraphUtil.addEdgeLineTransition(time,flow_edge,parallelTransition,Color.rgb( i: 180,  i1: 13,  i2: 13,  v: 0.2));
        if (res_edge.line!=null)
            GraphUtil.addEdgeLineTransition(time,res_edge,parallelTransition,Color.DARKORANGE);
        if (res_edge.cubicCurveLine!=null)
            GraphUtil.addEdgeCubicTransition(time,res_edge,parallelTransition,Color.DARKORANGE);
        temp.getChildren().add(parallelTransition);
        temp.setOnFinished(actionEvent -> {
            flow_edge.setFlow(flow_edge.getFlow()+x);
        });
        animationLinkedList.add(temp);
        playAnimationAndWaitForFinish(temp);
```

Απόσπασμα Υλοποίησης Βήματος 3

Στο πιο πάνω απόσπασμα μπορούμε να δούμε τις λεπτομέρειες υλοποίησης για την διαδικασία επαύξησης.

5.5.3. Περιγραφή Υλοποίησης Next Step Button

Λόγω του ότι ο κύριος σκοπός της εφαρμογής μας είναι παιδαγωγικός, ιδιαίτερη έμφαση δόθηκε στο παιδαγωγικό κόμματι της εφαρμογής. Για τον λόγο αυτό αναπτύχθηκε μια τρίτη επιλογή, η Stepwise επιλογή της εφαρμογής. Η βασική διαφορά αυτής της επιλογής είναι ότι επιτρέπει στον χρήστη να τρέξει τον αλγόριθμο βήμα-βήμα. Δηλαδή κάθε φορά που πατά ο χρήστης το κουμπί “next”, θα τρέχει η επόμενη επανάληψη του αλγόριθμου,

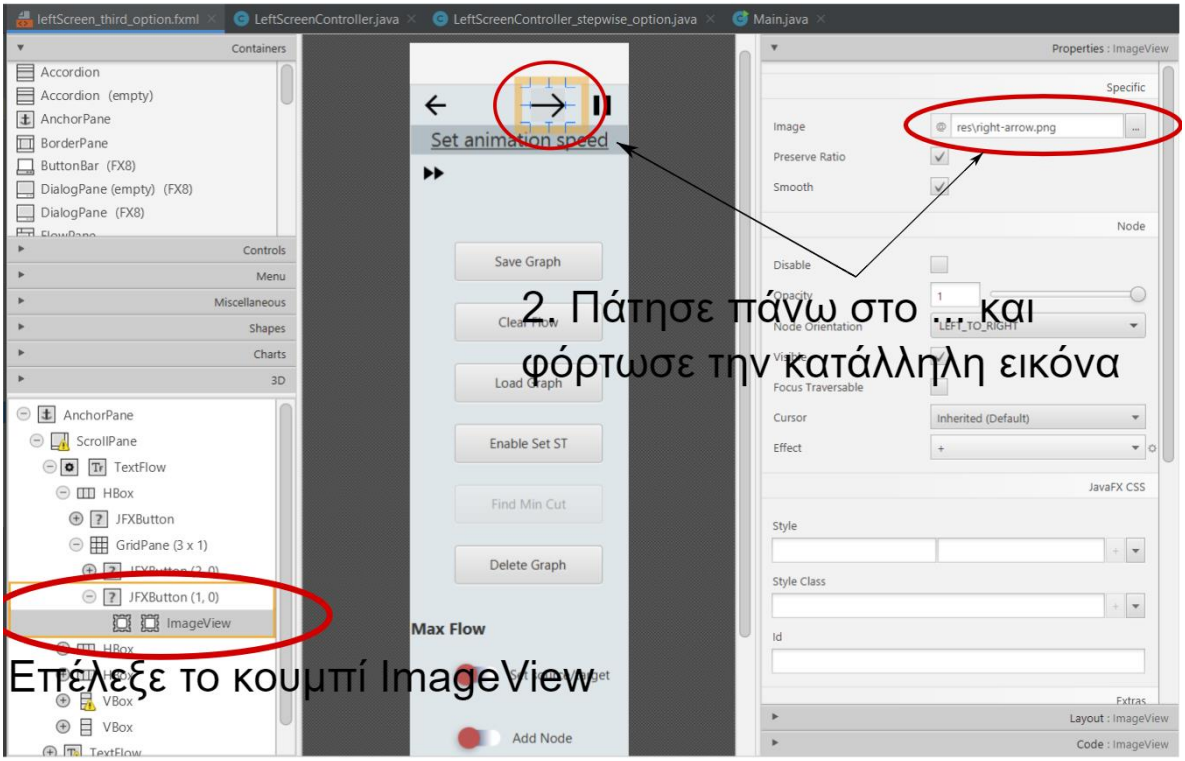
δηλαδή θα βρίσκεται ένα μονοπάτι επαύξησης στο υπολειπόμενο δίκτυο, θα αυξάνεται η ροή στο δίκτυο ροής και θα σταματά εκεί και δεν θα προχωρά στην επόμενη επανάληψη μέχρι ο χρήστης να πατήσει ξανά το κουμπί “next”.

Οι τροποποιήσεις που έγιναν για την εκτέλεση του αλγόριθμου βήμα-βήμα με χρήση animation είναι οι πιο κάτω:

- Αλλαγή Διεπαφής του χρήστη έτσι ώστε να αντικατασταθεί το κουμπί “play” από το κουμπί “next”.
- Αλγοριθμικές αλλαγές έτσι ώστε ο αλγόριθμος να τρέχει βήμα-βήμα.

Αλλαγή Διεπαφής:

Η αλλαγή διεπαφής ήταν σχετικά εύκολη διαδικασία διότι το μόνο που έπρεπε να αλλάξει σε σχέση με πριν ήταν το εικονίδιο του κουμπιού από “play” σε “next”. Αυτό μπορούσε εύκολα να γίνει μέσω του εργαλείου Scene Builder. Ουσιαστικά δημιουργήσαμε ένα καινούργιο FXML δημιουργώντας ένα αντίγραφου του υφιστάμενου FXML που έχουμε για τη κύρια διεπαφή που δείξαμε προηγουμένως. Ακολούθως το ανοίξαμε μέσω του Scene Builder και ακολουθήσαμε τα βήματα που φαίνονται στο πιο κάτω σχήμα.



Σχήμα 5.16 – Διαδικασία Αλλαγής Εικόνας Κουμπιού

Όπως βλέπουμε ήταν μια πολύ απλή τροποποίηση της είδη υπάρχων διεπαφής.

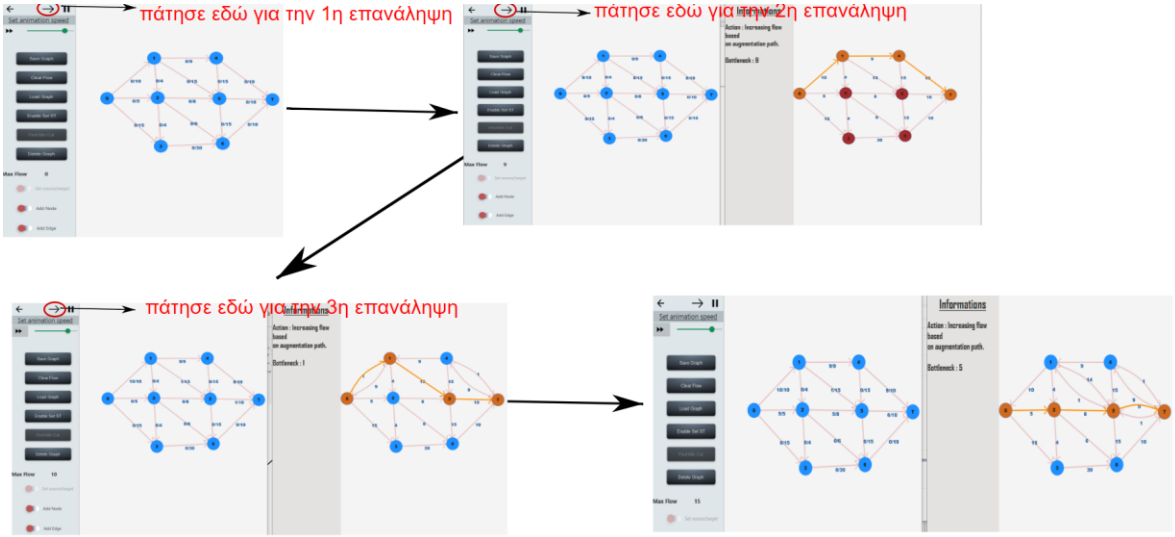
Αλγοριθμικές Αλλαγές :

Ο αλγόριθμος για την υλοποίηση του Stepwise Ford Fulkerson μοιάζει σε μεγάλο βαθμό με τον αλγόριθμο που εξηγήσαμε στο Κεφάλαιο 5.4.2. Διαφέρει μόνο σε 1 σημείο. Η βασική διαφορά του είναι ότι πλέον ο αλγόριθμος του Ford Fulkerson δεν είναι επαναληπτικός. Αντ' αυτού κάθε φορά που ο χρήστης πατά το κουμπί "next" τρέχει η επόμενη επανάληψη του αλγόριθμου. Ουσιαστικά εκτελούνται σχεδόν τα ίδια βήματα με αυτά που είδαμε στο Κεφάλαιο 5.4.2, δηλαδή τα πιο κάτω βήματα :

1. Δημιουργία υπολειπόμενου γραφήματος βάση υπάρχοντος δικτύου ροής όπως εξηγήθηκε στο Κεφάλαιο 5.3.
2. Εύρεση μονοπατιού επαύξησης πάνω στο υπολειπόμενο δίκτυο. Αν δεν υπάρχει μονοπάτι τότε τέλειωσε ο αλγόριθμος και βρέθηκε η μέγιστη ροή.

3. Αύξηση ροής στο δίκτυο ροής βάση της διαδρομής επαύξησης που βρέθηκε στο βήμα 2.

Η μόνη διαφορά με τα βήματα στο Κεφάλαιο 5.4.2 είναι ότι δεν υπάρχει το βήμα 4 το οποίο ήταν το βήμα της επανάληψης.



Σχήμα 5.17 – Τρέξιμο 4ων επαναλήψεων του Ford Fulkerson

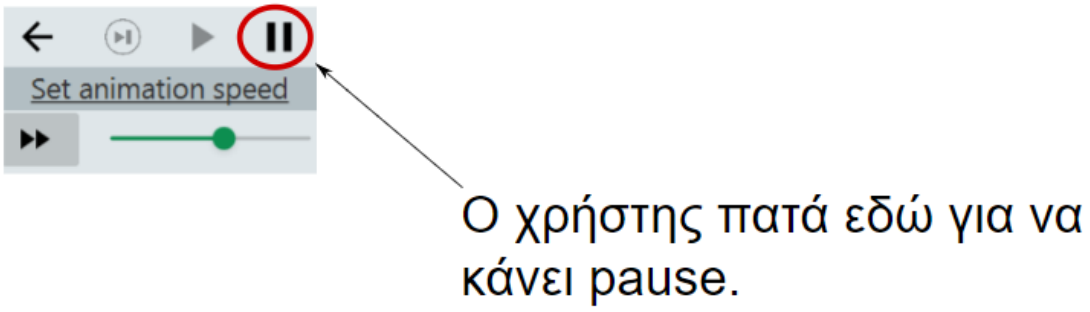
Πιο πάνω μπορούμε να δούμε πως λειτουργά ο Stepwise Ford Fulkerson στην πραγματικότητα. Όπως μπορούμε να δούμε κάθε φορά που ο χρήστης πατά το κουμπί "next" προχωρά στην επόμενη επανάληψη. Δηλαδή τρέχει ο αλγόριθμος runBFS(…) που είδαμε στο βήμα 2, στο Κεφάλαιο 5.4.2, πάνω στο υπολειπόμενο δίκτυο και στη συνέχεια εκτελείται το 3ο βήμα στο δίκτυο ροής, δηλαδή η διαδικασία επαύξησης, αλλά πλέον το βήμα 2 και βήμα 3 δεν βρίσκονται σε βρόγχο όπως φαίνεται και στον πιο κάτω ψευδοκώδικα αλλά εκτελούνται κάθε φορά που πατηθεί το κουμπί next .

