ΑΝΑΦΟΡΑ 1ης ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗΣ ΕΡΓΑΣΙΑΣ ΣΤΟ ΜΑΘΗΜΑ ΤΗΣ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

Κωνσταντίνος Κασφίκης
Αλέξανδρος Τερζής

Μας ζητήθηκε η δημιουργία προγράμματος το οποίο βάσει ενός text αρχείου θα δημιουργεί έναν γράφο. Στον γράφο αυτόν πρέπει να υλοποιηθούν συγκεκριμένες μέθοδοι αναζήτησης με σκοπό την εύρεση του βέλτιστου μονοπατιού από έναν αρχικό κόμβο σε ένα κόμβο στόχο.

ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΑΡΧΙΚΟΠΟΙΗΣΗ ΓΡΑΦΟΥ

Αρχικά αναπτύσσουμε μια κλάση InitFile η οποία αντλεί τα στοιχεία από το δοσμένο αρχείο δημιουργώντας τους κόμβους, τις πλευρές και τελικά τον γράφο. Το αρχείο περιέχει επιπλέον πληροφορίες σχετικές με το κόστος διάνυσης της κάθε πλευράς του γράφου, αλλά και πληροφορίες τόσο για την εκτιμώμενη όσο και την πραγματική κίνηση που θα επικρατεί για την εκάστοτε ημέρα σε κάθε πλευρά (δρόμο) του γράφου.

Στην InitFile:

Καλείται η ReadFile() η οποία θα διαβάσει το σύνολο του κειμένου που περιέχεται μέσα στο αρχείο με την πληροφορία για την δημιουργία του γράφου.

Καλείται η συνάρτηση Initialization () η οποία θα αποθηκεύσει σε κατάλληλες μεταβλητές τύπου string τα δεδομένα που εμπεριέχονται στο αρχείο ως προς τον γράφο ώστε να είναι εφικτή η επεξεργασία τους.

Στην συνέχεια καλείται η FillGraph(). Προκειμένου να λειτουργήσει είναι απαραίτητη η δημιουργία εκ των προτέρων των κλάσεων που θα συνθέσουν τον γράφο. Οι κλάσεις αυτές είναι η Vertex, Edge και Graph.

Στην FillGraph() παίρνουμε σειριακά τις μεταβλητές τύπου string, που δημιουργήθηκαν κατά το initialization και αντλούμε τα στοιχεία από αυτές.
Οι μεταβλητές αυτές είναι: η RoadInfo, η οποία περιέχει όλους τους δρόμους του γράφου και τους κόμβους από τους οποίους αποτελούνται, η TrafficPredictionInfo η οποία περιέχει για την εκάστοτε ημέρα την πρόβλεψη της κίνησης για κάθε δρόμο και η TrafficActualInfo η οποία περιέχει για την εκάστοτε ημέρα την πραγματική κίνηση για κάθε δρόμο.

Ενώ τα παραπάνω στοιχεία στις μεταβλητές βρίσκονται συγκεντρωτικά για το σύνολο των δρόμων η FillGraph() με την χρήση βρόγχων αρχικοποιεί έναν – έναν τους κατάλληλους κόμβους τύπου Vertex, τις κατάλληλες πλευρές τύπου Edge ενώ οι συνδέσεις που προκύπτουν μεταξύ των παραπάνω, αποτελούν τελικά τον γράφο. Τα παραπάνω δομούνται με τις πληροφορίες που προέρχονται από την μεταβλητή RoadInfo. Όταν τελείωσει η δημιουργία του συνόλου των δρόμων, για κάθε έναν από αυτούς, προστίθεται η πραγματική και η εκτιμώμενη κίνηση για κάθε ημέρα

Η FillGraph πριν αρχίσει να δημιουργεί δρόμους καλεί την CleanDublicates η οποία με όρισμα το σύνολο των δρόμων που περιέχοναι στον γράφο (ως ένας πίνακας από strings) βρίσκει πεδία από τον πίνακα που αποτελούνται από τους ίδιους κόμβους αλλά έχουν διαφορετικό κόστος και διαγράφει όλους τους σχετικούς δρόμους με κόστος μεγαλύτερο του μικρότερου.
Ετσι στην μεταβλητή RoadInfo μετά την εκτέλεση της CleanDoublicates υπάρχουν το πολύ μία πλευρά για κάθε ζευγάρι κόμβων η οποια μάλιστα έχει το χαμηλότερο κόστος. Η παραπάνω διαδικασία ονομάζεται “κλάδεμα” του γράφου.

