ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

ΑΛΓΟΡΙΘΜΟΙ ΓΡΑΦΩΝ ΓΙΑ ΟΔΙΚΑ ΔΙΚΤΥΑ

ΠΑΠΑΘΑΝΑΣΙΟΥ ΚΩΝΣΤΑΝΤΙΝΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας

Λαμία, 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

ΑΛΓΟΡΙΘΜΟΙ ΓΡΑΦΩΝ ΓΙΑ ΟΔΙΚΑ ΔΙΚΤΥΑ

ΠΑΠΑΘΑΝΑΣΙΟΥ ΚΩΝΣΤΑΝΤΙΝΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Βασιλακόπουλος Μιχαήλ
Αναπληρωτής Καθηγητής του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας

Λαμία, 2015

ΑΛΓΟΡΙΘΜΟΙ ΓΡΑΦΩΝ ΓΙΑ ΟΔΙΚΑ ΔΙΚΤΥΑ

ΠΑΠΑΘΑΝΑΣΙΟΥ ΚΩΝΣΤΑΝΤΙΝΑ

Τριμελής Επιτροπή:

Βασιλακόπουλος Μιχαήλ, Αναπληρωτής Καθηγητής του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας (επιβλέπων)

Μάρκου Ευριπίδης, Επίκουρος Καθηγητής του Τμήματος Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας

Λουκόπουλος Αθανάσιος, Λέκτορας του Τμήματος Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας

Ευχαριστίες

Η παρούσα πτυχιακή εργασία πραγματοποιήθηκε στο Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας από το Φεβρουάριο του 2012 έως το Φεβρουάριου του 2015.

Έχοντας ολοκληρώσει την πτυχιακή εργασία μου, θα ήθελα να ευχαριστήσω θερμά:

Τον επιβλέποντα καθηγητή μου, κ. Βασιλακόπουλο Μιχαήλ, Αναπληρωτή Καθηγητή του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας, για τη συνεχή παρακολούθηση, υποστήριξη και ενθάρρυνση κατά τη διάρκεια της εκπόνησης της παρούσας εργασίας. Όπως επίσης, και για την πολύτιμη βοήθεια και καθοδήγησή του, για την επίλυση των διάφορων θεμάτων καθ' όλη την πορεία της ανάπτυξης και της συγγραφής αυτής της εργασίας.

Τον κ. Μάρκου Ευριπίδη, μέλος της Τριμελούς Επιτροπής και Επίκουρο καθηγητή του Τμήματος Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας και τον κ. Λουκόπουλο Αθανάσιο, μέλος της Τριμελούς Επιτροπής και Λέκτορα του Τμήματος Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας, για τη συμμετοχή τους στην Εξεταστική Επιτροπή, διαβεβαιώνοντας τους ότι οι παρατηρήσεις τους θα ληφθούν σοβαρά υπ' όψιν και θα ενσωματωθούν στο τελικό κείμενο.

Το Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική του Πανεπιστημίου Θεσσαλίας και τους διδάσκοντες για το θετικό ακαδημαϊκό κλίμα και την άψογη συνεργασία μας.

Επιπλέον, θα ήθελα να απευθύνω τις ευχαριστίες μου στην οικογένειά μου, η οποία με στήριξε ποικιλοτρόπως, τόσο ηθικά όσο και υλικά, καθ' όλη τη διάρκεια της μέχρι τώρα ζωής μου.

Τέλος, την Καλαθά Ειρήνη καθώς και τον σύζυγό μου Σοϊλεμετζίδη Μιχάλη, τόσο για την εποικοδομητική βοήθεια που μου παρείχαν όσο και για την ψυχολογική υποστήριξη τους.

Κωνσταντίνα Παπαθανασίου

Λαμία, Φεβρουάριος 2015

Περίληψη

Σκοπός της παρούσας πτυχιακής εργασίας είναι η υλοποίηση μιας εφαρμογής σε java η οποία παρέχει στο χρήστη δυνατότητες εύρεσης συντομότερης διαδρομής μεταξύ δύο σημείων του οδικού δικτύου που εκείνος επιθυμεί να μεταβεί. Η εύρεση της βέλτιστης διαδρομής πραγματοποιείται με την βοήθεια τριών υλοποιημένων αλγορίθμων, του Dijkstra, του Bellman-Ford και του Floyd–Warshall. Στο πλαίσιο της υλοποίησης της ανωτέρω εφαρμογής δημιουργείται μια χωρική βάση δεδομένων με στόχο την αξιόπιστη αποθήκευση των στοιχείων, τα οποία έχουν ληφθεί από την ανοικτή βάση δεδομένων του Open Street Map.

Η αναπαράσταση τόσο του οδικού δικτύου όσο και της συντομότερης διαδρομής, η οποία προκύπτει ως αποτέλεσμα της διαδικασίας αναζήτησης επιτυγχάνεται με την βοήθεια του Java Open Street Map editor ενώ ο σχεδιασμός της εφαρμογής υλοποιείται σε γλώσσα κωδικοποίησης Java. Τέλος, πραγματοποιείται σύγκριση των αποτελεσμάτων των τριών αλγορίθμων δρομολόγησης για δύο διαφορετικού μεγέθους χάρτες. Τα αποτελέσματα της σύγκρισης έδειξαν ότι το μέγεθος του συνόλου δεδομένων επηρεάζει την αποδοτικότητα των αλγορίθμων.

Λέξεις κλειδιά: γράφος, οδικό δίκτυο, εύρεση συντομότερου μονοπατιού, χωρική βάση δεδομένων, Open Street Map (OSM), Java Open Street Map editor (JOSM), MySql, αλγόριθμος Dijkstra, αλγόριθμος Bellman-Ford, αλγόριθμος Floyd–Warshall

Abstract

The aim of this thesis is the implementation of an application in java which provides to user possibilities in order to find the shortest path between two points on the road network that he wants to go. Finding the optimal path is performed by three implemented algorithms : Dijkstra, Bellman-Ford and Floyd–Warshall. To be more specific a spatial database is created in order to store the data reliably. It is good to mention that the data was taken from the open database of Open Street Map.

The representation both of the road and the shortest path, which arises as a result of the search process, is accomplished with Java Open Street Map editor. On the other hand the application' design is implemented in Java code language. Finally, we compare the results of the three routing algorithms for two different sized maps. The comparison' results showed that the size of the dataset affects the efficiency of algorithms.

Keywords: graph, road network, shortest path, spatial data base, Open Street Map (OSM), Java Open Street Map editor (JOSM), MySql, Dijkstra algorithm, Bellman-Ford algorithm, Floyd–Warshall algorithm

Πίνακας περιεχομένων

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

Πρόλογος

Την τελευταία δεκαετία έχει παρατηρηθεί μια ραγδαία αύξηση της χρήσης συστημάτων πλοήγησης οχημάτων, απόρροια της οποίας συνιστά η αποτελεσματική εύρεση του συντομότερου μονοπατιού μεταξύ δύο σημείων μιας διαδρομής, προκειμένου να εξασφαλιστεί το καλύτερο δυνατό αποτέλεσμα για τους χρήστες του εκάστοτε δικτύου. Η αντιπροσωπευτική γεωγραφική αναπαράσταση του οδικού δικτύου, η εύρεση της γεωγραφικής θέσης του οχήματος πάνω σ ’αυτό αλλά και η εύρεση του συντομότερου μονοπατιού λαμβάνοντας υπόψη και πληροφορίες για την κυκλοφοριακή κατάσταση αποτελούν σημαντικά θέματα ενδιαφέροντος με πολλές ερευνητικές προσπάθειες να εστιάζουν στην επίλυσή τους.

Ωστόσο, παρά τους αλγόριθμους που έχουν προταθεί για την επίλυση του προβλήματος, η αναζήτηση βέλτιστης διαδρομής σε μεγάλα και πολύπλοκα δίκτυα καθιστά τη διαδικασία επίλυσης του προβλήματος απαγορευτική και ως εκ τούτου αναζητούνται καινοτόμες και βελτιωμένες μέθοδοι επίλυσης.

Στην παρούσα πτυχιακή εργασία πραγματοποιείται η υλοποίηση εφαρμογής, στην οποία δημιουργείται ένα σύστημα επίδειξης αλγορίθμων δρομολόγησης, το οποίο καταλήγει σε οπτικοποίηση του αποτελέσματος (εμφάνιση της συντομότερης διαδρομής που βρέθηκε σε χάρτη). Οι επιμέρους λειτουργίες της εφαρμογής εστιάζουν κυρίως στην αναπαράσταση ενός οδικού δικτύου, στη διαγραμματική αναπαράστασή του με τη βοήθεια ενός γράφου, στην αποθήκευση των στοιχείων του δικτύου σε μια χωρική βάση δεδομένων, στην αξιοποίηση υλοποιημένων αλγόριθμων δρομολόγησης για την εύρεση της βέλτιστης διαδρομής μεταξύ δύο σημείων του δικτύου και στην εξαγωγή γεωμετρικής περιγραφής ως αποτέλεσμα μιας διαδικασίας αναζήτησης.

Στα επιμέρους κεφάλαια της παρούσας εργασίας παρουσιάζονται βασικές ιδιότητες και ορισμοί της θεωρίας γράφων, οι κυριότεροι αλγόριθμοι αναζήτησης συντομότερης διαδρομής σε γράφους, τα βασικά χαρακτηριστικά των προγραμματιστικών εργαλείων που χρησιμοποιήθηκαν, τα επιμέρους στάδια σχεδιασμού της εφαρμογής καθώς και κάποια συμπεράσματα που προέκυψαν μετά την ολοκλήρωση εφαρμογής.

Κάθε κεφάλαιο αυτής της πτυχιακής εργασίας αριθμείται και υποδιαιρείται σε ενότητες οι οποίες αριθμούνται με δυο αριθμούς, ενώ μερικά αριθμούνται με τρεις αριθμούς. Ο πρώτος αριθμός αναφέρεται στο κεφάλαιο, ο δεύτερος στην ενότητα και ο τρίτος, όπου υπάρχει, στην υποδιαίρεσή της.

Στο πρώτο κεφάλαιο λαμβάνει χώρα μια εκτενής εισαγωγή στην θεωρία γράφων και στην γενική συμβολή της στην επιστημονική έρευνα. Επιπλέον, γίνεται αναφορά στις βασικές έννοιες της, όπως για παράδειγμα ο ορισμός του γράφου καθώς και οι σημαντικότερες ιδιότητες του.

Το δεύτερο κεφάλαιο επικεντρώνεται κυρίως στην παρουσίαση πληροφοριών που σχετίζονται με δεδομένα τα οποία αναφέρονται στον χώρο. Πιο συγκεκριμένα, ασχολείται με την διαχείριση, ανάλυση, επεξεργασία, μοντελοποίηση και απεικόνιση γεωγραφικών δεδομένων.

Η αποθήκευση των χωρικών δικτύων σε χωρικές βάσεις δεδομένων και η υλοποίηση μίας εκ των προαναφερόμενων αποτελεί το θέμα ανάπτυξης του τρίτου κεφαλαίου. Βαρύτητα δόθηκε επίσης και στην πλήρη αποσαφήνιση των προγραμματιστικών εργαλείων που συμμετείχαν στην ολοκλήρωση της συγκεκριμένης χωρικής βάσης.

Στο τέταρτο κεφάλαιο παρατίθεται μια μελέτη αλγορίθμων εύρεσης συντομότερης διαδρομής σε οδικά δίκτυα. Ειδικότερα, πραγματοποιείται ανάλυση των αλγορίθμων Dijkstra, Bellman-Ford και Floyd–Warshall με μια εκτενή παρουσίαση των χαρακτηριστικών τους, της λειτουργίας τους και της απόδοσής τους.

Το πέμπτο και κυριότερο κεφάλαιο της παρούσας πτυχιακής εργασίας αποτελεί επεξήγηση των επιμέρους σταδίων υλοποίησης της εφαρμογής μέσω παρουσίασης εικόνων αλλά και αποτελεσμάτων εκτέλεσης δοκιμών απόδοσης.

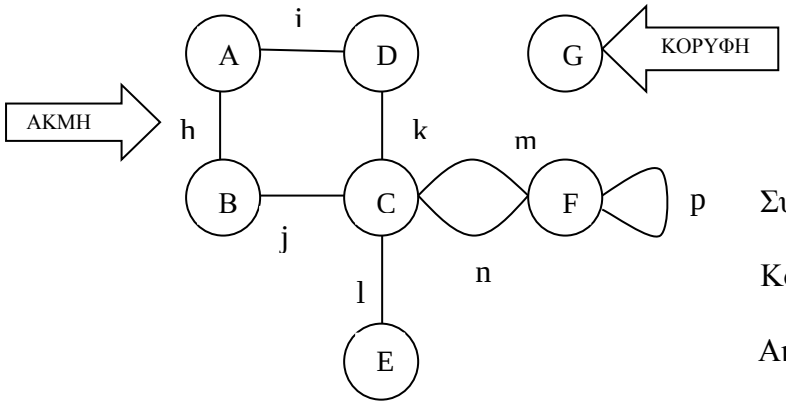
Στο τέλος της πτυχιακής εργασίας υπάρχουν τα συμπεράσματα, βιβλιογραφία με τα σχετικά συγγράμματα, τα οποία αναφέρονται στο κείμενο και πολλά από αυτά είναι χρήσιμα για περαιτέρω μελέτη των προαναφερόμενων εννοιών και εμβάθυνση, καθώς και ένα παράρτημα που περιλαμβάνει τον κώδικα υλοποίησης της εφαρμογής σε γλώσσα κωδικοποίησης java.

1ο ΚΕΦΑΛΑΙΟ
ΕΝΝΟΙΟΛΟΓΙΚΟ ΠΛΑΙΣΙΟ

Το πρώτο κεφάλαιο της παρούσας πτυχιακής εργασίας συνιστά μια εκτενή παρουσίαση των βασικών ορισμών και εννοιών της Θεωρίας Γράφων. Η προσέγγιση του αντικειμένου πραγματοποιείται κυρίως μέσω της γνώσης μερικών βασικών μαθηματικών θεμελιώσεων αλλά και με την περιγραφή αλγορίθμων, που έχουν αναπτυχθεί για την επίλυση γεωγραφικών κυρίως προβλημάτων, η αναπαράσταση των οποίων υλοποιείται με τη βοήθεια γράφων. Επιπλέον, γίνεται αναφορά σε κάποια από τα πιο γνωστά θεμελιώδη προβλήματα της Θεωρίας Γράφων, με σκοπό την πλήρη αποσαφήνιση του γνωστικού υποβάθρου.

1.1. Βασικοί ορισμοί - Γράφος

Η συμβολή της θεωρίας των γράφων όσον αφορά την επίλυση προβλημάτων σε δίκτυα δεδομένων είναι σημαντική, καθώς η δομή ενός γράφου είναι κατάλληλη για τη μοντελοποίηση των δικτύων αυτών, των αντικειμένων τους και των συνδέσεων που υφίστανται μεταξύ των αντικειμένων αυτών. Σύμφωνα με την μαθηματική θεμελίωση του όρου, ένας γράφος είναι ένα διατεταγμένο ζεύγος G = (V, E) αποτελούμενο από ένα σύνολο κορυφών ή κόμβων V(vertices) = $\{V_1, V_2, V_3, V_4 \ldots V_n\}$ όπου n>0 και από ένα σύνολο ακμών ή γραμμών E(edges) = {{x, y} με x, y ∈ V } με |E|≥0. Οι ακμές του συνόλου αυτού αποτελούν υποσύνολα δύο στοιχείων V (δηλαδή μια ακμή σχετίζεται με δύο κορυφές και η σχέση απεικονίζεται ως μη ταξινομημένο ζεύγος των κορυφών σε σχέση με τη συγκεκριμένη ακμή) (Μανωλόπουλος, 1996). Σε μια άλλη πιο απλουστευμένη εκδοχή του όρου, ο γράφος αποτελεί μία οπτική αναπαράσταση αποτελούμενη από ένα σύνολο σημείων (κορυφών ή κόμβων) που συνδέονται με γραμμές (ακμές), με σκοπό την ανάδειξη των σχέσεων που αναπτύσσουν ορισμένες ποσότητες, σχεδιασμένες σε σχέση με ένα σύνολο αξόνων.



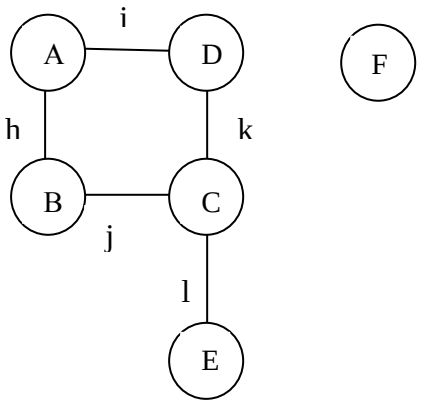
Συμβολισμός : G=(V,E)

Κορυφές: V={A,B,C,D,E,F,G}

Ακμές: E={h,i,j,k,l,m,n,p}

Εικόνα 1- Παράδειγμα γράφου

Ως υπογράφος του G = (V, E) ορίζεται ένας γράφος G’ = (V’, E’) τέτοιος ώστε V’⊆ V και E’ ⊆ E. Ένας κόμβος μπορεί να ανήκει στο σύνολο των κόμβων (V) ενός γράφου ακόμη και εάν δεν συνδέεται με κάποια από τις ακμές του ίδιου γράφου. Επιπρόσθετα, το σύνολο των ακμών του γράφου μπορεί να αποτελεί κενό σύνολο, εναντιθέση με το σύνολο τον κόμβων το οποίο απαραίτητα είναι μη κενό. Τέλος, βρόγχοι καλούνται τα σημεία στα οποία μια ακμή συνδέεται με μία κορυφή και στα δύο άκρα της (Μανωλόπουλος, 1996).



Συμβολισμός : G’=(V’,E’)

Κορυφές: V’={A,B,C,D,E,F}

Ακμές: E’={h,i,j,k,l}

Εικόνα 2- Παράδειγμα υπογράφου

1.2. Ιδιότητες γράφων

Στην υποενότητα αυτή γίνεται μία συνοπτική αναφορά των σημαντικότερων ιδιοτήτων των γράφων, οι οποίες αποτελούν τα βασικά χαρακτηριστικά διαχωρισμού και ένταξης τους στις κατηγορίες που θα αναφερθούν στην επόμενη υποενότητα.

Πρώτη ιδιότητα και ίσως το σημαντικότερο χαρακτηριστικό που προσδιορίζει έναν γράφο είναι ο βαθμός του. Ως βαθμός[1] του γράφου ορίζεται ο αριθμός των ακμών που συνδέονται με αυτόν και συμβολίζεται |E|. Αντίστοιχα, ο βαθμός ενός κόμβου ισούται με το άθροισμα των ακμών που συνδέονται σε αυτόν και συμβολίζεται |V|. Ένα σημαντικό συμπέρασμα που απορρέει από τις παραπάνω ιδιότητες, είναι ότι σε κάθε γράφο το άθροισμα των βαθμών των κόμβων του ισούται με το διπλάσιο του αριθμού των ακμών του, δηλαδή :

$$\Sigma \deg(v) = 2*|E|$$ (Μανωλόπουλος, 1996).

Ένα δεύτερο χαρακτηριστικό που αναντίρρητα προσδιορίζει έναν γράφο είναι οι σχέσεις γειτνίασης που αναπτύσσονται μεταξύ των ακμών και των κόμβων του. Ως εκ τούτου, γειτονικές χαρακτηρίζονται οι ακμές οι οποίες συνδέονται με τον ίδιο κόμβο και γειτονικοί κόμβοι αντίστοιχα εκείνοι που συνδέονται με μία κοινή ακμή. Μία διαδρομή του γράφου στην οποία μετέχει μια ακολουθία κόμβων, οι οποίοι έχουν την ιδιότητα ότι οι μεταξύ τους ακμές ανήκουν στο γράφο καλείται « μονοπάτι ». Συνεπώς, ως μονοπάτι από τον κόμβο αφετηρίας s (source) προς τον κόμβο προορισμού d (destination) στον γράφο G, ορίζεται μια πεπερασμένη ακολουθία $p=(s=n_1,n_2,\ldots, n_k=d)$ κόμβων του G, για τις οποίες ισχύει $(n_i, n_{i+1}) \in E$ για $i=1,2,\ldots,k-1$, όπου $k>0$. Αν ο κόμβος s ταυτίζεται με τον κόμβο d το μονοπάτι καλείται «κύκλος».

Το βάρος ή το κόστος ενός μονοπατιού p ισούται με το άθροισμα των βαρών των συνδέσμων που το αποτελούν. Δηλαδή :

$$\mathbf{p} = \sum_{i=1}^{k-1} \boldsymbol{w}(\boldsymbol{n_i}, \boldsymbol{n_{i+1}})$$ (Σομπόνης, 2012)

Με τον όρο βάρος εννοείται το μέγεθος που προσδιορίζει την ακμή ενός κόμβου. Το βάρος αυτό, ανάλογα με την περίπτωση μπορεί να συμβολίζει το κόστος διάσχισής της, το μήκος της ή οποιοδήποτε άλλο μέγεθος απαιτεί η εκάστοτε εφαρμογή. Το συνολικό βάρος ενός σταθμισμένου γράφου ισούται με το άθροισμα των βαρών του συνόλου των ακμών του γράφου (Joan & Robin, 2000).

[1] Ένας βρόχος συνυπολογίζεται δύο φορές στον βαθμό του γράφου.

Τέλος, ένα από τα ζητούμενα στις διάφορες εφαρμογές των γράφων είναι η εύρεση του συντομότερου μονοπατιού. Ως συντομότερο μονοπάτι από τον κόμβο s στον d, ορίζεται ένα μονοπάτι p μεταξύ των δύο κόμβων με βάρος w(s, d) , που δίνεται από την σχέση (Σομπόνης, 2012) :

$$w(s, d) = \begin{cases} \min[w(p): s \to d], \text{ αν υπάρχει το μονοπάτι απο τον } s \text{ στον } d \\ \infty, \text{ διαφορετικά} \end{cases}$$

1.3. Κατηγορίες γράφων

Αναλογιζόμενοι το γεγονός της χρήσης των γράφων σε μια πληθώρα εφαρμογών, καταλήγουμε στο συμπέρασμα πώς τα κριτήρια διαφοροποίησης που τίθενται κατά περίπτωση ποικίλουν. Η ταξινόμηση τους σε επιμέρους κατηγορίες γίνεται κυρίως βάση των δομικών χαρακτηριστικών τους, του προσανατολισμού των ακμών τους, της ύπαρξης βάρους ακμών και της συνδεσιμότητας των γράφων. Ορισμένες από τις συγκεκριμένες κατηγορίες διατυπώνονται στη συνέχεια.

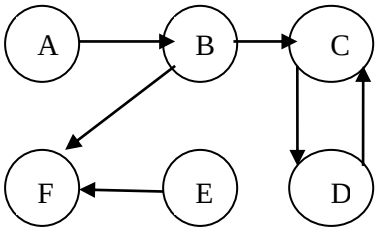
1.3.1. Κατεύθυνση ακμών

Σε ένα κατευθυντικό (Κ.Γ.) ή προσανατολισμένο γράφο κάθε ακμή είναι ένα διατεταγμένο ζεύγος κορυφών. Η ακμή (u, v) όπου u, v στοιχεία συνόλου V των ακμών του γράφου, είναι διαφορετική από την (v, u) (αφού πρόκειται για διατεταγμένο ζεύγος) (Μανωλόπουλος, 1996). Μια ακμή (u, v) λέμε ότι είναι εξερχόμενη ακμή της κορυφής u και εισερχόμενη ακμή της κορυφής v. Επιπλέον, μπορεί να υπάρχουν αυτοαναφορικές ακμές (self-loops), δηλαδή από μια κορυφή στον εαυτό της : (u, u). Ένας κατευθυντικός γράφος χωρίς αυτοαναφορικές ακμές, λέγεται απλός. Ο βαθμός ενός κόμβου σε έναν κατευθυντικό γράφο ορίζεται ως εξής (Μανωλόπουλος, 1996):

- Ο έξω-βαθμός, είναι το πλήθος των ακμών που ξεκινούν από την u.
- Ο έσω-βαθμός, είναι το πλήθος των ακμών που καταλήγουν στην u.
- Ο βαθμός της u είναι : (έσω-βαθμός) + (έξω-βαθμός).

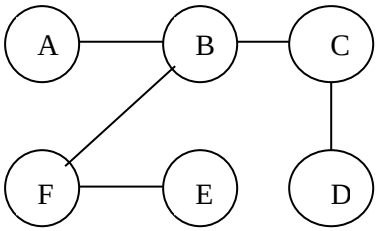
Σε έναν αμφίδρομο γράφο (Α.Μ) μια ακμή είναι ένα μη διατεταγμένο ζεύγος (u, v) όπου u, v στοιχεία συνόλου V των ακμών του γράφου και u ≠ v. Συνεπώς,

απαγορεύονται οι αυτοαναφορικές ακμές. Η ακμή (u, v) ταυτίζεται με την (v, u) (αφού πρόκειται για μη διατεταγμένο ζεύγος) και ο βαθμός ενός κόμβου u σε έναν αμφίδρομο γράφο ισούται με το πλήθος των γειτονικών ακμών του u (Μανωλόπουλος, 1996).



Συμβολισμός G=(V,E)
Κορυφές V={A,B,C,D,E,F}
Ακμές E={(A,B),(B,C),(C,D),(D,C),(B,F),(E,F)}
Έσω-βαθμός C=2 Έξω-βαθμός C=1

Εικόνα 3- Κατευθυντικός (Συνεκτικός) γράφος



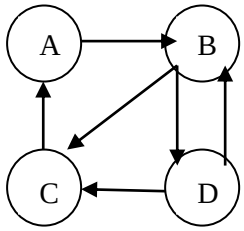
Συμβολισμός G=(V,E)
Κορυφές V={A,B,C,D,E,F}
Ακμές E={(A,B),(B,C),(C,D),(B,F),(E,F)}
Βαθμός C=2

Εικόνα 4- Αμφίδρομος (Συνεκτικός) γράφος

1.3.2. Συνδεσμικότητα γράφου

Σε έναν συνεκτικό αμφίδρομο γράφο (Εικόνα 4) κάθε κορυφή είναι προσβάσιμη από όλες τις άλλες. Σε αντίθετη περίπτωση ονομάζεται μη-συνεκτικός γράφος. Οι συνεκτικές συνιστώσες ενός γράφου είναι οι κλάσεις ισοδυναμίας των κορυφών υπό την ιδιότητα « είναι προσβάσιμη από ». Ένας αμφίδρομος γράφος είναι συνεκτικός αν έχει ακριβώς μία συνεκτική συνιστώσα.

Ισχυρά συνεκτικός κατευθυντικός γράφος είναι εκείνος στον οποίο οποιεσδήποτε δύο κορυφές είναι αμοιβαία προσβάσιμες. Οι ισχυρά συνεκτικές συνιστώσες του είναι οι κλάσεις ισοδυναμίας των κορυφών υπό την ιδιότητα « είναι αμοιβαία προσβάσιμες ». Η έννοια « αμοιβαία προσβάσιμες » σημαίνει ότι για οποιεσδήποτε δύο κορυφές, κάθε μία είναι προσβάσιμη απ' την άλλη. Ένας κατευθυντικός γράφος είναι ισχυρά συνεκτικός αν έχει ακριβώς μία ισχυρά συνεκτική συνιστώσα (Μανωλόπουλος, 1996).



Συμβολισμός G=(V,E)

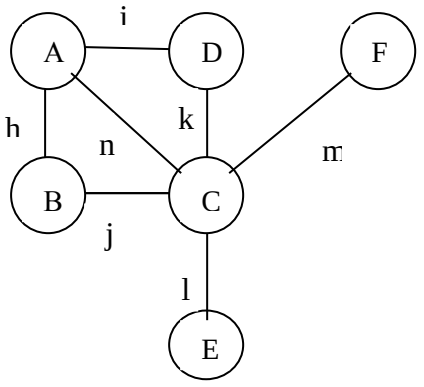
Κορυφές V={A,B,C,D}

Ακμές E={(A,B),(B,C),(D,C),(B,D),(D,B),(C,A)}

Εικόνα 5- Συνεκτικός κατευθυντικός γράφος

1.3.3. Βάρη ακμών

Ζυγισμένος ή σταθμισμένος ονομάζεται ο γράφος οι ακμές του οποίου χαρακτηρίζονται από βάρη w (Μανωλόπουλος, 1996). Τα βάρη αυτά σε ένα οδικό δίκτυο δύναται να προσδιορίζουν χιλιομετρικές αποστάσεις, κυκλοφοριακή συμφόρηση, γεωγραφικές συντεταγμένες, χρονικούς περιορισμούς και πολλά άλλα.



Συμβολισμός : G=(V,E)

Κορυφές: V={A,B,C,D,E,F}

Ακμές: E={h,i,j,k,l,m,n}

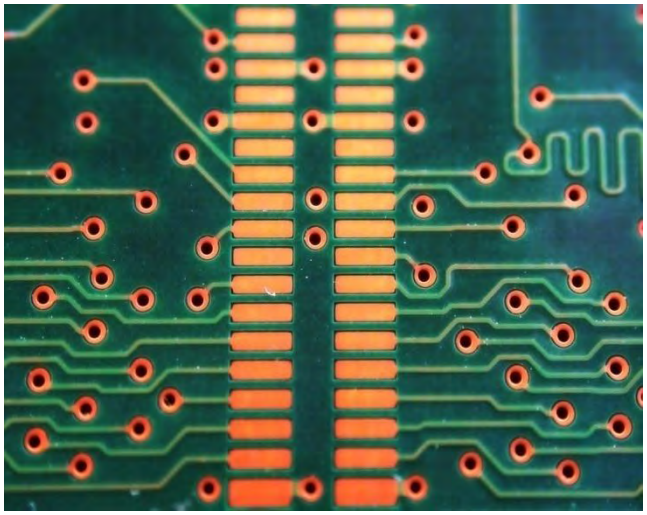
Εικόνα 6- Ζυγισμένος γράφος

1.3.4. Επιπεδικότητα

Το ερώτημα που διαχωρίζει τους γράφους και τους εντάσσει στην συγκεκριμένη κατηγορία είναι το εξής :

« Μπορεί ένας γράφος να σχεδιαστεί ώστε να μην υπάρχουν τεμνόμενες ακμές; »

Ένας γράφος G καλείται επίπεδος αν δύο οποιεσδήποτε ακμές του G συναντώνται μόνο σε προσκείμενες τερματικές κορυφές. Σε αντίθετη περίπτωση, όταν δηλαδή κάποιες πλευρές του γράφου διασταυρώνονται στο επίπεδο τότε καλείται μη επίπεδος (Σομπόνης, 2012).



Εικόνα 7- Επίπεδος γράφος

1.4. Αναπαράσταση γράφων

Ένας γράφος μπορεί εύκολα να αναπαρασταθεί με δύο διαφορετικούς τρόπους (Μανωλόπουλος, 1996):

1. Πίνακα γειτνίασης (*adjacency matrix*).
2. Λίστα γειτνίασης (*adjacency list*).

Βασική προϋπόθεση και των δύο κατηγοριών για την αναπαράσταση ενός γράφου, είναι ένα μονοσήμαντο σύστημα αρίθμησης των κόμβων.

