Κεφάλαιο 11

Υπολογισιμότητα και Πολυπλοκότητα

11.1 Ιστορία - Εισαγωγή

Η Συλλογιστική του Αριστοτέλη αποτέλεσε την πρώτη προσπάθεια θεμελίωσης της λογικής και των μαθηματικών. Ο *Leibni(t)z* πρότεινε το εξής πρόγραμμα:

1. Να δημιουργηθεί μια τυπική γλώσσα (*formal language*), με την οποία να μπορούμε να περιγράψουμε όλες τις μαθηματικές έννοιες και προτάσεις.
2. Να δημιουργηθεί μια μαθηματική θεωρία (δηλαδή ένα σύνολο από αξιώματα και συμπερασματικούς κανόνες συνεπαγωγής), με την οποία να μπορούμε να αποδεικνύουμε όλες τις ορθές μαθηματικές προτάσεις.
3. Να αποδειχθεί ότι αυτή η θεωρία είναι συνεπής (*consistent*), (δηλαδή ότι η πρόταση "Α και όχι Α" ($A \wedge \neg A$) δεν είναι δυνατόν να αποδειχθεί σ' αυτή τη θεωρία).

Η πραγμάτωση αυτού του προγράμματος άρχισε πολύ αργότερα, προς το τέλος του 19ου αιώνα. Πολλοί επιστήμονες ασχολήθηκαν με τον ορισμό της ενιαίας γλώσσας της μαθηματικής (ή συμβολικής) λογικής (*Boole, Frege*, κ.α). 'Αλλοι ασχολήθηκαν με τον ορισμό της ενιαίας θεωρίας των συνόλων (*Cantor*, κ.α.) και άλλοι με την παραγωγή (*derivation*) όλων των αληθών μαθηματικών προτάσεων με χρήση της Συνολοθεωρίας (*Russel, Whitehead*, κ.α.).

Στην αρχή αυτού του αιώνα ο *Hilbert* βάλθηκε να πραγματοποιήσει το 3ο μέρος του προγράμματος του *Leibni(t)z*, δηλαδή να βρει έναν αλγόριθμο που να αποκρίνεται (*decides*) για την ορθότητα κάθε μαθηματικής πρότασης. Τελικά, όμως, το 1931 ο *Gödel* απέδειξε ότι:

- Δεν υπάρχει τέτοιος αλγόριθμος.
- Είναι αδύνατον να αποδειχθεί η συνέπεια της Συνολοθεωρίας.
- Επιπλέον, οποιαδήποτε (δηλαδή όχι μόνο η Συνολοθεωρία) αξιωματική θεωρία των Μαθηματικών, που περιλαμβάνει τουλάχιστον την Αριθμοθεωρία, θα περιλαμβάνει και μη αποκρίσιμες (*undecidable*) προτάσεις.

- Κωδικοποιώντας προτάσεις με φυσικούς αριθμούς (αυτή η κωδικοποίηση λέγεται σήμερα "Γκεντελοποίηση": *Gödelization*) μπόρεσε να παρουσιάσει μια συγκεκριμένη πρόταση που είναι μη αποκρίσιμη.

Το αποτέλεσμα αυτό του *Gödel* ήταν η αιτία μιας σημαντικής κρίσης στα κλασσικά μαθηματικά, μα συγχρόνως και η απαρχή των μοντέρνων δυναμικών μαθηματικών. Το κεντρικό ερώτημα δεν είναι πια απλά αν μια πρόταση είναι αληθής η ψευδής, αλλά αν είναι "αποκρίσιμη ή μη αποκρίσιμη", δηλαδή αν είναι "υπολογιστή (*computable*) ή όχι". Αυτό ακριβώς είναι και το αντικείμενο της **Θεωρίας της Υπολογιστότητας** (*computability*). Αν δοθεί ότι μια συνάρτηση f είναι υπολογιστή, ποιο είναι το κόστος ή τα αγαθά (*resources*) που χρειάζονται για να υπολογίσουμε την f; Αυτό είναι το βασικό ερώτημα της **Θεωρίας της Πολυπλοκότητας** (complexity)[1].