```
if (run_BFS(resSource,resTarget,residual)) {
      execute_step_three(…)
} else { do_some_action(..) /** Algorithm is done */ }
```

Ψευδοκώδικας που εκτελείτε κάθε φορά που θα πατηθεί το κουμπί next.

5.5.4. Περιγραφή Υλοποίησης του Κουμπιού Pause

Καθώς τρέχουν τα animation δίνεται η δυνατότητα στον χρήστη να τα σταματήσει προσωρινά μέσω του κουμπιού Pause.



Σχήμα 5.18 – Pause Button

Για να υλοποιήσουμε την λειτουργία Pause, κάθε φορά που δημιουργούσα ένα Animation το πρόσθετα σε μια συνδεδεμένη λίστα .

Απόσπασμα Προσθήκης Animation σε συνδεδεμένη λίστα.

```
animationLinkedList.add(sequential_transition);
```

Λόγω του ότι το Animation Object της JavaFX σου δίνει τη δυνατότητα να ελέγξεις την κατάσταση του Animation, δηλαδή αν τρέχει, αν είναι paused ή αν τέλειωσε, μια γραμμική διάσχιση της λίστας ήταν αρκετή. Αν βρίσκαμε κάποιο Animation σε κατάσταση “Running” τότε το κάναμε “Pause”.

Απόσπασμα Γραμμικής Διάσχισης Λίστας για να κάνουμε Pause:

```
 for (Animation animation:BFS_Algorithm.animationLinkedList){
    if
(animation.statusProperty().get()==Animation.Status.RUNNING){
        animation.pause();
    }}
```

Αν ο χρήστης θέλει να συνεχίσει το Animation να τρέχει τότε αρκεί να πατήσει το κουμπί “play”. Ουσιαστικά όποτε πατά το κουμπί “play” γίνεται μια γραμμική διάσχιση πάνω στην συνδεδεμένη λίστα και αν υπάρχει κάποιο Animation που είναι “paused” τότε το αφήνουμε να τρέξει.

Απόσπασμα Γραμμικής Διάσχισης Λίστας για να γίνει UNPAUSE το animation :

```
for (Animation animation:BFS_Algorithm.animationLinkedList){
    if
(animation.statusProperty().get()==Animation.Status.PAUSED) {
        animation.play();
        break;
    }
}
```

5.6. Εύρεση Ελάχιστης Αποκοπής Δικτύου Ροής

Θεώρημα Μέγιστής Ροής/Ελάχιστης Αποκοπής [1,2]:

Θεώρημα Μέγιστής Ροής/Ελάχιστης Αποκοπής:

Έστω (G, s, t, c) ένα δίκτυο ροής και έστω f είναι η ροή σε αυτό το δίκτυο. Τα ακολούθα είναι ισοδύναμα:

1. Η ροή , δηλαδή το f, μεγιστοποιείται σε ένα δίκτυο ροής
2. Το G_f , δηλαδή το υπολειπόμενο δίκτυο, δεν έχει μονοπάτι επαύξησης.
3. Υπάρχει μια αποκοπή C(S,T) τέτοια ώστε C(S,T) = value(f) (δηλαδή ίσο με τη μέγιστη

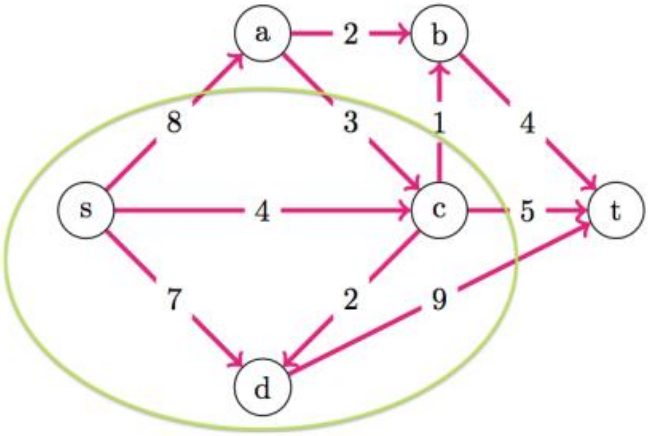
Άρα με πιο απλά λόγια το θεώρημα ελάχιστης αποκοπής δηλώνει ότι η μέγιστη ροή σε ένα δίκτυο ροής είναι ακριβώς ίση με το άθροισμα της ελάχιστης αποκοπής. Δηλαδή Max_Flow(G)= Min (C(S,T))

Τί είναι όμως μια αποκοπή?

Ορισμός: Έστω (G, s, t, c) ένα δίκτυο ροής, τότε **η αποκοπή s-t** στο G είναι ένας διαχωρισμός του V (V= όλοι οι κόμβοι στο G) , σε δύο σύνολα το S και το T τέτοια ώστε:

1. S ∪ T = V
2. S ∩ T = Ø
3. s ∈ S and t ∈ T

Πιο κάτω έχουμε ένα παράδειγμα αποκοπής C(S, T) όπου S = {s, c, d} και T = {a, b, t }.



Σχήμα 5.19 – Παράδειγμα αποκοπής

Η χωρητικότητα μιας αποκοπής C(S,T), συμβολίζεται ως c(S,T) και είναι το άθροισμα των χωρητικοτήτων των ακμών (u,v) όπου u ∈ S και v ∈ T και υπολογίζεται από την πιο κάτω πράξη:

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

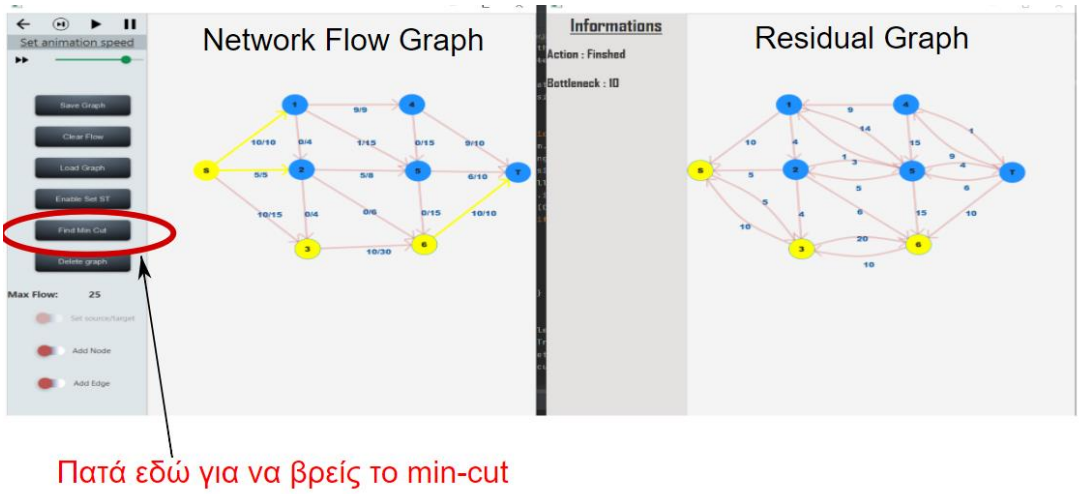
Στο πιο πάνω παράδειγμα η αποκοπή είναι c(S, T) = c(s,a)+c(c,b)+c(c,t)+c(d,t)=23.

Αφού εξηγήσαμε τι είναι μια αποκοπή και πως σχετίζεται με την μέγιστη ροή σε ένα δίκτυο ροής πάμε να εξηγήσουμε πως μπορούμε να βρούμε την ελάχιστη αποκοπή.

Οι χρήστες της εφαρμογής μας έχουν την δυνατότητα να δουν ποια είναι η ελάχιστη αποκοπή ενός δικτύου ροής αφού τελειώσει την εκτέλεση του ο αλγόριθμος Ford Fulkerson.

Για να βρεθεί η ελάχιστη αποκοπή σε ένα δίκτυο ροής πρέπει να εκτελεστούν τα πιο κάτω βήματα [1,2]:

1. Εκτέλεση του αλγόριθμου Ford Fulkerson πάνω σε ένα δίκτυο ροής.
2. Εκτέλεση του αλγόριθμου Breadth First Search (ξεκινώντας από τον κόμβο πηγή) πάνω στο υπολειπόμενο δίκτυο που προκύπτει μετά το τέλος της εκτέλεσης του αλγόριθμου. Όσοι κόμβοι είναι προσβάσιμοι από την πηγή (η πηγή θεωρείται εξορισμού ότι ανήκει στο σύνολο S) ανήκουν στο S και οι υπόλοιποι ανήκουν στο T.



Σχήμα 5.20 – Εύρεση Ελάχιστης αποκοπής στην εφαρμογή μας

Όπως μπορούμε να δούμε στο σχήμα 5.20, όταν ο χρήστης πατήσει το κουμπί Find Min Cut τρέχει ο αλγόριθμος Breadth First Search πάνω στο Residual Graph. Όπως βλέπουμε, οι κόμβοι που είναι προσβάσιμοι από την πηγή είναι οι κόμβοι S,3,6 και αυτοί είναι οι κόμβοι που έχουν χρώμα κίτρινο. Αυτούς του κόμβους μπορούμε να τους θεωρήσουμε ως τους κόμβους που ανήκουν στο σύνολο S και οι υπόλοιποι αυτοί που ανήκουν στο σύνολο T. Επίσης στο Δίκτυο ροής μπορούμε να δούμε ότι οι ακμές που ενώνουν το σύνολο S με το

σύνολο Τ έχουν χρώμα κίτρινο και επίσης το άθροισμα των χωρητικοτήτων αυτών των ακμών είναι 25, που αυτή είναι και η μέγιστη ροή.

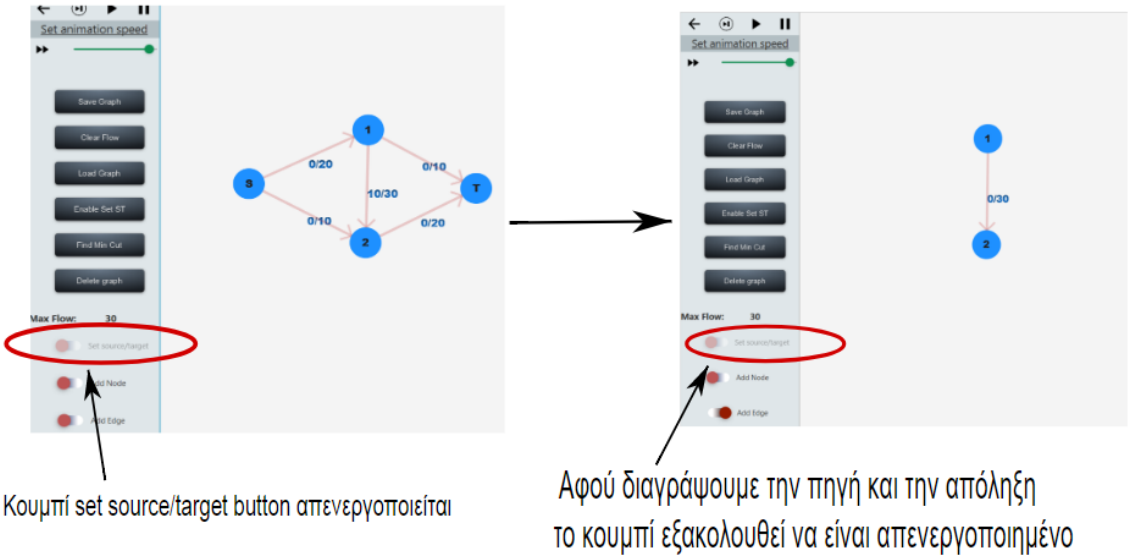
Απόσπασμα υλοποίησης Εύρεσης Min-Cut:

```
public static void findMinCut(Cell resSource,Cell
resTarget,Graph residual,int time,Graph graph){
    BFS_Algorithm.run_BFS(resSource,resTarget,residual);
    HashSet<String> a_star=new HashSet<>();
    ParallelTransition min_cut=new ParallelTransition();
    for (Cell cell:residual.getModel().getAllCells()){
        if (cell.isVisited()){
            for (Cell
flow_cell:graph.getModel().getAllCells()){
                if
(flow_cell.getCellId().equals(cell.getCellId())) {
                    Circle u=(Circle)flow_cell.getView();
                    FillTransition ft = new
FillTransition(Duration.millis(time), u);
                    ft.setToValue(Color.YELLOW);
                    min_cut.getChildren().add(ft);
                    a_star.add(flow_cell.getCellId());
                }
            }

            Circle u=(Circle)cell.getView();
            FillTransition ft = new
FillTransition(Duration.millis(time), u);
            ft.setToValue(Color.YELLOW);
            min_cut.getChildren().add(ft);
        }
    }
```

```
    for (Edge edge:graph.getModel().getAllEdges()){
        if (a_star.contains(edge.getSource().getCellId()) &&
!a_star.contains(edge.getTarget().getCellId())) {
            if (edge.line!=null){
                StrokeTransition ft1 = new
StrokeTransition(Duration.millis(time), edge.line);
                ft1.setToValue(Color.YELLOW);
                min_cut.getChildren().add(ft1);
                ft1 = new
StrokeTransition(Duration.millis(time),
edge.line.getAttachedArrow());
                ft1.setToValue(Color.YELLOW);
                min_cut.getChildren().add(ft1);
            }
            if (edge.cubicCurveLine!=null){
                StrokeTransition ft1 = new
StrokeTransition(Duration.millis(time), edge.cubicCurveLine);
                ft1.setToValue(Color.YELLOW);
                min_cut.getChildren().add(ft1);
                ft1 = new
StrokeTransition(Duration.millis(time),
edge.cubicCurveLine.getAttachedArrow());
                ft1.setToValue(Color.YELLOW);
                min_cut.getChildren().add(ft1);
            }

        }
    }
    playAnimationAndWaitForFinish(min_cut);
```

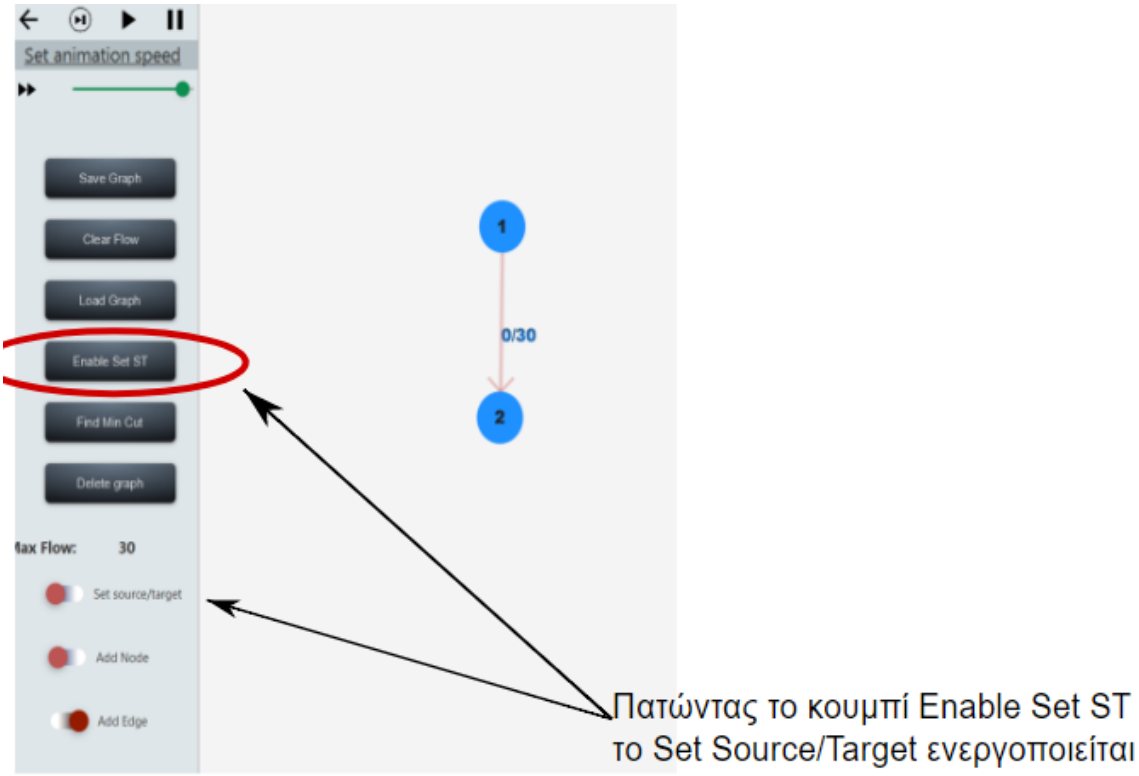
5.7. Δυσκολίες, Προβλήματα και Τρόπος Αντιμετώπισης τους

Όπως αναφέραμε προηγούμενος, ο χρήστης έχει τη δυνατότητα να θέσει την πηγή και την απόληξη στο γράφημα που θα σχεδιάσει, όταν το κάνει αυτό τότε το κουμπί που θέτει την πηγή και την απόληξη απενεργοποιείται(το κουμπί που εξηγήθηκε στο Κεφάλαιο 5.2.10).



Σχήμα 5.21 – Πρόβλημα με το κουμπί “set source/target”

Επομένως αν ο χρήστης ήθελε να διαγράψει και να προσθέσει καινούργιους κόμβους ως πηγή και απόληξη δεν μπορούσε να το κάνει λόγω του απενεργοποιημένου κουμπιού. Για την αντιμετώπιση αυτού του προβλήματος υλοποιήθηκε το κουμπί Enable ST το οποίο ενεργοποιούσε ξανά το κουμπί για την πηγή και την απόληξη. Όπως φαίνεται και από το πιο κάτω σχήμα, αυτό το κουμπί λύνει αυτό το πρόβλημα.



Σχήμα 5.22 – Περίπτωση Χρήσης του κουμπιού “Enable Set ST”

Απόσπασμα Υλοποίησης Enable Set ST Button:

```
public static void enableToggleButton(Graph graph, JFXToggleButton add_source_targetToggleBtn){
    boolean foundSource=false;
    boolean foundTarget=false;
    for(Cell cell:graph.getModel().getAllCells()){
        CircleCell cell1=(CircleCell) cell;
        if (cell1.getLabel_id().equals("S"))
            foundSource=true;
        if (cell1.getLabel_id().equals("T"))
            foundTarget=true;
    }
    if (!foundSource && !foundTarget)
        add_source_targetToggleBtn.setDisable(false);
}
```

Όπως βλέπουμε και από το πιο πάνω κομμάτι κώδικα, ελέγχονται όλοι οι κόμβοι και αν δεν βρεθεί ούτε η πηγή και ούτε η απόληξη το κουμπί γίνεται enable.

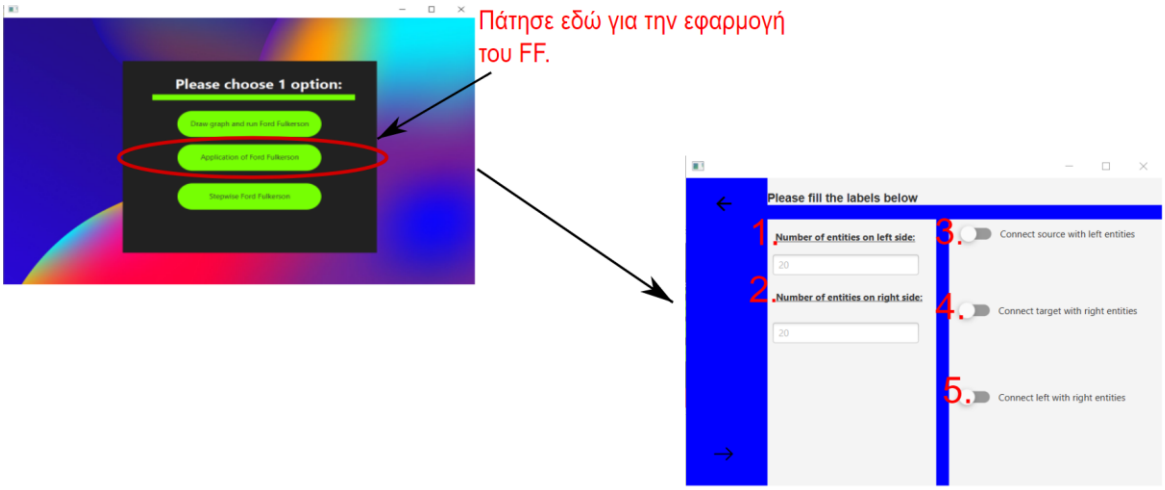
Κεφάλαιο 6

Υλοποίηση Διεπαφής Εφαρμογής Αλγόριθμου Ford Fulkerson

6.1. Εισαγωγή

Στα πλαίσια της διπλωματικής υλοποιήθηκε και μια τρίτη επιλογή η οποία είναι κυρίως χρήσιμη για εφαρμογές του αλγόριθμου σε πραγματικά προβλήματα. Συνήθεις περιπτώσεις εφαρμογής του δικτύου ροής έχουμε όταν θέλουμε να ταιριάξουμε ένα σύνολο οντοτήτων με ένα άλλο σύνολο όπως εξηγήθηκε στο Κεφάλαιο 3.1.2. Επομένως με αυτή την επιλογή δίνεται η δυνατότητα στο χρήστη να αυτοματοποιήσει τον σχεδιασμό δικτύων ροής όπου έχουμε δυο σύνολα τα οποία θέλουμε να τα αντιστοιχήσουμε μεταξύ τους. Για τον αυτόματο σχεδιασμό τέτοιων συνόλων θα ζητείται από τον χρήστη να δηλώσει πόσες οντότητες (κόμβους) θα θέλει να έχει στο $1^{ο}$ σύνολο, πόσες οντότητες να έχει στο $2^{ο}$ και να δίνει τις χωρητικότητες των ακμών που θα ενώνουν αυτά τα δυο σύνολα μεταξύ τους. Επίσης θα δίνει και τις χωρητικότητες των ακμών που θα προέρχονται από την πηγή και αυτών που θα καταλήγουν στην απόληξη.

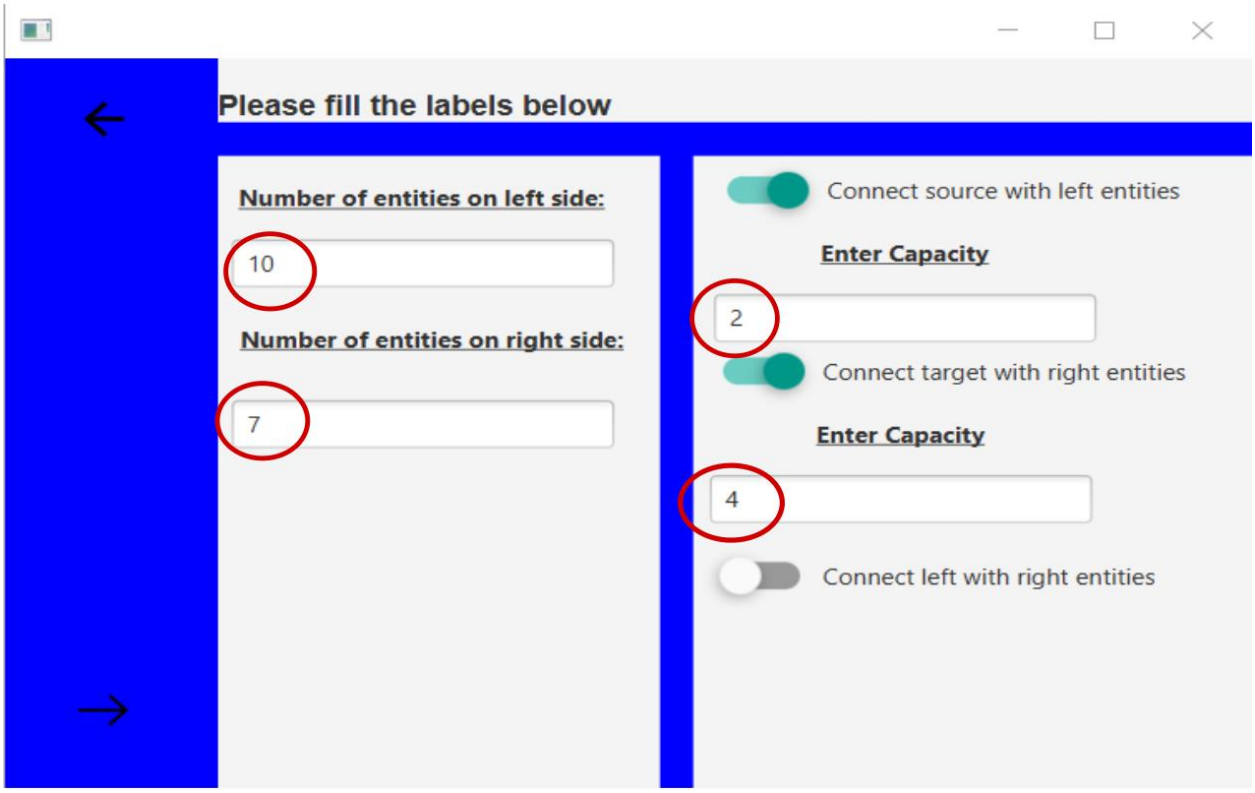
6.2. Λεπτομέρειες Υλοποίησης Αυτοματοποιημένης Δημιουργίας Δικτύου Ροής



Σχήμα 6.1 – Επιλογή Εφαρμογής Ford Fulkerson

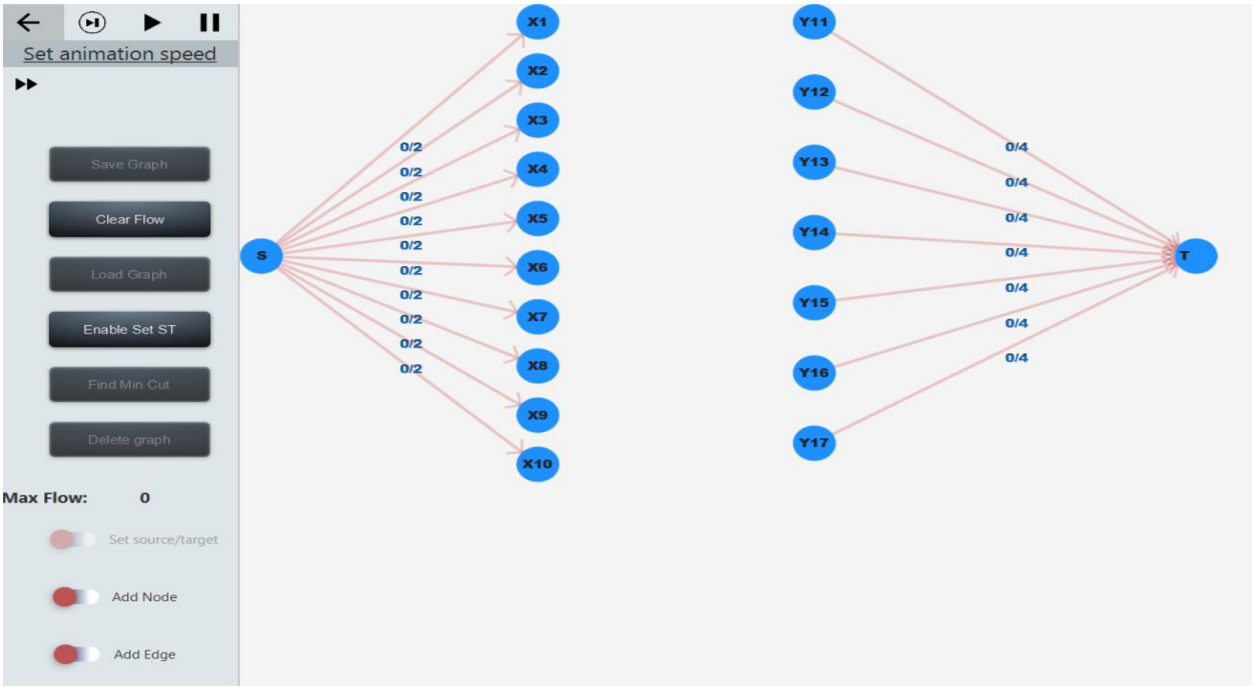
Στην πιο πάνω φόρμα ο χρήστης μπορεί να δηλώσει πόσες οντότητες μπορεί να έχει στο αριστερό μέρος της οθόνης (Σημείο 1: “number of entities on left side”), πόσες οντότητες μπορεί να έχει στο δεξί μέρος της οθόνης, (Σημείο 2: “number of entities on right side”), αν θέλει να συνδέσει την πηγή του δικτύου ροής με τις οντότητες στο αριστερό μέρος (Σημείο 3: “connect source with left entities"), παρόμοια και για τις οντότητες της δεξιάς πλευράς με την απόληξη (Σημείο 4: “connect target with right entities”) και τέλος αν θέλει να συνδέσει τις οντότητες της αριστερής και δεξιάς πλευράς μεταξύ τους (Σημείο 5: “connect left with right entities”).

Για παράδειγμα αν ο χρήστης συμπληρώσει την φόρμα με τον εξής τρόπο:



Σχήμα 6.2 – Φόρμα πληροφοριών για αυτόματη δημιουργία δικτύου ροής

Δηλαδή αν δηλώσει ότι στο αριστερό σύνολο θα έχουμε 10 οντότητες και θα είναι συνδεμένες με την πηγή με χωρητικότητα 2, στο σύνολο στα δεξιά θα έχουμε 7 οντότητες και θα είναι συνδεδεμένες με την απόληξη με χωρητικότητα 4 τότε το πιο κάτω δίκτυο ροής θα δημιουργηθεί αυτόματα πατώντας το κουμπί με σχήμα δεξιού τόξου, κάτω αριστερά.



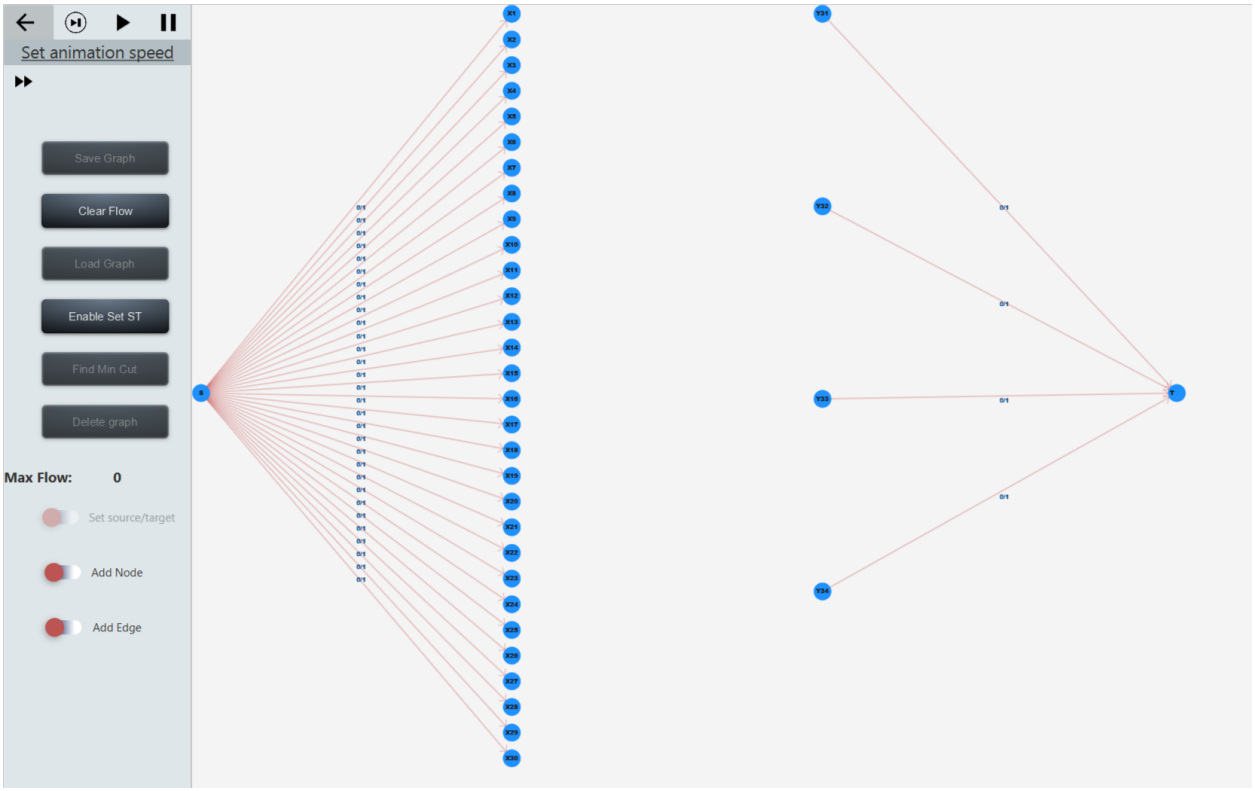
Σχήμα 6.3: Δίκτυο ροής που προκύπτει από τις πληροφορίες του Σχήματος 6.2

Όπως μπορούμε να παρατηρήσουμε από το πιο πάνω σχήμα, οι κόμβοι του αριστερού συνόλου δεν είναι συνδεδεμένοι με τους κόμβους του συνόλου στα δεξιά διότι η επιλογή “connect left with right entities” δεν είναι επιλεγμένη.

Πάμε όμως τώρα να δούμε κάποιες λεπτομέρειες υλοποίησης για το πως δημιουργείται αυτόματα το δίκτυο ροής.

<u>Καθορισμός βαθμού μεγέθυνσης της οθόνης:</u>

Λόγω του ότι ο χρήστης μπορεί να πληκτρολογήσει οποιονδήποτε αριθμό κόμβων πρέπει η οθόνη να μεγεθύνεται και να σμικρύνεται ανάλογα με το πόσους κόμβους πρέπει να φιλοξενήσει η οθόνη. Για παράδειγμα στο προηγούμενο σχήμα έχουμε το πολύ 10 κόμβους κάθετα και επομένως η οθόνη έχει γίνει σμικρυνθεί πολύ λίγο. Αν όμως ο χρήστης δώσει για παράδειγμα 40 κόμβους αντί 10 τότε η μεγέθυνση της σκηνής θα προσαρμοστεί αναλόγως έτσι ώστε να φορέσουν όλοι οι κόμβοι μέσα στην οθόνη όπως στο πιο κάτω σχήμα. Όπως βλέπουμε η οθόνη έχει σμικρυνθεί αρκετά σε σχέση με το προηγούμενο σχήμα.



Σχήμα 6.4 – Αυτοματοποιημένη δημιουργία δικτύου ροής με προσαρμογή στη μεγέθυνση

Η υλοποίηση της λειτουργίας που μόλις περιγράψαμε γίνεται με τα πιο κάτω βήματα:

1. Υπολογισμός απαιτούμενου ύψους για να χωρέσουν όλοι οι κόμβοι. Δεδομένου ότι κάθε κόμβος απέχει με τον κάτω του κόμβο 90 χιλιοστά, το απαιτούμενο ύψος θα είναι αριθμός οντοτήτων * 90.
2. Υπολογισμός βαθμού μεγέθυνσης , η αλλιώς scale έτσι ώστε να χωρούν όλοι οι κόμβοι στην οθόνη.

<u>Βήμα 1ο - Υπολογισμός απαιτούμενου ύψους για να χωρέσουν όλοι οι κόμβοι</u>:

```
int desired_height=Math.max(left_entities,right_entities)*90;
```

Βήμα 2ο – Υπολογισμός Βαθμού μεγέθυνσης:

```
for (double scale=0.0001;scale<=1;scale+=0.0001){
    double appr_height=scene.getHeight()/scale;
    if (Math.abs(desired_height-appr_height)<50)
        final_scale=scale;
}
((ZoomableScrollPane)
graph.getScrollPane()).zoomTo(final_scale);
```

Όπως μπορούμε να δούμε, στο βήμα 2 υπολογίζεται ένα προσεγγιστικό ύψος, ανάλογα με το βαθμό μεγέθυνσης, με την πράξη scene.getHeight() / scale και αν αυτό είναι αρκετά κοντά με το επιθυμητό ύψος (δηλαδή το προσεγγιστικό με το επιθυμητό διαφέρουν λιγότερο από 50 χιλιοστά) τότε αποθηκεύουμε αυτή την τιμή μεγέθυνσης.

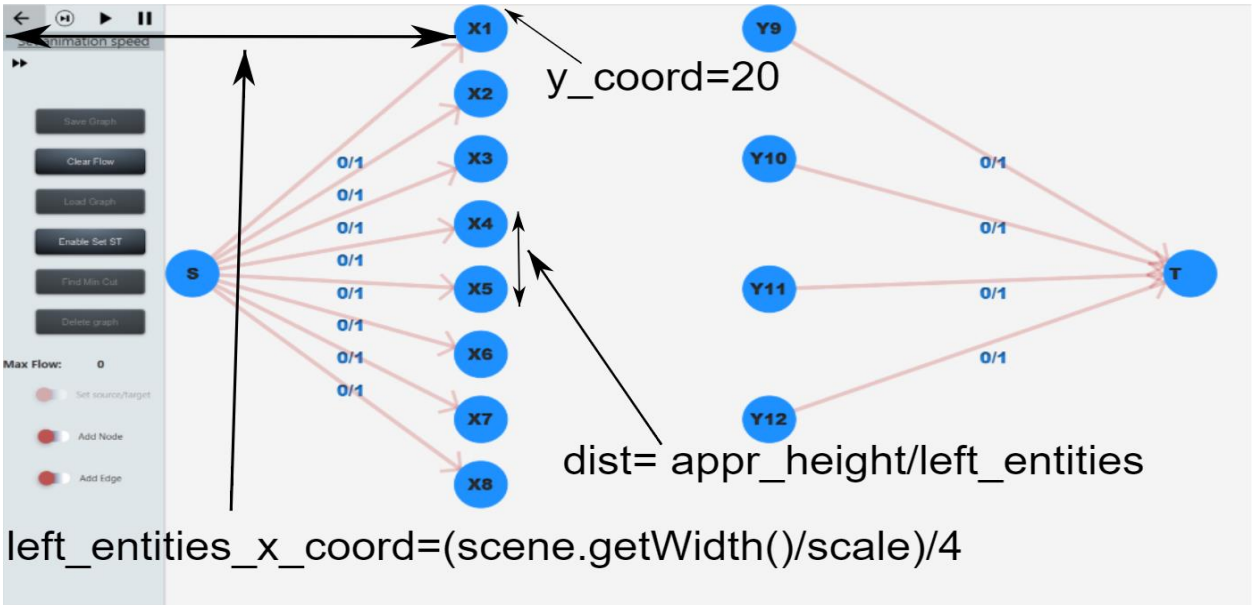
Πάμε τώρα να εξηγήσουμε πως υπολογίζονται οι συντεταγμένες των κόμβων του αριστερού συνόλου, δηλαδή του συνόλου των κόμβων που το όνομα τους ξεκινά από Χ.

Αρχικά υπολογίζουμε το σωστό πλάτος της οθόνης, συνάρτηση της μεγέθυνσης , για αυτό και το διαιρούμε με τη μεγέθυνση που υπολογίσαμε στο βήμα 2 στο πιο κάτω απόσπασμα κώδικα. Στη συνέχεια το διαιρούμε δια 4 έτσι ώστε να υπολογίσουμε την αρχική συντεταγμένη Χ. Στη συνέχεια υπολογίζουμε το ύψος της οθόνης και πάλι συναρτήσει της υφιστάμενης μεγέθυνσης και τέλος υπολογίζουμε την συντεταγμένη Υ της οποίας δίνουμε τιμή 20 έτσι ώστε να ξεκινήσουμε από την αρχή της οθόνης. Ακολούθως μπορούμε να δούμε ότι η συντεταγμένη Υ αυξάνεται κατά απόσταση “dist”. Η απόσταση “dist” υπολογίζεται ανάλογα με το πόσους κόμβους θα πρέπει να φορέσουν κάθετα μέσα στην οθόνη, για αυτό και υπολογίζεται με τη φόρμουλα: ύψος οθόνης/ αριθμός οντοτήτων .