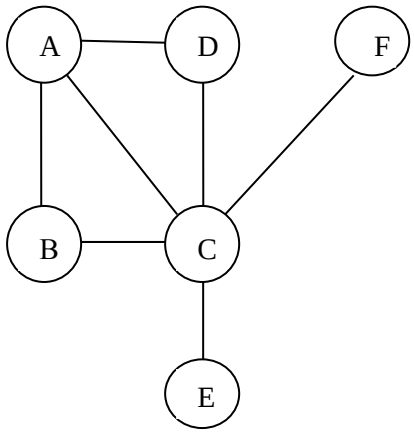
1.4.1. Αναπαράσταση με πίνακες γειτνίασης

Ο πίνακας γειτνίασης ενός γράφου G = (V, E), θεωρώντας ότι οι κορυφές είναι αριθμημένες V = {1, 2,…,n}, είναι ένας n × n πίνακας A[i, j] για τον οποίο ισχύει (Μανωλόπουλος, 1996):

$$A[i, j] = \begin{cases} 1, \text{αν } (i, j) \in E \\ 0, \text{ διαφορετικά} \end{cases}$$

Η αναπαράσταση με πίνακες γειτνίασης προτιμάται κυρίως σε περιπτώσεις όπου ο γράφος είναι πυκνός ή έχει σχετικά μικρό συνολικό μέγεθος και αυτό γιατί η απαιτούμενη μνήμη για την αναπαράσταση, ανεξάρτητα του πλήθους ακμών του γράφου είναι $\Theta(V^2)$, καθώς πρόκειται για μια πολύ πυκνή αναπαράσταση. Η υλοποίηση όμως της συγκεκριμένης αναπαράστασης είναι σχετικά απλή και επιπλέον μας δίνεται η δυνατότητα να ελέγξουμε άμεσα αν μια ακμή υπάρχει στο γράφο ή όχι (ελέγχοντας αν το αντίστοιχο στοιχείο του πίνακα είναι 1 ή 0). Τέλος, για γράφους χωρίς βάρη, απαιτείται μόνο ένα bit για κάθε στοιχείο του πίνακα (Μανωλόπουλος, 1996).

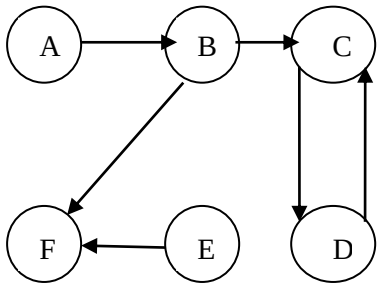
Για έναν αμφίδρομο γράφο G = (V, E) :



	A	**B**	**C**	**D**	**E**	**F**
A	0	1	1	1	0	0
B	1	0	1	0	0	0
C	1	1	0	1	1	1
D	1	0	1	0	0	0
E	0	0	1	0	0	0
F	0	0	1	0	0	0

Εικόνα 8- Αμφίδρομος γράφος - Αντίστοιχος πίνακας γειτνίασης

Για έναν κατευθυντικό γράφο G = (V, E) :



	A	**B**	**C**	**D**	**E**	**F**
A	0	1	0	0	0	0
B	0	0	1	0	0	1
C	0	0	0	1	0	0
D	0	0	1	0	0	0
E	0	0	0	0	0	1
F	0	0	0	0	0	0

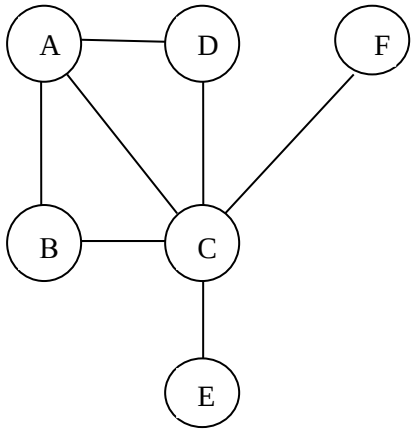
Εικόνα 9- Κατευθυντικός γράφος - Αντίστοιχος πίνακας γειτνίασης

1.4.2. Αναπαράσταση με λίστες γειτνίασης

Σε γράφο G=(V,E), λίστα γειτνίασης μιας κορυφής u ∈ V ορίζεται μια λίστα Adj[u] με τις κορυφές που είναι γειτονικές της u. Η συνολική αναπαράσταση είναι ένας πίνακας Adj με |V| το πλήθος όλων των λιστών. Σε ένα αμφίδρομο γράφο η λίστα γειτνίασης ισούται με τον βαθμό της κορυφής u, |Adj[u]| = βαθμός(u) ενώ σε έναν κατευθυντικό γράφο ισχύει : |Adj[u]| = έξω-βαθμός(u). Μια λίστα γειτνίασης Adj[u] μπορεί να θεωρηθεί είτε ως σύνολο (των γειτονικών της u) κορυφών, είτε ως κλασική συνδεδεμένη λίστα (με κόμβους - κορυφές και δείκτες) ενώ το πλήθος των στοιχείων της λίστας Adj[u] συμβολίζεται με |Adj[u]| [2]. Κύριο πλεονέκτημα της συγκεκριμένης τεχνικής αναπαράστασης είναι πως η απαιτούμενη μνήμη είναι Θ(V+E) καθώς η αναπαράσταση είναι αρκετά αραιή (Μανωλόπουλος, 1996).

[2]Αν την θεωρήσουμε ως σύνολο. Αν την θεωρήσουμε ως συνδεδεμένη λίστα, τότε είναι το μήκος της.

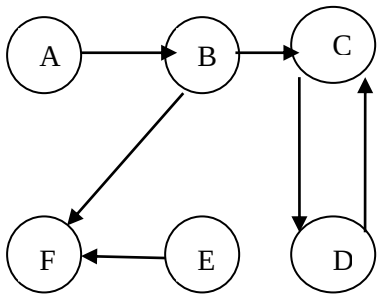
Για έναν αμφίδρομο γράφο G = (V, E) :

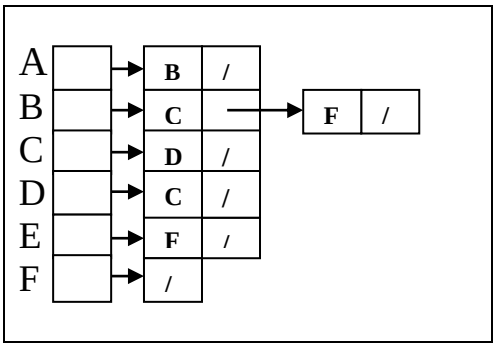


Adj[A]={B,D,C}
Adj[B]={A,C}
Adj[C]={A,B,D,E,F}
Adj[D]={A,C}
Adj[E]={C}
Adj[F]={C}

Εικόνα 10- Αμφίδρομος γράφος - Αντίστοιχη λίστα γειτνίασης

Για έναν κατευθυντικό γράφο G = (V, E) :





Εικόνα 11- Κατευθυντικός γράφος - Αντίστοιχη λίστα γειτνίασης

1.5. Διάσχιση γράφων

Στην υποενότητα αυτή θα γίνει μία γενική αναφορά σε τεχνικές διάσχισης γράφων, οι οποίες διαφέρουν μεταξύ τους στη σειρά με την οποία εξετάζουν τους κόμβους των γράφων. Η διαπέραση του συνολικού αριθμού των κόμβων ενός γράφου μπορεί να γίνει με δύο τρόπους :

- Διάσχιση κατά βάθος (DFS)
- Διάσχιση κατά πλάτος (BFS)

1.5.1. Διάσχιση γράφου κατά βάθος (DFS)

Στην διάσχιση του γράφου κατά βάθος, οι κορυφές διαπερνώνται με κατακόρυφη κατεύθυνση και όχι με οριζόντια. Συνεπώς, ακολουθείται μια διαδικασία "όσο το δυνατόν βαθύτερα". Αναλυτικότερα, ο αλγόριθμος για την διάσχιση κατά βάθος αρχίζει από την πρώτη κορυφή και μαρκάρει όσες έχει επισκεφθεί. Ο αλγόριθμος εξερευνά τις ακμές της πιο πρόσφατα εντοπισμένης κορυφής v [3] και στην περίπτωση που δεν υπάρχει τέτοιος κόμβος, γίνεται επιστροφή προς τα πίσω και συνεχίζει με τις γειτονικές της προηγούμενης κορυφής. Η διαδικασία συνεχίζεται μέχρι να ανακαλυφθούν όλες οι προσβάσιμες κορυφές από την αρχική κορυφή-αφετηρία. Εάν παραμένουν κορυφές που δεν έχουν ανακαλυφθεί επιλέγει μία απ' αυτές ως νέα κορυφή-αφετηρία και επαναλαμβάνει την προηγούμενη διαδικασία. Τέλος, όταν ανακαλυφθούν όλες οι κορυφές του γράφου ο αλγόριθμος τερματίζει. Ακολουθεί παράδειγμα του αλγορίθμου.

[3] Εξερευνά κάθε φορά το βαθύτερο και μεγαλύτερο μονοπάτι – εξού και "κατά βάθος".

ΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ DFS

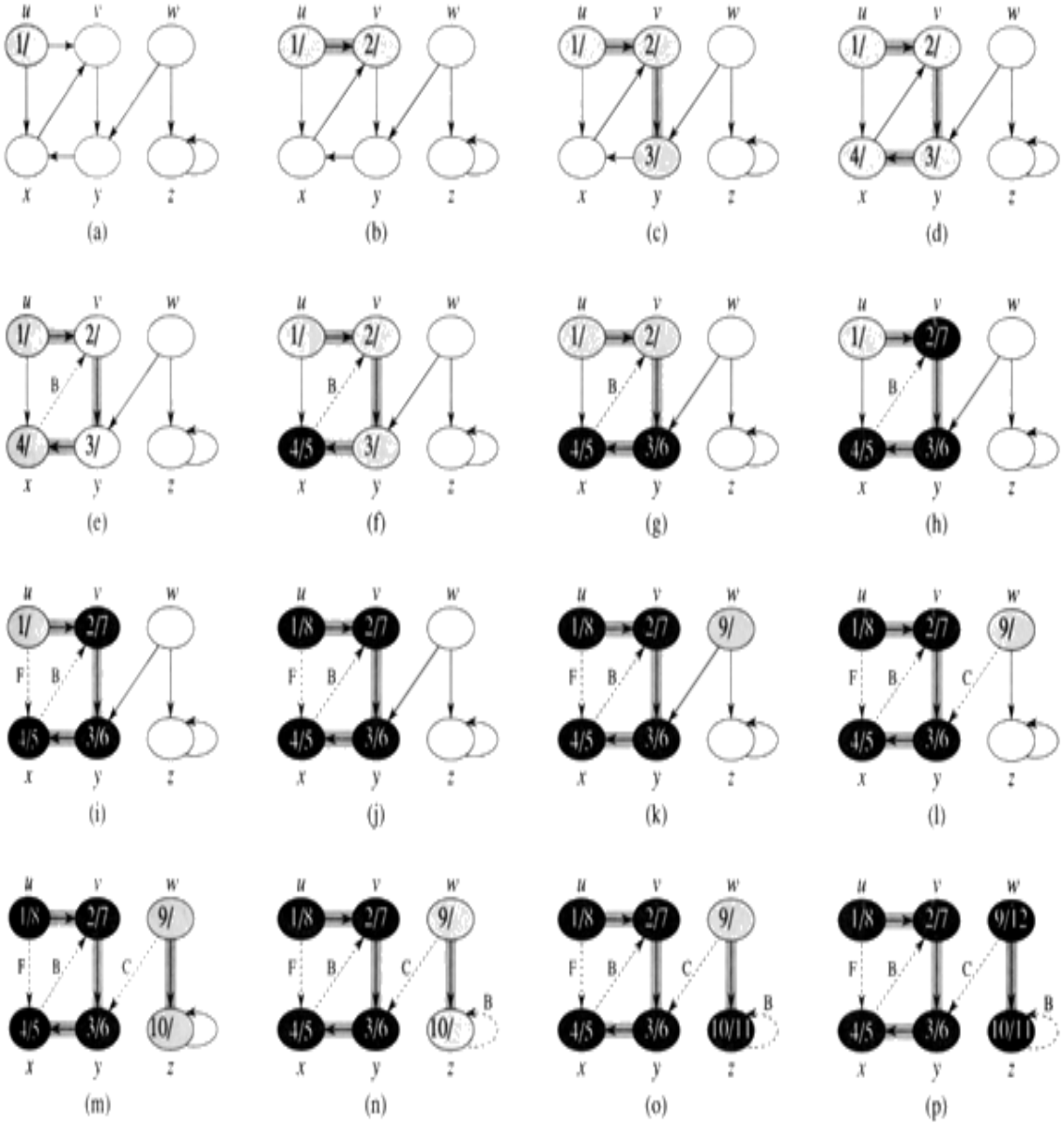
```
DFS(G)
1.  for each vertex u ∈ G.V
2.        u.color = WHITE
3.        u.π = NIL
4.  time = 0
5.  for each vertex u ∈ G.V
6.        if u.color == WHITE
7.              DFS-VISIT(G,u)
```

```
DFS-VISIT(G,u)
1.  time = time + 1
2.  u.d = time
3.  u.color = GRAY
4.  for each v ∈ G.Adj[u]
5.          if v.color ==WHITE
6.             v.π = u
7.             DFS-VISIT(G,v)
8.  u.color = BLACK
9.  time = time + 1
10. u.f = time
```

Για τον παραπάνω κώδικα υλοποίησης του αλγορίθμου DFS ισχύουν τα εξής :

- Οι κορυφές χαρακτηρίζονται με τρία χρώματα :
  - λευκό : ανεξερεύνητες κορυφές, αρχικά όλες οι κορυφές είναι λευκές
  - γκρι : κορυφές οι οποίες έχουν ανακαλυφθεί
  - μαύρο : οι κορυφές οι οποίες έχουν ολοκληρωθεί και δεν υπάρχει άλλη κορυφή στην λίστα γειτνίασης τους
- Κάθε κορυφή v έχει δύο χρονοσφραγίδες :
  - u.d : καταγράφει την ανακάλυψη της v (οπότε και γίνεται γκρίζα)
  - u.f : καταγράφει την ολοκλήρωση της v (οπότε και γίνεται μαύρη)
- Ισχύει ότι : u.d < u.f

- Το χρώμα της u είναι :
  - λευκό : για τις χρονικές στιγμές < u.d
  - γκρι : για τις χρονικές στιγμές [u.d..u.f]
  - μαύρο : για τις χρονικές στιγμές > u.f
- Τα δένδρα κατά βάθος του DFS είναι μεταξύ τους ασύνδετα εξαιτίας της ύπαρξης του χρωματισμού των κορυφών.



Εικόνα 12- Παράδειγμα λειτουργίας αλγορίθμου DFS

1.5.2. Διάσχιση γράφου κατά πλάτος (BFS)

Σε αντίθεση με τον DFS, ο αλγόριθμος BFS πραγματοποιεί διαπέραση των κόμβων του γράφου με οριζόντια κατεύθυνση. Εξαιτίας της απλότητας που παρουσιάζει η υλοποίηση του, ο συγκεκριμένος αλγόριθμος αποτελεί το αρχέτυπο για πολλούς σημαντικούς αλγορίθμους. Πρέπει να τονιστεί πως ο αλγόριθμος BFS εφαρμόζεται σε κατευθυντικούς και αμφίδρομους γράφους. Η διαδικασία υλοποίησης που ακολουθείται δοθέντος ενός γράφου G = (V, E) και μιας κορυφής-αφετηρίας s είναι η εξής :

- Αρχικά για κάθε κορυφή v του γράφου (αρχικά v = s) εξετάζονται πρώτα όλες οι ακμές της v και έπειτα ακολουθούν οι ακμές κάθε γειτονικής κορυφής της.
- Εν συνεχεία, ανακαλύπτονται όλες οι κορυφές που είναι προσβάσιμες από την s και υπολογίζονται οι αποστάσεις τους από εκείνη.
- Παράγεται ένα δένδρο κατά πλάτος με αφετηρία την s, το οποίο περιλαμβάνει όλες τις προσβάσιμες από την s κορυφές.
- Για κάθε κορυφή v προσβάσιμη από την s, το απλό μονοπάτι στο δένδρο κατά πλάτος αποτελεί και ελάχιστο μονοπάτι.
- Τέλος, όταν ανακαλυφθούν όλες οι κορυφές του γράφου ο αλγόριθμος τερματίζει.

ΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ BFS

```
BFS(G,s)
1.  for each vertex u ∈  G.V – {s}
2.        u.color = WHITE
3.        u.d = ∞
4.        u.π = NIL
5.  s.color = GRAY
6.  s.d = 0
7.  s.π = NIL
8.  Q =  Ø
9.   ENQUEUE ( Q,s )
10.    while Q ≠ Ø
11.        u = DEQUEUE (Q)
12.        for each v  ∈  G.Adj[u]
13.                if v.color ==WHITE
14.                     v.color = GRAY
```

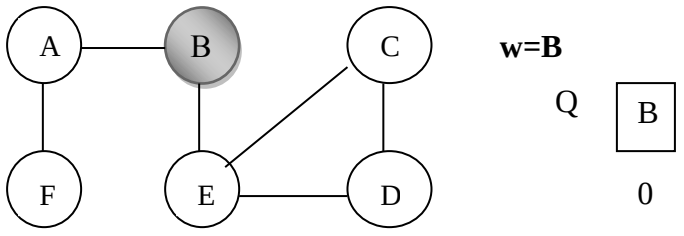
```
15.                 v.d  = v.d + 1
16.                 v.π  = u
17.                 ENQUEUE (Q,v)
18.         u.color = BLACK
```

Για τον παραπάνω κώδικα υλοποίησης του αλγορίθμου BFS ισχύουν τα εξής :

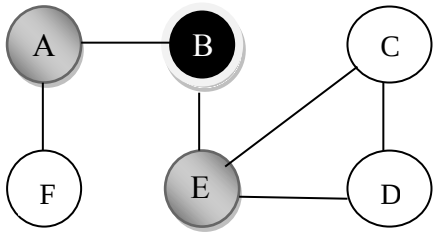
- Οι κορυφές χαρακτηρίζονται με τρία χρώματα :
  - λευκό : ανεξερεύνητες κορυφές, αρχικά όλες οι κορυφές είναι λευκές
  - γκρι : κορυφές οι οποίες έχουν ανακαλυφθεί
  - μαύρο : οι γκρίζες κορυφές για τις οποίες ισχύει πως και όλες οι γειτονικές τους κορυφές έχουν ανακαλυφθεί
- Οι ιδιότητες κάθε κορυφής είναι :
  - u.color : το χρώμα της κορυφής
  - u.π : η προηγούμενη κορυφή ,στην περίπτωση που δεν υπάρχει τότε ισούται με NIL
  - u.d : η απόσταση από την αρχική ρίζα που αποτελεί την αφετηρία
- Η ουρά Q που περιέχει τις γκρίζες κορυφές είναι μια δομή δεδομένων FIFO (first in-first out). Σε κάθε επανάληψη οι κορυφές που χρωματίζονται γκρι εισέρχονται στην ουρά, ενώ όσες εξέρχονται από αυτή χρωματίζονται με μαύρο χρώμα.

ΕΚΤΕΛΕΣΗ ΑΛΓΟΡΙΘΜΟΥ

ΠΡΩΤΟ ΒΗΜΑ



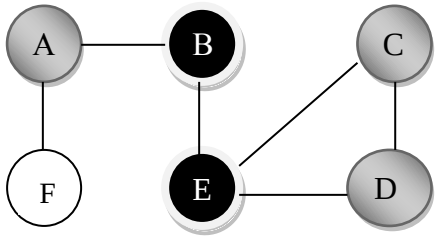
ΔΕΥΤΕΡΟ ΒΗΜΑ



Q

E	A
1	1

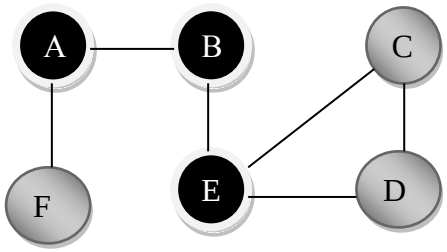
ΤΡΙΤΟ ΒΗΜΑ



Q

A	C	D
1	2	2

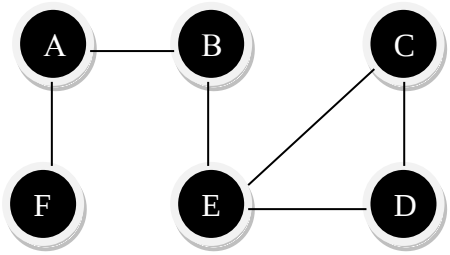
ΤΕΤΑΡΤΟ ΒΗΜΑ



Q

C	D	F
2	2	2

ΠΕΜΠΤΟ ΒΗΜΑ



Q

Ø

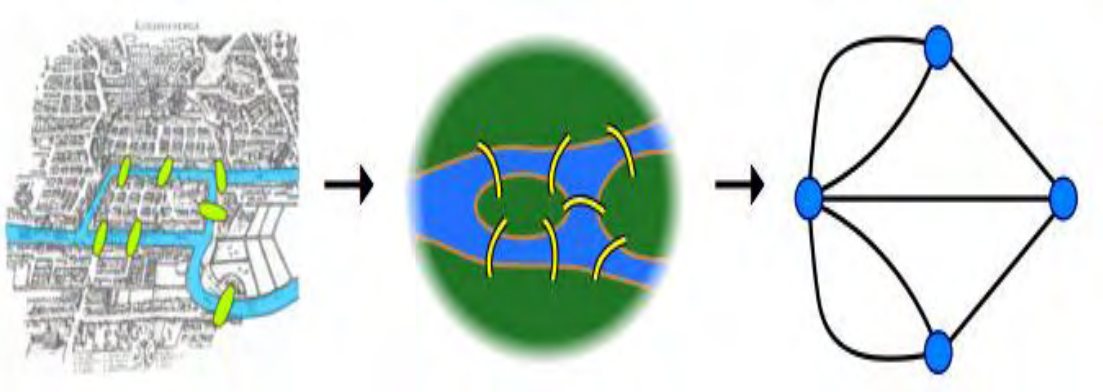
1.6. Γνωστά γεωγραφικά προβλήματα

Η αρχή του γνωστικού υποβάθρου που παρέχει η θεωρία γράφων σε διάφορους επιστημονικούς τομείς σήμερα, τέθηκε περίπου στα μέσα του 18ου αιώνα. Η θεμελίωση του πρώτου προβλήματος από τον Ελβετό φυσικό και μαθηματικό Λέοναρντ Πωλ Όυλερ το 1736, αποτέλεσε την απαρχή για την εξέλιξη της θεωρίας γράφων σε ένα εκτεταμένο επιστημονικό πεδίο μελέτης προβλημάτων με εφαρμογή σε επιστημονικούς αλλά και μη επιστημονικούς κλάδους. Η παρουσίαση ορισμένων εκ των προαναφερόμενων συνιστά το αντικείμενο της παρούσας υποενότητας.

1.6.1. Οι γέφυρες του Königsburg

Όπως αναφέρθηκε και στην εισαγωγή της παρούσας υποενότητας, το 1973 ο Ελβετός φυσικός και μαθηματικός Λέοναρντ Πωλ Όυλερ διατύπωσε το γνωστό πρόβλημα των γεφυρών του Königsburg. Το βασικό ερώτημα του χωρικού αυτού προβλήματος ήταν εύρεση μίας διαδρομής βάση της οποίας κάποιος θα είχε την δυνατότητα να διασχίσει και τις επτά γέφυρες της πόλης του Königsburg το πολύ μία φορά, έχοντας το σημείο εκκίνησης της διαδρομής και ως σημείο τερματισμού (Euler, 1741).

Οι γέφυρες του Königsberg δημιούργησαν έναν διάσημο μαθηματικό γρίφο για τον οποίο όμως ο Λέοναρντ Όυλερ κατέληξε στο απλό συμπέρασμα ότι το πρόβλημα δεν είχε λύση, δηλαδή δεν υπήρχε κανένας τρόπος να διασχίσει κάποιος και τις επτά γέφυρες περνώντας από κάθε μία μόνο μία φορά. (Euler, 1741)



Εικόνα 13 - Οι επτά γέφυρες του Konigsberg (Οι γέφυρες του Königsberg , 2014)

Προς τιμήν της μελέτης του Ελβετού φυσικο - μαθηματικού Όυλερ, οποιοδήποτε γράφημα έχει τις ίδιες ιδιότητες με το γράφημα αναπαράστασης των γεφυρών της πόλης του Königsburg ονομάζεται δίκτυο Euler. Δηλαδή, εάν ένα γράφημα είναι συνεκτικό με άρτιο αριθμό βαθμών, τότε πρόκειται για γράφημα Euler στο οποίο κάθε πλευρά μπορεί να διασχιστεί μία μόνο φορά αρχίζοντας και τελειώνοντας στην ίδια κορυφή.

1.6.2. Πρόβλημα περιοδεύοντος πωλητή

Η διατύπωση του προβλήματος των επτά γεφυρών του Königsberg αποτέλεσε την βάση σε παρόμοια προβλήματα που τίθενται σήμερα και αφορούν δίκτυα και την επίλυση προβλημάτων που εντοπίζονται σε αυτά, όπως για παράδειγμα το πρόβλημα του περιοδεύοντος πωλητή (Traveling Salesman Problem - TSP). Το συγκεκριμένο πρόβλημα συνιστά ένα από τα βασικότερα προβλήματα που αναφέρεται στην βιβλιογραφία της θεωρίας γράφων και πραγματεύεται την εύρεση του συντομότερου μονοπατιού, σε ένα δεδομένο σύνολο πόλεων με καθορισμένη απόσταση μεταξύ κάθε ζεύγους πόλεων, με τον περιορισμό της επίσκεψης σε κάθε πόλη ακριβώς μια φορά.

Αναλυτικότερα, για να επιτευχθεί ο παραπάνω στόχος ένας πωλητής πρέπει να ξεκινήσει από την πόλη του και στο τέλος πρέπει να επιστρέψει σ’ αυτή. Ο πωλητής πρέπει να επιλέξει την σειρά με την οποία θα επισκεφτεί τις πόλεις έτσι ώστε η απόσταση που θα διανύσει να είναι η μικρότερη δυνατή. Προφανώς, έτσι θα κερδίσει χρόνο και καύσιμο. Καθώς ο πωλητής προσπαθεί να βρει την συντομότερη διαδρομή, αντιμετωπίζει το αποκαλούμενο πρόβλημα του πλανόδιου πωλητή.

Το παραπάνω πρόβλημα συμπεριλαμβάνεται στο σύνολο των NP - Complete προβλημάτων, που σημαίνει ότι δεν υπάρχει κάποια γνωστή τεχνική επίλυσης που να είναι σημαντικά καλύτερη πέρα από την απλή αναζήτηση όλων των πιθανών εναλλακτικών (Parallel Distributed Processing Laboratory (PDP Lab), 2014).

1.6.3. Το πρόβλημα του Κινέζου ταχυδρόμου

Ομοίως με τα δύο προηγούμενα προβλήματα, το πρόβλημα του κινέζου ταχυδρόμου (chinese postman problem), το οποίο ονομάστηκε έτσι γιατί τέθηκε από ένα κινέζο μαθηματικό, τον M.K Kwan το 1962, έχει ως σκοπό την εύρεση της συντομότερης και ωφελιμότερης διαδρομής, η οποία θα διασχίζει όλες τις κορυφές ενός γραφήματος μία μόνο φορά και θα έχει κοινή κορυφή αφετηρίας και τερματισμού (Μαθήματα Θεωρίας Γράφων, 1996). Στην προκειμένη περίπτωση, οι κορυφές του γραφήματος είναι οι σταθμοί παράδοσης της αλληλογραφίας. Το πρόβλημα αυτό διατυπώθηκε στα ταχυδρομεία της Κίνας προκειμένου να ελαττωθεί ο αριθμός των ταχυδρόμων που απαιτούνταν για την διανομή της αλληλογραφίας και έτσι να επιτευχθεί εξοικονόμηση χρημάτων και ωρών εργασίας. Η εύρεση της βέλτιστης λύσης οδήγησε στην δημιουργία γραφημάτων αναπαράστασης των σταθμών που έπρεπε να επισκεφθούν οι ταχυδρόμοι. Οι σταθμοί ή οι πόλεις αναπαριστώνται από τις κορυφές, ενώ οι ακμές δείχνουν τις χιλιομετρικές αποστάσεις μεταξύ τους. Κατά παρόμοιο τρόπο δημιουργούνται και οι οδικοί χάρτες των πόλεων.

2ο ΚΕΦΑΛΑΙΟ
ΧΩΡΙΚΑ ΔΕΔΟΜΕΝΑ

Το δεύτερο κεφάλαιο της παρούσας πτυχιακής εργασίας επικεντρώνεται σε ένα θέμα το οποίο τις τελευταίες δεκαετίες έχει απασχολήσει πολλούς ερευνητές από διάφορους επιστημονικούς κλάδους. Συγκεκριμένα, ασχολείται με την διαχείριση και ανάλυση γεωγραφικών δεδομένων αλλά και τον τρόπο με τον οποίο γίνεται η επεξεργασία, μοντελοποίηση και απεικόνιση τους. Το ζήτημα αυτό της διαχείρισης των δεδομένων τα οποία αναφέρονται στο χώρο (μεταβάλλονται με το χρόνο κάποιες φορές) και βρίσκονται αποθηκευμένα σε μια χωρική βάση δεδομένων αποτελεί την βασική λειτουργία ενός Συστήματος Διαχείρισης Βάσεων Δεδομένων χωρικών πληροφοριών. Στη συνέχεια επιχειρείται μια προσέγγιση των σημαντικότερων στοιχείων και χαρακτηριστικών αυτών των συστημάτων.

2.1. Συστήματα Διαχείρισης Βάσεων Δεδομένων

Η ραγδαία αύξηση της τεχνολογίας αποτέλεσε τον κυριότερο παράγοντα για συλλογή χωρικών δεδομένων, ο τεράστιος όγκος των οποίων απαιτεί την αξιόπιστη αποθήκευσή τους. Μία χωρική βάση δεδομένων εκτός από τις βασικές λειτουργίες που παρέχει μια τυπική βάση δεδομένων (ταχεία και αξιόπιστη αναζήτηση και προσπέλαση δεδομένων), δίνει επιπλέον δυνατότητες. Οι δυνατότητες αυτές αφορούν την αποθήκευση και την επεξεργασία χωρικών πληροφοριών και τύπων όπως είναι οι γεωγραφικές συντεταγμένες και η ακριβής θέση ενός αντικειμένου στο χώρο (παρέχεται η δυνατότητα για πολυδιάστατους χώρους), την αποθήκευση ολόκληρων σχημάτων και των ιδιοτήτων τους καθώς και την υποστήριξη πράξεων μεταξύ τους, βάση της διαφορετικής φύσης και των χωρικών σχέσεων που αναπτύσσονται κατά περίπτωση, ανάλογα με τον τρόπο και τα συστήματα που χρησιμοποιούνται για τη συλλογή τους αλλά και το σκοπό για τον οποίο συλλέγονται (S.Shekhar & S.Chawla, 2003).