Διάφοροι επιστήμονες (*Turing, Church, Kleene, Post, Markov*, κ.α.) βάλθηκαν να ξεκαθαρίσουν τις έννοιες: υπολογιστό ή επιλύσιμο (*solvable*) με αλγόριθμο, υπολογιστή συνάρτηση και αποκρίσιμο πρόβλημα. Κατέληξαν, λοιπόν, σε διαφορετικά υπολογιστικά μοντέλα, τα οποία όμως αποδείχθηκαν όλα ισοδύναμα μεταξύ τους.

Η περίφημη **Θέση (thesis) των Church-Turing** λέει λοιπόν απλουστευμένα: "Όλα τα γνωστά και τα "άγνωστα" μοντέλα της έννοιας "υπολογιστός" είναι μηχανιστικά ισοδύναμα (*effectively equivalent*)". Δηλαδή δοθέντος ενός αλγορίθμου σε ένα μοντέλο για μια συγκεκριμένη συνάρτηση f, μπορούμε μηχανιστικά (με τη βοήθεια μηχανής) να κατασκευάσουμε αλγόριθμο σε ένα άλλο μοντέλο για την ίδια συνάρτηση f.

Ας χρησιμοποιήσουμε εδώ ένα γνωστό μοντέλο υπολογισμού, μια γλώσσα προγραμματισμού υψηλού επιπέδου. Μια συνάρτηση[2] τότε, θα λέγεται υπολογιστή, αν υπάρχει πρόγραμμα που υπολογίζει την τιμή της για κάθε όρισμα. Είναι προφανές ότι υπάρχουν συναρτήσεις μη υπολογιστές, γιατί:

- Υπάρχουν άπειρα μεν, αλλά μόνο αριθμήσιμα (*countable*) διαφορετικά προγράμματα. Εκτός αυτού μπορούμε χρησιμοποιώντας κωδικοποίηση να τα απαριθμήσουμε μηχανιστικά (*effectively enumerate*)[3]

[1] Συχνά χρησιμοποιείται και ο όρος υπολογίσιμος αντί για υπολογιστός

[2] θα πρέπει εδώ να διευκρινίσουμε ότι αναφερόμαστε σε συναρτήσεις $f : \mathbb{N}^n \to \mathbb{N}$, αφού τα δεδομένα σε ένα πραγματικό υπολογιστή κωδικοποιούνται με φυσικούς αριθμούς (ακολουθίες από 0 και 1). Γενικότερα, αναφερόμαστε σε συναρτήσεις από αριθμήσιμα σύνολα σε αριθμήσιμα σύνολα

[3] Απόδειξη: Κάθε πρόγραμμα μιας γλώσσας προγραμματισμού είναι στοιχείο του Σ^*, όπου $\Sigma = \{a_1, a_2, \ldots, a_m\}$ το αλφάβητο της γλώσσας. Το Σ^* όμως αποτελεί την ένωση $\bigcup_{n=0}^{\infty} \Sigma_n$, όπου Σ_n το σύνολο των συμβολοσειρών του αλφάβητου Σ που έχουν μήκος n. Κάθε σύνολο Σ_n είναι πεπερασμένο και έτσι αν διατάξουμε τα στοιχεία του αλφαβητικά μπορούμε να θεωρήσουμε την ακόλουθη αρίθμηση για το Σ^*:

$$\begin{array}{ll} \Sigma_0: & \{\varepsilon\} \\ \Sigma_1: & \{a_1, a_2, \ldots, a_m\} \\ \Sigma_2: & \{a_1a_1, a_1a_2, \ldots, a_1a_m, \ldots, a_ma_m\} \\ \vdots & \end{array}$$

Η παραπάνω αρίθμηση του Σ^* είναι μηχανιστική, δηλαδή μπορεί να γίνει με πρόγραμμα. Επομένως, με κατάλληλη χρήση compiler για τον έλεγχο ορθότητας μπορούμε να κατασκευάσουμε μηχανιστική αρίθμηση των συντακτικά ορθών n προγραμμάτων