```
double
appr_width=(scene.getWidth()+widthOfLeftPane)/final_scale;
double left_entities_x_coord= appr_width/4;
double appr_height=scene.getHeight()/final_scale;
double left_entities_y_coord=20;

for (int i=0;i<left_entities;i++) {
nodeID++;
graph.getModel().addCell(nodeID,left_entities_x_coord,left_en
tities_y_coord);
double dist=appr_height/left_entities;
left_entities_y_coord+=dist;
}
```

Απόσπασμα υλοποίησης αυτόματης προσθήκης κόμβων αριστερού συνόλου

Πιο κάτω μπορούμε να δούμε και ένα απόσπασμα το οποίο δείχνει το αποτέλεσμα του πιο πάνω αποσπάσματος κώδικα.



Σχήμα 6.5 - Επεξήγηση υπολογισμού συντεταγμένων κόμβων αριστερού συνόλου

Αφού προστέθηκαν οι κόμβοι αυτόματα πρέπει να προστεθούν και οι ακμές αυτόματα. Αν ο χρήστης επιλέξει το κουμπί “connect source with left entities” τότε μπορεί να δηλώσει την χωρητικότητα της ακμής και θα συνδεθούν όλοι οι κόμβοι του αριστερού συνόλου με την πηγή, με την χωρητικότητα που δηλώθηκε. Πιο κάτω μπορούμε να δούμε και ένα απόσπασμα υλοποίησης της αυτόματης προσθήκης ακμών.

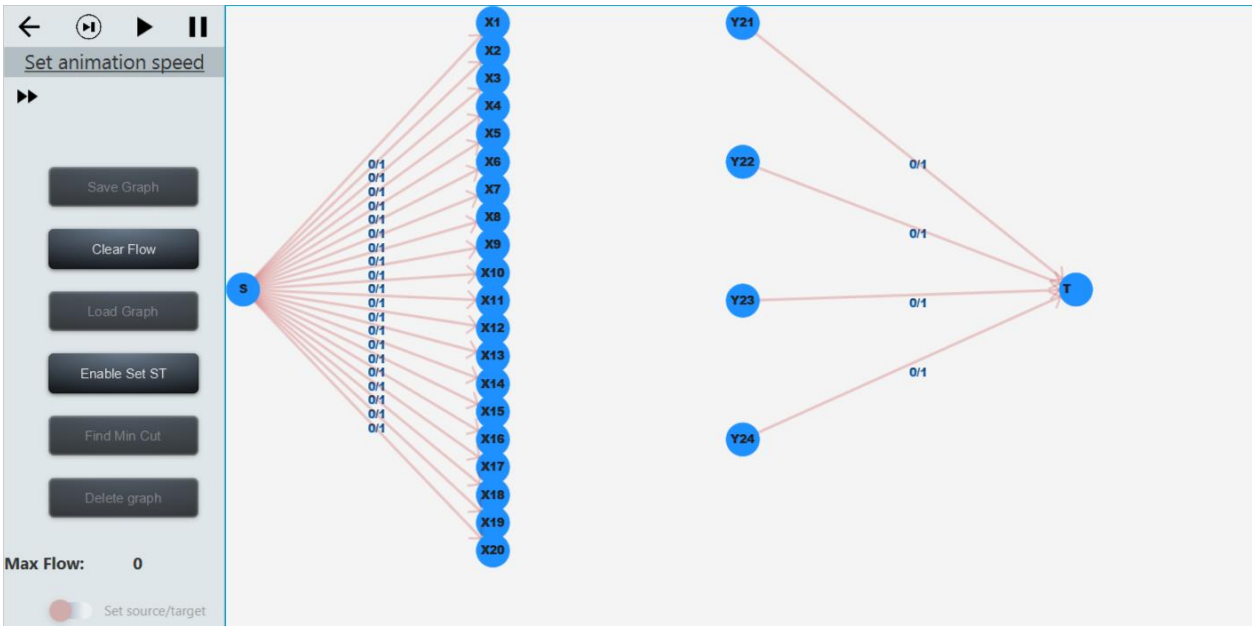
```
for (Cell target:leftCells){

int capacity=1;
if
(informationScreenController.left_toggle_button.isSelected())
{
capacity=Integer.parseInt(informationScreen.left_capacity_tex
tField.getText()
}
CircleCell circleCell1=(CircleCell)target;
circleCell1.setOption2Label(circleCell1.getLabel_id());
circleCell1.setLabel_id("X"+circleCell1.getLabel_id());
graph.removeSelectedNodes(source,target);
graph.addNodesAndEdge(source,target,capacity,offset,false,opt
ion2);

