ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και υλοποίηση περιβάλλοντος για την εξομοίωση του αλγορίθμου δρομολόγησης DV

Παναγιώτης Κωνσταντόπουλος , ΑΜ. 0454

Επιβλέπων καθηγητής

Τριανταφύλλου Βασίλης

ΑΝΤΙΡΡΙΟ 2016

Περιεχόμενα

Παραρτήματα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1 Εισαγωγη

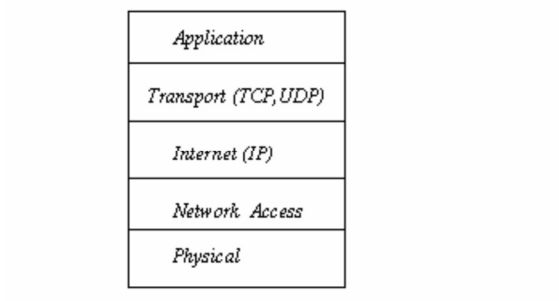
Τα τελευταία χρόνια τα δίκτυα υπολογιστών παίζουν ένα πολύ σημαντικό ρόλο στην καθημερινότητά μας. Με την έννοια δίκτυο υπολογιστών ονομάζουμε ένα σύνολο από υπολογιστές που είναι διασυνδεδεμένοι μεταξύ τους με μέσα μετάδοσης. Τα δίκτυα συνεπώς αποτελούνται από υπολογιστές και μέσα μετάδοσης. Και τα δύο αυτά στοιχεία είναι κρίσιμα και απαραίτητα για την λειτουργία των δικτύων, για το λόγο αυτό τα τελευταία χρόνια έχει συντελεσθεί πολύ σημαντική πρόοδος, που οφείλεται στην ισόρροπη εξέλιξη και των δύο.

Πολλές πτυχές της οικονομίας και κατ' επέκταση και της καθημερινής μας ζωής να βασίζονται στα δίκτυα υπολογιστών λόγω της μεγάλης ανάπτυξής τους. Όλες οι σύγχρονες εταιρίες στηρίζουν την λειτουργία τους σε αυτά (απαραίτητη υποδομή) και τα τελευταία χρόνια έχουν εισχωρήσει στην προσωπική ζωή του ανθρώπου μέσω του Διαδικτύου (Internet).

Η πρόοδος στον τομέα των δικτυακών τεχνολογιών έφερε την επέκταση των τοπικών δικτύων και τη διασύνδεσή τους σε δίκτυα ευρείας περιοχής με αποτέλεσμα το Διαδίκτυο(Internet).

Το Διαδίκτυο για τη μετάδοση της πληροφορίας χρησιμοποιεί την τεχνική της μεταγωγής πακέτου (packet switching). Η τεχνική της μεταγωγής πακέτου έχει ως αποτέλεσμα πακέτα πολλών ροών δεδομένων χρησιμοποιούν στο μέγιστο τους δικτυακούς πόρους και μπορούν να ακολουθήσουν διαφορετικά μονοπάτια στο δίκτυό τους ακόμα και αν έχουν τον ίδιο προορισμό.

Η σχεδίαση των δικτύων έγινε σε επίπεδα όπου το καθένα «χτίζεται» πάνω στο κατώτερό του. Ο στόχος αυτής της σχεδίασης έχει ως στόχο να μειώσει την πολυπλοκότητα. Χαρακτηριστικό μοντέλο σχεδίασης ενός δικτύου αποτελεί το μοντέλο TCP/IP όπου είναι και αυτό που χρησιμοποιείται σήμερα στο Διαδίκτυο. Αυτό αποτελείται από 5 επίπεδα, όπως παρουσιάζεται και στην Εικόνα 1.



Εικόνα Σφάλμα! Δεν έχει καθοριστεί ακολουθία.**: Μοντέλο TCP/IP**

Η αρχιτεκτονική αυτή του Διαδικτύου διαχωρίζει τα τελικά συστήματα από τους κόμβους του δικτύου. Τα πρωτόκολλα της αρχιτεκτονικής του Διαδικτύου είναι γνωστά και ως σουίτα πρωτοκόλλων TCP/IP (TCP/IP protocol suite).

Κύριος στόχος του μοντέλου TCP/IP είναι να μπορεί να συνδέει με διάφανο τρόπο πολλαπλά δίκτυα, τα οποία πιθανώς θα χρησιμοποιούν διαφορετικές τεχνολογίες. Αυτό επιτυγχάνεται με την ανεξαρτησία και τη διάφανη λειτουργία των επιπέδων. Το πιο βασικό επίπεδο του μοντέλου είναι το επίπεδο Διαδικτύου (internet layer) που έχει σαν αποστολή να επιτρέπει στους πελάτες-χρήστες(clients) να εισάγουν πακέτα στο δίκτυο σε οποιοδήποτε σημείο και στη συνέχεια αυτό αναλαμβάνει να τα αποστείλει στον προορισμό. Πολύ πιθανό ο παραλήπτης της συγκεκριμένης πληροφορίας να βρίσκεται σε άλλο δίκτυο από αυτό που βρίσκεται ο αποστολέας, και επίσης πρέπει να αναφέρουμε ότι ο χειρισμός κάθε πακέτου γίνεται ξεχωριστά (τεχνολογία μεταγωγής πακέτου). Συνεπώς αν ένας πομπός εισάγει πολλά πακέτα στη σειρά μπορεί αυτά να ληφθούν από τον παραλήπτη με διαφορετική σειρά και να έχουν ακολουθήσει διαφορετικές διαδρομές προκειμένου να φτάσουν σ' αυτόν. Πρακτικά το επίπεδο Διαδικτύου ορίζει μια προτυποποιημένη μορφή πακέτου (IP packet) και πρωτοκόλλου που λέγεται πρωτόκολλο διαδικτύου IP (Internet Protocol) και έχει ως στόχο να μεταδίδει στον παραλήπτη τα IP πακέτα.

Το επίπεδο Διαδικτύου

Το επίπεδο διαδικτύου φροντίζει για την παράδοση των πακέτων δεδομένων στον καθορισμένο παραλήπτη. Δεν χρησιμοποιείται κάποια σύνδεση μεταξύ του αποστολέα και του παραλήπτη οπότε τα δεδομένα της συγκεκριμένης επικοινωνίας να ακολουθήσουν διαφορετικές διαδρομές το καθένα για να φτάσουν στον προορισμό τους. Στο συγκεκριμένο επίπεδο, το πρωτόκολλο που χρησιμοποιείται είναι το IP (Internet Protocol). Δεν παρέχει το επίπεδο κάποια εγγύηση για την παράδοση των

πακέτων. Τα πακέτα μπορεί να χαθούν λόγω λαθών δικτύου, ή υπερχείλισης των μνήμης των ενδιάμεσων συσκευών (δρομολογητές). Οι δρομολογητές διαχειρίζονται τα πακέτα δεδομένων που βρίσκονται αποθηκευμένα στη μνήμη τους (προσωρινή μνήμη) με τους εξής τρόπους:

- FIFO(First In First Out)
- RED (Random Early Detection)
- Δίκαιη διαχείριση ουρών(Fair queuing mechanism -FQ)

Τα πακέτα δεδομένων στο επίπεδο Διαδικτύου μεταδίδονται με τους εξής τρόπους:

- Unicast: Τα πακέτα δεδομένων μεταδίδονται από τον αποστολέα σε ένα συγκεκριμένο παραλήπτη
- Multicast: Τα πακέτα δεδομένων μεταδίδονται σε μια ομάδα παραληπτών
- Broadcast: Τα πακέτα δεδομένων μεταδίδονται από έναν αποστολέα σε όλους του σταθμούς του δικτύου και χρησιμοποιείται συνήθως στα τοπικά δίκτυα
- Anycast: Τα πακέτα δεδομένων πηγαίνουν σε έναν παραλήπτη από μια συγκεκριμένη ομάδα αποστολέων

Ένα πρωτόκολλο που χρησιμοποιείται για τη μεταφορά δεδομένων όσον αφορά τη διατύπωση προβλημάτων επικοινωνίας είναι το ICMP (Internet Control Message Protocol).

Το επίπεδο Μεταφοράς

Το επίπεδο μεταφοράς έχει ως αποστολή την από άκρο σε άκρο μετάδοση των πακέτων δεδομένων στο τελικό σύστημα. Στο επίπεδο αυτό υπάρχουν δύο πρωτόκολλα μεταφοράς:

- Το TCP (Transmission Control Protocol)
- Το UDP (User Datagram Protocol)

Το πρωτόκολλο TCP παρέχει αξιόπιστη μεταφορά πακέτων και είναι connection oriented ενώ το πρωτόκολλο UDP δεν παρέχει εγγυήσεις για την αξιόπιστη μετάδοση των πακέτων δεδομένων.

Όταν θέλουμε εγγυημένη παράδοση δεδομένων το πρωτόκολλο TCP είναι απαραίτητο. Αντίθετα για εφαρμογές πολυμέσων που έχουν ως στόχο την έγκαιρη

μετάδοση των δεδομένων, για παράδειγμα μετάδοση ποδοσφαιρικός αγώνας, θα στηρίξουν τη μετάδοση των δεδομένων τους στο πρωτόκολλο UDP.

1.2 ΠΡΩΤΟΚΟΛΛΑ ΔΡΟΜΟΛΟΓΗΣΗΣ ΚΑΙ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΕΝΟΣ ΜΟΝΟΠΑΤΙΟΥ ΣΤΟ ΔΙΑΔΙΚΤΥΟ

Βασική λειτουργία των πρωτοκόλλων δρομολόγησης είναι η εύρεση και η επιλογή του καλύτερου μονοπατιού για τα δίκτυα προορισμού με τη χρήση κατάλληλων αλγορίθμων δρομολόγησης (routing algorithms). Ο αλγόριθμος δρομολόγησης δημιουργεί έναν αριθμό, τον οποίο ονομάζουμε τιμή κόστους (metric), για κάθε διαδρομή στο δίκτυο. Η διαδρομή με το μικρότερο κόστος για τον ίδιο προορισμό καταχωρείται τελικά στον πίνακα δρομολόγησης. Ανάλογα με την υλοποίηση, ως κόστος μπορεί να χρησιμοποιηθεί ο αριθμός των δρομολογητών (hop count) που περνά το μήνυμα μέχρι να φτάσει στον προορισμό του, το εύρος ζώνης της γραμμής (bandwidth), η καθυστέρηση (delay), το φορτίο της γραμμής (load) και μια σειρά άλλων παραμέτρων ή ένας συνδυασμός από αυτές. Οι αλγόριθμοι δρομολόγησης χωρίζονται σε δυο κατηγορίες:

- Αλγόριθμοι διανύσματος απόστασης (Distance Vector Algorithms), όπου οι πίνακες δρομολόγησης αποτελούνται από μια σειρά από προορισμούς (vectors) και κόστη τις αποστάσεις (distances) που διανύονται για την προσέγγιση του προορισμού.
- Αλγόριθμοι της κατάστασης της σύνδεσης (Link State Algorithms)

Οι δρομολογητές είναι συνδεδεμένοι ιεραρχικά στο Διαδίκτυο. Οι δρομολογητές οι οποίοι θα είναι συνδεμένοι μαζί με ένα σύνολο δικτύων και θα έχουν ένα κοινό διαχειριστικό σύστημα, το συνολικό σύστημα θα λέγεται αυτόνομο και τα πρωτόκολλα που θα χρησιμοποιούν θα είναι IGP (Interior Gateway Protocol).

Αντίθετα οι δρομολογητές που θα διακινούν πληροφορία μεταξύ των αυτόνομων συστημάτων θα χρησιμοποιούν πρωτόκολλα EGP (Exterior Gateway Protocol).

Τα δύο πιο γνωστά πρωτόκολλα IGP είναι:

- Routing Information Protocol (RIP)
- Open Shortest Path First (OSPF)

Το πρωτόκολλο RIP χρησιμοποιεί πίνακα αποστάσεων (Distance Vector) και βασίζεται στον αλγόριθμο Bellman-Ford. Ένα από τα μειονεκτήματά του είναι η

ανάγκη για περιοδική μετάδοση όλου του πίνακα δρομολόγησης στους διπλανούς κόμβους με αποτέλεσμα σπατάλη εύρους ζώνης και επίσης αργεί να επανέλθει σε περίπτωση αστοχίας του δικτύου. Το πρωτόκολλο RIP είναι κατάλληλο για τη λειτουργία μικρών δικτύων. Στους πίνακες δρομολόγησης που προκύπτουν υπάρχουν πληροφορίες για το δρόμο και το κόστος της απόστασης προς τα δίκτυα προορισμού.

Ως κόστος χρησιμοποιείται ο αριθμός των ενδιάμεσων δρομολογητών μέχρι να φτάσουμε στο δίκτυο προορισμού (hop count). Ο αριθμός των ενδιάμεσων δρομολογητών μέχρι το δίκτυο προορισμού μπορεί να είναι μέχρι 15. Στο πρωτόκολλο RIP οι δρομολογητές περιοδικά, ανακοινώνουν ολόκληρο το περιεχόμενο του πίνακα δρομολόγησής τους, στους άμεσα γειτονικούς δρομολογητές.

Ο πίνακας δρομολόγησης μπορεί να μεταδοθεί κι όταν υπάρξει κάποια αλλαγή στην τοπολογία του δικτύου. Έτσι επιτρέπεται στο κάθε δρομολογητή να βλέπει το δίκτυο του γειτονικού δρομολογητή και να προσθέτει το ανάλογο κόστος στην απόσταση που έχει ήδη προσθέσει ο δεύτερος. Όπως αναφέρθηκε το μειονέκτημα της προσέγγισης αυτής είναι ότι καθώς το δίκτυο μεγαλώνει, ανταλλάσσεται ένα μεγάλο ποσό πληροφορίας ανά τακτά χρονικά διαστήματα, ακόμα κι όταν η τοπολογία του δικτύου δεν έχει αλλάξει, με αποτέλεσμα να περιορίζεται το διαθέσιμο εύρος ζώνης και να αυξάνεται ο χρόνος σύγκλισης. Ως χρόνος σύγκλισης (convergence time), ορίζεται ο χρόνος που περνά μέχρι όλοι οι δρομολογητές να συμφωνήσουν σχετικά με την τοπολογία του δικτύου, από τη στιγμή που θα προκύψει μια αλλαγή. Όταν αλλάζει η τοπολογία του δικτύου, εκτελείται ο αλγόριθμος δρομολόγησης και σταματά η κίνηση των δεδομένων που μεταφέρει ο δρομολογητής προς τα διάφορα interfaces του, γιατί δεν γνωρίζει αν το δίκτυο προορισμού είναι διαθέσιμο ή όχι. Άρα, όσο πιο γρήγορα γίνεται η σύγκλιση τόσο πιο γρήγορα θα μεταφερθούν τελικά τα δεδομένα προς τον προορισμό τους.

Το πρωτόκολλο OSPF είναι τύπου κατάστασης συνδέσμων (Link State) και βασίζεται στο αλγόριθμο Dijkstra. Μπορούν να επανέλθουν πιο γρήγορα σε αποτέλεσμα σε σχέση με τα Distance Vector πρωτόκολλα αλλά είναι πιο πολύπλοκα, χρειάζονται περισσότερη μνήμη και περισσότερη υπολογιστική ισχύ.

Με βάση την υπάρχουσα τεχνολογία, οι δρομολογητές του Διαδικτύου δε μπορούν να ενημερώνουν τις εφαρμογές των τελικών χρηστών για την τρέχουσα κατάσταση του δικτύου. Αυτό έχει ως αποτέλεσμα, οι εφαρμογές να πρέπει να εκτιμήσουν τις συνθήκες του δικτύου με μετρήσεις σε διάφορες παραμέτρους και να

οδηγηθούν σε συμπεράσματα και να προσαρμοστούν ανάλογα. Οι παράμετροι είναι οι εξής:

- Εύρος ζώνης
- Ρυθμός απώλειας δεδομένων
- Διακύμανση καθυστέρησης
- Χρόνος καθυστέρησης μετάδοσης μετά επιστροφής RTT (Round Trip Time)

Οι παράμετροι αυτοί είναι χρήσιμοι για παράδειγμα σε μία μετάδοση ποδοσφαιρικού αγώνα ώστε να διαπιστωθεί αν τυχόν υπάρχει κάποιο πρόβλημα και να ληφθούν αντίστοιχα μέτρα.

1.3 Η ΓΛΩΣΣΑ PYTHON

Η γλώσσα Python δημιουργήθηκε το 1991 από τον Guido van Rossum. Θεωρείται απόγονος της ABC και δανείζεται στοιχεία από τη Modula-3. Διοικείται από το μη κερδοσκοπικό οργανισμό Python Software Foundation και έχει ένα ανοικτό μοντέλο ανάπτυξης. Το φάσμα χρήσης της γλώσσας είναι:

- Google, NASA, YouTube, …
- Πρώτος BitTorrent client, Blender, ArcGIS, Civilization IV, …
- Anaconda, Portrage, standard σε κάθε έκδοση του Linux.
- Γρήγορη προτυποποίηση (prototyping)
- Προγραμματισμός στον Παγκόσμιο Ιστό
- Scripting
- Εκπαίδευση
- Επιστήμη
- Εφαρμογές με γραφική διεπαφή

Επίσης ο κώδικας της Python είναι cross platform δηλαδή υποστηρίζει όλα τα σύγχρονα λειτουργικά συστήματα. Ο κώδικάς της είναι ιδιαίτερα ευανάγνωστος και χρησιμοποιείται τόσο για εκπαιδευτικούς λόγους αλλά και για ανάπτυξη λογισμικού. Πιο συγκεκριμένα τα χαρακτηριστικά της είναι:

- Εύκολη: Η Python είναι μια απλή και μινιμαλιστική γλώσσα. Η ανάγνωσή ενός καλού προγράμματος σε Python είναι σαν το διάβασμα των Αγγλικών και έχει μεγάλη ομοιότητα με ψευδοκώδικα, γεγονός που αποτελεί ένα από τα

πιο ισχυρά σημεία της. Στόχος είναι η συγκέντρωση στη λύση του προβλήματος αντί στην ίδια τη γλώσσα.

- Εκμάθηση: Έχει πολύ απλή σύνταξη και είναι πολύ απλό να ξεκινήσουμε με Python
- Αναγνωσιμότητα: Έχει πολύ καθαρό και αναγνώσιμο συντακτικό
- Συντήρηση: Υποστηρίζει πολύ υψηλού βαθμού συντηρησιμότητα
- Επεκτάσιμη: Μπορει πολύ εύκολα να συνδεθεί με προγράμματα που έχουν γραφτεί σε άλλες γλώσσες, προγράμματα που θα μπορούσαν να εκτελέσουν ειδικού σκοπού λειτουργίες ή να τρέξουν πιο γρήγορα.
- Ανοικτού Κώδικα: Η Python είναι ένα παράδειγμα *ΕΛΛΑΚ* (Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα). Μπορούμε να διανείμουμε αντίγραφα αυτού του λογισμικού, να διαβάσουμε τον πηγαίο κώδικά του, να κάνουμε αλλαγές σ' αυτό και να χρησιμοποιήσετε κομμάτια του σε νέα ελεύθερα προγράμματα.
- Παίζει σχεδόν παντού: Έχει σχεδιαστεί να είναι φορητή, δηλαδή να λειτουργεί σε πολλά διαφορετικά λειτουργικά συστήματα.
- Ενσωματώσιμη: Μπορεί να ενσωματωθεί ως scripting γλώσσα σε προγράμματα άλλων γλωσσών
- Εκτεταμένες βιβλιοθήκες: Η Πρότυπη βιβλιοθήκη της Python υποστηρίζει ένα τεράστιο σε αριθμό σύνολο δυνατοτήτων. Μπορούμε να υλοποιήσουμε θέματα σχετικά με κανονικές εκφράσεις, δημιουργία τεκμηρίωσης, δοκιμές μονάδων, νημάτωση, βάσεις δεδομένων, περιηγητές ιστού, CGI, FTP, email, XML, XML-RPC, HTML, αρχεία WAV, κρυπτογράφηση, γραφικές διεπαφές χρήστη (GUI -graphical user interfaces), Tk, και άλλα πράγματα που εξαρτούνται από το σύστημα. Πέρα από την πρότυπη βιβλιοθήκη, υπάρχουν διάφορες άλλες βιβλιοθήκες υψηλής ποιότητας όπως η wxPython , η Twisted, η Python Imaging Library και πολλές άλλες.
- Ώριμη
- Όχι segmentation faults
- Αυτόματη διαχείριση μνήμης
- Γρήγορη Ανάπτυξη Εφαρμογών
- Διερμηνευόμενη
- Πολύ υψηλού επιπέδου δομές δεδομένων

Κλείνοντας θα σας παρουσιάσουμε το πρώτο πρόγραμμα στις αντίστοιχες τρεις γλώσσες προγραμματισμού.

```
#include <iostream.h>
void main( )
{
  cout << "Hello world!";
}
```

Πρώτο πρόγραμμα σε γλώσσα προγραμματισμού C++.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

Πρώτο πρόγραμμα σε γλώσσα προγραμματισμού Java.

```
print("Hello World!")
```

Πρώτο πρόγραμμα σε γλώσσα προγραμματισμού Python.

Εικόνα Σφάλμα! Δεν έχει καθοριστεί ακολουθία.**: Πρώτο πρόγραμμα σε τρεις διαφορετικές γλώσσες προγραμματισμού**

1.4 ΑΝΤΙΚΕΙΜΕΝΟ ΚΑΙ ΔΟΜΗ ΤΗΣ ΕΡΓΑΣΙΑΣ

Αντικείμενο της παρούσας εργασίας είναι η παρουσίαση της λειτουργίας του πρωτοκόλλου δρομολόγησης OSPF. Πιο συγκεκριμένα, θα σχεδιαστεί και θα υλοποιηθεί περιβάλλον για την εξομοίωση της δρομολόγησης χρησιμοποιώντας Distance Vector πρωτόκολλο. Ο αλγόριθμος που θα χρησιμοποιηθεί θα είναι ο Bellman-Ford για να υπολογίσουμε τις διαδρομές (paths).

Αρχικά θα γίνει παρουσίαση αλγορίθμων δρομολόγησης για δίκτυα υπολογιστών. Θα γίνει περιγραφή των Distance Vector πρωτοκόλλων και θα δοθεί έμφαση στην χρήση του αλγορίθμου Bellman-Ford. Θα γίνει παρουσίαση του τρόπου υπολογισμού της απόστασης και της κατεύθυνσης δρομολόγησης της ζητούμενης

πληροφορίας καθώς και των προβλημάτων που υπάρχουν αλγοριθμικά (count-to-infinity-problem). Επίσης θα μιλήσουμε και τις προτεινόμενες λύσεις για συγκεκριμένα πρωτόκολλα όπως το RIP και το OSPF.

Στο Τρίτο Κεφάλαιο θα γίνει εισαγωγή στη δρομολόγηση δικτύων υπολογιστών. Πώς γίνεται η δρομολόγηση σ' ένα δίκτυο υπολογιστών, πως χρησιμοποιούμε τη διευθυνσιοδότηση IP, ποιες είναι οι αρχιτεκτονικές στοίβας πρωτοκόλλων και τοπολογίας δρομολογητή και τέλος θα γίνει αναφορά στις τεχνολογίες επικοινωνίας.

Στο Τέταρτο Κεφάλαιο θα παρουσιάσουμε τη γλώσσα προγραμματισμού Python. Μετά την εισαγωγή θα αναφερθούμε στα αντικείμενα, στις εκφράσεις που υποστηρίζει η γλώσσα, στον έλεγχο ροής και στις συναρτήσεις. Σκοπός είναι να δούμε όλα τα στοιχεία της γλώσσας περιληπτικά τα οποία θα μας χρησιμεύσουν κατά τη διάρκεια της ανάπτυξης του περιβάλλοντος εξομοίωσης.

Το Πέμπτο Κεφάλαιο της εργασίας θα έχει ως αντικείμενο το σχεδιασμό και υλοποίηση του αλγορίθμου Bellman-ford στη γλώσσα προγραμματισμού Python. Θα παρουσιαστούν οι υλοποιήσεις και τα παραδείγματα που θα εκτελέσουμε.

Τέλος η παρούσα εργασία ολοκληρώνεται με τα Κεφάλαιο Έξι όπου βρίσκονται τα συμπεράσματα.

ΚΕΦΑΛΑΙΟ 2

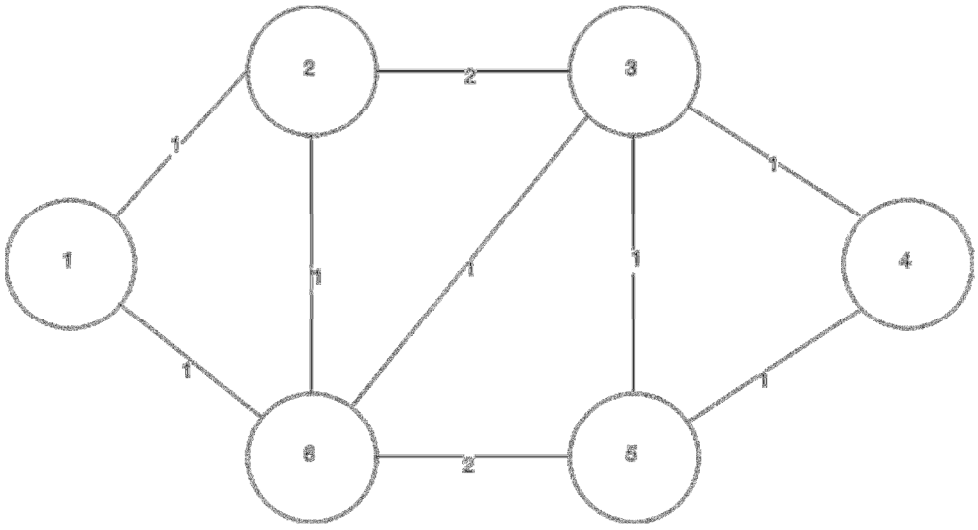
Αλγόριθμοι δρομολόγησης

2.1 Εισαγωγη

Γενικά ένα δίκτυο επικοινωνίας αποτελείται από κόμβους (nodes) και συνδέσμους (links). Ανάλογα με το είδος του δικτύου οι κόμβοι έχουν διαφορετικά ονόματα. Σε ένα δίκτυο IP οι κόμβοι ονομάζονται δρομολογητές, σε ένα τηλεφωνικό τελικό γραφείο (end office). Σε ένα οπτικό δίκτυο ο κόμβος ονομάζεται οπτικός μεταγωγέας (optical switch).

