ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΟΜΕΑΣ ΜΑΘΗΜΑΤΙΚΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

$Αυτοαναφορικές, διαγώνιες\ μέθοδοι\ και\ πλήρης\ μελέτη\ του\ σχετικοποιημένου$ $P^A = NP^A\ και\ P^B \neq NP^B$ $των\ Baker - Gill - Solovay$

$ΜΕΡΟΣ\ B'$

ΦΑΡΑΖΗ ΜΑΡΙΝΑ

Επιβλέπων : Αρβανιτάκης Αλέξανδρος
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2014

Πίνακας περιεχομένων

2. Το πρόβλημα $P\ vs. NP$……4

2.1 Εισαγωγή……...4

2.2 Μηχανές Turing……4

2.3 Γλώσσες και αλφάβητα……..8

2.3.1 Αναγνωρίσιμες γλώσσες και Μηχανές Turing……10

2.4 Υπολογιστική πολυπλοκότητα……14

2.4.1 Οι κλάσεις P και NP……...14

2.4.2 Παραδείγματα κλάσεων P και NP……..16

2.5 Το πρόβλημα $P\ vs. NP$……21

2.5.1 Ιστορική αναδρομή του προβλήματος……23

2.6 Χρονική πολυπλοκότητα……..25

2.6.1 Οι συμβολισμοί κεφαλαίου και πεζού όμικρον……..25

2.6.2 Κλάσεις χρονικής πολυπλοκότητας……28

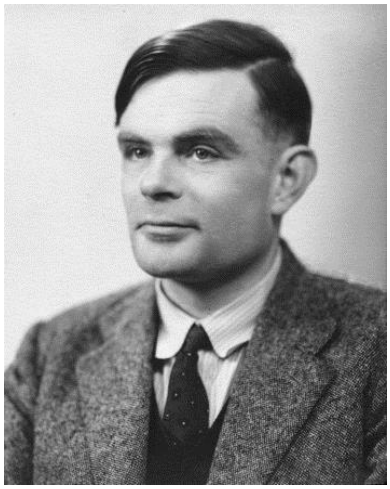
2.7 Χωρική πολυπλοκότητα……28

2.7.1 Κλάσεις χωρικής πολυπλοκότητας……29

2.7.2 Θεώρημα Savitch……....30

2.8 $P^A = NP^A$ και $P^B = NP^B$……33

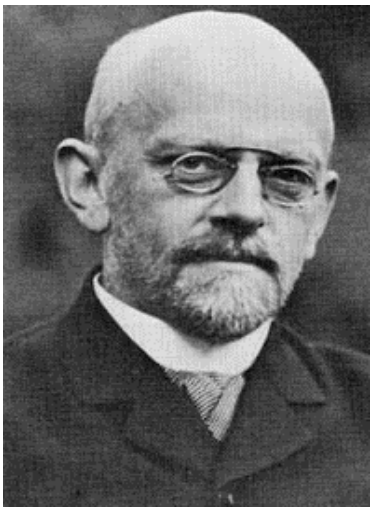
Βιβλιογραφικές αναφορές……40



Alan Turing

Kurt Gödel

John von Neumann



David Hilbert

ΤΟ ΠΡΟΒΛΗΜΑ $P\ vs.\ NP$

2.1 Εισαγωγή

Τι μπορούμε να υπολογίσουμε με τους υπολογιστές; Υπάρχουν προβλήματα που δεν μπορούν να λυθούν από υπολογιστή; Πόσο αποδοτικά μπορούμε να λύσουμε ένα αλγοριθμικό πρόβλημα; Τι είναι αυτό που κάνει κάποια προβλήματα υπολογιστικώς δύσκολα και κάποια άλλα εύκολα;

Η **Θεωρία Υπολογισμού** αφορά τη θεμελίωση της Επιστήμης της Πληροφορικής και πραγματεύεται το εάν και το πόσο αποδοτικά είναι δυνατόν να λυθεί κάποιο πρόβλημα με τη χρήση κάποιου αλγορίθμου σε έναν εξιδανικευμένο υπολογιστή, δηλαδή ένα υπολογιστικό μοντέλο. Το πεδίο διαιρείται σε δύο υποπεδία: τη **Θεωρία Υπολογισιμότητας** και τη **Θεωρία Πολυπλοκότητας**. Στην πρώτη, ο στόχος είναι να χαρακτηρίσουμε τα διάφορα προβλήματα ως επιλύσιμα ή ανεπίλυτα, ενώ η δεύτερη στοχεύει στην ταξινόμηση των επιλύσιμων προβλημάτων σε εύκολα και δύσκολα. Για να πραγματοποιήσουν μία ενδελεχή μελέτη της έννοιας του υπολογισμού, οι επιστήμονες αξιοποιούν διάφορες διατυπώσεις υπολογιστικών μοντέλων. Το απλούστερο δυνατό μοντέλο ονομάζεται πεπερασμένη μηχανή καταστάσεων ή πεπερασμένο αυτόματο και αποτελεί μία ικανοποιητική αναπαράσταση υπολογιστών.

2.2 Μηχανές Turing

Το 1936, ο Βρετανός Alan Turing (1912-1954) χαρακτηρίστηκε ως θεμελιωτής της επιστήμης των υπολογιστών. ‘Ορισε ένα μαθηματικό μοντέλο για μία υπολογιστική μηχανή γενικής χρήσης, η οποία ονομάστηκε μετά προς τιμή του Μηχανή Turing (Turing Machine). Επιπλέον, απάντησε αρνητικά στο τρίτο πρόβλημα του Hilbert (Entscheidungsproblem), αποδεικνύοντας ότι το πρόβλημα, εάν μία Μηχανή Turing τερματίζει για μία δοσμενη είσοδο, είναι μη αλγοριθμικά αποφασίσιμο. Με άλλα λόγια, δεν υπάρχει αλγόριθμος

(Μηχανή Turing) που να δέχεται ως είσοδο την περιγραφή της μηχανής και της εισόδου της και να αποφασίζει αν κάποτε θα σταματήσει η μηχανή. Αυτό είναι και το λεγόμενο πρόβλημα του τερματισμού (Halting Problem) που αναφέραμε στο προηγούμενο κεφάλαιο. Παρεκκλίνοντας λίγο, στην πραγματικότητα, η απόδειξη του ήταν μία απλούστερη έκδοση της απόδειξης μη πληρότητας του Gödel. Επίσης, πέρα από το ζήτημα της επιλυσιμότητας ενός προβλήματος (με το να επιδείξουμε έναν αλγόριθμο ή μηχανή Turing, που να το επιλύει) δόθηκε ιδιαίτερη έμφαση στον χρόνο επίλυσης.

Η συνεισφορά του Turing, είχε μεγάλη επίδραση στη μετέπειτα πορεία της επιστήμης των υπολογιστών. Πρόσφερε μία νέα και βαθυστόχαστη ενόραση στην αρχαία τέχνη του υπολογισμού. Σχεδόν κάθε εργασία που μπορούμε να φανταστούμε που περιλαμβάνει σύμβολα, αριθμούς ή κείμενο, είτε είναι ήδη εντός της δυνατότητας των υπολογιστών είτε κάποιος ειδικός επιμένει ότι σύντομα θα είναι.

Η **Μηχανή Turing** (Turing Machine, για συντομία TM) είναι μία λογική υπολογιστική μηχανή, η εξέλιξη των αυτομάτων και το πιο σύνηθες μοντέλου υπολογισμού, αφού διαθέτει άπειρη μνήμη (την ταινία εισόδου-εξόδου), ενώ τα πεπερασμένα αυτόματα και τα αυτόματα στοίβας έχουν σοβαρούς περιορισμούς μνήμης. Ο ορισμός της εισήχθη από τον Άγγλο μαθηματικό, «πατέρα της επιστήμης υπολογιστών» Alan M. Turing το 1936. Με απλά λόγια, οι μηχανές αυτές λειτουργούν όπως ένας άνθρωπος που εκτελεί αριθμητικούς υπολογισμούς ακολουθώντας συγκεκριμένες οδηγίες, έχοντας στη διάθεσή του απεριόριστη ποσότητα χαρτιού, μολύβι, σβήστρα και απεριόριστο χρόνο. Η μηχανές του δηλαδή αποτελούν μία μίμηση του ανθρώπου. Διαισθητικά, μπορεί κανείς να τη φανταστεί σαν έναν προσωπικό υπολογιστή με άπειρη μνήμη που μπορεί να προσπελάζει χωρίς περιορισμούς. Ένα πιστό μοντέλο υπολογιστή γενικής χρήσης.

Η μηχανή Turing έχει απλή μαθηματική διατύπωση, μπορεί να αναλυθεί, να χρησιμοποιηθεί σε αποδείξεις, ενώ κατά πολλούς αντιπροσωπεύει το ισχυρότερο εφικτό μοντέλο υπολογισμού. Αυτό σημαίνει ότι ένα πρόβλημα δε μπορεί να λυθεί αλγοριθμικά αν και μόνο αν δεν υπάρχει μηχανή Turing που να το επιλύει. Ένα διάσημο μη επιλύσιμο πρόβλημα είναι το πρόβλημα του τερματισμού (η εύρεση μίας Μηχανής Τούρινγκ που αποφασίζει αν μία άλλη Μηχανή Τούρινγκ θα τερματίσει με συγκεκριμένη είσοδο ή θα πέσει σε ατέρμονα βρόχο), το οποίο χρησιμεύει ώστε να ανάγονται άλλα προβλήματα σε αυτό και να αποδεικνύεται έτσι ότι είναι μη επιλύσιμα.

Αν και η άπειρη μνήμη μπορεί να θεωρηθεί ανέπιτευκτη, στην ουσία κάθε πρόβλημα που πραγματεύεται η μηχανή Turing χρειάζεται πάντα πεπερασμένη μνήμη, άρα κάθε πρόβλημα που μπορεί να λυθεί από μία μηχανή Turing θα μπορούσε να λυθεί από έναν υπολογιστή με επαρκή μνήμη. Παρ'όλα αυτά, υπάρχουν προβλήματα που ούτε αυτή η μηχανή μπορεί να επιλύσει. Χρησιμοποιεί ως απεριόριστη μνήμη μία άπειρη ταινία. Διαθέτει μία κεφαλή η οποία έχει τη δυνατότητα, μετακινούμενη επάνω στην ταινία να διαβάζει και να γράφει σύμβολα.

Αρχικά, η ταινία περιέχει μόνο τη λέξη εισόδου και σύμβολα διαστήματος σε όλες τις υπόλοιπες θέσεις. Για να αποθηκεύσει κάποιες πληροφορίες, η μηχανή γράφει απλώς αυτές τις πληροφορίες σε κάποιο τμήμα της ταινίας. Για να διαβάσει κάποιες εγγεγραμμένες πληροφορίες, μετακινεί την κεφαλή στο αντίστοιχο τμήμα της ταινίας.

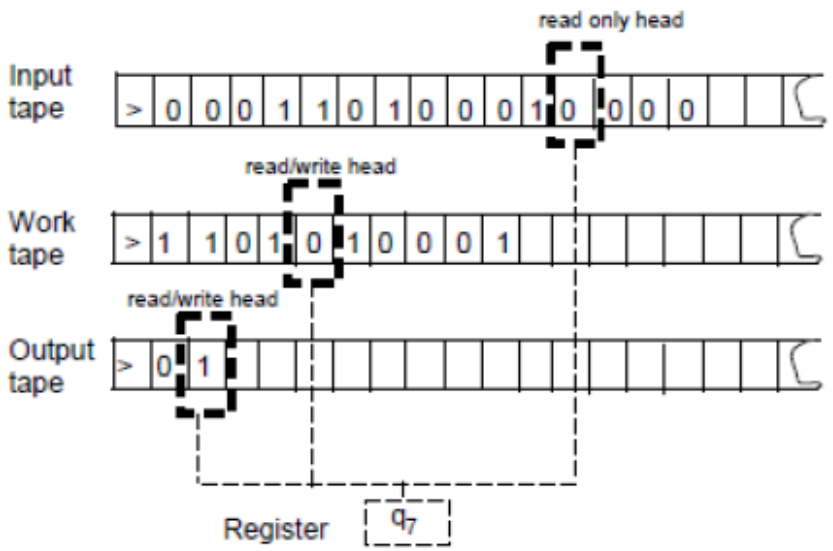
Η μηχανή συνεχίζει τον υπολογισμό μέχρις ότου αποφασίσει να παραγάγει μία έξοδο. Οι μοναδικές πιθανές έξοδοι είναι αποδοχή και απόρριψη, τις οποίες η μηχανή παράγει μεταβαίνοντας σε αντίστοιχες προκαθορισμένες καταστάσεις αποδοχής και απόρριψης. Εάν δεν μεταβεί ποτέ σε κάποια από αυτές τις καταστάσεις, συνεχίζει τη λειτουργία της επ΄άπειρον, χωρίς να τερματίζει ποτέ.