}
```

Απόσπασμα Υλοποίησης προσθήκης ακμών μεταξύ πηγής και κόμβων αριστερού συνόλου

Η υλοποίηση για την προσθήκη των κόμβων του δεξιού συνόλου ακολουθά ακριβώς την ίδια λογική και για αυτό δεν θα την εξηγήσω.

6.3. Δυσκολίες, Προβλήματα και Τρόπος Αντιμετώπισης τους

Τα κυριότερα προβλήματα που αντιμετώπισαμε κατά την υλοποίηση αυτοματοποιημένης δημιουργίας του δικτύου ροής ήταν ο σωστός υπολογισμός των αποστάσεων μεταξύ των κόμβων. Οι αρχικές προσπάθειες τοποθετούσαν τους κόμβους πολύ κοντά μεταξύ τους και επομένως το τελικό αποτέλεσμα του δικτύου ροής ήταν αρκετά άσχημο. Για παράδειγμα όπως φαίνεται και στο πιο κάτω σχήμα, το τελικό αποτέλεσμα είναι αρκετά άσχημο καθώς δεν υπάρχουν οι σωστές αποστάσεις μεταξύ των κόμβων.



Σχήμα 6.6 - Προβληματικό δίκτυο ροής.

Για την επίλυση αυτού του προβλήματος χρησιμοποιήθηκε ο υπολογίσιμος αποστάσεων που εξηγήθηκε στο Κεφάλαιο 6.2 και το αποτέλεσμα που προκύπτει στο Σχήμα 6.4, μπορούμε να δούμε πως είναι αρκετά καλύτερο.

Κεφάλαιο 7

Συμπεράσματα

7.1 Επίλογος

Συνοψίζοντας, με την υφιστάμενη διπλωματική εργασία έχουμε υλοποιήσει ένα χρήσιμο εργαλείο το οποίο όχι μόνο παρουσιάζει οπτικά με τη χρήση δυσδιάστατων γραφικών τον αλγόριθμο Ford Fulkerson, αλλά δίνει επίσης τη δυνατότητα στον χρήστη να αλληλοεπιδράσει με το σύστημα μας και να μάθει με τη μέθοδο δοκιμής και σφάλματος πως να σχεδιάζει ένα σωστό και έγκυρο δίκτυο ροής, να δημιουργήσει εύκολα και γρήγορα εφαρμογές του Ford Fulkerson σε πραγματικά προβλήματα και τέλος να τρέξει τον αλγόριθμο βήμα-βήμα και να δει λεπτομερώς όλα τα στάδια που περνά ο αλγόριθμος.

Αρχικά μετά την περιγραφή των εργαλείων JavaFX, FXML και του εργαλείου Scene Builder αλλά και στη συνέχεια με την αναλυτική επεξήγηση του αλγόριθμου Ford Fulkerson αλλά και των διεπαφών του προγράμματος ο χρήστης αποκτά βαθιά κατανόηση για το πως λειτουργεί η εφαρμογή μας λόγω της λεπτομερής επεξήγησης του κάθε κουμπιού, τόσο σε πρακτικό επίπεδο δηλαδή πως χρησιμεύει στα πλαίσια της εφαρμογής αλλά και σε αλγοριθμικό παραθέτοντας λεπτομέρειες υλοποίησης για κάθε κουμπί. Στη συνέχεια εξηγήθηκε πως υλοποιήθηκε το animation χρησιμοποιώντας το εργαλείο της JavaFX και επίσης εξηγήθηκε το θεώρημα Μέγιστης Ροής – Ελάχιστης Αποκοπής τόσο σε θεωρητικό βαθμό αλλά και σε πρακτικό αφού υποστηρίζεται σαν λειτουργία από την εφαρμογή μας.

Συνοψίζοντας στην εργασία αυτή υλοποιήθηκε ένα πολύ χρήσιμο εργαλείο το οποίο πιστεύω ότι θα βοηθήσει τόσο τους διδάσκοντες στην πιο εύκολη και διαδραστική

επεξήγηση του αλγόριθμου αλλά επίσης και τους φοιτητές να κατανοήσουν σε βάθος τον αλγόριθμο και να αντιληφθούν τη σημαντικότητα του αλγορίθμου σε πραγματικά προβλήματα. Επιπρόσθετα λόγω του ότι ο κύριος σκοπός του προγράμματος μου ήταν εκπαιδευτικός έγιναν αρκετές παρουσιάσεις υλοποιήσεων σε ένα κοινό από φοιτητές, φίλους και γνώστες της πληροφορικής με σκοπό να πάρουμε ανατροφοδότηση για την εφαρμογή και να την βελτιώσουμε. Η παρουσίαση του προγράμματος σε "Beta" μορφή στο μάθημα "ΕΠΛ 236:Αλγόριθμοι και πολυπλοκότητα" είχε πάρει καλές κριτικές από τους φοιτητές και έκανε την κατανόηση του αλγόριθμου αλλά και την εκμάθηση του πιο εύκολη και αποτελεσματική. Επίσης η "Beta" μορφή παρουσιάστηκε και σε μια ομάδα από άτομα τα οποία είτε παρακολούθησαν ήδη το μάθημα (συμφοιτητές μου) αλλά και σε άτομα τα οποία δεν γνώριζαν τον αλγόριθμο. Σε αυτή την ομάδα ατόμων η εφαρμογή πήρε καλά σχόλια και ισχυρίστηκαν ότι η εφαρμογή αυτή βοηθά σε μεγάλο βαθμό στην κατανόηση του αλγορίθμου. Όλες οι πιο πάνω παρουσιάσεις που έγιναν, συνοδεύτηκαν από ένα ερωτηματολόγιο που έπρεπε να απαντήσει κάθε άτομο έτσι ώστε να δώσει την άποψη του για την εφαρμογή. Τα ερωτηματολόγια αυτά βρίσκονται στο Παράρτημα Α.

7.2 Κυριότερα Προβλήματα και Τρόπος Αντιμετώπισης

Σε αυτό το σημείο θα αναφερθώ στα κυριότερα προβλήματα που αντιμετωπίσαμε κατά τη διάρκεια υλοποίησης της διπλωματικής εργασίας. Θα γίνει επομένως μια σύνοψη των προβλημάτων που αντιμετωπίσαμε :

1. Δυσκολία στην εκτέλεση κινουμένων σχεδίων λόγω του ότι η εκτέλεση animations στο JavaFX είναι ασύγχρονη. Δηλαδή εκτελεί την επόμενη εντολή χωρίς να μπλοκάρει μέχρι τον τερματισμό του animation. Η επίλυση αυτού του προβλήματος εξηγήθηκε στο Κεφάλαιο 5.4.
2. Αποθήκευση γράφου σε αρχείο. Αρχικά προσπαθήσαμε να αποθηκεύσουμε κάθε αντικείμενο που ήταν μέρος του γράφου σε ένα αρχείο. Για να μπορεί όμως ένα αντικείμενο να αποθηκευτεί σε ένα αρχείο πρέπει να υλοποιεί τη διεπαφή Serializable. Δεν υλοποιούσαν όμως όλα τα αντικείμενα τη διεπαφή Serializable και

επομένως για την αποθήκευση του γράφου χρησιμοποιήσαμε την μέθοδο που περιγράψαμε στο Κεφάλαιο 5.2.4.

3. Δυσκολία στην εγκατάσταση της εφαρμογής. Το εργαλείο JavaFX δεν παρέχει κάποιο τρόπο για αυτοματοποιημένη δημιουργία εκτελέσιμων αρχείων, δηλαδή .exe αρχείων ή .app αρχείων. Επομένως για την δημιουργία εκτελέσιμων αρχείων (.exe) χρησιμοποιήθηκαν κάποια εξωτερικά εργαλεία όπως το Launch4j [22] τα οποία δημιουργούν ένα εκτελέσιμο ενσωματώνοντας όλο το περιβάλλον της Java μέσα στην εφαρμογή μας. Το εργαλείο Launch4J χρησιμοποιήθηκε επίσης και για την δημιουργία ενός MAC εκτελέσιμου. Το μειονέκτημα του Launch4J όμως είναι ότι δεν δημιουργεί ένα .app αρχείο αλλά ένα ".sh" script που τρέχει το πρόγραμμα.

7.3 Μελλοντική Εργασία και Επεκτάσεις

Η εφαρμογή υποστηρίζει ένα μεγάλο αριθμό από λειτουργίες αλλά υπάρχουν αρκετές βελτιώσεις που μπορούν να γίνουν καθώς και επεκτάσεις. Μερικές από τις μελλοντικές βελτιώσεις αφορούν την εμφάνιση και υλοποίηση του προγράμματος και άλλες αφορούν πρόσθετες λειτουργίες που θα μπορούσε να υποστηρίζει η εφαρμογή μας.

Μερικές από τις πρόσθετες λειτουργίες που μπορούν να υλοποιηθούν στην εφαρμογή μας είναι οι εξής:

1. Χρήση τρισδιάστατων σχημάτων αντί δυσδιάστατων έτσι ώστε να γίνει πιο λεπτομερές και παραστατικό το τρέξιμο του αλγορίθμου. Η υλοποίηση αυτής της επέκτασης θα απαιτούσε αρκετές αλλαγές, καθώς φεύγουμε πλέον από τον δυσδιάστατό χώρο που βρισκόμασταν προηγουμένως και πάμε πλέον σε τρισδιάστατο.
2. Προσθήκη ήχων και ηχητικών εφέ πάνω στις κινήσεις αντικείμενων και πάνω στα κουμπιά. Θα μπορούσε και αυτό να υλοποιηθεί σχετικά εύκολα αφού η JavaFX παρέχει αυτοματοποιημένους τρόπους δημιουργίας ηχητικών εφέ.
3. Βελτίωση του μεγέθους των κουμπιών αλλά και των χρωμάτων των διεπαφών μας.