Ένας σύνδεσμος ενώνει δύο κόμβους. Ένας σύνδεσμος που ενώνει δύο δρομολογητές ονομάζεται IP σύνδεσμος και το τέλος του συνδέσμου που βγαίνει από ένα δρομολογητή ονομάζεται διεπαφή. Σε ένα τηλεφωνικό δίκτυο ένας σύνδεσμος ονομάζεται κορμός (trunk).

Ένα δίκτυο επικοινωνιών μεταφέρει πληροφορία, όπου η πληροφορία κινείται από ένα αρχικό κόμβο σε ένα τελικό κόμβο. Τυπικά ονομάζουμε τον αρχικό κόμβο ως τον κόμβο πηγή, και το τελικό κόμβο ως τον κόμβο προορισμού. Έστω ότι έχουμε το δίκτυο της Εικόνας 1. Έστω ότι έχουμε μεταφορά πληροφορίας από τον κόμβο 1 στον κόμβο 3. Ο κόμβος 1 είναι ο κόμβος πηγή ενώ ο κόμβος 3 είναι ο κόμβος προορισμού



Εικόνα 1: Δίκτυο επικοινωνιών

Μία βασική απαίτηση είναι να δρομολογήσουμε την πληροφορία από τον κόμβο πηγή στον κόμβο προορισμού. Για το λόγο αυτό πρέπει να καθορίσουμε τη διαδρομή.

Δρομολόγηση ορίζουμε τη διαδικασία εύρεσης της διαδρομής που πρέπει να ακολουθήσει ένα πακέτο πληροφορίας για να φτάσει στο προορισμό του.

Εάν καθορίσουμε τη διαδρομή χειροκίνητα τότε η **δρομολόγηση θα είναι στατική**. Γενικά όμως είναι προτιμητέο τη διαδρομή να την καθορίζει ένας αλγόριθμος δρομολόγησης. Ο στόχος του αλγορίθμου δρομολόγησης καθορίζεται από δίκτυο επικοινωνιών και από την υπηρεσία που θέλει να παρέχει καθώς και συγκεκριμένες έξτρα απαιτήσεις.

Ο αλγόριθμος δρομολόγησης θα πρέπει να έχει τα εξής χαρακτηριστικά:

Απλότητα: Ο αλγόριθμος πρέπει να είναι απλός – να περιέχει σαφείς και κατανοητούς κανόνες που διέπουν την λειτουργία του.

Ορθότητα: Ο αλγόριθμος πρέπει να επιλύει σωστά το πρόβλημα της δρομολόγησης.

Ανθεκτικότητα: Θα πρέπει ο αλγόριθμος να είναι σε θέση να αντιμετωπίζει αλλαγές στην τοπολογία του δικτύου (π.χ. όταν κάποιος κόμβος-γραμμή σταματήσει να λειτουργεί).

Δικαιοσύνη: Πρέπει πακέτα από διαφορετικές συνδέσεις να αντιμετωπίζονται με δίκαιο τρόπο (πακέτα μιας σύνδεσης να καθυστερούν σε σχέση με τα πακέτα κάποιας άλλης σύνδεσης).

Βελτιστοποίηση: Καλύτερη δυνατή αξιοποίηση των πόρων του δικτύου => μεγιστοποίηση της συνολικής κίνησης στο δίκτυο.

Οι βασικές λειτουργίες ενός αλγόριθμου δρομολόγησης είναι οι εξής:

- Επιλογή διαδρομής μεταφοράς δεδομένων από πηγή -> προορισμό.
- Παράδοση πακέτων πληροφορίας στον προορισμό από την διαδρομή που επιλέχθηκε.

Ένα ακόμη θέμα που πρέπει να αναφέρουμε είναι για το αν θα έχουμε δρομολόγηση αυτοδύναμων πακέτων ή εικονικών κυκλωμάτων δικτύου δεδομένων. Πιο συγκεκριμένα, για τα **Αυτοδύναμα Πακέτα** (Datagrams) η απόφαση δρομολόγησης γίνεται για κάθε πακέτο ενώ για τα **Εικονικά Κυκλώματα** (Virtual

Circuits) έχουμε μία απόφαση δρομολόγησης για όλα τα πακέτα της ίδιας συνόδου (session).

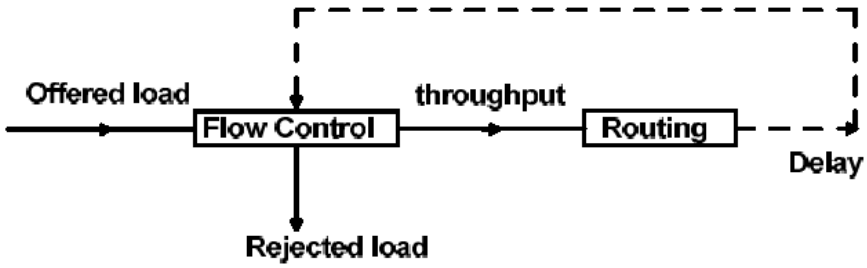
Ο κάθε αλγόριθμος δρομολόγησης για να μπορέσει να υπολογίσει το βέλτιστο μονοπάτι για την πορεία του εκάστοτε πακέτου ή των πακέτων μιας συνόδου, θα πρέπει να στηριχθεί σε κάποιες μετρικές. Οι μετρικές μπορεί να είναι η μέτρηση της χρήσης των συνδέσμων, το πλήθος των ενδιάμεσων αλμάτων, η ταχύτητα του μονοπατιού, το μήκος του, η απώλεια πακέτων, η καθυστέρηση, η αξιοπιστία του μονοπατιού, το εύρος του μονοπατιού, ο ρυθμός διεκπαιρεωτικής ικανότητας, το μέγεθος της ουράς και το κόστος. Οι μετρικές αυτές μπορούν να χρησιμοποιηθούν ως μία έννοια που θα την ονομάσουμε κόστος. **Οι πιο συχνά χρησιμοποιούμενες μετρικές** είναι:

- το **μήκος του μονοπατιού** (path's length), αφορά στο συνολικό μήκος του μονοπατιού που διασχίζει ένα πακέτο από την πηγή προέλευσής του έως τη μηχανή προορισμού.
- η **καθυστέρηση** (delay), αφορά στο χρόνο που απαιτείται για ένα πακέτο που στέλνεται από κάποια πηγή να φτάσει στον προορισμό του.
- το **εύρος του μονοπατιού** (path bandwidth), που αφορά στο μέγεθος των δεδομένων που μπορούν να διέλθουν από ένα σημείο του μονοπατιού σε ένα άλλο σε μια δεδομένη χρονική περίοδο.
- η **απώλεια πακέτων** (packet loss), πιο συχνά αναφερόμενη ως ποσοστό απώλειας πακέτων (Packet Loss Ratio).
- το **κόστος (cost),** όπου εδώ αναφέρεται ως το επικοινωνιακό κόστος και αφορά στο χρηματικό κόστος της χρήσης των συνδέσεων του δικτύου.
- το **μέγεθος της ουράς** (queue length), που αφορά στον αποθηκευτικό χώρο όπου εισερχόμενα πακέτα παραμένουν μέχρι να προωθηθούν σε κάποιο εξερχόμενο σύνδεσμο. Αν η ουρά είναι γεμάτη, τότε πακέτα που φτάνουν δεν μπορούν να αποθηκευτούν και χάνονται (packet loss). Αν η ουρά είναι άδεια τότε ο δρομολογητής μπορεί να ικανοποιήσει μεγαλύτερα ποσά κίνησης (throughput).

Για να μετρήσουμε την απόδοση του αλγορίθμου έχουμε τις εξής μετρικές:

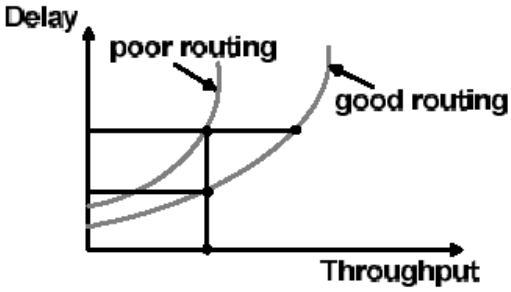
- Ρυθμαπόδοση (Throughput) – 'ποσότητα' της εξυπηρέτησης, δηλαδή ο ρυθμός μετάδοσης που επιτυγχάνεται

- Μέση καθυστέρηση πακέτου – ‘ποιότητα’ της εξυπηρέτησης, δηλαδή ο χρόνος (μέσος όρος) που χρειάζεται για να γίνει η δρομολόγηση των πακέτων στο προορισμό



Εικόνα 2: Ποσότητα και ποιότητα εξυπηρέτησης μεταφοράς δεδομένων

Καθώς ο αλγόριθμος δρομολόγησης (routing) επιτυγχάνει στο να κρατάει την καθυστέρηση χαμηλά, ο έλεγχος ροής (flow control) επιτρέπει περισσότερη κυκλοφορία στο δίκτυο όπως παρουσιάζεται στην εικόνα 2.



Εικόνα 3: Σχέση ρυθμαπόδοσης και καθυστέρησης

Όπως βλέπουμε και στην εικόνα 3 οι καλοί αλγόριθμοι δρομολόγησης έχουν υψηλότερη ρυθμαπόδοση για την ίδια καθυστέρηση, μικρότερη καθυστέρηση για δεδομένη ρυθμαπόδοση.

Ανάλογα με τους στόχους που θέλουμε να επιτύχουμε οι αλγόριθμοι χωρίζονται σε αλγορίθμους που λαμβάνουν υπόψη **τον χρήστη (user-oriented)** και αλγορίθμους που **λαμβάνουν υπόψη το δίκτυο (network-oriented)**. Αλγόριθμοι με βάση τον χρήστη σημαίνει ότι το δίκτυο παρέχει καλή υπηρεσία σε κάθε χρήστη ώστε η μεταφορά της πληροφορίας να γίνεται γρήγορα από την πηγή στον προορισμό για τον συγκεκριμένο χρήστη, αυτό όμως χωρίς να επιβαρύνει τους άλλους χρήστες. Οι αλγόριθμοι που λαμβάνουν υπόψη το δίκτυο παρέχουν αποτελεσματική και δίκαια δρομολόγηση για τους περισσότερους χρήστες και όχι την καλύτερη δρομολόγηση σε έναν συγκεκριμένο.

Μία άλλη κατηγοριοποίηση των αλγορίθμων δρομολόγησης είναι σε προσαρμοστικούς και μη προσαρμοστικούς. Οι προσαρμοστικοί αλγόριθμοι ονομάζονται και δυναμικοί ενώ η μη προσαρμοστικοί ως στατικοί. Μια δεύτερη βασική κατηγοριοποίηση σε γενικούς και αποκεντρικοποιημένους. Τέλος μία τρίτη κατηγοριοποίηση είναι οι αλγόριθμοι κατάστασης συνδέσμων και οι αλγόριθμοι διανύσματος απόστασης.

Όσον αφορά την πρώτη κατηγοριοποίηση που μόλις αναφέρθηκε έχουμε τους εξής:

- **Στατικοί** αλγόριθμοι: Χρησιμοποιούν σταθερές διαδρομές για τη μεταφορά των δεδομένων. Οι διαδρομές είναι ανεξάρτητες από την κίνηση του δικτύου και αλλάζουν μόνο όταν ένας κόμβος ή γραμμή τεθούν εκτός λειτουργίας. Οι στατικοί αλγόριθμοι δεν λαμβάνουν υπόψη κάποια από τις μετρικές που αναφέρθηκαν προηγουμένως.
- Αλγόριθμοι **προσαρμοζόμενης δρομολόγησης ή δυναμικοί αλγόριθμοι**: τροποποιούν τις διαδρομές ανάλογα με το φόρτο των γραμμών του δικτύου. Για να αποφασίσουν τις διαδρομές, οι αλγόριθμοι αυτοί εκτιμούν-μετρούν έμμεσα την κίνηση του δικτύου (και επικοινωνούν με τους αλγόριθμους απομακρυσμένων κόμβων). Επίσης η αλλαγή των δρομολογίων αλλάζει με δυναμικούς ρυθμούς και περιοδικά συνήθως.

Σχετικά με τη δεύτερη κατηγοριοποίηση χωρίζονται οι αλγόριθμοι στους συγκεντρωτικούς ή γενικούς και στους αποκεντρικοποιημένους ή κατανεμημένους.

- **Συγκεντρωτικοί** αλγόριθμοι: Οι αποφάσεις δρομολόγησης λαμβάνονται σε ένα κεντρικό κόμβο ο οποίος γνωρίζει την κατάσταση του δικτύου και οι πίνακες δρομολόγησης που θα διατηρεί έχουν μεγάλο μέγεθος. Ο κόμβος ρέπει να έχει δυνατότητα αποθήκευσης δεδομένων, ισχυρό επεξεργαστή για αναζήτηση στους πίνακες. Σε αυτή την κατηγορία ανήκει ο αλγόριθμος Dijkstra.
- **Κατανεμημένοι** αλγόριθμοι: Οι αποφάσεις δρομολόγησης λαμβάνονται μεταξύ των κόμβων του δικτύου οι οποίοι επικοινωνούν μεταξύ τους και ανταλλάσσουν πληροφορίες (π.χ. φορτίο γραμμής). Σε αυτή τη κατηγορία ανήκει ο αλγόριθμος Bellman-Ford.

Στη συνέχεια θα παρουσιάσουμε δύο πολύ σημαντικούς αλγορίθμους που έχουν σημαντική επίπτωση στα δίκτυα δεδομένων και συγκεκριμένα στη δρομολόγηση IP δικτύων. Οι αλγόριθμοι είναι:

1. Ο αλγόριθμος Bellman-ford
2. Ο αλγόριθμος Dijkstra

Και οι δύο είναι αλγόριθμοι δρομολόγησης συντομότερης διαδρομής, αλγόριθμοι δηλαδή που έχουν ως στόχο να βρουν τη συντομότερη διαδρομή από τον κόμβο πηγή στον κόμβο προορισμού. Ένας απλός τρόπος να κατανοήσουμε τη συντομότερη διαδρομή είναι λαμβάνοντας υπόψη την απόσταση, ποια είναι η πιο σύντομη διαδρομή μεταξύ δύο πόλεων. Η πιο σύντομη διαδρομή μπορεί να έχει να κάνει με τα χιλιόμετρα που έχουμε να διανύσουμε αλλά και με διαφορετική μετρική όπως ο χρόνος που χρειάζεται να δαπανήσουμε.

Μπορούμε να μην ασχοληθούμε με την μονάδα μέτρησης αλλά να έχουμε έναν αλγόριθμο που δουλεύει ανεξάρτητα από τη μονάδα μέτρησης και λαμβάνει μια γενική μέτρηση της απόστασης για κάθε σύνδεσμο του δικτύου. Στα δίκτυα επικοινωνιών μία γενική μέτρηση της απόστασης ονομάζεται κόστος ή κόστος συνδέσμου.

Για παράδειγμα στην εικόνα 1 η συντομότερη διαδρομή από τον κόμβο 1 στον κόμβο 3 είναι η 1-6-3 διαδρομή με συνολικό κόστος 2. Έτσι για τον υπολογισμό της συντομότερης διαδρομής η ιδιότητα της πρόσθεσης είναι γενικά αυτή που χρησιμοποιείται για την κατασκευή τη συνολικής απόστασης της διαδρομής με την πρόσθεση του κόστους ενός συνδέσμου, με την πρόσθεση του κόστους του επόμενου μέχρι να έχουν ληφθεί υπόψη όλα.

Στη συνέχεια θα γίνει αναφορά και στην σχέση ενός δικτύου και ενός γράφου. Ένα δίκτυο μπορεί να εκφραστεί ως ένας γράφος με την αντιστοίχηση κάθε κόμβου σε μία μοναδική κορυφή (vertex) στο γράφο όπου οι σύνδεσμοι μεταξύ των διαδικτυακών κόμβων απεικονίζονται από τις άκρες (edges) που συνδέουν τις αντίστοιχες κορυφές.

2.2 Ο ΑΛΓΟΡΙΘΜΟΣ BELLMAN-FORD

Ο αλγόριθμος Bellman-Ford ανήκει στην κατηγορία των δυναμικών αλγορίθμων που βασίζονται στο Διάνυσμα-Απόσταση(Distance-Vector).

Κάθε δρομολογητής του δικτύου διατηρεί έναν πίνακα με μία εγγραφή που αφορά τους άλλους κόμβους του δικτύου. Η εγγραφή αυτή περιέχει την καλύτερη γνωστή απόσταση προς κάθε προορισμό και τη διαδρομή προς αυτόν. Η ενημέρωση των πινάκων γίνεται με ανταλλαγή πληροφοριών μεταξύ γειτόνων του δικτύου. Η συντομότερη διαδρομή ή καλύτερη γνωστή διαδρομή βασίζεται:

- στο πλήθος των ενδιάμεσων αλμάτων
- στη χρονική καθυστέρηση
- στον αριθμό των πακέτων που βρίσκονται στις ουρές κατά μήκος της διαδρομής

Κάθε κόμβος γνωρίζει μόνο τους άμεσους γείτονές του και το κόστος που εμπλέκεται ώστε η πληροφορία να φθάσει σε αυτούς. Οι πληροφορίες αυτές δηλαδή:

- ο κατάλογος με τους προορισμούς
- το κόστος για να φτάσει η πληροφορία σε αυτούς
- και το επόμενο βήμα(hop)

δημιουργούν τον πίνακα αποστάσεων ή αλλιώς τον πίνακα δρομολόγησης. Ανά σύντομους περιόδους, κάθε κόμβος ενημερώνει κάθε γειτονικό του κόμβο για τη τρέχουσα οπτική του όσον αφορά τους προορισμούς που γνωρίζει. Οι γειτονικοί κόμβοι επεξεργάζονται αυτή τη πληροφορία, τη συγκρίνουν με την δικιά τους και εφόσον υπάρχει διαφορά, πιο πρόσφατη πληροφορία δηλαδή, την αλλάζουν. Καθώς περνάει ο χρόνος, όλοι οι κόμβοι ενημερώνονται για τον καλύτερο επόμενο βήμα για όλους τους κόμβους και για το καλύτερο συνολικό κόστος προορισμού. Ο αλγόριθμος συγκλίνει όταν μετά από μια επανάληψη δεν υπάρχουν αλλαγές στους πίνακες δρομολόγησης.

Ο αλγόριθμος Bellman-Ford βρίσκει τα κόστη των ελάχιστων διαδρομών από μια δεδομένη πηγή s προς κάθε κορυφή v. Ο ψευδοκώδικας παρουσιάζεται στη συνέχεια.

```
for all v∈V
  d[v] = ∞;
d[s]=0;

for (i=1; i<|V|; i++)
   for all edges (u,v)∈E
      if (d[v] > d[u]+w(u,v))
         d[v]= d[u] + w(u,v);

for all edges (u,v)∈E
   if (d[v] > d[u]+ w(u,v))
      NO SOLUTION
```

Εικόνα 4: Ο αλγόριθμος Bellman-Ford

2.3 Ο ΑΛΓΟΡΙΘΜΟΣ DIJKSTRA

Οι αλγόριθμοι κατάστασης συνδέσμων βασίζονται σε κάποια βασικά στοιχεία. Πιο συγκεκριμένα, ο κάθε κόμβος χρησιμοποιεί σαν αρχικά δεδομένα ένα χάρτη του δικτύου με την μορφή γράφου. Για να παραχθεί αυτός, κάθε κόμβος πλημμυρίζει ολόκληρο το δίκτυο με πληροφορίες σχετικά με το με ποιούς άλλους κόμβους μπορεί να συνδεθεί, εν συνεχεία κάθε κόμβος συγκεντρώνει όλες αυτές τις πληροφορίες και σχηματίζει έναν χάρτη. Χρησιμοποιώντας αυτό το χάρτη, κάθε δρομολογητής αποφασίζει ανεξάρτητα την καλύτερη διαδρομή από τον εαυτό του προς κάθε άλλο κόμβο.

Ο αλγόριθμος που χρησιμοποιείται για να επιλεγεί η βέλτιστη διαδρομή, ο αλγόριθμος του Dijkstra, το κάνει αυτό δημιουργώντας μια δομή δεδομένων, ένα δέντρο, με τον τρέχοντα κόμβο σαν ρίζα του δέντρου, που περιέχει όλους τους υπόλοιπους κόμβους του δικτύου. Ξεκινάει με ένα δέντρο που περιέχει μόνο τον εαυτό του. Μετά, έναν ένα τη φορά, από το σύνολο των κόμβων που δεν έχουν προστεθεί στο δέντρο, προσθέτει τον κόμβο που έχει το μικρότερο κόστος για να φτάσει έναν γειτονικό κόμβο ο οποίος ήδη υπάρχει στο δέντρο. Αυτό συνεχίζεται μέχρις ότου όλοι οι κόμβοι να υπάρχουν στο δέντρο.

Αυτό το δέντρο εξυπηρετεί στην κατασκευή του πίνακα δρομολόγησης του κάθε κόμβου, δείχνοντας το καλύτερο επόμενο βήμα (hop), για να φτάσει από τον εαυτό του σε οποιονδήποτε άλλο κόμβο στο δίκτυο.

Ο αλγόριθμος δρομολόγησης του Dijkstra ανήκει στους αλγόριθμους συντομότερης διαδρομής. Αναπαριστώντας το δίκτυο ως γράφημα όπου οι κόμβοι αντιπροσωπεύουν τους δρομολογητές και οι ακμές τους συνδέσμους, το μήκος της διαδρομής μπορεί να αφορά σε:

- πλήθος των αλμάτων μεταξύ κόμβων
- γεωγραφική απόσταση σε χιλιόμετρα

Σε όλους τους κόμβους του δικτύου τα κόστη των συνδέσεων είναι γνωστά. Η βασική ιδέα του αλγορίθμου του Dijkstra είναι η σήμανση των κόμβων του δικτύου ως μόνιμους ή προσωρινούς. Μία άλλη σημαντική ιδιότητα του αλγορίθμου είναι ότι υπολογίζει τις συντομότερες διαδρομές για όλους τους προορισμούς από μία πηγή και όχι μόνο για ένα συγκεκριμένο ζεύγος πηγής-προορισμού. Ο αλγόριθμος δουλεύει όταν τα κόστη των συνδέσεων είναι μη- αρνητικά.

Η περιγραφή του είναι η εξής:

- Έστω ότι θέλουμε να βρούμε τα βραχύτερα μονοπάτια από κάποιο κόμβο s προς όλους τους υπόλοιπους κόμβους σε κάποιο γράφο G.
- Ο αλγόριθμος αυτός διατηρεί ένα πίνακα S από κορυφές όπου αποθηκεύει τις κορυφές του G, για τις οποίες το μήκος του βραχύτερου μονοπατιού έχει υπολογισθεί.
- Επίσης διατηρεί ένα πίνακα d όπου για κάθε κόμβο Β του γράφου φυλάει την ανά πάσα στιγμή μικρότερη απόσταση του κόμβου Β από τον κόμβο s την οποία γνωρίζει.
- Αρχικά S=□ και d [|V|] = ∞.
- Ο αλγόριθμος επανελημένα διαλέγει την κοντινότερη κορυφή Β προς τον s , που δεν έχει μέχρι στιγμής επεξεργασθεί (δηλαδή Β∈V-S), και ελέγχει αν για οποιοδήποτε γείτονα Γ της Β χρήση της ακμής (Β,Γ) μπορεί να δημιουργήσει βραχύτερο μονοπάτι προς την Γ.
- Χρησιμοποιεί ουρά προτεραιότητας, Q, για αποθήκευση κορυφών, όπου η προτεραιότητα δίνεται από το d[v].
- S είναι το σύνολο των κορυφών i για τα οποία d[i]=δ(s,i).

Ο αλγόριθμος σε ψευδοκώδικα είναι ο εξής:

```
heap Q;
for all v∈V
    d[v]=∞;
d[s]=0;

S=∅;
Q=V;
while (Q ≠ ∅){
    u=DeleteMin(Q);
    S=S∪{u};
    για κάθε γείτονα v του u
        if d[v]>d[u]+w(u,v)
            d[v]=d[u]+w(u,v);
```

Εικόνα 5: Ο αλγόριθμος Dijkstra

2.4 ΣΥΓΚΡΙΣΗ ΤΩΝ ΔΥΟ ΑΛΓΟΡΙΘΜΩΝ

Οι δύο αλγόριθμοι που παρουσιάστηκαν έχουν αντίστοιχα κάποια πλεονεκτήματα και κάποια μειονεκτήματα.

Όσον αφορά την πολυπλοκότητα του μηνύματος ο αλγόριθμος Bellman-Ford ανταλλάσσει μηνύματα πληροφορίας μόνο με τους γειτονικούς κόμβους ενώ ο αλγόριθμος Dijkstra στέλνει O(nE) μηνύματα σε ένα δίκτυο με n κόμβους και E συνδέσεις μεταξύ των κόμβων. Υπάρχει μεν σημαντική διαφορά αλλά δεν υπάρχει πλεονέκτημα λόγω του προβλήματος μέτρησης στο άπειρο(count-to-infinity problem).

Όσον αφορά τον χρόνο σύγκλισης, ο αλγόριθμος Bellman-Ford ποικίλει μιας και μπορεί να δημιουργήσει βρόχους δρομολόγησης που θα οδηγήσουν στο πρόβλημα μέτρησης στο άπειρο. Ο αλγόριθμος Dijkstra απαιτεί χρόνο $O(n^2)$ και μπορεί να έχει διακυμάνσεις.

Τέλος, σχετικά με το τι συμβαίνει στην περίπτωση σφάλματος ή κατάρρευσης ενός κόμβου (δρομολογητή) ο αλγόριθμος Dijkstra μπορεί να μεταδώσει λάθος κόστος μίας σύνδεσης αλλά το σφάλμα ενδέχεται να μη φτάσει μακριά μιας και κάθε κόμβος έχει το δικό του πίνακα αποστάσεων (πίνακα δρομολόγησης). Αντίθετα ο αλγόριθμος Bellman-Ford μπορεί να διαδίδει λάθος κόστος διαδρομής (μονοπατιού).