Ένα Σύστημα Διαχείρισης Βάσεων Χωρικών Δεδομένων (ΣΔΒΧΔ) συνιστά ένα υπολογιστικό σύστημα σχεδιασμένο για να υποστηρίζει τη διαχείριση, ανάλυση, απεικόνιση και διάχυση των γεωγραφικών πληροφοριών για αποτελεσματική, έγκαιρη και αποδοτική συνεργασία, επίλυση προβλημάτων και λήψη αποφάσεων. Οι τέσσερις βασικές λειτουργίες (Marathon Data Systems, 2011) ενός ΣΔΧΒΔ είναι :

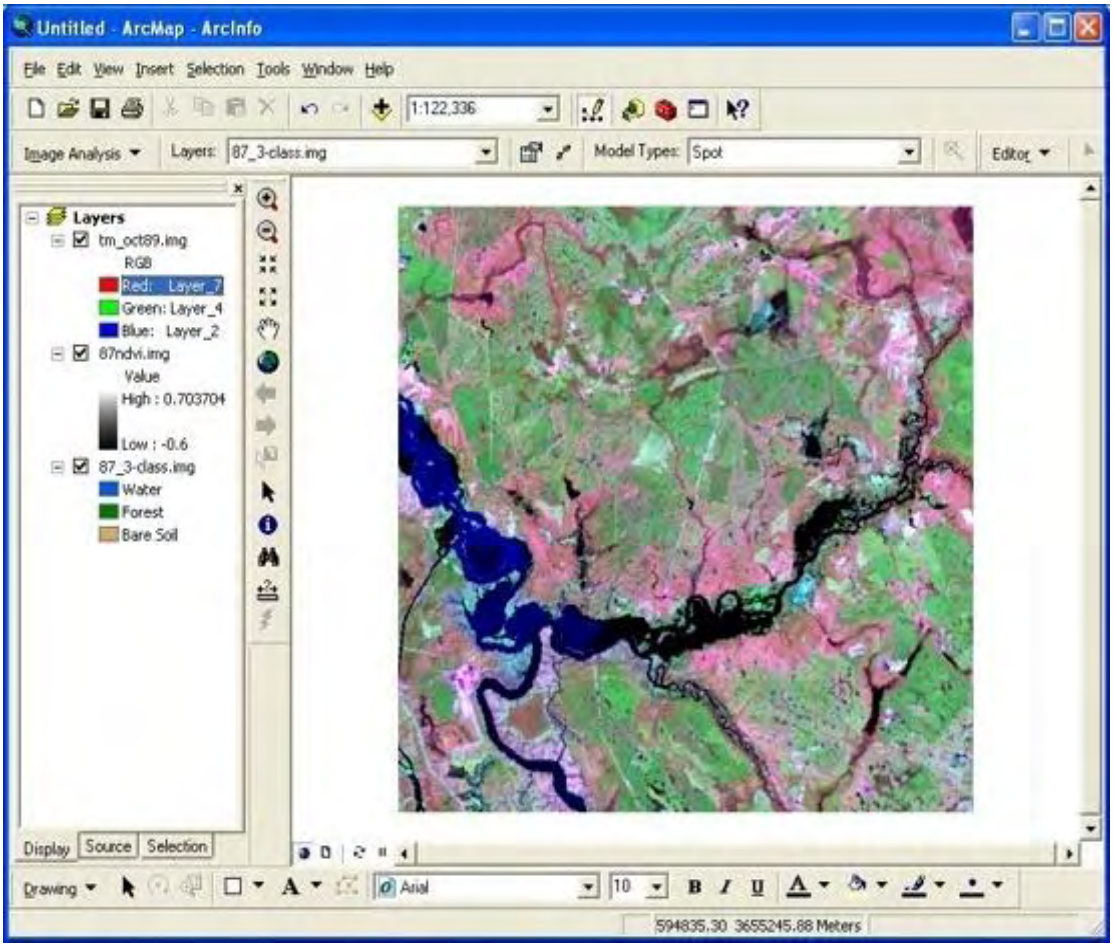
1. Χαρτογράφηση και απεικόνιση
2. Διαχείριση γεωγραφικών δεδομένων
3. Δημιουργία και επεξεργασία δεδομένων
4. Γεωγραφική ανάλυση

Ως εκ τούτου, κάποιες από τις βασικές εφαρμογές που παρέχονται από ένα ΣΔΒΧΔ είναι : η διασφάλιση της προσπέλασης και της ανάκτησης των δεδομένων γρήγορα από όσους χρήστες ζητηθεί, η εκτέλεση ερωτημάτων και η επιλογή μέσα από αναζητήσεις, χωρικών και μη στοιχείων καθώς και η αντιγραφή, μετακίνηση, διαγραφή και απεικόνιση των δεδομένων, πριν << προστεθούν >> σε ένα χάρτη ή σε οποιαδήποτε άλλη εφαρμογή. Ένα ΣΔΒΧΔ είναι ειδικά σχεδιασμένο για την επισήμανση μοτίβων και σχέσεων μεταξύ των γεωγραφικών δεδομένων και επιπλέον παρέχει μεταδεδομένα, επεξηγήσεις και υποσημειώσεις των στοιχείων τα οποία μπορούν να διαβαστούν είτε να δημιουργηθούν. Επιπρόσθετα, καθιστά δυνατή την χρήση χωρικών τελεστών και μηχανισμών δεικτοδότησης των χωρικών δεδομένων.

Βασικό χαρακτηριστικό των συστημάτων αυτών, το οποίο τα καθιστά πολύτιμα εργαλεία για πολλούς επιστημονικούς κλάδους, είναι η δυνατότητα παράλληλης σύνδεσής τους με άλλα συστήματα βάσεων δεδομένων (S.Shekhar & S.Chawla, 2003). Οι εφαρμογές ενός ΣΔΒΧΔ εμφανίζονται στο Δημόσιο, στον Ιδιωτικό και Ακαδημαϊκό χώρο και στη Τοπική Αυτοδιοίκηση και καλύπτουν αντικείμενα όπως (Marathon Data Systems, 2011) :

- Δάση - Πυρκαγιές
- Τηλεπικοινωνίες - Δορυφορικά Δεδομένα
- Τρισδιάστατα Μοντέλα
- Πωλήσεις – Marketing
- Στατιστικά
- Οδοποιία - Μεταφορές - GPS
- Αποχέτευση – Ύδρευση

- Internet Εφαρμογές
- Χωροταξία – Χαρτογραφία
- Γεωργία – Γεωλογία
- Κτηματολόγιο - Τοπική Αυτοδιοίκηση
- Βάσεις Δεδομένων
- Τουρισμός
- Εκπαίδευση



Εικόνα 14 - Παράδειγμα ΣΔΒΧΔ ArcGis (Gis Mapping Software,Solutions, Services,Map Apps and Data)

2.1.1. Κατηγορίες Συστημάτων Διαχείρισης Χωρικών Βάσεων Δεδομένων

Τα Συστήματα Διαχείρισης Χωρικών Βάσεων Δεδομένων διακρίνονται σε δύο κύριες κατηγορίες οι οποίες αποτελούν και χρονικούς σταθμούς στην ιστορία της δημιουργίας τους :

2.1.1.1. Σχεσιακά Συστήματα Διαχείρισης Χωρικών Βάσεων Δεδομένων

Το μοντέλο των συγκεκριμένων συστημάτων διαχειρίζεται συλλογές δεδομένων οργανωμένων σε συσχετιζόμενους πίνακες, οι οποίοι παρέχουν μηχανισμούς για ανάγνωση, εγγραφή, τροποποίηση αλλά και για λογικές πράξεις πάνω στα σύνολα δεδομένων. Όπως προαναφέρθηκε, κατά το σχεσιακό μοντέλο οι λογικές εγγραφές ομαδοποιούνται σε πίνακες οι οποίοι υπόκεινται στους ακόλουθους βασικούς περιορισμούς (Compucon SA, 2014) :

- Η τομή μεταξύ μιας γραμμής και μιας στήλης κάθε πίνακα περιέχει μια μόνο τιμή του πεδίου της στήλης.
- Οι τιμές κάθε στήλης είναι του ίδιου τύπου.
- Κάθε στήλη έχει μοναδικό όνομα.
- Οι γραμμές (λογικές εγγραφές) κάθε πίνακα πρέπει να διαφέρουν μεταξύ τους τουλάχιστον κατά την τιμή ενός πεδίου. Ο περιορισμός αυτός σημαίνει ότι οι πίνακες περιέχουν λογικές εγγραφές που μπορούν να προσδιοριστούν μοναδικά αναφέροντας τον κατάλληλο συνδυασμό τιμών των πεδίων.
- Η διάταξη των γραμμών και στηλών κάθε πίνακα δεν έχει σημασία. Αυτό σημαίνει ότι είναι δυνατό να αλλάξει η διάταξη των γραμμών και των στηλών ενός πίνακα χωρίς να επηρεαστεί το πληροφοριακό του περιεχόμενο.

Παρά το γεγονός ότι τα συστήματα αυτά είναι τα πιο δημοφιλή στις περισσότερες εφαρμογές διαχειριστικού τύπου (τραπεζικά συστήματα, συστήματα κράτησης θέσεων, κλπ), δεν παρέχουν στο χρήστη δυνατότητες για πιο πολύπλοκες διαδικασίες πάνω στα δεδομένα.

Αδυναμίες των σχεσιακών συστημάτων (Compucon SA, 2014)

- **Αδυναμία αναπαράστασης του πραγματικού κόσμου** : η διαδικασία της κανονικοποίησης οδηγεί στη δημιουργία σχέσεων (πινάκων), οι οποίες δεν αντιστοιχούν σε οντότητες του πραγματικού κόσμου. Ο διαμοιρασμός των δεδομένων σε πολλούς πίνακες οδηγεί στην εκτέλεση πολλών χρονοβόρων πράξεων σύνδεσης.

- **Ομοιογένεια** : το σχεσιακό μοντέλο ορίζει ότι κάθε γραμμή ενός πίνακα πρέπει να αποτελείται από τις ίδιες στήλες και κάθε στήλη του πίνακα πρέπει να δέχεται τιμές από το ίδιο πεδίο ορισμού. Οι δύο αυτές ιδιότητες καλούνται οριζόντια και κάθετη ομοιογένεια. Η δομή αυτή του πίνακα είναι αρκετά περιοριστική για αντικείμενα του πραγματικού κόσμου τα οποία έχουν πολύπλοκη δομή.
- **Περιορισμένες λειτουργίες** : ένα σχεσιακό ΣΔΒΔ έχει ένα περιορισμένο σύνολο λειτουργιών που μπορούν να εφαρμοσθούν στα δεδομένα, το οποίο καθορίζεται από τη γλώσσα SQL. Στις σύγχρονες εφαρμογές το σύνολο των λειτουργιών αυτών δεν επαρκεί και επομένως απαιτείται η δυνατότητα ορισμού νέων τύπων δεδομένων και νέων λειτουργιών.
- **Οι αλλαγές στο σχήμα της ΒΔ είναι χρονοβόρες** : αν απαιτηθεί αλλαγή στο σχήμα της ΒΔ θα πρέπει ο διαχειριστής να διακόψει προσωρινά τη λειτουργία του συστήματος και επιπλέον τα προγράμματα εφαρμογής πρέπει να διαμορφωθούν αναλόγως.
- **Συναλλαγές μικρής διάρκειας** : οι συναλλαγές στις παραδοσιακές εφαρμογές είναι συνήθως μικρής διάρκειας με αποτέλεσμα οι μηχανισμοί ταυτόχρονης εκτέλεσης, όπως το κλείδωμα δύο φάσεων, να είναι επαρκείς. Αν οι συναλλαγές έχουν μεγάλη διάρκεια, το οποίο παρατηρείται συνεχώς σε πολύπλοκα αντικείμενα, οι ανωτέρω μηχανισμοί δεν επαρκούν.

2.1.1.2. Αντικειμενοστραφή Συστήματα Διαχείρισης Χωρικών Βάσεων Δεδομένων

Σκοπός των αντικειμενοστραφών συστημάτων ήταν να καλύψουν τις ελλιπείς δυνατότητες που παρείχε το σχεσιακό μοντέλο συστημάτων μέσα από την αξιοποίηση χαρακτηριστικών του αντικειμενοστραφούς προγραμματισμού. Κατά συνέπεια, στα συγκεκριμένα συστήματα ένα αντικείμενο (object) συνιστά μία οντότητα του πραγματικού κόσμου και έτσι ο χρήστης έχει την δυνατότητα να ορίσει ακόμα και νέους τύπους δεδομένων (Compucon SA, 2014). Τα αντικειμενοστραφή συστήματα παρέχουν επιπλέον δυνατότητες, κυρίως στις λειτουργίες της αποθήκευσης αντικειμένων, της διαχείρισης μεγάλων βάσεων δεδομένων (απαιτείται αποτελεσματική διαχείριση δευτερεύουσας μνήμης), της επεξεργασίας πολύπλοκων

ερωτημάτων, του συγχρονισμού ταυτόχρονων προσπελάσεων σε αντικείμενα αλλά και της επανάκτησης δεδομένων.

Πλεονεκτήματα των αντικειμενοστραφών συστημάτων (Compucon SA, 2014)

- **Ενισχυμένες δυνατότητες μοντελοποίησης** : το αντικειμενοστραφές μοντέλο δεδομένων επιτρέπει την αναπαράσταση του πραγματικού κόσμου με μεγαλύτερη ακρίβεια σε σχέση με το σχεσιακό μοντέλο. Η έννοια του αντικειμένου που περιλαμβάνει δεδομένα και μεθόδους ανταποκρίνεται περισσότερο στην πραγματικότητα. Με τις ιεραρχίες εξειδίκευσης και συμπερίληψης μοντελοποιούνται σχετικά εύκολα ακόμη και πολύπλοκες οντότητες του πραγματικού κόσμου.
- **Επεκτασιμότητα** : σε ένα αντικειμενοστραφές ΣΔΒΔ μπορούμε να ορίσουμε νέες κλάσεις αντικειμένων χρησιμοποιώντας τις ήδη υπάρχουσες. Με τη χρήση της απλής και της πολλαπλής κληρονομικότητας και την εφαρμογή του πολυμορφισμού μπορούμε να παράγουμε νέες κλάσεις αντικειμένων που κληρονομούν χαρακτηριστικά και μεθόδους άλλων κλάσεων. Έτσι αποφεύγεται η επανάληψη του ορισμού χαρακτηριστικών και μεθόδων που έχουν ήδη ορισθεί σε άλλες κλάσεις.
- **Εξέλιξη σχήματος Βάσης Δεδομένων** : οι δυνατότητες του αντικειμενοστραφούς μοντέλου δεδομένων και οι ευκολίες που παρέχει το ΣΔΒΔ μπορούν να χρησιμοποιηθούν για την εξέλιξη του σχήματος της βάσης δεδομένων (schema evolution). Η εξέλιξη του σχήματος είναι αρκετά δύσκολη στα σχεσιακά συστήματα διότι επιφέρει πολλές αλλαγές στη λογική των εφαρμογών.
- **Υποστήριξη συναλλαγών μεγάλης διάρκειας** : ο τρόπος διαχείρισης κατά την εκτέλεση των ταυτόχρονων συναλλαγών από τα σχεσιακά συστήματα οδηγεί σε περιορισμένη απόδοση όταν οι συναλλαγές έχουν μεγάλη διάρκεια. Τα αντικειμενοστραφή συστήματα υλοποιούν διαφορετικούς μηχανισμούς με αποτέλεσμα οι συναλλαγές μεγάλης διάρκειας να εκτελούνται πιο αποδοτικά.

Μειονεκτήματα των αντικειμενοστραφών συστημάτων (Compucon SA, 2014)

- **Πολυπλοκότητα** : οι μηχανισμοί αποθήκευσης αντικειμένων, διαχείρισης συναλλαγών, εξέλιξης σχήματος της βάσης και επεξεργασίας ερωτημάτων είναι αρκετά πολύπλοκοι. Η πολυπλοκότητα αυτή οδηγεί στην υλοποίηση συστημάτων που είναι πιο απαιτητικά και δυσκολότερα στη διαχείρισή τους.
- **Έλλειψη υποστήριξης όψεων** : πολλά από τα σύγχρονα αντικειμενοστραφή συστήματα δεν υποστηρίζουν μηχανισμό όψεων, παρά τα πολλά πλεονεκτήματα που προσφέρουν.
- **Έλλειψη καθορισμένου μοντέλου** : παρά τα πλεονεκτήματα του αντικειμενοστραφούς μοντέλου δεν υπάρχει κάποιο συγκεκριμένο πρότυπο που να ακολουθείται από όλους τους κατασκευαστές συστημάτων.

2.2. Αρχιτεκτονική Συστήματος Διαχείρισης Βάσεων Χωρικών Δεδομένων

Ένα ΣΔΒΧΔ συνιστά ένα λογισμικό το οποίο (S.Shekhar & S.Chawla, 2003):

Σε λογικό επίπεδο

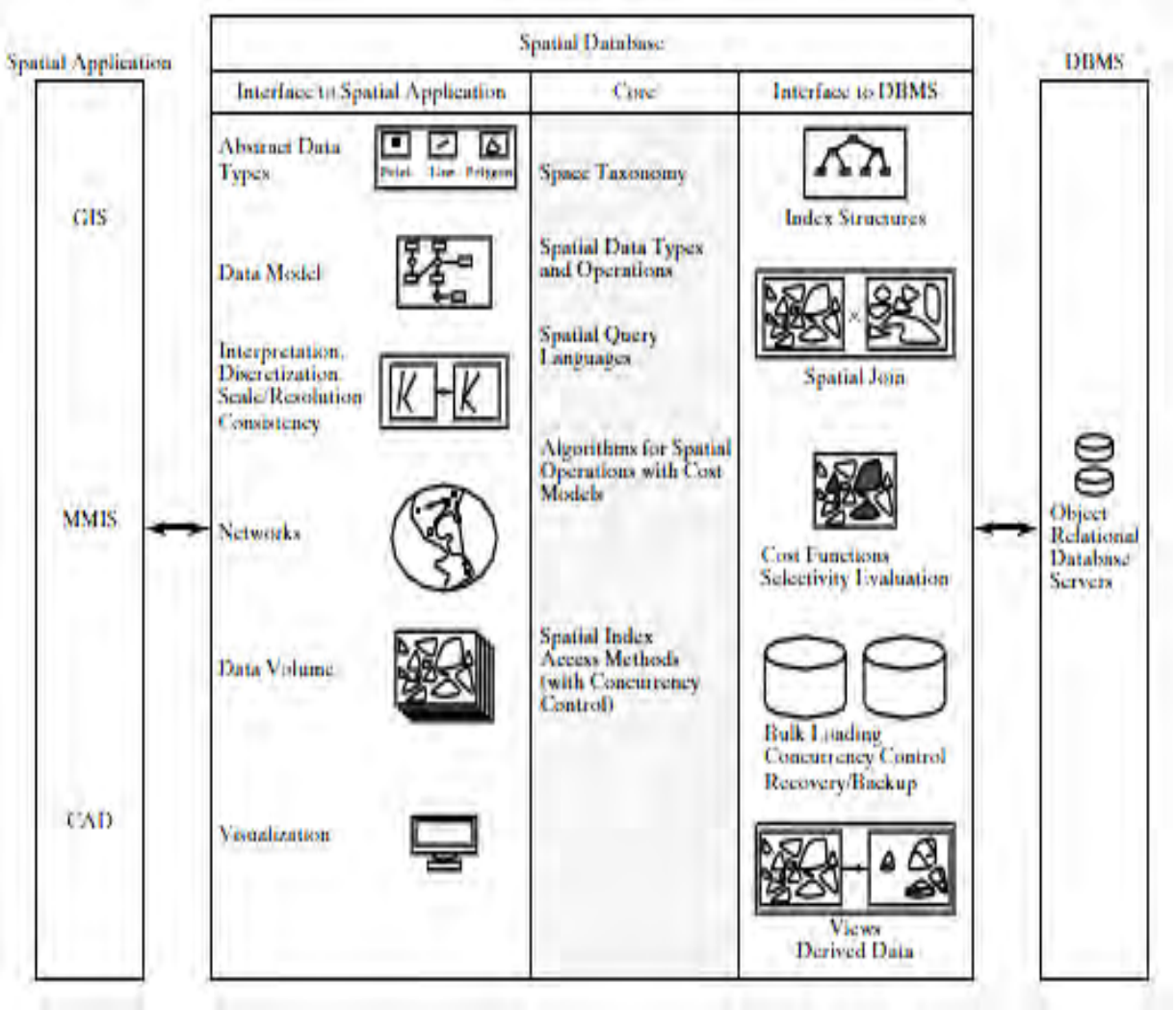
- συνεργάζεται με ένα υφιστάμενο ΣΔΒΔ
- υποστηρίζει ένα μοντέλο χωρικών δεδομένων και τους κατάλληλους τύπους αναπαράστασης χωρικής πληροφορίας , μέσω abstract data types (ADT)
- προσφέρει κατάλληλες επεκτάσεις στη γλώσσα ερωτήσεων (π.χ.SQL)

Σε φυσικό επίπεδο

- προσφέρει υποστήριξη για εξειδικευμένα ευρετήρια
- παρέχει αλγόριθμους επεξεργασίας χωρικών ερωτημάτων
- παρέχει προσαρμοσμένους κανόνες βελτιστοποίησης χωρικών ερωτημάτων

Όλες αυτές οι δυνατότητες ενός ΣΔΒΧΔ δομούνται βάση της αρχιτεκτονικής τριών επιπέδων. Όπως γίνεται αντιληπτό από την Εικόνα 15, στην τριών επιπέδων

αρχιτεκτονική το πρώτο επίπεδο αποτελούν οι χωρικές εφαρμογές, το δεύτερο επίπεδο οι χωρικές βάσεις δεδομένων και το τρίτο επίπεδο τα ΣΔΒΧΔ. Οι χωρικές βάσεις δεδομένων αποτελούν στην ουσία τον δίαυλο επικοινωνίας των χωρικών εφαρμογών με τα ΣΔΒΧΔ.



Εικόνα 15 - Τριών επιπέδων αρχιτεκτονική ενός ΣΔΒΧΔ (S.Shekhar & S.Chawla, 2003)

Εικόνα 16- Παράδειγμα κατασκευής θεματικού χάρτη μέσω εφαρμογής ArcCatalog του ArcGis (Gis Mapping Software,Solutions, Services,Map Apps and Data)

2.3. Μοντελοποίηση χώρου

Ιδιαίτερα σημαντικός είναι ο τρόπος με τον οποίο πραγματοποιείται η μοντελοποίηση του χώρου. Εάν θελήσουμε να αναπαραστήσουμε το περιβάλλον με ακρίβεια, τότε θα χρειαζόταν μία απείρως μεγάλη και πρακτικά μη πραγματοποιήσιμη βάση δεδομένων. Για το λόγο αυτό η επιλογή των στοιχείων προς αποθήκευση και η οπτική απεικόνισή τους εν συνεχεία, προϋποθέτει τον σαφή προσδιορισμό των αντικειμένων και την οργάνωσή τους σε κατηγορίες. Οι δύο αρχικές κατηγορίες (S.Shekhar, S.Chawla, S.Ravada, A.Fetterer, X.Liu, & C.Lu, Jan-Feb 1999) αναπαράστασης στις οποίες διαχωρίζονται τα δεδομένα είναι :

1. Αναπαράσταση του χώρου
2. Αναπαράσταση των αντικειμένων που βρίσκονται στο χώρο αλλά και των σχέσεων που αναπτύσσονται μεταξύ τους. Τα αντικείμενα αυτά μπορεί να είναι οδικά δίκτυα και κόμβοι πάνω σ' αυτά, πόλεις, σιδηρόδρομοι, εκκλησίες, νοσοκομεία κ.ά.

Η μοντελοποίηση του χώρου και των χωρικών συσχετίσεων είναι μια διαδικασία που προϋποθέτει την πλήρη αντίληψη του χώρου, του προσανατολισμού, της κλίμακας, της θέσης και του μεγέθους των αντικειμένων καθώς και το πως τα στοιχεία μέσα σε μια συλλογή δεδομένων σχετίζονται μεταξύ τους στο χώρο. Ανάλογα με τα κριτήρια με τα οποία επιθυμούμε να προσεγγίσουμε την περιγραφή για την οργάνωση του χώρου, χρησιμοποιούμε μια από τις τέσσερις επικρατέστερες μοντελοποιήσεις χώρου που ακολουθούν (S.Shekhar, S.Chawla, S.Ravada, A.Fetterer, X.Liu, & C.Lu, Jan-Feb 1999).

Τοπολογική Μοντελοποίηση

Ο τοπολογικός χώρος συνιστά την πιο γενική έννοια μοντελοποίησης χώρου, στον οποίο η γεωγραφική πραγματικότητα απεικονίζεται αρκετά απλοποιημένη. Όντας τόσο γενική η συγκεκριμένη μοντελοποίηση επιτρέπει την διατήρηση μόνο των απολύτως απαραίτητων στοιχείων του χάρτη αλλά και των σχετικών θέσεων μεταξύ των σημείων. Η κλίμακα και ο προσανατολισμός των στοιχείων σε ορισμένες περιπτώσεις παραμορφώνονται εξαιτίας του μεγάλου βαθμού γενίκευσης των δεδομένων που πραγματοποιείται (S.Shekhar, S.Chawla, S.Ravada, A.Fetterer, X.Liu, & C.Lu, Jan-Feb 1999).

Ευκλείδεια Μοντελοποίηση

Η αναπαράσταση του χώρου μέσω της Ευκλείδειας μοντελοποίησης χώρου προκύπτει από την αξιοποίηση των γεωμετρικών ιδιοτήτων των αντικειμένων. Ο ευκλείδειος χάρτης δημιουργείται μέσω της χρήσης του συστήματος συντεταγμένων με σκοπό την μετατροπή των χωρικών ιδιοτήτων των δεδομένων και των σχέσεων που αναπτύσσονται μεταξύ τους, σε πραγματικούς αριθμούς (S.Shekhar, S.Chawla, S.Ravada, A.Fetterer, X.Liu, & C.Lu, Jan-Feb 1999).

Δικτυακή Μοντελοποίηση

Στην δικτυακή αναπαράσταση του χώρου, η οποία συνιστά υποκατηγορία της τοπολογικής μοντελοποίησης χώρου, οι ιδιότητες συνεκτικότητας μεταξύ των κόμβων ικανοποιούν τις ιδιότητες των γράφων όπως συνεκτικότητα, ισομορφισμός,

συντομότερο μονοπάτι και επιπεδικότητα (S.Shekhar, S.Chawla, S.Ravada, A.Fetterer, X.Liu, & C.Lu, Jan-Feb 1999).

Set-based space Μοντελοποίηση

Η συγκεκριμένη μοντελοποίηση χρησιμοποιείται κυρίως στις σχεσιακές και στις σχεσιο - αντικειμενοστραφής βάσεις δεδομένων. Το μοντέλο αυτό μπορεί να χρησιμοποιηθεί για να εκφράσει ποιοτικά την απόσταση μεταξύ συμβολικών συντεταγμένων, μοντελοποιώντας περιοχές "γειτονιάς", ενώ η ποσοτική μοντελοποίηση της έννοιας της αποστάσεως είναι αδύνατη (Παπαταξιάρχης, 2006). Μία τέτοια μοντελοποίηση βασισμένη στη θεωρία συνόλων, φαίνεται πως μπορεί να απαντήσει ικανοποιητικά τις επερωτήσεις διαστήματος, όταν ο χώρος ορίζεται σαν ένα σύνολο συμβολικών συντεταγμένων και όλοι οι υποχώροι σαν υποσύνολα του συνόλου αυτού.

2.4. Μοντελοποίηση χωρικών δεδομένων

Τα χωρικά δεδομένα αποτελούν την ψηφιακή αναπαράσταση των γεωγραφικών πληροφοριών στον υπολογιστή. Η επιλογή των κατάλληλων μεταδεδομένων κρίνεται απαραίτητη, προκειμένου τα δεδομένα αυτά να καθίστανται διαχειρίσιμα και να μεταφέρουν μόνο τις χρήσιμες πληροφορίες για τον χρήστη. Η προσέγγιση αυτή της αποθήκευσης των απαραίτητων πληροφοριών μέσω της απλούστευσης, ορίζει την έννοια του μοντέλου δεδομένων. Τα κύρια μοντέλα για την αναπαράσταση των συγκεκριμένων πληροφοριών είναι δύο, το μοντέλο raster και το μοντέλο vector (Marathon Data Systems, 2011).

Το μοντέλο raster (ψηφιδωτή αναπαράσταση)

Το μοντέλο raster αναπαριστά τις γεωγραφικές οντότητες ως πλέγμα κελιών, τα οποία αποθηκεύουν αριθμητική πληροφορία και αναπαριστούν εικόνες ή θεματικά δεδομένα. Η αναπαράσταση που παρέχει το μοντέλο raster ενδείκνυται για συνεχή

φαινόμενα (υψόμετρα, βροχοπτώσεις, θερμοκρασία), αεροφωτογραφίες, δορυφορικές λήψεις, βλάστηση και τύπους εδάφους (Marathon Data Systems, 2011).

Το μοντέλο vector (διανυσματική αναπαράσταση)

Το μοντέλο vector αναπαριστά τις γεωγραφικές οντότητες ως σημεία (ζεύγος x,y συντεταγμένων), γραμμές (ακολουθία από ζεύγη x,y συντεταγμένων) ή πολύγωνα (μία γραμμή με ίδια αρχή και τέλος), στα οποία οι συντεταγμένες προσδιορίζουν το σχήμα και την θέση της γεωγραφικής οντότητας. Το συγκεκριμένο μοντέλο ενδείκνυται για διακριτά δεδομένα, ανθρωπογενή στοιχεία (πολιτικά όρια, δρόμους, κτίρια) και φυσικά στοιχεία (ποτάμια, λίμνες, δάση) (Marathon Data Systems, 2011).

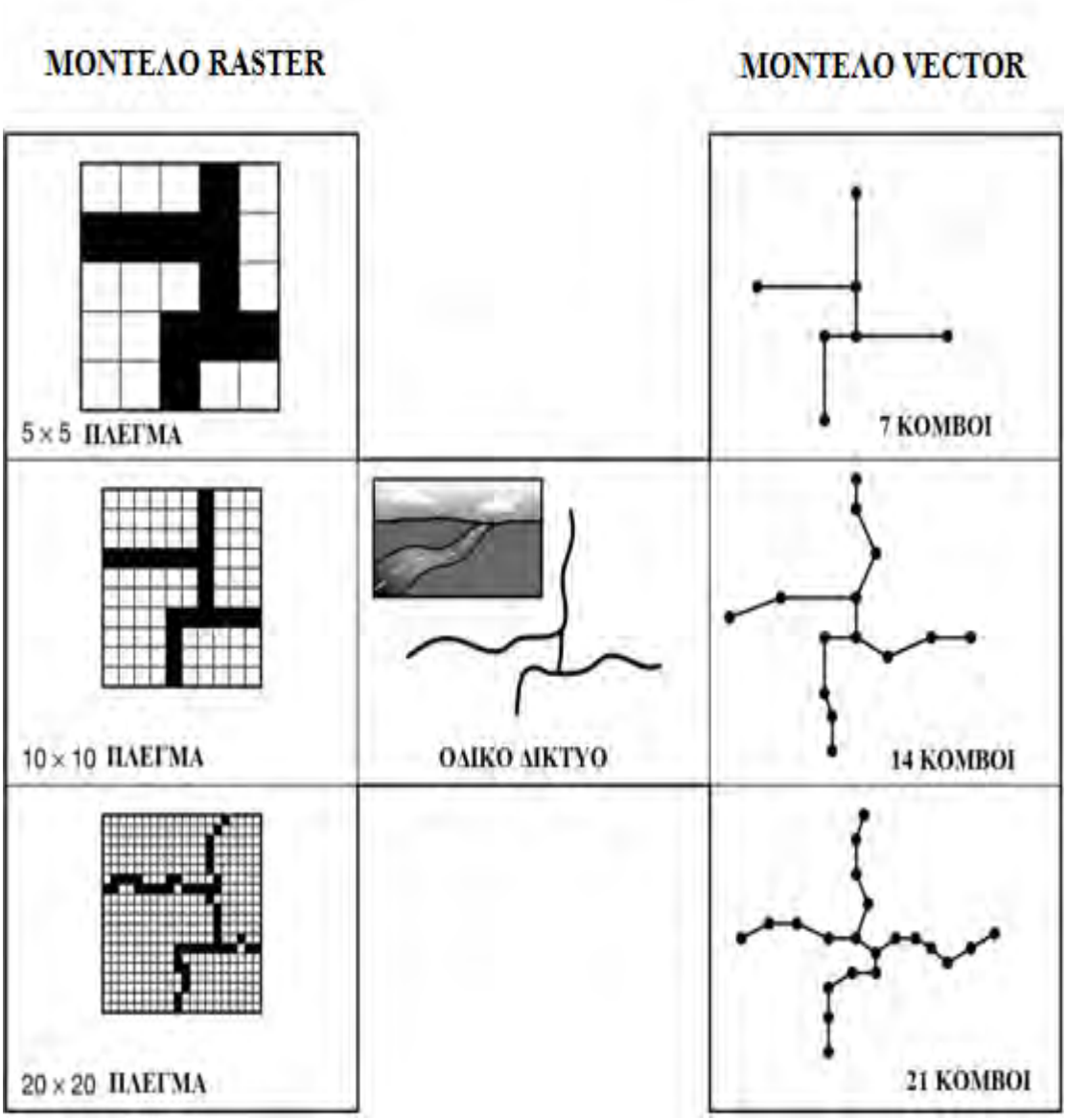
2.4.1. Σύγκριση μοντέλων Raster-Vector

Το κάθε μοντέλο είναι κατάλληλο για διαφορετικές εφαρμογές γι' αυτό και δεν μπορούμε να πούμε ότι το ένα υπερέχει του άλλου. Τα μοντέλα raster και vector μπορούν να συνυπάρχουν και επιπλέον τα vector δεδομένα μπορούν να μετατραπούν σε raster και το αντίστροφο, χρησιμοποιώντας τα κατάλληλα εργαλεία.

