ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ (ΕΔΡΑ: ΣΠΑΡΤΗ)
Τ.Ε.Ι. ΠΕΛΟΠΟΝΝΗΣΟΥ

ΤΟ ΠΡΟΒΛΗΜΑ ΤΟΥ ΠΕΡΙΟΔΕΥΟΝΤΟΣ ΠΩΛΗΤΗ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Πολυχρόνης Γεώργιος

ΑΜ: 2012213

Επιβλέπων Καθηγητής
Αν. Καθ. Καραγιώργος Γρηγόρης

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάση επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δε μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

Όνομα και Επώνυμο Συγγραφέα (Με Κεφαλαία):

. . . . . .

Υπογραφή (Ολογράφως, χωρίς μονογραφή):

. . . . . .

Ημερομηνία (Ημέρα – Μήνας – Έτος):

. . . . . .

Περιεχόμενα

Κατάλογος Σχημάτων

Περίληψη

Στη παρούσα πτυχιακή θα ασχοληθούμε με το Πρόβλημα του Περιοδεύοντος Πωλητή. Θα αναφερθούμε γενικά στην έννοια του προβλήματος καθώς επίσης και στους δυνατούς τρόπους επίλυσής του, δίνοντας μεγαλύτερη σημασία στις προσεγγιστικές επιλύσεις.

Έπειτα, θα αναλύσουμε μία απο τις προσεγγιστικές λύσεις με χρήση της τριγωνικής ανισότητας η οποία προσφέρει επίλυση του σε πολυωνυμικό χρόνο καθώς και προσέγγιση του βέλτιστου αποτελέσματος έως και 2 φορές.

Στη συνέχεια θα αναλύσουμε αλγοριθμικά την προσεγγιστική λύση καθώς και τους αναγκαίους αλγόριθμους που χρειάζονται για την υλοποίηση της.

Στο τέλος, θα υλοποιήσουμε προγραμματιστικά την προσεγγιστική λύση στη γλώσσα C και θα αναφέρουμε τις διαφορές της υλοποίησης απο την αλγοριθμική ανάλυση.

Ευχαριστίες

Με τη παρούσα πτυχιακή εργασία ολοκληρώνω τις σπουδές μου στο Τμήμα Μηχανικών Πληροφορικής Τ.Ε. του Τ.Ε.Ι Πελοποννήσου με έδρα τη Σπάρτη. Θα ήθελα να ευχαριστήσω τους καθηγητές οι οποίοι μου πρόσφεραν τις γνώσεις τους όλα αυτά τα χρόνια επιτρέποντας μου να ολοκληρώσω τις σπουδές μου.
Φυσικά, ευχαριστώ ιδιαίτερα τον επιβλέπων της πτυχιακής μου και Αναπληρωτή Καθηγητή Γρηγόρη Καραγιώργο, ο οποίος με τις γνώσεις του και τη στήριξη του με καθοδήγησε στην υλοποίηση της πτυχιακής αυτής, καθώς και τον ευχαριστώ για όλες τις γνώσεις που μου προσέφερε στον προγραμματισμό και τους αλγορίθμους όλα αυτά τα χρόνια.
Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου, η οποία στάθηκε δίπλα μου όλα αυτά τα χρόνια καθώς επίσης και για τη στήριξη που μου προσφέρει για τις επιλογές μου.

1 Γενικά για τους αλγορίθμους

Για να μπορέσουμε να αναλύσουμε κατάλληλα το πρόβλημα του περιοδεύοντος πωλητή, καθώς και να δημιουργήσουμε επιτυχώς τον αλγόριθμο του προβλήματος αλλά και της προγραμματιστικής υλοποίησης του αλγορίθμου αυτού σε γλώσσα C, είναι απαραίτητο να γνωρίζουμε τους ορισμούς οι οποίοι θα μας διευκολύνουν στην επίλυση του.

1.1 Ορισμός

Τι είναι οι αλγόριθμοι;

Ο όρος αλγόριθμος χρησιμοποιείται για να ορίσει κάθε ορθή και καλά ορισμένη διαδικασία η οποία δέχεται μία ή περισσότερες τιμές ως είσοδο, τις επεξεργάζεται και παράγει μία ή περισσότερες τιμές ως έξοδο. Συνεπώς, μπορούμε να ορίσουμε έναν αλγόριθμο ως μια ακολουθία βημάτων ο οποίος παράγει αποτέλεσμα στην έξοδο μετά από κατάλληλη επεξεργασία της εισόδου που του δόθηκε.
Επιπλέον, μπορούμε να θεωρήσουμε έναν αλγόριθμο ως εργαλείο επίλυσης ενός ορισμένου υπολογιστικού προβλήματος, που με τη χρήση του ως υπολογιστική διαδικασία δέχεται είσοδο και παράγει την κατάλληλη έξοδο, μια σχέση η οποία είναι ορισμένη από το υπολογιστικό πρόβλημα.

Πως ορίζεται ένας ορθός αλγόριθμος;

Ένας αλγόριθμος για να χαρακτηριστεί ως ορθός, πρέπει για κάθε είσοδο που δέχεται να παράγει το κατάλληλο ορθό αποτέλεσμα στην έξοδο του σύμφωνα με το πρόβλημα το οποίο προσπαθεί να επιλύσει.

Χρόνος εκτέλεσης Αλγόριθμου

Υπάρχουν πολλές περιπτώσεις στις οποίες μπορεί να αναπτυχθούν παραπάνω από ένας αλγόριθμοι για την επίλυση ενός προβλήματος. Παραδείγματος χάριν, για την ταξινόμηση ενός πλήθους αριθμών υπάρχουν πολλοί αλγόριθμοι ταξινόμησης (όπως ταξινόμηση φυσαλίδας ή ταξινόμηση με συγχώνευση) οι οποίοι δέχονται ως είσοδο τους αριθμούς που θέλουμε να ταξινομήσουμε και ως αποτέλεσμα στην έξοδο τους λαμβάνουμε τους αριθμούς ταξινομημένους κατάλληλα.
Επειδή υπάρχουν αρκετοί αλγόριθμοι για την επίλυση ενός προβλήματος, χρησιμοποιούμε κριτήρια για να επιλέξουμε αυτόν που μας προσφέρει μεγαλύτερη ικανοποίηση στα κριτήρια αυτά σε σχέση με τους υπόλοιπους. Συνήθως όμως, ως βασικό κριτήριο θέτουμε τον χρόνο που χρειάζεται ο αλγόριθμος για να παράγει ένα αποτέλεσμα.

1.2 Ασυμπτωτικός Συμβολισμός

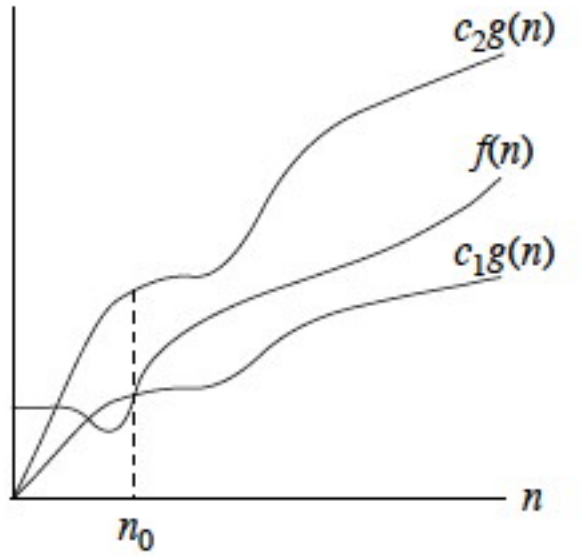
Όπως αναφέραμε, για την επιλογή του καλύτερου δυνατού αλγόριθμου για την επίλυση ενός προβλήματος χρησιμοποιούμε ως κριτήριο τον χρόνο που χρειάζεται για την εκτέλεση του. Υπάρχουν όμως περιπτώσεις για τις οποίες ο υπολογισμός του χρόνου εκτέλεσης ενός αλγορίθμου αποτελεί μια χρονοβόρα διαδικασία, ειδικά σε περιπτώσεις όπου η είσοδος του αλγορίθμου είναι μεγάλη.
Για να αποφύγουμε τον υπολογισμό αυτό, ασχολούμαστε κυρίως με τον αυξητικό χαρακτήρα του χρόνου εκτέλεσης του αλγορίθμου σε σχέση με το μέγεθος της εισόδου, οπότε και μελετάμε την ασυμπτωτική επίδοση του αλγορίθμου.
Συνήθως, ένας αλγόριθμος που έχει την καλύτερη δυνατή ασυμπτωτική επίδοση σε σχέση με τους υπόλοιπους αλγορίθμους επιλέγεται για την ταχύτερη επίλυση του προβλήματος, με μόνη πιθανή εξαίρεση την περίπτωση στην οποία η είσοδος είναι πολύ μικρή.

Υπάρχουν διάφοροι τύποι συμβολισμού τους οποίους χρησιμοποιούμε για να περιγράψουμε τον ασυμπτωτικό χρόνο εκτέλεσης ενός αλγορίθμου, με τους πιο βασικούς απο αυτούς να είναι οι O, Ω και Θ[7], οι οποίοι χρησιμοποιούνται για να περιγράψουν την συνάρτηση του χρόνου εκτέλεσης ενός αλγορίθμου στη χειρότερη περίπτωση T(n), ο οποίος ορίζεται συνήθως για ακέραιο μέγεθος εισόδου.
Παρακάτω, θα εξηγήσουμε τους συμβολισμούς O, Ω και Θ.

Συμβολισμός Θ

Όταν ο χρόνος χειρότερης περίπτωσης T(n) για κάποια συνάρτηση g(n) είναι ίσος με Θ(g(n)), το Θ(g(n)) συμβολίζει το σύνολο των συναρτήσεων για το οποιο ισχύει ότι:
Θ(g(n))={f(n) τετοιο ώστε υπάρχουν θετικές σταθερές c1,c2 και n0 τέτοιες ώστε $0 <= c1g(n) <= f(n) <= c2g(n)$ για κάθε $n > n0$ } .

Μια συνάρτηση f(n) ανήκει στο σύνολο Θ(g(n)) αν υπάρχουν σταθερές c1,c2 με τις οποίες η f(n) μπορεί να βρίσκεται ανάμεσα στις c1g(n) και c2g(n), απο μία τιμή του ν και πάνω.



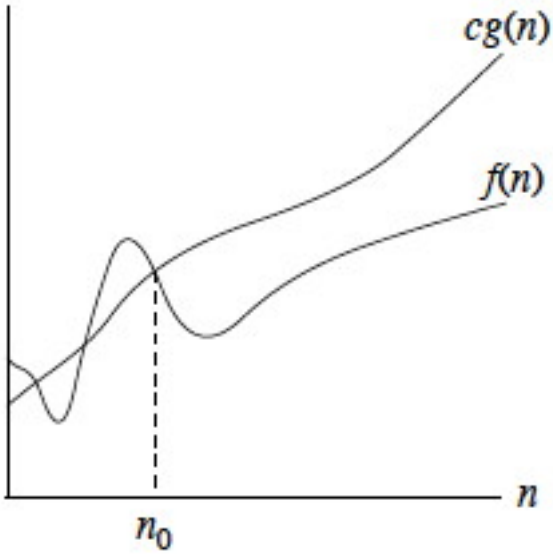
Σχήμα 1: Γράφημα Θ(f(n)).*University of Hawaii ICS311: Growth of Functions and Asymptotic Concepts*

Όπως μπορούμε να δούμε και στο σχήμα, η f(n) βρίσκεται ανάμεσα στις c1g(n) και c2g(n) για όλες τις τιμές του ν μεγαλύτερες του n0. Μπορούμε λοιπόν να πούμε οτι η f(n) είναι ασυμπτωτικά φραγμένη από επάνω και από κάτω.

Συμβολισμός Ο

Όταν έχουμε μόνο ένα ασυμπτωτικό φράγμα, το οποίο βρίσκεται στην άνω μεριά, το ονομάζουμε ασυμπτωτικό άνω φράγμα και χρησιμοποιούμε τον συμβολισμό Ο για να το ορίσουμε. Για μια δεδομένη συνάρτηση g(n), η έκφραση O(g(n)) δηλώνει το σύνολο των συναρτήσεων για το οποίο ισχύει ότι:
O(g(n))={f(n) τέτοιο ώστε υπάρχουν σταθερές c και n0 τέτοιες ώστε
$0 <= f(n) <= cg(n)$ για κάθε $n > n0$ }.

Ο συμβολισμός Ο χρησιμοποιείται για να ορίσουμε ένα άνω φράγμα για μια συνάρτηση με τη μορφή ενός σταθερού πολλαπλάσιου μίας άλλης.



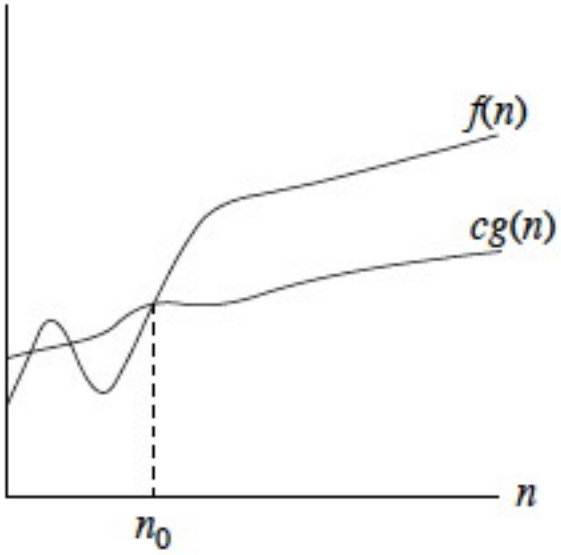
Σχήμα 2: Γράφημα O(f(n)).*University of Hawaii ICS311: Growth of Functions and Asymptotic Concepts*

Όπως βλέπουμε και στο σχήμα, η f(n) βρίσκεται κάτω από την cg(n) ή ισούται με αυτήν για κάθε $n > n0$.