ΟΡΙΣΜΟΣ 2.1. Τυπικός ορισμός της μηχανής Turing

Για να διατυπώσουμε τον ορισμό της μηχανής Turing θα χρησιμοποιήσουμε τη συνάρτηση μεταβάσεων. Ο σκοπός αυτής της συνάρτησης είναι να περιγράψει τον τρόπο που μεταβαίνει η μηχανή Turing από το ένα βήμα στο επόμενο. Η συνάρτηση αυτή ορίζεται ως εξής: $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{A, \Delta\}$, όπου το A συμβολίζει την κίνηση της κεφαλής μετά την εγγραφή προς τα αριστερά ενώ με Δ την κίνηση προς τα δεξιά.

Η μηχανή Turing είναι μία εφτάδα στοιχείων $(Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{\alpha\pi o\delta o\chi\acute{\eta}\varsigma}, q_{\alpha\pi\acute{o}\rho\rho\iota\psi\eta\varsigma})$, όπου τα $Q,\ \Sigma$ και Γ είναι πεπερασμένα σύνολα.

- Το Q είναι το σύνολο καταστάσεων
- Το Σ είναι το αλφάβητο εισόδου που δεν περιέχει το σύμβολο διαστήματος ⊔
- Το Γ είναι το αλφάβητο ταινίας, το οποίο περιέχει το σύμβολο διαστήματος ⊔ και $\Sigma \subseteq \Gamma$
- Το $\delta: Q\ \times \Gamma\ \longrightarrow Q\ \times \Gamma\ \times \{A, \Delta\}$ είναι η συνάρτηση μεταβάσεων
- Το σύμβολο $q_0 \in Q$ είναι η αρχική κατάσταση
- Το $q_{\alpha\pi o\delta o\chi\acute{\eta}\varsigma}\ \in Q$ είναι η κατάσταση αποδοχής
- Το $q_{\alpha\pi\acute{o}\rho\rho\iota\psi\eta\varsigma}\ \in Q$ είναι η κατάσταση απόρριψης και ισχύει ότι $q_{\alpha\pi\acute{o}\rho\rho\iota\psi\eta\varsigma}\ \neq\ q_{\alpha\pi o\delta o\chi\acute{\eta}\varsigma}$.



Για παράδειγμα αν q είναι η τρέχουσα κατάσταση της μηχανής, αν συμβολιστεί με α η τρέχουσα θέση της κεφαλής και αν οριστεί η συνάρτηση μεταβάσεων ως $\delta\ (q, \alpha) = (r, b, A)$ τότε η λειτουργία της μηχανής είναι η εξής: Αντικαθίσταται στην ταινία το σύμβολο α με το b, μεταβαίνει στην κατάσταση r και αφού υπάρχει το σύμβολο A τότε μετά την εγγραφή του b η κεφαλή θα κινηθεί προς τα αριστερά.

Τόσο τα αυτόματα όσο και οι Μηχανές Turing διαθέτουν ένα ισχυρό αλλά μη ρεαλιστικό μαθηματικό χαρακτηριστικό: το μη ντετερμινισμό, τη δυνατότητα σε κάθε βήμα της λειτουργίας τους να μαντεύουν τη σωστή διαδρομή από ένα πλήθος δυνατών διαδρομών. Μία πολυταινιακή μηχανή Turing δεν είναι παρά μία συνήθης μηχανή Turing με περισσότερες από μία ταινίες.

ΟΡΙΣΜΟΣ 2.2 Μία **αναιτιοκρατική μηχανή Turing** (nondeterministic Turing machine, για συντομία NTM) ορίζεται ως εξής: σε οποιοδήποτε σημείο ενός υπολογισμού, η μηχανή ενδέχεται να έχει περισσότερες απο μία δυνατότητες για τη συνέχεια. Κατά συνέπεια, ο υπολογισμός μίας τέτοιας μηχανής σχηματίζει ένα δένδρο, του οποίου οι κλάδοι αντιστοιχούν στις διαφορετικές δυνατότητες. Αντίθετα, μία **αιτιοκρατική μηχανή Turing** όταν βρίσκεται σε μία δεδομένη κατάσταση, έχει μία δυνατή επιλογή συμπεριφοράς.

Μία μηχανή Turing που μπορεί να προσομοιώσει μία οποιαδήποτε άλλη μηχανή Turing ονομάζεται **Καθολική Μηχανή Turing**. Η ύπαρξη της, οδήγησε σε ανεπίλυτα προβλήματα, προβλήματα για τα οποία αποδεδειγμένα δεν υπάρχει αλγόριθμος που να τα επιλύει. Ο λόγος είναι οτι υπάρχουν περισσότερες γλώσσες (προβλήματα) από τις μηχανές Turing (αλγόριθμοι) που τις αποφασίζουν. Τα προβλήματα αυτά καλούνται μη επιλύσιμα ή μη αποκρίσιμα αν πρόκειται για προβλήματα απόφασης και οι γλώσσες που αντιστοιχούν σε αυτά είναι μη διαγνώσιμες ή αλλιώς μη αναδρομικές.

2.3 Γλώσσες και αλφάβητα

ΟΡΙΣΜΟΣ 2.3. Ορίζουμε ως **αλφάβητο**, οποιοδήποτε μη κενό πεπερασμένο σύνολο και ως σύμβολα του αλφαβήτου τα μέλη του συνόλου αυτού.

ΟΡΙΣΜΟΣ 2.4. **Λέξη επί ενός αλφαβήτου** είναι οποιαδήποτε πεπερασμένη ακολουθία συμβόλων του αλφαβήτου αυτού. Για παράδειγμα, θεωρούμε ένα αλφάβητο $\Sigma_1 = \{0,1\}$. Η ακολουθία 01001 είναι μια λέξη επί του Σ_1.

ΟΡΙΣΜΟΣ 2.5. Το **μήκος μίας λέξης** w επί ενός αλφαβήτου Σ είναι το πλήθος των συμβόλων που περιέχει και συμβολίζεται με $|w|$. Η λέξη μηδενικού μήκους είναι η λεγόμενη κενή λέξη και συμβολίζεται με ε.

ΟΡΙΣΜΟΣ 2.6. Η **λεξικογραφική διάταξη λέξεων** είναι η συνήθης διάταξη που ακολουθούν τα λεξικά, με μοναδική διαφορά ότι οι βραχύτερες λέξεις προηγούνται των μακρύτερων. Συμπεραίνουμε ότι η λεξικογραφική διάταξη όλων των λεξών επί του αλφαβήτου $\{0,1\}$ είναι η ακόλουθη : $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$.

Οι γλώσσες κωδικοποιούν προβλήματα. Όμως κάθε γλώσσα $L \subseteq \Sigma^*$ μπορεί επίσης να θεωρηθεί ως πρόβλημα:

ΤΟ ΠΡΟΒΛΗΜΑ ΑΠΟΦΑΣΗΣ ΤΗΣ L: Δεδομένης μίας συμβολοσειράς $x \in \Sigma^*$, ισχύει ότι $x \in L$;

Είναι χρήσιμο να θεωρούμε ένα πρόβλημα και την αντίστοιχη γλώσσα ως το ίδιο πράγμα. Οι γλώσσες μπορούν να χρησιμοποιηθούν για την κωδικοποίηση σημαντικών υπολογιστικών προβλημάτων όλων των ειδών. Είναι καταλληλότερες σε σχέση με τις μηχανές Turing, ενώ τα προβλήματα είναι σαφέστερες διατυπώσεις πρακτικών υπολογιστικών προβλημάτων, για τα οποία πρέπει να αναπτύξουμε αλγόριθμους.

ΟΡΙΣΜΟΣ 2.7. Μία γλώσσα λέγεται (κατα Turing) **αναγνωρίσιμη** εάν υπάρχει μηχανή Turing που να την αναγνωρίζει. (Σε πολλά εγχειρίδια, οι αναγνωρίσιμες γλώσσες αποκαλούνται αναδρομικά απαριθμητές.)

ΟΡΙΣΜΟΣ 2.8. Μία γλώσσα λέγεται (κατά Turing) **διαγνώσιμη** εάν υπάρχει μηχανή Turing που να τη διαγιγνώσκει. (Σε πολλά εγχειρίδια, οι διαγνώσιμες γλώσσες αποκαλούνται αναδρομικές.)

Αν μία γλωσσα L είναι διαγνώσιμη τότε είναι και αναγνωρίσιμη. Το αντίθετο δεν ισχύει.

2.3.1 Αναγνωρίσιμες γλώσσες και μηχανές Turing

ΟΡΙΣΜΟΣ 2.9. Ένα σύνολο X λέγεται **αριθμήσιμο** εάν είναι πεπερασμένο ή ισομέγεθες με το $\mathbb{N}$ δηλαδή θα ισχύουν τα ακόλουθα:

- Θα πρέπει να υπάρχει μία απεικόνιση $f : \mathbb{N} \longrightarrow X$ που να είναι επί
- Θα πρέπει να υπάρχει μία απεικόνιση $g : X \longrightarrow \mathbb{N}$ που να είναι ένα-προς-ένα.

Θεώρημα 2.1. Οι Μηχανές Turing είναι αριθμήσιμες.

Απόδειξη

Ως βοήθεια στην κατανόηση της ζητούμενης απόδειξης, θα αποδείξουμε πρώτα ότι οι πεπερασμένες λέξεις είναι αριθμήσιμες. Θεωρούμε Σ ένα αλφάβητο, τότε το σύνολο όλων των πεπερασμένων λέξεων που μπορούμε να σχηματίσουμε από το Σ είναι αριθμήσιμο. Παραδείγματος χάριν, αν $\Sigma = \{0,1\}$, τότε μπορούμε να διατάξουμε λεξικογραφικά όλες τις πεπερασμένες λέξεις

$$\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, \ldots$$

όπου με αυτό τον τρόπο εύκολα προκύπτει η “1-1” και “επί” αντιστοίχηση με τους φυσικούς αριθμούς

$$1, 2, 3, 4, 5, 6, 7, 8, \ldots$$

Οι μηχανές Turing περιγράφονται ως λέξεις πεπερασμένου μήκους. Μία μηχανή Turing M αποτελείται από πεπερασμένα σύνολα συμβόλων. Αυτό σημαίνει ότι μπορούμε να κωδικοποιήσουμε τη M ως μία πεπερασμένη λέξη $w_M \in \Sigma^*$. Το σύνολο όλων των πεπερασμένων λέξεων που μπορούν να σχηματίσουν ένα αλφάβητο είναι αριθμήσιμο, δηλαδή μπορούν να μπουν σε λεξικογραφική σειρά

$$w_1, w_2, w_3, w_4, \ldots \tag{1}$$

Προφανώς, η $\boldsymbol{i}$-οστή λέξη w_i αυτής της σειράς μπορεί να μην κωδικοποιεί καμία μηχανή Turing. Όμως, διατρέχοντας τις λέξεις της λεξικογραφικής διατάξεως (1) από αριστερά προς τα δεξιά, συναντάμε την 1η λέξη w_{M_1} που κωδικοποιεί μία μηχανή Turing M_1, μετά την δεύτερη λέξη w_{M_2} που κωδικοποιεί μία μηχανή Turing M_2, κ.τ.λ. Με αυτό τον τρόπο προκύπτει η λεξικογραφικά διατεταγμένη υπακολουθία της (1)

$$w_{M_1}, w_{M_2}, w_{M_3}, w_{M_4}, \ldots \qquad (2)$$

αποτελούμενη μόνο από τις λέξεις που κωδικοποιούν μηχανές Turing. Αυτό σημαίνει ότι και το σύνολο όλων των μηχανών Turing είναι αριθμήσιμο, διότι η (2) είναι σε αντιστοιχία με τους φυσικούς αριθμούς. □

Θεώρημα 2.2 Το σύνολο όλων των γλωσσών είναι μη αριθμήσιμο.

Απόδειξη

Ορίζουμε μία γλώσσα σε ένα πεπερασμένο αλφάβητο Σ. Έστω Σ^* το σύνολο όλων των λέξεων (πεπερασμένου μήκους και μη) που μπορούν να σχηματιστούν από το αλφάβητο Σ. Μία γλώσσα L είναι ένα υποσύνολο του Σ^*, δηλαδή $L \subseteq \Sigma^*$. Για την καλύτερη κατανόηση του συνόλου όλων των γλωσσών που μπορούμε να κατασκευάσουμε από ένα δοσμένο αλφάβητο Σ, αρχικά δείχνουμε το παρακάτω.