VECTOR	RASTER
Αναπαράσταση με σημείο, γραμμή και πολύγωνο.	Σημεία, γραμμές και πολύγωνα όλα σε μορφή Pixels.
Σχετικά μικρό μέγεθος αρχείων (μικρός όγκος δεδομένων).	Μεγάλο μέγεθος αρχείων.
Άριστη αναπαράσταση δικτύων.	Μη βέλτιστος τρόπος για αναπαράσταση δικτύων.
Μπορεί να συνοδεύονται από μεγάλο όγκο περιγραφικής πληροφορίας.	Αντιστοίχιση μιας τιμής σε κάθε κελί.
Τα χωρικά στοιχεία είναι πιο λεπτομερή και ακριβή.	Γενίκευση των χωρικών στοιχείων (π.χ. όρια)- ως εκ τούτου η ακρίβεια μπορεί να μειωθεί.
Η δημιουργία και η ενημέρωση δεδομένων είναι περισσότερο χρονοβόρες.	Οι προσομοιώσεις και οι μοντελοποιήσεις είναι πιο εύκολες (χωρική ανάλυση , μοντελοποίηση επιφανειών κλπ).
Μεγαλύτερη ευκολία και ευελιξία σε	Ευκολότερη συντήρηση.

τοπολογικές αναλύσεις (όπως αναλύσεις δικτύων κλπ).	
Μη ενδεικτικός τρόπος αναπαράστασης συνεχών δεδομένων, όπως χρήσεις γης, υψόμετρο κλπ.	Εξαιρετική αναπαράσταση συνεχών δεδομένων .
Η απόδοση προβολών και μετασχηματισμών παίρνουν λιγότερο χρόνο και καταναλώνουν λιγότερη μνήμη του υπολογιστικού συστήματος.	Οι μετασχηματισμοί/μετατροπές των συστημάτων συντεταγμένων καταναλώνουν περισσότερο χρόνο και μνήμη.
Η τοπολογία κάνει την δομή των δεδομένων περίπλοκη.	Τα pixels κάνουν την δομή των δεδομένων απλούστερη.

Εικόνα 17 - Σύγκριση μοντέλων Raster –Vector (Marathon Data Systems, 2011)



Εικόνα 18- Συγκριτική παρουσίαση μοντέλων Raster και Vector (T.Evans)

2.5. Γλώσσα χωρικών ερωτημάτων και διεξαγωγή αυτών

Η εύρεση χωρικών στοιχείων σε μια βάση χωρικών δεδομένων, βάσει χωρικών χαρακτηριστικών συνιστά μία από τις βασικότερες λειτουργίες που παρέχονται από την γλώσσα SQL. Η SQL (Structured Query Language) αποτελεί μια γλώσσα του ANSI για να μπορούμε να έχουμε πρόσβαση σε βάσεις δεδομένων. Συνιστά μία δηλωτική γλώσσα ερωτημάτων η οποία αποτελεί γλώσσα πρότυπο για τα εμπορικά σχεσιακά ΣΔΒΔ και περιλαμβάνει τις ακόλουθες δυνατότητες (Κέντρο ΠΛΗΝΕΤ Ν.Φλώρινας, 2014) :

- Εκτέλεση ερωτημάτων (queries) σχετικά με μια βάση δεδομένων.
- Ανάκτηση δεδομένων από μια βάση δεδομένων.
- Εισαγωγή νέων εγγραφών σε μια βάση δεδομένων.
- Διαγραφή εγγραφών από μια βάση δεδομένων.
- Ενημέρωση εγγραφών σε μια βάση δεδομένων.

Η SQL είναι πολύ εύκολη στην εκμάθηση της και παρέχει στο χρήστη την δυνατότητα να καθορίσει την πληροφορία την οποία θέλει να επεξεργαστεί μέσα από single-scan ή multi-scan ερωτήματα. Τέλος, συνεργάζεται με προγράμματα βάσεων δεδομένων όπως είναι τα εξής : Access, Informix, Microsoft SQL Server, Oracle, Sybase και πολλά άλλα (Κέντρο ΠΛΗΝΕΤ Ν.Φλώρινας, 2014).

3ο ΚΕΦΑΛΑΙΟ
ΥΛΟΠΟΙΗΣΗ ΧΩΡΙΚΗΣ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ

Η δημιουργία και η εξέλιξη των συστημάτων πλοήγησης οχημάτων τις τελευταίες δεκαετίες αλλά και η ολοένα αυξανόμενη ενσωμάτωσή τους στα οχήματα, τα καθιέρωσε ως αναπόσπαστα και άκρως χρήσιμα εργαλεία για τον χρήστη. Η εύρεση του προορισμού δεδομένης συγκεκριμένης διεύθυνσης, ο υπολογισμός της συντομότερης διαδρομής από και προς ενός σημείου και η βοήθεια προς τον οδηγό ώστε να ακολουθήσει με ακρίβεια το δοθέν μονοπάτι είναι κάποιες από τις υπηρεσίες που παρέχονται από τα συστήματα πλοήγησης οχημάτων.

Τα χωρικά δίκτυα τα οποία συνιστούν την δομή των προαναφερόμενων συστημάτων πλοήγησης, αποτελούνται από πεπερασμένο σύνολο σημείων - κόμβων, ευθύγραμμα τμήματα - ακμές που συνδέουν τους κόμβους αλλά και επιπλέον πληροφορίες οι οποίες προσδιορίζουν την γεωμετρία και τα γνωρίσματα των συγκεκριμένων ακμών και κόμβων. Η αποθήκευση όλων αυτών των δεδομένων απαιτεί τη δημιουργία μιας χωρικής βάσης δεδομένων και αποτελεί το θέμα ανάπτυξης του τρίτου κεφαλαίου της πτυχιακής εργασίας.

3.1. XAMPP - MySql

Το XAMPP είναι ένα πακέτο προγραμμάτων ελεύθερου λογισμικού, ανοικτού κώδικα το οποίο τρέχει σε διαφορετικά λειτουργικά συστήματα (Microsoft Windows, Linux, Solaris, Mac Os X) ή πλατφόρμες υλικού. Το XAMPP είναι ακρωνύμιο και τα αρχικά του έχουν την εξής σημασιολογία :

- X : λογισμικό ανεξάρτητο πλατφόρμας
- A : http Apache Εξυπηρετητής Ιστοσελίδων
- M : MySql
- P : PHP
- P : Perl

Το πακέτο αυτό περιέχει τον εξυπηρετητή ιστοσελίδων http Apache, την βάση δεδομένων MySql και ένα διερμηνέα για κώδικα γραμμένο σε γλώσσες προγραμματισμού PHP και Perl. Ο εξυπηρετητής ιστοσελίδων ο οποίος εμπεριέχεται στο XAMPP, του παρέχει την δυνατότητα να εξυπηρετεί και δυναμικές ιστοσελίδες PHP και MySql (apache friends - xampp for windows, 2014).

Το XAMPP χρησιμοποιείται κυρίως για την φιλοξενία ιστοσελίδων αλλά και για την δημιουργία και διαχείριση βάσεων δεδομένων τύπου MySQL και SQLite. Η MySQL είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων ανοικτού κώδικα που μετρά περισσότερες από 11 εκατομμύρια εγκαταστάσεις. Το πρόγραμμα τρέχει έναν εξυπηρετητή (server) παρέχοντας πρόσβαση πολλών χρηστών σε ένα σύνολο βάσεων δεδομένων (MySql , 2014). Η MySql χρησιμοποιεί την Structured Query Language (SQL), την πιο γνωστή γλώσσα για την προσθήκη, την πρόσβαση και την επεξεργασία δεδομένων σε μία Βάση Δεδομένων. Όταν το XAMPP εγκατασταθεί στον τοπικό υπολογιστή διαχειρίζεται τον localhost ως ένα απομακρυσμένο κόμβο, ο οποίος συνδέεται με το πρωτόκολλο μεταφοράς αρχείων FTP. Η σύνδεση στον localhost μέσω του FTP μπορεί να γίνει με το όνομα χρήστη «newuser» και τον κωδικό «wampp». Για την βάση δεδομένων MySQL υπάρχει ο χρήστης «root» χωρίς κωδικό πρόσβασης (apache friends - xampp for windows, 2014).

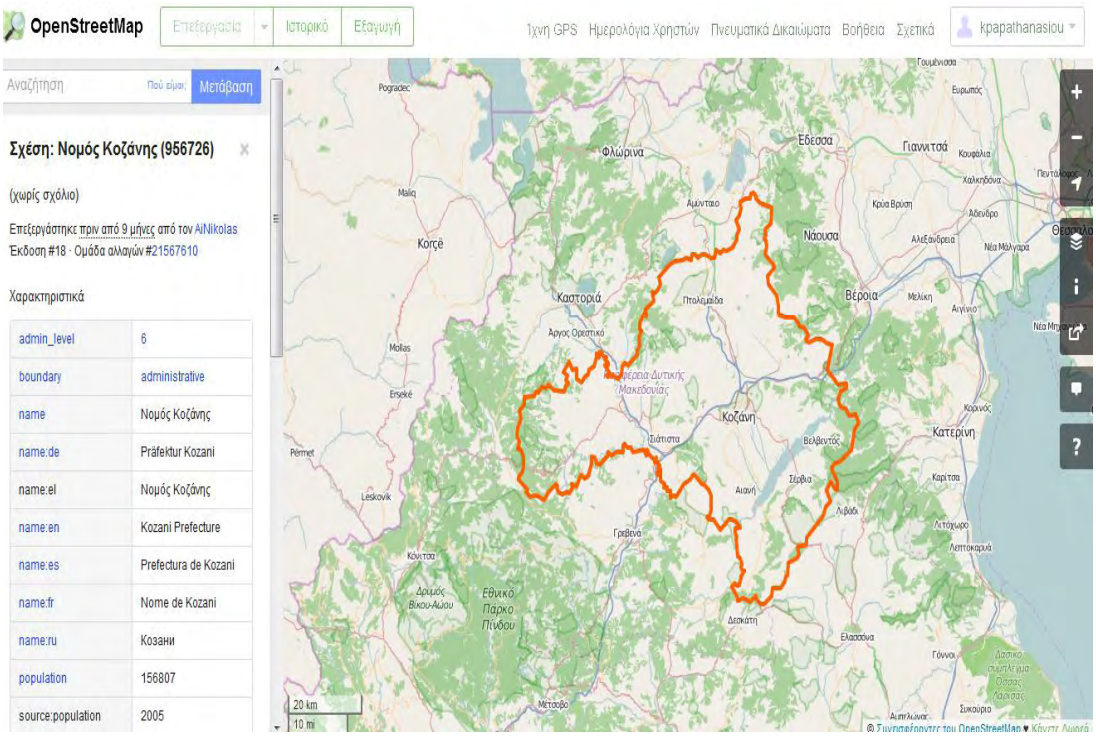
Εικόνα 19- XAMPP, MySql, PHP, Perl (apache friends - xampp for windows, 2014) (MySql , 2014) (PHP: Hypertext Preprocessor , 2014) (The Perl Programming Language , 2014)

Η MySql είναι ανοικτού κώδικα που σημαίνει ότι ο οποιοσδήποτε μπορεί να κατεβάσει την MySQL και να την διαμορφώσει σύμφωνα με τις ανάγκες του, ακολουθώντας πάντοτε την γενική άδεια που υπάρχει. Είναι γνωστή κυρίως για την ταχύτητα, την αξιοπιστία και την ευελιξία που παρέχει και προτιμάται κυρίως για την διαχείριση περιεχομένου και όχι για την εκτέλεση συναλλαγών. Τέλος, η MySQL

αυτή τη στιγμή μπορεί να λειτουργήσει σε περιβάλλον Linux, Unix, και Windows (isites-Τί είναι μια MySql βάση δεδομένων , 2014).

3.2. Λήψη δεδομένων - Open Street Map

Η λήψη των δεδομένων για την υλοποίηση της βάσης και εν συνεχεία της εφαρμογής εύρεσης του συντομότερου μονοπατιού πραγματοποιήθηκε μέσω της ανοικτής βάσης δεδομένων του Open Street Map. Η βάση αυτή αναπτύσσεται από μια κοινότητα χαρτογράφων που συνεισφέρουν και διατηρούν δεδομένα σχετικά με δρόμους, μονοπάτια, καφετέριες, σιδηροδρομικούς σταθμούς και πολλά περισσότερα, σε όλον τον κόσμο. Οι συνεισφέροντες χρησιμοποιούν αεροφωτογραφίες, συσκευές GPS και τοπικούς χάρτες χαμηλής τεχνολογίας για να σιγουρευτούν πως το Open Street Map είναι ακριβές και ενημερωμένο. Αποτελεί μια βάση δεδομένων την οποία ο οποιοσδήποτε είναι ελεύθερος να την χρησιμοποιήσει για οποιονδήποτε σκοπό, εφόσον μνημονεύσει το Open Street Map και τους συνεισφέροντες του (OpenStreetMap, 2014). Τα δεδομένα του Open Street Map είναι αδειοδοτημένα από την Open Data Commons Open Database License (ODbL).



Εικόνα 20 - Παράδειγμα απεικόνισης χάρτη Ν. Κοζάνης στο Open Street Map (OpenStreetMap, 2014)

3.2.1. File format του Open Street Map

Το Open Street Map αποτελείται από ανοικτά δεδομένα τα οποία παρέχονται στους χρήστες προς αντιγραφή, διανομή, μετάδοση και προσαρμογή σε μορφή αρχείων osm (OpenStreetMap, 2014). Η αρχιτεκτονική των αρχείων osm ακολουθεί την μορφή ενός xml σχήματος, το οποίο ουσιαστικά περιέχει μία λίστα χαρακτηριστικών για τα βασικά δεδομένα του χάρτη, των κόμβων και των σχέσεων που αναπτύσσονται μεταξύ τους. Αναλυτικότερα, κάθε αρχείο osm περιέχει την ακόλουθη περιγραφική πληροφορία (OpenStreetMap, 2014) :

- ένα XML πρόθεμα το οποίο περιέχει τον UTF-8 χαρακτήρα κωδικοποίησης για το αρχείο
- ένα στοιχείο αρχείου osm, το οποίο περιέχει την έκδοση API και ως εκ τούτου τα χαρακτηριστικά που χρησιμοποιούνται
- ένα πακέτο με πληροφορίες για τους κόμβους του χάρτη, το οποίο περιέχει ειδικά την θέση των κόμβων αυτών στο WGS84 σύστημα αναφοράς χωρικών δεδομένων :
    - ετικέτες του κάθε κόμβου
    - ένα πακέτο δρόμων το οποίο περιλαμβάνει :
        - αναφορές των κόμβων για κάθε δρόμο
        - ετικέτες του κάθε δρόμου
    - ένα πακέτο σχέσεων που δημιουργούνται μεταξύ κόμβων και δρόμων το οποίο περιλαμβάνει :
        - αναφορές για κάθε μέλος και τις σχέσεις που δημιουργούνται γι' αυτό
        - ετικέτες για κάθε σχέση που δημιουργείται

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.3.3 (15972 thorn-
03.openstreetmap.org)" copyright="OpenStreetMap and
contributors"
attribution="http://www.openstreetmap.org/copyright"
license="http://opendatacommons.org/licenses/odbl/1-0/">
 <bounds minlat="38.8924900" minlon="12.2487570"
maxlat="22.4310800" maxlat="38.8974500" maxlon="22.4417500"/>
```

```
 <node id="997732729" visible="true" version="1"
changeset="6422323" timestamp="2010-11-21T13:05:08Z"
user="greecemapper" uid="23785" lat="38.8957104"
lon="22.4292224"/>
 <node id="997732755" visible="true" version="1"
changeset="6422323" timestamp="2010-11-21T13:05:08Z"
user="greecemapper" uid="23785" lat="38.8965323"
lon="22.4266741"/>
 <node id="989463911" visible="true" version="1"
changeset="6373953" timestamp="2010-11-14T23:32:16Z"
user="greecemapper" uid="23785" lat="38.8931171"
lon="22.4263322">
  <tag k="railway" v="level_crossing"/>
 </node>
...
<way id="85354242" visible="true" version="7"
changeset="24525367" timestamp="2014-08-03T22:08:22Z"
user="kpapathanasiou…" uid="2230609">
  <nd ref="990015763"/>
  <nd ref="990015379"/>
  ...
  <nd ref="990015402"/>
  <nd ref="990015372"/>
  <tag k="highway" v="tertiary_link"/>
  <tag k="name" v="kpapathanasiou…"/>
  <tag k="oneway" v="yes"/>
 </way>
...
<relation id="400178" visible="true" version="84"
changeset="25885303" timestamp="2014-10-05T21:08:19Z"
user="kpapathanasiou" uid="2230609">
  <member type="way" ref="102443513" role=""/>
  <member type="way" ref="102448489" role=""/>
  ...
  <member type="way" ref="102448493" role=""/>
  <member type="way" ref="102448530" role=""/>
  ...
```

```
  <member type="way" ref="102448457" role=""/>
  <member type="way" ref="53663191" role=""/>
  ...
</relation>
 ...
</osm>
```

Εικόνα 21- Παράδειγμα osm αρχείου (OpenStreetMap, 2014)

3.3. Το μοντέλο Arc-Node (Κόμβου-Ακμής)

Το μοντέλο που χρησιμοποιείται κυρίως για την αναπαράσταση ενός οδικού δικτύου είναι το μοντέλο arc - node (κόμβου – ακμής) και προτάθηκε αρχικά από τον Sheffi. Όπως προαναφέρθηκε, σε ένα οδικό δίκτυο οι κόμβοι αντιστοιχούν γενικά σε διασταυρώσεις, ενώ οι ακμές αντιστοιχούν σε τμήματα δρόμων που βρίσκονται ανάμεσα στις διασταυρώσεις. Δύο κατευθυνόμενες ακμές αντίθετων διευθύνσεων αναπαριστούν έναν δρόμο διπλής κατεύθυνσης ή δύο παράλληλα τμήματα αυτοκινητόδρομου με αντίθετες κατευθύνσεις ροών. Μια γενικευμένη συνάρτηση κόστους αναπαριστά συνήθως το κόστος ροής ανά μονάδα για τη διάσχιση της ακμής (Πλέσσας, 2008).

3.4. Υλοποίηση βάσης

Στο πρώτο στάδιο υλοποίησης της παρούσας εφαρμογής δημιουργήθηκε μια νέα χωρική βάση δεδομένων με όνομα ‘map’ και στη συνέχεια ακολούθησε η δημιουργία πινάκων στη βάση για την εισαγωγή των χωρικών δεδομένων με τη βοήθεια κώδικα SQL. Οι πίνακες της βάσης δημιουργήθηκαν με την εντολή ‘CREATE TABLE’ ενώ η εισαγωγή των απαραίτητων πληροφοριών που αφορούν το χωρικό δίκτυο στα πεδία των συγκεκριμένων πινάκων πραγματοποιήθηκε μέσω των εντολών ‘INSERT’ και ‘UPDATE’. Τέλος, η επιλογή των κατάλληλων πληροφοριών που ήταν απαραίτητες σε κάθε στάδιο της εφαρμογής επιτεύχθηκε με την εντολή ‘SELECT’.

- Το ερώτημα βάσει του οποίου δημιουργήθηκε η βάση δεδομένων είναι το ακόλουθο :

```
"CREATE DATABASE IF NOT EXISTS " + dbName + " DEFAULT
CHARACTER SET utf8 COLLATE utf8_unicode_ci";
```

- Το ερώτημα βάσει του οποίου δημιουργήθηκε ο πίνακας με όνομα ‘nodes’, ο οποίος περιέχει τα πεδία ‘node_id’, ‘lnode_id’, ‘lat’, ‘lot’ και χαρακτηρίζει το πρώτο πεδίο ως πρωτεύον κλειδί, είναι το ακόλουθο :

```
"CREATE TABLE IF NOT EXISTS nodes " +
        "(node_id INT(15) NOT NULL AUTO_INCREMENT, " +
        "lnode_id BIGINT(15) NOT NULL DEFAULT 0, " +
        "lat FLOAT DEFAULT NULL, " +
        "lon FLOAT DEFAULT NULL, " +
        "PRIMARY KEY (node_id))";
```

- Το ερώτημα βάσει του οποίου δημιουργήθηκε ο πίνακας με όνομα ‘ways’, ο οποίος περιέχει τα πεδία ‘way_id’, ‘lway_id’, ‘access’, ‘cycleway’, ‘lanes’, ‘name’, ‘oneway’, ‘maxspeed’ και χαρακτηρίζει το πρώτο πεδίο ως πρωτεύον κλειδί, είναι το ακόλουθο :

```
 "CREATE TABLE IF NOT EXISTS ways " +
        "(way_id INT(15) NOT NULL AUTO_INCREMENT, " +
        "lway_id BIGINT(15) NOT NULL DEFAULT 0, " +
        "access VARCHAR(11) DEFAULT NULL, " +
        "cycleway VARCHAR(11) DEFAULT NULL, " +
        "lanes INT(11) DEFAULT NULL, " +
        "name VARCHAR(50) NOT NULL DEFAULT '-', " +
        "oneway VARCHAR(11) NOT NULL DEFAULT 'no', " +
        "maxspeed INT(11) DEFAULT NULL, " +
        "PRIMARY KEY (way_id))";
```

- Το ερώτημα βάσει του οποίου δημιουργήθηκε ο πίνακας με όνομα ‘ways_nodes’, ο οποίος περιέχει τα πεδία ‘id’, ‘way’, ‘node’ και χαρακτηρίζει το πρώτο πεδίο ως πρωτεύον κλειδί, είναι το ακόλουθο :

```
  "CREATE TABLE IF NOT EXISTS way_nodes " +
      "(id INT(15) NOT NULL AUTO_INCREMENT, " +
```

```
        "way INT(15) DEFAULT NULL, " +
        "node INT(15) DEFAULT NULL, " +
        "PRIMARY KEY (id))";
```

- Το ερώτημα βάσει του οποίου δημιουργήθηκε ο πίνακας για τον κόμβο που ψάχνουμε με όνομα ‘name_node’, ο οποίος περιέχει τα πεδία ‘name’ και ‘node’ είναι το ακόλουθο :

```
 "CREATE TABLE IF NOT EXISTS name_node " +
        "(name VARCHAR(50) NOT NULL DEFAULT '-', " +
        "node INT(15) DEFAULT NULL)";
```

- Το ερώτημα βάσει του οποίου διαγράφεται η βάση είναι το ακόλουθο :

```
"DROP DATABASE IF EXISTS " + dbName;
```

- Το ερώτημα βάσει του οποίου επιλέχθηκαν από τον πίνακα ‘nodes’ τα δεδομένα των πεδίων ‘node_id’ και ‘ lnode_id’ προκειμένου να δημιουργηθούν οι κορυφές του χάρτη είναι το ακόλουθο :

```
"SELECT node_id, lnode_id FROM nodes WHERE 1";
```

- Το ερώτημα βάσει του οποίου επιλέχθηκαν από τους πίνακες ‘way_nodes’ και ‘nodes’τα δεδομένα των πεδίων ‘node’,‘lat’,‘lon’ και ‘way’ προκειμένου να δημιουργηθούν οι ακμές του χάρτη είναι το ακόλουθο :

```
"SELECT way, node, lat, lon FROM way_nodes, nodes WHERE
way_nodes.node=nodes.node_id";
```

- Το ερώτημα βάσει του οποίου επιλέχθηκαν από τον πίνακα ‘name_node’ τα δεδομένα του πεδίου ‘node’ προκειμένου να Το χρησιμοποιούμε για να πάρουμε τον κόμβο από το όνομα του δρόμου που μας δίνει ο χρήστης είναι το ακόλουθο :

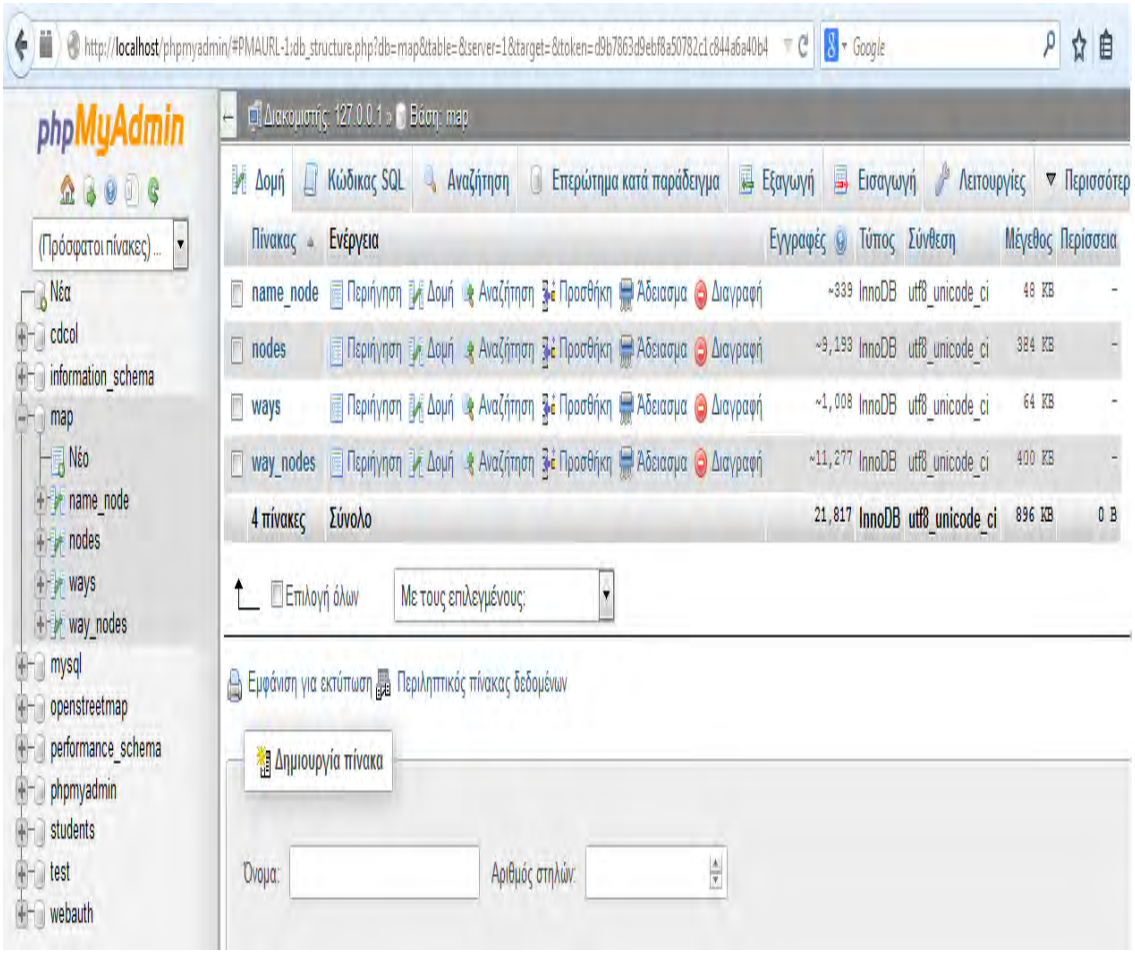
```
"SELECT node FROM name_node WHERE name='" + name +"'";
```

- Το ερώτημα βάσει του οποίου πραγματοποιήθηκε εισαγωγή δεδομένων στον πίνακα ‘nodes’ είναι το ακόλουθο :

```
"INSERT INTO nodes(lnode_id) VALUES (" + word + ");
```

- Το ερώτημα βάσει του οποίου πραγματοποιήθηκε ενημέρωση δεδομένων στον πίνακα ‘nodes’ για τα μήκη και πλάτη των κόμβων είναι το ακόλουθο :

```
"UPDATE nodes SET lat=" + word + " WHERE lnode_id=" + id);
```



Εικόνα 22- Χωρική βάση δεδομένων 'map'

4ο ΚΕΦΑΛΑΙΟ
ΑΛΓΟΡΙΘΜΟΙ ΕΥΡΕΣΗΣ ΣΥΝΤΟΜΟΤΕΡΟΥ ΜΟΝΟΠΑΤΙΟΥ

Το πρόβλημα της εύρεσης συντομότερου μονοπατιού μεταξύ μιας πηγής και ενός προορισμού σ' ένα οδικό δίκτυο και η παρουσίαση ορισμένων αλγοριθμικών τεχνικών που επιλύουν το συγκεκριμένο πρόβλημα, αποτελούν το θέμα ανάπτυξης του τέταρτου κεφαλαίου. Η μελέτη των προαναφερόμενων αλγορίθμων εστιάζεται κυρίως στα βήματα υλοποίησης τους, στις τεχνικές τους ιδιότητες και στην απόδοσή τους. Οι τρεις αλγόριθμοι που θα αποτελέσουν το αντικείμενο μελέτης είναι οι : Dijkstra, Bellman-Ford και Floyd-Warshall.

4.1. Κατηγορίες προβλημάτων συντομότερων μονοπατιών

Η συντομότερη διαδρομή μεταξύ δύο σημείων, της αφετηρίας και του προορισμού, αποτελεί την διαδρομή με το μικρότερο δυνατό συνολικό κόστος, δηλαδή το άθροισμα των τιμών των ακμών (βάρη) του γραφήματος. Επιπλέον, η συντομότερη διαδρομή δεν είναι απαραίτητο να περιέχει όλους τους κόμβους του δικτύου. Οι τέσσερις βασικές κατηγορίες των προβλημάτων συντομότερης διαδρομής είναι οι εξής :

- **Μονής-πηγής :** βρες το συντομότερο μονοπάτι από δοσμένη πηγή προς κάθε κορυφή
- **Μονού-προορισμού :** βρες το συντομότερο μονοπάτι από κάθε κορυφή προς δοσμένο προορισμό
- **Μονού-ζεύγους :** δοσμένων 2 κορυφών, βρες το συντομότερο μονοπάτι μεταξύ τους
- **Κάθε-ζεύγους:** βρες τα συντομότερα μονοπάτια για κάθε ζεύγος κορυφών

4.2. Αλγόριθμος Dijkstra

Ο αλγόριθμος του Dijkstra[4] ανήκει στην κατηγορία των αλγορίθμων μονής πηγής (single source algorithm), βρίσκοντας τα συντομότερα μονοπάτια δοσμένης πηγής προς όλους τους προσεγγίσιμους κόμβους. Βασική προϋπόθεση εκτέλεσης του είναι η ύπαρξη μόνο θετικών βαρών στις ακμές του γράφου, καθώς σε αντίθεση περίπτωση το αποτέλεσμα της εκτέλεσής του είναι λανθασμένο (H.Thomas, E.Charles, L.Ronald, & Clifford, 2001). Ο συγκεκριμένος αλγόριθμος συμπεριλαμβάνεται στην κατηγορία των άπληστων αλγορίθμων και είναι παρόμοιος με τον αλγόριθμο Prim για τα Ελάχιστα Ζευγνύοντα Δένδρα.

Σύμφωνα με την βασική ιδέα του αλγορίθμου, κατά την εκτέλεσή του διατηρείται ένα σύνολο S για τις κορυφές για τις οποίες έχουν προσδιοριστεί τα ελάχιστα συσσωρευμένα κόστη μετάβασης από την δοσμένη αφετηρία προς την κάθε κορυφή. Σε κάθε βήμα της επαναληπτικής διαδικασίας ο αλγόριθμος επιλέγει κάθε φορά την εγγύτερη κορυφή u (μεταξύ των κορυφών εκτός S) η οποία ανήκει στο συντομότερο μονοπάτι που ξεκινά από την αφετηρία και την προσθέτει στο σύνολο S και επιπλέον χαλαρώνει όλες τις ακμές που εξέρχονται από την κορυφή u.

ΨΕΥΔΟΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ DIJKSTRA