Συμβολισμός Ω

Παρόμοια με τον συμβολισμό O, ο οποίος έχει ένα ασυμπτωτικό φράγμα, έτσι και ο συμβολισμός Ω έχει μονάχα ένα ασυμπτωτικό φράγμα το οποίο βρίσκεται όμως στη κάτω μεριά, το οποίο και ονομάζουμε ασυμπτωτικό κάτω φράγμα. Για μια δεδομένη συνάρτηση g(n), η έκφραση Ω(g(n)) δηλώνει το σύνολο των συναρτήσεων για το οποίο ισχύει ότι:
Ω(g(n))={f(n) τέτοιο ώστε υπάρχουν σταθερές c και n0 τέτοιες ώστε
$0 <= cg(n) <= f(n)$ για κάθε $n > n0$ } .

Ο συμβολισμός Ω χρησιμοποιείται για να ορίσουμε ένα κάτω φράγμα για μια συνάρτηση με τη μορφη ενός σταθερού πολλαπλάσιου μίας άλλης.



Σχήμα 3: Γράφημα Ω(f(n)).*University of Hawaii ICS311: Growth of Functions and Asymptotic Concepts*

Όπως βλέπουμε και στο σχήμα, η f(n) βρίσκεται πάνω από την cg(n) ή ισούται με αυτήν για κάθε $n > n0$.

1.3 NP-Πληρότητα

Υπάρχουν πολλά είδη προβλημάτων για τα οποία ψάχνουμε και βρίσκουμε κατάλληλους αλγορίθμους για να τα επιλύσουμε. Για αρκετά από αυτά τα προβλήματα, έχουν κατασκευαστεί αλγόριθμοι οι οποίοι τα επιλύουν σε πολυωνυμικό χρόνο. Γενικά, ένα πρόβλημα για το οποίο έχει οριστεί κατάλληλος αλγόριθμος με χρόνο εκτέλεσης χειρότερης περίπτωσης $T(n) = O(n^k)$ [8] για κάποια σταθερά κ θεωρείται ευεπίλυτο ή εύκολο πρόβλημα. Όμως, υπάρχουν κάποια προβλήματα για τα οποία δεν έχει βρεθεί κάποιος αλγόριθμος ή ο αλγόριθμος που έχει κατασκευαστεί επιλύει το πρόβλημα σε υπερπολυωνυμικό χρόνο. Τέτοιου είδους προβλήματα ονομάζονται δυσεπίλυτα ή δύσκολα.

Μέσα σε αυτά τα προβλήματα, υπάρχει και ένα σύνολο προβλημάτων, τα οποία ονομάζονται NP-πλήρη προβλήματα, για τα οποία δεν έχουμε ακόμη κάποιο χαρακτηρισμό για την επίλυση τους. Με άλλα λόγια, για κανένα από τα NP-πλήρη προβλήματα δεν έχουν βρεθεί ακόμη αλγόριθμοι οι οποίοι τα επιλύουν σε πολυωνυμικό χρόνο, χωρίς αυτό να αποδεικνύει ότι δεν υπάρχει κάποιος αλγόριθμος πολυωνυμικού χρόνου για αυτά. Ένα από τα προβλήματα αυτά είναι και το πρόβλημα του περιοδεύοντος πωλητή το οποίο μελετάμε.

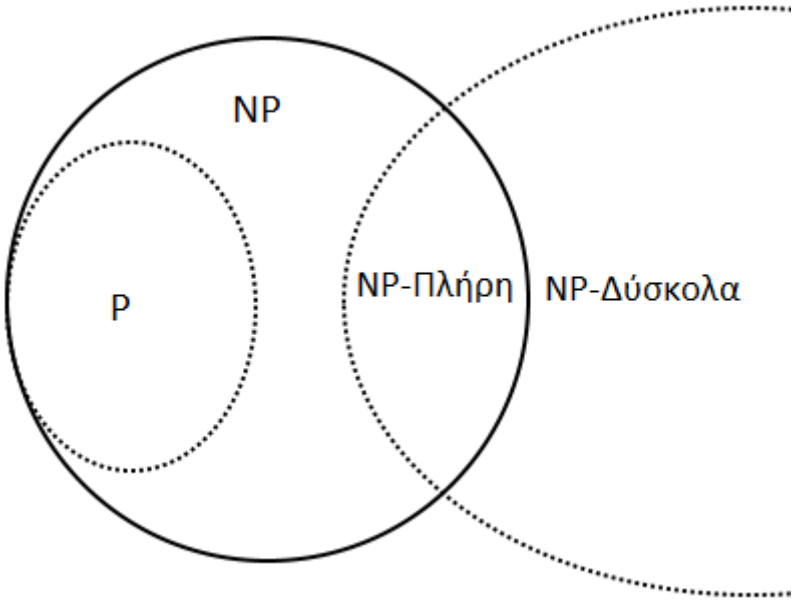
Το συγκεκριμένο ζήτημα ονομάζεται και $P \neq NP$ και τέθηκε για πρώτη φορά το 1971 από τον Stephen Cook [9] και έχει αναδειχθεί σε ένα από τα πιο σοβαρά ανοικτά προβλήματα της θεωρητικής επιστήμης των υπολογιστών, χωρίς μέχρι στιγμής να υπάρχει κάποια λύση.

Γενικά, έχουμε διαχωρίσει τα προβλήματα σε κλάσεις, οι οποίες χαρακτηρίζουν την πολυπλοκότητα τους. Οι κλάσεις πολυπλοκότητας που μας απασχολούν ονομάζονται P, NP και NP-πλήρη.

Η κλάση P αναφέρεται στα προβλήματα τα οποία υπάρχει αλγόριθμος πολυωνυμικού χρόνου και δε χρειάζεται να επαληθευτεί από μια μηχανή Turing η ορθότητα του αλγορίθμου. Αυτού του είδους προβλήματα θεωρούνται υποσύνολο της κλάσης NP, με ανοικτό ερώτημα να παραμένει αν η κλάση P αποτελεί γνήσιο υποσύνολο της κλάσης NP ή όχι.

Η κλάση NP περιέχει τα προβλήματα τα οποία μπορούμε να επαληθεύσουμε την επίλυση τους σε πολυωνυμικό χρόνο. Αναγκαίο για την επαλήθευση είναι να δοθεί ένα πιστοποιητικό μιας λύσης, ώστε να μπορέσει να επαληθευτεί η ορθότητα της, κάτι που απαιτεί πολυωνυμικό χρόνο σε σχέση με το μέγεθος της εισόδου.

Η κλάση NP-Πλήρη (αλλιώς NP-Complete) αποτελείται από τα δυσκολότερα προβλήματα της κλάσης NP. Θεωρητικά, για να ανήκει ένα πρόβλημα στη κλάση NP-πλήρη πρέπει να είναι τουλάχιστον το ίδιο δύσκολο με τα υπόλοιπα προβλήματα της κλάσης NP.
Χρησιμοποιώντας παράλληλα τον ορισμό του ζητήματος $P \neq NP$, αν για ένα πρόβλημα της κλάσης NP βρεθεί αλγόριθμος επίλυσης του με πολυωνυμικό χρόνο, τότε όλα τα προβλήματα της κλάσης NP λύνονται σε πολυωνυμικό χρόνο, κάτι που έπειτα από πολλή μελέτη δεν έχει βρεθεί, χωρίς φυσικά να σημαίνει ότι δεν υπάρχει.



Σχήμα 4: Διάγραμμα Euler για τις κλάσεις P, NP, NP-πλήρη και NP-Δύσκολα

1.4 Προσεγγιστικοί αλγόριθμοι

Πολλά από τα προβλήματα που εμφανίζουν πρακτικό ενδιαφέρον βρίσκονται στην κλάση πολυπλοκότητας NP-πλήρη, κάτι που τα καθιστά δυσεπίλυτα. Παράλληλα όμως, αυτά τα προβλήματα θεωρούνται σημαντικά, οπότε δεν είναι δυνατόν να τα παραβλέψουμε επειδή δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου για την επίλυση τους. Έτσι, επινοήθηκαν μέθοδοι οι οποίες παρακάμπτουν την NP-πληρότητα, ώστε να υπάρξει επίλυση των προβλημάτων αυτών.
Μία από τις μεθόδους αυτές ονομάζεται προσέγγιση και παράγει σχεδόν βέλτιστη λύση σε πολυωνυμικό χρόνο, κάτι που στη πράξη συνήθως θεωρείται επαρκής. Γενικά, οι αλγόριθμοι που προσφέρουν σχεδόν βέλτιστες λύσεις ονομάζονται προσεγγιστικοί αλγόριθμοι.
Οι προσεγγιστικοί αλγόριθμοι ορίζονται από ένα λόγο προσέγγισης, ο οποίος αναφέρει την επιτυχία του σε σχέση με την βέλτιστη λύση του προβλήματος. [10]
Ο λόγος προσέγγισης ορίζεται από το:

$$max\Big(\frac{C}{C^*}, \frac{C^*}{C}\Big) \leq \rho(n)$$

Ένας αλγόριθμος ο οποίος κατέχει λόγος προσέγγισης ρ(n) ονομάζεται και ρ(n)-προσεγγιστικός αλγόριθμος.
Οι παραπάνω ορισμοί ισχύουν και στα προβλήματα μεγιστοποίησης με:

$$0 < C \leq C^*$$

Όπου ο λόγος C*/C δίνει το παράγοντα κόστους μιας βέλτιστης λύσης υπερβαίνει την προσεγγιστική αλλα και σε προβλήματα ελαχιστοποίησης με:

$$0 < C^* \leq C$$

Όπου ο λόγος C/C* δίνει τον παράγοντα κατα το οποίο το κόστος της προσεγγιστικής λύσης υπερβαίνει αυτό της βέλτιστης.

2 Το Πρόβλημα του Περιοδεύοντος Πωλητή

Σε αυτό το κεφάλαιο θα εξηγήσουμε το πρόβλημα του περιοδεύοντος πωλητή και τους λόγους για τους οποίους μας ενδιαφέρει η επίλυση του. Επίσης, θα αναφέρουμε τους αλγόριθμους που έχουν δημιουργηθεί για την επίλυση του, καθώς και την πολυπλοκότητα τους.
Επίσης, θα αναλύσουμε την επίλυση του προβλήματος του περιοδεύοντος πωλητή χρησιμοποιώντας έναν προσεγγιστικό αλγόριθμο με τη χρήση της τριγωνικής ανισότητας.
Τέλος, θα αναφέρουμε παραδείγματα όπου χρησιμοποιείται η περιοδεία του περιοδεύοντος πωλητή στη πράξη και ποιά είναι τα οφέλη.

2.1 Ορισμός

Το πρόβλημα του περιοδεύοντος πωλητή, συνήθως αναφερόμενο με το ακρωνύμιο ΠΠΠ, αποτελεί ένα από τα πιο διαδεδομένα προβλήματα στην ιστορία της πληροφορικής.
Περιγραφικά, ένας περιοδεύων πωλητής προσπαθεί να πουλήσει την πραμάτεια του περνώντας απο όλες τις διαθέσιμες πόλεις μία και μόνο μία φορά, και να τελειώσει το ταξίδι του από τη πόλη που ξεκίνησε.
Πρόκειται για ένα πρόβλημα ελαχιστοποίησης, όπου ο πωλητής προσπαθεί να διανύσει τη λιγότερη δυνατή απόσταση, δηλαδή να κάνει μια διαδρομή με το ελάχιστο δυνατό κόστος, ώστε να επισκεφτεί όλες τις πόλεις και να επιστρέψει στην αρχική πόλη.

Για την εύρεση διαδρομών, το ΠΠΠ χρησιμοποιεί γραφήματα, τα οποία περιέχουν το σύνολο των πόλεων. Κάθε ακμή που συνδέει δύο πόλεις έχει ένα συγκεκριμένο κόστος, ώστε να μπορεί να υπολογιστεί το συνολικό κόστος. Ιδιαίτερο χαρακτηριστικό των συγκεκριμένων γραφημάτων είναι ότι αυτά είναι πλήρη, δηλαδή κάθε πόλη συνδέεται με όλες τις υπόλοιπες. Έτσι, μια τυπική γλώσσα που μπορεί να περιγράψει το πρόβλημα διάγνωσης είναι η εξής:

ΠΠΠ=$< G, c, k >$: $G = (V, E)$ είναι ένα πλήρες γράφημα, c είναι μία συνάρτηση απο το $VxV- > Z$, k ανήκει $\mathbb{Z}$, και το G έχει μία περιοδεία κόστους το πολύ ίσου με k. [9]

Το πρόβλημα του Περιοδεύοντος Πωλητή ανήκει στη κλάση NP-Πλήρη, κάτι που θα αποδείξουμε στην επίλυση του στο κεφάλαιο 2.2.

2.1.1 Ιστορικό Υπόβαθρο

Ιστορικά, το πρόβλημα του περιοδεύοντος πωλητή έλαβε μαθηματική υπόσταση τον 19ο αιώνα, απο τον μαθηματικό W.R. Hamilton και τον Thomas Kirkman. [4]

Η γενική φόρμα του ΠΠΠ εμφανίζεται να έχει μελετηθεί στη Βιέννη τη δεκαετία του 1930, όπου και ο Karl Menger ο οποίος ορίζει πλέον το πρόβλημα, παραθέτει τον βασικό αλγόριθμο επίλυσης του προβλήματος που στηρίζεται στην "ωμή βία" (brute-force), δηλαδή την εξαντλητική αναζήτηση καλύτερου αποτελέσματος με όλες τις πιθανές διαδρομές, για να παράξει ένα αποτέλεσμα. Για τον ίδιο λόγο, ο Karl Menger παρατήρησε τη μή-βελτιστότητα του.
Λίγο αργότερα, ο Hassler Whitney στο πανεπιστήμιο Princeton παρουσίασε για πρώτη φορά το όνομα Travelling Salesman Problem (το Πρόβλημα του Περιοδεύοντος Πωλητή).