Το σύνολο όλων των αναγνωρίσιμων γλωσσών είναι αριθμήσιμο. Έστω μία μηχανή Turing M. Η γλώσσα $L(M)$ της μηχανής M αποτελείται απο όλες τις λέξεις w $\in \Sigma^*$ για τις οποίες $M(w) = NAI$. Κάθε αναγνωρίσιμη γλώσσα $L \subseteq \Sigma^*$, έχει τουλάχιστον μία μηχανή Turing M_L ώστε για κάθε λέξη $w \in L$ η μηχανή M_L τερματίζει σε κατάσταση NAI. Αντιλαμβανόμαστε λοιπόν ότι για κάθε αναγνωρίσιμη γλώσσα $L \subseteq \Sigma^*$, αν είναι η μηχανή Turing M_L που την αναγνωρίζει, τότε ισχύει

$$L(M_L) \equiv L \ .$$

Με αυτό τον τρόπο, το σύνολο $ACCEPT$ όλων των αναγνωρίσιμων γλωσσών μπορεί να γραφτεί αυστηρά ως $ACCEPT = \{L \subseteq \Sigma^* \mid \exists\, M \text{ ώστε } L(M) = L\}$.
Παρατηρούμε ότι από την αρίθμηση (2) όλων των μηχανών Turing προκύπτει η αρίθμηση όλων των αναγνωρίσιμων γλωσσών

$$L(M_1), L(M_2), L(M_3), L(M_4), \ldots = L_{M_1}, L_{M_2}, L_{M_3}, L_{M_4}, \ldots \qquad (3)$$

Αποδείξαμε ότι όλες οι λέξεις πεπερασμένου μήκους που μπορούν να σχηματιστούν από ένα πεπερασμένο αλφάβητο Σ είναι αριθμήσιμες, αφού μπορούν να μπουν σε σειρά όπως οι λέξεις ενός λεξικού. Επιπλέον το σύνολο όλων των μηχανών Turing είναι αριθμήσιμο. Συμπεραίνουμε ότι επειδή κάθε μία Turing M ορίζει μονοσήμαντα και τη γλώσσα L_M που αναγνωρίζει (ενώ κάθε μία αναγνωρίσιμη γλώσσα L δεν ορίζει μονοσήμαντα ακριβώς μία μηχανή M_L που αναγνωρίζει την L διότι δύο ή περισσότερες διαφορετικές μηχανές μπορεί να αναγνωρίζουν την ίδια γλώσσα), καταλήγουμε ότι και το σύνολο όλων των αναγνωρίσιμων γλωσσών είναι αριθμήσιμο.

Όμως, το σύνολο όλων των γλωσσών είναι μη αριθμήσιμο. Άρα υπάρχουν και άλλες γλώσσες, εκτός από τις αναγνωρίσιμες, οι οποίες καθιστούν το σύνολο όλων των γλωσσών του Σ^* μη αριθμήσιμο. Θα αποδείξουμε ότι **το σύνολο όλων των γλωσσών είναι μη αριθμήσιμο** με τη μέθοδο της Διαγωνιοποίησης.

Θεωρούμε την αρίθμηση (1) όλων των πεπερασμένων λέξεων που μπορούν να σχηματιστούν από ένα αλφάβητο Σ. Η αρίθμηση αυτή έχει αριστερή αρχή, αλλά δεν έχει δεξιό πεπερασμένο τέλος. Έστω οποιαδήποτε γλώσσα $L \subseteq \Sigma^*$. Η γλώσσα L είναι ένα πεπερασμένο ή άπειρο σύνολο λέξεων του Σ^*.

Οι λέξεις της L μπορούν να παρασταθούν ως μία απείρου μήκους δυαδική ακολουθία συμβόλων 0,1. Δηλαδή

$$L \longrightarrow 00101110 \ldots, \qquad (4)$$

όπου στην i-θέση υπάρχει το ψηφίο 1 αν η i-στη λέξη που εμφανίζεται στι λεξικογραφική σειρά (1) είναι λέξη της γλώσσας L, αλλιώς εμφανίζεται το ψηφίο 0. Στην ουσία, κάθε

γλώσσα L την κωδικοποιούμε με μοναδικό τρόπο με τη μορφή μίας απείρου μήκους δυαδικής λέξεως.

Άν δείξουμε ότι το σύνολο όλων των απείρων δυαδικών ακολουθιών είναι αριθμήσιμο, τότε θα έχουμε αποδείξει ότι το σύνολο όλων των γλωσσών είναι αριθμήσιμο. Για να καταλήξουμε σε άτοπο, υποθέτουμε ότι το σύνολο των απείρων δυαδικών ακολουθιών της μορφής (4) είναι αριθμήσιμο, δηλαδή έστω μία αντιστοίχηση του με τους φυσικούς αριθμούς της μορφής

$$b_1, b_2, b_3, b_4, \ldots, \tag{5}$$

όπου κάθε b_i είναι μία δυαδική ακολουθία της μορφής (4). Τότε μπορούμε να κατασκευάσουμε μία νέα δυαδική ακολουθία που δεν εμφανίζεται στη σειρά (5), της μορφής

D = $d_1 d_2 d_3 \ldots d_i \ldots$, όπου κάθε $d_i \in \{0, 1\}$.

Το d_i είναι το αντίθετο ψηφίο του i-οστού ψηφίου της δυαδικής ακολουθίας b_i που εμφανίζεται στη σειρά (5). Προφανώς, η D δεν ταυτίζεται με καμία δυαδική ακολουθία της αντιστοίχησης με τους φυσικούς αριθμούς (5) (που υποθέσαμε ότι υπάρχει), αφού εκ κατασκευής διαφέρει από κάθε b_i τουλάχιστον στο i-οστό ψηφίο. Καταλήγουμε σε άτοπο, αφού η υπόθεση μας ήταν ότι η (5) περιέχει όλες τις δυαδικές λέξεις. □

2.4 Υπολογιστική Πολυπλοκότητα

2.4.1 Οι κλάσεις P και NP

Κάποια προβλήματα ενώ είναι επιλύσιμα και μπορούν να αντιμετωπιστούν υπολογιστικά, δεν έχει βρεθεί μέχρι στιγμής αλγόριθμος που να τα επιλύει σε λογικά χρονικά πλαίσια. Θεωρούνται πρακτικά ανεπίλυτα, αφού απαιτούν εξωπραγματικό χρόνο ή υπερβολικά πολύ μνήμη. Από τη θεωρία πολυπλοκότητας γνωρίζουμε ότι:

ΟΡΙΣΜΟΣ 2.10. **P** ονομάζεται το σύνολο των γλωσσών που αναπαριστούν υπολογιστικά προβλήματα, για τα οποία υπάρχει ντετερμινιστική Μηχανή Τούρινγκ που τα επιλύει σε πολυωνυμικό χρόνο.

ΟΡΙΣΜΟΣ 2.11 **NP** ονομάζεται το σύνολο των γλωσσών που αναπαριστούν υπολογιστικά προβλήματα για τα οποία υπάρχει μη ντετερμινιστική Μηχανή Τούρινγκ που τα επιλύει σε πολυωνυμικό χρόνο αλλά όχι ντετερμινιστική (οι ισοδύναμες ντετερμινιστικές Μηχανές Τούρινγκ έχουν αυξημένη εκθετική πολυπλοκότητα).

Στην κλάση P ανήκουν τα προβλήματα για τα οποία έχει ήδη βρεθεί κάποιος γρήγορος αλγόριθμος που για στιγμιότυπα μεγέθους n απαιτούνται το πολύ n^2, n^3 ή γενικά n^κ βήματα, για κάποιο σταθερό αριθμό κ. Η κλάση NP περιλαμβάνει προβλήματα που απαιτούν αριθμό βημάτων της μορφής 2^n, 3^n ή ακόμα και n^n, δηλαδή εκθετικά πολλά βήματα. Στην πράξη έχει μεγάλη σημασία στην κρυπτογραφία γιατί οι αλγόριθμοι κρυπτογράφησης είναι NP πολυπλοκότητας, το οποίο σημαίνει ότι όσο δεν μπορεί κανείς εύκολα να φτιάξει έναν αλγόριθμο που να σπάει αμέσως τον κώδικα, τα συστήματα παραμένουν πιο ασφαλή για περισσότερο χρονικό διάστημα.

Συμπεραίνοντας, τα προβλήματα τα οποία μπορούν να επιλυθούν σε πολυωνυμικό χρόνο, ανήκουν στην κλάση P. Στην NP ανήκουν προβλήματα τα οποία επιλύονται “δύσκολα” αλλά η απάντηση τους μπορεί να ελεχθεί σε πολυωνυμικό χρόνο.

Θα αναφέρουμε ένα πρόβλημα που δεν ανήκει στην κλάση NP . Έστω ότι είχαμε δύο παίκτες στο σκάκι που ο καθένας έπαιζε, επιλέγοντας κάθε φορά την καλύτερη δυνατή κίνηση. Έπαιζε δηλαδή με βέλτιστο τρόπο. Ποιός θα νικούσε; Τα λευκά, τα μαύρα ή θα έρχονταν σε ισοπαλία; Κανείς δεν γνωρίζει την απάντηση. Κατ'αρχήν θα μπορούσαμε να βρούμε την απάντηση ελέγχοντας το δέντρο όλων των πιθανών παιχνιδιών που θα μπορούσαν να παίξουν. Αυτός όμως είναι ένας αστρονομικά μεγάλος αριθμός και θα χρειαζόταν κανείς υπερβολικά πολύ χρόνο ώστε να το εξετάσει, γεγονός πρακτικά ανέφικτο. Συνεπώς, έχουμε ένα πρόβλημα που επιλύεται δύσκολα και η απάντηση δεν μπορεί να ελεχθεί σε πολυωνυμικό χρόνο.

Παρατηρούμε τη διαφορά μεταξύ ενός τυπικού πολυωνύμου όπως το n^3 και μίας τυπικής εκθετικής συνάρτησης, όπως η 2^n. Για παράδειγμα, έστω ότι το n, το οποίο αναπαριστά το μήκος εισόδου, ισούται με 1000. Τότε, το n^3 ισούται με ένα δισεκατομμύριο. Αντιθέτως, το 2^n ισούται με έναν αριθμό εξωφρενικά μεγάλο, μεγαλύτερος ακόμη και από το πλήθος των ατόμων του σύμπαντος. Συνεπώς, οι αλγόριθμοι πολυωνυμικού χρόνου είναι αρκετά ταχείς και χρήσιμοι, ενώ οι αλγόριθμοι εκθετικού χρόνου είναι τόσο βραδείς που σπανίως είναι χρήσιμοι. Αυτοί οι αλγόριθμοι προκύπτουν όταν προσπαθούμε να βρούμε τη λύση ενός προβλήματος ερευνώντας όλες τις περιπτώσεις μέχρι να φτάσουμε στη σωστή. Η μέθοδος αυτή ονομάζεται «εξαντλητική αναζήτηση».

Για παράδειγμα, η πρόσθεση, η αφαίρεση και ο πολλαπλασιασμός αποτελούν παραδείγματα προβλημάτων που ανήκουν στην κλάση P. Αντίθετα, για δοθέν ακέραιο n, η εύρεση της παραγοντοποίησης του σε πρώτους παράγοντες, είναι ένα πρόβλημα που ανήκει στην κλάση NP. Όταν αναλύουμε έναν αλγόριθμο για να δείξουμε ότι εκτελείται σε πολυωνυμικό χρόνο, πρέπει πρώτα να προσδιορίσουμε ένα πολυωνυμικό άνω φράγμα για το πλήθος των σταδίων από τα οποία διέρχεται ο αλγόριθμος όταν δέχεται μία είσοδο μήκους n. Έπειτα, θα πρέπει να εξετάσουμε το κάθε μεμονωμένο στάδιο της περιγραφής του αλγορίθμου ώστε να βεβαιωθούμε ότι μπορεί να υλοποιηθεί σε πολυωνυμικό χρόνο με κάποιο αιτιοκρατικό υπολογιστικό μοντέλο.

2.4.2 Παραδείγματα κλάσεων P και NP

Θα μελετήσουμε ενδιαφέροντα παραδείγματα προβλημάτων, τα οποία είναι γνωστό ότι ανήκουν στην κλάση P ενώ θα αντιπαραβάλλουμε προβλήματα, τα οποία δε φαίνεται να ανήκουν στην P. Αναφέρουμε ότι για να δείξουμε ότι ένα πρόβλημα ανήκει στην P μπορούμε να χρησιμοποιήσουμε το εδραιωμένο γεγονός ότι ένα άλλο πρόβλημα ανήκει στην P, δηλαδή με την αναγωγή του σε ένα ήδη λυμένο πρόβλημα.

Ένα γνωστό πρόβλημα που ανήκει στην κλάση P είναι η εύρεση του μέγιστου κοινού διαιρέτη δύο ακαιρέων. Για την επίλυση του συνήθως χρησιμοποιούμε τον αλγόριθμο του Ευκλείδη. Ο αριθμός των βημάτων που χρειάζονται για να βρούμε την απάντηση είναι πολυωνυμικός ως προς την είσοδο. Ένα άλλο πρόβλημα που ανήκει στο P, είναι το να αποφασίσουμε αν ένας θετικός ακέραιος αριθμός είναι πρώτος.