```
    Dijkstra(G,s)
1.  for each u ∈ V(G)
2.      d(u) = ∞
3.      π(u) = NIL
4.  d(s) = 0
5.  S ← ∅ // Set of vertices with d(v) = δ(s, v)
6.  Q ← V(G) // Q: priority queue keyed by d(v)
7.  while not isEmpty(Q)
8.      u ← extractMin(Q)
9.      S ← S ∪ {u}
10.     for each v ∈ adjacent(u) do
11.         Relax(u, v, G)
```

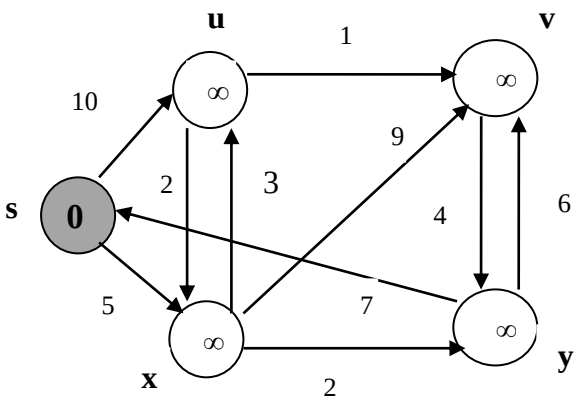
[4] Πήρε το όνομά του από τον Ολλανδό Έντσγκερ Ντάικστρα, ο οποίος τον επινόησε το1956.

Relax (u,v,w)

1. if d(v) > d(u)+ w(u,v) then
2. d(v) = d(u)+ w(u,v)
3. π(v) = u
4. decreaseKey(Q,v,d(v))

ΕΚΤΕΛΕΣΗ ΑΛΓΟΡΙΘΜΟΥ

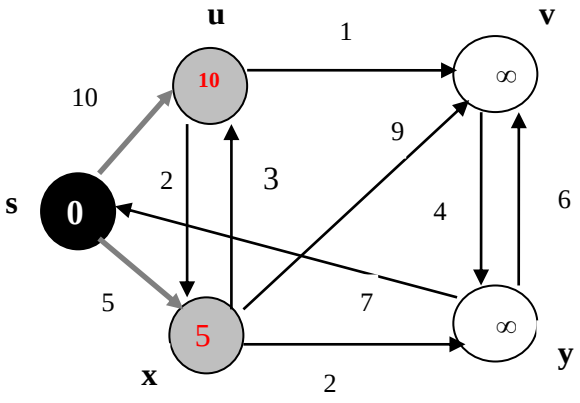
(H.Thomas, E.Charles, L.Ronald, & Clifford, 2001)



S = {}

Q= {s,u,v,x,y}

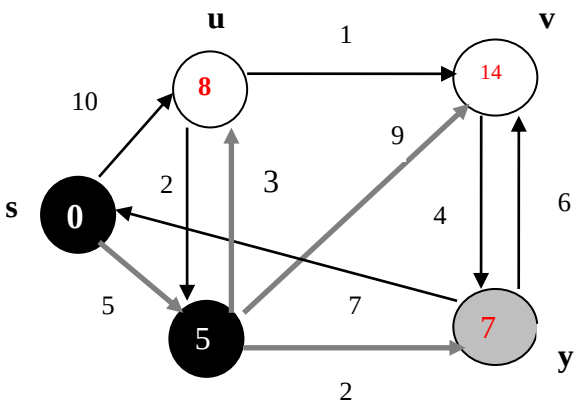
ΠΡΩΤΟ ΒΗΜΑ



S = {s}

Q= {x,u,v,y}

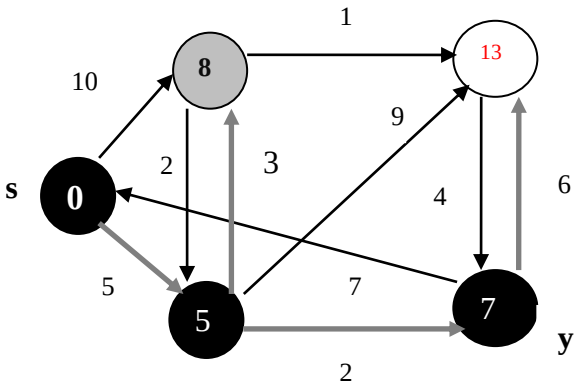
ΔΕΥΤΕΡΟ ΒΗΜΑ



S = {s,x}

Q= {y,u,v}

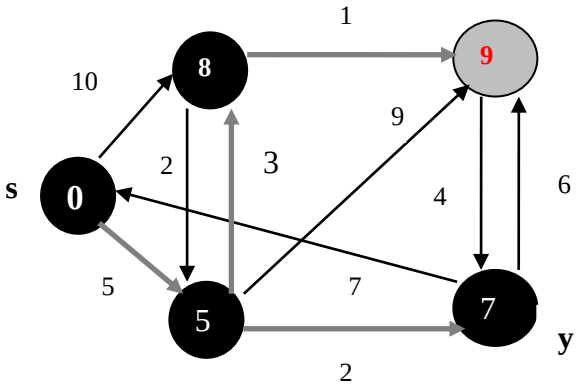
ΤΡΙΤΟ ΒΗΜΑ



S = {s,x,y}

Q= {u,v}

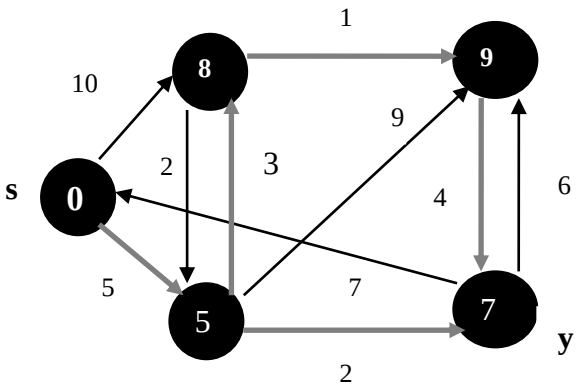
ΤΕΤΑΡΤΟ ΒΗΜΑ



S = {s,x,y,u}

Q= {v}

ΠΕΜΠΤΟ ΒΗΜΑ(ΤΕΛΕΥΤΑΙΟ)



S = {s,x,y,u,v}

Q= {}

Σε κάθε επανάληψη του αλγορίθμου καθορίζεται το συντομότερο μονοπάτι για μια κορυφή, την κορυφή οδηγό. Η ετικέτα d(u) της κορυφής οδηγού u είναι ίση με το κόστος του συντομότερου μονοπατιού και επομένως η ετικέτα είναι μόνιμη. Η μόνιμη αυτή ετικέτα μπορεί να αποτελέσει μια επιπλέον συνθήκη τερματισμού καθώς όταν μία κορυφή έχει μόνιμη ετικέτα, το συντομότερο μονοπάτι προς αυτή έχει βρεθεί.

Είναι προφανές πως για ένα γράφο *G (V, E)* ο αριθμός των επαναλήψεων του αλγόριθμου Dijkstra είναι το πολύ V, ενώ κάθε επανάληψη απαιτεί χρόνο O(V) για την απλή υλοποίηση. Συνεπώς, η πολυπλοκότητα του Dijkstra είναι ίση με $O(V^2)$. Επιπλέον, γίνεται αντιληπτό ότι όσο το πλήθος των ακμών του γράφου αυξάνεται, η πολυπλοκότητα του αλγόριθμου είναι απαγορευτική. Για τις περιπτώσεις αυτές, έχουν προταθεί αλγόριθμοι εύρεσης βέλτιστων διαδρομών που αξιοποιούν τις αρχές της τεχνητής νοημοσύνης. Ο πιο γνωστός αλγόριθμος αυτής της κατηγορίας, είναι ο *A** (Παπαδοπούλου, 2011).

4.3. Αλγόριθμος Bellman - Ford

Ο αλγόριθμος Bellman – Ford πήρε το όνομά του από τους εμπνευστές του, τον ο Richard Bellman και τον Lester Ford[5], οι οποίοι τον δημοσίευσαν το 1958 και το 1956, αντίστοιχα. Ο συγκεκριμένος αλγόριθμος ανήκει και αυτός όπως ο Dijkstra στην κατηγορία των αλγορίθμων μονής πηγής (single source algorithm), υπερτερώντας όμως σε ευελιξία καθώς μπορεί να βρει συντομότερα μονοπάτια σε

[5] Edward F. Moore δημοσίευσε επίσης τον ίδιο αλγόριθμο το 1957. Για το λόγο αυτό κάποιες φορές καλείται ο αλγόριθμος Bellman-Ford-Moore.

γράφους που περιέχουν και αρνητικά βάρη στις ακμές τους. Στηριζόμενος στην αρχή της χαλάρωσης, εκτελεί v-1 βήματα χαλάρωσης, εξετάζοντας όλες τις ακμές (όπου v οι κορυφές του γράφου). Χαλάρωση μια ακμής (*u*,*v*) σημαίνει έλεγχος αν μπορεί να βελτιωθεί το συντομότερο μονοπάτι προς την *v* που έχει βρεθεί ως τώρα, με διέλευση από την *u* (H.Thomas, E.Charles, L.Ronald, & Clifford, 2001). Εξαιτίας της ύπαρξης των αρνητικών βαρών είναι δυνατόν μια ακμή να χαλαρώσει περισσότερες από μία φορές και στην περίπτωση όπου στο v - οστό βήμα πραγματοποιηθεί χαλάρωση τότε συμπεραίνουμε ότι υπάρχει αρνητικός κύκλος. Στην περίπτωση ύπαρξης αρνητικού κύκλου τότε ο αλγόριθμος Bellman - Ford θα επιστρέψει false.

Για κάθε κορυφή v, διατηρούμε την εκτίμηση d(v) του συντομότερου μονοπατιού από την αρχική κορυφή s. Οι κορυφές αρχικοποιούνται με ∞ και ο αλγόριθμος χρησιμοποιεί την μέθοδο της απόδοσης ετικέτας. Μειώνει δηλαδή, προοδευτικά την <<προσωρινή>> ετικέτα d[u] της κορυφής u, που αντιπροσωπεύει το βάρος του συντομότερου μονοπατιού από την αφετηρία s προς κάθε κορυφή u ε V, μέχρις ότου επιτευχθεί το βάρος του απόλυτα συντομότερου μονοπατιού δ(s,u). Σε κάθε φάση επιλέγεται το καλύτερο γειτονικό από τους κόμβους. Βέβαια, ο καλύτερος γειτονικός κόμβος που επιλέγεται μπορεί να αλλάξει και ο αλγόριθμος τρέχει για n φάσεις. Στην τελευταία φάση δεν γίνονται αλλαγές οπότε και ο αλγόριθμος τερματίζει (Ε.Σωτηρίου, 2010).

Το δένδρο που κατασκευάζει ο Bellman – Ford είναι δένδρο συντομότερων μονοπατιών μόνο όταν τελειώσει το τρέξιμο του αλγορίθμου και η ετικέτα που αντιστοιχεί σε μια κορυφή είναι η <<μόνιμη>> ετικέτα κατά τον τερματισμό του αλγορίθμου. Η πολυπλοκότητα του αλγορίθμου με V κόμβους και E ακμές είναι ίση με O (V, E) (Ε.Σωτηρίου, 2010).

ΨΕΥΔΟΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ BELLMAN - FORD

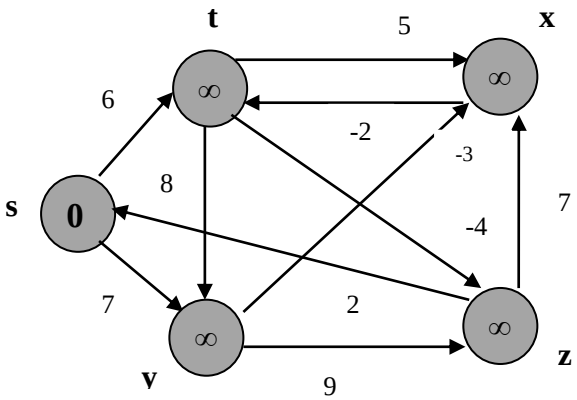
```
    Bellman - Ford(G,w,s)
1.  for each u ∈ V(G)
2.      d(u) = ∞
3.      π(u) = NIL
4.  d(s) = 0
5.  for i ← 1 to |V(G)|-1 do
6.      for each edge (u,v) ∈ E(G) do
```

```
7.          Relax (u,v,w)
8.  for each edge (u,v) ∈ E(G) do
9.          if d(v) > d(u) + w(u,v) then
10.                 return false
11. return true
```

```
    Relax (u,v,w)
1.  if d(v) > d(u)+ w(u,v) then
2.          d(v) = d(u)+ w(u,v)
3.          π(v) = u
```

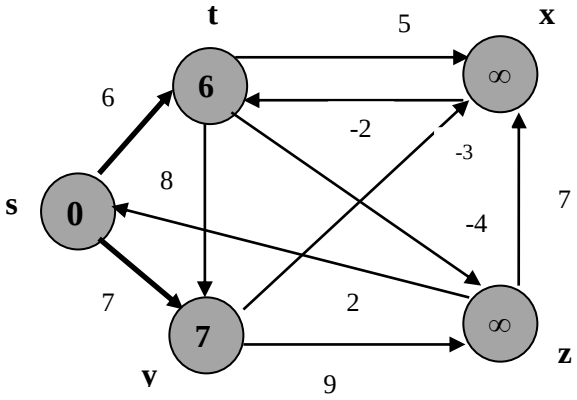
ΕΚΤΕΛΕΣΗ ΑΛΓΟΡΙΘΜΟΥ

(H.Thomas, E.Charles, L.Ronald, & Clifford, 2001)



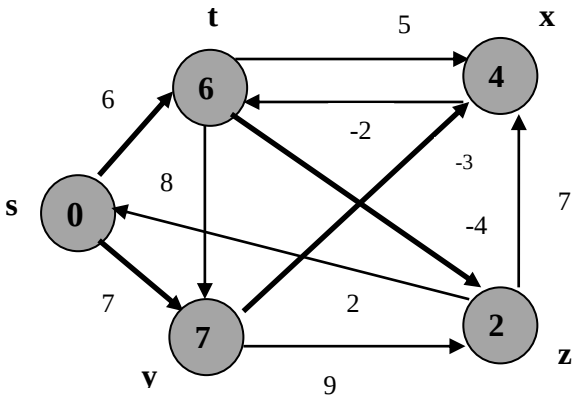
Σειρά χαλάρωσης (t,x),(t,y),(t,z),(x,t)
(y,x),(y,z),(z,x),(z,s),**(s,t),(s,y)**

ΠΡΩΤΟ ΒΗΜΑ



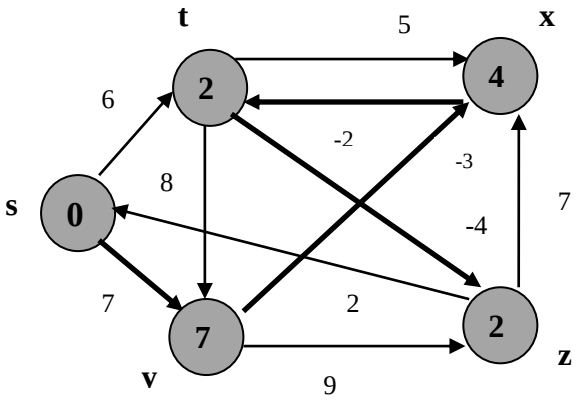
Σειρά χαλάρωσης (t,x),(t,y),(t,z),(x,t)
(y,x),(y,z),(z,x),(z,s),**(s,t),(s,y)**

ΔΕΥΤΕΡΟ ΒΗΜΑ



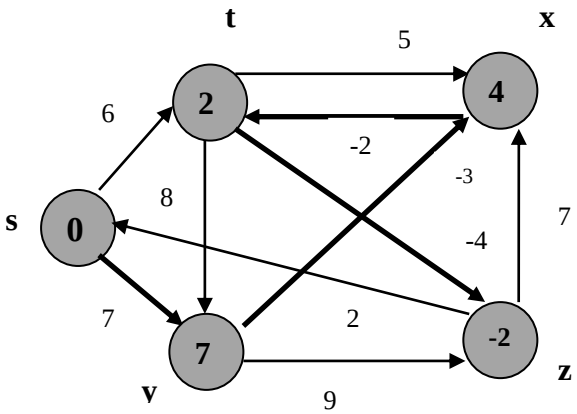
Σειρά χαλάρωσης (t,x),(t,y),**(t,z),**(x,t)
(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)

ΤΡΙΤΟ ΒΗΜΑ



Σειρά χαλάρωσης (t,x),(t,y),(t,z),**(x,t)**
(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)

ΤΕΤΑΡΤΟ ΒΗΜΑ (ΤΕΛΕΥΤΑΙΟ)



Σειρά χαλάρωσης (t,x),(t,y),**(t,z),**(x,t)
(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)

4.4. Αλγόριθμος Floyd - Warshall

Τρίτος αλγόριθμος που αποτελεί αντικείμενο μελέτης αυτού του κεφαλαίου είναι ο αλγόριθμος Floyd- Warshall[6]. Όπως και οι προηγούμενοι δυο αλγόριθμοι και αυτός έχει λάβει την ονομασία του από τους δημιουργούς του, τον Robert Floyd και τον Stephen Warshall. Χρησιμοποιείται για την εύρεση συντομότερων μονοπατιών σε σταθμισμένους γράφους με θετικά ή αρνητικά βάρη ακολουθώντας τις αρχές του δυναμικού προγραμματισμού.

Έχοντας έναν κατευθυνόμενο γράφο G=(V, E), η αναπαράσταση του οποίου πραγματοποιείται με έναν πίνακα διπλανών κορυφών (όχι λίστα γειτονικών κορυφών), τότε η είσοδος του αλγορίθμου είναι ένας πίνακας W με διαστάσεις n x n. Ο πίνακας αυτός συνιστά τα βάρη ακμών του γράφου G με n κορυφές (Ε.Σωτηρίου, 2010).

Προκύπτει δηλαδή ότι W=[wij], όπου :

$$\text{W i, j} = \begin{cases} \tau o\ \beta \acute{a} \rho o \varsigma\ \tau \eta \varsigma\ \kappa a \tau \epsilon \upsilon \theta \upsilon \nu \acute{o} \mu \epsilon \nu \eta \varsigma\ a \kappa \mu \acute{\eta} \varsigma\ (i,j), a \nu\ i <> j\ \kappa a \iota\ (i,j) \in E \\ 0, a \nu\ (i = j) \acute{\eta}\ a \nu\ (i <> j\ \kappa a \iota (i,j)\ \in E) \end{cases}$$

Η έξοδος του αλγορίθμου είναι πάλι ένας πίνακας με τις ίδιες διαστάσεις nxn, ο D=[dij], όπου dij είναι το βάρος του συντομότερου μονοπατιού από την κορυφή i στην κορυφή j. Ο αλγόριθμος θεωρεί τις ενδιάμεσες κορυφές ενός συντομότερου μονοπατιού, όπου ενδιάμεση κορυφή ενός απλού μονοπατιού p=[v1,v2,….,vl] είναι οποιαδήποτε κορυφή του p εκτός των v1 και vl, δηλαδή οποιαδήποτε κορυφή του συνόλου [v2,v3,…vl-1] (Ε.Σωτηρίου, 2010).

Έστω ότι οι κορυφές του G είναι το σύνολο V={1,2,….,n} και ας θεωρήσουμε ένα υποσύνολο {1,2,….,k} των κορυφών για κάποιο k. Για οποιοδήποτε ζεύγος κορυφών i,j ∈ V, ας θεωρήσουμε όλα τα μονοπάτια από το i στο j των οποίων οι ενδιάμεσες κορυφές προέρχονται όλες από το {1,2,….,k} και έστω p μονοπάτι ελαχίστου βάρους ανάμεσα στο μονοπάτι p και στα συντομότερα μονοπάτια από το i στο j με όλες τις ενδιάμεσες κορυφές στο σύνολο{1,2,3,…..,k-1}. Η σχέση εξαρτάται από το εάν είναι ή όχι το k ενδιάμεση κορυφή στο μονοπάτι p.

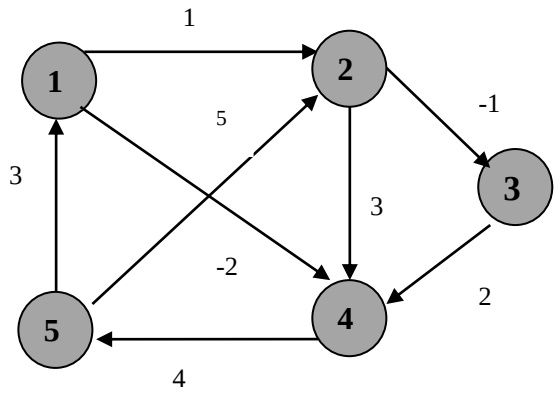
[6] Είναι επίσης γνωστός και ως αλγόριθμος του Floyd ή του Roy-Warshall ή του Roy-Floyd καθώς ο ίδιος αλγόριθμος είχε ήδη δημοσιευθεί από τον Bernard Roy το 1959.

Αν το k δεν είναι ενδιάμεση κορυφή στο μονοπάτι p, τότε όλες οι ενδιάμεσες κορυφές του μονοπατιού p είναι στο σύνολο {1,2,…..,k-1}. Έτσι, ένα συντομότερο μονοπάτι από την κορυφή i προς την κορυφή j με όλες τις ενδιάμεσες κορυφές στο σύνολο {1,2,….,k-1} είναι επίσης συντομότερο μονοπάτι από το i προς το j με όλες τις ενδιάμεσες κορυφές στο σύνολο {1,2,….,k}. Αν το k είναι ενδιάμεση κορυφή του μονοπατιού p, τότε χωρίζουμε το p σε i ~~~>(p1) και j ~~~>(p2) (Ε.Σωτηρίου, 2010). Ο χρόνος που τρέχει ο αλγόριθμος εξαρτάται από τα τριπλά εμφωλευμένα loops. Συνεπώς, η πολυπλοκότητα του συγκεκριμένου αλγορίθμου είναι ίση με $O(V^3)$.

ΨΕΥΔΟΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ BELLMAN - FORD (Ε.Σωτηρίου, 2010)

