Τεχνολογικό Εκπαιδευτικό Ίδρυμα Πελοποννήσου

Σχολή Τεχνολογικών Εφαρμογών (Έδρα Σπάρτη)

Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

NP- Πληρότητα και οι Αναγωγές

Στέργιος Στεργίου

Αριθμός Μητρώου:2006177

Επιβλέπων καθηγητής: Γρηγόριος Καραγιώργος

Σπάρτη, Νοέμβριος 2014

Πρόλογος

Η πτυχιακή αυτή έχει ως σκοπό την παρουσίαση και την σύντομη ιστορική μελέτη της ανάπτυξης των αλγορίθμων αλλά και της θεωρίας της πολυπλοκότητας για την επίλυση πραγματικών προβλημάτων. Θα αναφερθούμε στην απαρχή της πληροφορικής, από τα όνειρα της Αρχαιότητας στα τεχνητά όντα και στην σημερινή ανάπτυξη της χρήσης των υπολογιστών. Θα γίνει μια σύντομη περιγραφή της θεωρίας της πολυπλοκότητας ως του εργαλείου για την εκτίμηση της ύπαρξης ή όχι μηχανικής λύσης σε κάποιο πρόβλημα. Εάν ένα πρόβλημα, δηλαδή, λύνεται ακριβώς, σε πραγματικό χρόνο από κάποιον υπολογιστή. Η προσπάθεια για την επινόηση αλγορίθμων που θα αντικαθιστούν τους ανθρώπους στην λήψη αποφάσεων αλλά και στην παραγωγή γνώσης μπορεί να ενισχυθεί μέσω της θεωρίας της πολυπλοκότητας. Στην εργασία αυτή θα εξετάσουμε το πάθος του ανθρώπου για τη δημιουργία ενός μηχανικού υποκατάστατού του αλλά και τις δυσκολίες που συναντά προς την πραγματοποίηση αυτού του στόχου.

Ευχαριστίες

Από εδώ θα ήθελα να ευχαριστήσω ειλικρινά τον επιβλέποντα καθηγητή κ. Καραγιώργο για την σημαντική συμβολή του στην πραγματοποίηση της δεδομένης πτυχιακής εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τους συγγενείς και φίλους που με στήριξαν καθ' όλη την διάρκεια συλλογής των δεδομένων και καταγραφής αυτών, όπως και κατά την συγγραφή του πονήματος αυτού.

Στέργιος Στεργίου, Νοέμβριος 2014

Περιεχόμενα

Εισαγωγή

[Μια από τις παραδόσεις, από τις οποίες η Πληροφορική αντλεί ένα μέρος της ουσίας και της δυναμικής της, είναι η παράδοση των αυτόματων κατασκευών. Η τεχνητή επιθυμία για την κατασκευή μηχανών ή μηχανικών διατάξεων, που να διαθέτουν μια σχετική αυτονομία και να κινούνται «από το εσωτερικό τους», προέρχεται, χωρίς αμφιβολία, από την αγωνία των ανθρώπων να κατανοήσουν και να μεταφέρουν σε υλικά κατασκευάσματα μερικά από τα μυστικά της φύσης] (Breton, 1991). [Η κατασκευή των ρολογιών μαρτυρεί τη θέληση για να γίνει καταληπτή και να μπει σε τάξη η μέχρι τότε ασύλληπτη, όπως το ποτάμι που κυλά μπροστά από τα μάτια του ποιητή ή του Έλληνα φιλοσόφου, ροή του χρόνου. Η κατασκευή των αυτόματων μηχανών, στην αρχή με τη μορφή αυτομάτων με ανθρώπινα ή ζωώδη χαρακτηριστικά και έπειτα με τη μορφή αυτορρυθμιζόμενων βιομηχανικών μηχανών για παραγωγικές χρήσεις, μαρτυρεί την έγνοια των ανθρώπων να μιμηθούν τη φύση στο πιο θαυμαστό της δημιούργημα, στην κίνηση. Η τιθάσευση του χρόνου και της κίνησης είναι το πρωταρχικό ζητούμενο του αυτοματισμού] (Dreyfus & Dreyfus, 1984). Τα αυτόματα του 18ου αιώνα, τα ηλεκτροκίνητα ρομπότ του επόμενου κι αργότερα οι «τεχνητοί εγκέφαλοι» των πρώτων βημάτων της Πληροφορικής βρίσκονται στο σημείο σύγκλισης της τεχνολογίας και της μυθολογίας.

[Ο έλεγχος του χρόνου, τα ηλεκτρονικά ρολόγια και οι γενικές αρχές της αυτορρύθμισης θα χρησιμεύουν για την κατασκευή των πρώτων υπολογιστών που είναι μηχανές που λειτουργούν με κανονικούς παλμούς του «εσωτερικού ρολογιού» τους. Έτσι, οι υπολογιστές είναι κι αυτοί προγραμματιζόμενες μηχανές, που λειτουργούν αυτόματα. Οι

«διάτρητες κάρτες» θα είναι για πολύ καιρό το αναγκαίο βοήθημα για τον προγραμματισμό των υπολογιστών. Στα Γαλλικά, ο όρος Πληροφορική (informatique) δημιουργήθηκε εύστοχα από τον Φίλιπ Ντρέυφους, την άνοιξη του 1962, από τη συνένωση των όρων πληροφορία (information) και αυτοματική (automatique) – τον επιστημονικό κλάδο που μελετά τις εφαρμογές του αυτοματισμού. Μέσα στο πνεύμα του δημιουργού του, το νέο αυτό πεδίο ήταν ένα παρακλάδι της αυτοματικής: η αυτοματική των πληροφοριών. Η παράδοση του αυτοματισμού θα βοηθήσει σημαντικότατα στην πράξη την πρόοδο του νέου επιστημονικού κλάδου. Σε ανταπόδοση, η πληροφορική θα κάνει περισσότερα για την ανάπτυξη της αυτοματοποίησης απ' οποιαδήποτε άλλη προγενέστερη επιστήμη. Δεν θα λείπει πλέον από το προγραμματιζόμενό μας αυτόματο παρά ένας στόχος: ο υπολογισμός, η επεξεργασία της πληροφορίας. Είναι ακριβώς στο σημείο αυτό που η παράδοση του αυτοματισμού θα συναντήσει την υπερχιλιετή παράδοση του υπολογισμού και την, πιο πρόσφατη, έννοια της πληροφορικής. Λίγο λίγο αναδείχθηκε η ιδέα ότι η πληροφορία θα μπορούσε να κωδικοποιηθεί και ότι μια τέτοια κωδικοποίηση θα μπορούσε να γίνει ανεξάρτητα από τη σημασία των μηνυμάτων: ήταν πλέον ανοικτός ο δρόμος για μια γλώσσα συλλογισμών, που θα χρησιμοποιούσε όλα τα μέσα της λογικής και των σύγχρονων αυτομάτων. Επιτέλους, από το βάθος του χρόνου, ο αριθμητικός υπολογισμός θα γινόταν πλέον μια δουλειά για μηχανές και όχι για εργαλεία, όπως ήταν ακόμα ο πρωτόγονος άβακας. Αυτές οι νέες μηχανές θα επωφελούνταν από κάθε πρόοδο που θα σημειωνόταν στο έλεγχο του χρόνου και της κίνησης και στην εξερεύνηση των μηχανισμών του συλλογισμού] (Breton, 1991).

Η θεωρία της πληροφορίας

[Η πληροφορία, με τη σύγχρονη έννοια του όρου, θα αποκτήσει μεγάλη σημασία, καθώς η αυτόματη επεξεργασία της θα είναι το βασικό αντικείμενο, γύρω από το οποίο θα οργανωθεί, από τη μεταπολεμική περίοδο, ο κόσμος των υπολογιστών και των πληροφοριακών επιστημών. Η μοντέρνα «πληροφορία» προσδιορίστηκε μεταξύ του 1927, χρονολογία κατά την οποία η λέξη χρησιμοποιήθηκε με το σύγχρονο νόημά της από τον Χάρτλεϋ, και του 1948, όταν ο Αμερικανός μαθηματικός Κλοντ Σάννον δημοσιεύει την περίφημη «μαθηματική θεωρία της πληροφορίας». Η έννοια της πληροφορίας βρίσκεται στο σημείο επαφής παλιότερων τομέων, που έχουν ο καθένας τη δική του ιστορία, και των οποίων αρκετοί κλάδοι συνενώθηκαν για να σχηματίσουν ένα νέο πεδίο γνώσης. Τα ψήγματα της έννοιας αυτής μπορούν να αναζητηθούν προς τρεις τουλάχιστον κατευθύνσεις:

- Κατά πρώτο λόγο, στην ιδέα της διάκρισης μεταξύ μορφής και νοήματος: με αυτήν εκδηλώνεται μια αληθινή νοηματική ρήξη που καθιστά δυνατή την αντίληψη της έννοιας της πληροφορίας.
- Κατά δεύτερο λόγο, στις τεχνικές που αναπτύχθηκαν για να καλυφθούν οι ανάγκες της μετάδοσης μηνυμάτων. Η συσσώρευση των γνώσεων στον τομέα της επεξεργασίας των, ηλεκτρονικών, κυρίως, σημάτων θα οδηγήσει στον τηλέγραφο, στο τηλέφωνο και τελικά στις μοντέρνες τηλεπικοινωνίες, και
- Κατά τρίτο λόγο, στην παράδοση της έννοιας, αρχικά θεολογικής και αργότερα λογικής και μαθηματικής για τις συνθήκες αλήθειας των αποφάνσεων και την ακριβή φύση των συλλογισμών: ένα αποτέλεσμα αυτής της ερευνητικής σκέψης θα είναι και

η αποφασιστική διασαφήνιση της έννοιας του αλγόριθμου από τον Άγγλο μαθηματικό Άλαν Τούρινγκ στα 1936.] (Breton, 1991)

[Το κλασικό παράδειγμα του τηλεγραφήματος σκιαγραφεί τη διαφορά μεταξύ αυτών των εννοιών. Όταν κάποιος φέρνει ένα τηλεγράφημα στο ταχυδρομείο, το μήνυμά του διαβάζεται από τον αρμόδιο αλλά ο τελευταίος δεν ενδιαφέρεται για το νόημα αυτού που διαβάζει. Δεν παίρνει υπόψη του παρά τη μορφή του μηνύματος, δηλαδή τα σύμβολα που αυτό περιέχει. Αυτά τα σύμβολα θα μετατραπούν σε σήματα που θα μεταδοθούν τηλεγραφικά. Σύμβολα και σήματα μπορούν να αντιμετωπιστούν ανεξάρτητα από τη σημασία τους: συνιστούν τη μορφή του μηνύματος. Αυτή η πρώτη διάσπαση μεταξύ μορφής και νοήματος θα συνοδευτεί κατά τον 20ο αιώνα, από μια δεύτερη διάκριση που θα δείξει ότι η μορφή ενός μηνύματος μπορεί να αποσυνδεθεί σε σύμβολα και σε σήματα, τα οποία παράγονται από τον φυσικό φορέα του μηνύματος] (Breton, 1991).

[Η επικοινωνία από απόσταση φαίνεται να είναι, παρά τις αρκετά πρώιμες απόπειρες, μια δραστηριότητα που παρουσιάστηκε αργά στην ανθρώπινη ιστορία. Βέβαια, ο καθένας γνωρίζει τα σινιάλα καπνού των Ινδιάνων της Β. Αμερικής, ή τη χρήση του τύμπανου σε αρκετούς αφρικανικούς πολιτισμούς αλλά, εκτός από αυτές τις προσπάθειες, των οποίων η εμβέλεια είναι, μάλλον, περιορισμένη, πρέπει να περιμένουμε τον 18ο αιώνα για να οργανωθεί ένα συστηματικό δίκτυο επικοινωνίας από απόσταση. Ως τότε, ο αγγελιαφόρος που ταξίδευε περπατώντας, με άλογο ή καρότσα, ήταν αρκετός.

Ακόμα και η εγκατάσταση στα 1794 του εναέριου τηλέγραφου του Γάλλου μηχανικού Κλοντ Σαπ, δεν εκθρόνισε τους έφιππους αγγελιοφόρους, που ήταν βέβαια πιο αργοί, αλλά μπορούσαν να μεταφέρουν μαζί τους μια μεγάλη ποσότητα μηνυμάτων. Το

σύστημα του Σαπ αποτελείτο από κινητούς βραχίονες ανεβασμένους σε πύργους (116 από το Παρίσι ως την Τουλόν) και συνέφερε για σύντομα μηνύματα ή για διαταγές (ένα σήμα πήγαινε από το Παρίσι στην Τουλόν σε 20 λεπτά με καλό καιρό). Η τιθάσευση του ηλεκτρισμού και η χρήση του για μετάδοση ποικίλων σημάτων επρόκειτο να επιτρέψουν γρήγορα τη σχεδόν στιγμιαία επικοινωνία από απόσταση, καθώς και την υπερπόντια επικοινωνία] (Breton, 1991).

[Παράλληλα με την ανάπτυξη των μεταφορών, οι διπλωματικές ανάγκες για μυστικότητα είχαν προωθήσει μια όλο και συστηματικότερη έρευνα για την κρυπτογράφηση των μηνυμάτων. Ήταν, ακριβώς, κατά το σχεδιασμό ενός από αυτούς τους μυστικούς κώδικες που εφευρέθηκε ο δυαδικός συμβολισμός, ο οποίος θα χρησιμοποιηθεί από τον Λάιμπνιτζ για τους αριθμούς. Οι έρευνες για τα φυσικά μέσα μεταφοράς των μηνυμάτων θα οδηγήσουν στην αποσαφήνιση της έννοιας του σήματος, ενώ οι έρευνες για την κωδικοποίηση των μηνυμάτων θα καταλήξουν στην έννοια του συμβόλου. Η σύνδεση των δύο εννοιών θα γίνει με τη θεωρία της πληροφορίας. Η τελευταία ήταν προορισμένη να εμφανιστεί μαζί με τα ηλεκτρικά σήματα, καθώς τα τελευταία αποτελούσαν τον πρώτο φυσικό φορέα που τα χαρακτηριστικά του μπορούσαν να μετρηθούν με ακρίβεια. Κάτι που δεν μπορούσαν να το εξασφαλίσουν ούτε τα σήματα καπνού, ούτε τα χτυπήματα των τύμπανων, ούτε ακόμα και τα τηλεγραφικά μηνύματα του Σαπ.

Ένα από τα σημαντικότερα ζητούμενα της θεωρίας της πληροφορίας είναι να κωδικοποιηθούν αποτελεσματικά τα μηνύματα που μεταδίδονται μαζί με θόρυβο και παράσιτα, προκειμένου να μεταβιβαστούν όσο το δυνατόν γρηγορότερα και να ανασυσταθούν σωστά μετά την άφιξή τους. Ένα παράδειγμα κωδικοποίησης

σκιαγραφείται παρακάτω και επιτρέπει στα σήματα να αντιμετωπίσουν αποτελεσματικά τον θόρυβο και να έχουν την καλύτερη δυνατή απόδοση] (Dreyfus & Dreyfus, 1984).