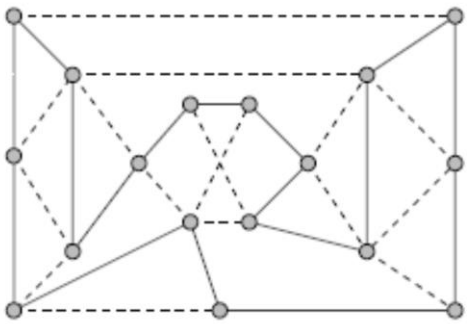
Πρόβλημα Reachability: Δεδομένων ενός κατευθυνόμενου γραφήματος $G \subseteq V \times V$, όπου $V = \{v_1, \dots, v_n\}$ είναι ένα πεπερασμένο σύνολο, δύο κόμβων $v_i, v_j \in V$, υπάρχει μονοπάτι από τον v_i στον v_j;
Το σύνολο εισόδων είναι το σύνολο όλων των τριάδων (G, v_i, v_j), όπου G είναι ένα πεπερασμένο γράφημα και v_i, v_j είναι δύο κόμβοι του G. Το ερώτημα που τίθεται είναι εάν υπάρχει μονοπάτι απο τον v_i στον v_j στο G. Χωρίς να επεκταθούμε, αναφέρουμε ότι μπορεί να επιλυθεί σε πολυωνυμικό χρόνο, άρα ανήκει στην κλάση P.

Πρόβλημα Κύκλου Euler: Δεδομένου ενός γραφήματος G, υπάρχει κλειστό μονοπάτι στο G, το οποίο χρησιμοποιεί κάθε ακμή του G ακριβώς μία φορά;
Μπορούμε να το διατυπώσουμε σε μορφή γλώσσας:
$L = \{k(G)\}: \tau o\ G\ \epsilon \acute{\iota} \nu \alpha \iota\ \acute{\epsilon} \nu \alpha\ \gamma \rho \acute{\alpha} \varphi \eta \mu \alpha\ Euler\ (= \acute{\epsilon} \chi \epsilon \iota\ \acute{\epsilon} \nu \alpha\ \kappa \acute{\upsilon} \kappa \lambda o\ Euler)\}$

Ένα γράφημα που περιέχει ένα τέτοιο μονοπάτι ονομάζεται γράφημα Euler ή μοναδικής διάσχισης. Δεν είναι δύσκολο να δούμε ότι η L ανήκει στο P γιατί υπάρχει ένας χαρακτηρισμός της μορφής «ο G είναι γράφος Euler» εάν και μόνο αν ισχύει κάτι, το οποίο μπορεί να ελεγχθεί σε πολυωνυμικό χρόνο.

Πρόβλημα Κύκλου Hamilton: Δεδομένου ενός γραφήματος G, υπάρχει ένας κύκλος που να περνά από κάθε κορυφή του G ακριβώς μια φορά;

Ένας τέτοιος κύκλος ονομάζεται κύκλος Hamilton και ένα γράφημα που τον περιέχει, ονομάζεται γράφημα Hamilton. Τώρα είναι οι κόμβοι , όχι οι ακμές, τους οποίους πρέπει να διασχίσουμε ακριβώς μία φορά, χωρίς να χρειάζεται να διασχίσουμε όλες τις ακμές. Παρά τη φαινομενική ομοιότητα των δύο προηγούμενων προβλημάτων, υπάρχει μία σημαντική διαφορά. Μετά από ενάμιση αιώνα εξονυχιστικής έρευνας, κανένας δεν κατάφερε να ανακαλύψει έναν πολυωνυμικό αλγόριθμο για το πρόβλημα κύκλος Hamilton.



Πρόβλημα Περιοδεύοντος Πωλητή: Δεδομένων ενός ακεραίου $n \geq 2$, ενός $n \times n$ πίνακα αποστάσεων $d_{i,j}$ και ενός ακεραίου $B \geq 0$ -διαισθητικά, ο προυπολογισμός (budget) του περιοδεύοντος πωλητή- βρείτε μία μετάθεση π του $\{1,2,\dots,n\}$ τέτοια ώστε $c(\pi) \leq B$.

Δηλαδή, δίνεται ένα σύνολο $\{c_1, c_2, \dots, c_n\}$ από πόλεις και ένας $n \times n$ πίνακας από μη αρνητικούς ακεραίους d, όπου ο $d_{i,j}$ συμβολίζει την απόσταση ανάμεσα από την πόλη c_i και c_j. Υποθέτουμε ότι $d_{ii} = 0$ και $d_{ij} = d_{ji}$ για όλα τα i, j. Μας ζητείται να βρούμε τη συντομότερη διαδρομή των πόλεων, δηλαδή μία αμφημονοσήμαντη αντιστοιχία π από το σύνολο $\{1,2,\dots,n\}$ στον εαυτό του (όπου διαισθητικά, $\pi(i)$ είναι η πόλη την οποία επισκεφτόμαστε στην i-οστή διαδρομή, τέτοια ώστε η ποσότητα $c(\pi) = d_{\pi(1)\pi(2)} + d_{\pi(2)\pi(3)} + \cdots + d_{\pi(n-1)\pi(n)} + d_{\pi(n)\pi(1)}$ να είναι όσο το δυνατόν μικρότερη.

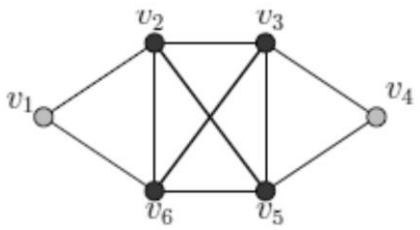
Για να κατανοήσουμε το πρόβλημα του περιοδεύοντος πωλητή, αναφέρουμε το εξής πρόβλημα: Έχοντας ξεκινήσει από την πόλη Α, μέχρι να φτάσεις στην πόλη Β κάνε ένα από τα εξής τρία βήματα:

- Εάν βρίσκεσαι στην πόλη Α και δεν υπάρχει δρόμος που να μην έχεις ακολουθήσει ξανά, τερμάτισε και δήλωσε ότι οι πόλεις Α και Β δεν συνδέονται μεταξύ τους.
- Από την πόλη που βρίσκεσαι, κοίταξε όλους του δρόμους που φεύγουν από αυτήν και ακολούθησε κάποιο δρόμο που δεν έχεις ακολουθήσει παλαιότερα.
- Εάν όμως έχεις ακολουθήσει κάποτε όλους τους δρόμους που φεύγουν από την τρέχουσα πόλη γύρισε πίσω σε μία πόλη που είχες επισκεφτεί πιο πριν ακολουθώντας προς τα πίσω το δρόμο από τον οποίο επισκέφτηκες για πρώτη φορά την τρέχουσα πόλη. Δήλωσε ότι οι πόλεις Α και Β συνδέονται μεταξύ τους.

Το πρόβλημα αυτό, είναι άλλο ένα πρόβλημα για το οποίο, παρά τις έντονες ερευνητικές προσπάθειες αρκετών δεκαετιών, κανένας πολυωνυμικός αλγόριθμος δεν είναι γνωστός. Αντίθετα με τα προηγούμενα προβλήματα, δεν είναι ο τύπος προβλήματος που απαιτεί ένα «ναι» ή ένα «όχι» ως απάντηση και μπορεί να μελετηθεί στα πλαίσια μίας γλώσσας. Είναι ένα πρόβλημα βελτιστοποίησης, αφού απαιτεί να βρούμε την καλύτερη (αναλογικά με κάποια συνάρτηση κόστους) ανάμεσα σε πολλές πιθανές λύσεις. Αν καταφέρναμε να λύσουμε το αρχικό πρόβλημα βελτιστοποίησης σε πολυωνυμικό χρόνο, δηλαδή αν είχαμε στη διάθεση μας έναν αλγόριθμο για τον υπολογισμό της φθηνότερης διαδρομής, τότε προφανώς θα μπορούσαμε να λύσουμε την παραλλαγή με προϋπολογισμό σε πολυωνυμικό χρόνο επίσης.

Ανεξάρτητο Σύνολο (***INDEPENDENT SET***): Δεδομένων ενός μη κατευθυνόμενου γραφήματος G και ενός ακεραίου $K \geq 2$, υπάρχει υποσύνολο C του V με $|C| \geq K$ τέτοιο ώστε, για κάθε $v_i, v_j \in C$, να μην υπάρχει ακμή μεταξύ των v_i και v_j;
Ακόμα ένα πρόβλημα, για το οποίο δεν έχει βρεθεί αλγόριθμος πολυωνυμικού χρόνου.
Το επόμενο πρόβλημα είναι υπό κάποια έννοια το ακριβώς αντίθετο.

ΚΛΙΚΑ (***CLIQUE***): Δεδομένων ενός μη κατευθυνόμενου γραφήματος G και ενός ακεραίου $K \geq 2$, υπάρχει υποσύνολο C του V με $|C| \geq K$ τέτοιο ώστε, για κάθε $v_i, v_j \in C$ να υπάρχει ακμή μεταξύ των v_i και v_j.



Ένα ακόμη παράδειγμα για την κλάση NP είναι το ακόλουθο. Ας υποθέσουμε ότι πρέπει να οργανώσουμε τα καταλύματα ώστε να στεγαστεί μία ομάδα τετρακοσίων φοιτητών. Ο χώρος είναι περιορισμένος και μόνο εκατό από τους μαθητές θα λάβουν θέσεις στον κοιτώνα. Για να περιπλέξει τα πράγματα, ο Πρύτανης έχει δώσει μία λίστα από ζεύγη φοιτητών, ώστε έτσι κανένα από αυτά τα ζευγάρια να μην εμφανίζονται στην τελική επιλογή μας. Αυτό είναι ένα παράδειγμα του τι ονομάζουν οι επιστήμονες των υπολογιστών NP-πρόβλημα, δεδομένου ότι είναι εύκολο να ελέγξουμε αν μία συγκεκριμένη επιλογή εκατό μαθητών που προτείνονται από κάποιον είναι ικανοποιητική, ωστόσο το έργο της δημιουργίας ενός τέτοιου καταλόγου από το μηδέν φαίνεται να είναι τόσο σκληρό ώστε να καθίσταται ανέφικτο. Πράγματι, ο συνολικός αριθμός των τρόπων επιλογής εκατό φοιτητών από των τετρακοσίων αιτούντων είναι μεγαλύτερος από τον αριθμό των ατόμων στο γνωστό σύμπαν. Έτσι, δεν έχει υπάρχει μέλλον στον πολιτισμό μας ώστε να ελπίζει ότι θα κατασκευάσει έναν υπερυπολογιστή ικανό να λύσει το πρόβλημα. Ωστόσο, αυτή η προφανής δυσκολία μπορεί να αντανακλά μόνο την έλλειψη εφευρετικότητας του προγραμματιστή.

Στην πραγματικότητα, ένα από τα εκκρεμή προβλήματα στην επιστήμη των υπολογιστών είναι να καθοριστεί αν υπάρχουν ερωτήσεις των οποίων η απάντηση μπορεί να ελεγχθεί γρήγορα, αλλά τα οποία απαιτούν ένα απίστευτα μεγάλο χρονικό διάστημα για να λυθούν. Προβλήματα όπως αυτό που αναφέρεται παραπάνω σίγουρα φαίνεται να είναι αυτού του είδους, αλλά μέχρι στιγμής κανείς δεν έχει καταφέρει να αποδείξει ότι κάποιο από αυτά είναι πραγματικά τόσο δύσκολο όσο φαίνεται, δηλαδή, ότι πραγματικά δεν υπάρχει εφικτός τρόπος να δώσουμε μία απάντηση με τη βοήθεια ενός υπολογιστή.

Το πρόβλημα SAT

Ένα NP πρόβλημα είναι το πρόβλημα SAT, πήρε την ονομασία του από τον όρο "satisfiability" που δηλώνει την ικανοποιησιμότητα. Οι μεταβλητές που χρησιμοποιούνται είναι οι λογικές μεταβλητές και παίρνουν τις τιμές $ΑΛΗΘΕΣ$ και $ΨΕΥΔΕΣ$, οι οποίες αναπαρίστουνται με 1 και 0 αντίστοιχα. Χρησιμοποιούνται πράξεις που ονομάζονται λογικές πράξεις που είναι η σύζευξη, η διάζευξη και η άρνηση και συμβολίζονται με $\wedge$, $\vee$ και $\neg$ αντίστοιχα. Επίσης για κάθε λογική μεταβλητή x η $\bar{x}$ είναι η άρνησή της.

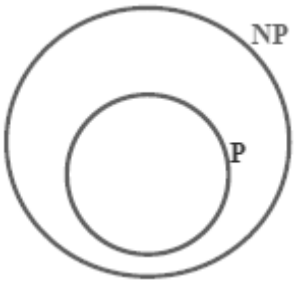
Μερικά παραδείγματα είναι :

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0$$
$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1 \quad \text{και} \quad \bar{0} = 1$$
$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$
$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

Ο λογικός τύπος είναι έκφραση που περιέχει λογικές πράξεις και λογικές μεταβλητές.