```
for  i=1 to N
   for j=1 to N
          if there is an edge from i to j
             dist[0][i][j]= the length of the edge from i to j
           else
             dist[0][i][j]= INFINITY
    dist[0][i][j]= 0
for  k=1 to N
    for i=1 to N
      for  j=1 to N
           dist[k][i][j]= min(dist[k-1][i][j], dist[k-1][i][k] + dist[k-1][k][j])
```

ΕΚΤΕΛΕΣΗ ΑΛΓΟΡΙΘΜΟΥ (Δ.Φωτάκης)



D_0 =

0	1	∞	-2	∞
∞	0	-1	3	∞
∞	∞	0	2	∞
∞	∞	∞	0	4
3	5	∞	∞	0

D_1=

0	1	∞	-2	∞
∞	0	-1	3	∞
∞	∞	0	2	∞
∞	∞	∞	0	4
3	4	∞	1	0

D_2=

0	1	0	-2	∞
∞	0	-1	3	∞
∞	∞	0	2	∞
∞	∞	∞	0	4
3	4	3	1	0

D_3 =

0	1	0	-2	∞
∞	0	-1	1	∞
∞	∞	0	2	∞
∞	∞	∞	0	4
3	4	3	1	0

$D_4 =$

0	1	0	-2	2
∞	0	-1	1	5
∞	∞	0	2	6
∞	∞	∞	0	4
3	4	3	1	0

$D_5 =$

0	1	0	-2	2
8	0	-1	1	5
9	10	0	2	6
7	8	7	0	4
3	4	3	1	0

5ο ΚΕΦΑΛΑΙΟ
ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΗΣ

Η υλοποίηση εφαρμογής η οποία επιτρέπει στο χρήστη την αναζήτηση βέλτιστων διαδρομών μεταξύ σημείων που αυτός επιλέγει σε ένα χάρτη, συνιστά το θέμα ανάπτυξης του πέμπτου κεφαλαίου. Η παρουσίαση του σχεδιασμού της εφαρμογής που υλοποιήθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας πραγματοποιείται μέσω της επεξήγησης των βασικών προγραμματιστικών εργαλείων που χρησιμοποιήθηκαν και της προβολής εικόνων των επιμέρους σταδίων της εφαρμογής.

5.1. Java : Γλώσσα προγραμματισμού της εφαρμογής

Η εφαρμογή που υλοποιήθηκε στην παρούσα πτυχιακή εργασία κωδικοποιήθηκε με την γλώσσα προγραμματισμού Java. Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού, η οποία παρουσιάστηκε επίσημα το 1995 από την εταιρεία πληροφορικής Sun Microsystems. Σχεδιάστηκε με σκοπό την ανάπτυξη εφαρμογών που θα τρέχουν σε ετερογενή δικτυακά περιβάλλοντα με βασικό πλεονέκτημά της την ανεξαρτησία του λειτουργικού συστήματος και της πλατφόρμας.

Η Java έχει τα ακόλουθα χαρακτηριστικά (Εισαγωγή στη γλώσσα προγραμματισμού Java) :

1. Αντικειμενοστραφής (ομοιότητες εντολών με τη C++).
2. Δημιουργία ανεξάρτητων εφαρμογών και applets.
3. Είναι Interpreted γλώσσα. Αυτό σημαίνει ότι ο java compiler δεν παράγει εκτελέσιμο κώδικα αλλά μια μορφή ψευδοκώδικα (bytecode) το οποίο από μόνο του δεν τρέχει σε καμία μηχανή. Προκειμένου να εκτελεστεί απαιτείται η χρήση ενός interpreter (διερμηνέα) για να μετατρέψει το bytecode σε πραγματικό εκτελέσιμο κώδικα. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα στα java

bytecodes να μπορούν να τρέξουν σε οποιοδήποτε μηχάνημα, κάτω από οποιοδήποτε λειτουργικό, αρκεί να έχει εγκατασταθεί ένας java interpreter. Επίσης, ένα άλλο χαρακτηριστικό του java bytecode είναι το μικρό του μέγεθος, μόλις λίγα Kilobytes. Αυτό το κάνει ιδανικό για μετάδοση μέσω του δικτύου.

4. Κατανεμημένη (distributed). Δηλαδή ένα πρόγραμμα σε Java είναι δυνατό να το φέρουμε από το δίκτυο και να το τρέξουμε. Επίσης είναι δυνατό διαφορετικά κομμάτια του προγράμματος να έρθουν από διαφορετικά sites.
5. Ασφαλής (secure). Στο δίκτυο όμως ελλοχεύουν πολλοί κίνδυνοι για τον χρήστη - παραλήπτη μιας δικτυακής εφαρμογής, γι' αυτό η Java έχει σχεδιαστεί έτσι ώστε να ελαχιστοποιείται η πιθανότητα προσβολής του συστήματος του χρήστη από κάποιο applet γραμμένο για τέτοιο σκοπό.
6. Είναι multithreaded. Η Java υποστηρίζει εγγενώς την χρήση πολλών threads. Προκειμένου να το επιτύχει αυτό σε συστήματα με έναν επεξεργαστή, το Java runtime system (interpreter) υλοποιεί ένα δικό του χρονοδρομολογητή (scheduler), ενώ σε συστήματα που υποστηρίζουν πολυεπεξεργασία η δημιουργία των threads ανατίθεται στο λειτουργικό σύστημα. Φυσικά όλα αυτά είναι αόρατα τόσο στον προγραμματιστή όσο και στον χρήστη.
7. Υποστηρίζει multimedia εφαρμογές. Η Java παρέχει ευκολίες στη δημιουργία multimedia εφαρμογών. Αυτό επιτυγχάνεται τόσο με την ευελιξία της σαν γλώσσα, όσο και με τις πλούσιες και συνεχώς εμπλουτιζόμενες βιβλιοθήκες της.

5.2. Java Open Street Map editor

Η οπτικοποίηση των δεδομένων του χωρικού δικτύου το οποίο εξάχθηκε από το Open Street Map καθώς και των αποτελεσμάτων της εύρεσης του συντομότερου μονοπατιού πραγματοποιήθηκε μέσω της εφαρμογής Java Open Street Map editor (JOSM). Το JOSM είναι μια desktop εφαρμογή που αναπτύχθηκε αρχικά από τον Immanuel Scholz και σήμερα συντηρείται από τον Dirk Stöcker και άλλους συνεισφέροντες.

Το JOSM συνιστά μια εφαρμογή η οποία προσφέρει μια μεγάλη ποικιλία χρήσιμων εργαλείων για ένα ευρύ φάσμα επεξεργασίας. Παρέχει τη δυνατότητα ανοίγματος χαρτών GPX και GPS απευθείας από το σκληρό δίσκο ή αρχείων κατεβασμένων από το OSM. Τα συγκεκριμένα αρχεία με το πέρας της επεξεργασίας

μπορούμε να τα ανεβάσουμε στο OSM. Επιπλέον, μέσω του JOSM μπορούμε εύκολα να κατεβάσουμε αεροφωτογραφίες οι οποίες θα αποτελέσουν το φόντο εύρεσης κάποιας περιοχής. Τέλος, το JOSM συνιστά ένα εργαλείο επεξεργασίας το οποίο υποστηρίζει επίσης τη χαρτογράφηση φωτογραφίας και τη χαρτογράφηση ήχου (JOSM, 2015).

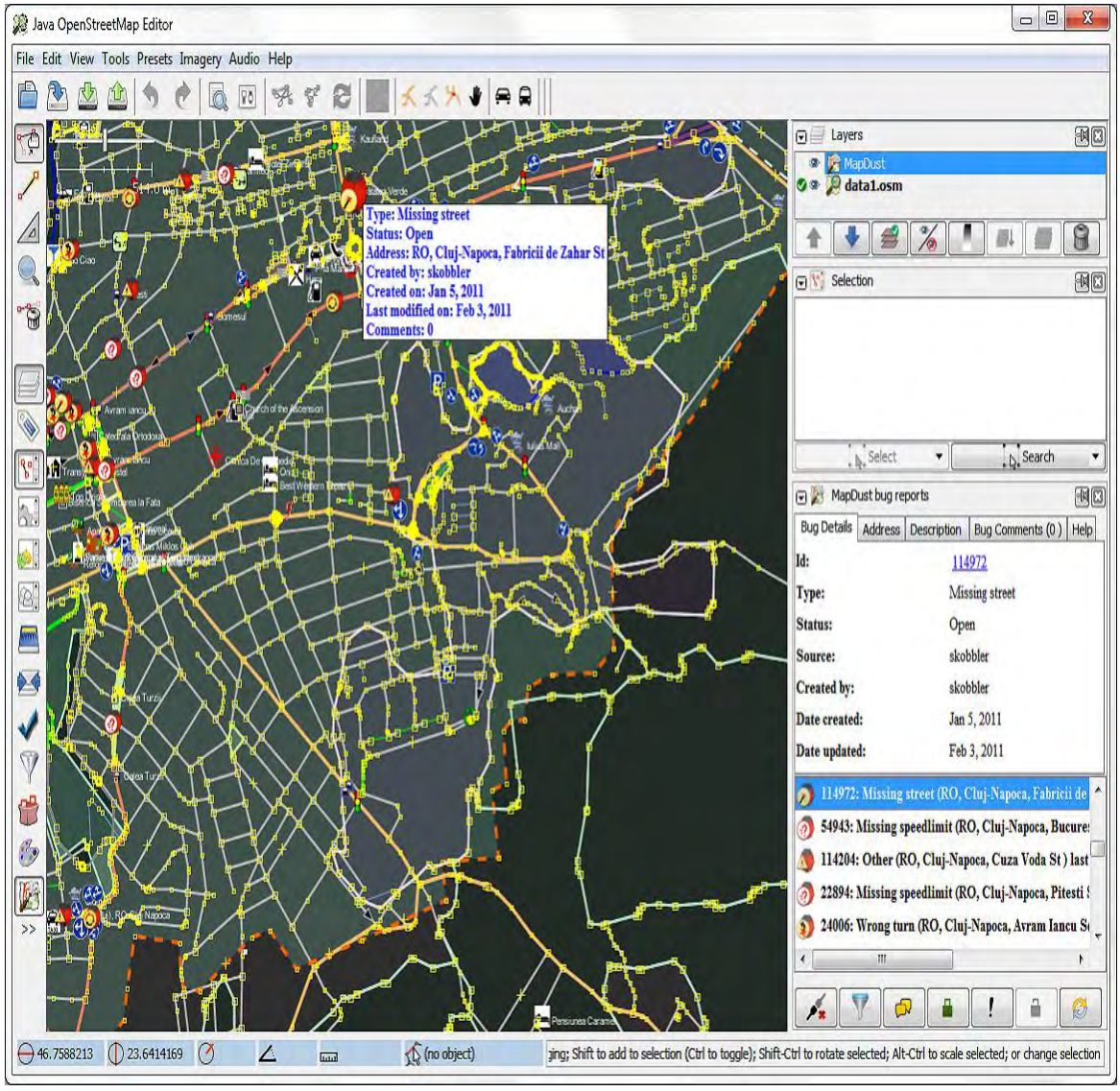
Εικόνα 23- Java Open Street Map editor (JOSM, 2015)

<u>Πλεονεκτήματα του JOSM</u> (JOSM, 2015)

- Γρήγορα κοντινά και μακρινά ζουμ για υπερ-ακριβή χαρτογράφηση.
- Μπορεί να λειτουργήσει χωρίς σύνδεση Internet, χρησιμοποιώντας ήδη κατεβασμένα αρχεία δεδομένων και επιπλέον μπορεί να συνεργαστεί με τις τοπικές φωτογραφίες και τα αρχεία GPX που είναι αποθηκευμένα στον υπολογιστή.
- Παρέχει προηγμένη λειτουργία επεξεργασίας π.χ. μπορεί να τοποθετήσει τους επιμέρους κόμβους ενός κυκλικού κόμβου σε έναν κύκλο.
- Μπορεί να χρησιμοποιήσει άμεσα ένα πλήθος αεροφωτογραφιών ως background για τον εντοπισμό χάρτη.
- Ακολουθεί μια διαδικασία επικύρωσης πριν από την μεταφόρτωση δεδομένων ώστε να αποφεύγονται τα κοινά λάθη στη χαρτογράφηση.
- Είναι αναπτυσσόμενο μ’ έναν πολύ ενεργό ρυθμό.
- Παρέχει την δυνατότητα επεξεργασίας δεδομένων ακόμα και όταν ο χρήστης είναι αποσυνδεδεμένος από το OSM και το Internet.

Μειονεκτήματα του JOSM (JOSM, 2015)

- Τα λεπτομερή στοιχεία της εφαρμογής απαιτούν περισσότερο χρόνο για την εκμάθησή τους.
- Είναι απαραίτητο να κατέβει το λογισμικό στον υπολογιστή ώστε να τρέξουμε την εφαρμογή (δεν είναι διαθέσιμη online).
- Η εφαρμογή απαιτεί την Java 7+ για να λειτουργήσει .

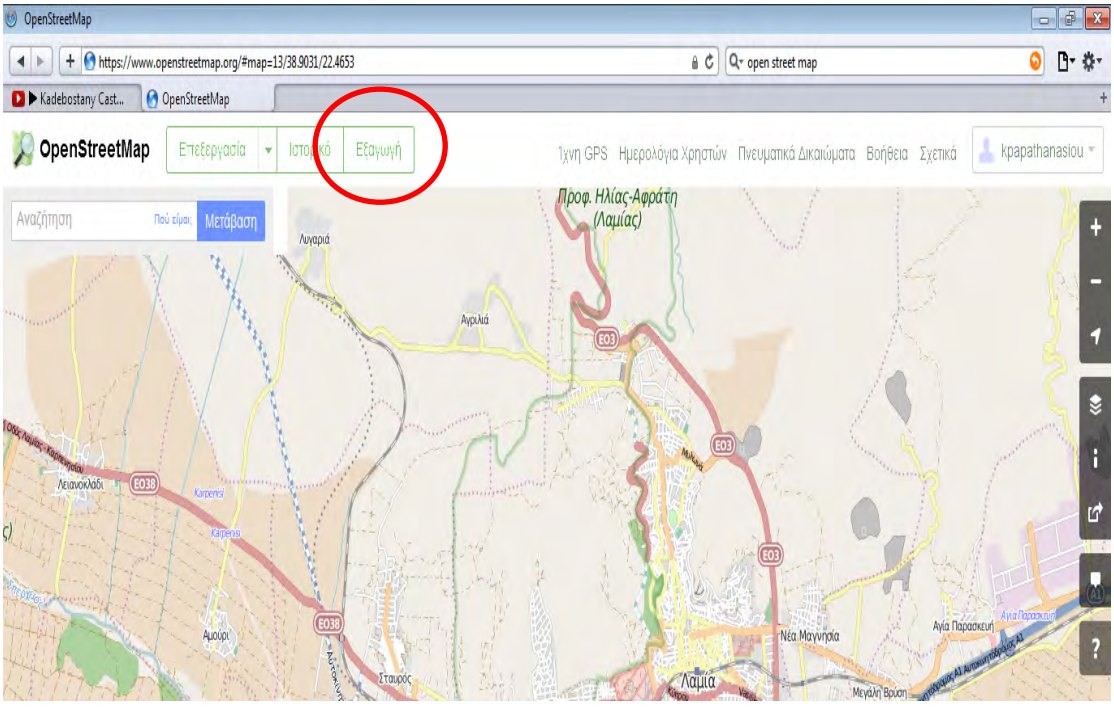


Εικόνα 24 - Παράδειγμα επεξεργασίας στο JOSM (JOSM, 2015)

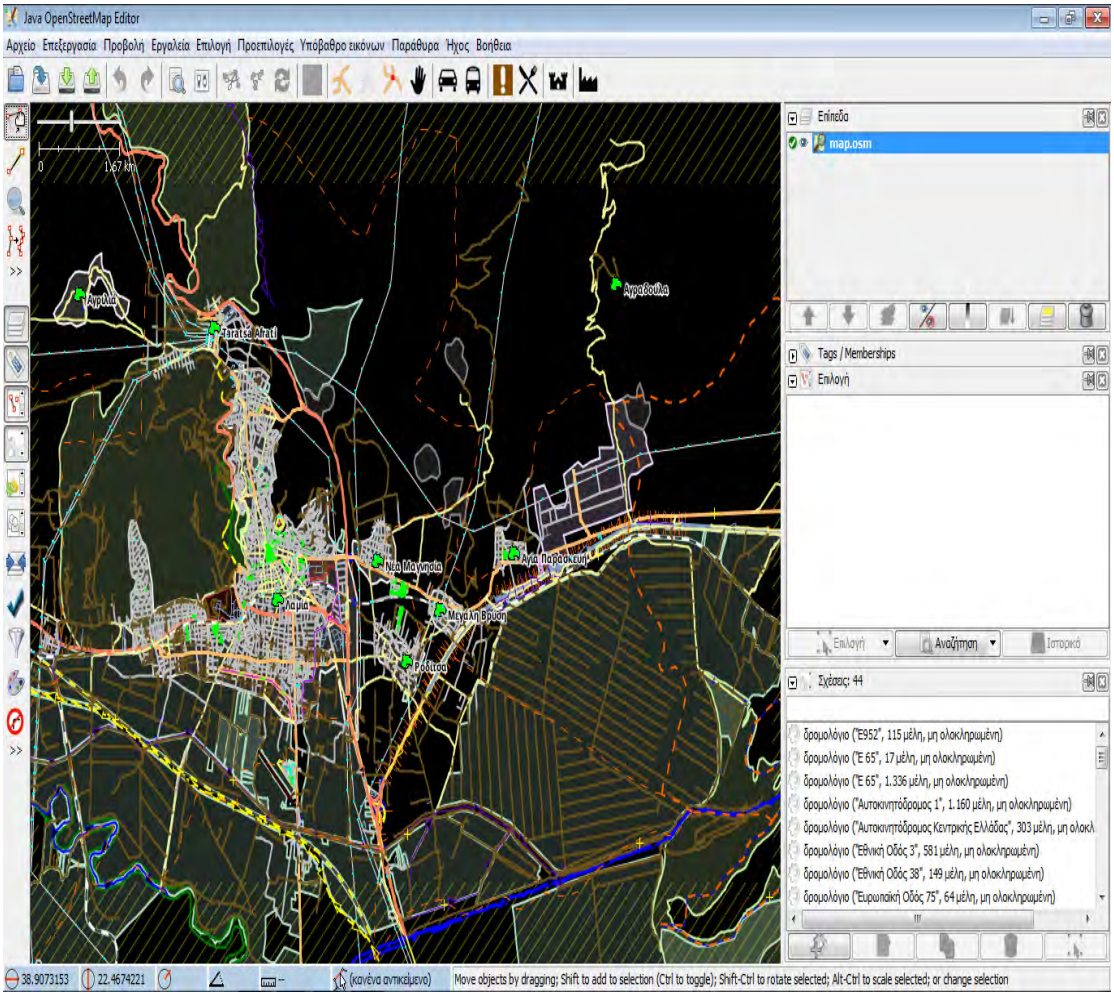
5.3. Λειτουργίες της εφαρμογής

Στόχος της υλοποίησης της παρούσας εφαρμογής εκτός από την αποτελεσματική και ακριβή εύρεση του συντομότερου μονοπατιού μεταξύ δύο σημείων του οδικού δικτύου που θα καθορίσει ο χρήστης, είναι και η δημιουργία ενός γραφικού περιβάλλοντος φιλικού προς το χρήστη μέσω του οποίου θα έχει τη δυνατότητα εύρεσης της διαδρομής που αναζητά εύκολα και γρήγορα. Το σύνολο των λειτουργιών που εκτελούνται στην εφαρμογή που σχεδιάστηκε, συνδέεται άμεσα και έμμεσα με το χαρτογραφικό υπόβαθρο το οποίο συνιστά το βασικότερο στοιχείο της υλοποίησης. Οι επιμέρους λειτουργίες που διεξάγονται σ' αυτό αφορούν κυρίως την οπτικοποίηση του, την επιλογή σημείων στο χάρτη από τον χρήστη και την οπτικοποίηση των αποτελεσμάτων των αλγορίθμων δρομολόγησης που προκύπτουν υπό μορφή osm.

Αρχικά, κρίνεται απαραίτητο ο χρήστης να επιλέξει από την ιστοσελίδα του Open Street Map τον χάρτη πάνω στον οποίο εκείνος επιθυμεί να εντοπίσει την συντομότερη διαδρομή. Επιλέγοντας τον χάρτη και πατώντας <<Εξαγωγή>> αποθηκεύεται στον υπολογιστή του ένα osm αρχείο σε μορφή xml το οποίο μπορεί να οπτικοποιηθεί με την εφαρμογή Java Open Street Map editor. Σε περίπτωση ενός ήδη αποθηκευμένου αρχείου απαιτείται από τον χρήστη η απευθείας φόρτωσή του στο JOSM.



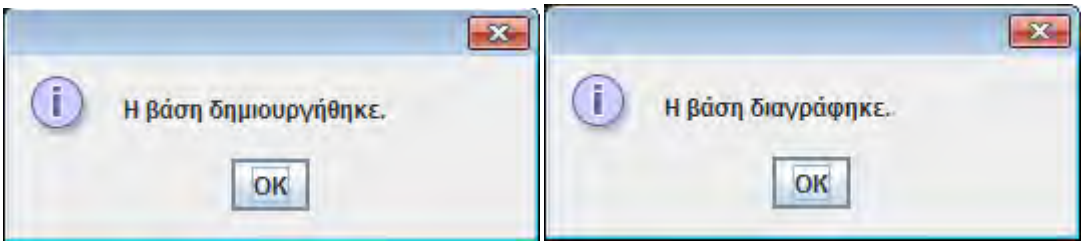
Εικόνα 25 - Open Street Map -Εξαγωγή Χάρτη



Εικόνα 26- Οπτικοποίηση του χάρτη με την εφαρμογή JOSM

Η αλληλεπίδραση του χρήστη με το χάρτη του OSM είναι απαραίτητη, προκειμένου τα χωρικά δεδομένα του osm αρχείου να αποσταλούν στη βάση όπου και λαμβάνουν χώρα οι πίνακες για τον εντοπισμό των στοιχείων της αφετηρίας και του προορισμού που θα προσδιορίσει ο χρήστης, αλλά και των υπολοίπων κόμβων του χάρτη. Η δημιουργία της βάσης αποτελεί έτσι το επόμενο βήμα του χρήστη, το οποίο πραγματοποιείται μέσα από την επιλογή << ΔΗΜΙΟΥΡΓΙΑ ΒΑΣΗΣ >> της εφαρμογής. Επιπλέον, σε περίπτωση ήδη υπάρχουσας βάσης παρέχεται η δυνατότητα στο χρήστη να την διαγράψει με την επιλογή << ΔΙΑΓΡΑΦΗ ΒΑΣΗΣ >> ώστε να φορτωθούν σωστά τα νέα δεδομένα. Η εφαρμογή ενημερώνει τον χρήστη τόσο για την επιτυχή δημιουργία της βάσης όσο και για την διαγραφή της ήδη υπάρχουσας.

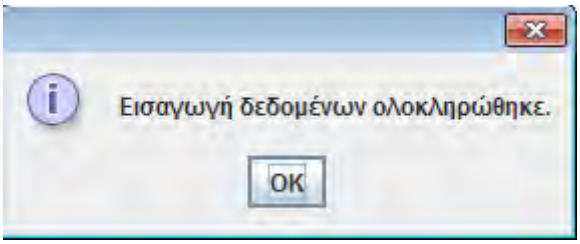
Εικόνα 27- Επιλογές Δημιουργίας βάσης και Διαγραφής Βάσης στην εφαρμογή



Εικόνα 28- Ενημέρωση για την επιτυχή δημιουργία και την διαγραφή της βάσης

Εφόσον έχει δημιουργεί η βάση μέσα στην οποία θα αποθηκευτούν τα δεδομένα που παρέχονται στο osm αρχείο ο χρήστης πρέπει να προχωρήσει στο επόμενο βήμα το οποίο είναι η εισαγωγή αυτών των πληροφοριών στη βάση. Το βήμα αυτό πραγματοποιείται μέσω της επιλογής <<ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΈΝΩΝ>> και επιπλέον για την επιτυχία αυτής ο χρήστης ενημερώνεται μέσω αντίστοιχου παραθύρου. Κατά την διάρκεια των παραπάνω επιλογών τα κουμπιά της ενεργοποίησης των αλγορίθμων στα αριστερά της εφαρμογής είναι απενεργοποιημένα για λόγους ασφαλείας.

Εικόνα 29- Επιλογή της εισαγωγής δεδομένων στη βάση



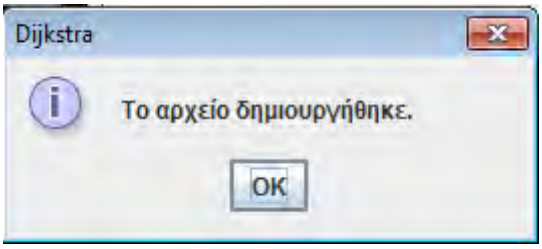
Εικόνα 30-Ενημέρωση για την επιτυχή εισαγωγή δεδομένων στη βάση

Η ολοκλήρωση της εισαγωγής των δεδομένων στη βάση αποτελεί το εναρκτήριο βήμα για την δημιουργία του γράφου, μέσω του οποίου θα γίνει η διαγραμματική αναπαράστασή του οδικού δικτύου. Το βήμα αυτό υλοποιείται μέσω της επιλογής <<ΕΚΤΕΛΕΣΗ>> και επιπλέον πραγματοποιείται ενεργοποίηση των επιλογών για τους τρείς αλγορίθμους δρομολόγησης, Dijkstra, Bellman - Ford και Floyd – Warshall. Έτσι, η εφαρμογή συνιστά ένα διαδραστικό μέσο αλληλεπίδρασης μεταξύ του χρήστη και της χωρικής βάσης δεδομένων μέσω του κώδικα java που καθορίζει τις λειτουργίες της.

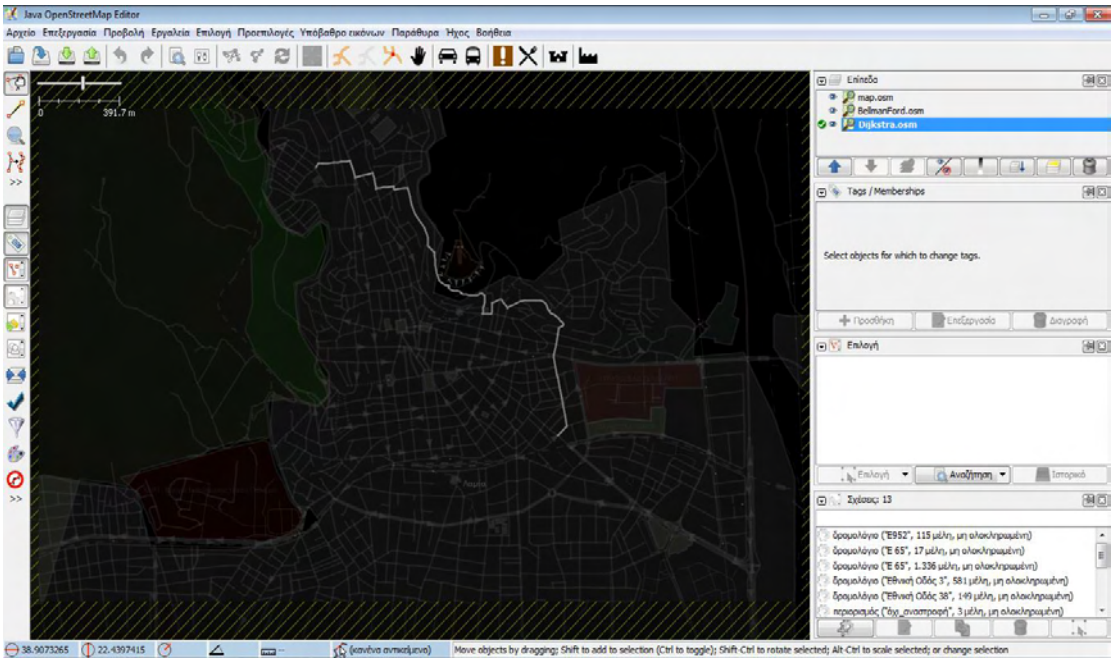
Εικόνα 31- Επιλογή της δημιουργίας του γράφου και ενεργοποίηση των επιλογών των τριών αλγορίθμων

Στη συνέχεια ο χρήστης μπορεί να επιλέξει την εκκίνηση και τον προσδιορισμό που εκείνος επιθυμεί για την εύρεση του συντομότερου μονοπατιού. Απαραίτητη κρίνεται και η επιλογή του αλγορίθμου δρομολόγησης βάση του οποίου θα υπολογιστή το προαναφερθέν μονοπάτι. Με το πέρας της εκτέλεσης του αλγορίθμου δημιουργείται ένα osm αρχείο με το όνομα του αλγορίθμου, το οποίο συνιστά το μέσο οπτικοποίησης των αποτελεσμάτων που προκύπτουν από τους υπολογισμούς που λαμβάνουν χώρα στη βάση, καθώς το συντομότερο μονοπάτι που κάθε φορά προκύπτει αναπαρίσταται ως μια διανυσματική γραμμική οντότητα στο χαρτογραφικό υπόβαθρο το οποίο έχει ήδη φορτωθεί στο JOSM. Σε αυτό το στάδιο της εφαρμογής ο χρήστης και πάλι ενημερώνεται για την επιτυχή εκτέλεση του αλγορίθμου δρομολόγησης και την δημιουργία του αντίστοιχου osm αρχείου, το οποίο είναι αποθηκευμένο στον φάκελο της εφαρμογής.

Εικόνα 32- Επιλογή της εκκίνησης και του προορισμού από τον χρήστη



Εικόνα 33- Ενημέρωση για την επιτυχή εκτέλεση του αλγορίθμου και την δημιουργία του αντίστοιχου αρχείου



Εικόνα 34- Άνοιγμα με το JOSM του νέου osm αρχείου που περιέχει το συντομότερο μονοπάτι

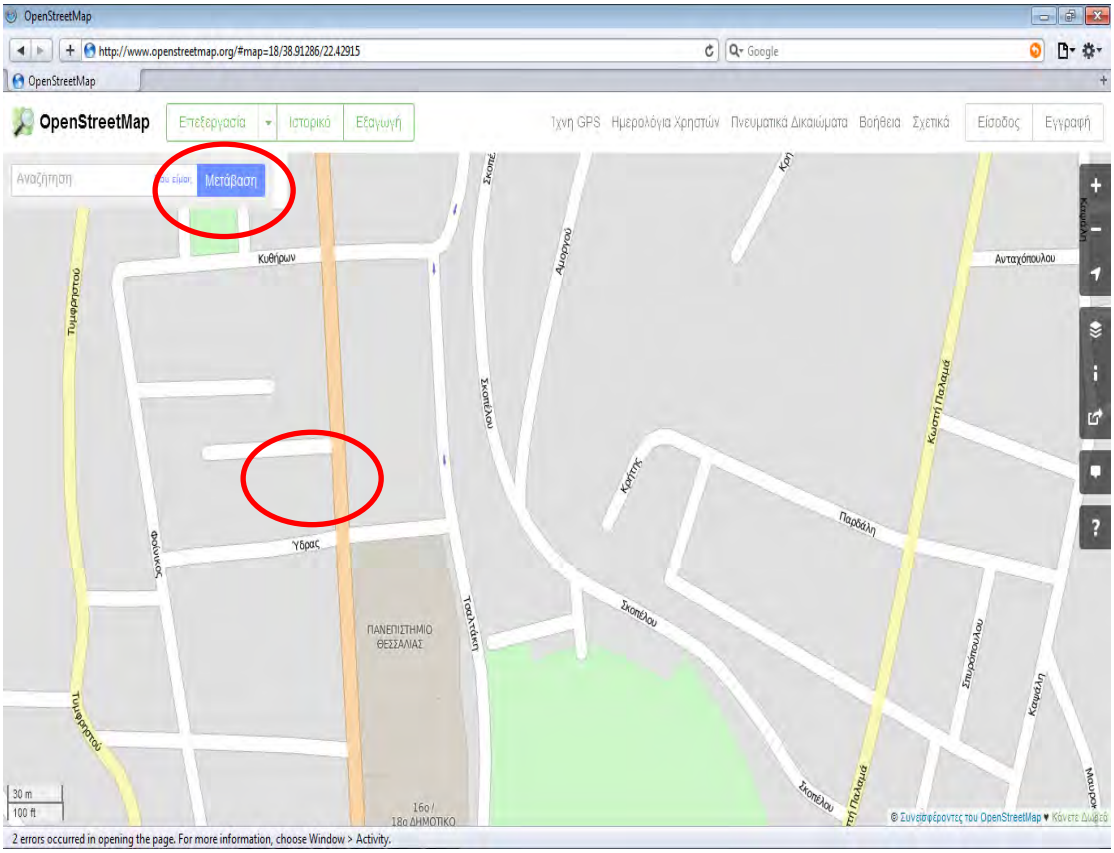
Στις επόμενες δύο υποενότητες του πέμπτου κεφαλαίου, παρουσιάζονται δύο παραδείγματα και τα ενδεικτικά αποτελέσματα που προέκυψαν ύστερα από την αναζήτηση της συντομότερης διαδρομής μεταξύ δύο σημείων του δικτύου με τη βοήθεια των αλγόριθμων Dijkstra, Bellman - Ford και Floyd – Warshall. Η αναζήτηση της βέλτιστης διαδρομής υλοποιήθηκε σε δύο διαφορετικού μεγέθους σύνολα δεδομένων. Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα, μαζί με τα απαραίτητα σχόλια που αποσαφηνίζουν το περιεχόμενό του.

5.4. Πρώτο παράδειγμα εκτέλεσης της εφαρμογής (μικρό σύνολο δεδομένων)

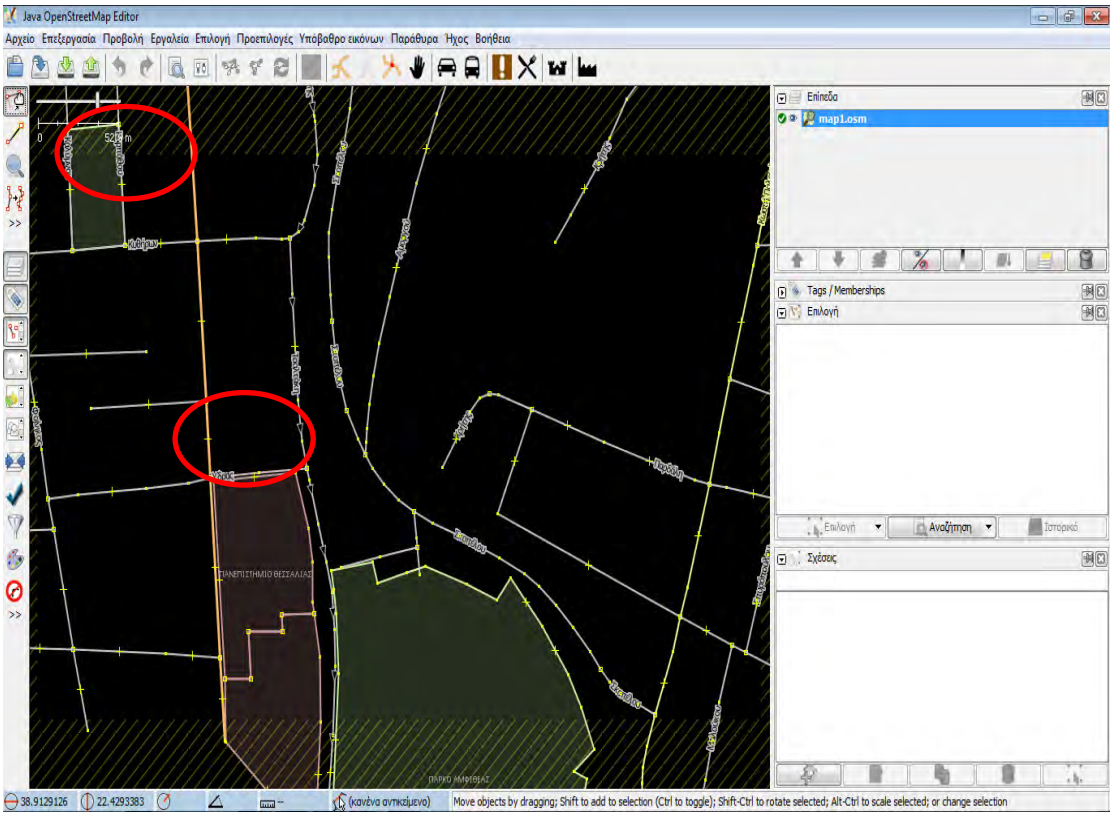
Στο πρώτο παράδειγμα εκτέλεσης της εφαρμογής το σύνολο δεδομένων που έχει επιλεχθεί είναι ένας μικρού μεγέθους χάρτης. Ο χάρτης αναπαριστά μια μικρή περιοχή της Λαμίας και αποτελείται από 248 κόμβους και 32 δρόμους. Ακολουθώντας τα βήματα που προαναφέρθηκαν στην προηγούμενη ενότητα ο χρήστης :

1. Εξάγει τον χάρτη από το OSM και τον αποθηκεύει στον υπολογιστή του.
2. Οπτικοποιεί το osm αρχείο του χάρτη στο JOSM.
3. Δημιουργεί την βάση μέσω της εφαρμογής.
4. Εισάγει τα δεδομένα του χάρτη στη βάση.
5. Δημιουργεί τον γράφο με τους κόμβους του χάρτη.
6. Επιλέγει την εκκίνηση και τον προορισμό για την εύρεση του συντομότερου μονοπατιού. Στην προκειμένη περίπτωση έχει επιλεχθεί ως εκκίνηση η οδός Καρπάθου και ως προορισμός η οδός Ύδρας.
7. Επιλέγει τον αλγόριθμο δρομολόγησης .
8. Φορτώνει το αρχείο με την συντομότερη διαδρομή που δημιουργήθηκε με την εκτέλεση του αντίστοιχου αλγορίθμου.

Ακολουθούν εικόνες οι οποίες παρουσιάζουν με ακρίβεια τα βήματα που προαναφέρθηκαν. Τα βήματα που είναι ίδια με τα βήματα που παρουσιάστηκαν στην προηγούμενη ενότητα παραλείπονται.

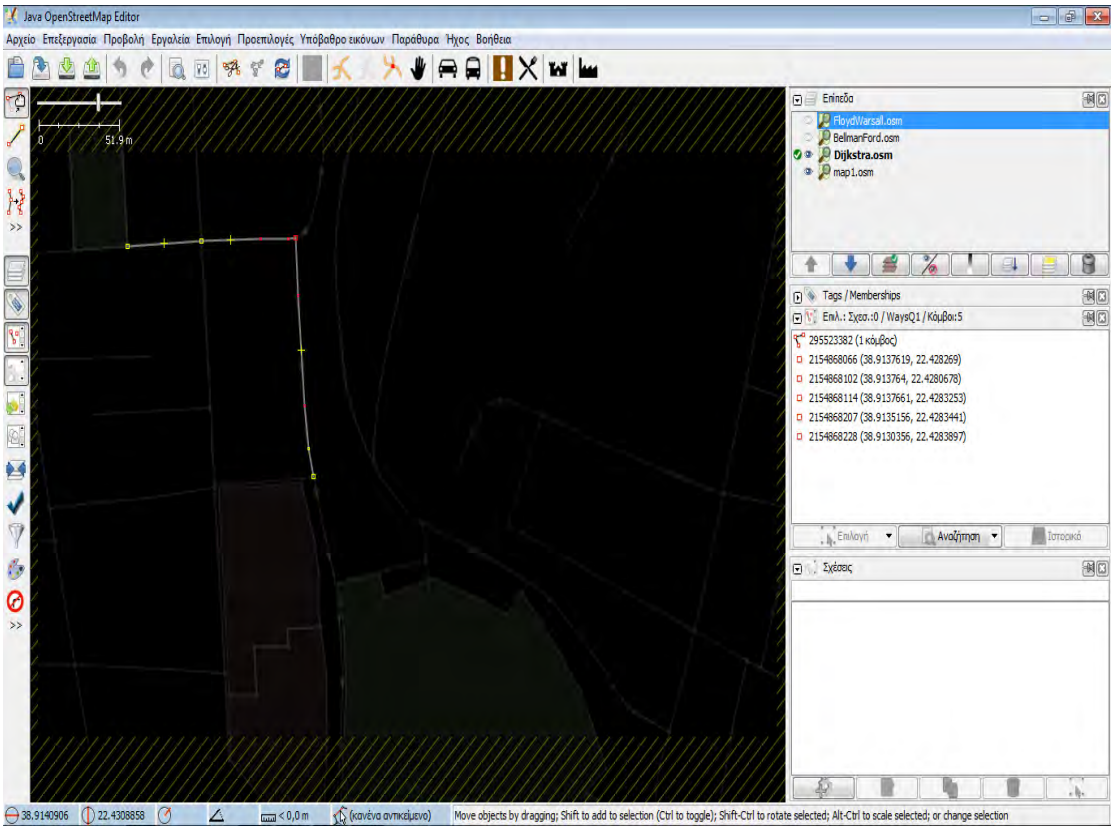


Εικόνα 35- Πρώτο παράδειγμα χάρτη-Εξαγωγή από το OSM

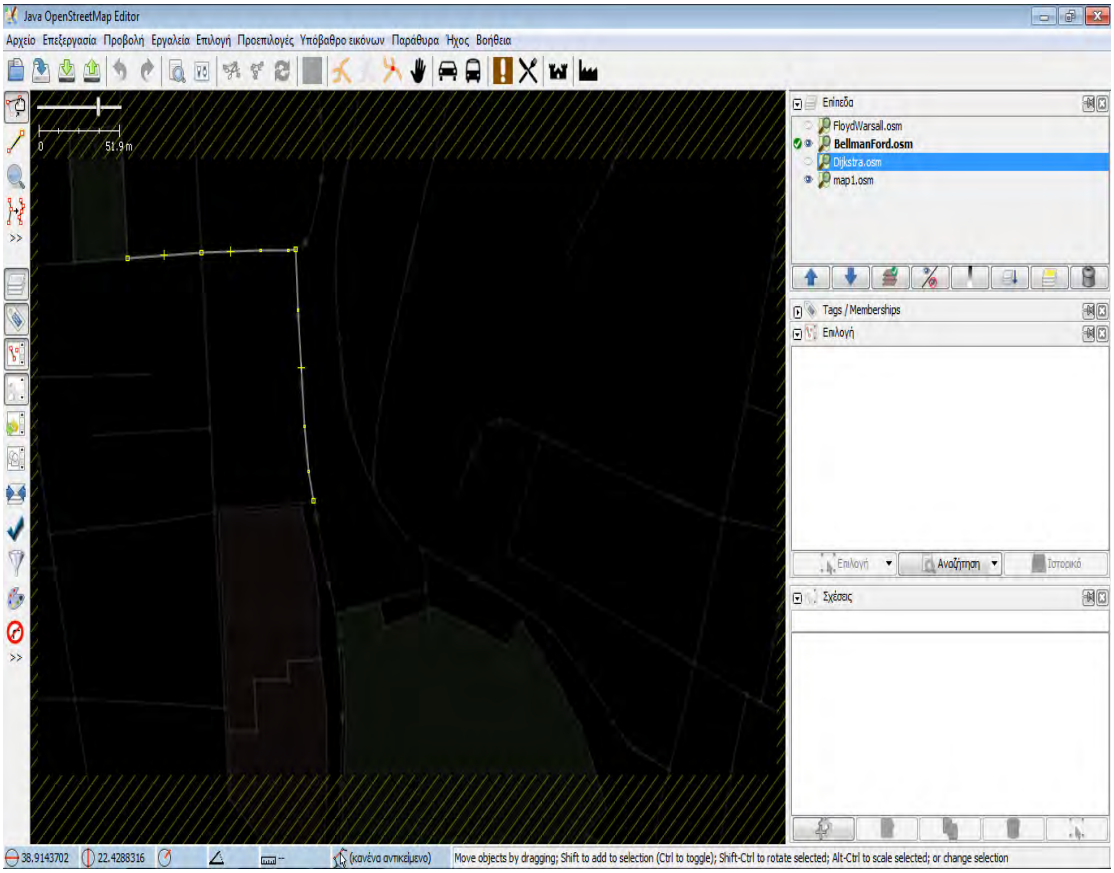


Εικόνα 36- Οπτικοποίηση του χάρτη με το JOSM

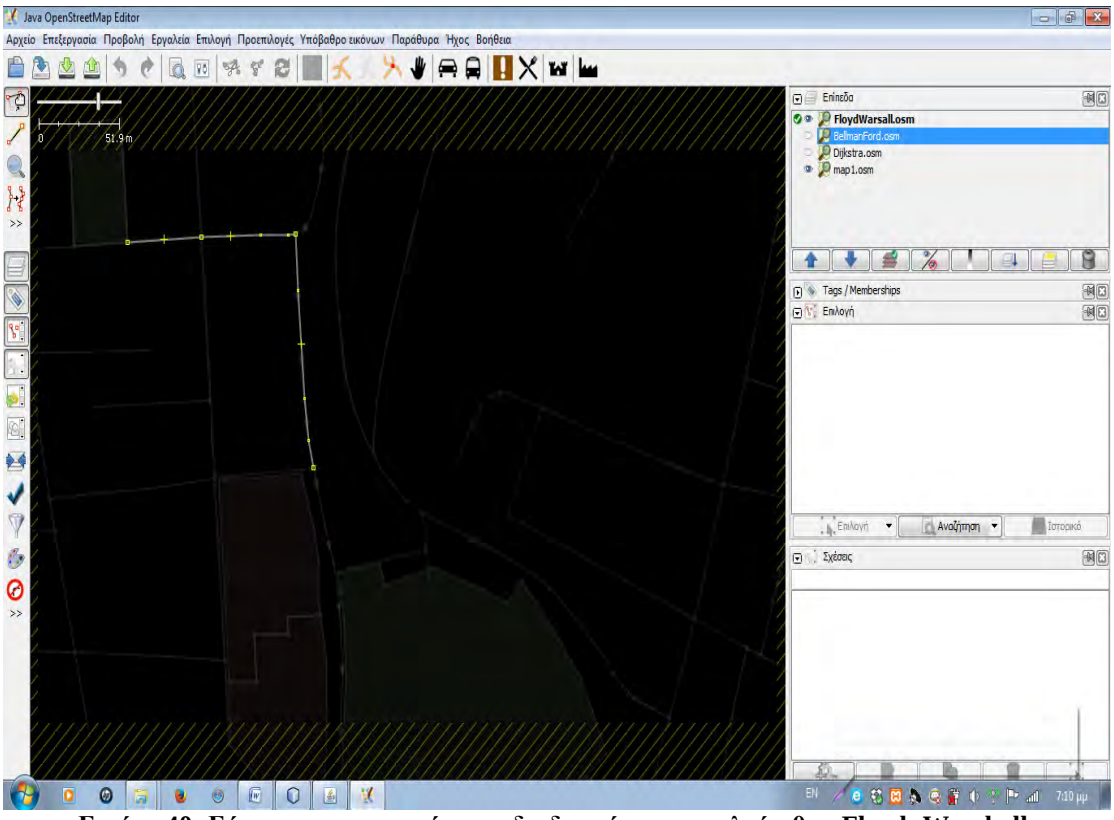