Τέλος στην InitFile περιέχεται και η συνάρτηση WriteOutputFile η οποία θα δημιουργήσει το αρχείο με τα τελικά αποτελέσματα από τις μεθόδους αναζήτησης.

Όπως προαναφέρθηκε ο γράφος υλοποιείται με τη χρήση της κλάσης Graph. Τα διακριτά σημεία του γράφου, δηλαδή οι πλευρές και οι κόμβοι εμπεριέχονται στο σύνολό τους στην Graph. Κάθε κόμβος χαρακτηρίζεται από ένα όνομα (label) και από μια λίστα (neighborhood) που περιέχει όλες τις πλευρές που ακουμπούν στον κόμβο. Κάθε πλευρά χαρακτηρίζεται από ένα όνομα (name) , από 2 κόμβους που την ορίζουν και από το κόστος της πλευράς.

Για την υλοποίηση των αναζητήσεων του βέλτιστου μονοπατιού, δημιουργήθηκε μία κλάση η οποία ονομάζερται Search και περιέχει μεθόδους GetRoads(), η οποία με όρισμα ένα σύνολο από κόμβους (λίστα τύπου Vertex) θα επιστέψει λίστα με τους δυνατούς δρόμους, η GetCost() η οποία με όρισμα μία λίστα με δρόμους (λίστα τύπου edge) θα υπολογίσει το κόστος για την διάνυση των δρόμων αυτών. Υπάρχουν επίσης οι συναρτήσεις PrintPaths() και PrintRoads() οι οποίες θα οδηγήσουν στην εκτύπωση των κόμβων και των δρόμων αντίστοιχα που περιέχονται σε μία λίστα όρισμα. Σημαντική συνάρτηση αποτελεί η CalculateCost η οποία επιστρέφει το κόστος μιας πλευράς για μια συγκεκριμένη μέρα βάσει της κίνησης που έχει.

Τέλος χρησιμοποιούνται για την χρονομέτρηση του πραγματικού χρόνου εκτέλεσης του εκάστοτε αλγόριθμου αναζήτησης οι συναρτήσεις SetTimerStart(), setTimerEnd, και GetTime() η λειτουργία των οποίων περιγράφεται από το όνομά τους. Ο χρόνος εκτέλεσης μετριέται σε nano seconds. Η κλάση Search αποτελεί υπερκλάση όλων των κλάσεων των μεθόδων αναζήτησης. Οι μέθοδοι, δηλαδή, που εμπεριέχονται σε αυτήν είναι προσβάσιμες και από τις υποκλάσεις της.

ΜΕΘΟΔΟΙ ΑΝΑΖΗΤΗΣΗΣ

Μας ζητήθηκε η εκπόνηση τριών μεθόδων αναζήτησης:

1. **Blind ή Uninformed Search**
2. **Informed Search**
3. **Online Search**

1) Για την υλοποίηση της απληροφόρητης αναζήτησης χρησιμοποιήθηκε ο αλγόριθμος **Dijkstra.**