Επίσης, ένα ακόμη πρόβλημα είναι ότι ο πίνακας δρομολόγησης κάθε κόμβου χρησιμοποιείται και από άλλους κόμβους, οπότε έχουμε διάδοση του σφάλματος σε όλο το δίκτυο και όχι μόνο τοπικά.

ΚΕΦΑΛΑΙΟ 3

Δρομολόγηση δικτύων

3.1 Εισαγωγη

Ως δίκτυο θεωρείται το σύνολο των μέσων που επιτυγχάνουν να γεφυρώσουν ενσύρματα η ασύρματα δύο ή περισσότερα τερματικά με σκοπό την μετάδοση οποιασδήποτε μορφής δεδομένων. Ένα κλασσικό παράδειγμα ενός τέτοιου δικτύου είναι το παραδοσιακό ταχυδρομικό δίκτυο

Όταν τα μέσα αυτά είναι ηλεκτρονικοί υπολογιστές, τότε μιλάμε για ένα δίκτυο υπολογιστών (computer networks) . Τα δίκτυα υπολογιστών ανήκουν στη γενικότερη κατηγορία των τηλεπικοινωνιακών δικτύων (telecommunication networks), δηλαδή σε εκείνα τα κατανεμημένα συστήματα που επιτρέπουν στους χρήστες τους να μεταβιβάζουν ή να ανταλλάσσουν πληροφορίες. Τα πλέον γνωστά και εκτεταμένα τηλεπικοινωνιακά δίκτυα είναι το τηλεφωνικό δίκτυο και το δίκτυο της τηλεόρασης.

Ένα δίκτυο υπολογιστών είναι ένα σύνολο από αυτόνομους ή μη αυτόνομους διασυνδεδεμένους υπολογιστές. Βασικός σκοπός της ύπαρξης των δικτύων είναι ο διαμερισμός των πόρων (υλικό, λογισμικό) του συστήματος και η ανταλλαγή πληροφοριών κάθε μορφής (προγράμματα, αρχεία, δεδομένα). Το πιο γνωστό και πολυχρησιμοποιημένο δίκτυο επικοινωνίας της σύγχρονης εποχής είναι το Διαδίκτυο

3.2 Δρομολογηση ενος δικτυου

Η διαδικασία αποστολής πληροφοριών μέσω του Διαδικτύου είναι εξαιρετικά σύνθετη. Σε γενικές γραμμές, όταν στέλνονται πληροφορίες μέσω του Διαδικτύου, αρχικά η πληροφορία τεμαχίζεται σε πακέτα κατάλληλου μεγέθους. Τα πακέτα διαθέτουν εκτός των άλλων και μια διεύθυνση IP του υπολογιστή δέκτη που είναι απαραίτητη για την μοναδική αναγνώριση του προορισμού. Ο υπολογιστής-πηγή,

στέλνει τα εν λόγω πακέτα στο τοπικό δίκτυο ή στον παροχέα υπηρεσιών Internet.

Από εκεί τα πακέτα ακλουθώντας μια διαδρομή, ταξιδεύουν μέσα από πολλαπλά επίπεδα δικτύων, υπολογιστών και τηλεπικοινωνιακών γραμμών πριν φτάσουν στον τελικό τους προορισμό υπολογιστή-δέκτη.

Ορισμός: Δρομολόγηση δικτύου (Network routing), είναι η διαδικασία εύρεσης της αποτελεσματικότερης, γρηγορότερης και με τις λιγότερες απώλειες διαδρομής, που πρέπει να ακολουθήσει ένα πακέτο πληροφορίας μέσα στο δίκτυο, για να πάει από ένα σημείο Α σε ένα σημείο Β.

Η δρομολόγηση δεν είναι πάντα εύκολη αφού ένα σύνθετο δίκτυο (π.χ Διαδίκτυο) μπορεί να διαθέτει πολλές εναλλακτικές διαδρομές και τα πακέτα μπορεί να χρειαστεί να διασχίσουν πολλούς κόμβους – σημεία διασταύρωσης διαδρομών – μέσα στο δίκτυο μέχρι να φτάσουν στο προορισμό τους. Οι κόμβοι αυτοί, στο Διαδίκτυο είναι γνωστοί ως δρομολογητές (routers).

Οι λειτουργίες ενός δρομολογητή είναι να:

- Διαβάζει τη διεύθυνση προορισμού που σημειώνεται σε ένα εισερχόμενο πακέτο IP,
- Συμβουλεύεται τις εσωτερικές πληροφορίες του, προκειμένου να προσδιορίσει μια εξερχόμενη σύνδεση με την οποία το πακέτο θα προωθηθεί,
- Προωθεί το πακέτο.

Ο τρόπος διευθυνσιοδότησης του Διαδικτύου, αναφέρεται ως διευθυνσιοδότηση Πρωτοκόλλου Διαδικτύου (Internet Protocol (IP) addressing).

Σύντομα: Μια διεύθυνση IP ορίζεται από δυο μέρη:

- Το netid: προσδιορίζει ένα συνεχόμενο μπλοκ διευθύνσεων
- Το hostid: προσδιορίζει μια διεύθυνση δέκτη.

Όπως κάθε σύστημα παροχής υπηρεσιών, έτσι και το Διαδίκτυο βασίζεται σε ένα μοντέλο παράδοσης δεδομένων, γνωστό ως TCP / IP (Transmission Control Protocol / Internet Protocol). Το σύστημα παράδοσης στηρίζεται σε ένα μοντέλο στο οποίο το TCP είναι υπεύθυνο για την αξιόπιστη παράδοση των πακέτων πληροφοριών, ενώ το IP είναι υπεύθυνο για την δρομολόγηση των πακέτων, χρησιμοποιώντας το μηχανισμό διευθυνσιοδότηση IP, ο οποίος περιγράφεται αναλυτικά στο επόμενο κεφάλαιο.

Μια σύνδεση δικτύου (network link) συνδέει δύο δρομολογητές και περιορίζεται από την ποσότητα δεδομένων που μπορεί να μεταφέρει ανά μονάδα χρόνου, δηλαδή από το εύρος ζώνης (bandwidth) ή την χωρητικότητα (capacity) της σύνδεσης.

Ένα δίκτυο λοιπόν, έχει κίνηση (traffic) στις συνδέσεις δηλαδή πακέτα πληροφορίας που παράγονται από διάφορες εφαρμογές και που μέσω των δρομολογητών μεταφέρονται στον τελικό προορισμό τους.

Εάν για οποιονδήποτε λόγο η κίνηση μέσα στο δίκτυο ξαφνικά αυξηθεί, τα πακέτα που δημιουργούνται μπορούν είτε να μπουν στις ουρές που διαθέτουν οι δρομολογητές είτε να πεταχτούν. Κάθε δρομολογητής διατηρεί μια πεπερασμένη ποσότητα αποθηκευτικού χώρου, γνωστή ως ενδιάμεση μνήμη (buffer) για την προσωρινή αποθήκευση των πακέτων που βρίσκονται στην ουρά (backlogged packets), Εάν η κίνηση είναι μεγάλη, υπάρχει περίπτωση ο αποθηκευτικός χώρος να εξαντληθεί. Δεδομένου ότι η βασική αρχή του πρωτοκόλλου TCP / IP επιτρέπει τη δυνατότητα ενός πακέτου IP να μην παραδοθεί ή να πέσει καθ 'οδόν, ο πεπερασμένος αποθηκευτικός χώρος του δρομολογητή δεν αποτελεί πρόβλημα.

Για να θεωρηθεί μια παράδοση αποτελεσματική, είναι επιθυμητό να έχει καμία ή ελάχιστη απώλεια πακέτων κατά τη διάρκεια της μετακίνησης. Αυτό οφείλεται στο γεγονός ότι μια αξιόπιστη παράδοση βασίζεται στην αρχή της αναμετάδοσης και της αναγνώρισης και κάθε απόρριψη πακέτου θα σήμαινε αύξηση της καθυστέρησης λόγω της ανάγκης για αναμετάδοση του. Επιπλέον, κατά τη μεταφορά των πακέτων, είναι πιθανό, το περιεχόμενο που περικλείεται σε κάποιο πακέτο δεδομένων να καταστραφεί για παράδειγμα εξ αιτίας διακοπής του ηλεκτρικού ρεύματος. Αυτό έχει ως αποτέλεσμα την αλλοίωση (garbling) του πακέτου. Για την διαδικασία επικοινωνίας, είτε το πακέτο είναι αλλοιωμένο, είτε έχει απορριφθεί, επιβαρύνει το ίδιο.

Έτσι, για την αποτελεσματική παράδοση των πακέτων, χρειάζονται:

- Δρομολογητές με αρκετό αποθηκευτικό χώρο,
- Συνδέσεις με επαρκές εύρος ζώνης,
- Πραγματική μετάδοση με ελάχιστα σφάλματα (για την ελαχιστοποίηση των αλλοιωμένων πακέτων)
- Αποδοτικότητα δρομολογητών στην μεταγωγή πακέτων στην κατάλληλη εξερχόμενη σύνδεση.

Για ελαχιστοποιηθεί ο χρόνος μεταγωγής σε έναν δρομολογητή, χρειάζονται αποτελεσματικοί μηχανισμοί που να μπορούν να αναζητούν μια διεύθυνση, να εντοπίζουν την κατάλληλη εξερχόμενη σύνδεση (κατεύθυνση) και να επεξεργάζονται το πακέτο γρήγορα, έτσι ώστε η καθυστέρηση επεξεργασίας να είναι όσο το δυνατό συντομότερη.

Υπάρχει, ωστόσο, μια άλλη σημαντική φάση που λειτουργεί παράλληλα με τη διαδικασία αναζήτησης σε έναν δρομολογητή. Αυτή είναι η ανανέωση του πίνακα στο δρομολογητή, γνωστό ως πίνακα δρομολόγησης (routing table), που περιέχει το αναγνωριστικό για τον επόμενο δρομολογητή, που είναι γνωστό ως επόμενο άλμα (next hop), για ένα δεδομένο netid προορισμό.

Ο πίνακας δρομολόγησης στην πραγματικότητα ενημερώνεται εκ των προτέρων. Για να ενημερωθεί ένας τέτοιος πίνακας, ο δρομολογητής πρέπει να αποθηκεύσει όλα τα netid που έχει μάθει μέχρι εκείνη τη στιγμή. Δεύτερον, εάν ένας σύνδεσμος προς τα κάτω έχει πέσει ή έχει συμφόρηση ή ένα netid δεν είναι προσβάσιμο για κάποιο λόγο, ο δρομολογητής πρέπει να το γνωρίζει έτσι ώστε μια εναλλακτική διαδρομή να μπορεί να προσδιοριστεί το συντομότερο δυνατό. Αυτό σημαίνει ότι απαιτείται ένας μηχανισμός αντιμετώπισης για την συμφόρηση της επικοινωνίας (communicating congestion), την αποτυχία ενός συνδέσμου ή την αδυναμία πρόσβασης ενός netid. Ο μηχανισμός αυτός είναι γνωστός ως μηχανισμός πρωτοκόλλου δρομολόγησης (routing protocol). Οι πληροφορίες που αντλήθηκαν μέσω ενός πρωτοκόλλου δρομολόγησης είναι που χρησιμοποιούνται εκ των προτέρων για την παραγωγή του πίνακα δρομολόγησης.

Αν υπάρξουν νέες πληροφορίες σχετικά με την κατάσταση των συνδέσεων ή κόμβων, ή την προσβασιμότητα του netid μέσω ενός πρωτοκόλλου δρομολόγησης, τότε ένας αλγόριθμος δρομολόγησης (routing algorithm) καλείται στο δρομολογητή για να καθορίσει την καλύτερο δυνατό επόμενο άλμα (hop) για κάθε προορισμό netid, προκειμένου να ενημερωθεί ο πίνακας δρομολόγησης. Για την αποτελεσματική επεξεργασία των πακέτων, υπάρχει και ένας άλλος πίνακας, γνωστός ως πίνακας προώθησης (forwarding table), που προέρχεται από τον πίνακα δρομολόγησης και προσδιορίζει τις εξερχόμενες διεπαφές σύνδεσης. Ο πίνακας προώθησης είναι επίσης γνωστός και ως Forwarding Information Base (FIB).

Ο αλγόριθμος δρομολόγησης χρειάζεται να λάβει υπόψη του, παράγοντες για μια σύνδεση, όπως η καθυστέρηση διάσχισης της σύνδεσης, ή το διαθέσιμο εύρος ζώνης της, προκειμένου να καθορίσει την καλύτερη δυνατή διαδρομή μεταξύ ενός

αριθμού πιθανών διαδρομών. Εάν μια σύνδεση κατά μήκος μιας διαδρομής δεν διαθέτει επαρκές εύρος ζώνης, μπορεί να έχει συμφόρηση ή καθυστέρηση. Για να ελαχιστοποιηθεί η καθυστέρηση, εκτελείται μια σημαντική λειτουργία, που ονομάζεται μηχανική της κίνησης (traffic engineering). Η μηχανική της κίνησης ασχολείται με τους τρόπους βελτίωσης της λειτουργικής απόδοσης του δικτύου και προσδιορίζει τις διαδικασίες ή τους ελέγχους που πρέπει να τεθούν σε εφαρμογή εκ των προτέρων προκειμένου να επιτευχθεί η καλή απόδοση του δικτύου.

Μια σημαντική έννοια που σχετίζεται με τα δίκτυα και κυρίως με την δρομολόγηση , είναι η αρχιτεκτονική (architecture).

Ο όρος αρχιτεκτονική ενός δικτύου (architecting a network) χρησιμοποιείται ποικιλοτρόπως:

- Για ένα πρωτόκολλο: οι διάφορες λειτουργίες χωρίζονται έτσι ώστε κάθε λειτουργία να μπορεί να εκτελεστεί ξεχωριστά και μια λειτουργία να μπορεί να εξαρτάται από μια άλλη μέσω κάποιας καλά καθορισμένης σχέσης.
- Για έναν δρομολογητή: ο όρος αναφέρεται στο πώς είναι οργανωμένος εσωτερικά για μια ποικιλία λειτουργιών, από το χειρισμό του πρωτοκόλλου δρομολόγησης μέχρι την επεξεργασία των πακέτων.
- Για ένα δίκτυο: αναφέρεται στην αρχιτεκτονική της τοπολογίας του δικτύου. Πώς πρέπει να οργανωθεί, που πρέπει να βρίσκονται οι δρομολογητές, ποιο πρέπει να είναι το εύρος ζώνης των συνδέσεων για την αποτελεσματική διαχείριση της κίνησης, κ.τ.λ.

Συνοψίζοντας, η δρομολόγηση ενός δικτύου έχει ένα ευρύ πεδίο εφαρμογής που περιλαμβάνει την διευθυνσιοδότηση των αλγόριθμων δρομολόγησης, τα πρωτόκολλα δρομολόγησης, και τις διάφορες πτυχές της αρχιτεκτονικής για την αποτελεσματική δρομολόγηση.

3.3 ΔΙΕΥΘΥΝΣΙΟΔΟΤΗΣΗ IP

Η αποστολή δεδομένων σε οποιονδήποτε δέκτη στο Διαδίκτυο, προϋποθέτει την αναγνώριση του με μοναδικό τρόπο ανάμεσα σε όλους τους υπόλοιπους δέκτες. Για το λόγο αυτό, υπάρχει η ανάγκη για ένα παγκόσμιο σύστημα διευθυνσιοδότησης στο οποίο δεν θα υπάρχουν δύο δέκτες με την ίδια διεύθυνση.

Μια διεύθυνση IP που εκχωρείται σε έναν δέκτη έχει μήκος 32 bits και πρέπει να είναι μοναδική. Η γνωστή και ως IPv4 διευθυνσιοδότηση, είναι γραμμένη σε μορφή bit, από τα αριστερά προς τα δεξιά, όπου το πιο αριστερό bit θεωρείται το πιο σημαντικό bit.

Η ιεραρχία στην διευθυνσιοδότηση IP, αντικατοπτρίζεται μέσα από δύο τμήματα:

- Το τμήμα δικτύου
- Το τμήμα δέκτη

που αναφέρεται ως ζεύγος (netid, hostid).

Το Διαδίκτυο δηλαδή, μπορεί να θεωρηθεί ως η διασύνδεση (interconnection) των δικτύων που προσδιορίζονται μέσω των netids, όπου κάθε netid διατηρεί μια συλλογή από δέκτες. Το τμήμα δικτύου (netid) προσδιορίζει το δίκτυο στο οποίο είναι συνδεδεμένος ο δέκτης και το τμήμα δέκτη (hostid) προσδιορίζει έναν δέκτη (host) στο δίκτυο αυτό. Το τμήμα δικτύου επίσης αναφέρεται και ως πρόθεμα IP (IP prefix). Όλες οι δέκτες που είναι συνδεδεμένοι στο ίδιο δίκτυο μοιράζονται το ίδιο τμήμα δικτύου στις IP διευθύνσεις τους, αλλά έχουν μοναδικό τμήμα δέκτη.

Ο χώρος διευθύνσεων IP αρχικά χωρίζονταν σε τρεις διαφορετικές κατηγορίες, κλάση A (Class A), κλάση B (Class B), και κλάση Γ (Class C) για τα δίκτυα και τους δέκτες. Κάθε τάξη διακρίνεται από τα πρώτα λίγα bits της 32-bit διεύθυνσης.

Για καλύτερη αναγνωσιμότητα, οι διευθύνσεις IP εκφράζονται ως τέσσερις δεκαδικοί αριθμοί, με μια τελεία μεταξύ τους. Αυτή η μορφή ονομάζεται διακεκομμένη δεκαδική σημειογραφία (dotted decimal notation). Ο συμβολισμός χωρίζει την 32-bit διεύθυνση IP σε 4 ομάδες των 8 bits και προσδιορίζει την τιμή κάθε ομάδας ξεχωριστά ως έναν δεκαδικό αριθμό που χωρίζεται από τελείες. Λόγω των 8-bit σημείων διακοπής, μπορούν να υπάρξουν το πολύ 256 (= 2^8) δεκαδικές τιμές για κάθε ομάδα. Και επειδή το μηδέν είναι μεταβιβάσιμη τιμή, καμία δεκαδική τιμή δεν μπορεί να είναι μεγαλύτερη από το 255.

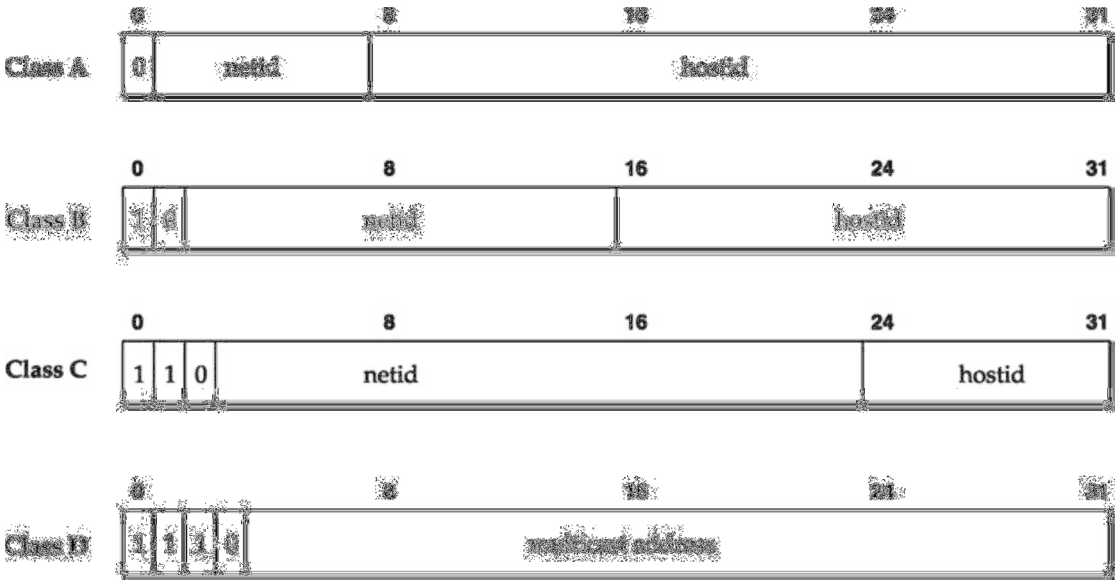
Κάθε διεύθυνση κλάσης A έχει το πρώτο bit στο “0” και ακολουθείται από 7 bits για το τμήμα του δικτύου, με μέγιστο αποτέλεσμα 128 (= 2^7) δίκτυα. Η υπόλοιπη διεύθυνση αποτελεί το τμήμα δέκτη μήκους 24-bit. Έτσι, η κλάση A υποστηρίζει έως 2^{24}-2 δέκτες ανά δίκτυο. Το 2 αφαιρείται επειδή τα 0s και 1s στο τμήμα δέκτη μιας διεύθυνσης της κλάσης A δεν μπορούν να ανατεθούν σε μεμονωμένους δέκτες.

Κάθε διεύθυνση δικτύου κλάσης B έχει τα πρώτα δύο bits στα “10” και ακολουθείται από 14-bit για το τμήμα δικτύου, τα οποία στη συνέχεια ακολουθούνται από τα 16-bit για το τμήμα δέκτη. Ο μέγιστος αριθμός δικτύων που μπορούν να οριστούν είναι 2^{14} με έως 2^{16}-2 δέκτες ανά δίκτυο.

Κάθε διεύθυνση δικτύου κλάσης C έχει τα πρώτα τρία bits στο “110” και ακολουθείται από 21-bit για το τμήμα δικτύου, με τα τελευταία 8 bits να προσδιορίζουν τα τμήμα δέκτη. Η κλάση Γ παρέχει υποστήριξη για το μέγιστο 2^{21} (= 2.097.152) δίκτυα με έως 254 (2^8 - 2) δέκτες ανά δίκτυο.

Οι τρεις αυτές κατηγορίες διευθύνσεων χρησιμοποιούνται για μονής διανομής (unicasting) στο Διαδίκτυο, η οποία είναι κατάλληλη για επικοινωνία δέκτη με δέκτη (host-to-host). Υπάρχει ακόμη μια κλάση διευθύνσεων IP, που είναι γνωστή ως κλάση D (Class D) και χρησιμοποιείται για πολλαπλή διανομή (multicasting) στο διαδίκτυο.

Κάθε διεύθυνση δικτύου κλάσης D έχει τα πρώτα τέσσερα bits στο “1110” για να δείξει ότι είναι μια διεύθυνση πολλαπλής διανομής. Ένας δέκτης μπορεί να χρησιμοποιήσει μια διεύθυνση πολλαπλής διανομής ως διεύθυνση προορισμού για ένα πακέτο που παράγεται για να δείξει ότι το πακέτο προορίζεται για οποιονδήποτε δέκτη στο Internet. Προκειμένου κάθε δέκτης να κάνει χρήση αυτής της δυνατότητας, θα πρέπει να χρησιμοποιεί έναν άλλο μηχανισμό για να συντονιστεί σε αυτή τη διεύθυνση.



Εικόνα 1: Η μορφή διευθύνσεων IP των τεσσάρων κατηγοριών.

Η αρίθμηση των διεύθυνση IP δεν είναι γεωγραφική. Ένα πλεονέκτημα μιας μη γεωγραφικής (nongeographic) διεύθυνσης είναι ότι ένας οργανισμός που του έχει

εκχωρηθεί ένα μπλοκ διευθύνσεων IP μπορεί να κρατήσει το ίδιο μπλοκ διευθύνσεων ακόμη και αν μετακινηθεί σε διαφορετική τοποθεσία ή εάν χρησιμοποιήσει διαφορετικό πάροχο για τη σύνδεση του στο Διαδίκτυο. Για αυτό η IP διευθυνσιοδότηση για το τμήμα δικτύου είναι επίπεδη και δεν υπάρχει ιεραρχία.

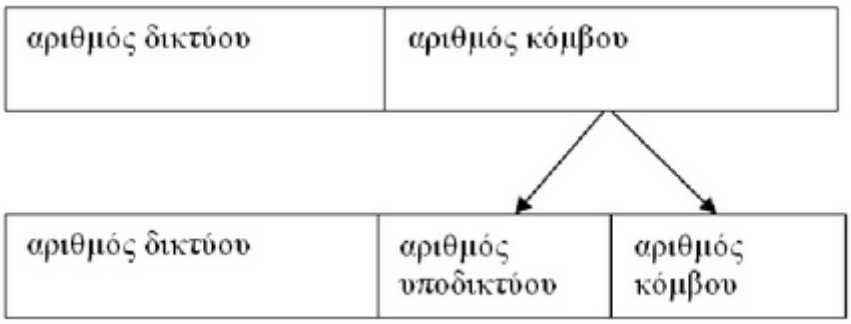
Όσο η δημοτικότητα του Διαδικτύου αυξάνεται, όλο και περισσότερα μειονεκτήματα του τρόπου διευθυνσιοδότησης προκύπτουν. Οι κυριότερες ανησυχίες έχουν να κάνουν με το ρυθμό με τον οποίο τα μπλοκ διευθύνσεων IP που προσδιορίζουν τα netids εξαντλούνται, ιδίως όταν έγινε αναγκαίο να ξεκινήσει η ανάθεση netids της κλασης C.

Όπως ειπώθηκε παραπάνω, τα IP netids είναι μη γεωγραφικά. Έτσι, όλες οι έγκυρες netids απαιτείται να αναγράφονται στους δρομολογητές του Διαδικτύου, μαζί με την απερχόμενη σύνδεση, έτσι ώστε τα πακέτα να μπορούν να διαβιβάζονται σωστά. Αν τώρα σκεφτεί κανείς, όλα τα netids της κλασης C που ανατίθενται, τότε υπάρχουν πάνω από 2.000.000 εγγραφές που θα πρέπει να αναγράφονται σε ένα δρομολογητή. Κανένας από τους υπάρχοντες δρομολογητές δεν μπορεί να χειριστεί αυτόν τον αριθμό εγγραφών χωρίς σοβαρή αργοπορία στην επεξεργασία των πακέτων.

Για το λόγο αυτό άρχισε να αναπτύσσεται από τις αρχές της δεκαετίας του 1990 η χωρίς κλάσεις (*classless*) διευθυνσιοδότηση.

Για να γίνει κατανοητή η χωρίς κλάσεις (*classless*) διευθυνσιοδότηση πρέπει πρώτα να περιγραφεί η έννοια του υποδικτύου και της μάσκας δικτύου (subnetting/netmask).