Παράδειγμα κωδικοποίησης

[Τι είδους κώδικα μπορούμε να χρησιμοποιήσουμε για να πραγματοποιήσουμε μια μεταβίβαση χωρίς σφάλματα μέσα από ένα κανάλι με παράσιτα; Έστω ότι επιδιώκουμε να μεταβιβάσουμε τα ψηφία:

1101 0011 0101 1000

που μπορούν εύκολα να αντιπροσωπεύουν οποιοδήποτε σύμβολο, αρκεί αυτό να έχει μεταφραστεί από πριν σε κάποιο δυαδικό κώδικα. Αλλά, αφού το κανάλι μεταβίβασης έχει παράσιτα, μας έχει διαφύγει ένα σφάλμα, το οποίο μετατρέπει ένα από τα ψηφία του μηνύματος σε μη αναγνώσιμο σήμα. Το πρόβλημα που έχουμε να αντιμετωπίσουμε είναι το εξής:

Πως μεταφέρουμε αυτό το μήνυμα – μαζί με το σφάλμα – έτσι ώστε να ανακατασκευάζεται πλήρως μετά την άφιξή του;

Η λύση του προβλήματος αυτού άπτεται της χρήσης των ψηφίων ελέγχου που παίζουν έναν μεγάλο ρόλο στην επεξεργασία των πληροφοριών. Τα βήματα που ακολουθούμε για τη λύση είναι τα ακόλουθα:

1. Αρχικά πινακοποιούμε τα ψηφία σε γραμμές και στήλες
2. Προσάπτουμε, μετά, ένα ψηφίο ελέγχου σε κάθε γραμμή και στήλη έτσι ώστε το άθροισμα των ψηφίων σε κάθε γραμμή και στήλη να είναι άρτιο

3. Ο πομπός και ο δέκτης κάνουν τη σύμβαση ότι τα 4 πρώτα ψηφία είναι τα ψηφία ελέγχου των στηλών και τα 4 επόμενα ψηφία είναι τα ψηφία ελέγχου των γραμμών. Τα 16 ψηφία που έπονται είναι αυτά του καθαρού μηνύματος

Η παραπάνω διαδικασία μπορεί να εφαρμοστεί εξίσου και στα ψηφία ελέγχου που μπορούν και τα ίδια να καταστραφούν από σφάλματα κατά τη μετάδοση. Το εκπεμπόμενο μήνυμα θα είναι, λοιπόν, το:

0011 1001 1101 0011 0101 1000

Το οποίο γράφεται με μορφή πίνακα ως:

Ψηφια ελεγχου στηλων

Ψηφια ελεγχου γραμμων	0	0	1	1
1	1	1	0	1
0	0	0	1	1
0	0	1	0	1
1	1	0	0	0

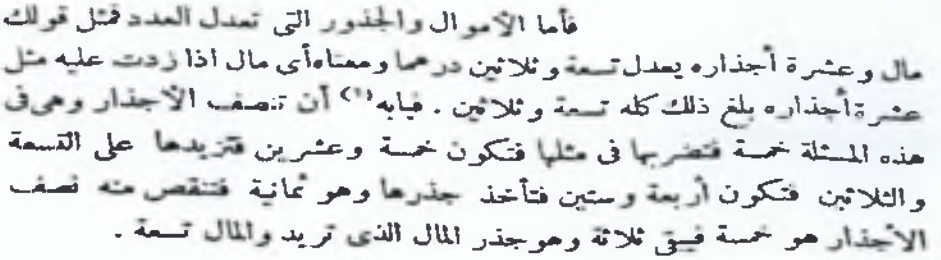
Έστω, ότι, για παράδειγμα, κατά τη μεταβίβαση του μηνύματος χάθηκε το ψηφίο στη θέση (3,2) του 4x4 πίνακα του καθαρού κειμένου.

	0	0	1	1
1	1	1	0	1
0	0	0	1	1
0	0		0	1
1	1	0	0	0

Από τις πληροφορίες που παίρνουμε από τα ψηφία ελέγχου στηλών και γραμμών, μπορούμε να συμπεράνουμε με ασφάλεια ότι το ψηφίο που λείπει είναι, αναγκαστικά, το 1] (Breton, 1991).

Αυτή η διαδικασία επεξεργασίας των πληροφοριών και η λύση ενός συγκεκριμένου προβλήματος, αποτελεί ένα σύνολο κανόνων που επιτρέπουν την αντιμετώπιση και επιτυχή μετάβαση της πλήρους πληροφορίας. Αυτό το σύνολο κανόνων – το οποίο οι Αγγλοσάξωνες ονομάζουν «αποτελεσματική διαδικασία» - ονομάζεται αλγόριθμος και θα μας απασχολήσει στο επόμενο μέρος της εργασίας αυτής.

Αλγόριθμοι


فأما الأموال والجذور التى تعدل العدد فمثل قولك
مال وعشرة أجذاره يعدل تسعة وثلاثين درهما ومعناه أى مال اذا زدت عليه مثل
عشرة أجذاره بلغ ذلك كله تسعة وثلاثين . فبابه[1] أن تنصف الأجذار وهى فى
هذه المسئلة خمسة فتضربها فى مثلها فتكون خمسة وعشرين فتزيدها على التسعة
والثلاثين فتكون أربعة وستين فتأخذ جذرها وهو ثمانية فتنقص منه نصف
الأجذار هو خمسة فيبقى ثلاثة وهو جذر المال الذى تريد والمال تسعة .

Οι αλγόριθμοι υπάρχουν από την αρχή της ιστορίας και πολύ πριν επινοηθεί μια συγκεκριμένη λέξη για να συνοψίσει και περιγράψει την λειτουργία τους. Οι αλγόριθμοι είναι απλά μια σειρά από βήματα, μια σειρά από βήμα-προς-βήμα οδηγίες, οι οποίες αν ακολουθηθούν μηχανικά, θα οδηγήσουν στην επίλυση ενός συγκεκριμένου προβλήματος. Από το στιγμή που ανακαλύπτεται η συγκεκριμένη ρουτίνα, ή 'συνταγή', αυτή μπορεί να μεταφερθεί ως πληροφορία και σε άλλους ώστε κι αυτοί, να μπορούν να λύσουν το ίδιο πρόβλημα.

Παράδειγμα αλγορίθμου

Για παράδειγμα, το αραβικό κείμενο του Μοχάμεντ Ιμπν Μουζά Αμπού Τζιφάρ Αλ-Κβαριζμί, του 9ου αιώνα, που φαίνεται στην εικόνα πιο πάνω, στο οποίο για πρώτη φορά εμφανίστηκε ο όρος «Αλγόριθμος» από την αραβική λέξη Al-Jabr (= αποκατάσταση) – η λέξη δεν είναι τυχαία μια που ένας από τους σκοπούς της άλγεβρας είναι και η

αποκατάσταση της ισότητας μέσα σε μια εξίσωση, περιγράφει την διαδικασία επίλυσης μιας δευτεροβάθμιας εξίσωσης της μορφής:

$$x^2+10x=39$$

Τα βήματα που προτείνει είναι τα ακόλουθα:

1. Παίρνουμε το μισό του αριθμού μπροστά από τον πρωτοβάθμιο όρο (= 10/2 = 5)
2. Πολλαπλασιάζουμε τον αριθμό που προκύπτει με τον εαυτό του (= 5*5 = 25)
3. Προσθέτουμε το αποτέλεσμα που προκύπτει με τον σταθερό όρο στα δεξιά της εξίσωσης (=25+39=64)
4. Βρίσκουμε τον αριθμό που αν τον πολλαπλασιάσουμε με το εαυτό του μας δίνει το παραπάνω αποτέλεσμα – βήμα 3 (= 8)
5. Αφαιρούμε από το παραπάνω αποτέλεσμα τον αριθμό στο βήμα 1 (= 8 -5 = 3)
6. Το παραπάνω αποτέλεσμα είναι η λύση της εξίσωσης (πράγματι, $3^2+10\times3=39$!)

Οι αλγόριθμοι δεν είναι μόνο διαδικασίες που χρησιμοποιούνται στα μαθηματικά. Οι Βαβυλώνιοι, π.χ., τους χρησιμοποίησαν στις δικαστικές – νομικές τους διαμάχες, οι Λατίνοι εκπαιδευτικοί τους επικαλέστηκαν κατά τη διδασκαλία της λατινικής γραμματικής ενώ έχουν χρησιμοποιηθεί από όλους τους πολιτισμούς για να προβλέψουν το μέλλον, προκειμένου να δικαιολογήσουν τη λήψη αποφάσεων αναφορικά με ιατρικές θεραπείες, ή για την προετοιμασία του φαγητού. Όλοι έχουμε χρησιμοποιήσει αλγόριθμους, είτε προκειμένου να μαγειρέψουμε το φαγητό της ημέρας, είτε προκειμένου να πλέξουμε κάποιο ρούχο είτε προκειμένου να σεττάρουμε κάποια ηλεκτρική – ηλεκτρονική συσκευή.

Όταν μιλάμε για αλγορίθμους, ουσιαστικά αναφερόμαστε σε 'συνταγές', κανόνες, τεχνικές, διαδικασίες, μεθόδους κλπ., χρησιμοποιώντας την ίδια λέξη αναλόγως κάθε φορά. Οι Κινέζοι, για παράδειγμα, χρησιμοποιούσαν τη λέξη shu (που σημαίνει κανόνας, διαδικασία ή στρατηγική) τόσο στα μαθηματικά όσο και στις πολεμικές τέχνες. Οι Γιαπωνέζοι χρησιμοποιούν την λέξη ju-jitsu (που σημαίνει κανόνες για την διαδικασία επίλυσης) – η λέξη ju από την κινέζικη shu. Η λέξη αλγόριθμος προέρχεται από το κείμενο του Αλ-Κβαριζμί, ενός μαθηματικού του πρώτου μισού του 9ου αιώνα και το βιβλίο του al-Mukhtasar fi Hisab Al-Jabr wa l-Muqabala εισάγει για πρώτη φορά την λέξη αλγόριθμος από την λέξη Al-Jabr, όπως είδαμε ήδη.

Τον 12ο αιώνα, η εργασία αυτή μαζί με άλλες, μεταφράζονται στα λατινικά, και το δεκαδικό σύστημα αρχίζει σταδιακά να εξαπλώνεται στην μεσαιωνική Ευρώπη. Αφού η αρχική πηγή ήταν στην αραβική, το δεκαδικό σύστημα αρίθμησης ονομάστηκε αραβικό, αν και τα σύμβολα του συστήματος διαμορφώθηκαν από την Ινδική πρακτική. Στα λατινικά κείμενα της εποχής, συχνά συναντούμε την λέξη αλγόρισμος, ή αλγόρισμους ή αλγόριθμος προκειμένου να περιγραφούν ρουτίνες αριθμητικών επιλύσεων.

[Κατά τη διάρκεια των αιώνων, το νόημα της λέξης **αλγόριθμος** επεκτάθηκε και έφτασε να σημαίνει – σύμφωνα με τον ντ'Αλεμπέρ:

Ο αραβικός όρος, ο οποίος χρησιμοποιείται από διάφορους συγγραφείς, και ιδίως από τους Ισπανούς, σημαίνει την πρακτική της άλγεβρας. Επίσης, συχνά χρησιμοποιείται για να επισημάνει πράξεις αριθμητικής με ψηφία. Η ίδια λέξη χρησιμοποιείται για να σημάνει, γενικά, οποιαδήποτε μέθοδο και συμβολισμό οποιασδήποτε μορφής υπολογισμού. Με αυτήν την έννοια, επικαλούμαστε τους αλγορίθμους του ολοκληρωτικού λογισμού, τους

αλγορίθμους του διαφορικού λογισμού, τους αλγορίθμους της τριγωνομετρίας κλπ.] (Bretton, 1991)

Στο τέλος, η λέξη αλγόριθμος αποκτά την έννοια οποιασδήποτε διαδικασίας συστηματικού υπολογισμού, μιας διαδικασίας που μπορεί να ακολουθηθεί αυτόματα. Σήμερα, κυρίως μετά την εξάπλωση και ανάπτυξη της πληροφορικής, ο αλγόριθμος αποκτά και την έννοια του πεπερασμένου σύνολο κανόνων, διαφοροποιώντας τον από περισσότερο ασαφείς έννοιες όπως η τεχνική, η διαδικασία ή η μέθοδος. Έτσι, ένας **αλγόριθμος** μπορεί να οριστεί ως (εγκυκλοπαίδεια Britannica, 15η έκδοση) :

το πλήρες σύνολο κανόνων – με τη μορφή πεπερασμένων βημάτων - που επιτρέπουν την επίλυση ενός δεδομένου προβλήματος ή την απάντηση σε κάποιο ερώτημα

Η ανάγκη για την εκτέλεση πεπερασμένων βημάτων προήλθε μετά την διατύπωση από τον Hilbert του 10ου προβλήματός του το 1900 στο 2ο διεθνές συνέδριο Μαθηματικών στο Παρίσι. Σύμφωνα με αυτό, αναζητείται, σε πεπερασμένα βήματα, το εάν μια Διοφαντική εξίσωση (Diophantine equation) ρητών πολυωνύμων της μορφής:

$$p(x_1, x_2, \ldots, x_n) = 0$$

$x_1, x_2, \ldots, x_n$με ρητούς συντελεστές έχει λύση και αυτή η λύση να ανήκει στους ρητούς αριθμούς. Η απάντηση στο ερώτημα αυτό είναι αρνητική και αποτελεί το αποτέλεσμα εργασιών των Davis, Matiyasevich, Putnam και Robinson.

Συνεπώς, αλγόριθμος είναι μια πεπερασμένη σειρά από ενέργειες αυστηρά καθορισμένες και εκτελέσιμες σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος. Ένας αλγόριθμος είναι μια υπολογιστική διαδικασία που παίρνει μία ή περισσότερες τιμές σαν είσοδο και παράγει μία ή περισσότερες τιμές σαν έξοδο. Ένας αλγόριθμος είναι, δηλαδή, μία ακολουθία υπολογιστικών βημάτων που μετασχηματίζει την είσοδο σε έξοδο. Ένας αλγόριθμος πρέπει να περιέχει μια ακριβή και σαφή περιγραφή η οποία καθιστά καθαρό τι ακριβώς είναι αυτό που πρέπει να γίνει. Σε μια συνταγή, π.χ., στη οποία αναφέρεται 'ψήστε μέχρι να ψηθεί', αυτό είναι μια ασαφής περιγραφή του χρόνου ψησίματος μια που δεν περιγράφει τι σημαίνει 'ψημένο'. Σε έναν υπολογιστικό κώδικα, από την άλλη μεριά, η έκφραση 'επιλέξτε έναν μεγάλο αριθμό' επίσης, είναι ασαφής μια που ο 'μεγάλος αριθμός' δεν αποσαφηνίζεται τι ακριβώς είναι.

Η εκτέλεση ενός αλγορίθμου πρέπει να είναι τυφλή εφαρμογή των κανόνων ή των υπολογιστικών βημάτων από τα οποία αυτός αποτελείται. Κανένας αλγόριθμος δηλαδή δεν 'σκέφτεται' αλλά ακολουθεί βήμα προς βήμα τους κανόνες του. Κάθε αλγόριθμος απαραίτητα ικανοποιεί τα παρακάτω κριτήρια:

- Είσοδος (input): Καμία, μία ή και περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται όταν ο αλγόριθμος δημιουργεί κι επεξεργάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια των συναρτήσεων παραγωγής τυχαίων αριθμών, ή με την βοήθεια άλλων απλών εντολών. Για παράδειγμα, ο αλγόριθμος μπορεί να παίρνει στην είσοδο δύο οποιουσδήποτε θετικούς αριθμούς ή μια λέξη ή μια λίστα με μηδέν ή άλλα νούμερα.

- Έξοδος (output): Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν άλλο αλγόριθμο. Θα μπορούσε να είναι ο μεγαλύτερος από τους δύο αριθμούς που δόθηκαν στην είσοδο, ή η λέξη με όλα τα γράμματα κεφαλαία, ή η λίστα σε αύξουσα μορφή.
- Καθοριστικότητα (definiteness): Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της. Λόγου χάριν, μία εντολή διαίρεσης πρέπει να θεωρεί και την περίπτωση, όπου ο διαιρέτης λαμβάνει τη μηδενική τιμή. Την καθοριστικότητα, κάποιοι συγγραφείς, την αναφέρουν ως 'προϋπόθεση', με την έννοια ότι κάθε αλγόριθμος θέτει κάποιους περιορισμούς στην είσοδό του (input) (π.χ. επιλογή αριθμών διάφορων του μηδενός για τον διαιρέτη). Εάν αυτοί οι περιορισμοί δεν τηρούνται τότε ο αλγόριθμος μπορεί να μην είναι εκτελέσιμος ή να δίδει την σωστή απάντηση.
- Περατότητα (finiteness): Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα συγκεκριμένο αριθμό βημάτων δεν αποτελεί αλγόριθμο, αλλά λέγεται απλά υπολογιστική διαδικασία (computational procedure). Δεν είναι και ιδιαίτερα χρήσιμο να τρέχει ένα πρόγραμμα χωρίς τέλος, μια που αυτό μπορεί να μην μας δώσει ποτέ την απάντηση.
- Αποτελεσματικότητα (effectiveness): Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή. Αυτό σημαίνει ότι μια εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη.