- Από την άλλη μεριά όμως, ξέρουμε ότι υπάρχουν μη αριθμήσιμες άπειρες (*uncountable*) διαφορετικές συναρτήσεις. Αυτό αποδεικνύεται με διαγωνιοποίηση (*diagonalization*), ανάλογη με αυτή που χρησιμοποιούμε για να δείξουμε ότι το σύνολο $\mathbb{R}$ είναι μη αριθμήσιμο[4]

Ας στραφούμε σε ένα συγκεκριμένο πρόβλημα που είναι μη αποκρίσιμο. Όπως ξέρουμε ο μεταγλωττιστής (*compiler*) μιας γλώσσας προγραμματισμού μπορεί να ελέγξει αν ένα πρόγραμμα είναι συντακτικά ορθό ή όχι. Τι γίνεται όμως με τα λάθη χρόνου εκτέλεσης (*run time errors*); Θα ήταν ωραίο να είχαμε ένα πρόγραμμα που θα μπορούσε να ελέγχει αν ένα συντακτικά ορθό πρόγραμμα θα σταματήσει κάποτε ή αν θα τρέχει για πάντα. Δυστυχώς τέτοιο πρόγραμμα δεν υπάρχει.

Θεώρημα 11.1. *Το **halting problem (HP)** είναι μη αποκρίσιμο.*

Απόδειξη. Έστω ότι $\pi_0, \pi_1, \pi_2, \ldots$ είναι μια μηχανιστική απαρίθμηση (*effective enumeration*) όλων των προγραμμάτων. Ας υποθέσουμε ότι το **HP** είναι επιλύσιμο. Τότε κατασκευάζουμε ένα πρόγραμμα π, που ελέγχει αν το πρόγραμμα π_n με είσοδο n σταματάει ή όχι και ανάλογα με την απάντηση σε αυτόν τον έλεγχο, το πρόγραμμα π σταματάει αν το $\pi_n(n)$ δεν σταματάει, και αντιστρόφως:

$$\pi\colon \text{read}(n); \textbf{if } \pi_n(n) \text{ terminates } \textbf{then } \text{loop\_forever } \textbf{else } \text{halt}$$

Φυσικά αυτό το πρόγραμμα π κάπου θα εμφανίζεται στην παραπάνω αρίθμηση. Ας πούμε ότι ο δείκτης για το π είναι i, δηλαδή $\pi = \pi_i$. Η ιδέα της διαγωνιοποίησης είναι να δώσουμε το δείκτη i για input στο π_i. Τότε το $\pi_i(i)$ σταματάει αν και μόνο αν το $\pi(i)$ σταματάει και αυτό συμβαίνει αν και μόνο αν το $\pi_i(i)$ δεν σταματάει. Αντίφαση. □

Τελικά πολλά άλλα προβλήματα είναι επίσης μη επιλύσιμα. Αν και το HP δεν είναι επιλύσιμο, μπορούμε να κατασκευάσουμε με μηχανιστικό τρόπο μια άπειρη λίστα όλων των προγραμμάτων, με την αντίστοιχη είσοδο για την οποία σταματούν. Αυτό δεν σημαίνει φυσικά ότι μπορούμε να επιλύσουμε το HP, γιατί αν για παράδειγμα το $\pi_k(n)$ δεν έχει εμφανισθεί στη λίστα μας, δεν ξέρουμε αν θα προστεθεί στη λίστα αργότερα ή αν δε θα εμφανισθεί ποτέ στη λίστα. Για να ακριβολογούμε λίγο περισσότερο δίνουμε τους παρακάτω ορισμούς.

Ορισμός 11.2. Ένα σύνολο S λέγεται αποκρίσιμο ή υπολογιστό ή επιλύσιμο (*decidable, computable, solvable*) αν και μόνο αν υπάρχει ένας αλγόριθμος που σταματάει ή μια υπολογιστική μηχανή που δίνει έξοδο "ναι" για κάθε είσοδο $a \in S$ και έξοδο "όχι" για κάθε είσοδο $a \notin S$.