Ορίζονται οι εξής λίστες:
Edges : το σύνολο των πλευρών που περιέχει ο γράφος.
settledNodes : οι κόμβοι που έχουν επεξεργαστεί
unSettledNodes : οι κόμβοι που δεν έχουν επεξεργαστεί
predecessors : ο κόμβος-πατέρας του κάθε κόμβου
distance : περιέχει για κάθε κόμβο την απόσταση του από τον αρχικό κόμβο.
Θέτουμε ως τον τρέχον κόμβο τον αρχικό.
Εξετάζουμε, για τον τρέχον κόμβο, όλους τους μη επεξεργασμένους γείτονες του και υπολογίζεται η συνολική απόσταση του καθενός από τον αρχικό κόμβο. Αποθηκεύουμε τις αποστάσεις στην λίστα distance εφόσον η υπάρχουσα τιμή στην λίστα για τον εκάστοτε κόμβο είναι μηδενική ή μεγαλύτερη από αυτην που υπολογίστηκε και μόνο τότε ο τρέχον κόμβος προστίθεται στην λίστα των predecessors για τον εκάστοτε κόμβο-γείτονα.
Όταν τελειώσει η εξέταση όλων των κόμβων-γειτόνων του τρέχοντος κόμβου, τότε ο τρέχον κόμβος προστίθεται στην λίστα settledNodes και ο αλγόριθμος δεν θα τον ξαναεπεξεργαστεί στο μέλλον,καθώς η απόσταση από τον αρχίκο κόμβο είναι η ελάχιστη.
Η ροή του αλγόριθμου συνεχίζεται θέτωντας ως νέο τρέχον κόμβο τον κόμβο με την μικρότερη ετικέτα απόστασης και η διαδικασία επαναλαμβάνεται μέχρι όλοι οι κόμβοι να είναι επεξεργασμένοι (να ανήκουν δηλαδή στην λίστα settledNodes).
Όταν όλοι οι κόμβοι έχουν σημειωθεί ως επεξεργασμένοι, ξεκινώντας από τον κομβο-προορισμό της αναζήτησης εκτυπώνουμε τον κόμβο που αναγράφεται στην λίστα predecessors, επαναλαμβάνουμε μέχρι να συναντήσουμε null.

Ο αλγόριθμος του Dijkstra είναι άπληστος. Δηλαδή, σε κάθε βήμα επιλέγει την τοπικά βέλτιστη λύση, ώσπου στο τελευταίο βήμα συνθέτει μια συνολικά βέλτιστη λύση. Η αποδοσή του, ωστόσο, θεωρητικά θα είναι χαμηλότερη από του αλγόριθμους που ακολουθούν, καθώς πρέπει να λάβει υπόψιν του όλους τους κόμβους και να τους επεξεργαστεί.

2) Για την υλοποίηση της πληροφορημένης αναζήτησης μας ζητήθηκε συγκεκριμένα η εύρεση αποτελέσματος με τη χρήση του αλγόριθμου **IDA* (Iterative Deepening A*).**

Η λειτουργία του IDA* μπορεί να συγκριθεί με την λειτουργία του αλγόριθμου Depth First με τη διαφορά ότι υπάρχει ένα όριο F(n) = g(n)+h(n) το οποίο άπαξ και ξεπεραστεί αποτρέπει την περαιτέρω εκβάθυνση, επιστρέφοντας στον κόμβο-πατερα.

Στην συνάρτηση F(n) :

n είναι ο κόμβος στον οποίο βρισκόμαστε

g(n) το μέχρι τώρα κόστος διαδρομής από την ρίζα

h(n) η ευριστική συνάρτηση.

Η ευριστική συνάρτηση προκύπτει από πληροφορίες που έχουμε για το συγκεκριμένο πρόβλημα και βοηθάει στην γρηγορότερη αναζήτηση,αποτελώντας μία εκτίμηση για το επιπλέον κόστος που θα διανύσουμε προκειμένου να φτάσουμε στον στόχο μας από τον κόμβο n . Στην προκειμένη περιπτωση ως ευριστική συνάρτηση τέθηκε, αν και αρκετά απλή, το μικρότερο δυνατό κόστος μεταξύ δύο σημείων του γράφου (MIN_COST), το οποίο υπολογίζεται κατά την εισαγωγή των στοιχείων στον γράφο.

Η διαδικασία ξεκινά με την startSearch() η οποία σε έναν βρόγχο καλεί συνεχόμενα την IDA* (που ουσιαστικά είναι breadth first μέχρι κάποιο όριο) και αυξάνει σταδιακά το όριο F(n) , προστίθωντας στην προηγούμενη τιμή του το μικρότερο κόστος πλευράς από τις πλευρές που ξεπέρασαν την F(n) στην προηγούμενη εκτέλεση της IDA*. Ο βρόγχος λειτουργεί μέχρι να βρεθεί τουλάχιστον ένα μονοπάτι που να ενώνει τους δύο ζητούμενους κόμβους.