Ορισμός: Κάθε δίκτυο των κλάσεων A, B, ή C διαμοιράζεται σε μικρότερα δίκτυα που καλούνται υποδίκτυα (sub-networks)



Εικόνα 2: Η μετατροπή των διευθύνσεων IP για την χωρίς κλάσεις διευθυνσιοδότηση. Χρήση αριθμού υποδικτύου.

Ορισμός: Το πλήθος των bits που χρησιμοποιούνται για τον ορισμό του αριθμού υποδικτύου δηλώνονται στην μάσκα υποδικτύου (subnet mask).

Η μάσκα υποδικτύου έχει την ίδια μορφή και αναπαράσταση με την διεύθυνση IP και ακολουθεί τον εξής κανόνα:

- Τα bits που αντιστοιχούν στα πεδία δικτύου και υποδικτύου (σημαντικά bits) λαμβάνουν την τιμή "1". Συμβολίζεται και ως "/ όριο σημαντικών bits".
- Τα υπόλοιπα bits που αντιστοιχούν στο πεδίο κόμβου λαμβάνουν την τιμή "0".

Κατηγορία	Πλήθος bits δικτύων/ κόμβου	**Προεπιλεγμένη Subnet Mask**			
A	8/24	11111111 (255)	00000000 (0)	00000000 (0)	00000000 (0)
B	16/16	11111111 (255)	11111111 (255)	00000000 (0)	00000000 (0)
C	24/8	11111111 (255)	11111111 (255)	11111111 (255)	00000000 (0)

Εικόνα 3: Οι μάσκες υποδικτύου για κάθε μια από τις κατηγορίες διευθύνσεων IP.

Για να προκύψει η διεύθυνση δικτύου μιας κλασης εκτελείται η λογική πράξη "AND" ανάμεσα στην διεύθυνση του δέκτη και στην μάσκα υποδικτύου.

Παράδειγμα: Έστω η διεύθυνση της κλασης C, 192.168.40.3, και η μάσκα δικτύου της, 11111111.11111111.11111111.00000000. Τότε η διεύθυνση δικτύου της κλασης C προκύπτει από την πράξη:

```
        11000000 10101000 00101000 00000011   → 192.168.40.3
AND     11111111 11111111 11111111 00000000   → netmask (/24)
        11000000 10101000 00101000 00000000   → 192.168.40.0
```

Εικόνα 4: Η πράξη του λογικού AND για την εύρεση της διεύθυνσης ενός δικτυού κατηγορίας C.

Το Classless Interdomain Routing (CIDR) χρησιμοποιεί μια ρητή μάσκα δικτύου με ένα μπλοκ διευθύνσεων IPv4 για τον εντοπισμό ενός δικτύου. Ένα πλεονέκτημα της ρητής μάσκας είναι ότι ένα μπλοκ διευθύνσεων μπορεί να εκχωρηθεί με οποιαδήποτε όριο bit. Επιπλέον, η εκχώρηση διευθύνσεων κλασης C για δίκτυα που μπορούν να εμφανιστούν στο παγκόσμιο πίνακα δρομολόγησης μπορεί να αποφεύγεται ή να ελαχιστοποιείται.

3.4 ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΣΤΟΙΒΑΣ ΠΡΩΤΟΚΟΛΛΩΝ

Οι αρχιτεκτονικές καλύπτουν πολλές διαφορετικές πτυχές των δικτυακών περιβαλλόντων. Η δρομολόγηση δικτύου πρέπει να αντιστοιχεί σε κάθε μία από τις ακόλουθες αρχιτεκτονικές.

- Αρχιτεκτονική Υπηρεσιών (Service Architecture)
- Αρχιτεκτονική Στοίβας Πρωτοκόλλου (Protocol Stack Architecture)
- Αρχιτεκτονική Δρομολογητή (Router Architecture)
- Αρχιτεκτονική Τοπολογίας Δικτύου (Network Topology Architecture)
- Αρχιτεκτονική Διαχείρισης Δικτύου (Network Management Architecture)

Ορισμένες από τις παραπάνω αρχιτεκτονικές θεωρούνται κρίσιμης σημασίας για θέματα δρομολόγησης. Αυτές είναι η Αρχιτεκτονική Στοίβας Πρωτοκόλλου που θα παρουσιαστεί σε αυτή την ενότητα και η Αρχιτεκτονική Δρομολογητή που ακολουθεί στην επόμενη ενότητα.

Δυο είναι τα μοντέλα αυτής της αρχιτεκτονικής

- Το μοντέλο αναφοράς OSI (Διασύνδεση Ανοικτών Συστημάτων - Open Systems Interconnections)
- Το μοντέλο αναφοράς IP

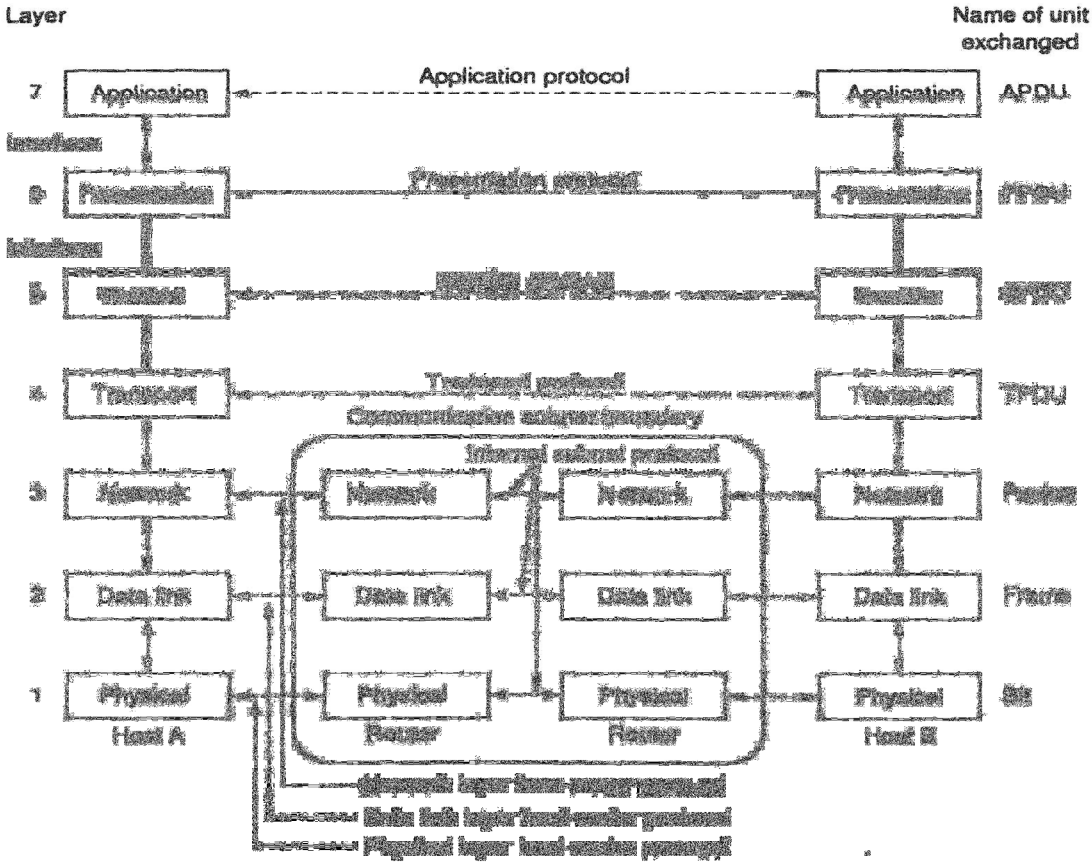
Το μοντέλο αναφοράς OSI αναπτύχθηκε στη δεκαετία του 1980 και σκοπός του ήταν να παρουσιάσει ένα γενικό μοντέλο αναφοράς για το πώς η αρχιτεκτονική ενός δικτύου ηλεκτρονικών υπολογιστών πρέπει να διαιρείται λειτουργικά. Ως μέρος του OSI, τα επόμενα χρόνια αναπτύχθηκαν διάφορα πρωτόκολλα.

Το μοντέλο OSI αποτελείται από επτά επίπεδα. Οι αρχές που εφαρμόστηκαν προκειμένου να δημιουργηθούν αυτά τα επτά επίπεδα συνοψίζονται ως εξής:

1. Όπου χρειάζεται μια διαφορετική λογική αφαίρεση πρέπει να δημιουργείται ένα επίπεδο.
2. Κάθε επίπεδο πρέπει να εκτελεί μια σαφώς καθορισμένη λειτουργία
3. Η λειτουργία κάθε επιπέδου πρέπει να επιλέγεται με στόχο τον καθορισμό διεθνώς τυποποιημένων πρωτοκόλλων
4. Τα σύνορα των επιπέδων πρέπει να επιλέγονται έτσι ώστε να ελαχιστοποιείται η ροη πληροφοριών μέσω της διασύνδεσης των επιπέδων

5. Το πλήθος των επιπέδων πρέπει να είναι αρκετά μεγάλο έτσι ώστε να μην χρειάζεται να ανακατεύονται χωρίς λόγο διαφορετικές λειτουργίες στο ίδιο επίπεδο και ταυτόχρονα αρκετά μικρό έτσι ώστε η αρχιτεκτονική να μη γίνεται άβολη.

Το μοντέλο OSI δεν αποτελεί από μόνο του μια αρχιτεκτονική δικτύου, επειδή δεν προσδιορίζει τις ακριβείς υπηρεσίες και πρωτόκολλα που πρέπει να χρησιμοποιούνται σε κάθε επίπεδο. Το μοντέλο απλώς ορίζει τι πρέπει να κάνει το κάθε επίπεδο.



Εικόνα 5: Η αρχιτεκτονική του μοντέλου αναφοράς OSI.

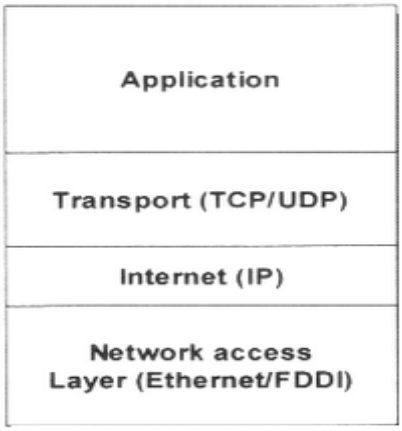
Πιο συγκεκριμένα:

- Στο φυσικό επίπεδο (physical layer) καθορίζονται οι ηλεκτρικές, μηχανικές και λειτουργικές προδιαγραφές για τη μετάδοση των δεδομένων πάνω από ένα φυσικό μέσο.
- Το επίπεδο συνδέσμου μετάδοσης δεδομένων (data link layer) παρέχει την αξιόπιστη μεταφορά των δεδομένων πάνω από τα φυσικά μέσα.
- Στο επίπεδο δικτύου (network layer) καθορίζεται ο τρόπος δρομολόγησης των πακέτων από τον αποστολέα στον παραλήπτη και ο έλεγχος συμφόρησης του

- Στο επίπεδο μεταφοράς (transport layer) υλοποιείται το κανάλι επικοινωνίας μεταξύ των τερματικών κόμβων, μέσω του οποίου θα μεταβιβάζονται αξιόπιστα τα μηνύματά τους.
- Στο επίπεδο συνδιάλεξης (session layer) διενεργούνται όλες οι απαραίτητες λειτουργίες για την εγκαθίδρυση, την επίβλεψη και τον τερματισμό των συνόδων (sessions) μεταξύ των τελικών εφαρμογών.
- Το επίπεδο παρουσίασης (presentation layer) ασχολείται με την αναπαράσταση των δεδομένων και έχει ως κύρια λειτουργία την εξασφάλιση της αναγνωσιμότητάς τους, ακόμα και μεταξύ κόμβων που χρησιμοποιούν διαφορετικές μορφές αναπαράστασης της πληροφορίας.
- Το επίπεδο εφαρμογής (application layer) παρέχει ένα σύνολο δικτυακών υπηρεσιών στις τελικές εφαρμογές των χρηστών.

Το μοντέλο αναφοράς IP που ως βασικό στόχο έχει την ικανότητα διασύνδεσης πολλών δικτύων με διαφανή τρόπο, αποτελείται από τέσσερα επίπεδα:

- Διασύνδεσης μεταξύ υπολογιστή υπηρεσίας και δικτύου(network interface)
- Διαδικτύου (IP layer)
- Μεταφοράς (transport layer)
- Εφαρμογών (application layer)



Εικόνα 6: Η αρχιτεκτονική του μοντέλου αναφοράς IP.

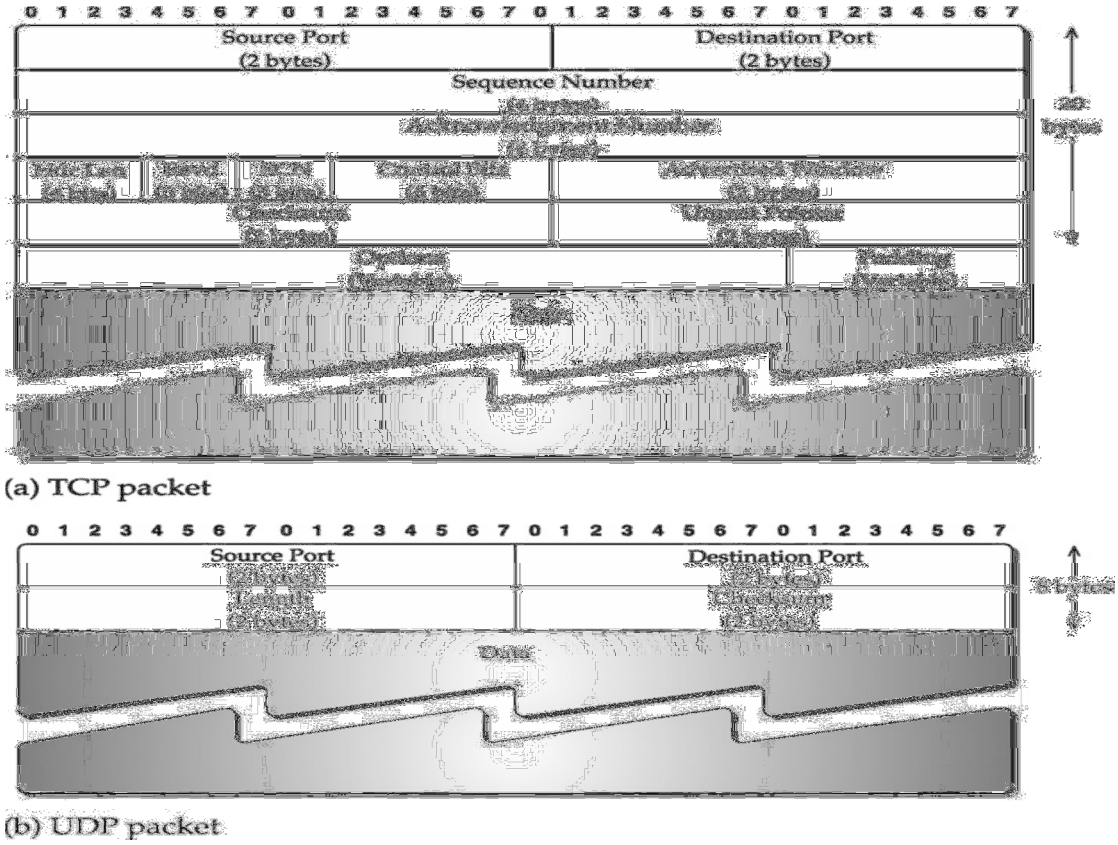
Πιο συγκεκριμένα:

- Το επίπεδο διασύνδεσης μεταξύ υπολογιστή υπηρεσίας και δικτύου(network interface) περιλαμβάνει εκείνα τα πρωτόκολλα επικοινωνίας που έχουν ως κύρια λειτουργία τη μετάδοση πακέτων μεταξύ συγκεκριμένων κόμβων του δικτύου. Οι κόμβοι επικοινωνούν μεταξύ τους είτε με σύνδεσμο σημείου με

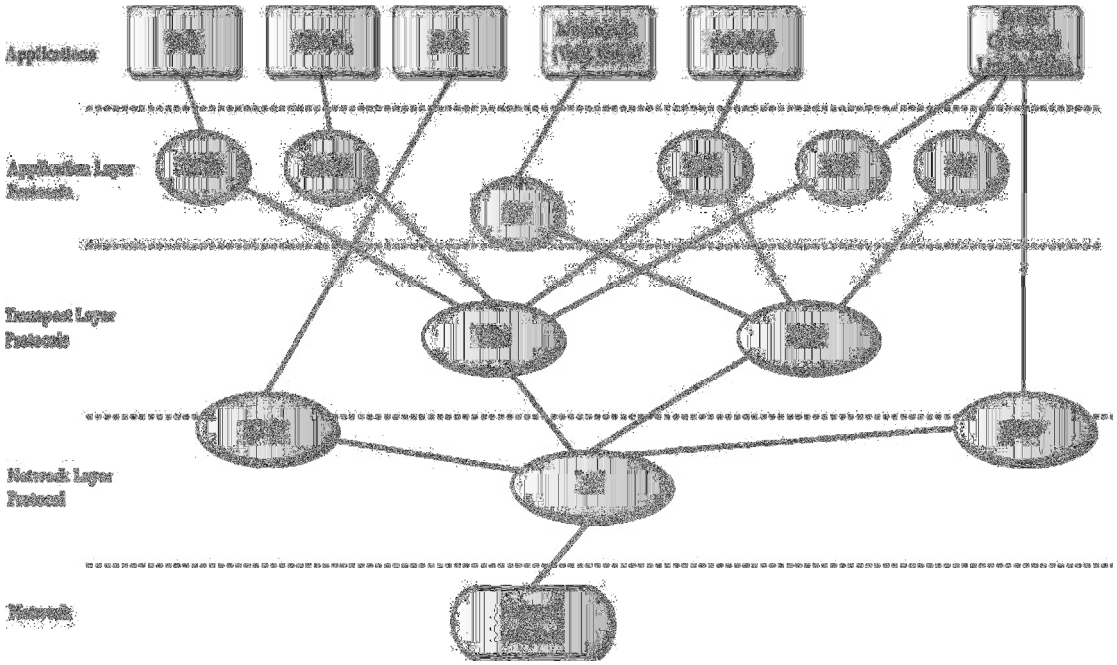
σημείο είτε μέσω κάποιου συνδέσμου πολλαπλής πρόσβασης. Μια μεγάλη ποικιλία πρωτοκόλλων έχουν αναπτυχθεί και χρησιμοποιούνται ευρέως για τη μετάδοση πακέτων πάνω από τους συνδέσμους (Ethernet, Token Ring, FDDI, PPP)

- Το επίπεδο διαδικτύου (IP layer) περιλαμβάνει μόνο το πρωτόκολλο IP (Internet Protocol), το οποίο ελέγχει τη διευθυνσιοδότηση των κόμβων του δικτύου και τη δρομολόγηση των πακέτων. Το IP, ωστόσο, δεν μπορεί να εγγυηθεί ότι θα παραδώσει όλα τα πακέτα δεδομένων στον προορισμό τους ή ότι θα τα παραδώσει με τη σωστή σειρά.
- Το επίπεδο μεταφοράς (transport layer) περιλαμβάνει τα πρωτόκολλα TCP (Transmission Control Protocol – πρωτόκολλο ελέγχου μετάδοσης) και UDP (User Datagram Protocol – πρωτόκολλο αυτοδύναμων πακέτων χρήστη), τα οποία ελέγχουν την ανταλλαγή των πακέτων μεταξύ των τερματικών κόμβων, ρυθμίζοντας έτσι την από άκρο σε άκρο επικοινωνία. Το πρωτόκολλο UDP παραδίδει ένα πακέτο στον προορισμό του, διενεργώντας μόνο έναν απλό έλεγχο για να διαπιστωθεί αν το πακέτο έχει υποστεί αλλοίωση κατά τη μεταφορά του μέσω του δικτύου. Αν έχει φθαρεί, τότε απορρίπτεται, αλλιώς προωθείται για περαιτέρω επεξεργασία. Αντίθετα, το πρωτόκολλο TCP διενεργεί πιο σύνθετους ελέγχους σφαλμάτων. Συγκεκριμένα, εάν ένα πακέτο διαπιστωθεί ότι έχει φθαρεί, τότε ζητείται από τον αποστολέα κόμβο η επανεκπομπή του πακέτου. Επιπλέον, το πρωτόκολλο TCP διενεργεί και έλεγχο ροής των πακέτων, φροντίζοντας να μειώσει το ρυθμό μεταφοράς τους σε καταστάσεις συμφόρησης του δικτύου και μέχρις ότου αυτές εξομαλυνθούν. Το επίπεδο μεταφοράς παρέχει μια άλλη μορφή διευθυνσιοδότησης, γνωστή ως αριθμό θύρας (port number). Οι αριθμοί θύρας έχουν μήκους 16 bits. Τόσο το TCP όσο και το UDP έχουν σαφώς καθορισμένες κεφαλίδες για να μπορούν να συμπεριλάβουν τον αριθμό θύρας και άλλες πληροφορίες. Λόγω της αμφίδρομης επικοινωνίας, όπως ένα πακέτο IP περιλαμβάνει και την διεύθυνση της πηγής και τη διεύθυνση προορισμού, έτσι και τα TCP και UDP περιλαμβάνουν τους αριθμούς θυρών και της πηγής και του προορισμού. TCP και UDP έχουν ένα πεδίο στην επικεφαλίδα IP, γνωστό ως πεδίο τύπου πρωτοκόλλου, που χρησιμοποιείται για την διάκριση τους.

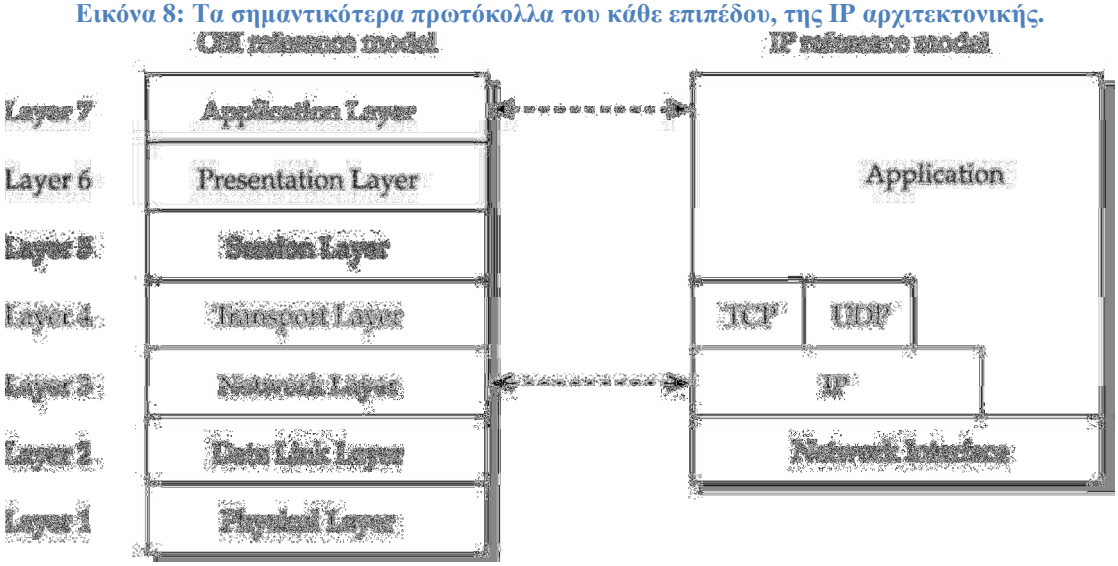
- Το επίπεδο εφαρμογών (application layer) περιλαμβάνει πρωτόκολλα εφαρμογής που χρησιμοποιούν τις υπηρεσίες του TCP για την επικοινωνία τους με τα ομότιμα πρωτόκολλα (FTP, SMTP, HTTP, TELNET,RTP), πρωτόκολλα εφαρμογής που χρησιμοποιούν το UDP (TFTP, SNMP, NFS) και μερικά πρωτόκολλα που χρησιμοποιούν και το TCP και το UDP (DNS). Η κύρια λειτουργία αυτών των πρωτοκόλλων εφαρμογής είναι η εξασφάλιση της διαλειτουργικότητας των αντίστοιχων εφαρμογών.



Εικόνα 7: Η μορφή των πακέτων δεδομένων TCP και UDP αντίστοιχα.



Εικόνα 8: Τα σημαντικότερα πρωτόκολλα του κάθε επιπέδου, της IP αρχιτεκτονικής.



Εικόνα 9: Αντιστοιχία επιπέδων μοντέλου αναφοράς OSI και μοντέλου αναφοράς IP.

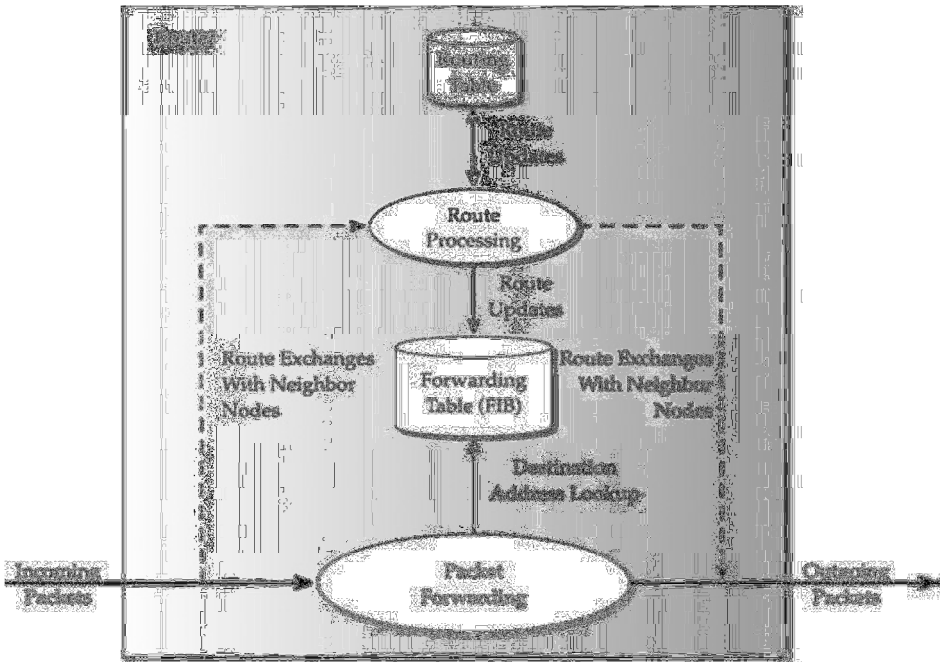
3.5 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΔΡΟΜΟΛΟΓΗΤΗ

Ένας δρομολογητής παρέχει πολλές σημαντικές λειτουργίες, προκειμένου να διασφαλιστεί η σωστή προώθηση πακέτων με το πιο αποτελεσματικό τρόπο. Ο δρομολογητής είναι ένας εξειδικευμένος υπολογιστής που χειρίζεται τρεις κύριες λειτουργίες:

- Πακέτο Προώθησης (Packet Forwarding): Κατά την παραλαβή των εισερχόμενων πακέτων, ο δρομολογητής επαληθεύει εάν το πακέτο έφτασε

χωρίς λάθη. Μετά την επιθεώρηση της επικεφαλίδας του πακέτου για την διεύθυνση προορισμού, εκτελεί μια λειτουργία αναζήτησης στον πίνακα δρομολόγησης, για να καθορίσει πως θα βρει την κατάλληλη εξερχόμενη σύνδεση.