Ορισμός 11.3. Ένα σύνολο S λέγεται καταγράψιμο (με μηχανιστική γεννήτρια) (*listable, effectively generatable*) αν και μόνο αν υπάρχει μια γεννήτρια διαδικασία ή μηχανή που καταγράφει όλα τα στοιχεία του S. Στην, πιθανώς άπειρη, λίστα εξόδου επιτρέπονται οι επαναλήψεις και δεν υπάρχει περιορισμός για την διάταξη των στοιχείων.

[4] Απόδειξη: Ας θεωρήσουμε το σύνολο των ολικών συναρτήσεων $\phi\colon \mathbb{N} \to \mathbb{N}$ και έστω $\phi_0, \phi_1, \phi_2, \ldots$ μια αρίθμηση τους (ολικές ονομάζονται οι συναρτήσεις που ορίζονται για κάθε $x \in \mathbb{N}$). Ορίζουμε μια συνάρτηση f ως εξής: $f(x) = \phi_x(x) + 1, \forall x \in \mathbb{N}$. Η f είναι προφανώς ολική συνάρτηση και επομένως θα αντιστοιχίζεται σε κάποιο δείκτη y στην παραπάνω αρίθμηση μας, δηλαδή $f = \phi_y$. Τότε όμως θα ισχύει ότι $\phi_y(y) = f(y) = \phi_y(y) + 1$ που είναι άτοπο. Επομένως το σύνολο των ολικών συναρτήσεων δεν είναι αριθμήσιμο.

Μερικές απλές ιδιότητες:

- Αν το S είναι αποκρίσιμο τότε και το $\overline{S}$ είναι αποκρίσιμο.
- Αν το S είναι αποκρίσιμο τότε το S είναι και καταγράψιμο.
- Αν το S και το $\overline{S}$ είναι καταγράψιμα τότε το S είναι αποκρίσιμο.
- Αν το S είναι καταγράψιμο με γνησίως αύξουσα διάταξη τότε το S είναι αποκρίσιμο.

11.2 Υπολογιστικά μοντέλα

Λόγω της θέσης των Church-Turing δεν χρειάζεται να καθορίσουμε ένα συγκεκριμένο υπολογιστικό μοντέλο για τη λύση κάποιου προβλήματος: όλα τα ντετερμινιστικά υπολογιστικά μοντέλα είναι ισοδύναμα μεταξύ τους, με την έννοια ότι αν ένα πρόβλημα λύνεται από κάποιο υπολογιστικό μοντέλο, τότε θα λύνεται και από οποιοδήποτε άλλο, με το πολύ πολυωνυμική απώλεια χρόνου. Μερικά υπολογιστικά μοντέλα είναι τα εξής:

- προγράμματα Pascal
- προγράμματα Pascal χωρίς αναδρομή (αφαίρεση αναδρομής με χρήση στοίβας)
- προγράμματα Pascal χωρίς αναδρομή και χωρίς άλλους τύπους δεδομένων εκτός από τους φυσικούς αριθμούς (επιτυγχάνεται με κωδικοποιήσεις)
- προγράμματα WHILE (μόνη δομή ελέγχου το WHILE)
- προγράμματα GOTO και IF
- Assembler-like RAM (random access machine), URM (universal register machine)
- SRM (single register machine) ένας καταχωρητής
- Μηχανή Turing (πρόσβαση μόνο σε μια κυψέλη ”cell” της ταινίας κάθε φορά)

Τα χαρακτηριστικά των παραπάνω μοντέλων είναι:

- ντετερμινιστική πολυπλοκότητα σε διακριτά βήματα
- πεπερασμένο σύνολο εντολών που εκτελούνται από επεξεργαστή
- απεριόριστη μνήμη

’Αλλα μοντέλα είναι:

- παραλλαγές από μηχανές Turing
- Thue: κανόνες επανεγγραφής (re-writing rules)