Για παράδειγμα $\varphi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$. Ο λογικός τύπος είναι ικανοποιήσιμος αν συνδυάσουμε τιμές 1 και 0 και τελικά πάρει την τιμή 1.
Οπότε ο παραπάνω λογικός τύπος φ θα είναι ικανοποιήσιμος για $x = 0$, $y = 1$ και $z = 0$, αφού λαμβάνει την τιμή 1, άρα ο φ είναι αληθής.

2.5 Το πρόβλημα $P\ vs.\ NP$



Το διάσημο πρόβλημα "$\boldsymbol{P}$ **έναντι** $\boldsymbol{NP}$" είναι ένα από τα βαθιά άλυτα προβλήματα που αντιμετωπίζουν οι μαθηματικοί και οι επιστήμονες της πληροφορικής σήμερα. Γνωρίζουμε ότι το P είναι υποσύνολο του NP. Ωστόσο δεν γνωρίζουμε κατά πόσον αυτή η σχέση είναι γνήσιου υποσυνόλου.

Τα προβλήματα του NP που δεν φαίνεται να ανήκουν στο P έχουν την ιδιότητα να ανάγονται όλα σε ένα σύνολο βασικών προβλημάτων. Υπάρχουν δηλαδή, κάποια προβλήματα στην κλάση NP για τα οποία ισχύει ότι αν κάποιο από αυτά μπορεί να λυθεί σε πολυωνυμικό χρόνο τότε όλα τα προβλήματα της NP μπορούν να επιλυθούν σε πολυωνυμικό χρόνο. Τα προβλήματα που ανήκουν στο NP και έχουν αυτή την ιδιότητα καλούνται **NP-πλήρη (NP-complete)**.

Το ερώτημα $P\ vs.\ NP$ αποτελεί το πιο σημαντικό ανοικτό πρόβλημα της υπολογιστικής πολυπλοκότητας, αφορά αν τελικά $P = NP$ ή $P \neq NP$. Αν ισχύει το πρώτο, οποιαδήποτε προβλήματα λύνονται εξετάζοντας όλες τις περιπτώσεις μέχρι να βρεθεί η σωστή λύση, θα μπορούν να λυθούν και χωρίς αυτή τη διαδικασία της «εξαντλητικής αναζήτησης». Κατά τα λεγόμενα του καθηγητή θεωρητικής πληροφορικής Χρίστου Παπαδημητρίου, το $P\ vs.\ NP$ ρωτάει δηλαδή αν μπορεί να λυθεί ένα πρόβλημα με τον καλύτερο δυνατό τρόπο

χρησιμοποιώντας αλγοριθμικές μεθόδους. Άμα «ψάξεις» τα πάντα θα βρείς τη λύση, αλλά το θέμα είναι πως για να «ψάξεις» τα πάντα χρειάζεσαι υπολογισμούς της τάξης μεγέθους του σύμπαντος, επομένως δεν μπορείς.

Αν ανακαλυφθεί Μηχανή Turing που να επιλύει κάποιο από αυτά ντετερμινιστικά σε πολυωνυμικό χρόνο (οπότε αυτό θα ανήκει στο P), τότε όλα τα NP-πλήρη προβλήματα θα ανήκουν στο P. Αυτό που γνωρίζουμε μέχρι στιγμής είναι ότι η κλάση P είναι υποσύνολο της κλάσης NP. Συνεπώς, οι ερευνητές είχαν στα χέρια τους προβλήματα για τα οποία είχαν ήδη βρει πολυωνυμικούς αλγορίθμους επίλυσης, και προβλήματα που δεν διαφαινόταν εάν μπορούσαν να επιλυθούν από πολυωνυμικό αλγόριθμο. Ωστόσο, άν παρεχόταν η λύση, υπήρχε πολυωνυμικός αλγόριθμος που να ελέγχει εάν πράγματι αυτή είναι λύση ή όχι.

Η επόμενη σκέψη ήταν να βρεθούν τα προβλήματα που θεωρούνται τα πιο δύσκολα, διότι εάν μπορούσαμε να λύσουμε πολυωνυμικά ένα από αυτά, θα μπορούσαν να λυθούν πολυωνυμικά όλα τα προβλήματα στο NP. Έτσι γεννήθηκε η έννοια της **αναγωγής πολυωνυμικού χρόνου** από ένα πρόβλημα σε ένα άλλο. Σύμφωνα με αυτή την έννοια, για να δείξουμε ότι ένα πρόβλημα είναι τουλάχιστον τόσο δύσκολο όσο ένα άλλο όσον αφορά την επίλυση με πολυωνυμικό αλγόριθμο, αρκεί να βρούμε μία απεικόνιση στιγμιότυπων του πρώτου σε στιγμιότυπα του δευτέρου, έτσι ώστε μια λύση στο δεύτερο πρόβλημα να αντιστοιχεί σε μία λύση του πρώτου. Έτσι, δείχνουμε ότι το δεύτερο πρόβλημα δεν μπορεί να είναι πιο εύκολο να λυθεί από το πρώτο.

Το πρώτο πρόβλημα που βρέθηκε να έχει αυτή την ιδιότητα, είναι το πρόβλημα της ικανοποιησιμότητας λογικών προτάσεων ($SATISFIABILITY$ ή SAT) και η σημαντική αυτή ανακάλυψη, γνωστή ως **Θεώρημα Cook-Levin**, έγινε ανεξάρτητα στις αρχές της δεκαετίας του ’70 στις πρωτοποριακές εργασίες των Stephen Cook, Richard Karp και Leonid Levin.

2.5.1 Ιστορική αναδρομή του προβλήματος

Stephen Cook Leonid Levin Richard Karp

Τη δεκαετία του 1960 οι ηλεκτρονικοί υπολογιστές άρχισαν να αντικαθιστούν προηγούμενες μεθόδους δύσκολων και μεγάλων υπολογισμών. Πολλοί μαθηματικοί έστρεψαν το ενδιαφέρον τους σ'αυτή την αναδυόμενη τεχνολογία και ξεκίνησαν να χτίζουν μία μαθηματική θεωρία γύρω απο αυτό. Δημιούργησαν μοντέλα με σκοπό να καθορίσουν έναν υπολογισμό ως πρακτικα εφικτό ή όχι, ασχολήθηκαν με το χρόνο επίλυσης των υπολογισμών και το χώρο (μνήμη) που απαιτεί. Έθεσαν δηλαδή, το μαθηματικό πλαίσιο της θεωρίας πολυπλοκότητας και απέδειξαν κάποια σημαντικά θεωρήματα. Το "$P\ vs. NP$" εδραιώθηκε ως ένα από τα πολύ σημαντικά προβλήματα το 1971, από τους Stephen Cook, Leonid Levin και Richard Karp. Δεκαπέντε χρόνια αργότερα, ανακαλύφθηκε ότι οι ρίζες του προβλήματος βρίσκονται πίσω στο 1956.

Στις αρχές του αιώνα μας, ένας από τους μεγαλύτερους μαθηματικούς, ο David Hilbert, στο Διεθνές Συνέδριο Μαθηματικών στο Παρίσι το 1900 έθεσε εικοσιτρία ανοικτά προβλήματα, τα οποία θεωρούσε ως τα πιο σημαντικά για εκείνη την εποχή. Το δέκατο από αυτά ρωτούσε εάν μπορούσε να βρεθεί μία μέθοδος που να απαντά στο εάν μία δοσμένη διοφαντική εξίσωση έχει λύση ή όχι. Αργότερα, το 1928, στο Διεθνές Συνέδριο

Μαθηματικών στη Μπολόνια, έθετε το πρόβλημα του να βρεθεί μία μέθοδος που μέσα σε πεπερασμένο αριθμό βημάτων, θα μπορούσε να ελέγχει εάν ένα μαθηματικό (λογικό) σύστημα, όπως είναι για παράδειγμα η Ευκλείδειος Γεωμετρία, είναι *συνεπές*, δηλαδή δεν εμπεριέχει *αντιφάσεις*. Με αυτόν τον τρόπο, ένας μαθηματικός που προτείνει ένα πολύπλοκο μαθηματικό σύστημα, το οποίο λόγω του όγκου των αξιωμάτων ή της πληθώρας των αποδεικτικών μεθόδων που περιέχει ξεφεύγει από την δυνατότητα ελέγχου από τον άνθρωπο, θα μπορούσε να πάρει απάντηση στο ερώτημα του αν το σύστημα αυτό περικλείει αντιφάσεις.

Το 1931, ο μαθητής του Hilbert και εξίσου μεγάλος μαθηματικός Kurt Gödel (1906-1978), απέδειξε στην ότι, δύο από τις πεποιθήσεις του Hilbert ήταν λανθασμένες. Η πρώτη προσπάθεια αξιωματοποίησης του συστήματος των φυσικών αριθμών είχε γίνει το 1910 από τους Whitehead και Russell, στο τρίτομο έργο τους *Principia Mathematica.* Ο Gödel παρουσίασε μια μαθηματική πρόταση στα πλαίσια του αξιωματικού συστήματος των φυσικών αριθμών για την οποία δεν ήταν δυνατό να αποδειχθεί με χρήση των αποδεικτικών μεθόδων του συστήματος αυτού ούτε ότι είναι θεώρημα ούτε ότι δεν είναι. Αυτή η μεγάλη ανακάλυψη, είναι το περίφημο **θεώρημα μη πληρότητας του Gödel**. Επιπλέον, απέδειξε ότι η συνέπεια ενός μαθηματικού συστήματος δεν είναι δυνατόν να αποδειχθεί με κανόνες και μεθόδους του ίδιου του συστήματος. Είναι αδύνατο, δηλαδή, να βρεθεί ένα πλήρες και συνεπές σύνολο από αξιώματα για όλα τα μαθηματικά.

Ο Gödel ήταν και ο πρώτος που διερεύνησε ζητήματα σχετικά με την πολυπλοκότητα της αλγοριθμικής επίλυσης προβλημάτων. Ήταν ένας από τους πιο σημαντικούς επιστήμονες της λογικής όλων των εποχών με τεράστια επιρροή στην επιστημονική και φιλοσοφική σκέψη του 20^{ου} αιώνα. Στην αλληλογραφία του με τον εξαιρετικά ευφυή ούγγρο μαθηματικό, John von Neumann (1903-1957) ανακαλύφθηκε ένα γράμμα από το 1956, που γινόταν αναφορά στην ιδέα του $P\ vs. NP$. Πενήντα χρόνια μετά, το πρόβλημα αυτό θα οριστεί ως ένα από τα άλυτα προβλήματα στην ιστορία της επιστήμης των υπολογιστών από το Μαθηματικό Ινστιτούτο Clay ανάμεσα στα επτά ισάριθμα «Βραβεία της Χιλιετίας» (Millennium Prizes) αξίας 1 εκατομμυρίου δολαρίων το καθένα.

Ο Gödel γράφει στον von Neumann ότι έστω ότι υπάρχει μία βέλτιστη μηχανή, η οποία θα μπορούσε να αποφανθεί για την ύπαρξη της απόδειξης μίας μαθηματικής πρότασης. Η ερώτηση του είναι πόσο χρόνο θα χρειαστεί αυτή η μηχανή. Είναι άκρως εντυπωσιακό το

γεγονός ότι ο Gödel οραματίστηκε και υπέδειξε αυτό που είναι σημαντικό στην εν λόγω ερώτηση. Πιο συγκεκριμένα, αναφέρεται στη μηχανή Turing ως υπολογιστικό μοντέλο, και μετά ρωτά ποιά είναι η συνάρτηση που φράσσει τον αριθμό των βημάτων που χρειάζονται για να βρεθούν αποδείξεις μήκους n. Σύμφωνα με τη σημερινή τεχνική ορολογία, ρωτούσε για την ντετερμινιστική υπολογιστική πολυπλοκότητα του προβλήματος της απόδειξης θεωρημάτων.

Στο ίδιο γράμμα, ο Gödel επίσης ρωτάει σχετικά με την υπολογιστική πολυπλοκότητα του ελέγχου εάν ένας φυσικός αριθμός είναι πρώτος και εκφράζει την πεποίθηση ότι η απόδειξη θεωρημάτων δεν πρέπει να είναι και τόσο δύσκολο πρόβλημα υπολογιστικά. Δυστυχώς, ο von Neumann ήδη έπασχε από καρκίνο και πέθανε ένα χρόνο αργότερα. Δεν υπήρξε απάντηση στο γράμμα, και φαίνεται ότι ο ίδιος δεν προσπάθησε να διερευνήσει περισσότερο το πολύ σημαντικό ερώτημα που έθεσε. Το γράμμα αυτό αναμφισβήτητα αποτελεί ένα πολύτιμο έγγραφο στην ιστορία της επιστήμης.