4. Επέκταση υλοποίησης σε κινητά τηλέφωνα Android και IOS καθώς η υπάρχουσα διεπαφή μπορεί να δουλεύει και με touch controls αντί με το ποντίκι. Η επέκταση σε κινητά τηλέφωνα απαιτεί αρκετή δουλεία, κυρίως λόγω θεμάτων που έχουν να κάνουν με την αποκριτικότητα της εφαρμογής και αυτόματη προσαρμογή μεγέθους της οθόνης σε κάθε διαφορετική οθόνη.
5. Επέκταση της λειτουργίας “Application of Ford Fulkerson” έτσι ώστε να υποστηρίζει και μικρό-εργαλεία ανά οντότητα.

Βιβλιογραφία

[1] Σημειώσεις Μαθήματος ΕΠΛ 236: Αλγόριθμοι και Πολυπλοκότητα.Accessed: 2022-5-02

[2] Jon Kleinberg and Eva Tardos. Algorithm design addison wesley. Boston, MA,2005.

[3] Overview JavaFX. https://docs.oracle.com/javase/8/javafx/api/toc.htm. Accessed: 2022-5-02.

[4] Sergey Grinev. Mastering JavaFX 10: Build advanced and visually stunning Java applications. Packt Publishing Ltd, 2018

[5] ΕΠΛ 131: Αρχές Προγραμματισμού https://www.cs.ucy.ac.cy/courses/EPL131/. Accessed: 2022-5-02

[6] ΕΠΛ 133: Αντικειμενοστρεφής Προγραμματισμός https://www.cs.ucy.ac.cy/courses/EPL133/. Accessed: 2022-5-02

[7] ΕΠΛ 231: Δομές Δεδομένων και Αλγόριθμοι https://www.cs.ucy.ac.cy/courses/EPL231/. Accessed: 2022-5-02/

[8] James L Weaver, Weiqi Gao, Stephen Chin, Dean Iverson, and J Vos. Pro javafx 2. A definitive Guide to Rich Clients with Java Technology. New York. Apress, 2012. Accessed: 2022-5-02

[9] Oracle, 'JAVAFX 2.0 THE PREMIER PLATFORM FOR RICH ENTERPRISE CLIENT APPLICATIONS'. Oracle, 2011 http://www.oracle.com/technetwork/java/javafx/overview/javafx-2-datasheet-496523.pdf .

[10] "JavaFX Architecture," in oracle, 2013. [Online].: http://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm#A1106498. Accessed: 2022-5-02

[11] Maven - Introduction. https://www.cs.ucy.ac.cy/courses/EPL131/. Accessed: 2022-5-02

[12] JavaFX Java GUI Design Tutorials
https://www.youtube.com/playlist?list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG

Accessed: 2022-5-02

[13] LR Ford and DR Fulkerson. maximal flow through a network, canadian journal of mathematics. 1956.

[14] draw.io – Diagrams for Confluence and Jira. https://drawio-app.com/. Accessed: 2022-5-02.

[15] Νικόλας Τζιωρτζής, Απεικόνιση Βασικών Αλγορίθμων με τη Χρήση της Μηχανής Ηλεκτρονικών Παιχνιδιών Unity3D, Διπλωματική εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2019.

[16] JavaFX Architecture. https://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm. Accessed:2022-5-02.

[17] How to apply MVC in JavaFX. https://edencoding.com/mvc-in-javafx/. Accessed: 2022-5-02.

[18] Maven – Maven Features. https://maven.apache.org/maven-features.html. Accessed: 2022-5-02.

[19] Myint Than Kyi and Lin Lin Naing. Application of ford-fulkerson algorithm to maximum flow in water distribution pipeline network. International Journal of Scientific and Research Publications, 8(12):306–310, 2018.

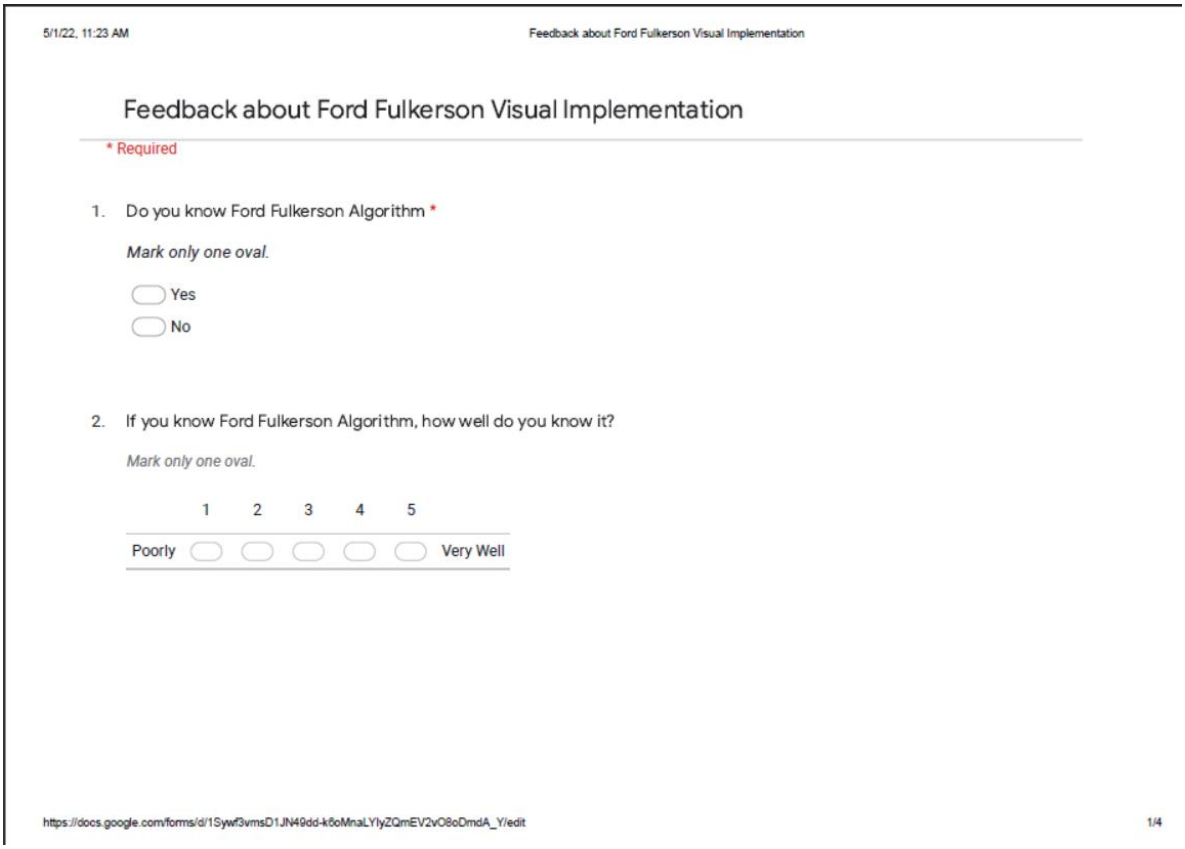
[20] Asif Iqbal, Md Sabir Hossain, and Harun Abir Ebna. Airline scheduling with max flow algorithm. Department of Computer Science and Engineer- ing, Faculty of Electrical & Computer Engineering, Chittagong University of Engineering and Technology, Chittagong, 4349, 2018.

[21] Αρχιτεκτονική JavaFX. https://tech.utugit.fi/soft/tools/lectures/dtek0097/frameworks/architecture/. Accessed: 2022-5-02.

[22] Launch4J. http://launch4j.sourceforge.net/ . Accessed: 2022-5-02.

Παράρτημα Α

Στο παράρτημα αυτό θα παρατεθούν τα ερωτηματολόγια τα οποία χρησιμοποιήθηκαν για την αξιολόγηση της εφαρμογής μας σε θέματα εμφάνισης, ευχρηστίας, λειτουργικότητας αλλά και πόσο βοηθά την εκμάθηση του αλγορίθμου. Επίσης θα παρατεθούν και οι απαντήσεις που έλαβαν αυτά τα ερωτηματολόγια. Αξίζει να σημειωθεί ότι η επιλογή και ο αριθμός των ατόμων έγινε έτσι ώστε οι μισοί να γνωρίζουν τον αλγόριθμο και οι άλλοι μισοί να μην τον γνωρίζουν έτσι ώστε η ανατροφοδότηση να καλύπτει και τις δυο ομάδες ατόμων.



5/1/22, 11:23 AM Feedback about Ford Fulkerson Visual Implementation

Feedback about Ford Fulkerson Visual Implementation

* Required

1. Do you know Ford Fulkerson Algorithm *

Mark only one oval.

- Yes
- No

2. If you know Ford Fulkerson Algorithm, how well do you know it?

Mark only one oval.

	1	2	3	4	5	
Poorly						Very Well

https://docs.google.com/forms/d/1Sywf3vmsD1JN49dd-k6oMnaLYlyZQmEV2vO8oDmdA_Y/edit 1/4

3. Considering your complete knowledge and experience about users' interface, how likely would you be to recommend a friend or colleague? *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very Unlikely	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Likely

4. If you know the Ford Fulkerson Algorithm how well do you believe that the application improves your understanding of the Ford Fulkerson Algorithm?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Poorly	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	In a great extent

5. If you don't know the Ford Fulkerson Algorithm how well do you believe that you understood the algorithm with the help of the application?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Poorly	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	In a great extend

6. How satisfied are you with the following: *

Mark only one oval per row.

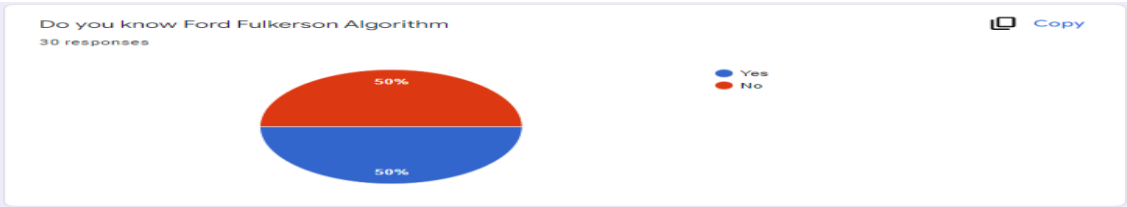
	Very dissatisfied	Not satisfied	Neutral	Satisfied	Very satisfied
Quality/Design of application					
Features supported by the application					
User Friendliness of the application					
Responsiveness of the applicataion					
Overall Product					

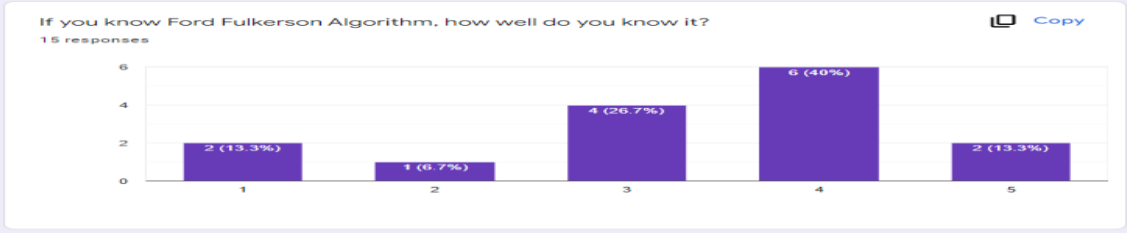
7. How difficult are the following operations? *

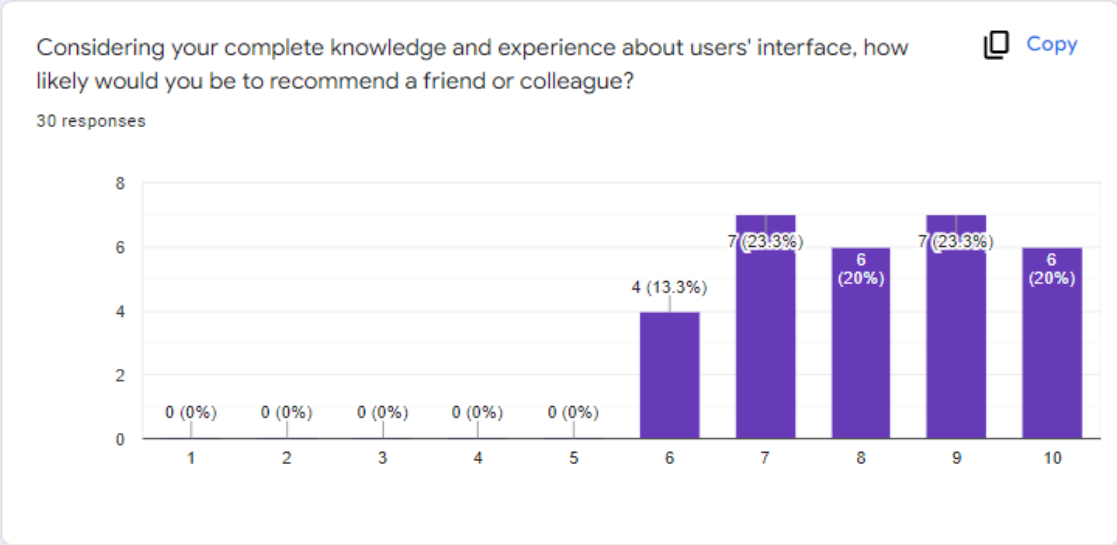
Mark only one oval per row.

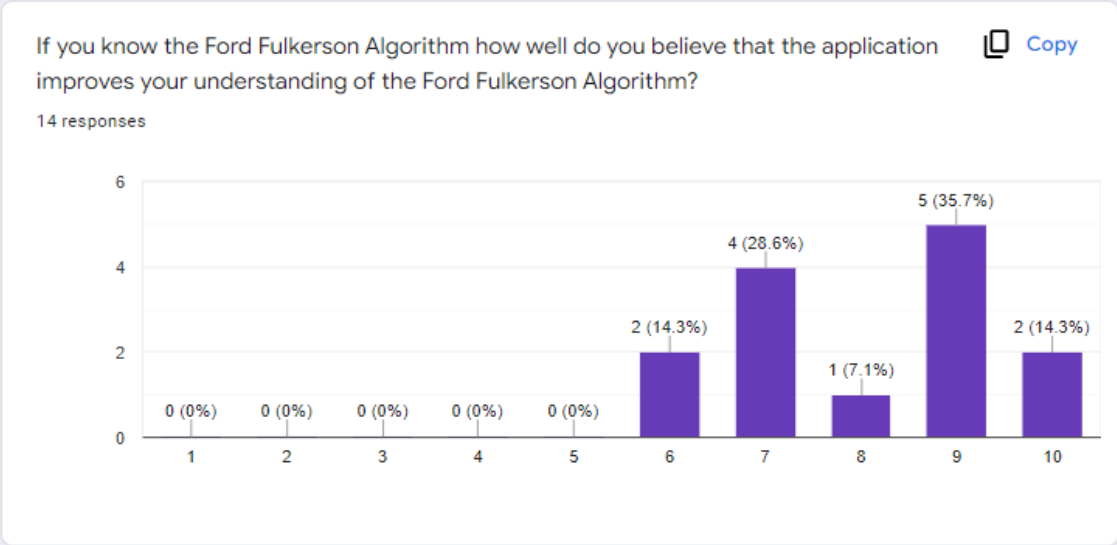
	Very difficult	Somewhat difficult	Neither difficult nor easy	Somewhat easy	Very easy
Learning to operate the system					
Exploring new features by trial and error					
Remembering names and commands used					
To understand help messages on screen					
User Friendliness of the system					
Correcting your mistakes					

Οι απαντήσεις που έλαβε το πιο πάνω ερωτηματολόγιο φαίνονται πιο κάτω.





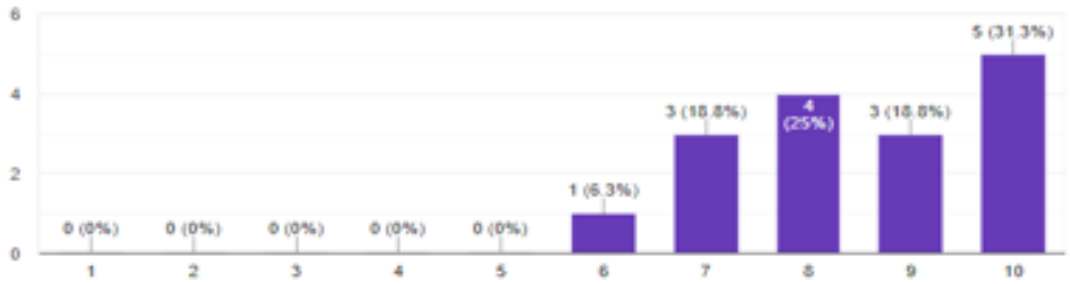




If you don't know the Ford Fulkerson Algorithm how well do you believe that you understood the algorithm with the help of the application?

Copy

16 responses



How satisfied are you with the following:

Copy

ow difficult are the following operations?

Copy