Τις δεκαετίες 1950 και 1960, το πρόβλημα άρχισε να γίνεται διάσημο στους επιστημονικούς κύκλους της Ευρώπης και τις Αμερικής, όταν και η εταιρεία RAND στη Santa Monica προσέφερε χρηματικό έπαθλο για βήματα προς την επίλυση του προβλήματος. Οι George Dantzig, Delbert Ray Fulkerson και Selmer M. Johnson της ίδιας εταιρείας [3], παρουσίασαν το πρόβλημα ώς ένα γραμμικό πρόβλημα ακεραίων και πρότειναν μία λύση για ένα στιγμιότυπο 49 πόλεων, χωρίς όμως να δώσουν μία αλγοριθμική επίλυση για το ΠΠΠ.

Μέσα στις επόμενες δεκαετίες, το πρόβλημα μελετήθηκε απο επιστήμονες διαφόρων κλάδων όπως μαθηματικών, επιστήμης υπολογιστών και φυσικής. Στη δεκαετία 1960, δημιουργήθηκε μία διαφορετική προσέγγιση στο πρόβλημα, με τους ανθρώπους να ψάχνουν τη χειρότερη αλγοριθμική λύση, καταφέρνοντας έτσι να δημιουργηθούν χαμηλότερα όρια για αυτό. Αυτές οι λύσεις μπορούσαν έπειτα να χρησιμοποιηθούν για προσεγγίσεις με διακλαδώσεις, όπως η χρήση του Δένδρου Ελαχίστου Κόστους του γράφου και διπλασιασμό του κόστους.

Το 1972, ο Richard M. Karp [2] έδειξε ότι το πρόβλημα του Χαμιλτόνιου κύκλου ανήκει στη κατηγορία των NP-πλήρη προβλημάτων, δίνοντας έτσι την μαθηματική εξήγηση ότι το πρόβλημα του περιοδεύοντος πωλητή αποτελεί ένα απο τα προβλήματα που θεωρούνται NP-Δύσκολα, καθώς επίσης και την υπολογιστική δυσκολία για να βρεθεί μία βέλτιστη διαδρομή.

Μία μεγάλη πρόοδο σε αυτού του είδους προσέγγιση έκανε ο Νίκος Χριστοφίδης [11] για το χειρότερο αποτέλεσμα που μπορούσε να δώσει ένας αλγόριθμος. Το 1976, παρουσιάσε τον αλγόριθμό του, ο οποίος στη χειρότερη περίπτωση έδινε αποτέλεσμα το πολύ 1.5 φορές πάνω από το καλύτερο δυνατό αποτέλεσμα ενός στιγμιότυπου.

2.2 Επίλυση Προβλήματος Περιοδεύοντος Πωλητή (ΠΠΠ)

Όπως αναφέραμε και στον ορισμό του προβλήματος του περιοδεύοντος πωλητή, το ΠΠΠ ανήκει στα προβλήματα της κατηγορίας NP-πλήρη, κάτι που προκύπτει από την απόδειξη παρακάτω:

Απόδειξη [9]

Για να μπορέσουμε να αποδείξουμε ότι το ΠΠΠ ανήκει στη κλάση NP-πλήρη, πρέπει να αποδείξουμε αρχικά οτι ανήκει στη κλάση NP.
Οπότε, για ένα δεδομένο στιγμιότυπο του προβλήματος, χρησιμοποιούμε ως πιστοποιητικό την ακολουθία των ν κόμβων της ακολουθίας. Με τη χρήση ενός αλγορίθμου επαλήθευσης, ο οποίος ελέγχει αν κάθε κόμβος περιλαμβάνεται στην ακολουθία ακριβώς μία φορά, και έπειτα αθροίζει τα κόστη και ελέγχει εάν το άθροισμα είναι μικρότερο ή ίσο του k, επιβεβαιώνει την ορθότητα της λύσης.
Η συγκεκριμένη διαδικασία μπορεί να διεκπαιρεωθεί σε πολυωνυμικό χρόνο.
Για να αποδείξουμε ότι το ΠΠΠ είναι NP-δυσχερές, θα δείξουμε ότι ο Χαμιλτονιανός Κυκλος, είναι μικρότερος ή ίσος μιας περιοδείας ΠΠΠ. ($^*HC \leq p$ ΠΠΠ).
Έστω γράφος $G = (V, E)$ ένα στιγμιότυπο του προβλήματος Χαμιλτονιανός κύκλος. Κατασκευάζουμε ένα στιγμιότυπο του ΠΠΠ το οποίο έχει έναν πλήρη Γράφο $G' = (V, E')$ όπου $E' = \{(i, j) : i, j \in V \text{ και } i \neq j\}$, και ορίζουμε τη συνάρτηση κόστους c ως:

$$c(i,j) = \begin{cases} 0, & \epsilon\alpha\nu(i,j) \in E \\ 1, & \epsilon\alpha\nu(i,j) \notin E \end{cases}$$

Επίσης, επειδή ο γράφος G είναι ακατεύθυντος καθώς και δεν περιέχει ακμές οι οποίες καταλήγουν στον κόμβο όπου ξεκίνησαν. Επομένως $c(v, v) = 1$, για όλους τους κομβους $v \in V$.
Οπότε, το στιγμιότυπο του ΠΠΠ είναι $< G', c, 0 >$ και μπορεί να κατασκευαστεί σε πολυωνυμικό χρόνο.
Εν συνεχεία, θα δείξουμε ότι ο γράφος G έχει χαμιλτονιανό κύκλο εάν και μόνο εάν το γράφημα G έχει περιοδεία με κόστος το πολύ 0.
Ας υποθέσουμε ότι ο γράφος G έχει έναν χαμιλτονιανό κύκλο h. Κάθε ακμή του κύκλου h ανήκει στο σύνολο ακμών E, και συνεπώς έχει κόστος 0 στο G'.
Επομένως, ο κύκλος η αποτελεί περιοδεία για το G' με κόστος 0. Αντιστρόφως, ας υποθέσουμε ότι ο γράφος G' έχει μία περιοδεία h' με κόστος το πολύ 0.
Δεδομένου ότι όλες οι ακμές του E' έχουν κόστος 0 ή 1, το κόστος της περιοδείας h' είναι ακριβώς 0, και η κάθε ακμή της θα πρέπει να έχει κόστος 0. Επομένως, η h' περιλαμβάνει μονάχα ακμές του συνόλου E, απ' όπου έπεται ότι η h' αποτελεί χαμιλτονιανό κύκλο για το γράφημα G. Επομένως, το ΠΠΠ ανήκει στη κλάση NP-πλήρη.

---

*Η αναφορά των αρχικών HC παραπέμπουν στον Χαμιλτονιανό Κύκλο (Hamiltonian Cycle) ο οποίος θα αναλυθεί διεξοδικά στο 3ο κεφάλαιο.

Ο πιο άμεσος αλγόριθμος που χρησιμοποιήθηκε για την επίλυση του ΠΠΠ, όπως αναφέρθηκε και στο 2.1.1, έβρισκε μια λύση για στιγμιότυπο προβλήματος με υπολογισμό όλων των πιθανών περιοδειών, σε χρόνο εκτέλεσης χειρότερης περίπτωσης $T(n) = O(n!)$, κάτι που τον καθιστά υπερβολικά χρονοβόρο, ειδικά σε περιπτώσεις που το πλήθος των πόλεων είναι μεγάλο.
Υπήρξαν επίσης υλοποιήσεις επίλυσης του ΠΠΠ, όπως ο αλγόριθμος Held-Karp [1] ο οποίος έλυνε το πρόβλημα σε $T(n) = O(n^2 2^n)$, που αν και αποτελούσε εμφανώς μια βελτιστοποίηση σε σχέση με τον άμεσο αλγόριθμο, ήταν αρκετά χρονοβόρος και αυτός.
Μάλιστα, οι συγκεκριμένοι χρόνοι εκτέλεσης είναι δύσκολο να βελτιωθούν, ενώ είναι αβέβαιο για το εάν υπαρχει αλγόριθμος ο οποίος προσφέρει χρόνο επίλυσης $T(n) = O(1.999^n)$.[5]

Επειδή οι αλγόριθμοι που προσπαθούσαν να παράξουν ένα αποτέλεσμα χρησιμοποιώντας την αμεσότητα είχαν πολυπλοκότητα υπερπολυωνυμική, αναζητήθηκαν άλλοι τρόποι για την εύρεση ενός αποτελέσματος σε πολυωνυμικό χρόνο. Για να μπορέσει να υπάρξει ένας αλγόριθμος με πολυωνυμικό χρόνο, υπήρξε συμβιβασμός μεταξύ ακρίβειας και χρόνου, με αποτέλεσμα τη χρήση ευρυστικών/προσεγγιστικών αλγορίθμων, οι οποίοι με γρήγορες εκτελέσεις βρίσκουν αποτελέσματα κοντά στην βέλτιστη λύση.

2.2.1 Χρήση της Τριγωνικής Ανισότητας

Όπως περιγράψαμε και προηγουμένως, στο Πρόβλημα του Περιοδεύοντος Πωλητή δίνεται ένας μη-κατευθυνόμενος γράφος G=(V,E), όπου σε κάθε ακμή (u,v) που περιέχεται στο E αντιστοιχεί ένα μη-αρνητικό ακέραιο βάρος c(u,v).
Σε αυτό, ζητείται να βρεθεί μια περιοδεία, δηλαδή ένας χαμιλτονιανός κύκλος του γράφου Γ με ελάχιστο κόστος.
Ορίζουμε επίσης ως c(A) το συνολικό κόστος των ακμών του υποσυνόλου $A \subseteq E$ ως εξής:

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

Σε πρακτικό επίπεδο, η μετάβαση από ένα σημείο u σε ένα σημείο v αποτελεί πιο οικονομική λύση σε σχέση με τη μετάβαση απο u σε v μέσω ενός ενδιάμεσου σταθμού w. Ως διαφορετική διατύπωση, μπορούμε να πούμε ότι μια διαδρομή με περισσότερους σταθμούς, δεν είναι δυνατόν να έχει μικρότερο κόστος από μια διαδρομή με λιγότερους. Αντιστρόφως, μπορούμε να πούμε ότι η περικοπή ενός ενδιάμεσου σταθμού δε πρόκειται να αυξήσει το συνολικό κόστος της διαδρομής. Σε τυπική μορφή, η έννοια αυτή μπορεί και να περιγραφεί ως εξής: η συνάρτηση κόστους c ικανοποιεί την τριγωνική ανισότητα έαν για όλους τους κόμβους u,v,w που ανήκουν στο V ισχύει:

$$c(u,v) \leq c(u,w) + c(w,v)$$

Η τριγωνική ανισότητα αποτελεί μία κατάσταση η οποία συνήθως ικανοποιείται εξ ορισμού.
Παραδείγματος χάριν, μεταξύ δύο σημείων σε ένα επίπεδο, μέσω της ευκλείδιας απόστασης τους η οποία ορίζεται ως κόστος μετακίνησης μεταξύ τους, η τριγωνική ανισότητα ικανοποιείται.

Για το Πρόβλημα του Περιοδεύοντος Πωλητή, δεν έχει βρεθεί κάποιος άμεσος αλγόριθμος που να επιλύει ακριβώς το πρόβλημα με την χρήση της τριγωνικής ανισότητας στη συνάρτηση κόστους, με αποτέλεσμα το ΠΠΠ να παραμένει ένα NP-πλήρες πρόβλημα.
Έτσι, για να μπορέσουμε να έχουμε ένα αποτέλεσμα σε πολυωνυμικό χρόνο, καταφεύγουμε στη χρήση των προσεγγιστικών αλγορίθμων.
Αξίζει να σημειωθεί βέβαια, ότι χωρίς τη χρήση της τριγωνικής ανισότητας, δεν υπάρχει προσεγγιστικός αλγόριθμος πολυωνυμικού χρόνου που να προσφέρει σταθερό λόγο προσέγγισης του προβλήματος, εκτός και εάν επιβεβαιωθεί ότι $P = NP$.

2.3 Που χρησιμοποιείται η Περιοδεία Περιοδεύοντος Πωλητή

Σε πρακτικό επίπεδο, η επίλυση του Προβλήματος του Περιοδεύοντος Πωλητή έχει αρκετές χρήσεις σε πολλούς επιστημονικούς τομείς. Τομείς όπως η βιολογία, η επιστήμη υπολογιστών, η κατασκευή εξαρτημάτων, καθώς επίσης και η χρήση για επαγγελματικούς σκοπούς, όπως η δημιουργία διαδρομών για εταιρείες μεταφορών ή την κατασκευή δρομολογίων για λεοφωρεία.
Γιαυτό το λόγο, έχει κριθεί αναγκαία η εύρεση ενός αλγορίθμου επίλυσης ο οποίος μπορεί να παράξει ένα ορθό αποτέλεσμα με τη μικρότερη δυνατή ανάλωση χρόνου.
Παρακάτω, θα αναφερθούμε σε μερικά παραδείγματα όπου χρησιμοποιείται ως υπόβαθρο το ΠΠΠ για την βελτιστοποίηση λειτουργιών.