2.6 Χρονική πολυπλοκότητα

2.6.1 Οι συμβολισμοί κεφαλαίου και πεζού όμικρον

Συχνά, ο ακριβής χρόνος εκτέλεσης ενός αλγοριθμού είναι δύσκολο να προσδιοριστεί. Για το λόγο αυτό, περιοριζόμαστε σε κάποια εκτίμηση του χρόνου αυτού, κυρίως σε αλγόριθμους με εισόδους μεγάλου μήκους. Συγκεκριμένα, στην έκφραση που αναπαριστά τον ακριβή χρόνο εκτέλεσης του αλγορίθμου, μας ενδιαφέρει μόνο ο μεγιστοβάθμιος όρος και αγνοούμε το συντελεστή του καθώς και τους όρους χαμηλότερου βαθμού.

Παραδείγματος χάριν, στη συνάρτηση $f(n) = 4n^4 + 3n^3 + 23n^2 + 5n + 64$, ο μεγιστοβάθμιος όρος είναι ο $4n^4$. Η σχέση αυτή αναπαρίσταται ως $f(n) = O(n^4)$. Η έκφραση αυτή, διαβάζεται ως "κεφαλαίο όμικρον του g του n" και πρόκειται για το λεγόμενο **ασυμπτωτικό συμβολισμό** ή **συμβολισμό κεφαλαίου όμικρον**.

ΟΡΙΣΜΟΣ 2.12. Θεωρούμε ότι το σύνολο $\mathbb{R}^+$ αναπαριστά το σύνολο των μη αρνητικών πραγματικών αριθμών. Έστω f και g δύο συναρτήσεις από το σύνολο $\mathbb{N}$ στο σύνολο $\mathbb{R}^+$. Λέμε ότι $f(n) = O(g(n))$ εάν υπάρχουν θετικοί ακέραιοι c και n_0 τέτοιοι ώστε, για κάθε ακέραιο $n \geq n_0$ να ισχύει

$$f(n) \leq cg(n).$$

Όταν $f(n) = O\big(g(n)\big)$, λέμε ότι η $g(n)$ αποτελεί **άνω φράγμα** της $f(n)$ ή ακριβέστερα, **ασυμπτωτικό άνω φράγμα** της $f(n)$, για να το τονίσουμε ότι αγνοούμε τους σταθερούς συντελεστές.

Ο συμβολισμός αυτός, παρουσιάζει ιδιαίτερη συμπεριφορά σε σχέση με τους αλγόριθμους. Όταν γράφουμε $f(n) = O(\log n)$, δε χρειάζεται να διευκρινίζουμε τη βάση, διότι ο σταθερός παράγοντας ούτως ή άλλως θα παραβλεφθεί. (Λόγω της ταυτότητας $\log_b n = \log_2 n / \log_2 b$). Έστω για παράδειγμα η συνάρτηση $f_2(n) = 6n \log_2 n + 3n\log_2 \log_2 n + 2$. Τότε, έχουμε $f_2(n) = O(n \log n)$, αφού η συνάρτηση $\log n$ υπερισχύει της $\log \log n$. Επίσης, στις αριθμητικές εκφράσεις όπως για παράδειγμα η $f(n) = O(n^2) + O(n)$, ο όρος $O(n^2)$ υπερισχύει του $O(n)$ και η έκφραση είναι ισοδύναμη με την $f(n) = O(n^2)$. Το ίδιο ισχύει όταν το σύμβολο εμφανίζεται στον εκθέτη. Η έκφραση $f(n) = 2^{O(n)}$ αντιπροσωπεύει ένα άνω φράγμα της μορφής 2^{cn} για κάποια σταθερά c. Η έκφραση $f(n) = 2^{O(\log n)}$ αντιπροσωπεύει ένα άνω φράγμα της μορφής n^c για κάποια σταθερά n. (Λόγω της $n = 2^{\log_2 n}$, έπεται ότι $n^c = 2^{c \log_2 n}$).

ΟΡΙΣΜΟΣ 2.13 Έστω f και g δυο συναρτήσεις από το σύνολο $\mathbb{N}$ στο σύνολο $\mathbb{R}^+$. Τότε θα λέμε ότι $f(n) = o(g(n))^2$ αν ισχύει ότι $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Δηλαδή η έκφραση $f(n) = o(g(n))$ σημαίνει ότι για οποιονδήποτε πραγματικό αριθμό $c > 0$ υπάρχει ακέραιος n_0 τέτοιος ώστε $f(n) < cg(n)$ για κάθε $n \geq n_0$.

Τα φράγματα της μορφής n^c, όπου c κάποια σταθερά μεγαλύτερη του 0, ονομάζονται **πολυωνυμικά**. Τα φράγματα της μορφής 2^{n^δ}, όπου δ οποιοσδήποτε θετικός πραγματικός αριθμός, ονομάζονται **εκθετικά**.

Παρατηρούμε ότι ο ρυθμός αύξησης ενός πολυωνύμου εξαρτάται από το βαθμό του. Υπάρχουν όμως συναρτήσεις με ρυθμό αύξησης μεγαλύτερο από αυτόν οποιουδήποτε πολυωνύμου. Ένα απλό παράδειγμα είναι η εκθετική συνάρτηση 2^n. Για να το αποδείξουμε, δείχνουμε αρχικά ότι για όλα τα $n \in \mathbb{N}, n \leq 2^n$. Θα χρησιμοποιήσουμε επαγωγή στο n. Για $n = 0$, προφανώς ισχύει. Υποθέτουμε ότι ισχύει για όλους τους φυσικούς μέχρι και το n. Τότε έχουμε

$$n + 1 \leq 2^n + 1 \leq 2^n + 2^n = 2^{n+1}$$

Στην πρώτη ανισότητα χρησιμοποιήσαμε την επαγωγική υπόθεση και στη δεύτερη ότι $1 \leq 2^n$ για όλα τα n. Τώρα θα αποδείξουμε το ίδιο για όλα τα πολυώνυμα.

Θα δείξουμε δηλαδή, ότι για κάθε $i \geq 1, n^i \in O(2^n)$. Άρα $n^i \leq c2^n + d$ για κατάλληλες σταθερές c και d. Παίρνουμε $c = (2i)^i$ και $d = (i^2)^i$. Προκύπτουν δύο περιπτώσεις. Άν $n \leq i^2$, η ανισότητα ισχύει διότι $n^i \leq d$. Αν όμως $n \geq i^2$, θα δείξουμε ότι ισχύει διότι τώρα $n^i \geq c2^n$. Θεωρώ m το πηλίκο της διαίρεσης του n με το i, το μοναδικό ακέραιο για τον οποίο ισχύει ότι $im \leq n < im + 1$. Τότε θα έχουμε

$$\begin{aligned}
n &\leq (im + i)^i \text{ κατα τον ορισμό του } m \\
&= i^i(m + 1)^i \\
&\leq i^i(2^{m+1})^i \text{ κατά την ανισότητα } n \leq 2^n \text{ για } n = m + 1 \\
&\leq c2^{mi} \text{ λόγω του } c = (2i)^i \\
&\leq c2^n \text{ λόγω του ορισμού του } m.
\end{aligned}$$

Συνεπώς, ο ρυθμός αύξησης οποιουδήποτε πολυωνύμου δεν είναι γρηγορότερος απο αυτόν του 2^n.

Το πλήθος των δυαδικών λέξεων μήκους n είναι 2^n. Δεδομένου ότι κάθε περιγραφή είναι μια δυαδική λέξη, το πλήθος των περιγραφών με μήκος μικρότερο από n είναι το πολύ ίσο με το άθροισμα των πληθών των λέξεων με μήκος έως και $n - 1$.

$$\sum_{0 \leq i \leq n-1} 2^i = 1 + 2 + 4 + 8 + \cdots + 2^{n-1} = 2^n - 1$$

2.6.2 Κλάσεις χρονικής πολυπλοκότητας

ΟΡΙΣΜΟΣ 2.14. Έστω M μία αιτιοκρατική μηχανή Turing που τερματίζει για κάθε είσοδο. Ο **χρόνος εκτέλεσης** ή η **χρονική πολυπλοκότητα** της M είναι η συνάρτηση $f : \mathbb{N} \to \mathbb{N}$ που για κάθε n επιστρέφει ως $f(n)$ το μέγιστο πλήθος βημάτων που είναι δυνατόν να πραγματοποιήσει η M όταν το μήκος της εισόδου είναι το n. Εάν ο χρόνος εκτέλεσης της M είναι $f(n)$, λέμε ότι η M εκτελείται σε χρόνο $f(n)$. Κατά σύμβαση, το μήκος της εισόδου αναπαρίσταται πάντοτε από τη μεταβλητή n.

Οι κλάσεις χρονικής πολυπλοκότητας $TIME\ (t(n))$ και $NTIME\ (t(n))$ ορίζονται ως εξής:

ΟΡΙΣΜΟΣ 2.15. $\boldsymbol{TIME\ (t(n))}$ = { L | η γλώσσα L διαγιγνώσκεται από κάποια αιτιοκρατική μηχανή Turing χώρου $O(t(n))$ }.

ΟΡΙΣΜΟΣ 2.16. $\boldsymbol{NTIME\ (t(n))}$ = { L | η γλώσσα L διαγιγνώσκεται από κάποια αναιτιοκρατική μηχανή Turing χώρου $O(t(n))$ }.

2.7 Χωρική Πολυπλοκότητα

ΟΡΙΣΜΟΣ 2.17. Έστω M μία αιτιοκρατική μηχανή Turing που τερματίζει για όλες τις εισόδους. Η **χωρική πολυπλοκότητα** της M είναι η συνάρτηση $f : \mathbb{N} \to \mathbb{N}$ που για καθε n επιστρέφει ως $f(n)$ το μέγιστο πλήθος θέσεων ταινίας που που είναι δυνατόν να διατρέξει η M όταν το μήκος της εισόδου της είναι n. Εάν η χωρική πολυπλοκότητα της M είναι $f(n)$, λέμε επίσης ότι η M λειτουργεί σε χώρο $f(n)$, ή ότι έχει χώρο εκτέλεσης $f(n)$. Εάν η M είναι μία ανταιτιοκρατική μηχανή Turing τέτοια ώστε για κάθε είσοδο να τερματίζουν όλοι οι κλάδοι του υπολογισμού, τότε η χωρική της πολυπλοκότητα $f(n)$ ορίζεται ως το μέγιστο πλήθος θέσεων ταινίας που είναι δυνατόν να διατρέξει η M σε οποιονδήποτε κλάδο όταν το μήκος της εισόδου της είναι n.

Για την εκτίμηση της χωρικής πολυπλοκότητας των μηχανών Turing χρησιμοποιούμε συνήθως τον ασυμπτωματικό συμβολισμό.

2.7.1 Κλάσεις Χωρικής Πολυπλοκότητας

Έστω $f : \mathbb{N} \longrightarrow \mathbb{R}$ οποιαδήποτε συνάρτηση. Οι κλάσεις χωρικής πολυπλοκότητας $SPACE\ (f(n))$ και $NSPACE\ (f(n))$ ορίζονται ως εξής:

ΟΡΙΣΜΟΣ 2.18 $\boldsymbol{SPACE\ (f(n))}$ = { L | η γλώσσα L διαγιγνώσκεται από κάποια αιτιοκρατική μηχανή Turing χώρου $O(f(n))$ }.

ΟΡΙΣΜΟΣ 2.19 $\boldsymbol{NSPACE\ (f(n))}$ = { L | η γλώσσα L διαγιγνώσκεται από κάποια αναιτιοκρατική μηχανή Turing χώρου $O(f(n))$ }.

ΟΡΙΣΜΟΣ 2.20. Η $\boldsymbol{PSPACE\ (f(n))}$ αποτελείται από τις γλώσσες που μπορούν να διαγνωστούν σε πολυωνυμικό χρόνο από κάποια αιτιοκρατική μηχανή Turing, δηλαδή

$$PSPACE = \bigcup_k SPACE(n^k).$$

ΟΡΙΣΜΟΣ 2.21 Μία γλώσσα L είναι $\boldsymbol{PSPACE - \pi\lambda\acute{\eta}\rho\eta\varsigma}$ αν ικανοποιεί τις εξής δύο συνθήκες:

- Η L ανήκει στην κλάση $PSPACE$.
- Κάθε γλώσσα $L' \in PSPACE$ ανάγεται στην L σε πολυωνυμικό χρόνο.

2.7.2 Θεώρημα Savitch

ΟΡΙΣΜΟΣ 2.22 Ένας γράφος $G = (V, E)$ είναι ένα σύνολο κόμβων V που ενώνονται με ένα σύνολο ακμών E.
Ένας γράφος $G = (V, E)$ με n κορυφές μπορεί εναλλακτικά να αναπαρασταθεί από έναν πίνακα γειτνίασης.

ΟΡΙΣΜΟΣ 2.23 Ο **πίνακας γειτνίασης** περιέχει τις τιμές 0 και 1, όπου αν η (i, j) είναι ακμή τότε

$$A[(i,j)] = 1, \quad \text{διαφορετικά} \quad A[(i,j)] = 0.$$

ΟΡΙΣΜΟΣ 2.24 Ένας γράφος ονομάζεται **κατευθυνόμενος** αν κάθε μία από τις ακμές του είναι προσανατολισμένη προς μία κατεύθυνση. Ορίζονται δηλαδή από διατεταγμένα ζεύγη κόμβων.