Επιπλέον, οι αλγόριθμοι θα πρέπει να εγγυούνται την παροχή της σωστής απάντησης πάντα. Είναι σπάνια χρήσιμο εάν ένας αλγόριθμος δίδει το σωστό αποτέλεσμα στο 99% των περιπτώσεων ενώ στο 1% το λανθασμένο.

Οι βασικές ερωτήσεις που κανείς καλείται να απαντήσει πριν χρησιμοποιήσει κάποιον αλγόριθμο είναι οι ακόλουθες:

- Υπάρχει κάποιος αλγόριθμος για να λύσει το συγκεκριμένο πρόβλημα;
- Εάν υπάρχει, τότε είμαστε σίγουροι ότι ο συγκεκριμένος αλγόριθμος είναι εφαρμόσιμος για κάθε πιθανή είσοδο;
- Πόση ώρα χρειάζεται για να εκτελεστεί ο συγκεκριμένος αλγόριθμος; Πόση υπολογιστική μνήμη χρειάζεται;
- Εάν υπάρχει αλγόριθμος που λύνει σωστά το πρόβλημα, τότε είμαστε σίγουροι ότι αυτός είναι ο μόνος τρόπος επίλυσης ή, μήπως, υπάρχουν κι άλλοι περισσότερο αποτελεσματικοί και γρήγοροι αλγόριθμοι;

Η μηχανή Τούρινγκ

Ο αλγόριθμος είναι μια έννοια γενική στην οποία, ο Άγγλος μαθηματικός Άλαν Τούρινγκ, θα δώσει για πρώτη φορά μια ολοκληρωμένη μορφή. Ο Τούρινγκ είχε περιγράψει μια υποθετική μηχανή που αποτελείτο, απλώς, από μια χάρτινη ταινία χωρίς τέλος και μια κεφαλή που μπορούσε να διαβάσει, να γράψει ή να σβήσει ένα σύμβολο, να μετακινήσει μια ταινία προς τα δεξιά ή προς τα αριστερά, να σημειώσει ένα από τα τετράφωνα της ταινίας και να σταματήσει. Αυτή η μηχανή θα ήταν ικανή να επιλύσει όλα τα προβλήματα που μπορούσαν να διατυπωθούν ως αλγόριθμοι. Σκοπός του Τούρινγκ, σε

κάθε περίπτωση, δεν ήταν να ανακαλύψει απλά μια μηχανή ως τέτοια, αλλά να ερευνήσει θεωρητικά τα θεμέλια και τα όρια της λογικής, ακολουθώντας έτσι μια παράδοση το λιγότερο δύο και πλέον χιλιετιών.

Οι Βρετανοί φιλόσοφοι και μαθηματικοί Άλφρεντ Νορθ Γουάιτχεντ και Μπέρτραντ Ράσσελ είχαν προσπαθήσει να συγκροτήσουν ένα ευρυτενές, παγκόσμιο λογικό σύστημα, και, από τότε αρκετοί επιστήμονες λογικής προσπαθούσαν να δείξουν ότι το σύστημα αυτό δεν απαντούσε σε όλες τις καταστάσεις. Η μηχανή του Τούριγκ, αν και έδειξε κάποια όρια λογικής – απέδειξε συγκεκριμένα τη μη αποφαστιστικότητα της θεωρίας των τυποποιημένων αριθμών – κατέδειξε την ισχύ της αλγοριθμικής προσέγγισης. Με έναν μηχανισμό τόσο απλό όσο μια χάρτινη ταινία, ένας ελεγκτής και κάποια σύμβολα, όλα τα προβλήματα που θα μπορούσαμε να περιγράψουμε εξαντλητικά μπορούσαν να επιλυθούν από μια μηχανή – μόνο στη θεωρία βέβαια, μια που η μηχανή του Τούρινγκ υποτίθεται ότι είχε άπειρη μνήμη. Ένα παράδειγμα της μηχανής Τούριγκ φαίνεται παρακάτω.

Παράδειγμα Μηχανής Τούριγκ 1

Για να θέσετε σε λειτουργία τη μηχανή που έχετε εδώ αρκεί να προμηθευτείτε τέσσερα νομίσματα ή χαλίκια στο μέγεθος των τετραγώνων της ταινίας καθώς κι έναν συνδετήρα που θα παίζει το ρόλο της κεφαλής. Το πρόβλημα που η μηχανή επιτρέπει να λυθεί είναι ο διπλασιασμός μιας αρχικής ποσότητας (εδώ πρόκειται για διπλασιασμό αλλά μπορείτε να δοκιμάσετε με τριπλασιασμό ή ακόμα και χιλιαπλασιασμό). Ο αλγόριθμος επίλυσης πρέπει να ακολουθηθεί σχολαστικά για να φτάσει κανείς στο αποτέλεσμα. Η μηχανική υπακοή στις οδηγίες εγγυάται τη σωστή τους εκτέλεση.

Κατάστα -ση	***Αν η θέση κάτω από τον συνδετήρα είναι ελεύθερη***	***Αν η θέση κάτω από τον συνδετήρα είναι κατειλημμένη***

1	Σταματήστε	Αδειάστε τη θέση, μετακινήστε το συνδετήρα 1 τετράγωνο αριστερά και περάστε στην κατάσταση 2
2	Βάλτε ένα κέρμα στη θέση κάτω από τον συνδετήρα, μετακινήστε το συνδετήρα 1 τετράγωνο αριστερά και περάστε στην κατάσταση 3	Μετακινήστε το συνδετήρα 1 τετράγωνο αριστερά και μείνετε στην κατάσταση 2
3	Βάλτε ένα κέρμα στη θέση κάτω από τον συνδετήρα, μετακινήστε το συνδετήρα 1 τετράγωνο δεξιά και περάστε στην κατάσταση 4	Μετακινήστε το συνδετήρα 1 τετράγωνο αριστερά και μείνετε στην κατάσταση 3
4	Μετακινήστε το συνδετήρα 1 τετράγωνο δεξιά και περάστε στην κατάσταση 5	Μετακινήστε τον συνδετήρα 1 τετράγωνο δεξιά και μείνετε στην κατάσταση 4
5	σταματήστε	Αδειάστε τη θέση, μετακινήστε το συνδετήρα 1 τετράφωνο αριστερά και περάστε στην κατάσταση 2

Σύμφωνα με τα παραπάνω βήματα παίρνουμε την παρακάτω αλληλουχία ταινιών:

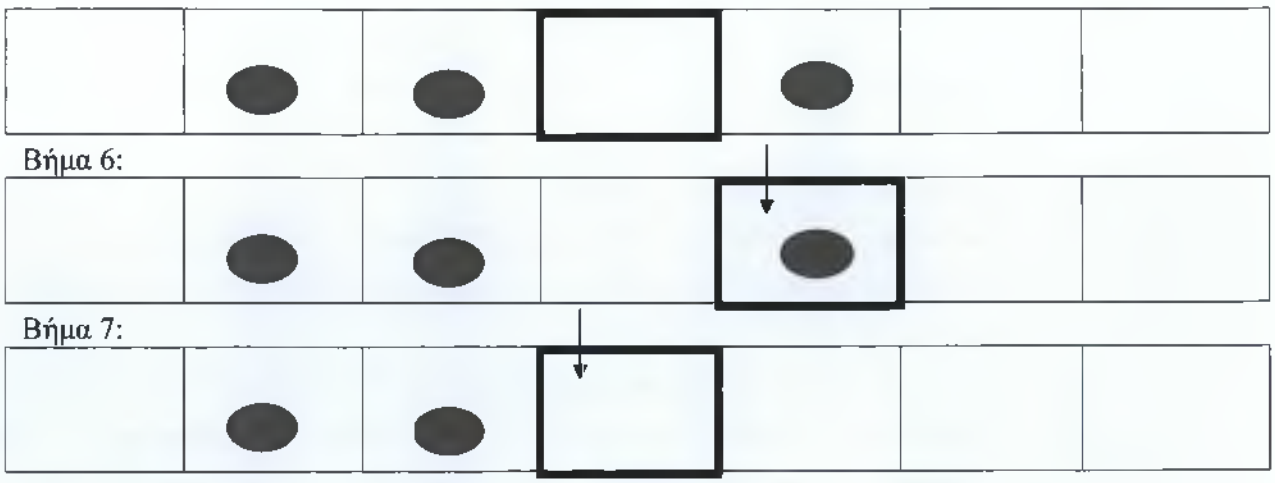
Βήμα 1: Αρχική κατάσταση (το βέλος αντιπροσωπεύει τον συνδετήρα)

Βήμα 2:

Βήμα 3:

Βήμα 4:

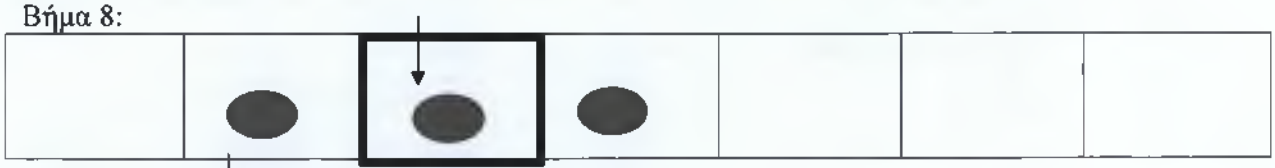
Βήμα 5:



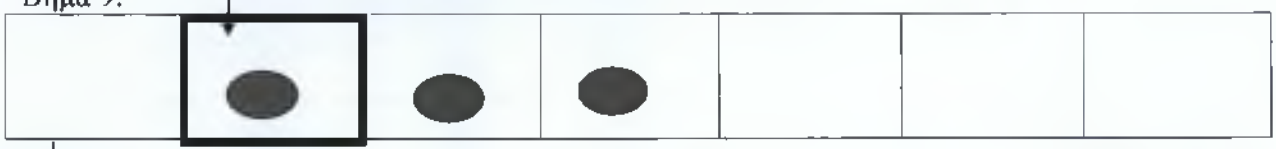
Βήμα 6:

Βήμα 7:

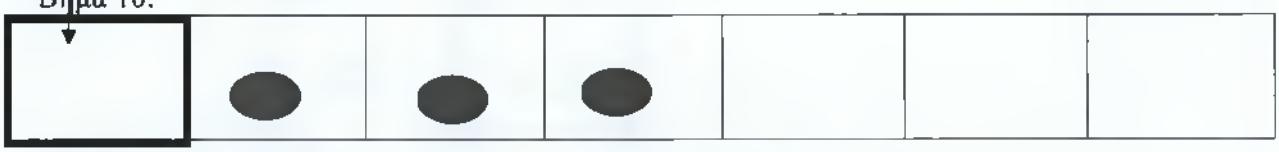
Βήμα 8:



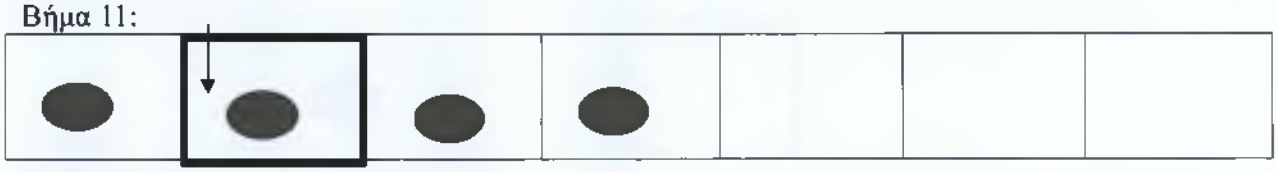
Βήμα 9:



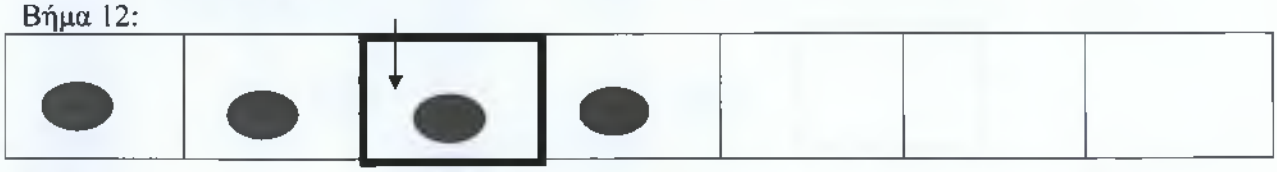
Βήμα 10:



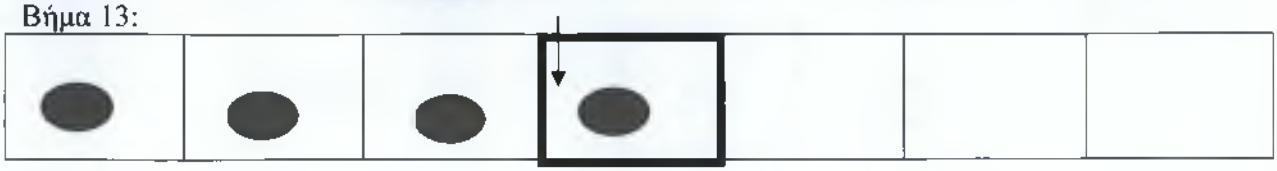
Βήμα 11:



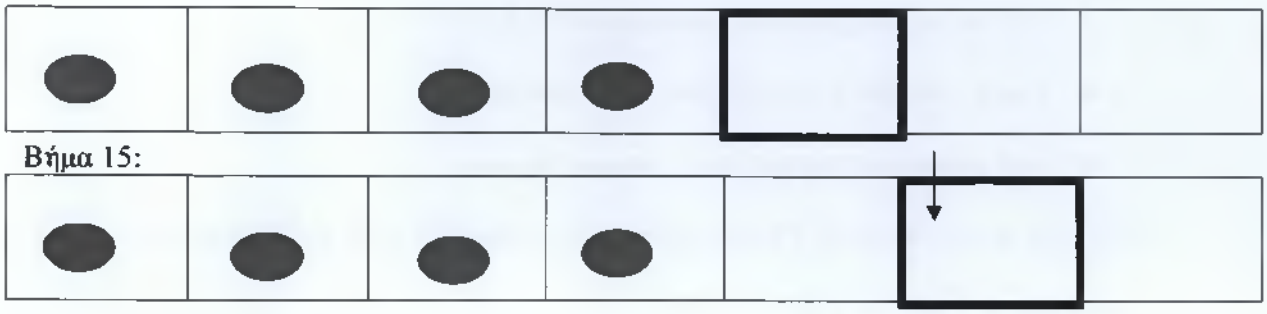
Βήμα 12:



Βήμα 13:



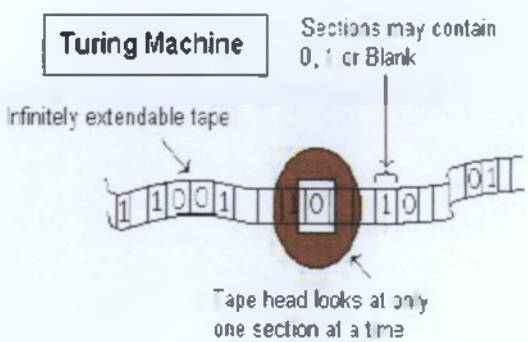
Βήμα 14:



Βήμα 15:

Η χαρτοταινία της μηχανής του Τούριγκ ήταν μια μνήμη γενικής χρήσης που αποθήκευε ταυτόχρονα δεδομένα κι εντολές. Η ίδια η μηχανή του Τούριγκ είναι το θεωρητικό ισοδύναμο των σύγχρονων υπολογιστών. Ένα άλλο χαρακτηριστικό της είναι ότι ήταν μια «μηχανή διακεκριμένων καταστάσεων» ή με άλλα λόγια μια μηχανή που λειτουργούσε μεταπίπτοντας από τη μια κατάσταση στην άλλη. Ο όρος «διακεκριμένη κατάσταση» χρησιμοποιείται εδώ για να δηλώσει ότι αφού εκτελεστεί μια εντολή εκτελείται μια άλλη και ότι από τη στιγμή που τελειώνει η εκτέλεση της πρώτης εντολής ως τη στιγμή που αρχίζει η εκτέλεση της δεύτερης, δεν συμβαίνει τίποτα (το «διακεκριμένος» αντιπαρατίθεται στο «συνεχής»). Ο Τούριγκ είχε κατά νου να κατασκευάσει έναν «εγκέφαλο» - είχε μιλήσει γι' αυτό στη μητέρα του το 1944 – μιμούμενος όχι τόσο τη φιλοσοφία του ανθρώπινου μοντέλου όσο τη λογική συμπεριφορά του, και θεωρώντας τις «καταστάσεις σκέψης» αντίστοιχες με τις εντολές της μηχανής. Επιθυμούσε, έτσι, να σχεδιάσει μια καθολική μηχανή που θα προσέφερε τις υπηρεσίες της στην ψυχολογία για τη μελέτη του ανθρωπίνου εγκεφάλου.