- Πρωτόκολλο Δρομολόγησης Επεξεργασία Μηνύματος (Routing Protocol Message Processing): Ο δρομολογητής χειρίζεται επίσης τα πακέτα πρωτοκόλλου δρομολόγησης. Με την βοήθεια κάποιου αλγορίθμου δρομολόγησης, όταν και εφόσον χρειάζεται, καθορίζει εάν πρέπει να γίνουν τυχόν αλλαγές στον πίνακα δρομολόγησης.
- Εξειδικευμένες Υπηρεσίες (Specialized Services): Ένας δρομολογητής είναι απαραίτητος για τον χειρισμό των εξειδικευμένων υπηρεσιών που μπορούν να βοηθήσουν στην παρακολούθηση και τη διαχείριση ενός δικτύου.



Εικόνα 10: Λειτουργική απεικόνιση της αρχιτεκτονικής δρομολογητή.

3.6 ΤΕΧΝΟΛΟΓΙΕΣ ΕΠΙΚΟΙΝΩΝΙΑΣ

Οι τεχνολογίες επικοινωνίας χρησιμοποιούνται για την εκτέλεση των υπηρεσιών του επιπέδου δικτύου, είτε πρόκειται για το Internet είτε για το PSTN (Public Switched Telephone Network). Με αυτή την έννοια, οι τεχνολογίες

επικοινωνίας παρέχουν υπηρεσίες μεταφορών τόσο για το Διαδίκτυο όσο και για το PSTN.

Για την παροχή υπηρεσιών μεταφοράς, η ανάπτυξη των δικτύων μεταφορών βασίζεται σε μία ή περισσότερες τεχνολογίες επικοινωνίας. Στο φυσικό επίπεδο χρησιμοποιούνται ίνες ή ομοαξονικά καλώδια για τις ενσύρματες υπηρεσίες μεταφορών. Τέτοια καλώδια μπορούν να βρίσκονται υπόγεια ή πάνω από την γη συνδεδεμένα σε στύλους. Ακόμη καλώδια που βρίσκονται κάτω από το νερό χρησιμοποιούνται για σύνδεση ανάμεσα σε διαφορετικές ηπείρους. Σήμερα, τα υποβρύχια καλώδια βασίζονται σχεδόν αποκλειστικά σε καλώδια οπτικών ινών. Στην κορυφή της καλωδίωσης, βρίσκεται ένα σύνολο ψηφιακών τεχνολογιών επικοινωνίας με καλά καθορισμένες ταχύτητες δεδομένων. Στην παρακάτω εικόνα φαίνονται οι διαφορετικές τεχνολογίες με τους αντίστοιχους ρυθμούς δεδομένων τους.

Signal/data rate name	Bit rate (Mbps)
DS0 (voice circuit)	0.064
T1 (DS-1)	1.54
E1	2.04
Ethernet	10.00
T3 (DS-3)	45.00
E3	34.36
STS-1	51.84
Fast Ethernet	100.00
OC-3/STS-3/STM-1	155.52
OC-12/STS-12/STM-4	622.08
Gigabit Ethernet	1,000.00
OC-48/STS-48/STM-16	2,488.32
OTU1 (Optical Transport Unit-1)	2,666.06
OC-192/STS-192/STM-64	9,953.28
OTU2 (Optical Transport Unit-2)	10,709.22
OC-768/STS-768/STM-256	39,813.12
OTU3 (Optical Transport Unit-3)	43,018.41

Εικόνα 11: Οι τεχνολογίες επικοινωνιών και οι αντίστοιχοι ρυθμοί μετάδοσης δεδομένων τους.

Η παροχή υπηρεσιών μεταφοράς με τη χρήση τεχνολογιών επικοινωνίας, οδηγεί σε μια σειρά από προβλήματα δρομολόγησης δικτύου μεταφοράς:

1. Πρέπει να ληφθεί υπόψη η ικανότητα μιας συγκεκριμένης τεχνολογίας επικοινωνιών και η συσκευή δρομολόγησης.
2. Η πολυεπίπεδη δικτύωση και η πολυεπίπεδη δρομολόγηση πηγαίνουν από το στρώμα 3 μέχρι στρώμα 1 λόγω της δρομολόγησης του δικτύου μεταφορών.
3. Οι νέες τεχνολογίες για τα δίκτυα μεταφοράς, συνεχώς αναπτύσσονται. Νέες δυνατότητες εμφανίζονται, δημιουργώντας νέες ευκαιρίες στη δρομολόγηση των δικτύων μεταφοράς.

4. Τα διαφορετικά δίκτυα μεταφοράς, έχουν πολύ μικρή ικανότητα να επικοινωνούν μεταξύ τους και συνεπώς, βασίζονται σε χειροκίνητη διαμόρφωση (manual configurations).

Τώρα ξεκίνησε η ανάπτυξη νέων δυνατοτήτων που προσφέρονται για δυναμική διαμόρφωση και έχουν την ικανότητα να ανταλλάσσουν πληροφορίες μεταξύ των δικτύων σε διάφορα στρώματα, έτσι ώστε τα επόμενα χρόνια να είναι εφικτή μια δυναμικά αναδιαρθρώσιμη πολυστρωματική δρομολόγηση. Ωστόσο, μια τέτοια πολυστρωματική δρομολόγηση φέρνει νέες προκλήσεις.

ΚΕΦΑΛΑΙΟ 4

Η γλώσσα Python

4.1 ΕΙΣΑΓΩΓΗ ΣΤΗΝ PYTHON

Η Python αποτελεί γλώσσα υψηλού επιπέδου, όπως για παράδειγμα η C, η C++ , η Perl και η Java, σε αντίθεση με γλώσσες χαμηλού επιπέδου, οι οποίες μερικές φορές αναφέρονται και ως ”γλώσσες μηχανής” ή ”συμβολικές γλώσσες”. Η συγκεκριμένη γλώσσα θεωρείται μία διερμηνευμένη γλώσσα επειδή τα προγράμματά της εκτελούνται από έναν διερμηνέα. Υπάρχουν δύο τρόποι χρήσης του διερμηνέα: διαδραστική λειτουργία (**interactive mode**) και σεναριακή λειτουργία (**script mode**).

Στην διαδραστική λειτουργία, πληκτρολογούμε προγράμματα σε Python και ο διερμηνέας εμφανίζει το αποτέλεσμα:

```
>>> 1 + 1
2
```

Το σύμβολο, >>>, είναι ο προτροπέας (**prompt**) που χρησιμοποιεί ο διερμηνέας για να υποδείξει ότι είναι έτοιμος. Αν πληκτρολογήσετε 1+1, ο διερμηνέας απαντάει 2. Εναλλακτικά, μπορείτε να αποθηκεύσετε κώδικα σε ένα φάκελο και να χρησιμοποιήσετε το διερμηνέα για να εκτελέσει τα περιεχόμενα του φακέλου, το οποίο ονομάζεται ένα σενάριο (script). Κατά παράδοση, τα σενάρια της Python έχουν ονόματα τα οποία έχουν κατάληξη .py. Για να εκτελεστεί το σενάριο, πρέπει να πούμε στο διερμηνέα το όνομα του φακέλου. Για παράδειγμα, αν έχουμε ένα σενάριο με όνομα my-test-prog.py και δουλεύουμε σε ένα παράθυρο εντολών UNIX, πληκτρολογούμε python my-test-prog.py. Σε άλλα περιβάλλοντα ανάπτυξης, οι λεπτομέρειες εκτέλεσης των σεναρίων είναι διαφορετικές. Αναλυτικές οδηγίες για το περιβάλλον υπάρχουν στην ιστοσελίδα της Python (http://python.org).

Όταν δουλεύουμε στην διαδραστική λειτουργία μας βοηθάει να εξετάζουμε μικρά κομμάτια κώδικα επειδή μπορούμε να τα πληκτρολογήσουμε και να εκτελεστούν άμεσα.

4.2 ΤΕΛΕΣΤΕΣ ΚΑΙ ΕΚΦΡΑΣΕΙΣ

Οι εντολές που γράφουμε περιέχουν εκφράσεις (expressions). Ένα απλό παράδειγμα μίας έκφρασης είναι 2+3. Μία έκφραση μπορεί να διαχωριστεί σε τελεστές (operators) και τελεστέους (operands). Οι τελεστές είναι λειτουργίες που κάνουν κάτι και μπορούν να αναπαρασταθούν με σύμβολα όπως το + ή με ειδικές λέξεις-κλειδιά. Οι τελεστές απαιτούν κάποια δεδομένα πάνω στα οποία θα λειτουργήσουν και αυτά τα δεδομένα ονομάζονται τελεστέοι. Στη συγκεκριμένη περίπτωση, οι τελεστέοι είναι το 2 και το 3. Ακολουθεί αναλυτικός πίνακας με τελεστές.

Τελεστής	**Όνομα**	**Εξήγηση**	**Παραδείγματα**
+	Συν	Προσθέτει δύο αντικείμενα.	Το 3 + 5 δίνει 8. Το 'a' + 'b' δίνει 'ab'.
-	Μείον	Είτε δίνει έναν αρνητικό αριθμό, ή αφαιρεί έναν αριθμό από έναν άλλο.	Το -5.2 δίνει έναν αρνητικό αριθμό. Το 50 - 24 δίνει 26.
*	Επί	Δίνει το γινόμενο δύο αριθμών ή μιά συμβολοσειρά (string) επαναλαμβανόμενη τόσες φορές.	Το 2 * 3 δίνει 6. Το 'la' * 3 δίνει 'lalala'.
**	Δύναμη	Επιστρέφει το x υψωμένο στη δύναμη y	Το 3 ** 4 δίνει 81 (δηλαδή 3 * 3 * 3 * 3).
/	Διά	Διαιρεί το x με το y	Το 4 / 3 δίνει 1.3333333333333333.
//	Διαίρεση στρογγυλοποιημένη προς τα κάτω (Floor Division)	Επιστρέφει τον κοντινότερο (προς τα κάτω) ακέραιο στο πηλίκο.	Το 4 // 3 δίνει 1.
%	Υπόλοιπο	Επιστρέφει το υπόλοιπο της διαίρεσης.	Το 8 % 3 δίνει 2. Το -25.5 % 2.25 δίνει 1.5
<<	Αριστερή μετάθεση	Μεταθέτει τα δυαδικά ψηφία (bits) του αριθμού προς τα αριστερά κατά το πλήθος των θέσεων που καθορίστηκε. (Κάθε αριθμός αναπαρίσταται στη μνήμη με δυαδικά ψηφία (bits, binary digits) - δηλαδή με 0 και 1).	Το 2 << 2 δίνει 8. Το 2 αναπαριστάται ως σε bits ως 10. Η μετάθεση προς τα αριστερά κατά 2 bits μας δίνει 1000 που παριστάνει το δεκαδικό 8.

>>	Δεξιά μετάθεση	Μεταθέτει τα bits του αριθμού προς τα δεξιά κατά το πλήθος των θέσεων που καθορίστηκε.	Το 11 >> 1 δίνει 5. Το 11 αναπαριστάται σε bits ως 1011 που όταν μετατεθούν δεξιά κατά 1 bit μας δίνει 101 το οποίο είναι το δεκαδικό 5.
&	Δυαδικό ΚΑΙ	Δυαδικό ΚΑΙ των αριθμών.	Το 5 & 3 δίνει 1.
|	Δυαδικό Ή	Δυαδικό Ή των αριθμών.	Το 5 | 3 δίνει 7.
^	Δυαδικό αποκλειστικό Ή	Δυαδικό αποκλειστικό Ή των αριθμών	Το 5 ^ 3 δίνει 6.
~	Δυαδική αντιστροφή	Το δυαδικό αντίστροφο του x είναι -(x+1).	Το ~5 δίνει -6.
<	Μικρότερο από	Επιστρέφει το αν το x είναι μικρότερο από το y. Όλοι οι τελεστές σύγκρισης επιστρέφουν True (Αληθής) ή False (Ψευδής). Σημειώστε οτι τα ονόματα αυτά ξεκινούν με κεφαλαίο	Το 5 < 3 δίνει False και το 3 < 5 δίνει True. Οι συγκρίσεις μπορούν να συνδυαστούν αλυσιδωτά κατά βούληση: Το 3 < 5 < 7 δίνει True.
>	Μεγαλύτερο από	Επιστρέφει το αν το x είναι μεγαλύτερο από το y.	Το 5 > 3 επιστρέφει True. Αν και οι δύο τελεστέοι είναι αριθμοί, πρώτα μετατρέπονται σε έναν κοινό τύπο. Αλλιώς, επιστρέφει πάντα False.
<=	Μικρότερο ή ίσο	Επιστρέφει το αν το x είναι μικρότερο από ή ίσο με το y.	Το x = 3; y = 6; x <= y επιστρέφει True.
>=	Μεγαλύτερο ή ίσο	Επιστρέφει το αν το x είναι μεγαλύτερο από ή ίσο με το y.	Το x = 4; y = 3; x >= 3 επιστρέφει True.
==	Ίσο	Συγκρίνει αν τα αντικείμενα είναι ίσα.	Το x = 2; y = 2; x == y επιστέφει True. Το x = 'str'; y = 'stR'; x == y επιστρέφει False. Το x = 'str'; y = 'str'; x == y επιστρέφει True.
!=	Διαφορετικό	Συγκρίνει αν τα αντικείμενα ΔΕΝ είναι ίσα.	Το x = 2; y = 3; x != y επιστρέφει True.
not	Λογικό ΟΧΙ	Αν το x είναι True, επιστρέφει False. Αν το x είναι False, επιστρέφει True.	x = True; not x επιστρέφει False.

and	Λογικό ΚΑΙ	Το x and y επιστρέφει False αν το x είναι False, αλλιώς επιστρέφει υπολογίζει και επιστρέφει την τιμή του y.	x = False; y = True; x and y επιστρέφει False αφού το x είναι False. Σε αυτή την περίπτωση, η Python δε θα ελέγξει την τιμή του y αφού γνωρίζει οτι η αριστερή πλευρά της έκφρασης 'and' είναι False που υποδηλώνει οτι ολόκληρη η έκφραση θα είναι False ανεξάρτητα από τις άλλες τιμές. Αυτή η τεχνική αποκαλείται short-circuit evaluation.
or	Λογικό Ή	Αν το x είναι True, επιστρέφει True, αλλιώς υπολογίζει και επιστρέφει την τιμή του y.	Το x = True; y = False; x or y επιστρέφει True. Η Short-circuit evaluation εφαρμόζεται και εδώ.

Ο ακόλουθος πίνακας μας δίνει τον πίνακα προτεραιοτήτων για την Python, από τη χαμηλότερη προς την υψηλότερη προτεραιότητα. Αυτό σημαίνει ότι σε μία δεδομένη έκφραση, η Python θα υπολογίσει τους τελεστές και τις εκφράσεις που βρίσκονται χαμηλότερα στον πίνακα πριν από αυτούς που βρίσκονται ψηλότερα.

Ο παρακάτω πίνακας, αντιγραμμένος από τη επίσημη σελίδα της Python, παρέχεται χάριν πληρότητας. Βέβαια, είναι πολύ καλύτερο να χρησιμοποιούνται παρενθέσεις για να ομαδοποιούνται σωστά τελεστές και τελεστέοι ώστε να είναι ξεκάθαρη η προτεραιότητα. Έτσι το πρόγραμμα γίνεται πιο ευανάγνωστο.

Τελεστής	**Περιγραφή**
lambda	Έκφραση Lambda
or	Λογικό Ή
and	Λογικό ΚΑΙ
not x	Λογικό ΟΧΙ
in, not in	Έλεγχοι συμμετοχής
is, is not	Έλεγχοι ταυτότητας
<, <=, >, >=, !=, ==	Συγκρίσεις
|	Δυαδικό Ή

^	Δυαδικό αποκλειστικό Ή
&	Δυαδικό ΚΑΙ
<<, >>	Μεταθέσεις
+, -	Πρόσθεση και αφαίρεση
*, /, //, %	Πολλαπλασιασμός, Διαίρεση, Διαίρεση στρογγυλοποιημένη προς τα κάτω, Υπόλοιπο
+x, -x	Θετικό, Αρνητικό
~x	Δυαδικό ΟΧΙ
**	Ύψωση σε δύναμη
x.attribute	Αναφορά σε χαρακτηριστικό
x[index]	Subscription
x[index1:index2]	Κομμάτιασμα (Slicing)
f(arguments ...)	Κλήση συνάρτησης
(expressions, ...)	Συνένωση ή εμφάνιση πλειάδας
[expressions, ...]	Προβολή λίστας
{key:datum, ...}	Προβολή λεξικού

Σημείωση: Τελεστές με την ίδια προτεραιότητα βρίσκονται στην ίδια γραμμή στον παραπάνω πίνακα. Για παράδειγμα, το + και το - έχουν την ίδια προτεραιότητα.

4.3 ΕΛΕΓΧΟΣ ΡΟΉΣ

Όπως και σε άλλες γλώσσες προγραμματισμού, υπάρχει η σειρά εντολών τις οποίες εκτελεί πιστά η Python με την ίδια σειρά. Αν θέλουμε όμως να αλλάξουμε την

ροή εκτέλεσης και για παράδειγμα, θέλουμε το πρόγραμμα να πάρει μερικές αποφάσεις και να κάνει διαφορετικά πράγματα υπό διαφορετικές προϋποθέσεις, όπως π.χ. να εκτυπώσει "καλημέρα" ή "καλησπέρα", ανάλογα με την ώρα, αυτό επιτυγχάνεται χρησιμοποιώντας εντολές ελέγχου ροής. Υπάρχουν τρεις εντολές ελέγχου ροής στην Python - if, for και while.

Η εντολή if

Η εντολή if χρησιμοποιείται για να ελεγχθεί μια συνθήκη και εάν (if) η συνθήκη αυτή είναι αληθής, τότε εκτελείται ένα σύνολο ή ομάδα εντολών (που ονομάζεται if-block), διαφορετικά (else) γίνεται επεξεργασία ενός άλλου συνόλου εντολών (που ονομάζεται else-block). Η χρήση του όρου else είναι προαιρετική.

Παράδειγμα:

```
number = 23
guess = int(input('Εισάγετε έναν ακέραιο αριθμό: '))
if guess == number:
        print('Συγχαρητήρια, τον μαντέψατε.')
        print('(Αλλά δεν κερδίζετε και κανένα βραβείο!)')
elif guess < number:
        print('Όχι, είναι λίγο μεγαλύτερος.')
else:
        print('Όχι, είναι λίγο μικρότερος.')
print('Τέλος')
```

Έξοδος:

```
Εισάγετε έναν ακέραιο αριθμό: 50
Όχι, είναι λίγο μικρότερος.
Τέλος
>>> (εκ νέου εκτέλεση)
Εισάγετε έναν ακέραιο αριθμό: 22
Όχι, είναι λίγο μεγαλύτερος.
Τέλος
>>> (εκ νέου εκτέλεση)
Εισάγετε έναν ακέραιο αριθμό: 23
Συγχαρητήρια, τον μαντέψατε.
(Αλλά δεν κερδίζετε και κανένα βραβείο!)
Τέλος
```

Σχολιασμός κώδικα:

Στο πρόγραμμα αυτό μαντεύει ο χρήστης αριθμούς και ελέγχουμε αν αντιστοιχούν στον αριθμό που έχουμε ορίσει. Ορίζουμε την μεταβλητή number ως οποιονδήποτε ακέραιο αριθμό επιθυμούμε, π.χ. 23. Ύστερα παίρνουμε την πρόγνωση του χρήστη χρησιμοποιώντας την συνάρτηση input(). Οι συναρτήσεις είναι απλά επαναχρησιμοποιήσιμα κομμάτια προγραμμάτων, όπως σχολιάζεται και στην συνέχεια του κεφαλαίου. Παρέχουμε μια συμβολοσειρά (string) στην ενσωματωμένη συνάρτηση input, η οποία εκτυπώνει στην οθόνη αυτή την συμβολοσειρά και αναμένει την εισαγωγή δεδομένων από τον χρήστη. Μόλις εισάγουμε κάτι και πατήσουμε το πλήκτρο enter, η συνάρτηση input() επιστρέφει αυτό που εισάγαμε ως συμβολοσειρά. Στην συνέχεια, αυτή η συμβολοσειρά μετατρέπεται σε ακέραιο

αριθμό με την χρήση της int κι αποθηκεύεται στην μεταβλητή guess. Στο επόμενο βήμα συγκρίνουμε το προγνωστικό του χρήστη με τον αριθμό που επιλέξαμε. Εάν είναι ίδιοι, εκτυπώνεται ένα μήνυμα επιτυχίας.

Πρέπει να παρατηρήσουμε ότι η εντολή if περιλαμβάνει άνω και κάτω τελεία στο τέλος – έτσι δηλώνουμε στην Python ότι ακολουθεί μια ομάδα εντολών.
Στην συνέχεια ελέγχουμε αν το προγνωστικό είναι μικρότερο από τον αριθμό μας, κι αν ναι, πληροφορούμε τον χρήστη ότι η πρόγνωσή του πρέπει να είναι λίγο μεγαλύτερη απ' αυτήν. Αυτό που χρησιμοποιήσαμε εδώ είναι ο όρος elif ο οποίος στην πραγματικότητα συνδυάζει δύο συσχετιζόμενες εντολές if else-if else σε μία εντολή if-elif-else. Αυτό κάνει το πρόγραμμα ευκολότερο και μειώνει τον αριθμό των εσοχών που απαιτούνται.

Αντίστοιχα, οι εντολές elif και else πρέπει επίσης να έχουν άνω και κάτω τελεία στο τέλος της λογικής γραμμής, ακολουθούμενες από τις αντίστοιχες ομάδες εντολών (με κατάλληλες εσοχές).

Σημείωση: τα τμήματα elif και else είναι προαιρετικά. Μια ελάχιστη έγκυρη εντολή if είναι:

```
if True:
	print('Ναι, είναι αληθές')
```

Αφού η Python τελειώσει την εκτέλεση ολόκληρης της εντολής if συμπεριλαμβανομένων και των συσχετιζόμενων όρων elif και else, προχωράει στην επόμενη εντολή στην ομάδα που περιέχει την εντολή if. Στην περίπτωσή μας συνεχίζει η εκτέλεση του προγράμματος και η επόμενη εντολή είναι print('Τέλος'). Μετά απ' αυτό η Python βλέπει το τέλος του προγράμματος και απλά τελειώνει εκεί.

Η εντολή while

Η εντολή while μας επιτρέπει να εκτελούμε επανειλημμένα μια ομάδα εντολών, όσο μια προϋπόθεση παραμένει αληθής. Η εντολή while είναι ένα παράδειγμα αυτού που αποκαλείται εντολή βρόχου (looping statement).

Παράδειγμα:

```
number = 23
running = True
while running:
	guess = int(input('Εισάγετε έναν ακέραιο αριθμό : '))
	if guess == number:
		print('Συγχαρητήρια, τον μαντέψατε.')
		running = False
	elif guess < number:
		print('Όχι, είναι λίγο μεγαλύτερος.')
	else:
		print('Όχι, είναι λίγο μικρότερος.')
print('Τέλος')
```

Έξοδος:

```
Εισάγετε έναν ακέραιο αριθμό : 50
Όχι, είναι λίγο μεγαλύτερος.
Εισάγετε έναν ακέραιο αριθμό : 22
Όχι, είναι λίγο μεγαλύτερος
Εισάγετε έναν ακέραιο αριθμό : 23
Συγχαρητήρια, τον μαντέψατε.
Τέλος
```

Σχολιασμός κώδικα:

Σ' αυτό το πρόγραμμα παίζουμε ακόμα το παιχνίδι πρόγνωσης, αλλά το πλεονέκτημα είναι ότι επιτρέπει στον χρήστη να συνεχίσει να μαντεύει μέχρι να βρει τον σωστό αριθμό -δεν χρειάζεται δηλ. να εκτελεί επανειλημμένα το πρόγραμμα για κάθε πρόγνωση, όπως γινόταν στο προηγούμενο παράδειγμα. Αυτό είναι ένα πολύ καλό παράδειγμα της χρήσης της εντολής while. Μετακινούμε τις εντολές input και if εντός του βρόχου while και ορίζουμε την μεταβλητή running σε True πριν τον βρόχο while. Πρώτα ελέγχουμε αν η μεταβλητή running έχει την τιμή True και στην συνέχεια προχωράμε στην εκτέλεση της αντίστοιχης while (while-block). Αφού εκτελεστεί αυτή η ομάδα, η προϋπόθεση ελέγχεται και πάλι, δηλ. στην περίπτωσή μας η μεταβλητή running. Εάν η τιμή της συνεχίζει να είναι true (αληθές), εκτελείται και πάλι ολόκληρη η ομάδα εντολών της while (while-block), διαφορετικά προχωράμε στην επόμενη εντολή.