2.3.1 Κατασκευή Κυκλωμάτων

Μία άμεση χρήση του ΠΠΠ βρίσκεται στα τυπωμένα κυκλώματα [6], όπου πρέπει να γίνουν τρύπες, ώστε να συνδεθούν αγωγοί που βρίσκονται σε διαφορετικά στρώματα της πλακέτας, ή για να δημιουργηθούν οι κατάλληλες υποδοχές για την τοποθέτηση ολοκληρωμένων κυκλωμάτων.
Οι τρύπες αυτές, ενδέχεται να είναι διαφόρων μεγεθών, και υπάρχει περίπτωση στα διαφορετικά στρώματα να πρέπει να χρησιμοποιηθούν παραπάνω από ένα τρυπάνια για τη δημιουργία της οπής.
Γι' αυτό το λόγο, η κεφαλή του μηχανήματος πρέπει να κινηθεί σε μία συγκεκριμένη θέση για να γίνει αλλαγή του τρυπανιού και συνεχίσει την εργασία της, μια διαδικασία η οποία αποδεικνύεται χρονοβόρα.
Έτσι, χρησιμοποιώντας μια σειρά από εκτελέσεις του περιοδεύοντος πωλητή, ετοιμάζονται περιοδείες για το μηχάνημα, ώστε να μειωθεί το πλήθος αλλαγών των τρυπανιών καθώς επίσης και για να ελαχιστοποιηθεί η συνολική απόσταση κάθε διαδρομής για τη δημιουργία των τρυπών ίδιου μεγέθους.

2.3.2 Κρυσταλλογραφία Ακτίνων Χ

Για τη μελέτη της δομής κρυστάλλων [6], υπάρχει η ανάγκη χρήσης του ΠΠΠ. Για να μπορέσουμε να λάβουμε δεδομένα για τη δομή ενός κρυστάλλου χρησιμοποιούμε ένα διαθλασίμετρο ακτίνων Χ, μαζί με έναν ανιχνευτή ο οποίος μετρά πόσο έντονες είναι οι αντανακλάσεις των ακτινών Χ από διάφορες θέσεις καθ' όλη τη διαδικασία.
Αν και η ίδια η μέτρηση αποτελεί μια γρήγορη διαδικασία, η τοποθέτηση στις διάφορες θέσεις αποτελεί χρονοβόρα, μιας και υπάρχουν περιπτώσεις όπου πρέπει να οριστούν χιλιάδες από αυτές για μερικά πειράματα.
Επειδή στα πειράματα αυτά ίσως και να χρειαστεί να μετακινηθούν μέχρι και 4 μοτέρ, η ελαχιστοποίηση του χρόνου μετακίνησης βελτιώνει αισθητά τον χρόνο του πειράματος και επειδή δεν μας ενδιαφέρει μια συγκεκριμένη σειρά για την λήψη των αποτελεσμάτων από τις θέσεις αυτές, χρησιμοποιείται μια περιοδεία του ΠΠΠ για την εύρεση μιας διαδρομής με τη λιγότερη δυνατή μετακίνηση των μοτέρ.

2.3.3 Επαγγελματική Χρήση

Η χρήση του ΠΠΠ μπορεί να διευκολύνει σημαντικά τον επαγγελματικό τομέα. Οι περιοδείες του μπορούν να χρησιμοποιηθούν για την εύρεση υλικών μέσα σε αποθήκες [6], για τυχόν παραγγελίες όπου κάποιο όχημα πρέπει να βρει και να μεταφέρει τα ζητηθέντα υλικό από τα σημεία που έχουν αποθηκευτεί για να ολοκληρώσει μια τρέχουσα εργασία.

Έτσι, με τον υπολογισμό της διαδρομής με τη χρήση του ΠΠΠ, μειώνεται η μετακίνηση του οχήματος στο ελάχιστο και ταυτόχρονα η παραγγελία ολοκληρώνεται ταχύτερα.

Μία άλλη χρήση του ΠΠΠ στον επαγγελματικό τομέα αφορά την εύρεση διαδρομών για οχήματα [6]. Παραδείγματος χάριν, η εύρεση διαδρομών για ένα στόλο οχημάτων ταχυδρομικής εταιρείας για ένα πλήθος ταχυδρομικών κουτιών μέσα σε ένα όριο χρόνου.
Χρησιμοποιώντας πολλαπλές φορές την επίλυση του ΠΠΠ, μπορούμε να βρούμε τις διαδρομές που πρέπει να ακολουθήσει κάθε διαφορετικό όχημα, καθώς επίσης υπάρχει η δυνατότητα να βρεθεί μια καλή αναλογία χρόνου και αναγκαίου πλήθους οχημάτων για την ελαχιστοποίηση κόστους, αν υπάρχει κάποια μεταβλητή ορίου σε σχέση με τη χωρητικότητα κάθε οχήματος.

3 Αλγοριθμική Επίλυση για το ΠΠΠ

3.1 Γενικά

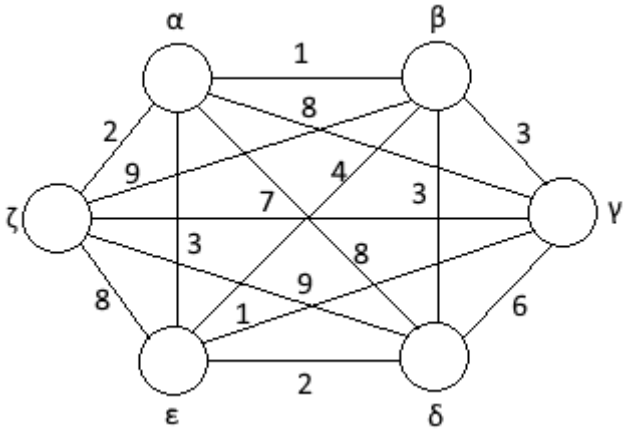
Όπως εξηγήσαμε και στο 2ο κεφάλαιο, η δυνατότητα εύρεσης ενός αλγορίθμου για τον υπολογισμό μίας επίλυσης ενός στιγμιοτύπου του Προβλήματος του Περιοδεύοντος Πωλητή σε πολυωνυμικό χρόνο θεωρείται αβέβαιη, γιαυτό και το πρόβλημα είναι NP-πλήρες. Οπότε, αυτό που θα υλοποιήσουμε είναι ένας προσεγγιστικός αλγόριθμος με την χρήση της τριγωνικής ανισότητας, ο οποίος επιλύει το πρόβλημα σε πολυωνυμικό χρόνο. Παρακάτω, θα αναφέρουμε και θα εξηγήσουμε τι χρειαζόμαστε για την υλοποίηση του παραπάνω αλγορίθμου, θα αναλύσουμε αλγοριθμικά τις λειτουργίες αυτές, καθώς επίσης θα αναφέρουμε και την αλγοριθμική επίλυση του ΠΠΠ με προσέγγιση και βαθμό προσέγγισης 2(n).

3.2 Τι χρειαζόμαστε

Ένας προσεγγιστικός αλγόριθμος για την επίλυση του προβλήματος του περιοδεύοντος πωλητή πέρα από την ανάγκη της ύπαρξης και δεσμευτικής προϋπόθεσης της τριγωνικής ανισότητας χρησιμοποιεί λειτουργίες άλλων αλγορίθμων για να μπορέσει να δημιουργήσει ένα αποτέλεσμα το οποίο πλησιάζει την βέλτιστη λύση σε πολυωνυμικό χρόνο. Η χρήση γράφων για την εισαγωγή στοιχείων στο αλγόριθμο, η χρήση του αλγορίθμου Prim για την εύρεση συντομότερης διαδρομής κόμβων, καθώς και η δημιουργία της περιοδείας με χρήση του χαμιλτονιανού κύκλου αποτελούν αναπόσπαστο κομμάτι της επίλυσης και θα τους αναλύσουμε αλγοριθμικά, καθώς επίσης και προγραμματιστικά στο 4ο κεφάλαιο.

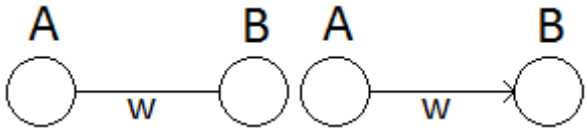
3.2.1 Γράφοι

Οι γράφοι έχουν έναν σημαντικό ρόλο στην επίλυση του ΠΠΠ. Όχι μόνο αλγοριθμικά, ως είσοδο του προβλήματος & εύρεσης της περιοδείας, αλλά και σχηματικά, βοηθώντας μας να δημιουργήσουμε μια εικόνα της εισόδου για οπτική διευκόλυνση μας.
Ένας γράφος αποτελείται απο τους κόμβους και ακμές, που στη περίπτωση μας αποτελούν τις πόλεις και τους «δρόμους» που τις ενώνουν αντίστοιχα. Γενικά, ένας γράφος συμβολίζεται ως G, με το σύνολο των πόλεων να συμβολίζεται ως V και τις ακμές ως E. Συνδυαστικά, ένας γράφος συμβολίζεται $G = (V, E)$. Συνήθως, στις ακμές υπάρχουν βάρη, τα οποία περιγράφουν το κόστος μετακίνησης μεταξύ δύο πόλεων.



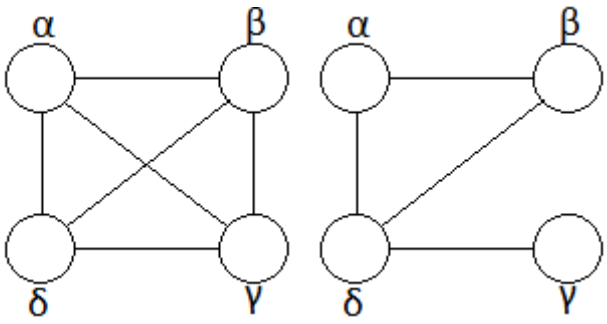
Σχήμα 5: Τυχαίος γράφος με 6 κόμβους

Οι γράφοι χωρίζονται σε δύο κατηγορίες ως προς τον τύπο των ακμών που περιέχει, τους ακατεύθυντους και τους κατευθυνόμενους. Οι ακατεύθυντοι γράφοι περιέχουν ακμές χωρίς κατεύθυνση, δηλαδή η ίδια ακμή χρησιμοποιείται για την σύνδεση 2 πόλεων και προς τις 2 μεριές χωρίς αλλαγή του κόστους μετακίνησης. Παραδείγματος χάριν, αν μία ακμή συνδέει τις πόλεις A και B με κόστος μετακίνησης w, για να μετακινηθείς από την πόλη A στη πόλη B κοστίζει w, και από την πόλη B στη πόλη A πάλι w.
Αντιθέτως, σε έναν κατευθυνόμενο γράφο για το ίδιο παράδειγμα, μία ακμή ορίζει μονάχα τη μία κατεύθυνση, οπότε αν πούμε ότι έχει οριστεί μονάχα μία ακμή απο το A στο B και καμία ακμή απο το B στο A, πηγαίνοντας στο B με κόστος w, δε μπορούμε να γυρίσουμε πίσω στο A.



Σχήμα 6: Αριστερά: μη κατευθυνόμενος γράφος Δεξιά: κατευθυνόμενος γράφος

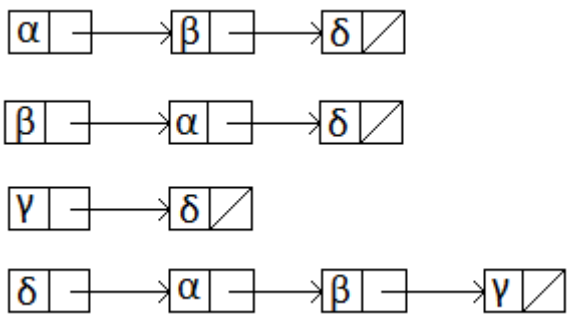
Για την επίλυση του ΠΠΠ, θα χρησιμοποιήσουμε πλήρεις γράφους. Ένας γράφος ονομάζεται πλήρης όταν κάθε κόμβος ο οποίος ανήκει στο σύνολο των κόμβων του συνδέεται με ακμές με όλους τους υπόλοιπους.



Σχήμα 7: Αριστερά: πλήρης γράφος με 4 κόμβους. Δεξιά: μη πλήρης γράφος

Μία ακόμη ιδιότητα που μας ενδιαφέρει για τους γράφους είναι η πυκνότητα τους, η οποία χωρίζεται σε αραιή και πυκνή. Όταν ένας γράφος περιέχει σχετικά λίγες ακμές σε σχέση με τους κόμβους που περιέχει, ονομάζεται αραιός, ενώ όταν περιέχει πολλες ακμές ονομάζεται πυκνός. Γενικά, χρησιμοποιούμε τη σύγκριση μεταξύ πλήθους ακμών και του τετραγώνου του πλήθους των κόμβων για να ορίσουμε την πυκνότητα του γράφου. Αν το πλήθος των ακμών είναι πολύ μικρότερο απο το V^2 τον ονομάζουμε αραιό, ενώ αν υπάρχει σχετικά μικρή διαφορά μεταξυ E και V^2 τον ονομάζουμε πυκνό.

Ο ορισμός της πυκνότητας μας βοηθάει να επιλέξουμε προγραμματιστικά πως θα αναπαραστήσουμε τους γράφους που θα χρησιμοποιήσουμε ως είσοδο. Ένας αραιός γράφος μπορεί να αναπαρασταθεί με λίστες γειτνίασης το οποίο επιτρέπει τη γρήγορη προσπέλαση τους και μείωση της δεσμευμένης μνήμης, ενώ ένας πυκνός γράφος ή ένας γράφος που θέλουμε άμεσα να μαθαίνουμε αν υπάρχει ακμή που συνδέει δύο κόμβους μπορεί να αναπαρασταθεί με πίνακα γειτνίασης.