- Post: κανονικά συστήματα (normal systems)
- Church: λογισμός λ (λ-calculus)
- Curry: συνδυαστική λογική (combinatory logic)
- Markov: M. αλγόριθμοι
- Kleene: γενικά αναδρομικά σχήματα (general recursive schemes)
- Shepherdson-Sturgis, Elgott: URM, SRM, RAM, RASP
- Σχήματα McCarthy (if ... then ... else ... $\Rightarrow$ LISP)

Θεώρημα 11.4. *f είναι TM υπολογιστή ανν*

- *f είναι WHILE-υπολογιστή*
- *f είναι GOTO-υπολογιστή*
- *f είναι PASCAL-υπολογιστή*
- *f είναι μερικά αναδρομική (partial recursive)*

Παραλλαγές Μηχανών Turing που έχουν την ίδια υπολογιστική δυνατότητα, όχι όμως και αποδοτικότητα (*efficiency*) είναι:

- πολλές ταινίες, μνήμη πλέγματος (grid memory), μνήμη περισσοτέρων διαστάσεων
- μεγαλύτερο Σ
- πολλές παράλληλες κεφαλές
- μη ντετερμινιστικές μεταβάσεις
- μίας κατευθύνσεως, απείρου μήκους ταινία
- εγγραφή και κίνηση της κεφαλής σε κάθε βήμα

11.3 Υπολογιστική Πολυπλοκότητα

Μια άλλη (πιο μοντέρνα) ταξινόμηση προβλημάτων (γλωσσών, συνόλων) σε κλάσεις μπορεί να γίνει με κριτήριο το **ποσόν** των αγαθών (χρόνος, χώρος, επεξεργαστές, κ.ο.κ.) που χρειάζεται ένας βέλτιστος αλγόριθμος για να τα επιλύσει (αναγνωρίσει).

Για να χρησιμοποιήσουμε όμως το χρόνο, χώρο, κ.τ.λ. για μέτρο πολυπλοκότητας χρειαζόμαστε επακριβείς ορισμούς του υπολογιστικού μοντέλου καθώς και του μεγέθους που μετράμε. Το κόστος ενός αλγορίθμου δεν είναι φυσικά μια σταθερά τιμή αλλά είναι συνάρτηση του μεγέθους

n της εισόδου. Συνήθως μετράμε το κόστος της χειρότερης περίπτωσης για είσοδο μεγέθους n. Έτσι το κόστος του αλγορίθμου (n) είναι το max του κόστους του αλγορίθμου από όλες τις δυνατές εισόδους μεγέθους n.

Το κόστος $C(n)$, τώρα, ενός προβλήματος $\Pi(n)$ είναι το $\min((n))$ από όλους τους αλγορίθμους που λύνουν το πρόβλημα Π. Συνεπώς για να προσδιορίσουμε το κόστος $C(n)$ ενός προβλήματος $\Pi(n)$ χρειαζόμαστε ένα **άνω όριο** (upper bound) δηλαδή έναν αλγόριθμο που έχει κόστος $C(n)$ αλλά και ένα **κάτω όριο** (lower bound), δηλαδή μια απόδειξη ότι το καλύτερο δυνατό κόστος με το τρέχον μοντέλο είναι $C(n)$. Έτσι, π.χ., η χρονική πολυπλοκότητα ταξινόμησης με συγκρίσεις (όπου μοντέλο είναι πρόγραμμα Pascal, και μετράμε τον αριθμό συγκρίσεων) είναι $\Theta(n \log n)$.