Μια μηχανή Turing είναι μια θεωρητική απεικόνιση ενός υπολογιστικού συστήματος. Όπως είδαμε και πιο πάνω, συμπεριφορά της μηχανής Turing (TM) μπορεί να προσδιοριστεί τελείως:



- Από την κατάσταση στην οποία βρίσκεται η μηχανή
- Από τον αριθμό στο τετράγωνο που διαβάζει
- Από τον πίνακα των εντολών ή πίνακα βημάτων

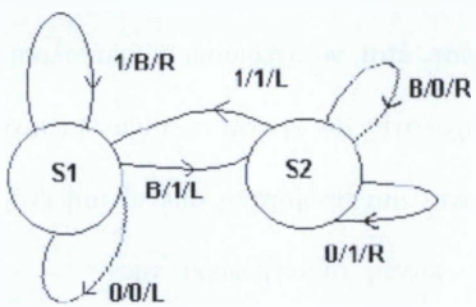
Σύμφωνα με την υπόθεση Church-Turing, μια συνάρτηση είναι υπολογίσιμη εάν μπορεί να υπολογιστεί από μια μηχανή Turing.

Παράδειγμα Μηχανής Τούρινγκ 2

Πέρα από το αναλυτικό αλλά ιδιαίτερα απλό παράδειγμα που είδαμε πιο πάνω, μια TM μπορεί να εκτελέσει, π.χ. και την παρακάτω πιο γενική διαδικασία:

	Διαβάζει	Γράφει	Κινείται	Επόμενη Κατάσταση
S1	0	0	L	S1
	B	1	L	S2
	1	B	R	S1
S2	0	1	R	S2
	B	0	R	S2
	1	1	L	S1

Αυτή η διαδικασία, όπως περιγράφεται στον πίνακα αριστερά, μπορεί να παρασταθεί και με διάγραμμα, όπως φαίνεται παρακάτω:



Τι είδους προβλήματα λύνονται με αλγορίθμους

Οι αλγόριθμοι χρησιμοποιούνται παντού αλλά ορισμένα πολύ συγκεκριμένα και σύγχρονα προβλήματα στα οποία οι αλγόριθμοι μπορούν να επιφέρουν το επιθυμητό αποτέλεσμα – λύση – είναι:

- Το πρόβλημα των ανθρώπινων γονιδίων: Το συγκεκριμένο πρόβλημα έχει κάνει άλματα προς τη λύση του – την αποκωδικοποίηση και ταυτοποίηση όλων των 100.000 γονιδίων του ανθρώπινου DNA, την αναγνώριση των δομών όλων των τριών δισεκατομμυρίων χημικών συμπλόκων του, την αποθήκευση της πληροφορίας αυτής σε βάσεις δεδομένων και την ανάπτυξη κατάλληλων κωδίκων επεξεργασίας της πληροφορίας αυτής. Κάθε ένα από αυτά τα βήματα απαιτεί πολύπλοκους αλγορίθμους. Οι υπολογιστικές μέθοδοι που αναπτύσσονται με βάση τους αλγορίθμους όχι μόνο κάνουν οικονομία στον χρόνο – τόσο τον ανθρώπινο όσο και της μηχανής – αλλά και στα χρήματα μια που δεν απαιτούν – προκειμένου να δώσουν μια απάντηση σε κάποιο ερώτημα – ακριβές εργαστηριακές τεχνικές.
- Με την ανάπτυξη του παγκόσμιου ιστού (Internet) τεράστια ποσότητα πληροφορίας είναι αποθηκευμένη σε διάφορες μηχανές ανά τον κόσμο. Με τη χρήση έξυπνων αλγορίθμων, κάποιες ιστοσελίδες μπορούν να ελέγχουν και να ρυθμίζουν αυτό το τεράστιο μέγεθος πληροφορίας. Για παράδειγμα, με τη χρήση αλγορίθμων μπορεί κανείς να βρει τις βέλτιστες οδούς μέσα από τις οποίες θα πρέπει να ταξιδέψουν τα δεδομένα από μια μηχανή αναζήτησης προκειμένου να εντοπίσουν γρήγορα και με αποτελεσματικότητα την μηχανή στην οποία η πληροφορία αυτή είναι αποθηκευμένη.
- Οι εμπορικές συναλλαγές που γίνονται μέσω Internet βασίζεται στην ιδιωτικότητα της προσωπικής πληροφορίας – όπως π.χ., οι αριθμοί της πιστωτικής κάρτας, τα

συνθηματικά πρόσβασης και οι τελευταίες τραπεζικές συναλλαγές. Οι βασικές τεχνολογίες e-commerce χρησιμοποιούν τεχνικές κρυπτογραφίας και ηλεκτρονικές υπογραφές που βασίζονται σε αλγορίθμους και τη θεωρία των αριθμών.

- Ο κατασκευαστικός κλάδος και οι εμπορικές εταιρίες, συχνά, χρειάζονται να γνωρίζουν που θα πρέπει να επενδύσουν ώστε να μεγιστοποιήσουν τα κέρδη τους. Μια εταιρία εξόρυξης πετρελαίου, π.χ., χρειάζεται να γνωρίζει που θα πρέπει να κάνει τις γεωτρήσεις της προκειμένου να μεγιστοποιήσει την παραγωγή της. Ένας πολιτικός υποψήφιος θα ήθελε να γνωρίζει που θα πρέπει να απευθυνθεί προκειμένου να μεγιστοποιήσει τις ψήφους του. Μια εταιρία παροχής υπηρεσιών Internet θα ήθελε να γνωρίζει που να προσθέσει πόρους ώστε να βελτιώσει τις υπηρεσίες που προσφέρει στους πελάτες της. Αυτά τα προβλήματα, αλλά και πολλά άλλα, μπορούν να αντιμετωπιστούν με τη χρήση αλγορίθμων γραμμικού προγραμματισμού.

Αν και υπάρχουν αλγόριθμοι λύσης για πάρα πολλά προβλήματα, εντούτοις υπάρχουν αρκετά προβλήματα τα οποία δεν αντιμετωπίζονται επιτυχώς με τη χρήση αλγορίθμων. Μια τέτοια ειδική κατηγορία προβλημάτων ονομάζονται προβλήματα πληρότητας – NP τα οποία έχουν και την ιδιάζουσα ιδιότητα ότι εφόσον υπάρξει ένας αποτελεσματικός αλγόριθμος για ένα από αυτά, τότε υπάρχει για όλα. Αλλά τέτοια προβλήματα θα μας απασχολήσουν αργότερα.

Θεωρία Πολυπλοκότητας

Η θεωρία πολυπλοκότητας είναι ένας κλάδος της πληροφορικής και των μαθηματικών ο οποίος εστιάζει στην κατηγοριοποίηση των υπολογιστικών προβλημάτων ανάλογα με την εγγενή δυσκολία τους και την συσχέτιση των κατηγοριών – κλάσεων που θα προκύψουν. Ένα υπολογιστικό πρόβλημα θεωρείται ότι είναι ένα πρόβλημα το οποίο, θεωρητικά, μπορεί να αντιμετωπιστεί από έναν υπολογιστή. Αυτό είναι ισοδύναμο με το να λέγαμε ότι ένα υπολογιστικό πρόβλημα μπορεί να αντιμετωπιστεί μηχανιστικά ακολουθώντας μηχανικά μια σειρά σαφώς καθορισμένων βημάτων ενός αλγορίθμου. Ένα πρόβλημα θεωρείται ενδογενώς δύσκολο εάν η λύση του απαιτεί σημαντικούς πόρους, ανεξάρτητα από τον αλγόριθμο που θα χρησιμοποιηθεί για τη λύση του.

Η θεωρία της πολυπλοκότητας θέτει σε φορμαλιστική βάση αυτό που φανερώνει η διαίσθηση: ότι ένα πρόβλημα είναι δύσκολο να επιλυθεί. Εισάγει μαθηματικά μοντέλα υπολογισμού για τη μελέτη τέτοιων προβλημάτων και ποσοτικοποιεί τους πόρους που απαιτούνται (μνήμη, χρόνος) προκειμένου να λυθούν τα προβλήματα αυτά. Άλλωστε, η απαιτούμενη μνήμη αλλά και ο απαιτούμενος χρόνος για τη λύση ενός προβλήματος

αποτελούν κριτήρια προκειμένου το τελευταίο να ταξινομηθεί ανάμεσα στα δύσκολα (ή όχι). Άλλα τέτοια κριτήρια είναι η ποσότητα της απαιτούμενης επικοινωνίας (χρησιμοποιείται στην επικοινωνιακή πολυπλοκότητας), ο αριθμός των πυλών σε ένα κύκλωμα (χρησιμοποιείται στην πολυπλοκότητα κυκλωμάτων) και ο αριθμός των επεξεργαστών (χρησιμοποιείται στον παράλληλο προγραμματισμό). Ένας από τους ρόλους της θεωρίας της πολυπλοκότητας είναι να εντοπίσει τα πρακτικά όρια των υπολογιστών: τι μπορούν και τι δεν μπορούν να κάνουν.

Το κριτήριο της αποδοτικότητας σύμφωνα με τον απαιτούμενο χρόνο και την ζητούμενη μνήμη (καθένα από τα οποία μετράται σε δευτερόλεπτα και MB αντίστοιχα) βασίζεται σε μια καθολική εκτίμηση των δύο τελευταίων κάτι, όμως, που δεν μπορεί να γίνει μια που τόσο ο χρόνος όσο και η απαιτούμενη μνήμη εξαρτώνται από την υπολογιστική ισχύ του εκάστοτε μηχανήματος αλλά και από το σύστημα δεδομένων που εισάγεται. Προκειμένου να συστηματικοποιηθούν οι μετρήσεις της αποδοτικότητας των αλγορίθμων αναπτύχθηκαν μαθηματικές συναρτήσεις που την συσχετίζουν με το μέγεθος της εισόδου. Το τελευταίο αναφέρεται στον αριθμό των στοιχείων των δεδομένων της εισόδου. Για παράδειγμα, όταν τοποθετούμε αλφαβητικά n λέξεις, το μέγεθος της εισόδου είναι n. Όταν εισάγουμε ένα γράφημα, τότε στην είσοδο δίνονται δύο μεταβλητές, ο αριθμός των σημείων τομής (vertex) και ο αριθμός των ακμών (edge). Η υπολογιστική πολυπλοκότητα αν και αναφέρεται τόσο στον χρόνο εκτέλεσης όσο και στην μνήμη που απαιτείται, εντούτοις, συνήθως, μελετάται ως προς τον χρόνο μια που οι απαιτήσεις σε μνήμη των περισσότερων αλγορίθμων κατά πολύ υπολείπονται των διαθέσιμων χαρακτηριστικών των σύγχρονων μηχανημάτων. Ο χρόνος ενός αλγόριθμου υπολογίζεται σύμφωνα με τον αριθμό των βασικών βημάτων που απαιτούνται σε

συνάρτηση με το μέγεθος της εισόδου. Επειδή, όμως, οι αλγόριθμοι περιέχουν υποθετικές κατασκευές (if-else), ο αριθμός των απαιτούμενων βημάτων διαφέρει από είσοδο σε είσοδο. Γι' αυτό, η χρήση της χειρότερης πολυπλοκότητας για έναν αλγόριθμο είναι, συχνά, ένα σημαντικό πλεονέκτημα.

Παράδειγμα Αλγορίθμου 1

[Για παράδειγμα, έστω ότι ψάχνουμε μια λέξη σε ένα λεξικό. Το μέγεθος της εισόδου N αναφέρεται στον συνολικό αριθμό λέξεων στο λεξικό, μια που κάθε λέξη του αποτελεί έναν πιθανό στόχο. Ο πρώτος αλγόριθμος που παραθέτουμε – ο οποίος εκτελεί μια γραμμική αναζήτηση – προσπαθεί να βρει τη λέξη στο λεξικό κάνοντας διαδοχικές συγκρίσεις με τις λέξεις που υπάρχουν εκεί:

Αλγόριθμος 1 Γραμμική Αναζήτηση

Γραμμική_Αναζήτηση(πίνακας λέξεων Dic[n], λέξη αναζήτησης t)

1. Για κάθε τιμή της μεταβλητής counter από 0 μέχρι n-1
    a. Εάν (Dic[counter] είναι το t) επιστροφή Dic[counter]
2. επιστροφή 'ΔΕΝ_ΒΡΕΘΗΚΕ'

Η γραμμική αναζήτηση, παραπάνω, έχει βαθμό πολυπλοκότητας n/2 μια που, κατά μέσο όρο, όλες οι λέξεις του λεξικού είναι πιθανές ως είσοδος] (Gries, 1989).

Παράδειγμα Αλγορίθμου 2

[Έστω ότι θέλαμε να αντιμετωπίσουμε το ίδιο πρόβλημα με παραπάνω (αναζήτηση λέξης σε λεξικό). Μια δυαδική αναζήτηση, αυτή τη φορά, εκμεταλλεύεται το γεγονός ότι οι λέξεις στο λεξικό είναι αλφαβητικά ταξινομημένες. Αυτό που κάνει ο παρακάτω αλγόριθμος είναι να συγκρίνει τη λέξη εισόδου με τη μεσαία λέξη του λεξικού και να την

τοποθετεί στο κατάλληλο 'μισό' – πάνω από τη μεσαία λέξη ή κάνω ανάλογα αν προπορεύεται αλφαβητικά ή όχι. Μετά εκτελεί ακριβώς τα ίδια βήματα, για το 'μισό' στο οποίο τοποθέτησε τη λέξη κ.ο.κ μέχρι να τη βρεί ή να δώσει το μήνυμα 'ΔΕΝ ΒΡΕΘΗΚΕ'.

Αλγόριθμος 2 Δυαδική Αναζήτηση

Δυαδική_Αναζήτηση(πίνακας λέξεων Dic[n], λέξη αναζήτησης t)

- Θέση κάτω = 0, πάνω = n-1
- όσο (κάτω <= πάνω) κάνε
    1. θέση μέσο = (κάτω+πάνω)/2
    2. εάν (Dic[μέσο] < t) κάτω = μέσο
    3. εάν όμως (Dic[μέσο] > t) πάνω = μέσο
    4. διαφορετικά επιστροφή Dic[μέσο]
- επιστροφή 'ΔΕΝ ΒΡΕΘΗΚΕ'

Η δυαδική αναζήτηση παραπάνω έχει σαφώς μικρότερη πολυπλοκότητα από την γραμμική, μια που σε κάθε βήμα πετάει τις μισές λέξεις του λεξικού] (Gries, 1989). Στην χειρότερη περίπτωση, η δυαδική αναζήτηση κάνει $\log_2 n$ βήματα (ή, όπως, συχνά συμβολίζεται στην πληροφορική $\lg n$) (ενώ, αντίστοιχα, η γραμμική εκτελεί n). Άρα η μέση πολυπλοκότητά της είναι ο μέσος όρος όλων των πιθανών βημάτων: $\log_2 n - 1 + \frac{3}{n}$

Όπως είδαμε παραπάνω, ο απώτερος στόχος της θεωρίας της πολυπλοκότητας είναι να:

- Βρει μια αυτοσυνεπή μέθοδο ποσοτικοποίησης της δυσκολίας ενός προβλήματος με απόλυτο τρόπο
- Παράσχει μια μέθοδο σύγκρισης μεταξύ των δυσκολιών των διαφορετικών

προβλημάτων

- Βρει έναν σαφή τρόπο προσδιορισμού της έννοιας του 'αποδοτικού αλγορίθμου' (π.χ. χρονική πολυπλοκότητα).