	α	β	γ	δ
α	-1	1	0	1
β	1	-1	0	1
γ	0	0	-1	1
δ	1	1	1	-1

Σχήμα 8: Αριστερά: λίστα γειτνίασης. Δεξιά: πίνακας γειτνίασης

Αν υπολογίσουμε τις ακμές ενός στιγμιότυπου της εισόδου στο πρόβλημα μας, θα δούμε ότι σε σχέση με το V^2 είναι αρκετά κοντά ως τιμές όπως μπορούμε να δούμε και από το τύπο, κάτι που ορίζει τους γράφους που θα χρησιμοποιήσουμε ως πυκνούς, οπότε η καλύτερη επιλογή είναι η αναπαράσταση μέσω πίνακα γειτνίασης. Παρακάτω, δίνεται ο τύπος υπολογισμού του πλήθους των ακμών για έναν πλήρη γράφο:

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

Παρακάτω θα δούμε βασικές λειτουργίες των γράφων, όπως τη δημιουργία τους και τη συμπλήρωση τους.

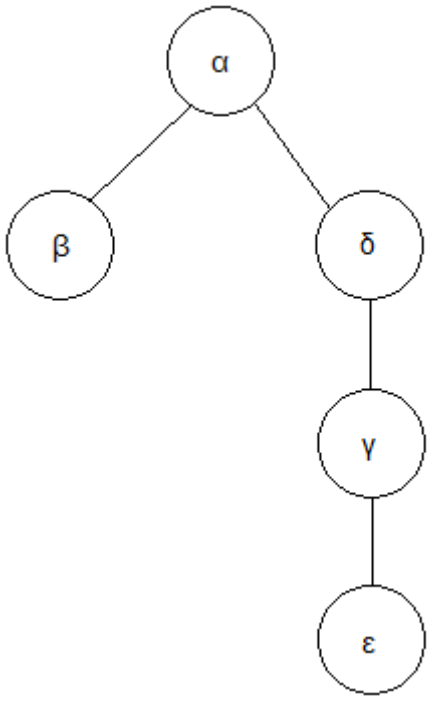
```
Κατασκευή-Γράφου(V)
1.  G ← Δέσμευσε-δισδιάστατο-πίνακα(πλήθος(V),πλήθος(V) )
2.  Επίστρεψε G
```

```
Συμπλήρωση-Γράφου(G, c)
1.    Για κάθε κόμβο u ∈ V
2.      Για κάθε κόμβο v ∈ V
3.        E(u, v) ← 0
4.        c(u, v) ← −∞
5.    Για κάθε κόμβο u ∈ V
6.      Για κάθε κόμβο v ∈ V − u
7.        Αν E(u, v) = 0
8.          Διάβασε w
9.          E(u, v) ← 1
10.         E(v, u) ← 1
11.         c(u, v) ← w
12.         c(v, u) ← w
```

Παρ' όλα αυτά, επειδή προγραμματιστικά ο τρόπος αναπαράστασης των γράφων έχει επίπτωση στην λειτουργία του αλγόριθμου του Prim και κατ' επέκταση και στον χρόνο ολοκλήρωσης του, θα αναλύσουμε περαιτέρω τις διαφοροποιήσεις σε σχέση με τους αλγορίθμους στο 4ο κεφάλαιο.

3.2.2 Prim

Ο αλγόριθμος του Prim χρησιμοποιείται για την κατασκευή ενός Ελαφρύτατου Συνδετικού Δένδρου για το στιγμιότυπο εισόδου που του δόθηκε. Ώς είσοδος, χρησιμοποιείται ένας Γράφος ο οποίος είναι ακατεύθυντος. Ελαφρύτατο Συνδετικό Δένδρο, αλλιώς ΕΣΔ, ονομάζεται ένα σύνολο ακμών T το οποίο αποτελεί υποσύνολο του E οι οποίες συνδέουν όλους τους κόμβους του γράφου χωρίς να δημιουργείται κύκλος. Η λέξη Ελαφρύτατο στην ονομασία χρησιμοποιείται διότι το κόστος του υποσυνόλου T είναι το ελάχιστο δυνατό, ενώ το Συνδετικό Δένδρο ορίζεται από τον τρόπο που σχηματίζεται το αποτέλεσμα του αλγορίθμου, όπου μοιάζει με δένδρο.



Σχήμα 9: Ελάχιστο Συνδετικό Δένδρο

Ο παραπάνω αλγόριθμος, ανάλογα τον τρόπο αναπαράστασης του γράφου έχει διαφορετικό χρόνο χειρότερης επίλυσης. Αν χρησιμοποιηθεί πίνακας γειτνίασης, παράγει αποτέλεσμα σε $O(V^2)$, αν χρησιμοποιηθούν δυαδικοί σωροί έχει χρόνο εκτέλεσης $O(ElogV)$, ενώ αν χρησιμοποιηθούν σωροί Fibonacci, ο αλγόριθμος Prim μπορεί να εξάγει αποτέλεσμα σε $O(E + VlogV)$, κάτι που αποτελεί αισθητή βελτίωση σε περιπτώσεις που το πλήθος των κόμβων είναι πολύ μικρότερο σε σχέση με το πλήθος των ακμών.
Για την κατασκευή του ΕΣΔ, ο αλγόριθμος Prim ταξινομεί τις ακμές σε σχέση με το βάρος τους και κατ' επανάληψη επιλέγει την ακμή με το μικρότερο βάρος, ελέγχει αν δημιουργείται κύκλος αν προστεθεί η επιλεγμένη ακμή, και σε περίπτωση που δεν δημιουργείται, την εισάγει στο υπάρχον δένδρο. Ο αλγόριθμος Prim τερματίζει μόλις εισαχθούν όλοι οι κόμβοι στο δένδρο. Λόγω της παραπάνω διαδικασίας για την επιλογή των ακμών με το μικρότερο βάρος, ο αλγόριθμος συγκαταλέγεται στους άπληστους αλγόριθμους.

Ο αλγόριθμος είναι ο εξής και ισχύει ότι:

$$A = \{(v, \pi[v]) : v \in V - r - Q\}$$

το οποίο σύνολο Α της διαδικασίας ΑΡΧΕΤΥΠΙΚΟ ΕΣΔ τηρείται σιωπηρά. Όταν ο αλγόριθμος τερματίζει, η ουρά προτεραιότητας Q είναι κενή, επομένως το ΕΣΔ Α για το G είναι:

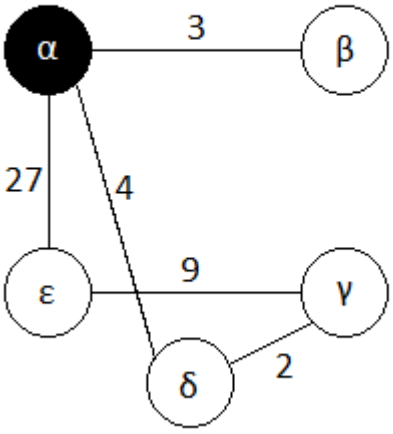
$$A = \{(v, \pi[v]) : v \in V - r\}$$

Παρακάτω, μια απλή αλγοριθμική επίλυση του Prim για τη δημιουργία του ΕΣΔ με χρήση ουράς, καθώς και μια εκτέλεση του για έναν γράφο.

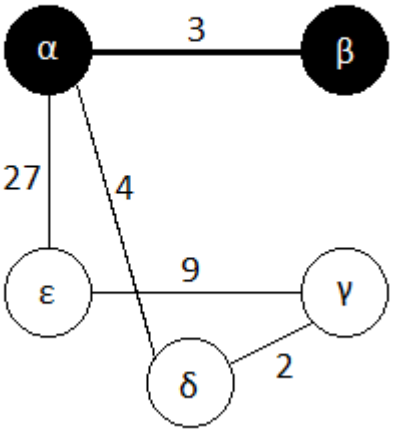
ΕΣΔ-$Prim(G, w, r)$
1. Για κάθε u που ανήκει στο $V[G]$
2. Κλειδί$[u] \leftarrow \infty$
3. $\pi[u] \leftarrow$ΚΕΝΟ
4. Κλειδί$[r] \leftarrow 0$
5. $Q \leftarrow V[G]$
6. Ενόσω $Q \neq 0$
7. $u \leftarrow$ Εξαγωγή Ελαχίστου(Q)
8. Για κάθε v που ανήκει στους $Adj[u]$
9. Αν v ανήκει στο Q και $w(u, v) <$ κλειδί$[v]$
10. Τότε $\pi[v] \leftarrow u$
11. Κλειδί$[v] \leftarrow w(u, v)$

Παρακάτω, η εκτέλεση του αλγόριθμου Prim σε γράφο 5 κόμβων.

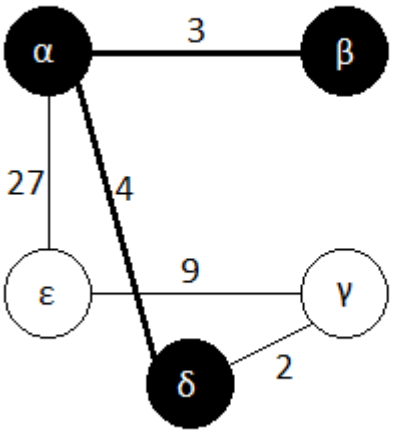
i)



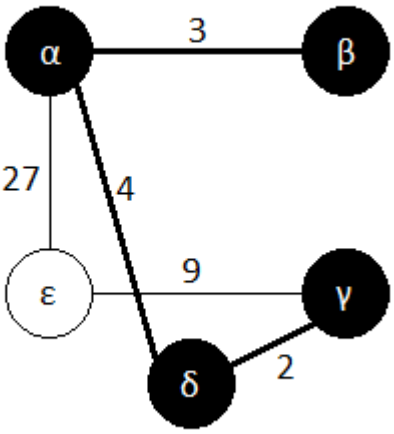
ii)

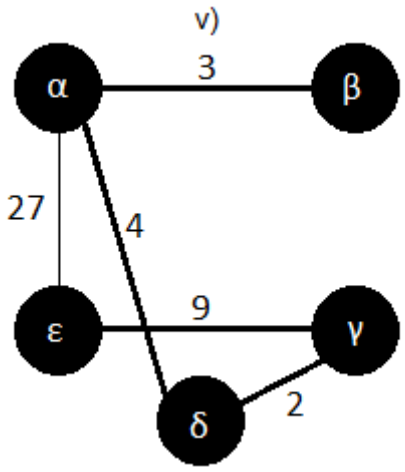


iii)



iv)





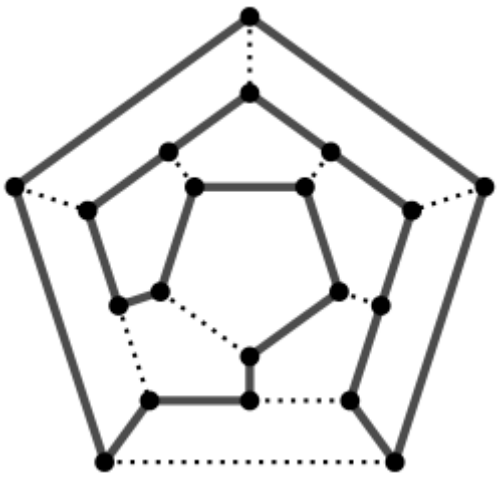
Σχήμα 10: Σχήμα εκτέλεσης πάνω σε γράφο 5 κόμβων

Απο το παραπάνω σχήμα, προκύπτει η διάνυση του δένδρου $a,\beta,a,\delta,\gamma,\epsilon,\gamma,\delta,a$.

3.2.3 Hamilton

Η χρήση του Hamilton έχει να κάνει με την εύρεση μιας διαδρομής σε έναν ακατεύθυντο γράφο ο οποίος περιέχει όλους τους κόμβους του που ανήκουν στο σύνολο των κόμβων του γράφου.
Η επιτυχής εύρεση ενός αποτελέσματος ονομάζει τον γράφο χαμιλτονιανό και το αποτέλεσμα ονομάζεται χαμιλτονιανός κύκλος, ενώ η αποτυχία εύρεσης αποτελέσματος χαρακτηρίζει το γράφο μη χαμιλτονιανό.

Ιστορικά, ο W.R. Hamilton, αναφέρει σε μία επιστολή του ένα μαθηματικό παιχνίδι σε δωδεκάεδρο, όπου ο ένας παίκτης καρφώνει βελόνες σε πέντε διαδοχικούς κόμβους και ο δεύτερος παίκτης πρέπει να δημιουργήσει μια διαδρομή η οποία θα συμπεριλαμβάνει όλους τους κόμβους, δημιουργώντας παράλληλα έναν κύκλο.



Σχήμα 11: Χαμιλτονιανός κύκλος σε δωδεκάεδρο.

Γενικά, ένας αλγόριθμος άμεσης εύρεσης του προβλήματος του χαμιλτονιανού κύκλου χρειάζεται $\Omega(m!) = \Omega(\sqrt{n}!) = \Omega(2^{\sqrt{n}})$ με m το πλήθος των κόμβων και n το μήκος της κωδικοποίησης του γράφου G. Ο παραπάνω χρόνος δεν μπορεί να εκφραστεί στη μορφή $O(n^k)$ για καμία σταθερά k, και κατά συνέπεια, το πρόβλημα ανήκει στα NP-Πλήρη προβλήματα. Κατ' επέκταση, ο ορισμός του ΠΠΠ ως NP-πλήρες πρόβλημα προέρχεται από τον ορισμό του χαμιλτονιανού κύκλου.

Όμως, επειδή αυτό που εξετάζουμε εμείς είναι ο προσεγγιστικός αλγόριθμος του ΠΠΠ, χρησιμοποιώντας το ΕΣΔ που προήλθε από τον αλγόριθμο του Prim, μπορούμε να δημιουργήσουμε μια περιοδεία η οποία προέρχεται από το ΕΣΔ. Αν διασχίσουμε το ΕΣΔ και αφαιρέσουμε τους κόμβους που εμφανίζονται παραπάνω από μία φορά δημιουργούμε την σειρά με την οποία επισκεπτόμαστε τους κόμβους του γράφου.Με αυτό το τρόπο, αθροίζοντας τις ακμές που συνδέουν τους κόμβους αυτούς να έχουμε το κόστος της διαδρομής του στιγμιότυπου του ΠΠΠ που εξετάζουμε.