Συνήθως ονομάζουμε αποδοτικό ένα αλγόριθμο αν ο χρόνος του είναι πολυωνυμικός ($n^{O(1)}$) ως προς το μέγεθος της εισόδου. **P** λέγεται η κλάση των προβλημάτων που λύνονται με ντετερμινιστικό αλγόριθμο σε χρόνο πολυωνυμικό. **NP** λέγεται η κλάση των προβλημάτων που λύνονται με μη ντετερμινιστικό αλγόριθμο σε χρόνο πολυωνυμικό. Λέμε ότι ένας μη ντετερμινιστικός αλγόριθμος λύνει ένα πρόβλημα Π εάν υπάρχει τουλάχιστον μια από τις δυνατές εκτελέσεις του Α που λύνει το Π. Προφανώς ισχύει **P** $\subseteq$ **NP** αλλά εδώ και τριάντα-πέντε χρόνια παραμένει άλυτο το πρόβλημα "**P** $\neq$ **NP**;". **NP**$-$πλήρη λέγονται τα δυσκολότερα προβλήματα της κλάσης **NP**. Αν πράγματι ισχύει **P** $\neq$ **NP**, τότε για τα **NP**$-$πλήρη προβλήματα δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου. **PSPACE** λέγεται η κλάση των προβλημάτων που λύνονται

με (ντετερμινιστικό ή μη ντετερμινιστικό αλγόριθμο) σε πολυωνυμικό χώρο (μνήμη).

Τέλος **NC** λέγεται η κλάση των προβλημάτων που λύνονται με αλγόριθμο που χρησιμοποιεί πολυλογαριθμικό χρόνο ($\log^{O(1)} n$) και πολυωνυμικό αριθμό επεξεργαστών.

Μερικά γνωστά προβλήματα σε αυτές τις κλάσεις:

- Στην κλάση **NP**: το πρόβλημα (SAT) ικανοποιησιμότητας τύπων της προτασιακής λογικής, το πρόβλημα (TSP) του πλανόδιου πωλητή, κ.τ.λ.
- Στην κλάση **PSPACE**: το πρόβλημα (QBF) αποτίμησης τύπων της κατηγορηματικής (boolean) λογικής, το πρόβλημα στρατηγικής σε διάφορα παιχνίδια, κ.τ.λ.
- Στην κλάση **NC**: το πρόβλημα (GAP) πρόσβασης (δηλαδή ύπαρξης μονοπατιού) σε ένα γράφο G μεταξύ δυο κόμβων.
- Το πρόβλημα ικανοποιησιμότητας τύπων της κατηγορηματικής λογικής είναι μη επιλύσιμο αλλά καταγράψιμο.

Ισχύει:

$$\textbf{NC} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} \subsetneq \textbf{REC} \subsetneq \textbf{R.E.}$$

Παρατήρηση:
REC = recursive = decidable
R.E. = recursively enumerable = listable

11.4 Αναγωγές μεταξύ προβλημάτων

Με τον όρο *αναγωγή* εννοούμε την μετατροπή ενός υπολογιστικού προβλήματος Π_A σε ένα άλλο πρόβλημα Π_B έτσι ώστε αν μπορούμε να λύσουμε το Π_B να μπορούμε να λύσουμε και το Π_A. Παρά το ότι υπάρχουν διάφορα είδη αναγωγών, εμείς θα ασχοληθούμε με μια απλή μορφή, κατάλληλη για προβλήματα απόφασης (όπου η απάντηση είναι "ναι'' ή "όχι''):

Ορισμός. Λέμε ότι ένα πρόβλημα απόφασης Π_A ανάγεται σε ένα πρόβλημα απόφασης Π, και γράφουμε $\Pi_A \leq \Pi_B$ αν υπάρχει μία υπολογίσιμη (computable) συνάρτηση h που απεικονίζει ένα (οποιοδήποτε) στιγμιότυπο (έγκυρη είσοδο) I_A του Π σε ένα στιγμιότυπο (έγκυρη είσοδο) I_B του Π τέτοια ώστε:

$$\text{answer}_{\Pi_A}(I_A) = \text{"yes''} \iff \text{answer}_{\Pi_B}(I_B) = \text{"yes''}$$

Δηλαδή, η απάντηση για το πρόβλημα Π_A με είσοδο I_A είναι "ναι'' *αν και μόνο αν* η απάντηση για το πρόβλημα Π_B με είσοδο $I_B = h(I_A)$ είναι "ναι''. [5]

Συχνά με τον όρο αναγωγή αναφερόμαστε τόσο στη συνάρτηση h όσο και στον *αλγόριθμο* που την υπολογίζει.