Κάθε πρόβλημα μπορεί να ειδωθεί ως μια άπειρη οικογένεια στιγμιότυπων (instance) των οποίων η συνάρτηση και περιορισμοί έχουν μια δεδομένη δομή. Ένα στιγμιότυπο προκύπτει εάν δώσουμε στις παραμέτρους του προβλήματος κάποιες τιμές. Προκειμένου να μετρήσουμε τη δυσκολία ενός προβλήματος μετράμε π.χ. το χρόνο που χρειάζεται για να λυθεί με το να βρίσκουμε τον αριθμό των ενδιάμεσων στοιχειωδών βημάτων. Κάθε στιγμιότυπο έχει κάποια δικιά του δυσκολία και το πρόβλημα είναι να βρούμε την δυσκολία αυτή και να αναλύσουμε το πρόβλημα αναλόγως. Ανάλογα με το χρόνο που χρειάζεται ένα πρόβλημα για να λυθεί διακρίνουμε τις παρακάτω κατηγορίες:

- Αλγόριθμος πολυωνυμικού χρόνου: Ένας αλγόριθμος του οποίου ο χρόνος εκτέλεσης φράζεται από μια πολυωνυμική συνάρτηση – **Παράδειγμα**: Το πρόβλημα της ελάχιστης διαδρομής με μη αρνητικά βάρη (Χρόνος εκτέλεσης: $O(n^2)$)
- Αλγόριθμος εκθετικού χρόνου: Ένας αλγόριθμος του οποίου ο χρόνος εκτέλεσης φράσσεται από μια εκθετική συνάρτηση – **Παράδειγμα**: Ένας αλγόριθμος που χρειάζεται να ελέγξει όλους τους αριθμούς με n ψηφία προκειμένου να βρει τη λύση (Χρόνος εκτέλεσης: $O(10^n)$)
- Αλγόριθμος ψευδο-πολυωνυμικού χρόνου: Ένας αλγόριθμος ο οποίος έχει πολυωνυμικό αριθμό δεδομένων όταν κωδικοποιούνται με μοναδιαίες πράξεις – **Παράδειγμα**: Το ακέραιο πρόβλημα του σακιδίου (Χρόνος εκτέλεσης: $O(nb)$, με b το βάρος του σακιδίου)

Ασυμπτωτικοί Συμβολισμοί

Στην θεωρία της πολυπλοκότητας ο ρυθμός αύξησης του χρόνου εκτέλεσης ενός αλγόριθμου σε συνάρτηση με το μέγεθος της εισόδου, είναι αυτό που μετράει για τον χαρακτηρισμό ενός αλγόριθμου ως αργού ή γρήγορου. Έστω, για παράδειγμα, δύο αλγόριθμοι Α και Β. Ο Α τρέχει αργά για μικρό αριθμό δεδομένων εισόδου ενώ ο Β έχει μεγαλύτερο ρυθμό αύξησης του χρόνου εκτέλεσης όσο αυξάνεται ο αριθμός των δεδομένων εισόδου. Σε πραγματικές συνθήκες, ο αλγόριθμος Β θα είναι, κατά μέσο όρο, πιο αργός από τον Α, μια που τα δεδομένα εισόδου στην πραγματικότητα είναι, συνήθως, πολλά. Αυτό σημαίνει ότι εάν η πολυπλοκότητα ενός αλγορίθμου εκφραστεί ως συνάρτηση, τότε μπορούμε, προκειμένου να εκτιμήσουμε την πολυπλοκότητα αυτή, να εστιάσουμε στον επικρατών όρο. Οι όροι χαμηλότερου βαθμού καθίστανται ασήμαντοι όσο μεγαλώνει το μέγεθος εισόδου n. Αυτή η εστίαση, ονομάζεται ασυμπτωτική συμπεριφορά.

Παράδειγμα Ασυμπτωτικής Συμπεριφοράς 1

Για παράδειγμα, η πολυπλοκότητα του αλγορίθμου δυαδικής αναζήτησης $\log_2 n - 1 + \frac{3}{n}$ θα μπορούσε απλά να γραφτεί ως $\log_2 n$ μια που, για μεγάλα n, οι όροι -1 και $\frac{3}{n}$ γίνονται ασήμαντοι.

Παράδειγμα Ασυμπτωτικής Συμπεριφοράς 2

Για τον αλγόριθμο της γραμμικής αναζήτησης η πολυπλοκότητα μπορεί να θεωρηθεί ίση με n αντί για n/2, μια που οι σταθερές που πολλαπλασιάζουν τον όρο που επικρατεί μπορούν να παραληφθούν μια που για μεγάλα n, δεν κάνουν ουσιαστική διαφορά. Ασυμπτωτικά, δηλαδή, δύο αλγόριθμοι με πολυπλοκότητες n και n/2 θεωρείται ότι έχουν

την ίδια πολυπλοκότητα.

Συμβολισμοί

Οι ασυμπτωτικοί συμβολισμοί είναι σύμβολα που χρησιμοποιούνται για να εκφράσουν την αποτελεσματικότητα των αλγορίθμων σύμφωνα με το ρυθμό αύξησης του χρόνου εκτέλεσης. Υπάρχουν τρεις συμβολισμοί που χρησιμοποιούνται συνήθως: ο συμβολισμός O, ο συμβολισμός Ω και ο συμβολισμός Θ.

Ο συμβολισμός Ο

Ο συμβολισμός O είναι ο πιο συχνός τρόπος έκφρασης της πολυπλοκότητας ενός αλγορίθμου. Συμβολίζει το ασυμπτωτικό άνω άκρο της συνάρτησης πολυπλοκότητας. Για μια δεδομένη συνάρτηση $g(n)$, η έκφραση $O(g(n))$ αντιπροσωπεύει το σύνολο των συναρτήσεων ώστε:

$$O(g(n)) = \{ \begin{matrix} f(n) : c \geq 0, n_0 \geq 0 : \\ 0 \leq f(n) \leq cg(n) \; \forall \; n \geq n_0 \end{matrix}$$

Μια μη αρνητική συνάρτηση $f(n)$ ανήκει στο σύνολο των συναρτήσεων $O(g(n))$ εάν υπάρχει μια θετική σταθερά c για την οποία να ισχύει $f(n) \leq cg(n)$ για επαρκώς μεγάλα n. Μπορούμε να γράψουμε $f(n) \in O(g(n))$ επειδή το $O(g(n))$ είναι σύνολο, αλλά, συμβατικά, αυτό μπορεί να γραφεί και ως $f(n) = O(g(n))$.

Παραδείγματα $O(g(n))$ συναρτήσεων

- $n^2 = O(n^2)$
- $n^3 + 1000n^2 + n = O(n^3)$
- $1000\, n = O(n)$

- $20n^3 = O(0.5n^3 + n^2)$

Στον παρακάτω πίνακα παραθέτουμε τους χρόνους εκτέλεσης αλγορίθμων ανάλογα με την ασυμπτωτική πολυπλοκότητά τους. Οι συναρτήσεις $O(1)$, ή συναρτήσεις σταθερής χρονικής πολυπλοκότητας, είναι οι πιο αποδοτικές μια που ο χρόνος εκτέλεσης του αλγορίθμου είναι ανεξάρτητος από το μέγεθος της εισόδου. Οι συναρτήσεις δίνονται σύμφωνα με την αποτελεσματικότητά τους (από μικρότερους σε μεγαλύτερους χρόνους εκτέλεσης). Παρατηρούμε ότι ένας αλγόριθμος με πολυπλοκότητα δευτέρου βαθμού είναι αρκετά γρήγορος για μικρά n. Όμως, όταν το n = 1.000.000, τότε βλέπουμε ότι τα πράγματα χειροτερεύουν κι ο αλγόριθμος χρειάζεται αρκετές μέρες για να εκτελεστεί. Για κυβική πολυπλοκότητα για μεγάλες εισόδους (n>10.000) δεν εκτελείται καθόλου ενώ αλγόριθμοι με εκθετική ή παραγοντική πολυπλοκότητα είναι ασύμφοροι και πρακτικά αδύνατοι.

	Ονομασία	$n = 100$	$n = 10000$	$n = 1000000$
$O(1)$	Σταθερού χρόνου	0.000001sec	0.000001sec	0.000001sec
$O(\lg n)$	Λογαριθμικού χρόνου	0.000007sec	0.000013sec	0.00002sec
$O(n)$	Γραμμικού χρόνου	0.0001sec	0.01sec	1sec
$O(n \lg n)$		0.00066sec	0.13sec	20sec
$O(n^2)$	Τετραγωνικού χρόνου	0.01sec	100sec	278 ώρες
$O(n^3)$	Κυβικού χρόνου	1sec	278 ώρες	317 αιώνες
$O(2^n)$	Εκθετικού	10^{14} αιώνες	10^{2995} αιώνες	10^{30087} αιώνες

	χρόνου			
$O(n!)$	Παραγοντικού χρόνου	10^{143} αιώνες	10^{35645} αιώνες	∞

Ασυμπτωτικές πολυπλοκότητες για μια μηχανή που εκτελεί ένα εκατομμύριο βήματα σε ένα δευτερόλεπτο

Οι συμβολισμοί Ω και Θ

Ο συμβολισμός Ω είναι ο ανάποδος του συμβολισμού *O*. Ενώ ο *O* εκφράζει το άνω άκρο, ο συμβολισμός Ω εκφράζει το κάτω άκρο – φράγμα των συναρτήσεων πολυπλοκότητας. Δηλαδή,

$$\Omega(g(n)) = \{ \begin{matrix} f(n) : c \geq 0, n_0 \geq 0 : \\ 0 \leq cg(n) \leq f(n) \; \forall \; n \geq n_0 \end{matrix}$$

ή διαφορετικά,

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

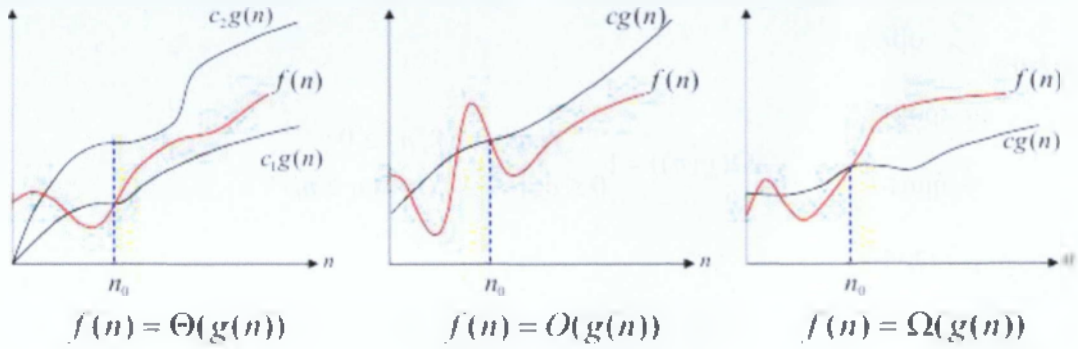
Οι συναρτήσεις Ω δεν ενδιαφέρουν τόσο όσο οι συναρτήσεις O μια που, κατά κανόνα, ενδιαφερόμαστε για τον μέγιστο χρόνο που απαιτεί ένας αλγόριθμος προκειμένου να εκτελεστεί και όχι για τον ελάχιστο.

Ο συμβολισμός Θ, από την άλλη μεριά, εκφράζει τόσο το άνω όσο και το κάτω άκρο και οι συναρτήσεις που περιλαμβάνει αφορούν σε αυτές που, για επαρκώς μεγάλο n και θετικές σταθερές c_1 και c_2, η πολυπλοκότητα του αλγόριθμου $f(n)$ περικλείεται και πάνω και κάτω από αυτές. Ο συμβολισμός Θ, αν και πιο ακριβής από τον συμβολισμό O, δεν χρησιμοποιείται τόσο όσο ο τελευταίος μια που είναι αφενός πολύ πιο δύσκολος στο να αποδειχθεί ενώ, στις περισσότερες περιπτώσεις, το άνω όριο είναι αρκετό.

Συμβολισμός	Ορισμός	Αναλογία με αριθμητική σύγκριση
$f(n) = O(g(n))$	Υπάρχει $c > 0$ και $n_0 > 0$ έτ. ώ. $0 \le f(n) \le cg(n)$	
$f(n) = \Omega(g(n))$	Υπάρχει $c > 0$ και $n_0 > 0$ έτ. ώ. $0 \le cg(n) \le f(n)$	
$f(n) = \Theta(g(n))$	$f(n) = O(g(n))$ και $f(n) = \Omega(g(n))$	

Συμβολισμοί Ο, Ω και Θ

Οι παραπάνω ορισμοί μπορούν να ειδωθούν και γραφικά όπως φαίνεται παρακάτω:



$f(n) = \Theta(g(n))$ $f(n) = O(g(n))$ $f(n) = \Omega(g(n))$

Είδη Πολυπλοκότητας

Εάν η χρονική πολυπλοκότητα ενός αλγορίθμου ασυμπτωτικά φράσσεται από ένα πολυώνυμο, τότε λέμε ότι η πολυπλοκότητα είναι πολυωνυμική ενώ διαφορετικά, λέμε ότι η πολυπλοκότητα είναι εκθετική.

Κλάσεις Πολυπλοκότητας

Είδαμε ότι η πολυπλοκότητα εκφράζει την αποτελεσματικότητα του αλγορίθμου. Η πολυπλοκότητα ενός προβλήματος αντιστοιχεί στην πολυπλοκότητα του πιο αποτελεσματικού αλγορίθμου. Είδαμε, π.χ., ότι στο παράδειγμα της αναζήτησης μιας

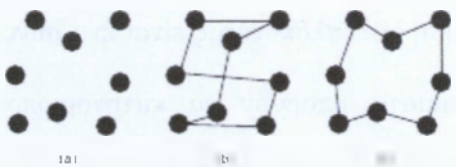
λέξης από ένα λεξικό, η πολυπλοκότητα του αλγορίθμου Δυαδική_Αναζήτηση ήταν $O(\lg n)$ που, όντας ο καλύτερος αλγόριθμος, αποτελεί και την πολυπλοκότητα του προβλήματος. Προκειμένου, έτσι, να διευκολυνθεί η αναζήτηση των πολυπλοκοτήτων διαφόρων προβλημάτων, αναπτύχθηκαν οι κλάσεις πολυπλοκότητας οι οποίες περιλαμβάνουν προβλήματα με τον ίδιο βαθμό πολυπλοκότητας.

Σε γενικές γραμμές, μια κλάση πολυπλοκότητας είναι ένα σύνολο προβλημάτων με ίδια πολυπλοκότητα. Τα προβλήματα μπορούν να κατηγοριοποιηθούν σε δύο γενικές κατηγορίες ανάλογα με το είδος των απαντήσεων που δίνουν:

- τα προβλήματα αποφάσεων και
- τα προβλήματα βελτιστοποίησης.