Παραδείγματος χάριν:
Έστω μια διάνυση ενός γράφου 7 κόμβων με τον αλγόριθμο Prim, ο οποίος στην έξοδο του εμφάνισε ένα ΕΣΔ. Από το ΕΣΔ, προέκυψε η διάνυση $a,\beta,\gamma,\beta,\delta,\zeta,\delta,\beta,a,\epsilon,\eta,\epsilon,a$. Ο αλγόριθμος του χαμιλτονιανού κύκλου, δέχεται την παραπάνω διάνυση στην είσοδο του, και για κάθε κόμβο ο οποίος εμφανίζεται πρώτη φορά στη διάνυση, τον εισάγει στον κύκλο. Ο χαμιλτονιανός κύκλος στην έξοδο του αλγορίθμου θα είναι $a,\beta,\gamma,\delta,\zeta,\epsilon,\eta,a$.

Ο αλγόριθμος που δημιουργεί ένα νέο χαμιλτονιανό κύκλο:

Κατασκευή-Κύκλου()
1. $H \leftarrow$ΚΕΝΟ
2. Επιστροφή H

Ο αλγόριθμος που εισάγει νέο στοιχείο στον χαμιλτονιανό κύκλο:

Εισαγωγή-Στοιχείου(H, t)
1. $c \leftarrow$ Τελευταίο-στοιχείο$(H\ )$
2. Επόμενος$(c) \leftarrow t$

Ο αλγόριθμος που βρίσκει το τελευταιο στοιχείο του κύκλου:

Τελευταίο-στοιχείο(H)
1. $c \leftarrow$ κεφαλή(H)
2. Όσο Επόμενος(c) $\neq$ΚΕΝΟ
3. $c \leftarrow$ Επόμενος(c)
4. Επίστρεψε c

Ο αλγόριθμος που κατασκευάζει τον χαμιλτονιανό στοιχείο:

$Hamilton(L, G)$
1. Για κάθε $u \in V[G]$
2. επίσκεψη[u] $\leftarrow$ ψευδές
3. $H \leftarrow$Κατασκευή-Κύκλου()
4. Για i από 1 μέχρι πλήθος[L] με βήμα 1
5. Αν επίσκεψη[$L[i]$]= ψευδές
6. Εισαγωγή-Στοιχείου($L[i]$)
7. επίσκεψη[$L[i]$] $\leftarrow$ αληθές
8. Επιστροφή H

3.3 Προσεγγιστικός αλγόριθμος ΠΠΠ

Ο αλγόριθμος που επιλύει το Πρόβλημα του Περιοδεύοντος Πωλητή, αποτελεί ένα κέλυφος το οποίο συγκεντρώνει τους επιμέρους αλγορίθμους που εξηγήσαμε παραπάνω. Λαμβάνοντας έναν γράφο ως είσοδο, με την προϋπόθεση ότι ο γράφος είναι ελεγμένος για την ορθότητα του, δηλαδή αν είναι πλήρης, ακατεύθυντος και τα κόστη των ακμών του είναι θετικοί αριθμοί πέρα από τη κύρια διαγώνιο, καθώς επίσης ότι ισχύει η τριγωνική ανισότητα για κάθε δυνατό συνδυασμό κόμβων, ο αλγόριθμος επιλέγει τυχαία (ή επιλέγουμε εμείς) έναν κόμβο ως έναρξη και στη συνέχεια με τα κατάλληλα βήματα δημιουργεί στην έξοδο του τον χαμιλτονιανό κύκλο ο οποίος αποτελεί την προσεγγιστική λύση του προβλήματος.

Προσεγγιστική Περιοδεία ΠΠΠ(G, c)
1. Επιλέγουμε έναν κόμβο $r \in V[G]$ ως ρίζα
2. Υπολογίζουμε ένα ΕΣΔ T για το G από τη ρίζα r με χρήση του ΕΣΔ-$Prim(G, c, r)$
3. Θέτουμε L τον κατάλογο κόμβων που διατρέχει μια καθοδική διάνυση δένδρου του T
4. Επιστροφή ο χαμιλτονιανός κύκλος H που διατρέχει τους κόμβους με τη σειρά του L

Όπως έχουμε ήδη αναφέρει, η προσεγγιστική περιοδεία του ΠΠΠ χρειάζεται πολυωνυμικό χρόνο για την δημιουργία αποτελέσματος. Με μια απλή υλοποίηση του ΕΣΔ-Prim ο αλγόριθμος χρειάζεται $\Theta(V^2)$, ενώ όπως αναφέραμε και στο 3.2.2 μια βελτιστοποίηση του ΕΣΔ-Prim μειώνει τον χρόνο του προσεγγιστικού αλγορίθμου. Τέλος, η απόδειξη ότι ο παραπάνω προσεγγιστικός αλγόριθμος υπό τη τριγωνική ανισότητα είναι ένας 2-προσεγγιστικός αλγόριθμος του ΠΠΠ είναι η εξής:

Απόδειξη:

Έστω ένα στιγμιότυπο του ΠΠΠ με γράφο $G = (V, E)$. Έστω ότι H* ονομάζεται μια βέλτιστη περιοδεία για το στιγμιότυπο. Δεδομένου ότι αν διαγράψουμε μια οποιαδήποτε ακμή από τη περιοδεία λαμβάνουμε ένα συνδετικό δένδρο, το βάρος του ελαφρύτατου συνδετικού δένδρου T αποτελεί κάτω φράγμα του κόστους μιας βέλτιστης περιοδείας, δηλαδή

$$c(T) <= c(H^*)(1)$$

Σε μια πλήρη διάνυση του T, οι κόμβοι παρατίθενται στο κατάλογο κάθε φορά που η διαδρομή περνάει από αυτούς. Έστω W η διάνυση αυτή. Δεδομένου ότι η πλήρης διάνυση διατρέχει την κάθε ακμή του T ακριβώς 2 φορές, αν επεκτείνουμε τον ορισμό του κόστους ώστε να μπορούμε να χειριστούμε πολυσύνολα ακμών, έχουμε ότι:

$$c(W) = 2c(T)(2)$$

από τις εξισώσεις (1) και (2) έχουμε ότι:

$$c(W) = 2c(H^*)$$

οπότε το κόστος της διάνυσης W είναι εντός των ορίων ενός παράγοντα 2 από το κόστος μιας βέλτιστης περιοδείας.
Επειδή όμως η διάνυση W δεν είναι περιοδεία, μιας και σε αυτήν αναφέρονται κόμβοι περισσότερο από μία φορές, χρησιμοποιούμε τον ορισμό της τριγωνικής ανισότητας για το πρόβλημα μας, όπου αναφέρει ότι είναι δυνατή η διαγραφή ενός κόμβου χωρίς την αύξηση του κόστους. Έτσι, κρατάμε στη διάνυση μόνο τις στιγμές που ένας κόμβος αναφέρεται πρώτη φορά, η οποία είναι ίδια με την καθοδική διάνυση ενός Ελαφρύτατου Συνδετικού Δένδρου T. Έστω H ο χαμιλτονιανός κύκλος που προκύπτει από τη συγκεκριμένη διάνυση. Επειδή ο κύκλος δημιουργήθηκε από τη διαγραφή κόμβων της διάνυσης W, ισχύει ότι:

$$c(H) \leq c(W)$$

και κατ' επέκταση ότι:

$$c(H) \leq 2c(H^*)$$

3.3.1 Πολυπλοκότητα

Όπως αναφέραμε και παραπάνω, ο χρόνος εκτέλεσης χειρότερης περίπτωσης του προσεγγιστικού αλγορίθμου για την επίλυση του ΠΠΠ εξαρτάται από αρκετούς παράγοντες.

Τρόπος αλγοριθμικής απεικόνισης του γράφου:

Για τη καλύτερη δυνατή άντληση στοιχείων του γράφου απο τον αλγόριθμο είναι ο πίνακας γειτνίασης, όπως εξηγήσαμε και στο κεφάλαιο 3.2.1. Αυτό ωφείλεται στην αμεσότητα προσπέλασης του σε περίπτωση που ο αλγόριθμος θέλει να μάθει για την ύπαρξη μιας ακμής καθώς και το κόστος αυτής. Η πράξη αυτή έχει χρόνο $\Theta(1)$.

Τρόπος Εισαγωγής στοιχείων στον αλγόριθμο Prim:

Παράλληλα όμως, η χρήση του πίνακα γειτνίασης δεν ευνοεί την εύρεση ΕΣΔ με τον αλγόριθμο ΕΣΔ-Prim, ο οποίος χρειάζεται $O(V^2)$ για να μπορέσει να παράξει το ΕΣΔ. Αλγοριθμικά, η καλύτερη λύση όπως εξηγήσαμε και στο κεφάλαιο 3.2.2 είναι η χρήση των σωρών fibonacci, όπου με τη βοήθεια τους ο χρόνος εκτέλεσης του Prim ελαχιστοποιείται στο $O(E + VlogV)$, ενώ μια ενδιάμεση λύση αποτελεί η χρήση απλών δυαδικών σωρών με χρόνο εκτέλεσης $O(ElogV)$.

Στο πρόβλημα μας, η εύρεση της διάνυσης απο το ΕΣΔ καθώς και η εύρεση της προσεγγιστικής περιοδείας απο τη διάνυση αυτή αποτελούν πράξεις οι οποίες είναι αμελητέες μπροστά στην δυσκολία της δημιουργίας του ΕΣΔ, με αποτέλεσμα η πολυπλοκότητα του συνόλικού αλγορίθμου του ΠΠΠ να περιορίζεται μεταξύ $O(E + VlogV)$ και $O(V^2)$, η οποία εξαρτάται απο την υλοποίηση του αλγορίθμου Prim που θα επιλέξουμε.

3.3.2 Σύγκριση με άλλους τρόπους επίλυσης

Όπως εξηγήσαμε και στο 2ο κεφάλαιο, υπάρχουν διάφοροι αλγόριθμοι οι οποίοι επιλύουν το ΠΠΠ. Παρακάτω θα αναφέρουμε τους αλγόριθμους αυτούς και θα κάνουμε σύγκριση με τον προσεγγιστικό αλγόριθμο που αναλύσαμε.

Εξαντλητικός Αλγόριθμος με αναζήτηση όλων των δυνατών συνδυασμών:
Ο παραπάνω αλγόριθμος είναι ο πιο άμεσος τρόπος σκέψης για την παραγωγή ενός άμεσου αποτελέσματος. Επειδή δοκιμάζει όλους τους δυνατούς συνδυασμούς, ο χρόνος του ανέρχεται σε $O(n!)$. Αποτελεί μια υπερβολικά χρονοβόρα λύση για πλήθος κόμβων άνω των 15.

Ο αλγόριθμος Held–Karp[2]:
Πρόκειται για μια απο τις πρώτες υλοποιήσεις για την επίλυση του ΠΠΠ, με χρόνο εκτέλεσης $O(n^2 2^n)$.

Οι παραπάνω αλγόριθμοι αποτελούν τις άμεσες λύσεις του ΠΠΠ. Αν και παράγουν μια βέλτιστη περιοδεία, αποτελούν χρονοβόρες λύσεις, όπως φαίνεται και στον χρόνο εκτέλεσης τους, και είναι καλό να χρησιμοποιούνται μόνο όταν υπάρχει ανάγκη υπολογισμού της βέλτιστης περιοδείας για λόγους ακριβείας.

Αν και ο προσεγγιστικός αλγόριθμος που περιγράψαμε αποτελεί έναν 2-προσεγγιστικό αλγόριθμο με έναν ικανοποιητικό λόγο προσέγγισης, συνήθως η προσεγγιστική περιοδεία ΠΠΠ δεν αποτελεί την καλύτερη επιλογή για το πρόβλημα.

Ευρυστικοί Αλγόριθμοι:

Ο αλγόριθμος του κοντινότερου γειτονα:
Ο αλγόριθμος του κοντινότερου γείτονα (Nearest Neighbour Algorithm) προκειται για έναν άπληστο αλγόριθμο ο οποίος επιλέγει τον συντομότερο γειτονικό κόμβο που δεν έχει ακόμη επισκεφτεί. Είναι ένας γρήγορος αλγόριθμος ο οποίος παράγει αποτέλεσμα σε $O(logV)$ και κατα μέσο όρο δίνει 25% χειρότερα αποτελέσματα απότι ένας άμεσος αλγόριθμος.

Ο αλγόριθμος του Χριστοφίδη:
Πρόκειται για έναν απο τους πρώτους προσεγγιστικούς αλγόριθμους του ΠΠΠ. Χρησιμοποιεί γράφους Euler για να δημιουργήσει μονοπάτια Euler με χρήση ΕΣΔ. Παράγει αποτέλεσμα σε χρόνο $O(n^3)$ και στη χειρότερη περίπτωση, το αποτέλεσμα θα είναι μόλις 1.5 φορά μεγαλύτερο από το βέλτιστο.[11]

4 Υλοποίηση στη γλώσσα C

4.1 Γενικά

Ένα βασικό κομμάτι της πτυχιακής αυτής, ασχολείται με την υλοποίηση του προσεγγιστικού αλγόριθμου για το πρόβλημα του περιοδεύοντος πωλητή. Στο 3ο κεφάλαιο, ασχοληθήκαμε με την αλγοριθμική επίλυση του ΠΠΠ και αναλύσαμε έναν 2-προσεγγιστικό αλγόριθμο υπό την τριγωνική ανισότητα. Σε αυτό το κεφάλαιο, χρησιμοποιώντας ως βάση την αλγοριθμική επίλυση, θα υλοποιήσουμε τον αλγόριθμο αυτόν, καθώς και τους αλγόριθμους που χρησιμοποιεί για να λειτουργήσει, στη γλώσσα προγραμματισμού C.