Ο βρόχος for

Η εντολή for..in είναι άλλη μία εντολή βρόχου, η οποία επαναλαμβάνεται σε μια ακολουθία αντικειμένων, δηλ. εκτελείται σε κάθε αντικείμενο σε μια ακολουθία. Θα δούμε περισσότερες λεπτομέρειες σχετικά με τις ακολουθίες στα επόμενα κεφάλαια. Αυτό που πρέπει να γνωρίζετε προς το παρόν είναι ότι μια ακολουθία είναι απλά μια ταξινομημένη συλλογή αντικειμένων.

Παράδειγμα:

```
for i in range(1, 5):
        print(i)
```

Έξοδος:

```
1
2
3
4
```

Σχολιασμός κώδικα:

Σ' αυτό το πρόγραμμα εκτυπώνουμε μια ακολουθία αριθμών. Παράγουμε αυτή την ακολουθία αριθμών με την ενσωματωμένη συνάρτηση range. Αυτό που κάνουμε εδώ είναι ότι παρέχουμε δύο αριθμούς και η range επιστρέφει μια ακολουθία

αριθμών, ξεκινώντας από τον πρώτο αριθμό έως το δεύτερο αριθμό. Για παράδειγμα, η range(1,5) δίνει την ακολουθία [1, 2, 3, 4]. Εξ ορισμού, η range έχει ως αριθμό κλιμάκωσης (step count) το 1. Εάν παρέχουμε έναν τρίτο αριθμό στην range, τότε αυτός γίνεται ο αριθμός κλιμάκωσης (step count). Για παράδειγμα, range(1,5,2) δίνει ως αποτέλεσμα [1,3]. Το εύρος των αριθμών εκτείνεται έως τον δεύτερο αριθμό, δηλ. δεν συμπεριλαμβάνει τον δεύτερο αριθμό. Στην συνέχεια, ο βρόχος for επαναλαμβάνεται σ' αυτό το εύρος - for i in range(1,5) είναι αντίστοιχο του for i in [1, 2, 3, 4] που είναι σαν να αντιστοιχούμε κάθε αριθμό (ή αντικείμενο) της ακολουθίας στο i ξεχωριστά και στην συνέχεια να εκτελούμε την ομάδα των εντολών για κάθε τιμή της i. Στην περίπτωσή μας, απλά εκτυπώνουμε την τιμή στην ομάδα των εντολών.

4.4 ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Οι δομές δεδομένων αποτελούν δομές που μπορούν να κρατήσουν μαζί μερικά δεδομένα. Με άλλα λόγια χρησιμοποιούνται για να αποθηκεύουν δεδομένα που έχουν σχέση μεταξύ τους. Υπάρχουν τέσσερις δομές δεδομένων ενσωματωμένες στη Python, οι λίστες, οι πλειάδες, τα λεξικά και τα σύνολα και ακολουθεί η περιγραφή τους στη συνέχεια.

Λίστα

Μια λίστα είναι μια δομή δεδομένων που συγκρατεί μια διατεταγμένη συλλογή στοιχείων, δηλαδή μπορούμε να αποθηκεύσουμε μια ακολουθία (sequence) αντικειμένων στη λίστα. Αυτό είναι εύκολο να το φανταστούμε αν σκεφτούμε μια λίστα για αγορές, όπου έχουμε μια λίστα με αντικείμενα που πρέπει να αγοράσουμε, και εκτός αυτού πιθανόν έχουμε κάθε στοιχείο σε διαφορετική γραμμή στη λίστα αγορών μας, ενώ στην Python τοποθετούμε κόμματα ανάμεσα στα στοιχεία. Η λίστα των στοιχείων πρέπει να κλείνεται σε αγκύλες (δηλαδή [και]) έτσι ώστε να καταλαβαίνει η Python ότι καθορίζεται μια λίστα. Αφού έχει δημιουργηθεί μια λίστα στη συνέχεια μπορούμε να προσθέσουμε, να μετακινήσουμε ή να ψάξουμε για στοιχεία σ' αυτή τη λίστα. Από τη στιγμή που μπορούμε να προσθέσουμε και να μετακινήσουμε στοιχεία, λέμε ότι η λίστα είναι ένας μεταβλητός τύπος δεδομένων (mutable data type), δηλαδή αυτός ο τύπος μπορεί να αλλαχθεί.

Παράδειγμα:

```
# Αυτή είναι η λίστα αγορών μου
shoplist = ['μήλο', 'μάνγκο', 'καρότο', 'μπανάνα']
print('Πρέπει ν\' αγοράσω', len(shoplist), 'πράγματα.')
print('Τα πράγματα αυτά είναι:', end=' ')
for item in shoplist:
print(item, end=' ')
print('\nΠρέπει επίσης ν\' αγοράσω ρύζι.')
shoplist.append('ρύζι')
print('Η λίστα αγορών μου τώρα είναι:', shoplist)
print('Θα ταξινομήσω τη λίστα μου τώρα')
shoplist.sort()
print('Η ταξινομημένη λίστα μου είναι', shoplist)
print('Το πρώτο πράγμα που θ\' αγοράσω είναι', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('Αγόρασα το', olditem)
print('Η λίστα αγορών μου τώρα είναι', shoplist)
```

Έξοδος:

```
Πρέπει ν' αγοράσω 4 πράγματα.
Τα πράγματα αυτά είναι: μήλο μάνγκο καρότο μπανάνα
Πρέπει επίσης ν' αγοράσω ρύζι.
Η λίστα αγορών μου τώρα είναι: ['μήλο', 'μάνγκο', 'καρότο', 'μπανάνα', 'ρύζι']
Θα ταξινομήσω τη λίστα μου τώρα
Η ταξινομημένη λίστα μου είναι ['καρότο', 'μάνγκο', 'μήλο', 'μπανάνα', 'ρύζι']
Το πρώτο πράγμα που θ' αγοράσω είναι καρότο
Αγόρασα το καρότο
Η λίστα αγορών μου τώρα είναι ['μάνγκο', 'μήλο', 'μπανάνα', 'ρύζι']
```

Σχολιασμός κώδικα:

Η μεταβλητή shoplist είναι μια λίστα αγορών για κάποιον που πηγαίνει στην αγορά. Στη shoplist αποθηκεύουμε συμβολοσειρές των ονομάτων των αντικειμένων που θα αγοράσουμε, αλλά μπορούμε να προσθέσουμε οποιοδήποτε είδος αντικειμένου στη λίστα συμπεριλαμβανομένων αριθμών ή ακόμα και άλλες λίστες. Επίσης έχουμε χρησιμοποιήσει το βρόχο for..in για να επανελέγξουμε τα αντικείμενα της λίστας. Μέχρι τώρα, πρέπει να έχετε καταλάβει ότι η λίστα είναι επίσης μια ακολουθία. Ας παρατηρηθεί πως χρησιμοποιούμε τη λέξη-κλειδί όρισμα end στη συνάρτηση print, για να δείξουμε ότι θέλουμε να τελειώσουμε (end) την έξοδο με ένα διάστημα (space) αντί της συνηθισμένης νέας γραμμής (line break). Κατόπιν προσθέτουμε ένα στοιχείο στη λίστα χρησιμοποιώντας τη μέθοδο append του αντικειμένου λίστας. Τότε, ελέγχουμε ότι το στοιχείο έχει πραγματικά προστεθεί στη λίστα, τυπώνοντας τα περιεχόμενα της λίστας, απλά περνώντας τη λίστα στην εντολή print και την τυπώνει. Τότε, ταξινομούμε τη λίστα χρησιμοποιώντας τη μέθοδο sort της λίστας. Κατόπιν, όταν αγοράσουμε ένα πράγμα από τη λαϊκή, θέλουμε να το αφαιρέσουμε από τη λίστα. Αυτό το επιτυγχάνουμε χρησιμοποιώντας την εντολή del. Εδώ αναφέρουμε ποιο στοιχείο της λίστας θέλουμε να αφαιρέσουμε και η εντολή del το αφαιρεί από τη λίστα για μας. Καθορίζουμε ότι θέλουμε να αφαιρέσουμε το πρώτο αντικείμενο από τη λίστα και έτσι χρησιμοποιούμε την del shoplist[0].

Πλειάδα

Οι πλειάδες χρησιμοποιούνται για να συγκρατήσουν μαζί πολλαπλά αντικείμενα, παρόμοια με τις λίστες), αλλά χωρίς την εκτεταμένη λειτουργικότητα που η κλάση της λίστας μας δίνει. Ένα κύριο χαρακτηριστικό των πλειάδων είναι ότι είναι αμετάβλητες όπως οι συμβολοσειρές, δηλαδή μετά την δημιουργία δεν υπάρχει δυνατότητα τροποποίησης.

Οι πλειάδες ορίζονται καθορίζοντας στοιχεία που διαχωρίζονται με κόμματα, μέσα σε ένα προαιρετικό ζευγάρι παρενθέσεων. Οι πλειάδες χρησιμοποιούνται, συνήθως, στις περιπτώσεις όπου μια εντολή ή μια συνάρτηση οριζόμενη από το χρήστη, μπορεί με ασφάλεια να θεωρήσει ότι η συλλογή των τιμών δηλ. η πλειάδα των τιμών που χρησιμοποιούνται δε θα αλλάξει.

Παράδειγμα:

```
zoo = ('πύθωνας', 'ελέφαντας', 'πιγκουίνος')
print('Ο αριθμός των ζώων στο ζωολογικό κήπο είναι', len(zoo))
new_zoo = ('μαϊμού', 'καμήλα', zoo)
print('Ο αριθμός των κλουβιών στο νέο ζωολογικό κήπο είναι',
len(new_zoo))
print('Όλα τα ζώα στο νέο ζωολογικό κήπο είναι', new_zoo)
print('Όλα τα ζώα που έφεραν από τον παλιό ζωολογικό κήπο είναι',
new_zoo[2])
print('Το τελευταίο ζώο που έφεραν από τον παλιό ζωολογικό κήπο είναι', new_zoo[2][2])
print('Ο αριθμός των ζώων που είναι στο νέο ζωολογικό κήπο',
len(new_zoo)-1+len(new_zoo[2]))
```

Έξοδος:

```
Ο αριθμός των ζώων στο ζωολογικό κήπο είναι 3
Ο αριθμός των κλουβιών στο νέο ζωολογικό κήπο είναι 3
Όλα τα ζώα στο νέο ζωολογικό κήπο είναι ('μαϊμού', 'καμήλα', ('πύθωνας', 'ελέφαντας',
'πιγκουίνος'))
Όλα τα ζώα που έφεραν από τον παλιό ζωολογικό κήπο είναι ('πύθωνας', 'ελέφαντας',
'πιγκουίνος')
Το τελευταίο ζώο που έφεραν από τον παλιό ζωολογικό κήπο είναι πιγκουίνος
Ο αριθμός των ζώων που είναι στο νέο ζωολογικό κήπο 5
```

Σχολιασμός κώδικα:

Η μεταβλητή zoo αναφέρεται σε μια πλειάδα στοιχείων. Βλέπουμε ότι η συνάρτηση len μπορεί να χρησιμοποιηθεί για να πάρει το μήκος της πλειάδας. Αυτό επίσης δείχνει ότι η πλειάδα είναι επίσης και μια ακολουθία. Τώρα μετακινούμε αυτά τα ζώα σε ένα νέο ζωολογικό κήπο επειδή ο παλιός κλείνει. Συνεπώς, η πλειάδα the new_zoo περιέχει κάποια ζώα που βρίσκονται ήδη εκεί, μαζί με τα ζώα που έφεραν από τον παλιό ζωολογικό κήπο.

Μπορούμε να έχουμε πρόσβαση στα αντικείμενα μέσα στην πλειάδα, καθορίζοντας τη θέση του στοιχείου μέσα σε ένα ζευγάρι αγκύλες, ακριβώς όπως κάναμε για τις λίστες. Αυτό ονομάζεται τελεστής ευρετηρίασης (indexing operator). Παίρνουμε το τρίτο στοιχείο μέσα στο new_zoo καθορίζοντας new_zoo[2] και

παίρνουμε το τρίτο στοιχείο μέσα στο τρίτο στοιχείο στην πλειάδα new_zoo καθορίζοντας new_zoo[2][2].

Λεξικό

Ένα λεξικό είναι σαν ένας τηλεφωνικός κατάλογος όπου μπορούμε να βρούμε τη διεύθυνση ή άλλα στοιχεία επικοινωνίας για ένα άτομο, γνωρίζοντας μόνο το όνομά του/της, δηλαδή συσχετίζουμε κλειδιά (ονομασία) με τιμές (λεπτομέρειες) αλλά με βασικό κριτήριο πως το κλειδί πρέπει να είναι μοναδικό. Ζευγάρια κλειδιών και τιμών καθορίζονται στο λεξικό χρησιμοποιώντας το συμβολισμό `d = {key1 : value1, key2 : value2 }`. Γίνεται σαφές ότι τα ζευγάρια κλειδί-τιμή διαχωρίζονται με διπλή τελεία και τα ζευγάρια διαχωρίζονται μεταξύ τους με κόμματα και όλα αυτά περικλείονται σε ένα ζευγάρι άγκιστρων.

Πρέπει να τονίσουμε σε αυτό το σημείο πως στο λεξικό τα στοιχεία δεν ταξινομούνται με κανένα τρόπο. Αν θέλουμε μια ειδική σειρά ταξινόμησης, τότε πρέπει να τα ταξινομήσουμε από μόνοι μας πριν τα χρησιμοποιήσουμε.

Παράδειγμα:

```
ab = { 'Swaroop' : 'swaroop@swaroopch.com',
       'Larry' : 'larry@wall.org',
       'Matsumoto' : 'matz@ruby-lang.org',
       'Spammer' : 'spammer@hotmail.com'
       }
print("Swaroop's address is", ab['Swaroop'])
del ab['Spammer']
print('\nThere are {0} contacts in the address-book\n'.format(len(ab)))
for name, address in ab.items():
       print('Contact {0} at {1}'.format(name, address))

ab['Guido'] = 'guido@python.org'
if 'Guido' in ab:
       print("\nGuido's address is", ab['Guido'])
```

Έξοδος:

```
Swaroop's address is swaroop@swaroopch.com
There are 3 contacts in the address-book
Contact Swaroop at swaroop@swaroopch.com
Contact Matsumoto at matz@ruby-lang.org
Contact Larry at larry@wall.org
Guido's address is guido@python.org
```

Σχολιασμός κώδικα:

Δημιουργούμε το λεξικό ab χρησιμοποιώντας το συμβολισμό που ήδη συζητήσαμε. Τότε εισάγουμε ζευγάρια κλειδί-τιμή καθορίζοντας το κλειδί, χρησιμοποιώντας τον τελεστή ευρετηρίασης (indexing operator) όπως συζητήθηκε στο απόσπασμα των λιστών και πλειάδων. Μπορούμε να διαγράψουμε ζευγάρια κλειδί-τιμή χρησιμοποιώντας τον παλιό μας φίλο, την εντολή del. Εμείς απλά καθορίζουμε το λεξικό και τον τελεστή ευρετηρίασης για το κλειδί που θα αφαιρεθεί

και τα περνάμε στην εντολή del χωρίς να γνωρίζουμε την τιμή που αντιστοιχεί στο κλειδί για αυτή τη λειτουργία.

Έπειτα, εισάγουμε κάθε ζευγάρι κλειδί-τιμή του λεξικού χρησιμοποιώντας τη μέθοδο items του λεξικού, η οποία επιστρέφει μια λίστα πλειάδων, όπου κάθε πλειάδα περιέχει ένα ζευγάρι στοιχείων -το κλειδί ακολουθούμενο από την τιμή. Ανακτούμε αυτό το ζευγάρι και το εκχωρούμε στις μεταβλητές name (ονομασία) και address (διεύθυνση) αντιστοίχως για κάθε ζευγάρι, χρησιμοποιώντας το βρόχο for..in και μετά τυπώνει αυτές τις τιμές στην εντολή for. Μπορούμε να προσθέσουμε νέα ζευγάρια κλειδί-τιμή, απλά χρησιμοποιώντας τον τελεστή ευρετηρίασης για να εισάγουμε ένα κλειδί και να εκχωρήσουμε σ' αυτό μια τιμή, όπως έχουμε κάνει για το Guido στην ανωτέρω περίπτωση. Μπορούμε να ελέγξουμε εάν ένα ζευγάρι κλειδί-τιμή υπάρχει, χρησιμοποιώντας τον τελεστή in, ή ακόμα και τη μέθοδο has_key της κλάσης dict.

Σύνολο (Set)

Τα σύνολα είναι μη ταξινομημένες συλλογές απλών αντικειμένων. Αυτά χρησιμοποιούνται όταν η ύπαρξη ενός αντικειμένου σε μια συλλογή είναι πιο σπουδαία από την εντολή ή πόσες φορές αυτή συμβαίνει. Χρησιμοποιώντας τα σύνολα, μπορείτε να ελέγξετε για ένταξη (membership), εάν είναι ένα υποσύνολο (subset) ενός άλλου συνόλου, να βρείτε την τομή (intersection) ανάμεσα σε δύο σύνολα και ούτω καθεξής.

Παράδειγμα:

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

Σχολιασμός κώδικα:

Το παράδειγμα εξηγείται από μόνο του, διότι περιλαμβάνει βασική θεωρία μαθηματικών συνόλων που διδάσκεται στο σχολείο.

4.5 ΣΥΝΑΡΤΗΣΕΙΣ

Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα μέρη προγραμμάτων. Μας επιτρέπουν να δώσουμε ένα όνομα σε ένα σύνολο εντολών και να τρέξουμε εκείνο το σύνολο εντολών χρησιμοποιώντας το όνομά τους, οπουδήποτε στο προγραμμά μας και όσες φορές θελήσουμε. Αυτό είναι γνωστό σαν κλήση (calling) της συνάρτησης. Η έννοια των συναρτήσεων είναι πιθανόν το πιο σπουδαίο δομικό στοιχείο οποιουδήποτε μη στοιχειώδους προγράμματος (σε όλες τις γλώσσες προγραμματισμού), γι' αυτό θα διερευνήσουμε μερικές πτυχές των συναρτήσεων σε αυτό το υποκεφάλαιο.

Οι συναρτήσεις ορίζονται χρησιμοποιώντας τη λέξη κλειδί def, μετά την οποία ακολουθεί ένα όνομα που ταυτοποιεί την εκάστοτε συνάρτηση και κατόπιν ακολουθεί ένα ζευγάρι παρενθέσεων που μπορούν να περικλείουν μερικά ονόματα μεταβλητών, και η γραμμή τελειώνει με διπλή τελεία (:). Παρακάτω ακολουθεί ένα απλό παράδειγμα με ένα σύνολο εντολών που αποτελούν μέρος αυτής της συνάρτησης.

Παράδειγμα:

```
def sayHello():
       print('Hello World!') # σύνολο εντολών που ανήκουν στη συνάρτηση
sayHello() # κλήση της συνάρτησης
sayHello() # κλήση της συνάρτησης ξανά
```

Έξοδος:

```
Hello World!
Hello World!
```

Σχολιασμός κώδικα:

Ορίζουμε μια συνάρτηση με το όνομα sayHello ακολουθώντας τη σύνταξη όπως εξηγήσαμε παραπάνω. Αυτή η συνάρτηση δεν έχει παραμέτρους γι' αυτό δε δηλώνονται καθόλου μεταβλητές ανάμεσα στις παρενθέσεις. Οι παράμετροι στις συναρτήσεις είναι απλά η είσοδος στη συνάρτηση ώστε να περνάμε διαφορετικές τιμές στη συνάρτηση και να παίρνουμε αντίστοιχα αποτελέσματα.

Παράμετροι συναρτήσεων

Μια συνάρτηση μπορεί να δεχθεί παραμέτρους, οι οποίες είναι τιμές που δίνουμε στη συνάρτηση, έτσι ώστε αυτή να μπορεί να κάνει κάτι αξιοποιώντας αυτές τις τιμές. Αυτές οι παράμετροι μοιάζουν με τις μεταβλητές, διαφέροντας ως προς το ότι οι τιμές αυτών των μεταβλητών ορίζονται όταν καλούμε τη συνάρτηση και τους έχουν ήδη εκχωρηθεί τιμές όταν τρέχει η συνάρτηση. Οι παράμετροι καθορίζονται μέσα στο ζευγάρι των παρενθέσεων στον ορισμό της συνάρτησης και διαχωρίζονται με κόμμα. Όταν καλούμε τη συνάρτηση δίνουμε και τις τιμές με τον ίδιο τρόπο.

Σημείωση για την ορολογία που χρησιμοποιείται: οι ονομασίες που δίνουμε στον ορισμό της συνάρτησης ονομάζονται παράμετροι ενώ οι τιμές που δίνουμε όταν καλούμε τη συνάρτηση ονομάζονται ορίσματα.

Παράδειγμα:

```
def printMax(a, b):
	if a > b:
		print(a, 'είναι το μέγιστο')
	elif a == b:
		print(a, 'είναι ίσο με το', b)
	else:
		print(b, 'είναι το μέγιστο')

printMax(3, 4)
x = 5
y = 7
printMax(x, y)
```

Έξοδος:

```
4 είναι το μέγιστο
7 είναι το μέγιστο
```

Σχολιασμός κώδικα:

Εδώ ορίζουμε μια συνάρτηση που ονομάζεται printMax με δυο παραμέτρους τις a και b. Βρίσκουμε το μεγαλύτερο νούμερο χρησιμοποιώντας μια απλή εντολή if..else και μετά τυπώνουμε το μεγαλύτερο νούμερο. Στην πρώτη χρήση της printMax, απευθείας δίνουμε τους αριθμούς, δηλαδή τα ορίσματα. Στη δεύτερη χρήση της, καλούμε τη συνάρτηση χρησιμοποιώντας μεταβλητές. Η printMax(x, y) προκαλεί την τιμή του ορίσματος x να δοθεί στην παράμετρο a και την τιμή του ορίσματος y να δοθεί στην παράμετρο b. Η συνάρτηση printMax λειτουργεί με τον ίδιο τρόπο και στις δυο περιπτώσεις.

Προεπιλεγμένες τιμές ορίσματος

Σε κάποιες συναρτήσεις ίσως να θέλουμε να κάνουμε μερικές παραμέτρους του προαιρετικές και να χρησιμοποιήσουμε προεπιλεγμένες τιμές εάν ο χρήστης δε θέλει να δώσει τιμές σε τέτοιες παραμέτρους. Αυτό μπορεί να επιτευχθεί με τη βοήθεια των προεπιλεγμένων τιμών ορίσματος. Μπορούν να καθοριστούν προεπιλεγμένες τιμές ορισμάτων για παραμέτρους, τοποθετώντας μετά το όνομα της παραμέτρου στον ορισμό της συνάρτησης τον τελεστή εκχώρησης (=) να ακολουθείται απο την προεπιλεγμένη τιμή. Η προεπιλεγμένη τιμή ορίσματος πρέπει να είναι μια σταθερά. Για την ακρίβεια η προεπιλεγμένη τιμή ορίσματος πρέπει να είναι αμετάβλητη.

Παράδειγμα:

```
def say(message, times = 1):
	print(message * times)
say('Hello')
```

```
say('World', 5)
```
Έξοδος:
```
Hello
WorldWorldWorldWorldWorld
```
Σχολιασμός κώδικα:

Η συνάρτηση με το όνομα say χρησιμοποιείται για να τυπώσει μια συμβολοσειρά, τόσες φορές όσες έχει καθοριστεί. Εάν δεν έχει δοθεί τιμή, τότε από προεπιλογή η συμβολοσειρά τυπώνεται μια φορά. Αυτό το πετυχαίνουμε καθορίζοντας μια προεπιλεγμένη τιμή ίση με 1 για τη παράμετρο times.

Στην πρώτη χρήση της συνάρτησης say παρέχουμε μόνο τη συμβολοσειρά και τυπώνει τη συμβολοσειρά μόνο μια φορά. Στη δεύτερη χρήση της say παρέχουμε και τη συμβολοσειρά και ένα όρισμα 5 δηλώνοντας έτσι ότι θέλουμε να πούμε (say) το μήνυμα της συμβολοσειράς 5 φορές.

Συμβολοσειρές τεκμηρίωσης (DocStrings)

Η Python έχει ένα θαυμάσιο χαρακτηριστικό που ονομάζεται συμβολοσειρές τεκμηρίωσης (documentation strings) και συνήθως αναφέρεται με τη συντομογραφία DocStrings. Οι συμβολοσειρές τεκμηρίωσης είναι ένα σπουδαίο εργαλείο που πρέπει να χρησιμοποιείτε διότι βοηθάει να τεκμηριώσουμε το πρόγραμμα καλύτερα και έτσι γίνεται πιο εύκολα κατανοητό. Το εντυπωσιακό είναι ότι μπορούμε να πάρουμε επιστροφή τη συμβολοσειρά τεκμηρίωσης από μια συνάρτηση για παράδειγμα, ενώ το πρόγραμμα πραγματικά τρέχει!

Παράδειγμα:
```
def printMax(x, y):
        '''Prints the maximum of two numbers.
        Οι δύο τιμές πρέπει να είναι ακέραιοι αριθμοί.'''
        if x > y:
                print(x, 'is maximum')
        else:
                print(y, 'is maximum')

printMax(3, 5)
print(printMax.__doc__)
```
Έξοδος:
```
5 is maximum
Prints the maximum of two numbers.
 Οι δύο τιμές πρέπει να είναι ακέραιοι αριθμοί.
```
Σχολιασμός κώδικα:

Μία συμβολοσειρά στην πρώτη λογική γραμμή της συνάρτησης είναι η συμβολοσειρά τεκμηρίωσης για αυτή τη συνάρτηση. Μπορούμε να έχουμε πρόσβαση στη συμβολοσειρά τεκμηρίωσης της συνάρτησης printMax χρησιμοποιώντας το ιδιοχαρακτηριστικό (ονομασία που ανήκει σε) __doc__ της συνάρτησης.

Μερικά αυτοματοποιημένα εργαλεία μπορούν να ανακτήσουν την τεκμηρίωση από το πρόγραμμα με αυτόν τον τρόπο. Συνεπώς συνιστάται να χρησιμοποιούνται συμβολοσειρές τεκμηρίωσης σε κάθε *μη τετριμμένη* συνάρτηση.

4.6 ΕΙΣΟΔΟΣ – ΕΞΟΔΟΣ

Θα υπάρξουν καταστάσεις όπου το πρόγραμμα πρέπει να αλληλεπιδράσει με το χρήστη. Για παράδειγμα, θέλουμε να πάρουμε είσοδο από το χρήστη και μετά να τυπώσουμε πίσω μερικά αποτελέσματα. Μπορούμε να το επιτύχουμε αυτό χρησιμοποιώντας αντίστοιχα τις συναρτήσεις input() και print().