Εικόνα 37- Επιλογή εκκίνησης και προορισμού



Εικόνα 38- Εύρεση της συντομότερης διαδρομής με τον αλγόριθμο Dijkstra



Εικόνα 39- Εύρεση της συντομότερης διαδρομής με τον αλγόριθμο Bellman - Ford



Εικόνα 40- Εύρεση της συντομότερης διαδρομής με τον αλγόριθμο Floyd -Warshall

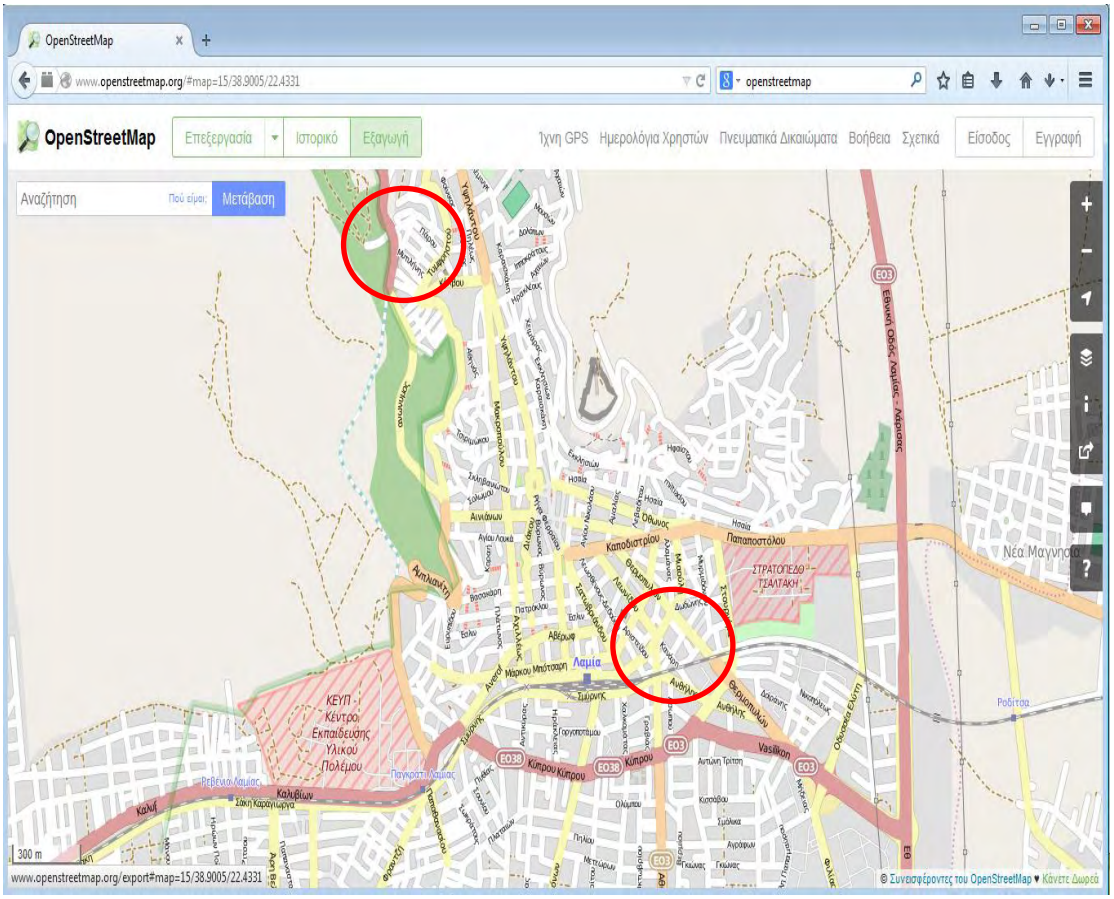
Οι τρεις αλγόριθμοι δρομολόγησης Dijkstra, Bellman - Ford και Floyd – Warshall, στο πρώτο παράδειγμα εκτέλεσης της διαδρομής εντόπισαν το συντομότερο μονοπάτι ακολουθώντας την ίδια διαδρομή, όπως παρουσιάζεται και στις προηγούμενες εικόνες.

5.5. Δεύτερο παράδειγμα εκτέλεσης της εφαρμογής (μεγάλο σύνολο δεδομένων)

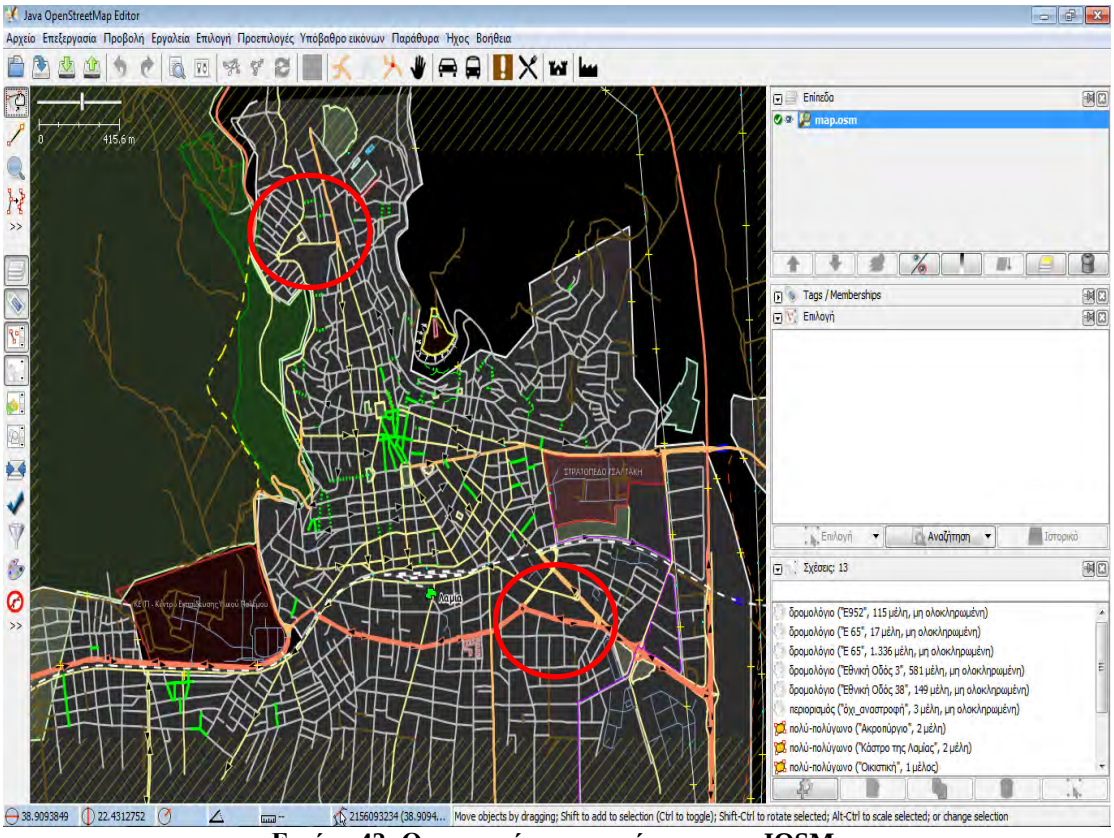
Στο δεύτερο παράδειγμα εκτέλεσης της εφαρμογής το σύνολο δεδομένων που έχει επιλεχθεί είναι μεγαλύτερου μεγέθους από τον προηγούμενο χάρτη. Αναπαριστά και αυτός περιοχή της Λαμίας αλλά αποτελείται από αρκετά περισσότερους κόμβους και δρόμους, για την ακρίβεια από 9.859 κόμβους και 1.035 δρόμους. Ακολουθώντας τα βήματα που προαναφέρθηκαν στην προηγούμενη ενότητα ο χρήστης :

1. Εξάγει τον χάρτη από το OSM και τον αποθηκεύει στον υπολογιστή του.
2. Οπτικοποιεί το osm αρχείο του χάρτη στο JOSM.
3. Δημιουργεί την βάση μέσω της εφαρμογής.
4. Εισάγει τα δεδομένα του χάρτη στη βάση.
5. Δημιουργεί τον γράφο με τους κόμβους του χάρτη.
6. Επιλέγει την εκκίνηση και τον προορισμό για την εύρεση του συντομότερου μονοπατιού. Στην προκειμένη περίπτωση έχει επιλεχθεί ως εκκίνηση η οδός Πάρου και ως προορισμός η οδός Κανάρη.
7. Επιλέγει τον αλγόριθμο δρομολόγησης .
8. Φορτώνει το αρχείο με την συντομότερη διαδρομή που δημιουργήθηκε με την εκτέλεση του αντίστοιχου αλγορίθμου.

Ακολουθούν εικόνες οι οποίες παρουσιάζουν με ακρίβεια τα βήματα που προαναφέρθηκαν. Τα βήματα που είναι ίδια με τα βήματα που παρουσιάστηκαν στην ενότητα 5.3 παραλείπονται.

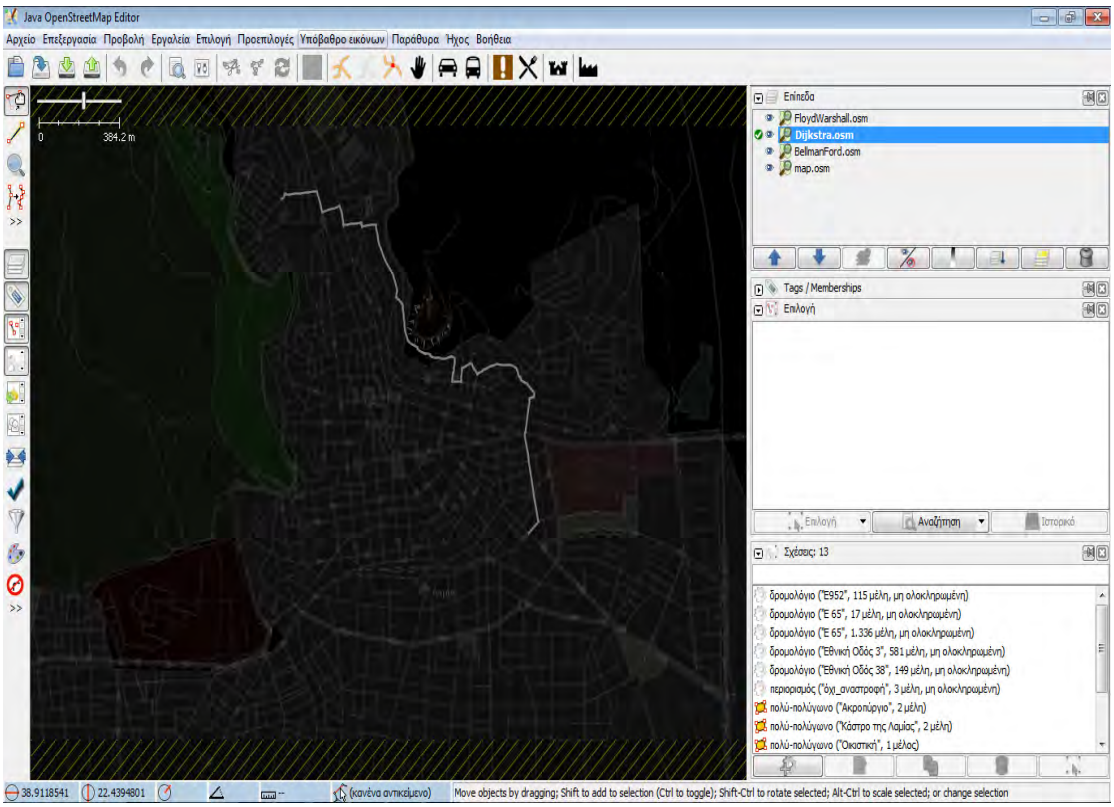


Εικόνα 41-Δεύτερο παράδειγμα χάρτη - Εξαγωγή από το OSM

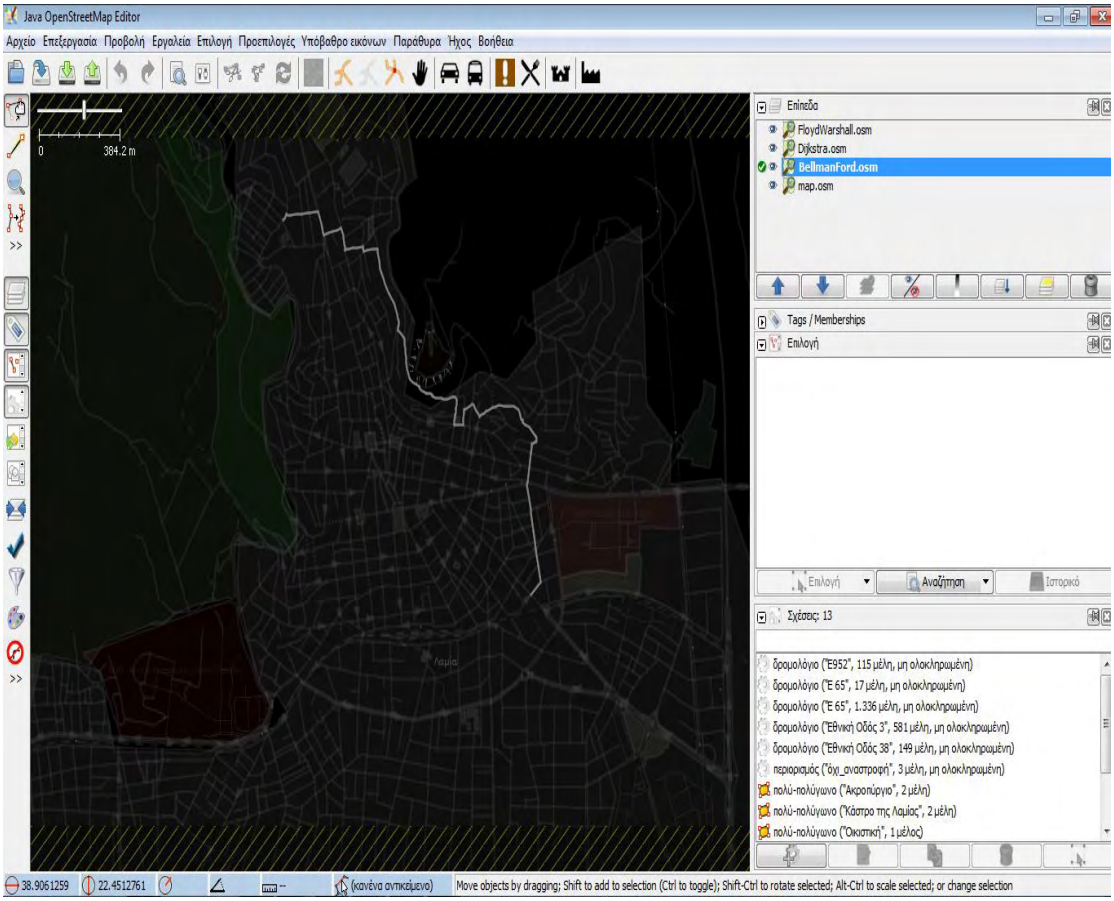


Εικόνα 42- Οπτικοποίηση του χάρτη με το JOSM

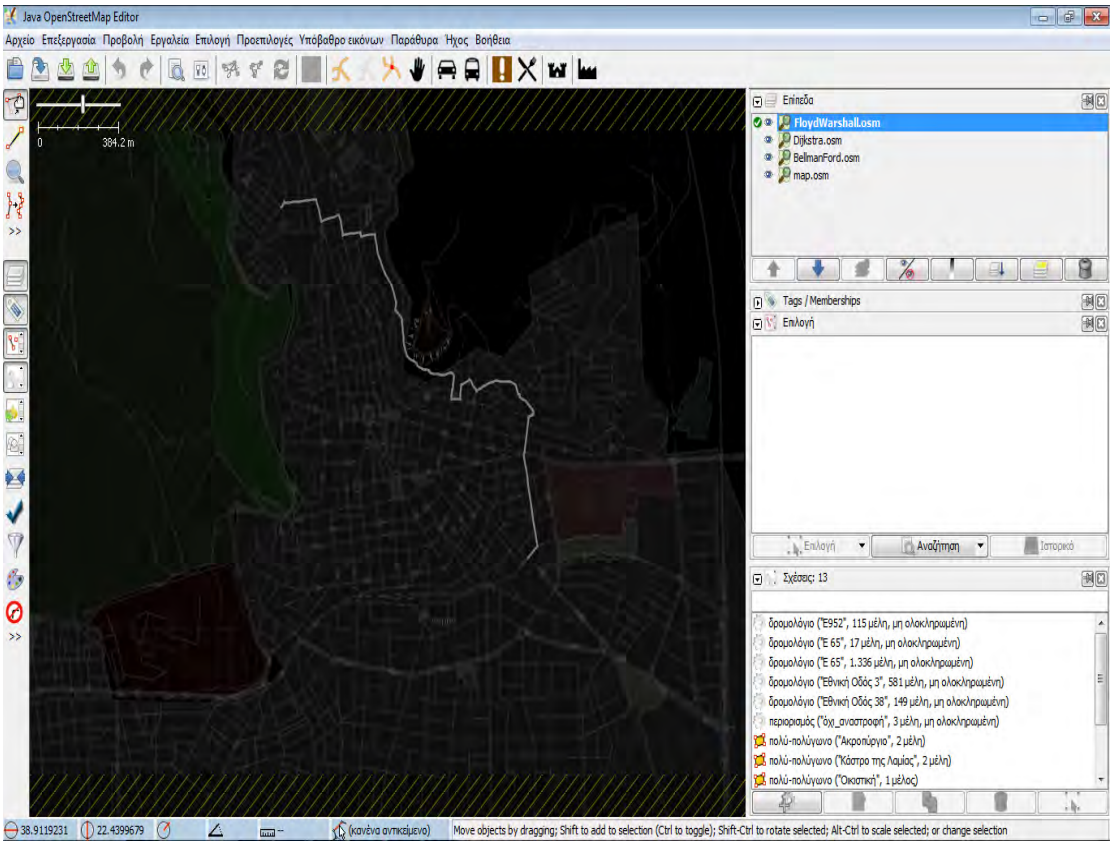
Εικόνα 43- Επιλογή εκκίνησης και προορισμού



Εικόνα 44- Εύρεση της συντομότερης διαδρομής με τον αλγόριθμο Dijkstra



Εικόνα 45- Εύρεση της συντομότερης διαδρομής με τον αλγόριθμο Bellman - Ford



Εικόνα 46- Εύρεση της συντομότερης διαδρομής με τον αλγόριθμο Floyd - Warshall

Οι τρεις αλγόριθμοι δρομολόγησης Dijkstra, Bellman - Ford και Floyd – Warshall, και στο δεύτερο παράδειγμα εκτέλεσης της διαδρομής εντόπισαν το συντομότερο μονοπάτι ακολουθώντας την ίδια διαδρομή όπως παρουσιάζεται και στις προηγούμενες εικόνες. Ωστόσο, παρά την εύρεση του ίδιου συντομότερου μονοπατιού οι τρείς αλγόριθμοι παρουσίασαν διαφορές, κυρίως στο μεγάλο σύνολο δεδομένων, στο χρόνο εκτέλεσής τους και στο μέγεθος της χρήσης της μνήμης RAM. Τα αποτελέσματα των παραπάνω δοκιμών θα παρουσιαστούν αναλυτικότερα στο επόμενο και τελευταίο κεφάλαιο της πτυχιακής.

6ο ΚΕΦΑΛΑΙΟ
ΣΥΜΠΕΡΑΣΜΑΤΑ-ΜΕΛΛΟΝΤΙΚΕΣ ΕΦΑΡΜΟΓΕΣ

Ο στόχος της παρούσας εργασίας ήταν η υλοποίηση μιας εφαρμογής δρομολόγησης, μέσω της οποίας ένας χρήστης θα έχει τη δυνατότητα εύρεσης της συντομότερης διαδρομής που πρέπει να ακολουθήσει για τη μετάβασή του από ένα σημείο του δικτύου σε κάποιο άλλο. Ως χαρτογραφικό υπόβαθρο μπορούσε να χρησιμοποιήσει ένα χάρτη, η επιλογή του οποίου γινόταν μέσω της ανοικτής βάσης δεδομένων του Open Street Map. Μέσω της εφαρμογής ο χρήστης επέλεγε τα σημεία που όριζαν την προς αναζήτηση διαδρομή, την εκκίνηση και τον προορισμό αλλά και τον επιθυμητό αλγόριθμο δρομολόγησης. Το αίτημά του επεξεργαζόταν και στη συνέχεια ο χρήστης λάμβανε το αποτέλεσμα της αναζήτησης σε ένα αρχείο χάρτη το οποίο μπορούσε να αναπαρασταθεί στην εφαρμογή Java Open Street Map editor ως γεωμετρική οντότητα.

Για το σκοπό αυτό, κρίθηκε απαραίτητη η δημιουργία μιας χωρικής βάσης δεδομένων τύπου MySQL στην οποία αποθηκεύτηκε το σύνολο των περιγραφικών και γεωμετρικών χαρακτηριστικών των γεωμετρικών οντοτήτων του δικτύου. Το δίκτυο αναπαραστάθηκε στη βάση ως γράφος, οι κόμβοι του οποίου συνέδεαν τα επιμέρους τμήματα του οδικού δικτύου. Επιπλέον, αναγκαίος κρίθηκε και ο σχεδιασμός μιας εφαρμογής σε γλώσσα Java μέσω της οποίας ο χρήστης αλληλεπιδρούσε με τη βάση δεδομένων, στέλνοντας ερωτήματα και ανακτώντας τα από τη βάση δεδομένων με τη μορφή ενός χάρτη. Πρόκειται, λοιπόν, για μια εφαρμογή GIS με στόχο τη διαχείριση της βάσης δεδομένων και την επίλυση συγκεκριμένων προβλημάτων.

Η εφαρμογή που σχεδιάστηκε συνιστά το περιβάλλον μέσα από το οποίο ο χρήστης αλληλεπιδρά με την βάση, καθορίζει τα δεδομένα εισόδου στον κώδικα και λαμβάνει τα αποτελέσματα της εκάστοτε αναζήτησης. Συνεπώς, οι διαδικασίες υλοποίησης της συγκεκριμένης εφαρμογής που αναπτύχθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας προϋποθέτουν την ανάπτυξη επιμέρους εφαρμογών, κάθε μια από τις οποίες επιτελεί μια συγκεκριμένη λειτουργία. Το τελικό

αποτέλεσμα, προκύπτει ως συνδυασμός των επιμέρους εφαρμογών οι οποίες επικοινωνούν μεταξύ τους σε επίπεδο ανταλλαγής πληροφοριών.

Βασικό πλεονέκτημα της εφαρμογής είναι ότι τα διαθέσιμα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν κατά τη διαδικασία ανάπτυξης της εφαρμογής συνιστούν ελεύθερα λογισμικά ανοικτού κώδικα, οι δυνατότητες των οποίων ανταποκρίθηκαν πλήρως στις απαιτήσεις της σχεδιαζόμενης εφαρμογής. Επιπλέον, τόσο το περιβάλλον της εφαρμογής, όσο και του Open Street Map και του Java Open Street Map editor στο οποίο πραγματοποιήθηκε η οπτικοποίηση των αποτελεσμάτων, είναι φιλικά προς το χρήστη και σχετικά απλά στην εκμάθησή τους. Το JOSM αποτελεί ένα εργαλείο κυρίως οπτικοποίησης δεδομένων και κατά δεύτερο ρόλο ένα εργαλείο παροχής λειτουργιών ανάλυσης και επεξεργασίας. Επομένως, οι λειτουργίες ανάλυσης που παρέχονται καλύπτουν ένα ευρύ φάσμα επεξεργασίας, για τις ανάγκες των περισσότερων χρηστών.

6.1. Συμπεράσματα δοκιμών

Όπως προαναφέρθηκε στο πέμπτο κεφάλαιο της εργασίας και οι τρεις αλγόριθμοι δρομολόγησης : Dijkstra, Bellman - Ford και Floyd – Warshall, εντόπισαν το συντομότερο μονοπάτι ακολουθώντας την ίδια διαδρομή και στα δύο σύνολα δεδομένων. Ωστόσο, παρά την εύρεση του ίδιου συντομότερου μονοπατιού οι τρείς αλγόριθμοι παρουσίασαν διαφορές, κυρίως στο μεγάλο σύνολο δεδομένων, στο χρόνο εκτέλεσής τους και στο μέγεθος της χρήσης της μνήμης RAM.

Πιο συγκεκριμένα, στο μικρό χάρτη ο οποίος αποτελεί ένα μικρό σύνολο δεδομένων, για την ακρίβεια αποτελείται από 248 κόμβους και 32 δρόμους, οι αλγόριθμοι Dijkstra, Bellman - Ford και Floyd – Warshall χρειάστηκαν σχεδόν τον ίδιο χρόνο εκτέλεσης και τα αποτελέσματα των αναζητήσεών τους ήταν ίδια. Στην αντίθεση περίπτωση του δεύτερου παραδείγματος όπου ο χάρτης είναι μεγαλύτερος και το σύνολο δεδομένων αποτελείται από 9.859 κόμβους και 1.035 δρόμους, οι αλγόριθμοι Dijkstra, και Bellman – Ford χρειάστηκαν σχεδόν τον ίδιο χρόνο εκτέλεσης, ενώ ο αλγόριθμος Floyd – Warshall πραγματοποίησε αναζήτηση συντομότερης διαδρομής σε απαγορευτικό χρόνο εκτέλεσης. Το γεγονός αυτό προέκυψε από την επιλογή της αναδρομικής εκδοχής του αλγορίθμου Floyd – Warshall. Ως εκ τούτου, ο αλγόριθμος χρησιμοποίησε ένα πολύ μεγάλο κομμάτι της

μνήμης RAM για να εκτελεστεί και σε περιπτώσεις αρκετά μεγάλων συνόλων δεδομένων δεν κατάφερε καν να τερματίσει. Ακολουθεί πίνακας σχέσης δοκιμής-χρόνου για τους τρεις αλγόριθμους δρομολόγησης. Ο χρόνος εκτέλεσης των αλγορίθμων μετράται σε millisecond.

ΠΙΝΑΚΑΣ ΣΧΕΣΗΣ ΧΡΟΝΟΥ(millisecond) - ΔΟΚΙΜΗΣ ΤΩΝ ΤΡΙΩΝ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΤΑ ΔΥΟ ΣΥΝΟΛΑ ΔΕΔΟΜΕΝΩΝ

ΑΛΓΟΡΙΘΜΟΙ / **ΣΥΝΟΛΑ ΔΕΔΟΜΕΝΩΝ**	**DIJKSTRA**	**BELLMAN-FORD**	**FLOYD-WARSHALL**
ΠΡΩΤΟ (248 κόμβοι- 32 δρόμοι)	**1762 ms**	**1653 ms**	**1763 ms**
ΔΕΥΤΕΡΟ (9859 κόμβοι - 1035 δρόμοι)	**2496 ms**	**2340 ms**	∞

6.2. Μελλοντικές εφαρμογές

Συμπερασματικά, τα προγραμματιστικά εργαλεία που διατίθενται για το σχεδιασμό διαδικτυακών εφαρμογών και την ανάπτυξη εφαρμογών βάσεων δεδομένων, επαρκούν για την ανάπτυξη απλών και αποτελεσματικών εφαρμογών δρομολόγησης οι οποίες ανταποκρίνονται στις απαιτήσεις των χρηστών για πλοήγηση. Ωστόσο, η ραγδαία ανάπτυξη και εξάπλωση των υπηρεσιών πλοήγησης με τη βοήθεια διαδικτυακών χαρτών, απαιτεί την όσο το δυνατόν πιο αντιπροσωπευτική γεωγραφική αναπαράσταση του οδικού δικτύου, την εύρεση της γεωγραφικής θέσης

του οχήματος πάνω σ ’αυτό, την εύρεση της βέλτιστης διαδρομής λαμβάνοντας υπόψη και πληροφορίες για την κυκλοφοριακή κατάσταση, αλλά και τη διαρκή ανανέωση και ενίσχυση των διαθέσιμων προγραμματιστικών εργαλείων με περισσότερες δυνατότητες που ανταποκρίνονται στις διαρκώς αυξανόμενες ανάγκες των χρηστών. Τέλος, εξαιρετικά σημαντικό είναι και το ζήτημα της ελεύθερης διάθεσης χωρικών δεδομένων (αρχεία τύπου xml, shapefile, kml κ.λπ.) που απαιτούνται για την ανάπτυξη αυτού του είδους των εφαρμογών, καθώς ένας σημαντικός αριθμός χρηστών δεν έχει πρόσβαση σε υψηλής ακρίβειας και ποιότητας χωρικά δεδομένα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. *apache friends - xampp for windows*. (2014). Retrieved 16 Δεκέμβριος, 2014, from XAMPP installers and Downloads for Apache friends: www.apachefriends.org
2. Compucon SA. (2014, Δεκέμβριος 15). *Συστήματα Βάσεων Δεδομένων (oo-or.ppt).* Retrieved Δεκέμριος 15, 2014, from Συστήματα Βάσεων Δεδομένων (oo-or.ppt): delab.csd.auth.gr/courses/c_mmdb/oo-or.ppt
3. Euler, L. (1741). *Solutio problematis ad geometriam situs pertinentis.*
4. *Gis Mapping Software,Solutions, Services,Map Apps and Data*. (n.d.). Retrieved Νοέμβριος 10, 2014, from Gis Mapping Software,Solutions, Services,Map Apps and Data: www.esri.com/software/arcgis
5. H.Thomas, C., E.Charles, L., L.Ronald, R., & Clifford, S. (2001). *Introduction to algorithms.* Massachussets Institute of Technology. The MIT Press .
6. *isites-Τί είναι μια MySql βάση δεδομένων* . (2014). Retrieved Δεκέμβριος 2014, from http://www.isites.gr/clients/knowledgebase/19/---MySQL--.html
7. Joan, A. M., & Robin, W. J. (2000). *Graphs and Applications: An Introductory Approach.* Springer Publication .
8. *JOSM*. (2015). Retrieved Ιανουάριος 16, 2015, from JOSM: https://josm.openstreetmap.de/
9. Marathon Data Systems. (2011). *ArcGis I Εισαγωγή στο ArcGis.* Marathon Data Systems.
10. *MySql* . (2014). Retrieved Δεκέμβριος 2014, from http://www.mysql.com
11. *OpenStreetMap*. (2014). Retrieved Δεκέμβριος 19, 2014, from OpenStreetMap: https://www.openstreetmap.org
12. *PHP: Hypertext Preprocessor* . (2014). Retrieved Δεκέμβριος 17, 2014, from http://php.net/
13. S.Shekhar, & S.Chawla. (2003). *Spatial Databases : A tour.* Prentice Hall .
14. S.Shekhar, S.Chawla, S.Ravada, A.Fetterer, X.Liu, & C.Lu. (Jan-Feb 1999). *Spatial Databases-Accoplishments and Research Needs.* IEEE Transections on Knowledge and Data Engineering.

15. T.Evans. (n.d.). *http://www.indiana.edu/~gisci/courses/g338/images/scale_vector_raster.jpg.* Retrieved Δεκέμβριος 16, 2014, from Departhment of geography- Indiana University: http://www.indiana.edu/~gisci/courses/g338/images/scale_vector_raster.jpg
16. *The Perl Programming Language* . (2014). Retrieved Δεκέμβριος 19, 2014, from https://www.perl.org/
17. Δ.Φωτάκης. *Συντομότερες Διαδρομές.* Εθνικό Μετσόβειο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών .
18. Ε.Σωτηρίου, Κ. (2010). *Ανάπτυξη υπερεσίας trip planning δρομολόγηση υπό περιοσισμούς, εφαρμογή σε οδικό δίκτυο.* Μεταπτυχιακό Δίπλωμα, Χαροκόπειο Πανεπιστήμιο , Τμήμα Γεωγραφίας, Αθήνα .
19. *Εισαγωγή στη γλώσσα προγραμματισμού Java* . Εθνικό Μετσόβειο Πολυτεχνείο, Τμήμα Ηλεκτολόγων Μηχανικών και Μηχανικλων Η/Υ.
20. *Κέντρο ΠΛΗΝΕΤ Ν.Φλώρινας*. (2014). Retrieved Νοεμβρίου 12, 2014, from Κέντρο ΠΛΗΝΕΤ Ν.Φλώρινας: http://dide.flo.sch.gr/Plinet/Tutorials/Tutorials-SQL.html
21. Μανωλόπουλος, Γ. (1996). *Μαθήματα Θεωρίας Γράφων.* Αθήνα: Εκδόσεις Νέων Τεχνολογιών.
22. Μαργαρίτης, Κ. Γ. (2014). *Parallel Distributed Processing Laboratory (PDP Lab).* Retrieved Δεκέμβριος 16, 2014, from http://www.it.uom.gr/project/parallel/kef10/prog10.htm
23. *Οι γέφυρες του Königsberg* . (2014, Φεβρουαρίου 28). Retrieved Ιούνιος 15, 2014, from http://thalesandfriends.org/el/2014/02/28/oi-gefires-tou-konigsberg/
24. Παπαδοπούλου, Χ. (2011). *Επεξεργασία και Αναπαράσταση Βέλτιστων Διαδρομών.* Διπλωματική Εργασία , Εθνικό Μετσόβιο Πολυτεχνείο , Αθήνα.
25. Παπαταξιάρχης, Β. Ε. (2006). *Οντολογική Προσέγγιση στη Μοντελοποίηση Εσωτερικών.* Πτυχιακή εργασία, ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ, ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ, Αθήνα.
26. Πλέσσας, Α. (2008). *Επιστήμη και Τεχνολογία Υπολογιστων* . Διπλωματική εργασία , Πολυτεχνική Σχολή - Πανεπιστήμιο Πατρών , Μηχανικών Η/Υ & Πληροφορικής.

27. Σομπόνης, Γ. (2012). *Ευφυής Προσδιορισμός Βέλτιστων Διαδρομών βάσει Μεθόδων Μηχανικής Μάθησης*. Διπλωματική Εργασία, Εθνικό Μετσόβειο Πολυτεχνείο , Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών , Αθήνα .

ΠΑΡΑΡΤΗΜΑ

Στο παρόν παράρτημα παρουσιάζεται ο κώδικας της εφαρμογής υλοποιημένος σε γλώσσα προγραμματισμού Java. Το project της εφαρμογής απαρτίζεται από 4 κλάσεις, η επεξήγηση των οποίων παρατίθεται παρακάτω.

1. Κλάση XamppConnector