4.1.1 Τύποι δεδομένων

Γενικά, για την γενική λειτουργία της προγραμματιστικής επίλυσης, χρησιμοποιείται ως βασικός τύπος δεδομένων οι ακέραιοι αριθμοί (int). Επειδή θα χρησιμοποιηθούν λίστες γειτνίασης για την αναπαράσταση του γράφου, έχουν οριστεί ειδικοί τύποι δεδομένων με τη χρήση δομών δεδομένων struct, για την καλύτερη δυνατή απόδοση του προγράμματος. Για τις αναπαραστάσεις των ΕΣΔ καθώς και του χαμιλτονιανού κύκλου, θα χρησιμοποιηθούν λίστες, οι οποίες χρησιμοποιούν δομές δεδομένων struct, οι οποίες θα οριστούν και θα περιγραφούν στα κεφάλαια της επίλυσης των αντίστοιχων αλγορίθμων.

4.1.2 Εισαγωγή στοιχείων

Η εισαγωγή του γράφου στην είσοδο του προγράμματος γίνεται από κατάλληλα κατασκευασμένη συνάρτηση η οποία δέχεται έναν θετικό μη μηδενικό ακέραιο αριθμό ο οποίος αντιπροσωπεύει το πλήθος των κόμβων που θέλουμε να χρησιμοποιήσουμε. Στη συνέχεια, ο αλγόριθμος κατασκευάζει τον ανάλογο πίνακα γειτνίασης χρησιμοποιώντας τυχαίους αριθμούς ως βάρη των ακμών μεταξύ των κόμβων και τον διορθώνει ώστε να ισχύει η τριγωνική ανισότητα για κάθε ζευγάρι κόμβων. Τέλος, κατασκευάζει τις λίστες γειτνίασης του προγράμματος και τις επιστρέφει στο κυρίως πρόγραμμα. Επειδή μια τέτοια διαδικασία ελέγχου είναι χρονοβόρα και δημιουργήθηκε κυρίως για την δημιουργία και την εισαγωγή τυχαίων γράφων για τον έλεγχο ορθότητας του προγράμματος, η πολυπλοκότητα της δε προστίθεται στη συνολική πολυπλοκότητα του προγράμματος, μιας και κάλλιστα θα μπορούσε να θεωρηθεί αυτούσιο τυχαιοκρατικό πρόγραμμα κατασκευής τυχαίων γράφων ενός επιλεγμένου πλήθους κόμβων, αφήνοντας έτσι την υλοποίηση του προσεγγιστικού αλγορίθμου σχεδόν ίδια με την αλγοριθμική επίλυση.

```
1  int** InitializeGraph(int);
2  int** FillGraph(int**,int);
3  void CheckGraph(int**,int,int*);
4  void PrintGraph(int**,int);
5  void PrintGraphToFile(int**,int,char*,int);
6  void FreeGraph(int **graph,int size);
7
8  int** CreateGraph(int size)
9  {
10   int i,j;
11   int** graph;
```

```
12    int totalTries=1;
13    graph=FillGraph(InitializeGraph(size),size);
14    CheckGraph(graph,size,&totalTries);
15    return graph;
16  }
17
18  int** InitializeGraph(int size)
19  {
20    int i,j;
21    int** graph;
22    graph=malloc(size*sizeof(int*));
23    for(i=0;i<size;i++)
24    {
25      graph[i]=malloc(size*sizeof(int));
26    }
27
28
29    for(i=0;i<size;i++)
30    {
31      for(j=0;j<size;j++)
32      {
33        graph[i][j]=0.0f;
34      }
35    }
36    return graph;
37  }
38
39
40  int** FillGraph(int** graph, int size)
41  {
42    time_t t;
43      srand((unsigned) time(&t));
44    int i,j,num;
45
46    for(i=0;i<size;i++)
47    {
48      for(j=0;j<size;j++)
49      {
50        if(i>j)
51          continue;
52        if(i==j)
53          graph[i][j]=-1;
54        else
55        {
56          graph[i][j]=((rand()%200)+20);
57          graph[j][i]=graph[i][j];
58        }
59      }
60    }
61    return graph;
```

```
62 }
63
64 void PrintGraph(int** graph,int p)
65 {
66   int i,j;
67   printf("\n-------------------------\n");
68   for(i=0;i<p;i++)
69   {
70     for(j=0;j<p;j++)
71     {
72       printf("%4d ",graph[i][j]);
73     }
74     printf("\n");
75   }
76   printf("\n-------------------------\n");
77 }
78
79
80 void PrintGraphToFile(int** graph,int p,char* mode,int
     totalTries)
81 {
82   FILE *fp;
83   int i,j;
84   fp=fopen("t.txt",mode);
85   if(mode[0]=='w')
86     fprintf(fp,"start:\n");
87   else
88     fprintf(fp,"end:\n");
89   for(i=0;i<p;i++)
90   {
91     for(j=0;j<p;j++)
92     {
93       fprintf(fp,"%4d ",graph[i][j]);
94     }
95     fprintf(fp,"\n");
96   }
97   fprintf(fp,"\n");
98   if(mode[0]=='a')
99     fprintf(fp,"Total Tries: %d\n",totalTries);
100  fclose(fp);
101 }
102
103
104 void CheckGraph(int** graph,int p,int *tryCount)
105 {
106   int i,j,k;
107   int flag=0;
108
109   for(i=0;i<p;i++)
110   {
```

```
111       for(j=0;j<p;j++)
112       {
113         if(i>=j)
114           continue;
115         for(k=0;k<p;k++)
116         {
117           if(k==i||k==j)
118             continue;
119           if(graph[i][j]<graph[i][k]+graph[k][j])
120           {
121           }
122           else if(graph[i][j]==graph[i][k]+graph[k][j])
123           {
124           }
125           else
126           {
127             graph[i][j]=(rand()%graph[i][k]+graph[k][j]-20)
    +20;
128             graph[j][i]=graph[i][j];
129             flag=1;
130           }
131         }
132       }
133     }
134     if(flag==1)
135     {
136       (*tryCount)++;
137       CheckGraph(graph,p,tryCount);
138     }
139 }
140
141
142 void FreeGraph(int **graph,int size)
143 {
144   int i;
145   for(i=0;i<size;i++)
146     free(graph[i]);
147   free(graph);
148 }
```

Listing 1: graph.h

4.2 Υλοποίηση Prim

Η υλοποίηση του αλγόριθμου του Prim βαδίζει στη λογική της αλγοριθμικής επίλυσης του. Ο αλγόριθμος δέχεται τον γράφο και τον ριζικό κόμβο, και δημιουργεί ένα ΕΣΔ και το επιστρέφει στην προσεγγιστική επίλυση του ΠΠΠ. Παρακάτω, η επίλυση και οι δομές δεδομένων που χρησιμοποιούνται.

```
1  struct Node
2  {
3    int name;
4    struct Node *parent;
5    struct Node **childs;
6    int childCount;
7  };
8  struct tmpNode
9  {
10   int name;
11   struct tmpNode *next;
12 };
13
14 struct MST
15 {
16   struct Node *root;
17   int totalCost;
18 };
19
20 struct tmpNode** TempNodes(int size,int *parents);
21 void PrintMST(struct MST *mst);
22 void PrintNode(struct Node *node);
23 void PrintChilds(struct Node *node);
24 void FreeMST(struct MST *T);
25 struct MST* CreateMinimumSpanningTree(int** graph,int size
     ,int startingPoint);
26 void SimplePrim(int** graph,int * keys,int *parents,int
     startingPoint,int size);
27
28 struct MST* CreateMinimumSpanningTree(int** graph,int size
     ,int startingPoint)
29 {
30   int *keys;
31   int *parents;
32   int i,j;
33   int count;
34   int totalCost;
35   keys=malloc(size*sizeof(int));
36   parents=malloc(size*sizeof(int));
37   totalCost=0;
38   SimplePrim(graph,keys,parents,startingPoint,size);
39   struct MST *mst;
40   mst=malloc(1*sizeof(struct MST));
41   mst->root=NULL;
```

```
42    mst->totalCost=0;
43    struct tmpNode **tmpNodes;
44    tmpNodes=TempNodes(size,parents);
45    struct Node **nodes=malloc(size*sizeof(struct node*));
46    for(i=0;i<size;i++)
47      nodes[i]=NULL;
48    struct tmpNode *tmp;
49    for(i=0;i<size;i++)
50    {
51      j=0;
52      if(tmpNodes[i]==NULL)
53      {
54
55      }
56      else
57      {
58        for(tmp=tmpNodes[i];tmp!=NULL;tmp=tmp->next)
59          j++;
60      }
61
62      nodes[i]=malloc(1*sizeof(struct Node));
63      nodes[i]->name=i;
64      nodes[i]->parent=NULL;
65      nodes[i]->childCount=j+0;
66      if(j>0)
67        nodes[i]->childs=malloc(nodes[i]->childCount*sizeof(
     struct node*));
68      for(j=0;j<nodes[i]->childCount;j++)
69        nodes[i]->childs[j]=NULL;
70      if(parents[i]==-1)
71        mst->root=nodes[i];
72    }
73    for(i=0;i<size;i++)
74    {
75      j=0;
76      for(tmp=tmpNodes[i];tmp!=NULL;tmp=tmp->next)
77      {
78        nodes[i]->childs[j]=nodes[tmp->name];
79        nodes[tmp->name]->parent=nodes[i];
80        j++;
81      }
82    }
83    for(i=0;i<size;i++)
84    {
85      totalCost+=keys[i];
86    }
87    mst->totalCost=totalCost;
88    return mst;
89  }
90
```

```
91 void PrintMST(struct MST *mst)
92 {
93   PrintNode(mst->root);
94   printf("\ntotalCost:%d\n",mst->totalCost);
95 }
96 void PrintNode(struct Node *node)
97 {
98   struct Node *t;
99   int i;
100  printf("\nNode: %d->",node->name+1);
101  PrintChilds(node);
102  for(i=0;i<node->childCount;i++)
103  {
104    t=node->childs[i];
105    PrintNode(t);
106  }
107 }
108 void PrintChilds(struct Node *node)
109 {
110   struct Node *t;
111   int i;
112   if(node->childCount==0)
113     printf("/");
114   for(i=0;i<node->childCount;i++)
115   {
116     t=node->childs[i];
117     printf("%d",t->name+1);
118     if(i+1<node->childCount)
119       printf(", ");
120   }
121 }
122
123 struct tmpNode** TempNodes(int size,int *parents)
124 {
125   struct tmpNode **tmpNodes=malloc(size*sizeof(struct
    tmpNode*));
126   int i,j;
127   int count;
128   for(i=0;i<size;i++)
129     tmpNodes[i]=NULL;
130   for(i=0;i<size;i++)
131   {
132     for(j=0;j<size;j++)
133     {
134       if(j==i)
135         continue;
136       if(parents[j]==i)
137       {
138         struct tmpNode *tmp=malloc(1*sizeof(struct tmpNode
    ));
```

```
139             tmp->name=j;
140             tmp->next=NULL;
141             if(tmpNodes[i]==NULL)
142             {
143               tmpNodes[i]=tmp;
144             }
145             else
146             {
147               struct tmpNode *c;
148               for(c=tmpNodes[i];c->next!=NULL;c=c->next)
149               {
150               }
151               c->next=tmp;
152             }
153           }
154       }
155   }
156   return tmpNodes;
157 }
158
159 void SimplePrim(int** graph,int * keys,int *parents,int
      startingPoint,int size)
160 {
161   int *queue;
162   int *checked;
163   int queueSize;
164   int i,j,u,v;
165
166   for(i=0;i<size;i++)
167   {
168     keys[i]=INT_MAX;
169     parents[i]=-1;
170   }
171   keys[startingPoint]=0;
172   queue=CreateQueue(size,keys,&queueSize);
173   checked=malloc(size*sizeof(int));
174   for(i=0;i<size;i++)
175   {
176     checked[i]=-1;
177   }
178   while(!QueueEmpty(queueSize))
179   {
180     u=QueueExtractMin(queue,keys,&queueSize);
181     checked[u]=1;
182     for(v=0;v<size;v++)
183     {
184       if(u==v)
185         continue;
186       if(QueueContains(checked,v)&&graph[u][v]<keys[v])
187       {
```

```
188             parents[v]=u;
189             int prevkey=keys[v];
190             keys[v]=graph[u][v];
191           }
192           else
193           {
194           }
195       }
196       QueueReorder(queue,keys,&queueSize);
197     }
198 }
199 void FreeMST(struct MST *T)
200 {
201   free(T);
202 }
```