Ένας ακόμα συνηθισμένος τύπος εισόδου/εξόδου είναι ο χειρισμός των αρχείων (files). Η ικανότητα να δημιουργούμε, διαβάζουμε και να γράφουμε αρχεία είναι βασική σε πολλά προγράμματα και θα εξερευνήσουμε αυτή την πτυχή σε αυτό το υποκεφάλαιο.

Είσοδος από το χρήστη

Παράδειγμα:

```
def reverse(text):
        return text[::-1]

def is_palindrome(text):
        return text == reverse(text)

something = input('Enter text: ')
if (is_palindrome(something)):
        print("Yes, it is a palindrome")
else:
        print("No, it is not a palindrome")
```

Έξοδος:

```
Enter text: sir
No, it is not a palindrome
>>> (εκ νέου εκτέλεση)
Enter text: madam
Yes, it is a palindrome
>>> (εκ νέου εκτέλεση)
Enter text: racecar
Yes, it is a palindrome
```

Σχολιασμός κώδικα:

Χρησιμοποιούμε τον τεμαχισμό (κομμάτιασμα) για να αναστρέψουμε το κείμενο. Έχουμε ήδη δει πώς μπορούμε να κάνουμε κομμάτια από ακολουθίες, χρησιμοποιώντας τον κώδικα seq[a:b], αρχίζοντας από τη θέση a μέχρι τη θέση b. Μπορούμε επίσης να δώσουμε ένα τρίτο όρισμα το οποίο προσδιορίζει το βήμα με το οποίο γίνεται το κομμάτιασμα. Το προκαθορισμένο βήμα είναι το 1 εξαιτίας του οποίου επιστρέφει ένα συνεχές τμήμα του κειμένου. Δίνοντας ένα αρνητικό βήμα δηλ. -1, θα επιστρέψει το κείμενο ανάστροφα.

Η συνάρτηση input() παίρνει μια συμβολοσειρά σαν όρισμα και το παρουσιάζει στον χρήστη. Τότε περιμένει το χρήστη να τυπώσει κάτι και να πιέσει το return. Άπαξ και ο χρήστης έχει εισάγει κάτι, η συνάρτηση input() τότε θα επιστρέψει αυτό το κείμενο.

Παίρνουμε αυτό το κείμενο και το αναστρέφουμε. Εάν το αυθεντικό κείμενο και το ανεστραμμένο είναι ίσα, τότε το κείμενο είναι ένα παλίνδρομο.

Αρχεία

Μπορούμε να ανοίξουμε και να χρησιμοποιήσουμε αρχεία για διάβασμα ή γράψιμο, δημιουργώντας ένα αντικείμενο της κλάσης file και χρησιμοποιώντας τις μεθόδους της, read, readline ή write κατάλληλα για να διαβάσει από ή να γράψει στο αρχείο. Η ικανότητα να διαβάζει ή να γράφει στο αρχείο εξαρτάται από τον τρόπο (mode) που έχει καθοριστεί για το άνοιγμα του αρχείου. Στο τέλος, καλούμε τη μέθοδο close για να δηλώσουμε ότι τελειώσαμε με τη χρήση του αρχείου και να γίνει απελευθέρωση των αντίστοιχων πόρων συστήματος.

Παράδειγμα:

```
poem = '''\
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
'''
f = open('poem.txt', 'w')
f.write(poem)
f.close()
f = open('poem.txt')
by default
while True:
        line = f.readline()
        if len(line) == 0:
                break
        print(line)
f.close()
```

Έξοδος:

```
Programming is fun
When the work is done
if you wanna make your work also fun:
use Python!
```

Σχολιασμός κώδικα:

Αρχικά ανοίγουμε ένα αρχείο χρησιμοποιώντας την ενσωματωμένη συνάρτηση open καθορίζοντας την ονομασία του αρχείου και τον τρόπο με τον οποίο θέλουμε να ανοίγει το αρχείο. Ο τρόπος μπορεί να είναι με διάβασμα ('r', read mode), με γράψιμο ('w', write mode) ή με πρόσθεση ('a', append mode). Μπορούμε επίσης να χειριστούμε ένα αρχείο κειμένου ('t', text file) ή ένα δυαδικό αρχείο ('b', binary file). Στην πραγματικότητα υπάρχουν πάρα πολλοί τρόποι διαθέσιμοι και η help(open) θα

μας δώσει περισσότερες λεπτομέρειες γι αυτούς. Από προεπιλογή η open() θεωρεί το αρχείο ως αρχείο κειμένου ('t'ext file) και το ανοίγει με τον τύπο 'r'ead.

Στο δικό μας παράδειγμα, αρχικά ανοίγουμε το αρχείο σε εγγραφή και χρησιμοποιούμε τη μέθοδο write του αντικειμένου του αρχείου, για να γράψουμε στο αρχείο και τότε τελικά κλείνουμε (close) το αρχείο. Κατόπιν ανοίγουμε το ίδιο αρχείο πάλι για ανάγνωση. Δε χρειάζεται να καθορίσουμε έναν τύπο, γιατί η 'ανάγνωση' είναι ο προκαθορισμένος τρόπος. Διαβάζουμε κάθε γραμμή του αρχείου χρησιμοποιώντας τη μέθοδο readline σε βρόχο. Αυτή η μέθοδος επιστρέφει μια ολοκληρωμένη γραμμή περιλαμβάνοντας το χαρακτήρα νέας γραμμής (newline character) στο τέλος της γραμμής. Όταν μια άδεια συμβολοσειρά επιστρέφεται, σημαίνει ότι έχουμε φθάσει στο τέλος του αρχείου και 'ξεφεύγουμε' (break) από το βρόχο.

Από προεπιλογή η συνάρτηση print() τυπώνει το κείμενο καθώς και μια αυτόματη νέα γραμμή (newline) στην οθόνη. Τέλος, τελικά κλείνουμε (close) το αρχείο.

ΚΕΦΑΛΑΙΟ 5

Σχεδιασμός και υλοποίηση αλγόριθμου Bellman-ford

Σε αυτό το κεφάλαιο θα γίνει αρχικά επεξήγηση του πηγαίου κώδικα, ενώ στη συνέχεια θα δούμε και τη συμπεριφορά του σε συγκεκριμένα δίκτυα αλλά και πως αντιμετωπίζει πιθανές απώλειες σύνδεσης μεταξύ δύο κόμβων.

5.1 ΥΛΟΠΟΙΗΣΗ, ΠΕΡΙΓΡΑΦΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ

Ο πηγαίος κώδικας, ο οποίος παρατίθεται πλήρως στο παράρτημα περιλαμβάνει 3 βασικές συναρτήσεις (`draw_graph`, `draw_right_table`, `bellman_ford`) οι οποίες επιλύουν τον αλγόριθμο Bellman-Ford για δοθέν δίκτυο, ενώ υπάρχουν και 3 επιπλέον συναρτήσεις για τις δοκιμές μας (`example_01_presentation_example`, `example_02_straight_line`, `example_03_complex_network`) και θα σχολιαστούν στο επόμενο υποκεφάλαιο.

bellman_ford()

```
def bellman_ford(graph, source, graph_name):
```

Η συνάρτηση `bellman_ford` δέχεται 3 παραμέτρους για την σωστή κλήση συνάρτησης. Η πρώτη παράμετρος εποτελεί τον γράφο πάνω στον οποίο θα γίνει επίλυση του αλγορίθμου. Ο γράφος είναι σε μορφή "λεξικό από λεξικά", όπου κάθε κλειδί αποτελεί έναν κόμβο X και η πληροφορία του κλειδιού είναι ένα λεξικό κάθε κλειδί του οποίου υποδηλώνει τη σύνδεση με κόμβο Y και η αντίστοιχη πληροφορία

σημειώνει το βάρος της σύνδεσης. Για παράδειγμα, βλέπετε παρακάτω πως μπορούμε να δηλώσουμε τον γράφο που φαίνεται στην εικόνα.

```
    graph = {
        'a': {'b': 1},
        'b': {'c': 1},
        'c': {'d': 1},
        'd': {'e': 1},
        'e': {}
        }
```

Η δεύτερη παράμετρος δηλώνει ποια θα είναι η αφετηρία για την επίλυση του γράφου, ενώ η τρίτη παράμετρος αποτελεί περιγραφικό όνομα του γράφου ώστε να μπει ως τίτλος στα αρχεία εξόδου.

Στην συνέχεια, δημιουργείται το αρχείου εξόδου σε μορφή HTML για εύκολη προβολή του αποτελέσματος σε φυλλομετρητή. Επιγραμματικά γίνεται εκτύπωση ενός αρχικού τίτλου ενώ ακολουθεί και η εκτύπωση του πλήρη γράφου χρησιμοποιώντας την συνάρτηση `draw_graph` που θα σχολιάσουμε σε λίγο. Τέλος, εκτυπώνεται και ο πίνακας όλων των συνδέσεων ώστε ο μελετητής να έχει εύκολη πρόσβαση στις συνδέσεις του γράφου σε μορφή λίστας.

```
    # Dimiourgia arxeiou eksodou
    f = open('%s-result.html' % graph_name, 'w')
    f.write('<html>')
        ....
    f.write('</pre></td>')
    f.write('</tr>')
```

Σε αυτό το σημείο, ξεκινά η επίλυση του αλγορίθμου Blellman-Form με το πρώτο στάδιο της αρχικοποίησης όλων των απαραίτητων μεταβλητών. Η μεταβλητή d μας δίνει το κόστος μετακίνησης από τον αρχικό κόμβο προς όλους τους υπόλοιπους κόμβους ενώ η μεταβλητή p αναφέρει για κάθε κόμβο ποιος είναι ο πατρικός του, στην βέλτιστη διαδρομή. Τέλος, η αρχικοποίηση περιλαμβάνει και τον ορισμό της απόστασης από την αφετηρία μέχρι και τον ίδιο τον κόμβο με μηδέν.

```
    # Step 1: For each node prepare the destination and predecessor
    d = {} # Stands for destination
    p = {} # Stands for predecessor
    for node in graph:
        d[node] = float('Inf') # We start admiting that the rest of nodes are very
very far
        p[node] = None

    # For the source we know how to reach
    d[source] = 0
```

Μετά το βήμα της αρχικοποίησης, ακολουθεί το δεύτερο βήμα του αλγορίθμου και αυτό δεν είναι άλλο παρά η επίλυση του γράφου και η αναζήτηση της

βέλτιστης διαδρομής από τον κόμβο αφετηρίας προς όλους τους υπόλοιπους κόμβους του γράφου (αναλυτικά ο κώδικας βρίσκεται στο παράρτημα).

```
    # Step 2: relax edges repeatedly
    for i in range(len(graph)): #Run this until is converges
        for u in graph:
            for v in graph[u]: #For each neighbour of u
                # If the distance between the node and the neighbour
                # ... is lower than the one I have now
                if d[u] + graph[u][v] < d[v]:
        ...

                    f.write(draw_right_table(graph, source, d, p, v))
                    f.write('</pre></td>')
                    f.write('</tr>')
```

Η διαδικασία που ακολουθείται είναι αρκετά απλή στην υλοποίηση αλλά βοηθάει στην εύκολη κατανόηση του αλγορίθμου. Οι επαναλήψεις βοηθούν ώστε να έχουμε το κατάλληλο αριθμό εκτελέσεων και ο αλγόριθμος να συγκλίνει επιτυχώς, ενώ σε κάθε επανάληψη γίνεται έλεγχος όλων των κόμβων και όλων των επόμενων συνδέσεών τους για το αν βρέθηκε μια νέα διαδρομή,ακόμα πιο γρήγορη. Αν σε κάποιο βήμα, βρεθεί βελτιωμένη διαδρομή, τότε αυτόματα καταχωρείται στο αρχείο εξόδου με την κατάλληλη δημιουργία εικόνας, όπως θα εξηγήσουμε και παρακάτω.

Στην συνέχεια, και για λόγους πληρότητας, εκτυπώνεται ο γράφος με όλα τα βέλτιστα μονοπάτια, και τέλος, ακολουθεί το τελευταίο βήμα του αλγορίθμου το οποίο είναι ο έλεγχος αρνητικών μονοπατιών, όπως αναφέρεται και στην θεωρία.

```
    # Step 3: check for negative-weight cycles
    for u in graph:
        for v in graph[u]:
            if d[u] + graph[u][v] < d[v]:
                print "Graph contains a negative circle"
                f.write("Graph contains a negative circle")

    f.write('</center></body></html>')
    f.close()
    return d, p
```

draw_right_table()

```
def draw_right_table(graph, source, d, p, cv):
```

Η συνάρτηση `draw_right_table` δέχεται 4 παραμέτρους για την σωστή εκτέλεσή της και σκοπός της είναι η προβολή σε μορφή κειμένου της λίστας βέλτιστων συνδέσεων μεταξύ των κόμβων από το κόμβο αφετηρίας. Οι 4 πρώτες είναι βασικές σχετικά με την επίλυση του αλγορίθμου (κόμβος αφετηρία, πλήρης γράφος, μεταβλητές d και p σύμφωνα με την επίλυση). Η 5η παράμετρος απλά

υποδηλώνει σε ποιο κόμβο μόλις έγινε αλλαγή βέλτιστου μονοπατιού. Με αυτό το τρόπο, η εκτύπωση δίνει ένα οπτικό πλεονέκτημα και ο μελετητής μπορεί εύκολα να ακολουθήσει τη ροή του αλγορίθμου. Ο κώδικας βρίσκεται αναλυτικά στο παράρτημα ενώ στο παράρτημα Β φαίνεται στα δεξιά το αποτέλεσμα εκτέλεσης της συνάρτησης σε μορφή πίνακα.

draw_graph()

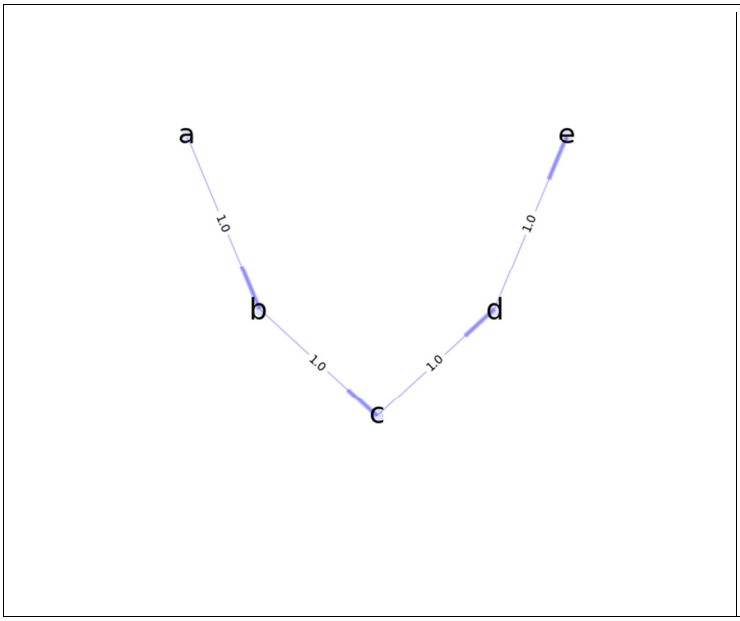
```
# network visualization
# draw_graph function
def draw_graph(graph_name, graph, labels=None, graph_layout='spectral',
               node_size=100, node_color='blue', node_alpha=0.1,
               node_text_size=24,
               edge_color='blue', edge_alpha=0.3, edge_tickness=1,
               edge_text_pos=0.5,
               text_font='sans-serif', edge_labels = None, d = None, p = None):
```

Η συνάρτηση `draw_graph` δέχεται πληθώρα παραμέτρων οι οποίες σχετίζονται με την εμφάνιση του γράφου, όπως για παράδειγμα χρώμα και πάχος σύνδεσης, μέγεθος κειμένων κ.α. Ως προαπαιτούμενο, θα πρέπει να είναι εγκατεστημένα δύο επιπλέον βιβλιοθήκες, οι `networkx` για διαχείριση γράφων και `matplotlib` για απεικόνιση γραφημάτων γενικού σκοπού. Η βασική λειτουργία της χρειάζεται τουλάχιστον το όνομα του γράφου αλλά και πλήρως τον γράφο με τις συνδέσεις του (`graph_name`, `graph`). Παρόλαυτα, για την πληρέστερη απεικόνιση του γράφου στην εκάστοτε κατάσταση, χρειαζόμαστε και τις παραμέτρους edge_labels, d, p. Η συνάρτηση αναλύεται μέσα από τα σχόλια σε κάθε εντολή και βρίσκεται αναλυτικά στο παράρτημα.

Με την ολοκλήρωση της συνάρτησης, ένα νέο αρχείο δημιουργείται με επέκταση PNG με μοναδικό όνομα για αποφυγή καταστροφής αρχείων εικόνων από διαφορετική εκτέλεση, και επιστρέφεται το μοναδικό αυτό όνομα στον καλούντα ώστε να χρησιμοποιηθεί στην προβολή του αποτελέσματος. Αναλυτικά, αποτέλεσμα της εκτέλεσης φαίνεται στην συνάρτηση `bellman_ford` όπως σχολιάστηκε πιο πάνω ή στο παράρτημα Β.

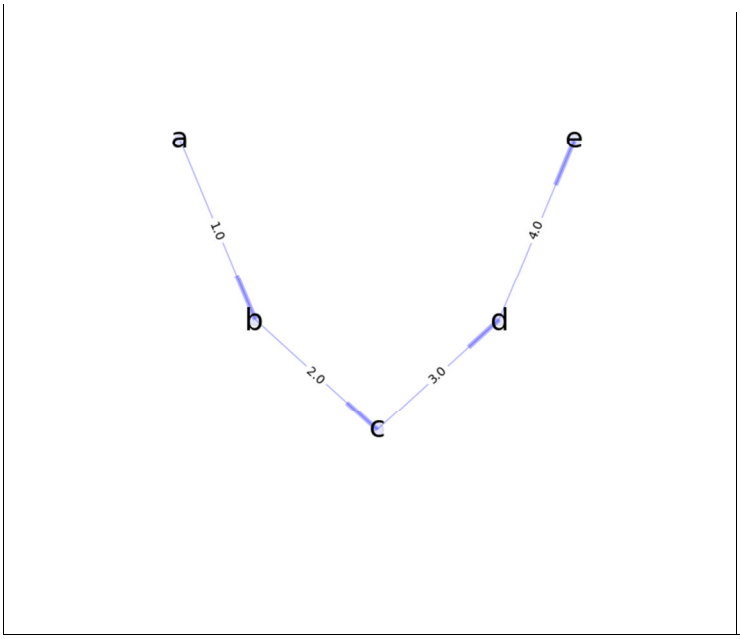
5.2 ΜΕΛΕΤΗ ΛΕΙΤΟΥΡΓΙΑΣ

Στην συνέχεια του κεφαλαίου θα μελετήσουμε τη συμπεριφορά του αλγορίθμου σε δύο συγκεκριμένα δίκτυα, αλλά και πως ο ίδιος ο αλγόριθμος αντιμετωπίζει μια πιθανή κατάρρευση σύνδεσης δύο κόμβων. Αρχικά ας δούμε ένα απλό δίκτυο όπου κάθε κόμβος συνδέεται σε άλλον ένα κόμβο και μόνο (δεν υπάρχουν κόμβοι μη συνδεδεμένοι) ώστε να εξοικειωθούμε καταλλήλως με τις εικόνες και τα στατιστικά προτού προχωρήσουμε σε ένα πιο περίπλοκο δίκτυο. Αριστερά βλέπουμε το δίκτυο με τα βάρη μεταξύ κάθε κόμβων, ενώ με έντονη γραμμή υποδηλώνεται ο τερματικός κόμβος (δηλαδή τη φορά) σύνδεσης μεταξύ των κόμβων. Αντίστοιχα, δεξιά παρουσιάζονται οι ίδες πληροφορίες σε μορφή πίνακα, ποιος κόμβος συνδέεται σε ποιον κόμβο και με τι βάρος.




```
a ->   b   1.0
b ->   c   1.0
c ->   d   1.0
d ->   e   1.0
```

Για λόγους απλότητας, έχουμε ορίσει ως βάρος κάθε σύνδεσης με το ίδιο βάρος και ισούται με τη μονάδα. Με την ολοκλήρωσης εκτέλεσης του αλγορίθμου, καταλήγουμε στην εξής κατάσταση.

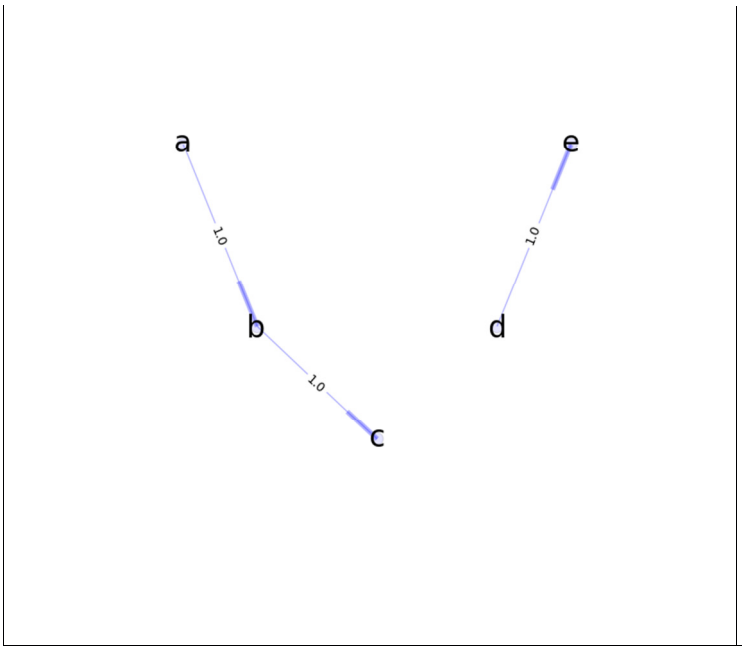


v	distTo[]	edgeTo[]
a	0.0	-
b	1.0	a -> b
c	2.0	b -> c
d	3.0	c -> d
e	4.0	d -> e

Παρατηρούμε πως στο τελικό δίκτυο όπου έχει γίνει και καταγραφή της βέλτιστης λύσης (που αποτελεί και μοναδικό μονοπάτι), αναγράφεται μεταξύ των κόμβων το ελάχιστο δυνατό κόστος που χρειάζεται η μετακίνηση από τον κόμβο αφετηρίας a προς οποιονδήποτε άλλο κόμβο. Αντίστοιχα, δεξιά στον πίνακα παρουσιάζεται η ίδια πληροφορία σε μορφή λίστας. Η πρώτη στήλη αποτελεί τον

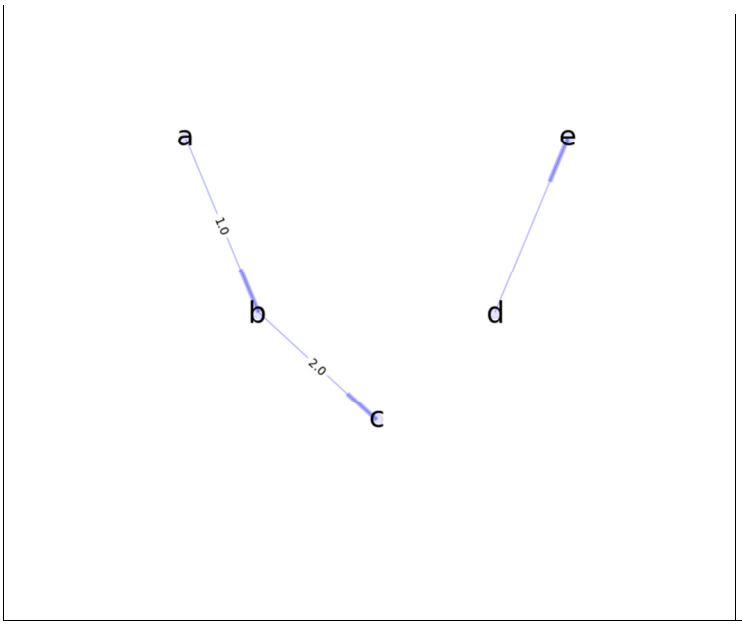
κόμβο τερματισμού, η δεύτερη στήλη αποτελεί το συνολικό κόστος για να καταλήξουμε στον συγκεκριμένο κόμβο μέσω του βέλτιστου μονοπατιού, ενώ η τρίτη στήλη δείχνει ποιος είναι ο κόμβος προέλευσης για το βέλτιστο μονοπάτι.