Θεώρημα 2.3 (Θεώρημα Savitch)

$$Reachability \in SPACE(log^2 n)$$

Απόδειξη

Έστω G, ένας γράφος με n κόμβους. Θεωρώ x και y κόμβους του G και έστω $i \geq 0$. Ορίζουμε ότι το κατηγόρημα $PATH(x, y, i)$ είναι αληθές εαν το ακόλουθο είναι αληθές: Υπάρχει ένα μονοπάτι από το x στο y μέσα στο G, μήκους το πολύ 2^i. Παρατηρούμε αμέσως ότι εφόσον ένα μονοπάτι στο G είναι το πολύ μήκους n, μπορούμε να λύσουμε το πρόβλημα $Reachability$ στο G αν έχουμε τη δυνατότητα να υπολογίσουμε την αληθοτιμή του $PATH(x, y, \lceil logn \rceil)$ για δύο οποιουσδήποτε δοσμένους κόμβους στο G.

Θα σχεδιάσουμε μία μηχανή Turing με δυο $working\ strings$ και ένα $input\ string$ που υπολογίζει την αληθοτιμή του $PATH(x, y, i)$. Ο πίνακας γειτνίασης του G δίνεται στο $input\ string$. Υποθέτουμε ότι το πρώτο $working\ string$ περιέχει κόμβους x και y και έναν ακέραιο i, τα οποία είναι δυαδικής μορφής. Επίσης περιέχει διάφορες τριάδες και η

(x, y, i) είναι η πρώτη από τα αριστερά. Η δεύτερη $working\ string$ χρησιμοποιείται ως κενός χώρος – ο χώρος $O(logn)$ επαρκεί.

Θα περιγράψουμε με ποιον τρόπο η μηχανή υπολογίζει την αληθοτιμή για το $PATH(x, y, i)$.

- Αν $i = 0$ εξετάζουμε αν το x και το y συνδέονται μέσω ενός μονοπατιού μήκους το πολύ $2^{\ i} = 1$, απλά ελέγχοντας αν $x = y$ ή αν το x και το y είναι γειτονικά κοιτάζοντας το $input$.
- Αν $i \geq 1$ αποφασίζουμε το $PATH(x, y, i)$ από τον ακόλουθο αναδρομικό αλγόριθμο: Για όλους τους κόμβους z ελέγχουμε αν το $PATH(x, z, i - 1)$ και το $PATH(z, y, i - 1)$ είναι αληθή.

Αυτό το πρόγραμμα εκτελεί μια πολύ απλή ιδέα. Οποιοδήποτε μονοπάτι μήκους 2^i από το x στο y έχει ένα μέσο z από όπου τόσο το x όσο και το y απέχουν το πολύ 2^{i-1}. Για την εξοικονόμηση χώρου δημιουργούμε όλους τους πιθανούς κόμβους z επαναχρησιμοποιώντας χώρο.

Κάθε φορά που δημιουργείται ένα καινούριο z προστίθεται στην πρώτη $working\ string$ μια τριάδα $(x, z, i - 1)$ και ξεκινά η αναδρομική λύση του προβλήματος της αληθοτιμής της. Αν κατά την αναδρομή βρεθεί $PATH(x, z, i - 1) = false$ διαγράφεται η τριάδα και προστίθεται μια καινούρια με νέο z. Εάν $PATH(x, z, i - 1) = true$ διαγράφουμε την τριάδα και προστίθεται η $(z, y, i - 1)$ για έλεγχο (το y το παίρνουμε από την αμέσως αριστερή τριάδα στο $string$) ώστε να αποφασιστεί αν το $PATH(x, y, i - 1)$ είναι αληθές. Εάν δεν είναι, διαγράφουμε την τριάδα και δοκιμάζουμε το επόμενο z. Αλλιώς, ελέγχουμε αν είναι η δεύτερη αναδρομική κλίση, συγκρίνοντας με την τριάδα (x, y, i) και αποδίδεται αληθοτιμή στο $PATH(x, y, i) = true$.

Είναι εμφανές ότι ο παραπάνω αλγόριθμος υπολογίζει την αληθοτιμή του κατηγορήματος $PATH(x, y, i)$ και αποφασίζει ανάλογα εάν υπάρχει μονοπάτι από το x στο y. Το πρώτο $working\ string$ περιέχει ανά πάσα στιγμή $\lceil logn \rceil$ ή λιγότερες τριάδες που η κάθε μία είναι μήκους το πολύ $3logn$. Για να λυθεί το πρόβλημα $Reachability$ ξεκινάμε αυτό τον αλγόριθμο με την τριάδα $(x, y, \lceil logn \rceil)$ στο πρώτο $working\ string$. □

ΟΡΙΣΜΟΣ 2.25 Ορίζουμε τον επόμενο **"$configuration$" γράφο** της M με $input\ x$, συμβολίζοντας $G\ (M, x)$ να είναι ο γράφος ο οποίος έχει ως κόμβους όλα τα δυνατά "$configurations$" της M. Μεταξύ δύο "$configurations$" c_1 και c_2 υπάρχει μία ακμή αν και μόνο αν $c_1 \overset{M}{\rightarrow} c_2$.

Η έννοια του γράφου των $configurations$ αποτελεί διαισθητικά ένα στιγμιότυπο του υπολογισμού της M με δοθέν $input\ x$.

Από το θεώρημα του Savitch προκύπτει ένα σημαντικό αποτέλεσμα.

Πόρισμα: $NSPACE\ (f(n) \subseteq SPACE\ (f^2(n))$ για οποιαδήποτε συνάρτηση πολυπλοκότητας $f(n) \geq logn$.

Απόδειξη

Για την απόδειξη αυτού τρέχουμε τον αλγόριθμο του θεωρήματος Savitch στο γράφο των $configurations$ της μη ντετερμινιστικής μηχανής Turing.

Προσομοιώνοντας μια $f(n)$-$space\ bounded$ μη ντετερμινιστική μηχανή Turing M με $input\ x$, $|x| = n$, τρέχουμε τον αλγόριθμο του θεωρήματος του Savitch στο $configuration$ γράφο της M με $input\ x$, δηλαδή επιλύουμε το πρόβλημα $Reachability$ σε χώρο $O(log^2 n)$. Όμως επειδή $f(n) \geq logn$ το πλήθος των κόμβων στο $configuration\ graph\ G$ είναι $c^{f(n)}$. Άρα επαρκεί χώρος $O(f^2(n))$. Οι υπόλοιπες εργασίες (διατήρηση στη μνήμη της τρέχουσας, διαμόρφωση και δημιουργία της επόμενης) καταλαμβάνουν χώρο $O(f(n))$. Επειδή όμως το τετράγωνο του πολυωνύμου είναι επίσης πολυώνυμο, συμπεραίνουμε ότι $PSPACE = NPSPACE$. □

2.8 $P^A = NP^A$ και $P^B \neq NP^B$

Η απαρχή της ιδέας του υπολογισμού με χρήση αλγορίθμων που επιλύουν κάποιο άλλο πρόβλημα, είναι λίγο παράξενη. Το υπολογιστικό μοντέλο που εκφράζει αυτήν την έννοια της σχετικής υπολογισιμότητας δημιουργήθηκε από τον Alan Turing. Η ιδέα αυτής της έννοιας, καθώς και η έννοια των μηχανών με μαντείο, γεννήθηκε από την προσπάθεια του Turing να μελετήσει τις βαθύτερες συνέπειες του Θεωρήματος της Μη-Πληρότητας του Kurt Gödel όσον αφορά τη Θεωρία Υπολογισιμότητας.

Η ιδέα της μηχανής Turing με μαντείο, οδηγεί σε μία γενικότερη έννοια αναγωγής και θα μας βοηθήσει να διατυπώσουμε και να αποδείξουμε ένα από τα πιο ενδιαφέροντα θεωρήματα στη Θεωρία Υπολογιστικής Πολυπλοκότητας, το θεώρημα των Baker, Gill & Solovay που αφορά τη "σχετικοποίηση" του προβλήματος $P\ vs. NP$. Μία μη-αιτιοκρατική μηχανή Turing με μαντείο, μπορεί να μαντέψει τη σωστή απάντηση ή λύση. Διευκρινίζουμε πως δεν πρόκειται για ρεαλιστικό υπολογιστικό μοντέλο, αλλά για μαθηματική κατασκευή που βοηθά να επεξεργαστούμε ερωτήματα απο τη σκοπιά της πολυπλοκότητας.

ΟΡΙΣΜΟΣ 2.26. Μια **μηχανή Turing με μαντείο**, $M^?$, είναι μία ντετερμινιστική μηχανή Turing πολλών συμβολοσειρών, η οποία έχει τρείς ειδικές καταστάσεις. Το ερώτημα κατάστασης $q_?$ και τα ερωτήματα απάντησης q_{yes}, q_{no}. Το σύμβολο "?" στον εκθέτη δηλώνει ό,τι οποιαδήποτε γλώσσα μπορεί να χρησιμοποιηθεί ως ένα μαντείο.

Συμπεραίνουμε λοιπόν ότι το μαντείο δέχεται ερωτήσεις από τη μηχανή και απαντά με yes ή no αν κάποιο $string$ ανήκει σε μια γλώσσα L. Η γλώσσα L δεν είναι παρά μία παράμετρος και η μηχανή μπορεί να λειτουργήσει με μαντείο για οποιαδήποτε γλώσσα στη θέση της L. Σύμφωνα με τα παραπάνω είναι προφανές ότι υπάρχουν τα εξής ενδεχόμενα. Πρώτον για κάθε επιλογή μαντείου L, υπάρχουν $input\ strings$ για τα οποία δεν τερματίζει. Δεύτερον, για κάποια μαντεία η μηχανή τερματίζει πάντα και για τα υπόλοιπα, υπάρχουν $input\ strings$ για τα οποία δεν τερματίζει. Τρίτον, η μηχανή τερματίζει σε όλα τα $input\ strings$ για οποιοδήποτε μαντείο της έχει παρασχεθεί.

Υποθέτουμε ότι B ⊆ $\{0,1\}^*$:

ΟΡΙΣΜΟΣ 2.27. $\boldsymbol{P^B}$ είναι το σύνολο που περιέχει κάθε γλώσσα που μπορεί να αποφασιστεί από μία πολυωνυμικού χρόνου μηχανή Turing με μαντείο B.

ΟΡΙΣΜΟΣ 2.28. $\boldsymbol{NP^B}$ είναι το σύνολο που περιέχει κάθε γλώσσα που μπορεί να αποφασιστεί απο μία πολυωνυμικού χρόνου μη ντετερμινιστική μηχανή Turing με μαντείο B.

Στη συνέχεια, θα αποδείξουμε δύο χρήσιμες ανισότητες, τις οποίες θα χρησιμοποιήσουμε στην απόδειξη $P^A = NP^A$ και $P^B \neq \mathrm{N}P^B$.

1.

$$\sum_{j=1}^{i} \boldsymbol{j^{log\, j} < 2^i}$$

Απόδειξη

Από τις ιδιότητες του λογαρίθμου, ισχύει ότι $j = 2^{log\, j}$. Έχω ότι

$$\sum_{j=1}^{i} j^{log\, j} = \sum_{j=1}^{i} 2^{(log\, j)^2} \overset{?}{<} 2^i$$

Το 2^i γράφεται :

$2^i = 1 + 1 + 2 + 2^2 + \cdots + 2^{\iota-1} = 1 + \sum_{j=1}^{i} 2^{j-1}$.

Άρα έχουμε ότι :

$$\sum_{j=1}^{i} 2^{(log\, j)^2} < 1 + \sum_{j=1}^{i} 2^{j-1}$$

Αρκεί να δείξουμε ότι $(log j)^2 \leq j - 1$ το οποίο ισχύει για $j \geq 2$.

Θα κάνουμε μελέτη της συνάρτησης $f(x) = x - 1 - (log x)^2,\ x > 0$.

Θα δείξουμε ότι η συνάρτηση $f : (0, \infty) \to \mathbb{R}$ είναι γνησίως αύξουσα στο $(2, \infty)$. Άν το δείξουμε, επειδή $f(1) = 0, f(2) = 0, \forall\, x > 2, f(x) > 0 \Rightarrow x - 1 > (log x)^2$.

Μελέτη συνάρτησης : Η $f(x) = x - 1 - (logx)^2,\ x > 0$ έχει πεδίο ορισμού το (0, ∞).

$$f'(x) = 1 - \frac{2logx}{x}$$

$$1 - \frac{2logx}{x} > 0 \Leftrightarrow$$

$$\frac{logx}{x} < \frac{1}{2} \Leftrightarrow$$

$$x - 2logx > 0.$$

Θα μελετήσουμε αυτήν τη συνάρτηση ώστε να αποφανθούμε για την $f(x)$.