Στα προβλήματα αποφάσεων ένα 'ναι' ή 'όχι' ως απάντηση είναι αρκετό. Ένα τέτοιο πρόβλημα αποφάσεων ήταν και η αναζήτηση λέξης που είδαμε προηγουμένως. Ένα, όμως, πρόβλημα βελτιστοποίησης αναζητά την βέλτιστη τιμή για μια μεταβλητή. Για παράδειγμα, σε ένα συνδυαστικό κύκλωμα, η κρίσιμη διαδρομή αποτελεί τον μέγιστο δρόμο από την είσοδο στην έξοδο. Η βελτιστοποίηση στο παράδειγμα αυτό, σημαίνει μεγιστοποίηση. Σε άλλα προβλήματα, η βελτιστοποίηση σημαίνει ελαχιστοποίηση. Ένα διάσημο πρόβλημα βελτιστοποίησης – ελαχιστοποίησης είναι το πρόβλημα του ταξιδεύοντος εμπόρου (TSP) στο οποίο κανείς θέλει να βρει την ελάχιστη κλειστή διαδρομή ανάμεσα σε συγκεκριμένα μέρη. Το πρόβλημα TSP διαθέτει και ισοδύναμη εκφώνηση και ως πρόβλημα αποφάσεων. Για παράδειγμα, μπορεί κανείς να ρωτήσει εάν υπάρχει μια κλειστή διαδρομή ανάμεσα στα σημεία τέτοια ώστε το μήκος της να είναι μικρότερο από μια τυχαία σταθερά k. Το πρόβλημα βελτιστοποίησης TSP είναι σαφώς δυσκολότερο να λυθεί από το πρόβλημα λήψης αποφάσεων TSP. Εξάλλου, εάν το πρώτο

λυθεί, το δεύτερο αυτόματα απαντάται. Σε αρκετές περιπτώσεις, τα προβλήματα βελτιστοποίησης αναλύονται σε μικρότερα προβλήματα λήψης αποφάσεων με χρήση διαφορετικών σταθερών k, μέχρι να βρεθεί το βέλτιστο αποτέλεσμα. Συνεπώς, το πρόβλημα βελτιστοποίησης έχει σαφώς μεγαλύτερη πολυπλοκότητα από το αντίστοιχο λήψης αποφάσεων.



Πρόβλημα TSP (a), μια τυχαία διαδρομή (b), η βέλτιστη διαδρομή (c)

Πολυπλοκότητες τύπου P και NP

Η πολυπλοκότητα τύπου P αναφέρεται στην κλάση προβλημάτων τα οποία μπορούν να λυθούν με αλγόριθμους οι οποίοι έχουν συνάρτηση πολυπλοκότητας χρόνου ένα γνωστό πολυώνυμο. Με άλλα λόγια, ένα οποιοδήποτε πρόβλημα κλάσης P έχει αλγόριθμο πολυπλοκότητας $O(n^k)$ με k μια σταθερά. Η γραμμική αναζήτηση προηγουμένως ήταν ένας αλγόριθμος με πολυπλοκότητα $O(n)$ (k=1).

Η πολυπλοκότητα τύπου NP από την άλλη μεριά, αναφέρεται σε μη ντετερμινιστικά πολυώνυμα τα οποία αποδίδονται σε έναν μη ντετερμινιστικό υπολογιστή ο οποίος δουλεύει σε πολυωνυμικό χρόνο. Ο μη ντετερμινιστικός υπολογιστής είναι μια αφηρημένη έννοια και, άρα, δεν μπορεί να κατασκευαστεί από συγκεκριμένα μέρη. Υπάρχει ως θεωρητικό εργαλείο μόνο στη θεωρία της πολυπλοκότητας. Ο ντετερμινιστικός υπολογιστής, από την άλλη μεριά, είναι κάθε άλλος – κοινός – υπολογιστής ο οποίος λύνει ντετερμινιστικούς αλγορίθμους. Ένας αλγόριθμος ονομάζεται ντετερμινιστικός όταν σε κάθε στιγμή της υπολογιστικής διαδικασίας, το

επόμενο βήμα είναι σαφώς ορισμένο έτσι ώστε για κάθε όμοια είσοδο η έξοδος να είναι πάντα η ίδια. Ένας αλγόριθμος ονομάζεται μη ντετερμινιστικός όταν σε κάθε στιγμή της υπολογιστικής διαδικασίας υπάρχουν διάφορες επιλογές για το επόμενο βήμα και ο υπολογιστής κάνει μια μη ντετερμινιστική επιλογή ανάμεσα σε αυτές. Τα προβλήματα στην κλάση NP έχουν τα εξής χαρακτηριστικά:

1. Είναι προβλήματα λήψης αποφάσεων
2. Μπορούν να λυθούν σε πολυωνυμικό χρόνο από έναν μη ντετερμινιστικό υπολογιστή
3. Η λύση τους μπορεί να εξακριβωθεί από έναν ντετερμινιστικό υπολογιστή σε πολυωνυμικό χρόνο

Παράδειγμα προβλήματος NP 1

Ένα παράδειγμα προβλήματος πληρότητας – NP είναι αυτό του 'ταξιδεύοντος εμπόρου' (travelling salesman) το οποίο μπορεί να δοθεί όπως παρακάτω:

Έστω ότι μια εταιρία ταχυμεταφορών έχει ένα κεντρικό γραφείο και κάθε μέρα, γεμίζει κάθε ένα φορτηγό με προϊόντα που πρέπει να παραδοθούν και στέλνει τα φορτηγά αυτά στις αντίστοιχες διευθύνσεις. Στο τέλος κάθε μέρας, κάθε ένα φορτηγό πρέπει να γυρίσει στο κεντρικό γραφείο ώστε να είναι έτοιμο να φορτωθεί για την επόμενη μέρα. Προκειμένου να ελαττώσει το κόστος, η εταιρία θα ήθελε να γνωρίζει εάν θα μπορούσε να διαλέξει με τέτοιο τρόπο τις θέσεις παράδοσης ώστε να βρεί την ελάχιστη απόσταση την οποία θα διανύουν τα φορτηγά της οποιαδήποτε μέρα. Αυτό είναι ένα παράδειγμα προβλήματος πληρότητας – NP το οποίο δεν έχει κάποιο αποτελεσματικό αλγόριθμο λύσης. Κάτω, όμως, από ορισμένες προϋποθέσεις, υπάρχουν αλγόριθμοι που δίνουν μια συνολική απόσταση όχι πολύ διαφορετική από την ελάχιστη. Αλλά, αυτό δεν αποτελεί

και καθοριστική λύση του προβλήματος.

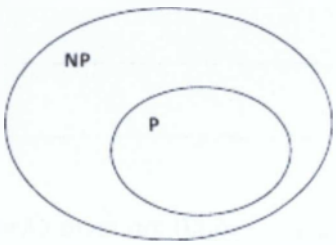
Όπως είχε αναφέρει ο Εντστερ Νιτζκστρα – ο εφευρέτης του αλγόριθμου εύρεσης της μικρότερης απόστασης μεταξύ δύο σημείων – στο άρθρο του 'μερικές σκέψεις στον προηγμένο προγραμματισμό': 'ένα πρακτικό πρόβλημα, το οποίο ήδη αρχίζει να γίνεται ιδιαίτερα απειλητικό χωρίς να φαίνεται να υπάρχει κάποια ικανοποιητική λύση του, είναι ο έλεγχος της ακρίβειας ενός μεγάλου προγράμματος'. Πέρα από αυτό, η κατηγοριοποίηση των προβλημάτων ανάλογα με τη δυσκολία τους αλλά και η εκτίμηση του κόστους της λύσης τόσο σε χρόνο, όσο και σε μνήμη και υπολογιστική ισχύ, αποτελεί ένα μεγάλο κεφάλαιο της πληροφορικής που λέγεται θεωρία πολυπλοκότητας η οποία αναζητά, δεδομένου των διαθέσιμων πόρων (μνήμη – χρόνος), την βέλτιστη μέθοδο αντιμετώπισης ενός προβλήματος. Διαφοροποιείται, έτσι, από την ανάλυση αλγορίθμων μια που η τελευταία αναζητά τον βέλτιστο αλγόριθμο για ένα συγκεκριμένο πρόβλημα. Η θεωρία της πολυπλοκότητας θέτει το ερώτημα του κατά πόσο το συγκεκριμένο πρόβλημα μπορεί να επιλυθεί από διάφορους αλγόριθμους, δεδομένου των διαθέσιμων πόρων.

Το πρόβλημα TSP στην μορφή του ως πρόβλημα λήψης αποφάσεων ικανοποιεί τα δύο πρώτα κριτήρια παραπάνω προκειμένου να ταξινομηθεί στην κλάση NP. Επίσης, ικανοποιεί και το τρίτο κριτήριο μια που το μήκος της διαδρομής μπορεί να εξακριβωθεί ότι είναι (ή όχι) το μικρότερο από έναν ντετερμινιστικό υπολογιστή σε γραμμικό χρόνο συναρτήσει των σημείων από τα οποία πρέπει να περάσει ο έμπορος. Συνεπώς, το πρόβλημα TSP είναι ένα πρόβλημα NP.

Επειδή ένα πρόβλημα που μπορεί να λυθεί σε πολυωνυμικό χρόνο από έναν ντετερμινιστικό υπολογιστή μπορεί να λυθεί σε πολυωνυμικό χρόνο κι από έναν μη

ντετερμινιστικό υπολογιστή ισχύει:

$$P \subset NP$$



Η ερώτηση του κατά πόσο NP = P είναι ακόμα ανοικτή και το Clay Mathematics Institute δίνει ένα εκατομμύριο δολάρια σε όποιον το λύσει. Προκειμένου να λυθεί το πρόβλημα, τα πιο δύσκολα προβλήματα στην κλάση NP ομαδοποιήθηκαν σε μια καινούρια ομάδα με το όνομα NP-πλήρης (NPC) έτσι ώστε το NP = P θεώρημα να μπορεί να γίνει NP-πλήρης = P.

Παράδειγμα προβλήματος NP 2

Ένα άλλο παράδειγμα προβλήματος NP είναι το πρόβλημα λήψης αποφάσεων subset sum problem (πρόβλημα αθροίσματος υποσυνόλου). Σύμφωνα με το πρόβλημα αυτό, έστω ένα ακέραιο άνω όριο W και μια συλλογή n αντικειμένων με w_i ακέραια βάρη. Υπάρχει ένα υποσύνολο S αυτής της συλλογής τέτοιο ώστε να μεγιστοποιεί το άθροισμα των βαρών αλλά να διατηρεί το άθροισμα αυτό μικρότερο του W;

Ένας ψευδοκώδικας που αντιμετωπίζει το πρόβλημα αυτό φαίνεται παρακάτω:

Αλγόριθμος 3 Subset Sum Problem

```
SubsetSum(n, W):

Αρχικοποιήστε τον πίνακα  M[0,r] = 0 για κάθε  r = 0,...,W // οι στήλες
Αρχικοποιήστε τον πίνακα  M[j,0] = 0 για κάθε  j = 1,...,n  //οι γραμμές
Για  j = 1,...,n:
  Για  r = 0,...,W:
    Εάν w[j] > r:
      M[j,r] = M[j-1,r]
```

```
    M[j,r] = max(M[j-1,r],w[j] + M[j-1, W-w[j]])
```

Επιστροφή M[n,W]

Ο χρόνος εκτέλεσης του παραπάνω αλγόριθμου είναι $O(nW+n) = O(nW)$ και ονομάζεται χρόνος ψευδο-πολυωνυμικός μια που εξαρτάται από το μέγεθος των αριθμών της εισόδου. Το πρόβλημα αυτό είναι NPC.

Παράδειγμα προβλήματος NP 3

Υπάρχουν προβλήματα τα οποία είναι NP αλλά δεν είναι NPC. Ένα τέτοιο είναι το halting problem (πρόβλημα τερματισμού) σύμφωνα με το οποίο, δεδομένης μιας περιγραφής ενός υπολογιστικού κώδικα, εξετάστε εάν ο κώδικας αυτός θα τερματίσει ή αν θα συνεχίσει να τρέχει οπότε θα τρέχει συνεχώς. Σε γενικές γραμμές το πρόβλημα αυτό εξετάζει εάν, δεδομένου ενός προγράμματος και μιας εισόδου, το πρόγραμμα αυτό θα τερματίσει ή θα τρέχει συνεχώς.

Αλγόριθμος 4 Halting Problem

```
procedure compute_g(i):
    f(i,i) == 0 then
      return 0
   else
      loop forever
```

Η συνάρτηση f στον αλγόριθμο ονομάζεται ολικά υπολογίσιμη συνάρτηση (total computable function) και αποφασίζει εάν ένα τυχαίο πρόγραμμα i σταματά σε τυχαία είσοδο x. Δεδομένης της f μπορούμε να υπολογίσουμε την μερική συνάρτηση g η οποία

δίνει 0 όταν το f(i,i)=0 και επιστρέφει 'ΔΕΝ ΟΡΙΖΕΤΑΙ' σε διαφορετική περίπτωση. Μια εικόνα πιθανών τιμών της g για δεδομένες εξόδους της f δίνεται παρακάτω:

f(i,j)	i					
	1	2	3	4	5	6
1	**1**	0	0	1	0	1
2	0	**0**	0	1	0	0
3	0	1	**0**	1	0	1
4	1	0	0	**1**	0	0
5	0	0	0	1	**1**	1
6	1	1	0	0	1	**0**
f(i,i)	1	0	0	1	1	0
g(i)	U	0	0	U	U	0

Η NP-πληρότητα

Η NP-πληρότητα (NPC) αποτελείται από όλα τα προβλήματα NP που είναι δύσκολο να λυθούν. Με μαθηματικούς ορισμούς, η NP-πληρότητα ορίζεται ως εξής:

Για ένα τυχαίο πρόβλημα P_a στην κλάση NP και ένα οποιοδήποτε πρόβλημα P_b στην κλάση NPC, υπάρχει πολυωνυμικός μετασχηματισμός (αναγωγή) τέτοιος ώστε να μετατρέπει ένα παράδειγμα του P_a σε ένα παράδειγμα του P_b.

Κάθε πρόβλημα NPC μπορεί να αναχθεί από οποιοδήποτε πρόβλημα NP. Επίσης, κάθε πρόβλημα NPC μπορεί να μετασχηματιστεί σε οποιοδήποτε άλλο NPC πρόβλημα μέσω κάποιου πολυωνυμικού μετασχηματισμού. Συνεπώς, για να αποδείξουμε ότι ένα πρόβλημα P_t είναι NPC μόνο δύο κριτήρια πρέπει να ικανοποιούνται:

1. Το πρόβλημα P_t θα πρέπει να είναι NP: θα πρέπει να λύνεται σε πολυωνυμικό χρόνο από έναν μη ντετερμινιστικό υπολογιστή και να δίνει λύση που να ελέγχεται σε πολυωνυμικό χρόνο από ντετερμινιστικό υπολογιστή
2. Ένα πρόβλημα που ήδη γνωρίζουμε ότι είναι NPC να μπορεί να μετασχηματιστεί

πολυωνυμικά στο πρόβλημα P_1.

Αν και τα προβλήματα NPC είναι πολύ δύσκολα στο να λυθούν, υπάρχουν προβλήματα ακόμη πιο δύσκολα: τα NP-δύσκολα (NP-hard). Τα NP-δύσκολα είναι προβλήματα που είναι τουλάχιστον τόσο δύσκολα να λυθούν όσο τα NPC. Τα NP-δύσκολα είναι προβλήματα που είναι τουλάχιστον τόσο δύσκολα να λυθούν όσο τα NPC. Τα NP-δύσκολα προβλήματα ακόμα κι αν λυθούν σε πολυωνυμικό χρόνο από μη ντετερμινιστικό υπολογιστή, εντούτοις οι λύσεις τους, δεν μπορούν να ελεγχθούν σε πολυωνυμικό χρόνο από ντετερμινιστικό υπολογιστή. Πολλά προβλήματα βελτιστοποίησης στην κλάση NPC είναι NP-δύσκολα.