Listing 2: prim.c

4.3 Υλοποίηση Hamilton

Η δημιουργία του χαμιλτονιανού κύκλου ακολουθεί μετά την δημιουργία του καταλόγου των κόμβων της διάνυσης. Η συνάρτηση λαμβάνει τον κατάλογο και δημιουργεί την περιοδεία σε αυτούς με τα ανάλογα κόστη και επιστρέφει την κεφαλή Η στη προσεγγιστική επίλυση του ΠΠΠ. Παρακάτω οι συναρτήσεις της υλοποίησης καθώς και οι δομές δεδομένων που χρησιμοποιούνται.

```
1  struct HamiltonianNode
2  {
3    int item;
4    struct HamiltonianNode *next;
5    int cost;
6  };
7
8  struct HamiltonianCycle
9  {
10   struct HamiltonianNode *root;
11   int totalCost;
12 };
13
14 struct VNode
15 {
16   int item;
17   struct VNode *next;
18 };
19
20 struct NodeList
21 {
22   struct VNode *root;
23 };
24
25 struct NodeList* GetNodeList(struct MST *T);
```

```
26 struct HamiltonianCycle* GetHamiltonianCycle(struct
      NodeList *L,int **graph,int size);
27 struct VNode* GetVNodes(struct NodeList *nodeList,struct
      VNode *c,struct Node *t);
28 void PrintHamiltonianCycle(struct HamiltonianCycle *H);
29 void CompleteCosts(struct HamiltonianCycle *H,int **graph)
      ;
30 void FreeNodeList(struct NodeList *L);
31 void FreeHamiltonianCycle(struct HamiltonianCycle *H);
32
33
34 struct NodeList* GetNodeList(struct MST *T)
35 {
36   struct Node *r=T->root;
37   struct NodeList *nodeList;
38   nodeList=malloc(1*sizeof(struct NodeList));
39   nodeList->root=NULL;
40   struct VNode *c,*tmp;
41   c=GetVNodes(nodeList,c,r);
42
43   return nodeList;
44 }
45
46 struct VNode* GetVNodes(struct NodeList *nodeList,struct
      VNode *c,struct Node *t)
47 {
48   int rootNode=0;
49   if(nodeList->root==NULL)
50   {
51     rootNode=1;
52     struct VNode *tmp;
53     tmp=malloc(1*sizeof(struct VNode));
54     tmp->item=t->name;
55     tmp->next=NULL;
56     nodeList->root=tmp;
57     c=nodeList->root;
58   }
59   if(t->childCount==0)
60   {
61     struct VNode *tmp;
62     tmp=malloc(1*sizeof(struct VNode));
63     tmp->item=t->name;
64     tmp->next=NULL;
65     c->next=tmp;
66     c=c->next;
67   }
68   else
69   {
70     int i;
71     for(i=0;i<t->childCount;i++)
```

```
72      {
73        if(rootNode==0)
74        {
75
76          struct VNode *tmp;
77          tmp=malloc(1*sizeof(struct VNode));
78          tmp->item=t->name;
79          tmp->next=NULL;
80          c->next=tmp;
81          c=c->next;
82        }
83        else
84        {
85          rootNode=0;
86        }
87        c=GetVNodes(nodeList,c,t->childs[i]);
88      }
89      struct VNode *tmp;
90      tmp=malloc(1*sizeof(struct VNode));
91      tmp->item=t->name;
92      tmp->next=NULL;
93      c->next=tmp;
94      c=c->next;
95    }
96    return c;
97  }
98
99  void FreeNodeList(struct NodeList *L)
100 {
101   struct VNode *t,*c;
102   for(t=L->root;t!=NULL;t=c)
103   {
104     c=t->next;
105     free(t);
106   }
107   free(L);
108 }
109
110 struct HamiltonianCycle* GetHamiltonianCycle(struct
      NodeList *L,int **graph,int size)
111 {
112   int i;
113   int *visited=malloc(size*sizeof(int));
114   for(i=0;i<size;i++)
115     visited[i]=0;
116   struct VNode *r=L->root;
117   struct VNode *t;
118   struct HamiltonianCycle *H=malloc(1*sizeof(struct
      HamiltonianCycle));
119   struct HamiltonianNode *c,*tmp;
```

```
120   H->root=NULL;
121   H->totalCost=0;
122   for(t=r;t!=NULL;t=t->next)
123   {
124     if(visited[t->item]==0)
125     {
126       tmp=malloc(1*sizeof(struct HamiltonianNode));
127       tmp->item=t->item;
128       tmp->next=NULL;
129       tmp->cost=0;
130       visited[t->item]=1;
131       if(H->root==NULL)
132       {
133         H->root=tmp;
134         c=H->root;
135       }
136       else
137       {
138         c->next=tmp;
139         c=c->next;
140       }
141     }
142   }
143   CompleteCosts(H,graph);
144   return H;
145 }
146
147 void PrintHamiltonianCycle(struct HamiltonianCycle *H)
148 {
149   printf("\n");
150   struct HamiltonianNode *t;
151   for(t=H->root;t!=NULL;t=t->next)
152   {
153     printf("%d (cost:%d)-> ",t->item+1,t->cost);
154   }
155   printf("%d (end)\n",H->root->item+1);
156 }
157
158 void CompleteCosts(struct HamiltonianCycle *H,int **graph)
159 {
160   struct HamiltonianNode *t;
161   int totalCost=0;
162   for(t=H->root;t!=NULL;t=t->next)
163   {
164     if(t->next==NULL)
165     {
166       t->cost=0;
167     }
168     else
169     {
```

```
170         t->cost=graph[t->item][t->next->item];
171       }
172       totalCost+=t->cost;
173     }
174     H->totalCost=totalCost;
175 }
176
177 void FreeHamiltonianCycle(struct HamiltonianCycle *H)
178 {
179     struct HamiltonianNode *t,*c;
180     for(t=H->root;t!=NULL;t=c)
181     {
182       c=t->next;
183       free(t);
184     }
185     free(H);
186 }
```

Listing 3: hamilton.c

4.4 Υλοποίηση περιοδείας ΠΠΠ

Η υλοποίηση της περιοδείας ΠΠΠ ακολουθεί την αλγοριθμική επίλυση. Όπως αναφέραμε και στο 3ο κεφάλαιο, ο προσεγγιστικός αλγόριθμος αποτελεί ένα «κέλυφος» το οποίο στεγάζει όλες τις επιμέρους διαδικασίες για να δημιουργηθεί η περιοδεία. Έτσι και εδώ, πέρα από μερικές δηλώσεις μεταβλητών, η υλοποίηση λαμβάνει και παραδίδει δεδομένα στις υπόλοιπες συναρτήσεις και στο τέλος επιστρέφει την περιοδεία στο κυρίως πρόγραμμα.

```
1  #include "graph.h"
2  #include "queue.h"
3  #include "prim.c"
4  #include "hamilton.c"
5
6  struct HamiltonianCycle* TravelingSalesman(int size)
7  {
8    int **graph;
9    int i;
10   graph=CreateGraph(size);
11   struct MST *T;
12   struct NodeList *L;
13   struct HamiltonianCycle *H;
14   int startingPoint=0;
15   PrintGraph(graph,size);
16   T=CreateMinimumSpanningTree(graph,size,startingPoint);
17   L=GetNodeList(T);
18   H=GetHamiltonianCycle(L,graph,size);
19
20   FreeMST(T);
21   FreeNodeList(L);
22   FreeGraph(graph,size);
```

```
23   return H;
24 }
```

Listing 4: TSP.c

4.5 Συνολικός κώδικας υλοποίησης

Παρακάτω παρατίθεται το σύνολο της υλοποίησης του προσεγγιστικού αλγόριθμου του Προβλήματος του Περιοδεύοντος Πωλητή στη γλώσσα προγραμματισμού C.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "TSP.c"
5
6
7 #define SIZE 5
8
9 int main()
10 {
11   int i;
12   struct HamiltonianCycle *H;
13   H=TravelingSalesman(SIZE);
14   printf("total cost: %d\n",H->totalCost);
15   FreeHamiltonianCycle(H);
16   return 0;
17 }
```

Listing 5: main.c

```
1 int* CreateQueue(int size,int* keys,int* queueSize);
2 void QueueReorder(int* queue,int* keys,int* queueSize);
3 int QueueExtractMin(int* queue,int* keys,int* queueSize);
4 int QueueContains(int* checked,int v);
5 int QueueEmpty(int queueSize);
6 void PrintQueue(int* queue,int queueSize);
7
8 int* CreateQueue(int size,int* keys,int* queueSize)
9 {
10   int* queue;
11   int i;
12   queue=malloc(size*sizeof(int));
13   (*queueSize)=size;
14   for(i=0;i<(*queueSize);i++)
15   {
16     queue[i]=i;
17   }
18   QueueReorder(queue,keys,queueSize);
19   return queue;
20 }
21
22 void QueueReorder(int* queue,int* keys,int* queueSize)
```

```
23 {
24   int i,j;
25   int tmp;
26   for(i=0;i<(*queueSize)-1;i++)
27   {
28     for(j=0;j<(*queueSize)-1;j++)
29     {
30       if(keys[queue[j]]>keys[queue[j+1]])
31       {
32         tmp=queue[j];
33         queue[j]=queue[j+1];
34         queue[j+1]=tmp;
35       }
36     }
37   }
38
39 }
40
41 int QueueExtractMin(int* queue,int* keys,int* queueSize)
42 {
43   int min=queue[0];
44   int i;
45   for(i=0;i<(*queueSize)-1;i++)
46   {
47     queue[i]=queue[i+1];
48   }
49   (*queueSize)--;
50   return min;
51
52 }
53
54 int QueueContains(int* checked, int v)
55 {
56   if(checked[v]==-1)
57     return 1;
58   return 0;
59
60 }
61
62 int QueueEmpty(int queueSize)
63 {
64   return queueSize==0;
65 }
66
67 void PrintQueue(int* queue,int queueSize)
68 {
69   int i;
70   for(i=0;i<queueSize;i++)
71   {
72     printf("%d ",queue[i]);
```

```
73     }
74     printf("<queue\n");
75 }
```

Listing 6: queue.h

Συμπέρασμα

Ως συμπέρασμα λοιπόν από την ανάλυση του προβλήματος του περιοδεύοντος πωλητή, μπορούμε να δούμε το πόσο σημαντική είναι η ύπαρξη μιας λύσης. Από τη δημιουργία ηλεκτρονικών κυκλωμάτων μέχρι την επιλογή της συντομότερης διαδρομής για την παράδοση προϊόντων με το ελάχιστο δυνατό κόστος και τη μεγαλύτερη δυνατή ταχύτητα, η λογική του ΠΠΠ αποτελεί αναπόσπαστο κομμάτι της καθημερινότητας. Έτσι, συνεχίζονται οι προσπάθειες για εύρεση αλγορίθμων οι οποίοι προσφέρουν καλύτερα αποτελέσματα με μειωμένη πολυπλοκότητα. Παράλληλα, η χρήση των προσεγγιστικών αλγορίθμων αποτελεί μια σταθερή βάση για την παραγωγή ταχύτατων αποτελεσμάτων με μικρές απώλειες. Επίσης, η ευκολία που υπάρχει για τη δημιουργία ενός προγράμματος με μία προσεγγιστική επίλυση του προβλήματος, καθιστά εφικτή, αν όχι απαραίτητη την ύπαρξη μιας σε κάθε τομέα όπου την αξιοποιεί.

Βιβλιογραφία

[1] Εισαγωγή στους αλγορίθμους, Thomas Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Πανεπιστημιακές εκδόσεις Κρήτης, 2015.

[2] Αλγόριθμοι σε C, Robert Sedgewick, Κλειδάριθμος, 2005.

[3] Graph Theory, 1736-1936, Biggs, N.; Lloyd, E. and Wilson, R., Oxford University Press, 1986.

[4] Alexander Schrijiver 2005, paper, 2005.

[5] The Travelling Salesman Problem: a guided Tour of Combinational Optimization, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B Shmoys, John Wiley & Sons, Chichester, 1985.

[6] Worst-Case Analysis of a new heuristic for the Travelling Salesman Problem, Nicos Christofides, 1976.

[7] The Travelling Salesman Problem and Minimum Spanning Trees: Part II, M. Held & R. M. Karp, 1971

[8] A dynamic approach to sequencing problems, M. Held & R. M. Karp, 1962

[9] Exact Algorithms for NP-Hard Problems: A survey. Combinatorial Optimization, Woeginger, G.J., 2003.

[10] Travelling Salesman Problem: An overview of applications, formulations & solution approaches, Matai Rajesh, Singh Surya Prakash, Murari Lal Mittal, 2010.

Αναφορές

[1] Michael Held, Richard M. Karp. «A Dynamic Programming Approach to Sequencing Problems». Στο: *Journal of the Society for Industrial and Applied Mathematics* 10 (1 1962), 196–210, DOI: 10.1137/0110015.

[2] Michael Held, Richard M. Karp. «The traveling-salesman problem and minimum spanning trees: Part II». Στο: *Mathematical Programming* 1 (1971), 6–25 DOI: 10.1007/BF01584070.

[3] E. L. Lawler κ.ά. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[4] Norman Biggs., Lloyd E. και Wilson R. *Graph Theory 1736-1936*. Oxford University Press, 1986.

[5] Gerhard J. Woeginger. «Exact Algorithms for NP-Hard Problems: A Survey». Στο: *Combinatorial Optimization — Eureka, You Shrink!* 2570 (2003), 185–207 DOI: 10.1007/3-540-36478-1_17.

[6] Rajesh Matai, Surya Singh, Murari Lal Mittal. «Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches». Στο: *Traveling Salesman Problem, Theory and Applications* 1 (2010), 1–24, DOI: 10.5772/12909.

[7] Thomas Cormen κ.ά. «Εισαγωγή στους αλγορίθμους». Στο: Πανεπιστημιακές Εκδόσεις Κρήτης, 2015. Κεφ. 3.1.

[8] Thomas Cormen κ.ά. «Εισαγωγή στους αλγορίθμους». Στο: Πανεπιστημιακές Εκδόσεις Κρήτης, 2015. Κεφ. 34.

[9] Thomas Cormen κ.ά. «Εισαγωγή στους αλγορίθμους». Στο: Πανεπιστημιακές Εκδόσεις Κρήτης, 2015. Κεφ. 34.5.4.

[10] Thomas Cormen κ.ά. «Εισαγωγή στους αλγορίθμους». Στο: Πανεπιστημιακές Εκδόσεις Κρήτης, 2015. Κεφ. 35.

[11] Nicos Christofides. *Worst-Case Analysis of a new heuristic for the Travelling Salesman Problem, http://www.dtic.mil/dtic/tr/fulltext/u2/a025602.pdf*.