Έστω $g(x) = x - 2logx$. Έχουμε ότι $g'(x) = 1 - \frac{2}{x}$

Άν $g'(x) = 0$, έχουμε ότι $1 - \frac{2}{x} = 0 \Leftrightarrow 1 = \frac{2}{x} \Leftrightarrow x = 2$

$g'(x) > 0 \Leftrightarrow 1 - \frac{2}{x} > 0 \Leftrightarrow x > 2$

Συμπεραίνουμε ότι η $g(x)$ είναι γνησίως αύξουσα στο διάστημα $(2, \infty)$. Άρα και η $f(x)$ είναι γνησίως αύξουσα στο $(2, \infty)$. Συνεπώς ισχύει ότι $\sum_{j=1}^{i} 2^{(logj)^2} \overset{?}{<} 2^i$ και άρα $\sum_{j=1}^{i} j^{log\, j} < 2^i$. □

2.

$$\boldsymbol{x^{logx} > p(x)}$$

Απόδειξη

Για την απόδειξη αυτού, αρκεί να αποδείξουμε την ανισότητα $x^{logx} > x^n$ για κάθε $n \in \mathbb{N}$.

Άρα θα πρέπει να δείξουμε ότι $\lim_{x \to \infty} \frac{x^n}{logx} = 0$

Ισχύει ότι $\frac{x^n}{logx} > \frac{x}{logx} > \frac{(\sqrt{x})^2}{logx} > \sqrt{x} \longrightarrow \infty$

Από την προηγούμενη απόδειξη γνωρίζουμε ότι $logx < \sqrt{x}$ $\gamma\iota\alpha\ x > 2$.

Συνεπώς $x^{logx} > p(x)$. □

Θεώρημα 2.4. Υπάρχει ένα μαντείο A για το οποίο ισχύει ότι:

1. $P^A = NP^A$
2. $P^B \neq \mathrm{N}P^B$.

Απόδειξη

1. Αν ισχύει η ισότητα, τότε όλα τα προβήματα που βρίσκονται στην κλάση NP θα μπορούν να λυθούν σε πολυωνυμικό χρόνο.

Θεωρούμε το A τέτοιο ώστε να είναι $PSPACE - πλήρης$ γλώσσα και θα ισχύει το εξής:

$PSPACE \subseteq P^A \subseteq \mathrm{N}P^A \subseteq NPSPACE \subseteq PSPACE$

Έτσι συμπεραίνεται ότι $P^A = \mathrm{N}P^A$.

Αρχικά το $PSPACE \subseteq P^A$ ισχύει γιατί το A είναι $PSPACE - πλήρες$ άρα κάθε γλώσσα $L \in PSPACE$ μπορεί να αποφασιστεί από μία ντετερμινιστική μηχανή Turing πολυωνυμικού χρόνου η οποία εκτελεί τη μετατροπή από L σε A και μετά χρησιμοποιεί το μαντείο μία φορά. Η δεύτερη συμπερίληψη είναι προφανής γιατί το $P^A \subseteq \mathrm{N}P^A$ ισχύει πάντα. Το $\mathrm{N}P^A \subseteq NPSPACE$ είναι αληθές γιατί κάθε μη ντετερμινιστική μηχανή Turing πολυωνυμικού χρόνου με μαντείο A μπορεί να προσομοιωθεί από μια μη ντετερμινιστική πολυωνυμικά φραγμένη μηχανή Turing. Η τελευταία συμπερίληψη $NPSPACE \subseteq PSPACE$ προκύπτει από το θεώρημα του Savitch, το οποίο σχετίζεται με τη χωρική πολυπλοκότητα και αφορά την προσομοίωση των μη ντετερμινιστικών μηχανών από ντετερμινιστικές. □

2. Η βασική ιδέα του μαντείου B είναι η εξής: Μία μηχανή, η οποία θέτει πολυωνυμικού αριθμού ερωτήματα δε μπορεί να παίρνει πάντα τις σωστές αποφάσεις για κάθε είσοδο. Αντίθετα, μία μηχανή που θέτει εκθετικού αριθμού ερωτήματα, μπορεί.

Για την απόδειξη αυτή χρησιμοποιείται ένα μαντείο B και μία γλώσσα τέτοια ώστε $L \in \mathrm{N}P^B - P^B$ όπου ορίζεται $L = \{ 0^n$: Υπάρχει ένα $x \in B$ με $|x| = n \}$. Παρατηρούμε αμέσως ότι $L \in NP^B$. Δηλαδή μία μη ντετερμινιστική μηχανή με μαντείο B και $input$ 0^n απλώς μαντεύει μία λέξη x με μήκος n και χρησιμοποιεί το μαντείο ώστε να επαληθεύσει αν το $x \in B$.

Στη συνέχεια για να δείξουμε ότι η γλώσσα $L \notin P^B$ θα ορίσουμε το μαντείο B ως $B \subseteq \{0,1\}^*$. Αυτό θα γίνει με τη μέθοδο της «αργής διαγωνοποίησης» όλων των ντετερμινιστικών μηχανών Turing με μαντείο σε πολυωνυμικό χρόνο. Θεωρούμε μία απαρίθμηση $M_1^?$,$M_2^?$,…….. όλων των μηχανών Turing. Υποθέτουμε ότι κάθε μηχανή επαναλαμβάνεται άπειρες φορές στην αρίθμηση.

Κατασκευάζουμε το μαντείο B σε στάδια. Έχουμε υπολογίσει στην αρχή το B_{i-1} δηλαδή το σύνολο όλων των $strings$ στο B που έχουν μήκος μικρότερο από i. Επιπλέον ορίζουμε το $X_{i-1} \subseteq \{0,1\}^*$ ως το σύνολο των εξαιρέσεων και εκεί τοποθετούμε τα $strings$ τα οποία πρέπει να θυμόμαστε ότι δε θα βάλουμε μέσα στο σύνολο B. Θεωρούμε αρχικά ότι $B_0 = X_0 = \emptyset$. Στη συνέχεια θα ορίσουμε το σύνολο B_i αποφασίζοντας ποιά $strings$ μήκους i θα πρέπει να τοποθετήσουμε στο B. Αυτή η επιλογή θα γίνει προσωμοιώνοντας μία μηχανή Turing $M_i^B(0^i)$ με μαντείο B και $input$ 0^i για i^{logi} βήματα. Επιλέξαμε αυτό τον αριθμό βημάτων γιατί είναι πολύ μικρότερος από την εκθετική συνάρτηση και συγχρόνως πολύ μεγαλύτερος από την πολυωνυμική συνάρτηση.

Κατά τη διάρκεια της προσωμοίωσης η μηχανή $M_i^B(0^i)$ με μαντείο ρωτάει αν η λέξη $x \in B$. Σ' αυτή την ερώτηση απαντάει διακρίνοντας δυο περιπτώσεις για το x.

- Στην πρώτη περίπτωση αν το μήκος του x δεν ξεπερνάει το i δηλαδή αν $|x| < i$ ελέγχουμε απλά αν το x είναι μέσα στο B_{i-1}. Αν είναι μέσα στο B_{i-1} τότε η μηχανή M_i^B απαντάει q_{yes} , αλλιώς απαντάει q_{no} και ο υπολογισμός συνεχίζεται.
- Στην δεύτερη περίπτωση αν το μήκος του x είναι μεγαλύτερο ή ίσο του i , δηλαδή αν $|x| \geq i$ τότε η μηχανή Turing M_i^B απαντάει q_{no}. Σε αυτή την περίπτωση προσθέτουμε το x στο σύνολο των εξαιρέσεων έτσι ώστε να θυμόμαστε ότι το $x \notin B$.

Υποθέτουμε μετά από i^{logi} ή λιγότερα βήματα η μηχανή τερματίζει. Αυτό που θέλουμε να δείξουμε είναι ότι το $L \notin P^B$ δηλαδή ότι η μηχανή Turing με μαντείο M_i^B δε μπορεί να αποφανθεί για τη γλώσσα L. Έτσι ορίζουμε το $B_i = B_{i-1} \cup \{ x \in \{0,1\}^* : |x| = i , x \notin X\}$ οπότε είμαστε σίγουροι ότι το $input$ $0^i \in L$ από τον ορισμό της γλώσσας L όμως αφού η μηχανή M_i^B τερματίζει, έχει ήδη απορρίψει το $input$ 0^i και αποτυγχάνει να είναι η μηχανή που αποφαίνεται για το L, άρα $L(M_i^B) \neq L$. Το κλειδί εδώ βρίσκεται στην παρατήρηση ότι η μηχανή προλαβαίνει στη λειτουργία της να κάνει μόνο πολυωνυμικές το πλήθος ερωτήσεις $(p_i(n))$ στο μαντείο B, ενώ υπάρχουν εκθετικά το πλήθος (2^n) $strings$ μήκους n.

Για να ορίσουμε όμως το σύνολο B_i όπως παραπάνω πρέπει να ξέρουμε ότι το σύνολο $\{ x \in \{0,1\}^* : |x| = i , x \notin X\}$ είναι μη κενό. Αυτό αποδεικνύεται γιατί γνωρίζουμε ότι το σύνολο X των εξαιρέσεων δεν μπορεί να περιέχει περισσότερα από $\Sigma_{j=1}^{i} j^{logj}$ $strings$ μήκους i, δηλαδή όσο είναι ο αριθμός των βημάτων μέσα στις μηχανές μέχρι τώρα. Επιπλέον αυτό το άθροισμα είναι πάντα μικρότερο του 2^i που είναι ο αριθμός των $strings$ που περιέχονται στο σύνολο $\{0,1\}^i$ οπότε το σύνολο $\{ x \in \{0,1\}^* : |x| = i , x \notin X \}$ θα περιέχει τα υπόλοιπα $strings$ άρα δε θα είναι ποτέ κενό. Σε περίπτωση που η μηχανή M_i^B δέχεται το $input$ 0^i σε καθορισμένο χρόνο τότε ορίζουμε $B_i = B_{i-1}$ όμως από τον ορισμό της γλώσσας L το $0^i \notin L$ άρα και πάλι θα ισχύει $L\ (M_i^B) \neq L$.

Τι γίνεται όμως αν η μηχανή Turing με μαντείο M_i^B αποτύχει να τερματίσει σε i^{logi}βήματα; Άλλωστε, το πολυωνυμικό όριο της $p(n)$ μπορεί να είναι πολύ μεγάλο ενώ η τιμή του i πολύ μικρή έτσι ώστε να ισχύει ότι $i^{logi} < p(i)$. Άν συμβεί αυτό και η μηχανή έχει δεχτεί το $input$ 0^i, ορίζουμε πάλι $B_i = B_{i-1}$ οπότε $0^i \notin L$ άρα $L\ (M_i^B) \neq L$.

Επιπλέον, ισοδύναμες με αυτή μηχανές Turing θα εμφανίζονται απείρως συχνά στην αρίθμηση μας και τελικά, κάποια θα εμφανίζεται σαν M_I^B με ένα δείκτη I πολύ μεγάλο τέτοιο ώστε $I^{logI} \geq p(I)$ το οποίο διασφαλίζει ότι $L \neq L\ (M_I^B) = L\ (M_i^B)$.

Αν συνεχίσουμε αυτή τη διαδικασία, ορίζουμε ολότελα το μαντείο B. Όμως αυτός ο ορισμός του B αποκλείει όλες τις πολυωνυμικές μηχανές με μαντείο B να αποφανθούν για τη γλώσσα L, οπότε συμπεραίνουμε ότι $L \notin P^B$. □

Βιβλιογραφικές αναφορές

[1] Elements of the theory of computation, Harry R. Lewis – Christos H. Papadimitriou, 1998

[2] The Universal Computer, Martin Davis, 2000

[3] Introduction to the theory of Computation, Michael Sipser, 1997

[4] Computational Complexity, Christos H. Papadimitriou, 1993

[5] Lecture : Beyond Computation: The P vs. NP Problem, Michael Sipser, MIT Tuesday, October 3, 2006 at 7:00 PM, Harvard University Science Center - Hall B One Oxford Street, Cambridge, MA, 02138

[6] The Infinite Book, John D. Barrow, 2005

[7] Computational Complexity : A Modern Approach, Textbook by Sanjeev Arora and Boaz Barak, Cambridge University Press, 2007

[8] Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers, Vol. 1 (Studies in Logic and the Foundations of Mathematics, Vol. 125), Piergiorgio Odifreddi, 1992

[9] Computational Complexity: A Conceptual Perspective, Oded Goldreich, 2008

[10] Algorithms, S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani, 2006

[11] Computability and logic, George Boolos, John P. Burgess, Richard Jeffrey, 2007

[12] Computability : a mathematical sketchbook , Douglas S. Bridges, 1994

[13] Αλγόριθμοι και πολυπλοκότητα: οι θεμέλιοι λίθοι της σύγχρονης κρυπτογραφίας, Παύλος Σπυράκης, Γιάννης Κ. Σταματίου