Παράδειγμα προβλήματος NPC 1

Το πρόβλημα TSP που εξετάσαμε παραπάνω είναι ένα παράδειγμα προβλήματος NPC. Το πρόβλημα βελτιστοποίησης TSP το οποίο ψάχνει την κλειστή διαδρομή από όλα τα σημεία η οποία να έχει το μικρότερο μήκος είναι ένα πρόβλημα NP-δύσκολο. Προκειμένου να λύσουμε το πρόβλημα αυτό θα πρέπει να υπολογίσουμε τα μήκη όλων των πιθανών διαδρομών η οποία είναι μια διαδικασία με χρονική πολυπλοκότητα $O(n \cdot n!)$, με n τον αριθμό των σημείων. Αυτή η πολυπλοκότητα δεν έχει πολυωνυμικό χρόνο και άρα το πρόβλημα βελτιστοποίησης TSP είναι ένα πρόβλημα NP-δύσκολο.

Παράδειγμα προβλήματος NPC 2

Δεδομένου ενός γραφήματος G, υπάρχει κύκλος που περνάει από κάθε κόμβο του G ακριβώς μία φορά. Ενας τέτοιος κύκλος ονομάζεται Hamiltonian και ένα γράφημα που έχει έναν τέτοιο ονομάζεται γράφημα Hamilton. Πρέπει να διασχίσουμε τους κόμβους ακριβώς μία φορά.

Υπάρχουν n! διαφορετικές επιλογές (μεταθέσεις) κόμβων οι οποίες θα μπορούσαν να φτιάξουν έναν κύκλο Hamilton δεδομένου ενός γραφήματος με n κόμβους. Έτσι, ένας 'απλοϊκός' αλγόριθμος θα μπορούσε να ελέγξει εάν όλες οι πιθανές επιλογές κόμβων θα μπορούσαν να είναι κύκλοι Hamilton. Κάτι τέτοιο, όμως αναμένεται να είναι ιδιαίτερα αργό.

Υπάρχουν, όμως, αρκετές γρηγορότερες προσεγγίσεις. Η διαδικασία του Rubin διαιρεί τις ακμές του γραφήματος σε τρεις κλάσεις: αυτές που αναγκαστικά πρέπει να ανήκουν στον κύκλο, αυτές που δεν μπορούν να ανήκουν στον κύκλο και αυτές που δεν γνωρίζουμε εάν θα πρέπει να ανήκουν ή όχι στον κύκλο. Όσο προχωρά η αναζήτηση, μια σειρά κανόνων επιλέγει τις ακμές για τις οποίες δεν υπάρχει κάποια σίγουρη απάντηση ναι/όχι ενώ, ταυτόχρονα, αποφασίζει εάν θα πρέπει να σταματήσει την αναζήτηση ή να την συνεχίσει. Ο αλγόριθμος διαιρεί το γράφημα σε κομμάτια τα οποία μπορούν να λυθούν ξεχωριστά.

Επίσης, η διαδικασία των Bellman, Held, και Karp μπορεί να χρησιμοποιηθεί προκειμένου να λύσει το πρόβλημα σε χρόνο $O(n^2 2^n)$. Σύμφωνα με αυτήν την μέθοδο, κανείς ελέγχει εάν για κάθε σύνολο S ακμών και κάθε ακμή v στο S, υπάρχει διαδρομή που να περνά από όλα τα σημεία του S και να καταλήγει στο v. Για να ελεγχθεί αυτό χρησιμοποιείται το γεγονός ότι για κάθε επιλογή S και v, υπάρχει διαδρομή (S,v) εάν και μόνο εάν v έχει ένα γειτονικό σημείο w τέτοιο ώστε αντίστοιχη διαδρομή υφίσταται για το (S-v, w). Η τελευταία πληροφορία μπορεί να αντληθεί από τη δυναμική εκτέλεση του προγράμματος.

Η δυσκολία του προβλήματος αυτού οδήγησε τον Adleman να αποδείξει ότι το πρόβλημα του χαμιλτονιανού κύκλου μπορεί να λυθεί μέσω ενός υπολογιστή DNA.

Εκμεταλλευόμενος τις ιδιότητες των χημικών αντιδράσεων, το πρόβλημα μπορεί να λυθεί χρησιμοποιώντας έναν αριθμό από βήματα χημικών αντιδράσεων που να είναι γραμμικά σε σχέση με τον αριθμό των κόμβων στο γράφημα. Ο αλγόριθμος αυτός, όμως, απαιτεί παραγοντικό αριθμό μορίων DNA να συμμετέχουν στις αντιδράσεις.

Παράδειγμα προβλήματος NPC 3

Το πρόβλημα subset sum που είδαμε παραπάνω.

Παράδειγμα προβλήματος NPC 4

Η χρήση μετασχηματισμών για την απόδειξη ότι κάποιο πρόβλημα είναι NPC βασίζεται στο γεγονός ότι ήδη γνωρίζουμε ότι κάποιο πρόβλημα είναι NPC μέσω κάποιου άλλου τρόπου. Το 1971 ο Cook απέδειξε, με άλλο τρόπο, ότι το πρόβλημα SAT (satisfiability) είναι NPC. Έστω μια σειρά από λογικές (Boolean) μεταβλητές. Τότε, μια έκφραση της μορφής:

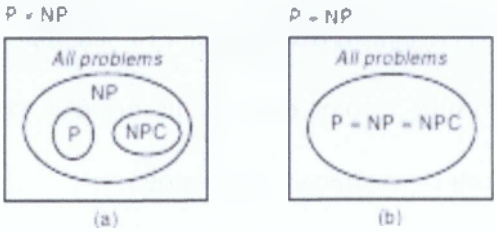
$$(x_1 + x_2 + x_3)(x_2 + \bar{x}_4)(\bar{x}_1 + \bar{x}_3)(\bar{x}_2 + x_3 + x_4)$$

εξετάζεται εάν δίνει απάντηση TRUE. Εάν αυτό συμβαίνει, τότε η λύση είναι SATISFIABLE. Οι παύλες πάνω από τις μεταβλητές συμβολίζουν την αντίθετη είσοδο. Π.χ. εάν η x_1 είναι TRUE τότε η $\bar{x}_1$ είναι FALSE.

Το πρόβλημα SAT είναι ένα πρόβλημα λήψης αποφάσεων. Έχει δύο πιθανές εξόδους: TRUE/FALSE. Το ότι το πρόβλημα SAT είναι NPC αποδεικνύεται σύμφωνα με το θεώρημα του Cook το οποίο λέει ότι όλα τα προβλήματα NPC μπορούν να αναχθούν – σε πολυωνυμικό χρόνο – στο πρόβλημα SAT. Η απόδειξη του θεωρήματος του Cook έχει την εξής λογική: κάθε NP πρόβλημα λύνεται σε πολυωνυμικό χρόνο από έναν μη ντετερμινιστικό υπολογιστή. Για ένα τυχαίο NP πρόβλημα, εάν σημειώσουμε όλα τα

βήματα που ακολουθούνται προκειμένου να λυθεί το πρόβλημα, σύμφωνα με το θεώρημα του Cook, τα βήματα αυτά μπορούν να αναχθούν σε γινόμενα αθροισμάτων, τα οποία, στην ουσία είναι γινόμενα SAT. Έτσι, κάθε πρόβλημα NP μπορεί να αναχθεί σε ένα πρόβλημα SAT.

Μια ανοιχτή ερώτηση στην θεωρία της πληροφορικής είναι εάν υπάρχει πρόβλημα που να είναι και P και NPC. Αυτό σημαίνει ότι εάν βρεθεί ένας αλγόριθμος ο οποίος να λύνεται σε πολυωνυμικό χρόνο από ντετερμινιστικό υπολογιστή και ταυτόχρονα να μπορεί να λύσει ένα πρόβλημα NPC, τότε, αφού όλα τα NPC προβλήματα ανάγονται το ένα στο άλλο, θα μπορεί να λύσει – με τον κατάλληλο μετασχηματισμό – και όλα τα άλλα. Συνεπώς, σε αυτήν την περίπτωση NP = P = NPC. Αν και η γενική σκέψη είναι ότι P ≠ NP, δεν υπάρχει, ακόμα, κάποιος σαφής και ασφαλής λόγος να υποθέσουμε ότι αυτό είναι αλήθεια.



Σχέση ανάμεσα στις κλάσεις NP και P (a) όταν NP ≠ P και (b) NP = P

Μια άλλη εκδοχή του προβλήματος SAT θεωρεί την παρακάτω αλληλουχία εισόδων:

$$(x_1 + \bar{x}_2 + x_3 + \bar{x}_4)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3 + x_4)(x_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_5)(\bar{x}_1 + \bar{x}_2 + \bar{x}_5 + x_4)$$

Για $x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0$, τότε ικανοποιούνται και οι τέσσερις είσοδοι.

Αναγωγές

Πολυωνυμικές Αναγωγές

Ένας ***πολυωνυμικός μετασχηματισμός (αναγωγή)*** ορίζεται ως εξής: Έστω δύο προβλήματα λήψης αποφάσεων P_a και P_b, ένας πολυωνυμικός μετασχηματισμός (αναγωγή) από το P_a στο P_b μπορεί να εκφράσει κάθε παράδειγμα του P_a ως παράδειγμα του P_b. Τότε, το ανηγμένο παράδειγμα P_b μπορεί να λυθεί από έναν αλγόριθμο για το P_b και η απάντησή του (αλγόριθμου) να μπορεί να αναχθεί σε απάντηση για το παράδειγμα P_a. Ένας πολυωνυμικός μετασχηματισμός έχει πάντα πολυωνυμική χρονική πολυπλοκότητα. Εάν υπάρχει πολυωνυμικός μετασχηματισμός από το P_a στο P_b τότε λέμε ότι το P_a είναι πολυωνυμικά ανηγμένο στο P_b.

Πρακτικά, λέμε ότι ένα πρόβλημα λήψης αποφάσεων Α ανάγεται σε ένα πρόβλημα Β όταν μπορούμε κάθε στιγμιότυπο του Α να το μεταφράσουμε σε ένα στιγμιότυπο του Β ώστε, εάν λύσουμε το Β, να έχουμε αυτόματα και μια απάντηση για το Α. Εάν η αναγωγή είναι πολυωνυμική τότε μπορούμε να ισχυριστούμε ότι το πρόβλημα Β είναι τουλάχιστον τόσο δύσκολο όσο το Α (αφού ο αλγόριθμος για τη λύση του Β λύνει και το Α). Συνεπώς, για να αποδείξουμε ότι το Α ανάγεται στο Β πρέπει να δείξουμε ότι:

- Υπάρχει μετασχηματισμός τέτοιος ώστε να μεταφράζει κάθε στιγμιότυπο του Α σε κάποιο στιγμιότυπο του Β σε πολυωνυμικό χρόνο
- Εάν η απάντηση στο πρόβλημα Β είναι TRUE τότε και μόνο τότε η απάντηση στο Α είναι επίσης TRUE

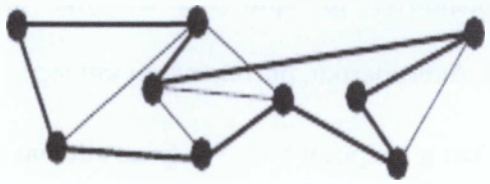
Τέτοιου είδους αναγωγές ονομάζονται αναγωγές Karp και συμβολίζονται ως: $A \leq_P B$ (το Α ανάγεται πολυωνυμικά στο Β)

Παράδειγμα πολυωνυμικής αναγωγής

Το πρόβλημα του Χαμιλτονιακού Κύκλου ανάγεται στο πρόβλημα του ταξιδεύοντος εμπόρου (TSP). Για ένα δεδομένο αριθμό σημείων και γραμμών τέτοιων ώστε η κάθε

γραμμή να ενώνει δύο σημεία, ένας χαμιλτονιακός κύκλος είναι μια κλειστή διαδρομή τέτοια ώστε να περνά από όλα τα σημεία μόνο μια φορά. Ο χαμιλτονιακός κύκλος μπορεί να αναχθεί στο πρόβλημα TSP αποδίδοντας σε κάθε γραμμή μεταξύ δύο σημείων έναν αριθμό – το μήκος της γραμμής.

Για δύο σημεία που συνδέονται μέσω μιας γραμμής ορίζουμε π.χ. απόσταση 1 ενώ για όλα τα άλλα σημεία ορίζουμε απόσταση π.χ. 2. Δηλαδή, κάθε ζεύγος γειτονικών σημείων έχει απόσταση 1 ενώ οι αποστάσεις μη γειτονικών σημείων είναι 2. Το πρόβλημα TSP, έτσι, ρωτάει εάν υπάρχει διαδρομή από όλα αυτά τα σημεία η οποία να μην ξεπερνά σε μήκος το n – τον αριθμό των σημείων. Αυτή η διατύπωση είναι ισοδύναμη με την διατύπωση του προβλήματος του Χαμιλτονιακού κύκλου. Αυτή η αναγωγή από το πρόβλημα του Χαμιλτονιακού κύκλου στο πρόβλημα TSP βασίζεται στην απόδοση μηκών σε κάθε γραμμή και έχει τετραγωνική χρονική πολυπλοκότητα σε σχέση με τον αριθμό των σημείων n. Συνεπώς, ο μετασχηματισμός είναι πολυωνυμικός.



Χαμιλτονιακός Κύκλος

Γνωρίζουμε, για παράδειγμα, ότι το πρόβλημα του χαμιλτονιακού κύκλου είναι NPC και είδαμε παραπάνω ότι μπορεί να αναχθεί πολυωνυμικά στο πρόβλημα TSP. Άρα και το TSP είναι NPC.

Αναγωγές Τούρινγκ

Οι αναγωγές Τούρινγκ είναι γενικότερες από τις πολυωνυμικές που αναφέρθηκαν πιο πάνω και αφορούν όχι μόνο σε προβλήματα λήψης αποφάσεων. Εάν υπάρχει μια

αναγωγή Τούρινγκ από το πρόβλημα Α στο πρόβλημα Β τότε κάθε αλγόριθμος για το Β μπορεί να χρησιμοποιηθεί για να λύσει και το Α. Η λύση του Α προκύπτει εάν η μη ντετερμινιστική μηχανική Τούρινγκ χρησιμοποιεί σε κάθε βήμα για τη λύση του Α τον αλγόριθμο για το Β και κάθε στιγμιότυπο του Α προκύπτει από ένα στιγμιότυπο του Β σε πολυωνυμικό χρόνο. Για παράδειγμα, το πρόβλημα του υπολογισμού του μέγιστου αριθμού μη αλληλο-επικαλυπτόμενων τόξων σε ένα κύκλο είναι ένα πρόβλημα αναγωγής Τούρινγκ.

Παράδειγμα Αναγωγής 1

Το πρόβλημα του σακιδίου και πως αυτό μπορεί να αναχθεί στο πρόβλημα TSP.

Λύση

[Το πρόβλημα του σακιδίου μπορεί να διατυπωθεί ως εξής: Έστω ένας σάκος με συγκεκριμένη αντοχή (σε βάρος). Έστω N αντικείμενα, κάθε ένα με δικό του βάρος και αξία. Το πρόβλημα που τίθεται αφορά στο ποια αντικείμενα μπορούν να μπουν στον σάκο ώστε να μην ξεπεραστεί το όριο της αντοχής του και, παράλληλα, να μεγιστοποιηθεί η αξία του. Διαφορετικά, διατυπώνεται και ως:

Έστω n ακέραιοι $u_1 \cdots u_n$ και n ακέραιοι $w_1 \cdots w_n$ όπως και δύο ακόμα ακέραιοι U και W.