Η απόδοση της αναζήτησης IDA* εξαρτάται άμεσα από τον F(n) και συγκριμένα απο την ευριστική. Προκειμένου ο IDA* να επιστρέψει το βέλτιστο μονοπάτι πρέπει η ευριστική να είναι admissible, δηλαδή η εκτίμηση της ευριστικής να ΜΗΝ είναι μεγαλύτερη από το πραγματικό κοστος. Όσο πιο κοντά η τιμή της ευριστικής στο πραγματικό κόστος διάσχυσης από τον κόμβο n στον κόμβο–στόχο , τόσο πιο γρήγορη είναι η εύρεση του αποτελέσματος.

Η IDA* για να υπολογίσει το κόστος κάθε πλευράς λαμβάνει υπόψιν της την πραγματική κίνηση που επικρατεί στους δρόμους.

3) Τέλος, όσον αφορά την Online αναζήτηση, χρησιμοποιήθηκε ο αλγόριθμος **LRTA*.** Στην μέθοδο αυτήν, σε αντίθεση με τις προηγούμενες , το κόστος μιας πλευράς γίνεται γνωστό μόνο μετα την διάσχιση του. Ο LRTA* λαμβάνει υπόψιν του συνθήκες που επικρατούσαν σε προηγούμενες εκτελέσεις του ίδιου προβλήματος,

και βάσει αυτών των δεδομένων αποφασίζει για το μονοπάτι που θα ακολουθήσει . Η υλοποίηση του είναι παρόμοια με του IDA* με την διαφορά ότι γίνεται διαφορετική κοστολόγιση των πλευρών με την χρήση της CalculateCost(). Συγκεκριμένα , κάθε φορά που εξετάζουμε

τους κόμβους-γείτονες ενός κόμβου , υπολογίζουμε το κόστος της εκάστοτε πλευράς με βάσει την μέρα που βρισκόμαστε. Δηλαδή αντλούμε το πραγματικό κόστος της εκάστοτε πλευράς από τις πραγματική κίνηση που επικρατούσε τις προηγούμενες μέρες και την εκτιμώμενη κίνηση για τις επόμενες . Βάσει των παραπάνω, ανάλογα με το πλήθος των μερών που έχουν heavy traffic (count1) και το πλήθος των μερών που έχουν low traffic (count2) υποθέτουμε ότι η συγκεκριμένη πλευρά (δρόμος) του γράφου θα έχει τιμή :

Low : αν count1<count2
Heavy: αν count1>count2
Normal : αν count1=count2

Έτσι , ο επόμενος κόμβος που θα ακολουθηθεί βασίζεται τόσο σε υπαρκτά δεδομένα (από τις κινήσεις των προηγούμενων μερών) καθώς και από εκτιμήσεις των επόμενων μερών. Ανάλογα με το πόσες φορές έχει εκτελεστεί ο LRTA* για το συγκεκριμένο πρόβλημα , τόσο αυξάνεται η αποδοτικότητα του.

ΣΥΜΠΕΡΑΣΜΑΤΑ ΩΣ ΠΡΟΣ ΤΗΝ ΑΠΟΔΟΣΗ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ

Τόσο η πρώτη μέθοδος αναζήτησης, όσο και η δεύτερη, εφόσον βασίζονται στην πραγματική κίνηση για κάθε δρόμο θα επιστρέψουν το βέλτιστο μονοπάτι σε κάθε περίπτωση, ενώ η τρίτη μέθοδος αναζήτησης μπορεί να επιστρέψει μονοπάτι το οποίο να μην είναι βέλτιστο, καθώς βασίζεται εν μέρει σε εκτιμήσεις. Στα αποτελέσματα του προγράμματος φαίνεται ότι το μονοπάτι που προτείνεται από τον LRTA*, ενώ τις πρώτες μέρες δεν είναι το βέλτιστο (αν και το κόστος του προτεινόμενου μονοπατιού δεν είναι πολύ μεγαλύτερο από αυτό του βέλτιστου), τις τελευταίες μέρες προτείνεται όντως η βέλτιστη διαδρομή που δίνεται και από τις άλλες δύο μεθόδους αναζήτησης.
Από τις μετρήσεις πραγματικού χρόνου εκτέλεσης, ο Dijkstra φαίνεται να είναι ο πιο γρήγορος αλγόριθμος, ωστόσο, αυτό συμβαίνει λόγω της μικρής τιμής της ευριστικής συνάρτησης που χρησιμοποιείται στον IDA*. Για τιμή ευριστικής κοντινή στο πραγματικό κόστος η IDA* θα πιο αποδοτική.

```
#################### SEARCH FOR DAY :79 ##############################
============ PERFORMING UNINFORMED SEARCH ============================
 ---------TAKING IN ACCCOUNT PREDICTED/ACTUAL TRAFFIC------------------
The search was completed in : 316097 nano secs
Extended 4 nodes in order to reach goal
The nodes of the optimal path are:10Node4e ==> 10Node5e ==> 11Node8e ==> 11Node10e
The roads of the optimal path are:Road36e ==> Road144e ==> Road101e
with cost :105
 --------------WITHOUT PREDICTED/ACTUAL TRAFFIC------------------------
The search was completed in : 271612 nano secs
Extended 4 nodes in order to reach goal
The nodes of the optimal path are:10Node4e ==> 10Node5e ==> 11Node8e ==> 11Node10e
The roads of the optimal path are:Road36e ==> Road144e ==> Road101e
with cost :101
======================================================================
========== PERFORMING INFORMED SEARCH ================================
 ---------TAKING IN ACCCOUNT PREDICTED/ACTUAL TRAFFIC------------------
The search was completed in : 718596 nano secs
Informed Search :ATTEMPT 1 using frontier :2
Informed Search :ATTEMPT 2 using frontier :14
Informed Search :ATTEMPT 3 using frontier :35
Informed Search :ATTEMPT 4 using frontier :77
Extended 5 nodes in order to reach goal
The nodes of the optimal path are:10Node4e ==> 10Node6e ==> 11Node5e ==> 11Node3e ==> 11Node10e
The roads of the optimal path are:Road37e ==> Road145e ==> Road86e ==> Road89e
with cost :97
 --------------WITHOUT PREDICTED/ACTUAL TRAFFIC------------------------
The search was completed in : 926904 nano secs
Informed Search :ATTEMPT 1 using frontier :2
Informed Search :ATTEMPT 2 using frontier :15
Informed Search :ATTEMPT 3 using frontier :37
Informed Search :ATTEMPT 4 using frontier :81
Extended 4 nodes in order to reach goal
The nodes of the optimal path are:10Node4e ==> 10Node5e ==> 11Node8e ==> 11Node10e
The roads of the optimal path are:Road36e ==> Road144e ==> Road101e
with cost :101
======================================================================
========== PERFORMING ONLINE SEARCH ==================================
 ---------TAKING IN ACCCOUNT PREDICTED/ACTUAL TRAFFIC------------------
The search was completed in : 11052692 nano secs
Informed Search :ATTEMPT 1 using frontier :2
Informed Search :ATTEMPT 2 using frontier :16
Informed Search :ATTEMPT 3 using frontier :40
Informed Search :ATTEMPT 4 using frontier :87
Extended 4 nodes in order to reach goal
The nodes of the optimal path are:10Node4e ==> 10Node5e ==> 11Node8e ==> 11Node10e
The roads of the optimal path are:Road36e ==> Road144e ==> Road101e
with cost :95
 --------------WITHOUT PREDICTED/ACTUAL TRAFFIC------------------------
The search was completed in : 7206920 nano secs
Informed Search :ATTEMPT 1 using frontier :2
Informed Search :ATTEMPT 2 using frontier :15
Informed Search :ATTEMPT 3 using frontier :37
Informed Search :ATTEMPT 4 using frontier :81
Extended 4 nodes in order to reach goal
The nodes of the optimal path are:10Node4e ==> 10Node5e ==> 11Node8e ==> 11Node10e
The roads of the optimal path are:Road36e ==> Road144e ==> Road101e
with cost :101
======================================================================

######################################################################
AFTER 80 REPEATS THE AVERAGE COSTS FOR EACH ALGORITHM WITHOUT TRAFFIC ARE:
UNINFORMED SEARCH: 101
INFORMED SEARCH  : 101
ONLINE SEARCH    : 101
######################################################################

######################################################################
AFTER 80 REPEATS THE AVERAGE COSTS FOR EACH ALGORITHM WITH TRAFFIC ARE:
UNINFORMED SEARCH: 101
INFORMED SEARCH  : 99
ONLINE SEARCH    : 95
######################################################################
```