```
private static final String JDBCDriver = "com.mysql.jdbc.Driver";
String host = "jdbc:mysql://localhost:3306/";

    // Δημιουργία βάσης δεδομένων

public void createDatabase(String dbName, String username, String password)
{
   Connection conn = null;
   Statement stmt = null;

   try{
      Class.forName(JDBCDriver);

      conn = DriverManager.getConnection(host, username, password);

      stmt = conn.createStatement();

      String sql = "CREATE DATABASE IF NOT EXISTS " + dbName + "
DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci";
      stmt.executeUpdate(sql);

   }catch(SQLException | ClassNotFoundException se){
   }finally {
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
         }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
      }//end finally try
   }//end try
}
```

```
    // Διαγραφή βάσης δεδομένων

public void dropDatabase(String dbName, String username, String password) {
   Connection conn = null;
   Statement stmt = null;

   try{
      conn = establishXamppConnection(dbName, username, password);

      stmt = conn.createStatement();

      String sql = "DROP DATABASE IF EXISTS " + dbName;
      stmt.executeUpdate(sql);
   }catch(SQLException | ClassNotFoundException se){
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
         }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
      }//end finally try
   }//end try
}

    // Δημιουργία όλων των πινάκων της βάσης δεδομένων

public void createTables(String dbName, String username, String password)
{
   String sql;

   sql = "CREATE TABLE IF NOT EXISTS nodes " +
         "(node_id INT(15) NOT NULL AUTO_INCREMENT, " +
         "lnode_id BIGINT(15) NOT NULL DEFAULT 0, " +
         "lat FLOAT DEFAULT NULL, " +
         "lon FLOAT DEFAULT NULL, " +
         "PRIMARY KEY (node_id))";

   createTable(dbName, username, password, sql);

   sql = "CREATE TABLE IF NOT EXISTS ways " +
         "(way_id INT(15) NOT NULL AUTO_INCREMENT, " +
         "lway_id BIGINT(15) NOT NULL DEFAULT 0, " +
         "access VARCHAR(11) DEFAULT NULL, " +
         "cycleway VARCHAR(11) DEFAULT NULL, " +
         "lanes INT(11) DEFAULT NULL, " +
```

```
            "name VARCHAR(50) NOT NULL DEFAULT '-', " +
            "oneway VARCHAR(11) NOT NULL DEFAULT 'no', " +
            "maxspeed INT(11) DEFAULT NULL, " +
            "PRIMARY KEY (way_id))";
        createTable(dbName, username, password, sql);

        sql = "CREATE TABLE IF NOT EXISTS way_nodes " +
            "(id INT(15) NOT NULL AUTO_INCREMENT, " +
            "way INT(15) DEFAULT NULL, " +
            "node INT(15) DEFAULT NULL, " +
            "PRIMARY KEY (id))";
        createTable(dbName, username, password, sql);

        sql = "CREATE TABLE IF NOT EXISTS name_node " +
            "(name VARCHAR(50) NOT NULL DEFAULT '-', " +
            "node INT(15) DEFAULT NULL)";

        createTable(dbName, username, password, sql);
    }
```

// Δημιουργία ενός πίνακα

```
    public void createTable(String dbName, String username, String password, String
sql)
    {
        Connection conn = null;
        Statement stmt = null;

        try{
            conn = establishXamppConnection(dbName, username, password);

            stmt = conn.createStatement();

            stmt.executeUpdate(sql);

        }catch(SQLException | ClassNotFoundException se){
        }finally{
            //finally block used to close resources
            try{
                if(stmt!=null)
                    stmt.close();
                }catch(SQLException se2){
            }// nothing we can do
            try{
                if(conn!=null)
                    conn.close();
            }catch(SQLException se){
            }//end finally try
        }//end try
    }
```

// Σύνδεση στη βάση δεδομένων

```
public Connection establishXamppConnection(String dbName, String username, String password) throws SQLException, ClassNotFoundException {

    Class.forName(JDBCDriver);

    Connection conn = DriverManager.getConnection(host+dbName+"?useUnicode=true&characterEncoding=utf-8", username, password);

    return conn;
}
```

2. Κλάση OpenStreetMap

// Διάβασμα από αρχείο και γέμισμα της βάσης

```
public void readOSMFile(String mapPath, String dbName, String username, String password) throws IOException, SQLException, ClassNotFoundException{
    FileReader osmFile = new FileReader(mapPath);
    BufferedReader dataBuffer = new BufferedReader(osmFile);
    StringTokenizer st;
    int i;
    String line = null;
    String word = null;
    String id = null;
    String[] mapData = null;

    XamppConnector xampp = new XamppConnector();

    Connection conn = xampp.establishXamppConnection(dbName, username, password);

    Statement stm = (Statement) conn.createStatement();
    ResultSet rs = null;
    ResultSetMetaData rsMetaData = null;

    i=1;
    line=dataBuffer.readLine();
    while (line != null) {

      st = new StringTokenizer(line, " '=\"");
      word = st.nextToken();

      // load nodes
      if(word.equals("<node")) {
```

```
            while(!word.equals("/>") && !word.equals(">")) {
               if(word.equals("id")) {
                  word = st.nextToken();

                  rs = stm.executeQuery("SELECT lnode_id FROM nodes WHERE
lnode_id=" + word);
                  if (!rs.next()) {
                     id = word;
                     stm.execute("INSERT INTO nodes(lnode_id) VALUES (" + word +
")");
                  }
               }
               else if(word.equals("lat")) {
                  word = st.nextToken();

                  if (id != null) {
                     stm.execute("UPDATE nodes SET lat=" + word + " WHERE
lnode_id=" + id);
                  }
               }
               else if(word.equals("lon")) {
                  word = st.nextToken();

                  if (id != null) {
                     stm.execute("UPDATE nodes SET lon=" + word + " WHERE
lnode_id=" + id);
                  }
               }
               word = st.nextToken();
            }
         }
         id = null;

         int firstWayNode = 0;

         // load ways
         if(word.equals("<way")) {
            while(!word.equals(">") && !word.equals("/>")) {
               if(word.equals("id")) {
                  word = st.nextToken();

                  rs = stm.executeQuery("SELECT lway_id FROM ways WHERE
lway_id=" + word);
                  if (!rs.next()) {
                     id = word;
                     stm.execute("INSERT INTO ways(lway_id) VALUES (" + word +
")");
                  }
               }
               word = st.nextToken();
```

```
            }

            line=dataBuffer.readLine();
            st = new StringTokenizer(line, " '=\"");
            word = st.nextToken();
            while(!word.equals("</way>")) {
               if(word.equals("<nd")) {
                  word = st.nextToken();
                  word = st.nextToken();

                  if(id != null) {
                     String query = "SELECT node_id FROM nodes WHERE
lnode_id=" + word;
                     ResultSet rs1 = stm.executeQuery(query);
                     rs1.next();

                     firstWayNode = rs1.getInt(1);

                     stm.execute("INSERT INTO way_nodes(way, node) VALUES (" +
id + "," + rs1.getInt(1) + ")");
                  }
               }

               if(word.equals("<tag")) {
                  word = st.nextToken();
                  word = st.nextToken();
                  if (word.equals("access")) {
                     word = st.nextToken();
                     word = st.nextToken();

                     if(id != null) {
                        stm.execute("UPDATE ways SET access=\"" + word + "\"
WHERE lway_id=" + id);
                     }
                  }
                  else if (word.equals("cycleway")) {
                     word = st.nextToken();
                     word = st.nextToken();

                     if(id != null) {
                        stm.execute("UPDATE ways SET cycleway=\"" + word + "\"
WHERE lway_id=" + id);
                     }
                  }
                  else if (word.equals("lanes")) {
                     word = st.nextToken();
                     word = st.nextToken();

                     if(id != null) {
```

```
                        stm.execute("UPDATE ways SET lanes=" + word + " WHERE
lway_id=" + id);
                    }
                }
                else if (word.equals("maxspeed")) {
                    word = st.nextToken();
                    word = st.nextToken();

                    if(id != null) {
                        stm.execute("UPDATE ways SET maxspeed=" + word + "
WHERE lway_id=" + id);
                    }
                }
                else if (word.equals("name")) {
                    String word2;

                    st.nextToken();
                    word = st.nextToken();
                    word2 = st.nextToken();

                    while(!word2.equals("/>")) {
                        word = word + " " + word2;
                        word2 = st.nextToken();
                    }

                    if(id != null) {
                        stm.execute("UPDATE ways SET name=\"" + word + "\"
WHERE lway_id=" + id);
                        stm.execute("INSERT INTO name_node(name, node) VALUES
(\"" + word + "\"," + firstWayNode + ")");
                    }
                }
                else if (word.equals("oneway")) {
                    word = st.nextToken();
                    word = st.nextToken();

                    if(id != null) {
                        stm.execute("UPDATE ways SET oneway=\"" + word + "\"
WHERE lway_id=" + id);
                    }
                }
            }

            line=dataBuffer.readLine();
            st = new StringTokenizer(line, " '=\"");
            word = st.nextToken();
        }
    }

    line=dataBuffer.readLine();
```

```
    }

    dataBuffer.close();
    stm.close();
    conn.close();
  }
```

3. Κλάση ProjectBrain

```
    public void retrieveFromDatabase(Connection con) throws SQLException
{
  conn = con;
}
```

// Προσθήκη κόμβων στο γράφο

```
void addVertexFromDatabase() throws SQLException
{
    Statement stmt = null;

    String query = "SELECT node_id, lnode_id FROM nodes WHERE 1";

    try {
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query);

        while (rs.next()) {
            graph.addVertex(rs.getInt(1));
        }
    }
    catch (SQLException e) {
    }
    finally {
        if (stmt != null) {
            stmt.close();
        }
    }
}
```

// Δημιουργία δρόμων μέσω της ένωσης των κόμβων

```
 void addEdgeFromDatabase() throws SQLException
{
    Statement stmt = null;
    boolean flag;

    String query = "SELECT way, node, lat, lon FROM way_nodes, nodes WHERE
way_nodes.node=nodes.node_id";
```

```
    try {
       stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery(query);
       DefaultWeightedEdge e;

       flag = rs.next();

       while (flag) {
          way_id1 = rs.getLong(1);
          node_id1 = rs.getInt(2);
          lat1 = rs.getFloat(3);
          lon1 = rs.getFloat(4);

          flag = rs.next();
          if(flag) {
             way_id2 = rs.getLong(1);
             node_id2 = rs.getInt(2);
             lat2 = rs.getFloat(3);
             lon2 = rs.getFloat(4);
             weight = calculateWeight();

             if(way_id1==way_id2) {
                e = graph.addEdge(node_id1, node_id2);
                if(e != null)
                   graph.setEdgeWeight(e, weight);

                e = graph.addEdge(node_id2, node_id1);
                 if(e != null)
                   graph.setEdgeWeight(e, weight);
             }
          }
       }
    }
    catch (SQLException e) {
    }
    finally {
       if (stmt != null) {
          stmt.close();
       }
    }
  }
```

// Υπολογισμός του βάρους ανάμεσα σε δύο κόμβους

```
float calculateWeight()
  {
    return (float) (Math.pow(lon2-lon1, 2) + Math.pow(lat2-lat1, 2));
  }
```

// Δημιουργία γράφου

```
  public void CreateDirectedGraph() throws SQLException
  {
     addVertexFromDatabase();
     addEdgeFromDatabase();
  }
```

// Υλοποίηση του αλγορίθμου Dijkstra

```
 public void runDijkstraAlgorithm(int start, int end) throws SQLException
  {
     //System.out.println("\n-------------------------------------------------------------");
     //System.out.println("Shortest path from " + start + " to " + end + " :\n");
     List shortest_path = DijkstraShortestPath.findPathBetween(graph, start, end);

     if (shortest_path != null) {
        //System.out.println("Dijkstra Path         : " + shortest_path);
        //System.out.println("-------------------------------------------------------------");

        parseNodesFromShortertPath(shortest_path, end);
        getNodesLongIdFromShortestPathList();
     }
     else
        System.out.println("NO PATH!!!");
  }
```

// Υλοποίηση του αλγορίθμου Bellman-Ford

```
  public void runBellmanFordAlgorithm(int start, int end) throws SQLException
  {
     //System.out.println("\n-------------------------------------------------------------");
     //System.out.println("Shortest path from " + start + " to " + end + " :\n");
     List shortest_path = BellmanFordShortestPath.findPathBetween(graph, start,
end);

     if (shortest_path != null) {
        //System.out.println("Bellman Ford Path     : " + shortest_path);
        //System.out.println("-------------------------------------------------------------");

        parseNodesFromShortertPath(shortest_path, end);
        getNodesLongIdFromShortestPathList();
     }
     else
        System.out.println("NO PATH!!!");
  }
```

// Υλοποίηση του αλγορίθμου Floyd Warsall

```
  public void runFloydWarsallAlgorithm(int start, int end) throws SQLException
  {
     //System.out.println("\n------------------------------------------------------------");
     //System.out.println("Shortest path from " + start + " to " + end + " :\n");
     FloydWarshallShortestPaths floydWarsallPath = new
FloydWarshallShortestPaths(graph);
     List shortest_path =  floydWarsallPath.getShortestPath(start, end).getEdgeList();

     if (shortest_path != null) {
       // System.out.println("FloydWarsall Path       : " + shortest_path);
       // System.out.println("------------------------------------------------------------");

        parseNodesFromShortertPath(shortest_path, end);
        getNodesLongIdFromShortestPathList();
     }
     else
        System.out.println("NO PATH!!!");
  }

  public SimpleDirectedWeightedGraph setGraph()
  {
     return graph;
  }
```

// Δημιουργία λίστας με τα μικρά id των κόμβων του ελάχιστου
// μονοπατιού

```
 void parseNodesFromShortertPath(List shortest_path, int end)
  {
     int i;
     String item;
     StringTokenizer st;

     shortestPathList = new ArrayList();

     i=0;
     while(i<shortest_path.size()) {
        st = new StringTokenizer(shortest_path.get(i).toString(), "( ");

        item = st.nextToken();

        shortestPathList.add(Integer.parseInt(item));

        i++;
     }
     shortestPathList.add(end);
  }
```

// Παίρνουμε τα μεγάλα id από τη λίστα των μικρών id τα οποία

// είναι απαραίτητα για τη δημιουργία του τελικού αρχείου

```
   void getNodesLongIdFromShortestPathList() throws SQLException
   {
      int i;

      nodesLongIdList = new ArrayList();

      Statement stmt = null;

      String query = "SELECT lnode_id FROM nodes WHERE ";

      for(i=0;i<shortestPathList.size();i++) {
         if (i==shortestPathList.size()-1)
            query = query + "node_id=" + shortestPathList.get(i).toString();
         else
            query = query + "node_id=" + shortestPathList.get(i).toString() + " || ";
      }

      try {
         stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(query);

         while (rs.next()) {
            nodesLongIdList.add(rs.getLong(1));
         }
      }
      catch (SQLException e) {
      }
      finally {
         if (stmt != null) {
            stmt.close();
         }
      }
   }
```

// Δημιουργία του τελικού αρχείου που εμφανίζει το ελάχιστο
// μονοπάτι

```
   public void createNewMapFile(String mapPath, String algorithm) throws
IOException
   {
      FileReader osmFile = new FileReader(mapPath);
      BufferedReader dataBuffer = new BufferedReader(osmFile);
      PrintWriter writer = new PrintWriter(algorithm+".osm", "UTF-8");

      StringTokenizer st;
      int i, j;
      boolean flag;
```

```
boolean inId;

String line = null;
String word = null;

i=1;
line=dataBuffer.readLine();

while (line != null) {
   inId = false;
   flag = false;
   st = new StringTokenizer(line, " '=\"");
   word = st.nextToken();

   // read nodes
   if(word.equals("<node")) {
      while(!word.equals("/>") && !word.equals(">")) {
         if(word.equals("id")) {
            inId = true;
            word = st.nextToken();

            flag = false;
            j=0;
            while(j<nodesLongIdList.size() && !flag){
               if(nodesLongIdList.get(j).toString().equals(word))
                  flag = true;
               j++;
            }

            if(flag)
               writer.println(line);
         }

         word = st.nextToken();

         if(word.equals(">") && flag) {
            line = dataBuffer.readLine();
            while(!line.equals(" </node>")) {
               writer.println(line);
               line = dataBuffer.readLine();
            }
            writer.println(line);
         }

         if(word.equals(">") && !flag) {
            line = dataBuffer.readLine();
            while(!line.equals(" </node>")) {
               line = dataBuffer.readLine();
            }
         }
```

```
            }
        }

        // read ways
        if(word.equals("<way")) {
            inId = false;
            st = new StringTokenizer(line, " '=\"");
            word = st.nextToken();

            writer.println(line);

            while(!word.equals("</way>")) {
                flag = false;
                if(word.equals("<nd")) {
                    st.nextToken();
                    word = st.nextToken();
                    inId = true;
                    flag = false;
                    j=0;
                    while(j<nodesLongIdList.size() && !flag){
                        if(nodesLongIdList.get(j).toString().equals(word)) {
                            flag = true;
                        }
                        j++;
                    }

                    if(flag)
                        writer.println(line);
                }

                line=dataBuffer.readLine();
                st = new StringTokenizer(line, " '=\"");
                word = st.nextToken();

                if(word.equals("</way>"))
                    writer.println(word);
            }
        }

        if (!inId)
            writer.println(line);

        line = dataBuffer.readLine();
    }

    dataBuffer.close();
    writer.close();
}

ArrayList getNodesLongIdList()
```

```
    {
        return nodesLongIdList;
    }
```

// Παίρνουμε το μικρό id ενός κόμβου από το όνομα του δρόμου
// το οποίο είναι απαραίτητο για να τρέξουμε του αλγόριθμους

```
 public int getNodeFromWayName(String name) throws SQLException
    {
        int flag;

        Statement stmt = null;

        String query = "SELECT node FROM name_node WHERE name='" + name
+"'";
        flag=-1;

        try {
            stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query);

            while (rs.next()) {
                flag = rs.getInt(1);
            }
        }
        catch (SQLException e) {
        }
        finally {
            if (stmt != null) {
                stmt.close();
            }
        }

        return flag;
    }
```

4. Κλάση StartPage

// Κλήση μεθόδου για τη δημιουργία της βάσης δεδομένων και
// πινάκων

```
private void createDatabaseButtonActionPerformed(java.awt.event.ActionEvent evt)
{
        xampp.createDatabase(dbName, username, password);
        xampp.createTables(dbName, username, password);
        JOptionPane.showMessageDialog(null, "Η βάση δημιουργήθηκε.", "",
JOptionPane.INFORMATION_MESSAGE);
```

```
    }

        // Κλήση μεθόδου για τη διαγραφή βάσης δεδομένων

 private void dropDatabaseButtonActionPerformed(java.awt.event.ActionEvent evt) {
        xampp.dropDatabase(dbName, username, password);
        JOptionPane.showMessageDialog(null, "Η βάση διαγράφηκε.", "",
JOptionPane.INFORMATION_MESSAGE);
    }

        // Κλήση μεθόδου για το γέμισμα της βάση δεδομένων από το αρχείο
        // του χάρτη (filePath)

    private void populateDatabaseButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            OpenStreetMap map = new OpenStreetMap();

            populateDatabaseButton.setText("Περιμένετε...");
            map.readOSMFile(filePath, dbName, username, password);
            JOptionPane.showMessageDialog(null, "Εισαγωγή δεδομένων
ολοκληρώθηκε.", "", JOptionPane.INFORMATION_MESSAGE);
            populateDatabaseButton.setText("ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ");
        } catch (IOException | SQLException | ClassNotFoundException ex) {
            Logger.getLogger(StartPage.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

        // Κλήση μεθόδου για τη δημιουργία γράφου από τη βάση
        // δεδομένων

   private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {
        Connection conn;

        try {
            conn = xampp.establishXamppConnection(dbName, username, password);
            brain.retrieveFromDatabase(conn);
            brain.CreateDirectedGraph();

            isRunPressed = true;

            runDijkstraAlgorithmButton.setEnabled(isRunPressed);
            runBellmanFordAlgorithmButton.setEnabled(isRunPressed);
            runFloydWarsallAlgorithmButton.setEnabled(isRunPressed);
        } catch (SQLException | ClassNotFoundException ex) {
            Logger.getLogger(StartPage.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

// Εκτέλεση του αλγορίθμου Dijkstra

```
 private void
runDijkstraAlgorithmButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int s,e;

        start = startPositonText.getText();
        end  = endPositonText.getText();

        try {
            s = brain.getNodeFromWayName(start);
            e = brain.getNodeFromWayName(end);

            if(s!=-1 && e!=-1) {
                brain.runDijkstraAlgorithm(s, e);
                brain.createNewMapFile(filePath, "Dijkstra");
                JOptionPane.showMessageDialog(null, "Το αρχείο δημιουργήθηκε.",
"Dijkstra", JOptionPane.INFORMATION_MESSAGE);
            }
            else {
                JOptionPane.showMessageDialog(null, "Λάθος εισαγωγή δεδομένων.",
"Dijkstra", JOptionPane.ERROR_MESSAGE);
            }
        } catch (SQLException | IOException ex) {
            Logger.getLogger(StartPage.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

// Εκτέλεση του αλγορίθμου Bellman-Ford

```
 private void
runBellmanFordAlgorithmButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int s,e;

        start = startPositonText.getText();
        end  = endPositonText.getText();

        try {
            s = brain.getNodeFromWayName(start);
            e = brain.getNodeFromWayName(end);

            if(s!=-1 && e!=-1) {
                brain.runBellmanFordAlgorithm(s, e);
                brain.createNewMapFile(filePath, "BellmanFord");
                JOptionPane.showMessageDialog(null, "Το αρχείο δημιουργήθηκε.",
"Bellman-Ford", JOptionPane.INFORMATION_MESSAGE);
            }
            else {
```

```
            JOptionPane.showMessageDialog(null, "Λάθος εισαγωγή δεδομένων.",
"Bellman-Ford", JOptionPane.ERROR_MESSAGE);
          }
       } catch (SQLException | IOException ex) {
          Logger.getLogger(StartPage.class.getName()).log(Level.SEVERE, null, ex);
       }
    }
```

// Εκτέλεση του αλγορίθμου Floyd Warsall

```
private void
runFloydWarsalltAlgorithmButtonActionPerformed(java.awt.event.ActionEvent evt)
{
       int s, e;

       start = startPositonText.getText();
       end  = endPositonText.getText();

       try {
          s = brain.getNodeFromWayName(start);
          e = brain.getNodeFromWayName(end);

          if(s!=-1 && e!=-1) {
             brain.runFloydWarsallAlgorithm(s, e);
             brain.createNewMapFile(filePath, "FloydWarsall");
             JOptionPane.showMessageDialog(null, "Το αρχείο δημιουργήθηκε.",
"FloydWarsall", JOptionPane.INFORMATION_MESSAGE);
          }
          else {
             JOptionPane.showMessageDialog(null, "Λάθος εισαγωγή δεδομένων.",
"FloydWarsall", JOptionPane.ERROR_MESSAGE);
          }
       } catch (SQLException | IOException ex) {
          Logger.getLogger(StartPage.class.getName()).log(Level.SEVERE, null, ex);
       }
    }
```