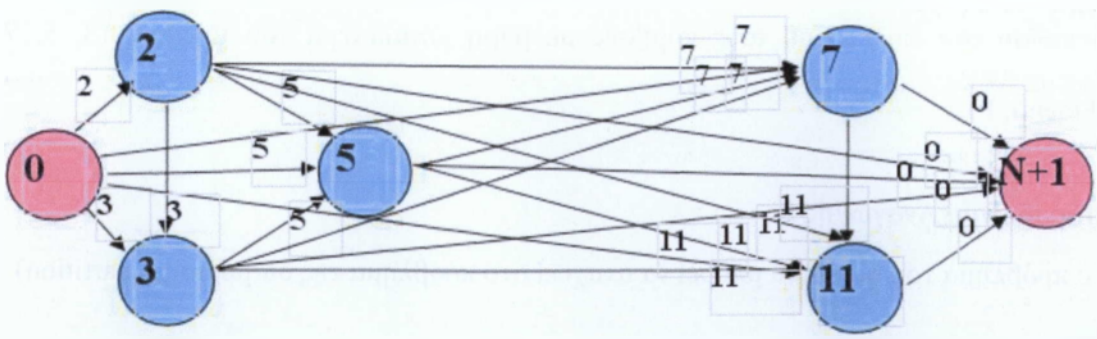
Υπάρχει ακέραιος I στο [1,n] τέτοιος ώστε $\sum_{i \in I} w_i \leq W$ και $\sum_{i \in I} u_i \geq U$;

Για να δούμε την αναγωγή, έστω ότι W = 15 (το συνολικό βάρος που αντέχει ο σάκος) και ότι έχουμε 5 αντικείμενα με βάρη 2, 3, 5, 7 και 11.

Βήμα 1ο: Δημιουργούμε έναν κόμβο για κάθε αντικείμενο

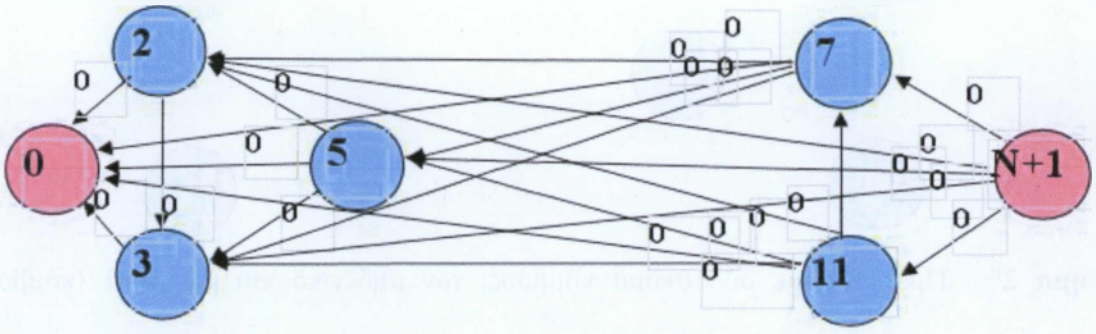
Βήμα 2ο: Προσθέτουμε δύο ακόμα κόμβους: τον μηδενικό και τον N+1 (κόμβος επιστροφής)

Βήμα 3ο: Ζωγραφίζουμε διαδρομές από τους μικρότερους κόμβους στους μεγαλύτερους με βάρος τον αριθμό του μεγαλύτερου κόμβου. Τα βάρη των διαδρομών προς τον N+1 κόμβο εξισώνονται με μηδέν.

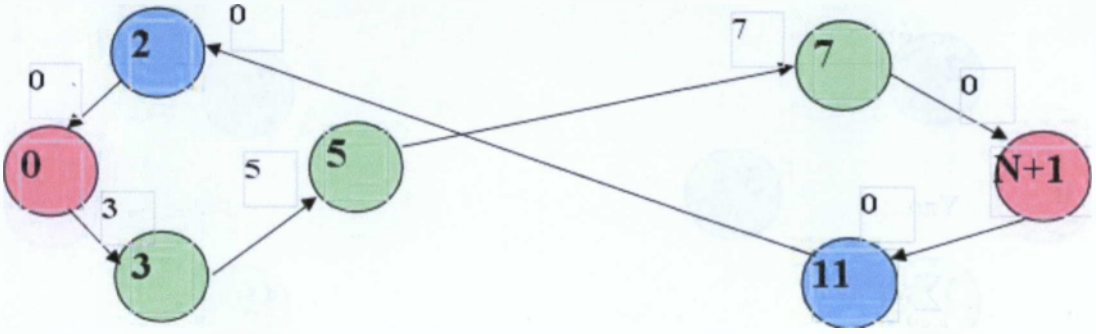


Βήμα 4ο: Ζωγραφίζουμε επιστροφές από τους μεγαλύτερους κόμβους στους μικρότερους

με βάρος μηδέν.



Βήμα 5ο: Επισκεπτόμαστε τους κόμβους σε αύξουσα σειρά ως εξής:



Παρατηρούμε ότι η λύση του προβλήματος του σακιδίου για το συγκεκριμένο παράδειγμα είναι το σετ σάκων με βάση 3, 5 και 7. Αυτή, όμως, είναι και η λύση του TSP μια που η διαδρομή 0-3-5-7-(N+1)-11-2-0 έχει συνολικό βάρος 15. Αντίστροφα, η λύση του TSP με διαδρομή το 0-3-5-7-(N+1)-11-2-0, είναι λύση και του προβλήματος του σακιδίου εάν επιλέξουμε τους κόμβους με βάρη μεγαλύτερα του μηδενός (3, 5, 7)] (Fraenti, P).

Παράδειγμα Αναγωγής 2

Το πρόβλημα του σακιδίου μπορεί να αναχθεί στο πρόβλημα της διαμέρισης (partition)

Λύση

Το πρόβλημα της διαμέρισης (partition) αναφέρεται σε ένα σετ n μη αρνητικών ακεραίων

$\{a_1, a_2, ..., a_n\}$ και αναρωτιέται εάν υπάρχει ένα υποσύνολο $P \subseteq [1, n]$ τέτοιο ώστε $\sum_{i \in P} a_i = \sum_{i \notin P} a_i$. Για να δούμε πως το πρόβλημα του σακιδίου μπορεί να αναχθεί στο πρόβλημα της διαμέρισης πρέπει να κάνουμε τα εξής βήματα:

Βήμα 1ο: Έστω S = $\{a_1, a_2, ..., a_n\}$ μια λύση του προβλήματος του σακιδίου. Ονομάζουμε

$$H = \frac{1}{2}\sum_{1}^{n} a_i$$

Βήμα 2ο: Προσθέτουμε και τους ακεραίους $a_{n+1} = 2H + 2K$ και $a_{n+2} = 4H$. Παρατηρούμε ότι $a_{n+1} \in P$ ενώ υποθέτουμε, χωρίς περιορισμό της γενικότητας, ότι $a_{n+2} \in P$.

Βήμα 3ο : Υπάρχει $P \subseteq [1, n+2]$ με $\sum_{i \in P} a_i = \sum_{i \notin P} a_i$ εάν και μόνο εάν υπάρχει $Q \subseteq S$ τέτοιο ώστε $\sum_{a_i \in Q} a_i = K$. Έστω ότι υπάρχει *P*:

$$4H + \sum_{a_i \in P \setminus a_{n+2}} a_i = 2H + 2K + \sum_{a_i \in S \setminus P} a_i$$

$$\Rightarrow \quad 4H + 2\sum_{a_i \in P \setminus a_{n+2}} a_i = 4H + 2K$$

$$\Rightarrow \quad \sum_{a_i \in P \setminus a_{n+2}} a_i = K$$

Έστω ότι υπάρχει *Q*:

$$\sum_{a_i \in Q} a_i + 4H = 4H + K = 2H + K + \sum_{a_i} a_i = 2H + 2K \sum_{a_i \notin Q} a_i$$

Άρα υπάρχει $P = Q \cup 4H$

Παράδειγμα Αναγωγής 3

Το πρόβλημα της διαμέρισης (partition) ανάγεται στο πρόβλημα ($P2||C_{max}$)

Λύση

Το πρόβλημα της διαμέρισης αναρωτιέται εάν υπάρχουν n ακέραιοι $\{a_1, a_2, ..., a_n\}$ τέτοιοι ώστε $\sum_i a_i$ =2b. Το πρόβλημα είναι να αποφανθούμε εάν υπάρχει υποσύνολο S τέτοιο ώστε $\sum_{a \in S} a_i$ =b

Εάν υπάρχει διαμέριση S η οποία αθροίζει στο b, τότε και οι υπόλοιποι αριθμοί αθροίζουν στο b.

Αυτό γίνεται εάν επιλέξουμε t = n όλα τα στοιχεία του P ώστε:

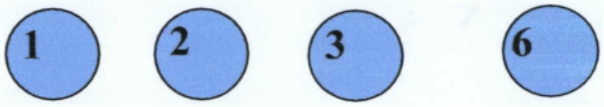
$$z = \frac{1}{2}\sum_{i=1}^{t} a_i = b.$$

Παράδειγμα Αναγωγής 4

Το πρόβλημα της διαμέρισης (partition) ανάγεται στο πρόβλημα του σακιδίου

Λύση

Έστω το σετ $N = \{1,2,3,6\}$. Μια λύση διαμέρισης είναι η $S = \{1,2,3\}$ και $N \setminus S = \emptyset$. Εάν τα βάρη κάθε αντικειμένου, για το πρόβλημα του σακιδίου, θεωρηθούν ότι είναι:



και οι αξίες των αντικειμένων $x_1 = x_2 = x_3 = 1$ και $x_4 = 0$ τότε μια λύση του προβλήματος του σακιδίου είναι η:

$$x_1 + 2x_2 + 3x_3 + 6x_4 \geq 6$$
$$x_1 + 2x_2 + 3x_3 + 6x_4 \leq 6$$

Διαφορετικά, το πρόβλημα της διαμέρισης μπορεί να αναχθεί στο πρόβλημα του

σακιδίου εάν n = t, με n το πλήθος των μη αρνητικών ακεραίων $\{a_1, a_2, ..., a_n\}$ και t το πλήθος των αντικειμένων που θα μπουν στον σάκο, $p_j = a_j$, με p_j το βάρος κάθε ενός αντικειμένου και $v_j = a_j$, με v_j την αξία κάθε αντικειμένου. Άρα:

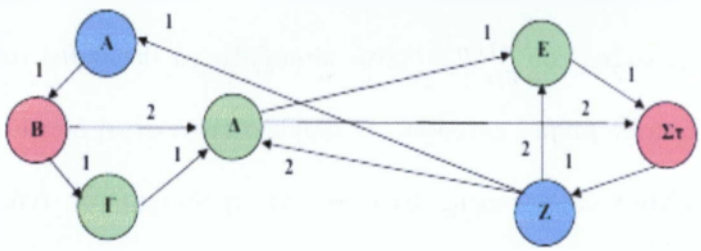
$$d = \frac{1}{2}\sum_{j=1}^{t} a_j = b,\ z = \frac{1}{2}\sum_{j=1}^{t} a_j = b.$$

Παράδειγμα Αναγωγής 5

Το πρόβλημα του Χαμιλτονιακού Κύκλου ανάγεται στο πρόβλημα του ταξιδεύοντος εμπόρου (TSP)

Λύση

Για δύο σημεία που συνδέονται μέσω μιας γραμμής ορίζουμε π.χ. απόσταση 1 ενώ για όλα τα άλλα σημεία ορίζουμε απόσταση π.χ. 2. Δηλαδή, κάθε ζεύγος γειτονικών σημείων έχει απόσταση 1 ενώ οι αποστάσεις μη γειτονικών σημείων είναι 2.



Η διαδρομή Α-Β-Γ-Δ-Ε-Στ-Ζ-Α είναι χαμιλτονιανός κύκλος (μια που περνάει από κάθε κόμβο μια φορά) αλλά και λύση του TSP μια που αντιστοιχεί στην μικρότερη διαδρομή (τοποθετήθηκαν ενδεικτικά μόνο τέσσερα βέλη με βάρος διαφορετικό της μονάδας (2). Θα μπορούσαμε να βάζαμε πολύ περισσότερα αλλά κάτι τέτοιο θα δυσκόλευε την ανάγνωση της εικόνας).

Επίλογος

Τα προβλήματα τα οποία μπορούν να λυθούν θεωρητικά αλλά πρακτικά παίρνουν πολλή ώρα προκειμένου να επιλυθούν, ονομάζονται δύσκολα. Στη θεωρία της πολυπλοκότητας, τα προβλήματα που δεν χρειάζονται πολυωνυμικό χρόνο για την εκτέλεσή τους είναι δύσκολα, εκτός από εξαιρετικά μικρές εισόδους. Σύμφωνα με τους Cobham-Edmonds, μόνο τα προβλήματα με πολυωνυμική χρονική πολυπλοκότητα μπορούν να λυθούν με μια πραγματική υπολογιστική μηχανή. Στα δύσκολα προβλήματα συγκαταλέγονται και όσα είναι εκθετικώς πολύπλοκα. Όπως είδαμε παραπάνω, για παράδειγμα, τα προβλήματα τύπου $O(2^n)$, για μικρές εισόδους, π.χ. n=100, και έστω ότι ο υπολογιστής κάνει 10^{12} (αντί για 10^6 που είχαμε προηγουμένως υποθέσει) βήματα το δευτερόλεπτο, το πρόγραμμα θα χρειαστεί κάπου στα 40 δισεκατομμύρια χρόνια για να τερματίσει, περίπου την ηλικία του σύμπαντος. Ακόμα και με πιο γρήγορους επεξεργαστές, πάλι το πρόβλημα δεν αλλάζει. Παραμένει δύσκολο.

Βέβαια, ακόμα και ο πολυωνυμικός χρόνος δεν είναι πάντα πρακτικός. Για παράδειγμα, πολυπλοκότητα της τάξης του $O(n^{10})$ είναι παράδοξο να θεωρηθεί αποτελεσματική ή εύκολη, εκτός από πολύ μικρές εισόδους. Η αλήθεια είναι ότι η έννοια της 'δυσκολίας' σε ένα πρόβλημα είναι κάπως ασαφής. Ένα, δηλαδή, πρόβλημα που ανήκει στην κλάση P δεν σημαίνει ότι μπορεί και να λύνεται γρήγορα. Ανάποδα, ένα πρόβλημα που ανήκει στην κλάση NP δεν σημαίνει ότι πάντα δεν λύνεται. Για παράδειγμα, το πρόβλημα SAT έχει λύσεις που βρίσκονται με αλγορίθμους που εκτελούνται σε λογικούς χρόνους, ακόμα και για σχετικά μεγάλα n.

Η θεωρία της πολυπλοκότητας είναι ένα πεδίο συνεχούς ανάπτυξης κι έρευνας. Το ερώτημα NP = P παραμένει άλυτο, ενώ τα τελευταία 40 χρόνια γίνονται συνεχείς

διερευνήσεις προβλημάτων που θα μπορούσαν να δώσουν απάντηση σε αυτό. Ας δούμε τι θα μας προσφέρει το μέλλον.

Βιβλιογραφία

1. S. Dasgupta, C. Papadimitriou, U. Vazirani, Αλγόριθμοι, Εκδόσεις Κλειδάριθμος, Αθήνα 2009.

2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Εισαγωγή στους αλγορίθμους Τόμος Ι, Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2006.

3. Michael Sipser (1997). Introduction to the Theory of Computation. PWS Publishing.

4. [Philippe Breton (1991). Histoire de l' Informatique. Editions La Decouverte, France]

5. Dreyfus H & Dreyfus S. (1984). Mindless machines: Computers don't think like experts and never will. The Sciences, Nov/Dec., 18-22.

6. David Gries (1989). The Science of Programming, Springer-Verlag

7. Huang et al. (1990). Introduction to Algorithms, McGraw Hill/The MIT Press

8. Κυρούσης, Παπαϊωάννου, Σταυρόπουλος (2008). Διδακτικές Σημειώσεις στα Ειδικά Θέματα Υπολογισμού και Πολυπλοκότητας

9. Endriss, U. (2012). *Complexity Theory Tutorial*, Computational Social Choice

10. [Fraenti, P., *Reducing Knapsack to TSP*, http://cs.uef.fi/pages/franti/asa/DAA-Knapsack-to-TSP.ppt]

11. [Dastidar, S.G (2003). *Complexity theory*, Lecture Notes]

12. Καραγιώργος Γ. Σημειώσεις 2010 – 2011, Τ.Ε.Ι Καλαμάτας

13. http://smithsonianchips.si.edu/augarten/p2.htm

14. https://fiftyexamples.readthedocs.org/en/latest/algorithms.html

15. http://cs-exhibitions.uni-klu.ac.at/index.php?id=389

16. http://en.wikipedia.org/wiki/Hilbert%27s_tenth_problem