Στη συνέχεια, έστω ότι κόβεται η σύνδεση μεταξύ των κόμβων c και d. Ως εκ τούτου έχουμε αντίστοιχα το παρακάτω δίκτυο.




```
a ->   b   1.0
b ->   c   1.0
d ->   e   1.0
```

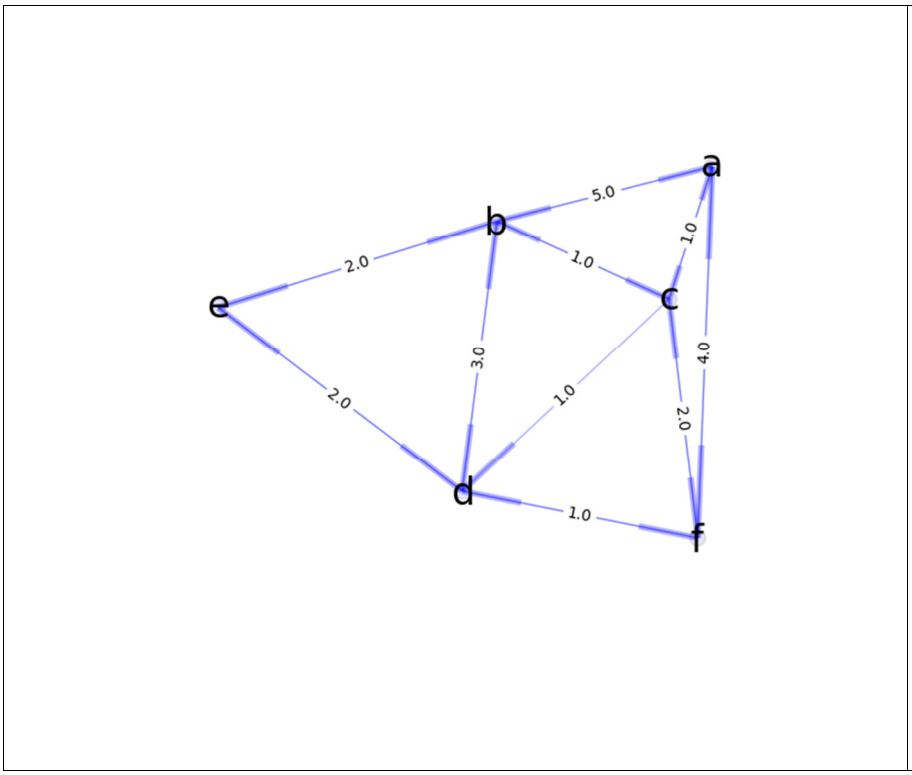
Η εκτέλεση του αλγορίθμου πλέον μπορεί να ολοκληρωθεί αλλά με το εξής τελικό αποτέλεσμα.



v	distTo[]	edgeTo[]
a	0.0	-
b	1.0	a -> b
c	2.0	b -> c
d		
e		

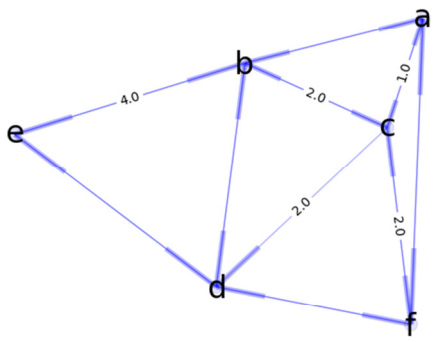
Βλέπουμε πως, όπως ήταν αναμενόμενο, δεν υπάρχει βέλτιστο μονοπάτι από τον κόμβο a προς τους κόμβους d και e, αφού δεν υπάρχει και μονοπάτι ευθύς εξαρχής. Δεξιά, αντίστοιχα, φαίνονται τα στοιχεία δρομολόγησης για όλους τους κόμβους που ο αλγόριθμος μπόρεσε να βρει βέλτιστη διαδρομή. Στην συνέχεια, ας μελετήσουμε ένα πιο περίπλοκο δίκτυο και πως συμπεριφέρεται ο αλγόριθμος αν χαθεί μια σύνδεση στο βέλτιστο μονοπάτι.

Το παρακάτω δίκτυο αποτελείται από 6 κόμβους, και για κάθε σύνδεση μεταξύ των κόμβων έχει επιλεχθεί τυχαίο βάρος.



a ->	c	1.0
a ->	b	5.0
a ->	f	4.0
b ->	a	2.0
b ->	c	1.0
b ->	e	2.0
b ->	d	3.0
c ->	a	1.0
c ->	b	1.0
c ->	d	1.0
c ->	f	1.0
d ->	b	1.0
d ->	e	2.0
d ->	f	3.0
e ->	b	1.0
e ->	d	1.0
f ->	a	4.0
f ->	c	2.0
f ->	d	1.0

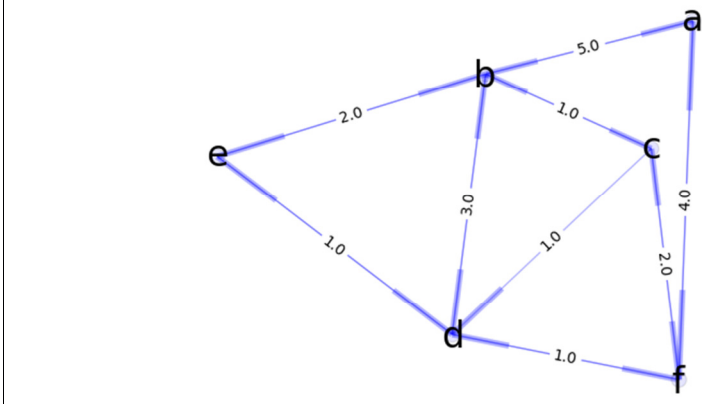
Μετά την εκτέλεση του αλγορίθμου, του οποίου αναλυτικώς έχει γίνει καταγραφεί και παρουσιάζεται στο Παράρτημα Β, έχουμε τα εξής βέλτιστα μονοπάτια από το κόμβο αφετηρίας a.



v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d	2.0	c -> d
e	4.0	b -> e
f	2.0	c -> f

Συγκεκριμένα, τα μονοπάτια με το ελάχιστο κόστος είναι τα εξής για τους αντίστοιχους τερματικούς κόμβους `a -> c -> f`, `a -> c -> d` και `a -> c -> b -> e`. Αντίστοιχα, δεξιά στον πίνακα φαίνεται η ίδια πληροφορία σε μορφή λίστας ενώ καταγράφεται και το συνολικό κόστος.

Στην συνέχεια, και για τυχαίο λόγο καταρρέει η σύνδεση μεταξύ των κόμβων a και c, η οποία όμως αποτελεί και μέρος του βέλτιστου μονοπατιού, ως εκ τούτου, ο ενημερωμένος γράφος μας είναι ο εξής.

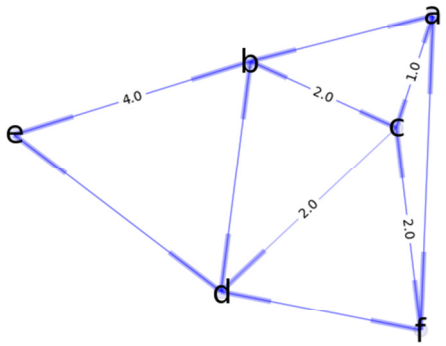


a ->	b	5.0
a ->	f	4.0
b ->	a	2.0
b ->	c	1.0
b ->	e	2.0
b ->	d	3.0
c ->	b	1.0
c ->	d	1.0
c ->	f	1.0
d ->	b	1.0
d ->	e	2.0
d ->	f	3.0
e ->	b	1.0
e ->	d	1.0
f ->	a	4.0
f ->	c	2.0
f ->	d	1.0

Ακολουθεί η επίλυση του γράφου με χρήση του αλγορίθμου Bellman-Form, κατά τα γνωστά. Στην συνέχεια, έχουν τοποθετηθεί σε αντιπαραβολή οι δύο περιπτώσεις “πριν” και “μετά” την κατάρρευση της σύνδεσης μεταξύ των κόμβων a και c.

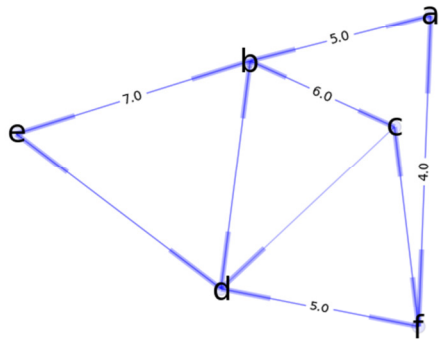
Στον πίνακα που ακολουθεί γίνεται σαφές πως ο αλγόριθμος έχει αναπροσαρμόσει τις βέλτιστες διαδρομές με τα αντίστοιχα βάρη. Πλέον, τα μονοπάτια με το ελάχιστο κόστος είναι τα εξής για τους αντίστοιχους τερματικούς κόμβους `a -> f -> d`, `a -> b -> c` και `a -> b -> e`.

Πριν



v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d	2.0	c -> d
e	4.0	b -> e
f	2.0	c -> f

Μετά



v	distTo[]	edgeTo[]
a	0.0	-
b	5.0	a -> b
c	6.0	b -> c
d	5.0	f -> d
e	7.0	b -> e
f	4.0	a -> f

ΚΕΦΑΛΑΙΟ 6

Συμπεράσματα

Ο αλγόριθμος των Belmman-Ford είναι αντιπροσωπευτικός της κατηγορίας των δυναμικών αλγορίθμων που βασίζονται σε διάνυσμα-απόσταση. Σε αυτή την κατηγορία, κάθε δρομολογητής του δικτύου διατηρεί έναν πίνακα με μία εγγραφή για κάθε άλλο κόμβο του δικτύου. Η εγγραφή αυτή περιέχει την καλύτερη γνωστή απόσταση προς κάθε προορισμό και τη διαδρομή προς αυτόν. Η ενημέρωση των πινάκων πραγματοποιείται με ανταλλαγή πληροφοριών μεταξύ γειτόνων του δικτύου. Η συντομότερη διαδρομή (καλύτερη γνωστή διαδρομή) μπορεί να βασίζεται:

- στο πλήθος των ενδιάμεσων αλμάτων
- στη χρονική καθυστέρηση
- στο συνολικό πλήθος των πακέτων που βρίσκονται στις ουρές κατά μήκος της διαδρομής

Στην μελέτη μας, θεωρήσαμε πως η συντομότερη διαδρομή βασίζεται σε ένα βάρος ανά σύνδεση που στην απλουστευμένη του μορφή αποτελεί έναν ακέραιο αριθμό για κάθε σύνδεση κόμβων. Η υλοποίηση του αλγορίθμου, εμφάνισε την συμπεριφορά του σε διαφορετικά δίκτυα αλλά και πως αντιμετωπίζει πιθανά προβλήματα που προκύπτουν κατά την επικοινωνία, υπολογίζοντας εκ νέου τους πίνακες δρομολόγησης με τα βέλτιστα μονοπάτια.

Στα πλαίσια της εργασίας, έγινε προσπάθεια η προβολή προσομοίωσης του δικτύου να είναι όσο το δυνατόν πιο αναλυτική ώστε να μπορεί να χρησιμοποιηθεί για εκπαιδευτικό σκοπό και να βοηθήσει άλλους φοιτητές να κατανοήσουν πλήρως τη λειτουργία του αλγορίθμου στη πράξη. Η προβολή γίνεται σε μορφή HTML που μπορεί να αναγνωσθεί από όλους τους γνωστούς φυλλομετρητές, οπότε τα αποτελέσματα γίνονται εμφανή όχι μόνο σε τοπικό υπολογιστή αλλά και μέσω διαδικτύου.

Βιβλιογραφία

- Python Official Documentation
  - https://www.python.org/
- Python programming, Αντωνία Τερζίδου (2013),
  - https://arch.icte.uowm.gr/docs/Python_Programming_Full_Book_Dasygenis_Terzidou.pdf
- Think Python, Πώς να Σκέφτεσαι σαν Επιστήμονας της Πληροφορικής, Allen Downey, Μετάφραση-Επιμέλεια: Ποικιλίδης Ζαχαρίας (2014)
- Αλγορίθμικά Θέματα δικτύων και Τηλεματικής, Συγκριτική μελέτη αλγορίθμων δρομολόγησης, Παναγιώτα Κατσικούλη,
  - http://ru6.cti.gr/ru6/system/files/bouras_site/ergasies_foithtwn/234_Routing_Algorithms.pdf
- Bellman–Ford algorithm,
  - https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
- Dijkstra's algorithm
  - https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- Classful network
  - https://en.wikipedia.org/wiki/Classful_network

Παράρτημα

Πηγαίος κώδικας

```
import networkx as nx

import matplotlib.pyplot as plt

graph_pos = None
number_of_images = 0

# network visualization
# draw_graph function
def draw_graph(graph_name, graph, labels=None, graph_layout='spectral',
               node_size=100, node_color='blue', node_alpha=0.1,
               node_text_size=24,
               edge_color='blue', edge_alpha=0.3, edge_tickness=1,
               edge_text_pos=0.5,
               text_font='sans-serif', edge_labels = None, d = None, p = None):

    global number_of_images

    # Gia na kratisw oles tis eikones
    number_of_images += 1

    # Ekatharisi plot
    plt.figure()
    plt.axis('off')

    # create networkx graph
    G=nx.DiGraph()

    # add edges
    for edge_from in graph:
        for edge_to in graph[edge_from]:
            G.add_edge(edge_from, edge_to)

    # these are different layouts for the network you may try
    # shell seems to work best
    global graph_pos
    if not graph_pos:
        # Gia na min alazei i topologia kathe fora
        if graph_layout == 'spring':
            graph_pos=nx.spring_layout(G)
        elif graph_layout == 'spectral':
            graph_pos=nx.spectral_layout(G)
        elif graph_layout == 'random':
            graph_pos=nx.random_layout(G)
        else:
            graph_pos=nx.shell_layout(G)


    # draw graph
    nx.draw_networkx_nodes(G, graph_pos, node_size=node_size, alpha=node_alpha,
node_color=node_color)
    nx.draw_networkx_edges(G, graph_pos, width=edge_tickness, alpha=edge_alpha,
edge_color=edge_color)
    nx.draw_networkx_labels(G, graph_pos, font_size=node_text_size,
font_family=text_font)

    if labels is None:
        labels = range(len(graph))

    # edge_labels = dict(zip(graph, labels))
    edge_labels_to_show = {}


    for edge_from in graph:
```

```
        if len(graph[edge_from]) <= 0:
            continue

        # edge_labels_to_show[ edge_from ] = {}
        for edge_to in graph[edge_from]:
            if edge_labels == 'all':
                edge_labels_to_show[ (edge_from, edge_to,) ] = '%.1f' %
float(graph[edge_from][edge_to])
            elif edge_labels == 'alld' and d and p and edge_to in p and p[edge_to] ==
edge_from:
                edge_labels_to_show[ (edge_from, edge_to,) ] = '%.1f' %
float(d[edge_to])
            elif edge_to in edge_labels and d and p and edge_to in p and p[edge_to] ==
edge_from:
                edge_labels_to_show[ (edge_from, edge_to,) ] = '%.1f (+%.1f)' %
(float(d[edge_to]), float(graph[edge_from][edge_to]))

    if len(edge_labels_to_show):
        nx.draw_networkx_edge_labels(G, graph_pos, edge_labels=edge_labels_to_show,
label_pos=edge_text_pos)

    # Apothikevw tin eikona mou
    plt.savefig('image-%s-%03d.png' % (graph_name, number_of_images))
    plt.close()

    # Gia na emfanistei sto html
    return 'image-%s-%03d.png' % (graph_name, number_of_images)

# Sinartisi pou me vasi ton iparxon grafo
# .. prospathw na ftiaksw enimerwtiko pinakaki
# ..  se morfi pinaka
#  graph: o pliris grafos
# source: an theloume na tonisoume enan sigkekrimeno arxiko kombo
#      d:
#      p: o komvos (veltisti) epilogi pou erxomaste
#     cv: komvos pou theloume na tonisoume
def draw_right_table(graph, source, d, p, cv):
    keys = graph.keys()
    keys.sort()
    out = "%3s %10s %10s" % ('v', 'distTo[]', 'edgeTo[]')
    out += "<br/>" + "-" * len(out) + "<br/>"

    for u in keys:
        t1 = ''
        t2 = ''
        if u == cv:
            t1 = '<span style="border: 1px solid black;">'
            t2 = '</span>'

        if u == source:
            out += t1 + "%3s %10.1f %10s" % (source, 0.0, '-') + t2 + "<br/>"
        elif d[u] == float('Inf'):
            out += t1 + "%3s %10s %10s" % (u, '', '') + t2 + "<br/>"
        else:
            out += t1 + "%3s %10.1f %10s" % (u, d[u], '%3s -> %3s' % (p[u], u)) + t2 +
"<br/>"


    return out

# Epiliei olo to grafo
# .. kai paralila dimiourgei to apotelesma
# .. html selida kai eikones
def bellman_ford(graph, source, graph_name):
    # Dimiourgia arxeiou eksodou
    f = open('%s-result.html' % graph_name, 'w')

    f.write('<html>')
    f.write('<head>')
    f.write('<title>%s: Bellman-Ford algorithm demo</title>' % graph_name)
    f.write('</head>')
    f.write('<body><center>')

    f.write('<h2>%s: Bellman-Ford algorithm demo</h2>' % graph_name)
    f.write('<table style="margin: 0 auto; width: 90%%;">')
```

```
    f.write('<tr><td colspan="2" style="text-align: center;"><hr/><h4>inital edge-
weighted digraph</h4></td></tr>')
    f.write('<tr>')
    f.write('<td style="width: 50%%; text-align: right;"><img src="%s" /></td>' %
draw_graph(graph_name, graph, edge_labels = 'all'))
    f.write('<td style="text-align: left; font-family: \'Courier New\', Courier,
monospace; width: 30%;"><pre>')
    keys = graph.keys()
    keys.sort()
    for u in keys:
        for v in graph[u]: #For each neighbour of u
            f.write('%3s -> %3s %5.1f</br>' % (u, v, float(graph[u][v])))
    f.write('</pre></td>')
    f.write('</tr>')

    # Step 1: For each node prepare the destination and predecessor
    d = {} # Stands for destination
    p = {} # Stands for predecessor
    for node in graph:
        d[node] = float('Inf') # We start admiting that the rest of nodes are very
very far
        p[node] = None

    # For the source we know how to reach
    d[source] = 0

    # List all passes
    list_of_edges = {}
    for u in graph:
        for v in graph[u]: #For each neighbour of u
            list_of_edges['%s -> %s' % (u, v)] = '0'

    # Step 2: relax edges repeatedly
    for i in range(len(graph)): #Run this until is converges
        for u in graph:
            for v in graph[u]: #For each neighbour of u
                # If the distance between the node and the neighbour is lower than the
one I have now
                if d[u] + graph[u][v] < d[v]:
                    # Record this lower distance
                    d[v]  = d[u] + graph[u][v]
                    p[v] = u

                    f.write('<tr><td colspan="2" style="text-align:
center;"><hr/><h4>pass %d</h4></td></tr>' % i)
                    f.write('<tr>')
                    f.write('<td valign="top" style="width: 50%%; text-align:
right;"><img src="%s" /></td>' % (draw_graph(graph_name, graph, edge_labels = [v,], d
= d, p = p)))
                    f.write('<td valign="top" style="text-align: left; font-family:
\'Courier New\', Courier, monospace"><pre>')
                    f.write(draw_right_table(graph, source, d, p, v))
                    f.write('</pre></td>')
                    f.write('</tr>')


    f.write('<tr><td colspan="2" style="text-align:
center;"><hr/><h4>Final</h4></td></tr>')

    f.write('<tr>')
    f.write('<td valign="top" style="width: 50%%; text-align: right;"><img src="%s"
/></td>' % (draw_graph(graph_name, graph, edge_labels = 'alld', d = d, p = p)))
    f.write('<td valign="top" style="text-align: left; font-family: \'Courier New\',
Courier, monospace"><pre>')
    f.write(draw_right_table(graph, source, d, p, ''))
    f.write('</pre></td>')
    f.write('</tr>')

    f.write('</table>')

    # Step 3: check for negative-weight cycles
    for u in graph:
        for v in graph[u]:
            if d[u] + graph[u][v] < d[v]:
                print "Graph contains a negative circle"
                f.write("Graph contains a negative circle")
```

```
    f.write('</center></body></html>')
    f.close()
    return d, p

def example_01_presentation_example():
    # To paradeigma apo to pdf
    global graph_pos, number_of_images

    graph_pos = None
    number_of_images = 0

    graph = {
        '0': {'1':  5, '7':  8, '4':  9},
        '1': {'3': 15, '2': 12, '7':  4},
        '2': {'3':  3, '6': 11},
        '3': {'6':  9},
        '4': {'7':  5, '5':  4, '6':  20},
        '5': {'2':  1, '6': 13},
        '6': {},
        '7': {'2':  7, '5':  6},
        }

    d, p = bellman_ford(graph, '0', 'example-01-presentation')

def example_02_straight_line():
    global graph_pos, number_of_images

    graph_pos = None
    number_of_images = 0

    graph = {
        'a': {'b': 1},
        'b': {'c': 1},
        'c': {'d': 1},
        'd': {'e': 1},
        'e': {}
        }

    d, p = bellman_ford(graph, 'a', 'example-02-straight-line')

    graph = {
        'a': {'b': 1},
        'b': {'c': 1},
        'c': {},
        'd': {'e': 1},
        'e': {}
        }

    d, p = bellman_ford(graph, 'a', 'example-02-straight-line-missing')

def example_03_complex_network():
    global graph_pos, number_of_images

    graph_pos = None
    number_of_images = 0

    graph = {
        'a': {'b': 5, 'c': 1, 'f': 4},
        'b': {'a': 2, 'c': 1, 'e': 2, 'd': 3},
        'c': {'a': 1, 'b': 1, 'd': 1, 'f': 1},
        'd': {'b': 1, 'e': 2, 'f': 3},
        'e': {'b': 1, 'd': 1},
        'f': {'a': 4, 'c': 2, 'd': 1},
        }

    d, p = bellman_ford(graph, 'a', 'example-03-complex-network')

    graph = {
        'a': {'b': 5,          'f': 4},
        'b': {'a': 2, 'c': 1, 'e': 2, 'd': 3},
        'c': {        'b': 1, 'd': 1, 'f': 1},
        'd': {'b': 1, 'e': 2, 'f': 3},
        'e': {'b': 1, 'd': 1},
        'f': {'a': 4, 'c': 2, 'd': 1},
        }

    d, p = bellman_ford(graph, 'a', 'example-03-complex-network-missing')
```

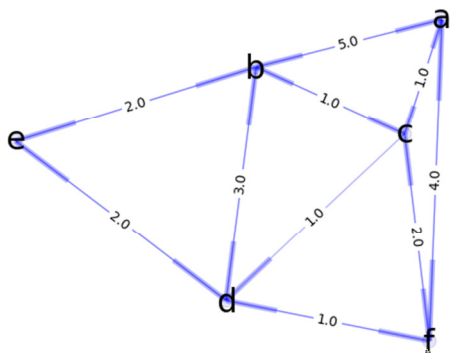
```
if __name__ == '__main__':
    example_01_presentation_example()
    example_02_straight_line()
    example_03_complex_network()
```

Παράρτημα Β

Αποτέλεσμα ενδεικτικής εκτέλεσης

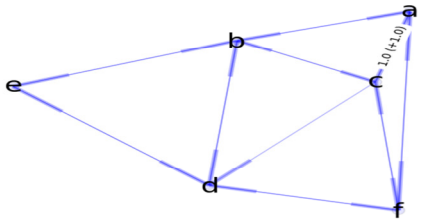
example-03-complex-network: Bellman-Ford algorithm demo

inital edge-weighted digraph



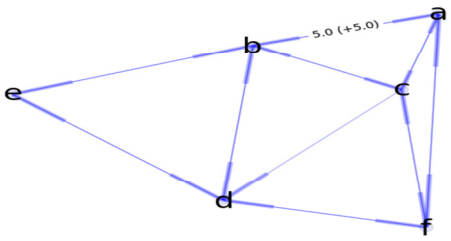

```
a ->   c   1.0
a ->   b   5.0
a ->   f   4.0
b ->   a   2.0
b ->   c   1.0
b ->   e   2.0
b ->   d   3.0
c ->   a   1.0
c ->   b   1.0
c ->   d   1.0
c ->   f   1.0
d ->   b   1.0
d ->   e   2.0
d ->   f   3.0
e ->   b   1.0
e ->   d   1.0
f ->   a   4.0
f ->   c   2.0
f ->   d   1.0
```

pass 0



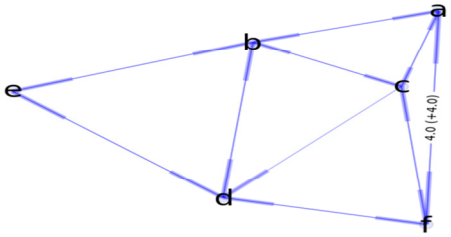
v	distTo[]	edgeTo[]
a	0.0	-
b		
c	1.0	a -> c
d		
e		
f		

pass 0



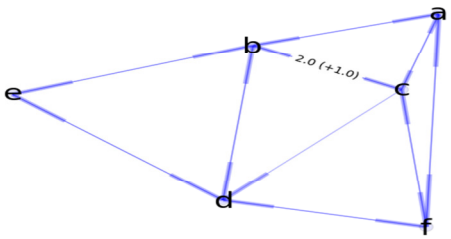
v	distTo[]	edgeTo[]
a	0.0	-
b	5.0	a -> b
c	1.0	a -> c
d		
e		
f		

pass 0



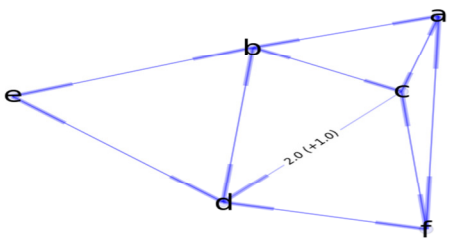
v	distTo[]	edgeTo[]
a	0.0	-
b	5.0	a -> b
c	1.0	a -> c
d		
e		
f	4.0	a -> f

pass 0



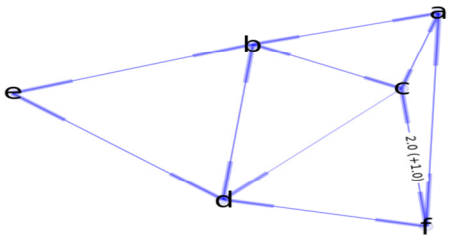
v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d		
e		
f	4.0	a -> f

pass 0



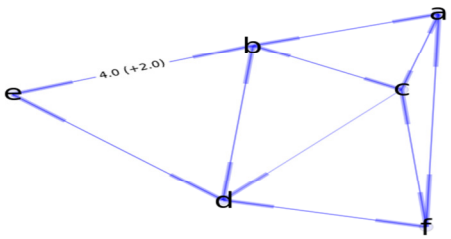
v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d	2.0	c -> d
e		
f	4.0	a -> f

pass 0



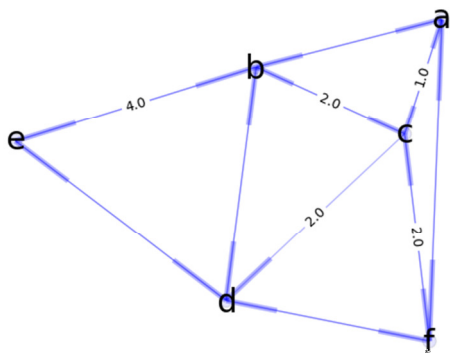
v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d	2.0	c -> d
e		
f	2.0	c -> f

pass 0



v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d	2.0	c -> d
e	4.0	b -> e
f	2.0	c -> f

Final



v	distTo[]	edgeTo[]
a	0.0	-
b	2.0	c -> b
c	1.0	a -> c
d	2.0	c -> d
e	4.0	b -> e
f	2.0	c -> f