11.4.1 Αναγωγές πολυωνυμικού χρόνου

Εάν η συνάρτηση h μπορεί να υπολογιστεί αποδοτικά, δηλαδή σε πολυωνυμικό χρόνο, τότε λέγεται *αναγωγή πολυωνυμικού χρόνου*. Η ύπαρξη αναγωγής πολυωνυμικού χρόνου του Π_A στο Π_B συμβολίζεται ως εξής:

$$\Pi_A \leq^p \Pi_B$$

Συχνά χρησιμοποιούνται και οι συμβολισμοί $\Pi_A \leq_P \Pi_B$ και $\Pi_A \leq_m^p \Pi_B$.

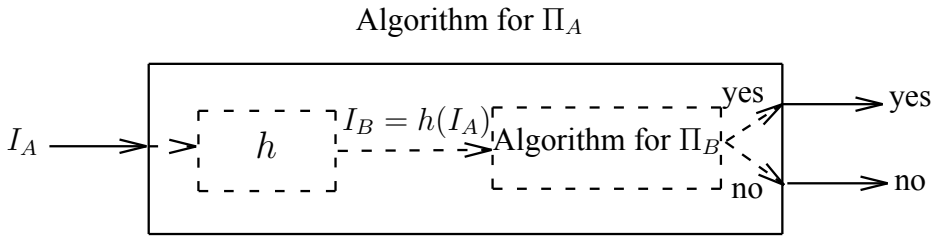
Παράδειγμα: Hamilton Cycle $\leq^p$ TSP Θα περιγράψουμε μια αναγωγή από το πρόβλημα Hamilton Cycle στο TSP.

Η ζητούμενη αναγωγή είναι, όπως αναφέρεται και πιο πάνω, μια συνάρτηση που απεικονίζει οποιοδήποτε στιγμιότυπο του Hamilton Cycle σε ένα στιγμιότυπο του TSP, ώστε η απάντηση για τα δύο στιγμιότυπα να είναι ίδια.

Για ευκολία, θα περιγράψουμε αναγωγή από το πρόβλημα Hamilton Cycle στο πρόβλημα U-TSP (Undirected TSP). [6] Θα δώσουμε τη διαδικασία μετατροπής ενός στιγμιοτύπου του Hamilton Cycle, που είναι απλά ένας μη κατευθυνόμενος γράφος G, σε ένα στιγμιότυπο του U-TSP, που

[5] Προσοχή στο 'αν και μόνο αν': το συνηθέστερο λάθος είναι να δείχνει κανείς την μία μόνο κατεύθυνση, δηλαδή το 'μόνο αν'. Αυτό όμως δεν μας εγγυάται ότι γνωρίζοντας την απάντηση για το I_B θα γνωρίζουμε και την απάντηση για το I_A. Πιο συγκεκριμένα, μας εξασφαλίζει μόνο ότι αν η απάντηση για το I_B είναι αρνητική τότε και η απάντηση για το I_A είναι αρνητική. Αν όμως η απάντηση για το I_B είναι καταφατική, μπορεί η απάντηση για το I_A να είναι είτε καταφατική είτε αρνητική.

[6] Αυτό δεν δημιουργεί πρόβλημα αφού οι μη κατευθυνόμενοι γράφοι μπορούν να θεωρηθούν συμμετρικοί κατευθυνόμενοι. Επομένως, το πρόβλημα U-TSP είναι ειδική περίπτωση του προβλήματος TSP.



Σχήμα 11.1: Αναγωγή από Π_A σε Π_B + αλγόριθμος για $\Pi_B \Rightarrow$ αλγόριθμος για Π_A.

είναι ένας μη κατευθυνόμενος γράφος G' με βάρη στις ακμές του, και ένα όριο κόστους (θυμηθείτε ότι μιλάμε για προβλήματα απόφασης: το ερώτημα στο πρόβλημα U-TSP είναι αν υπάρχει κύκλος Hamilton στον G' με κόστος $\leq B$).

Κατασκευή: Ο γράφος G' έχει το ίδιο πλήθος κόμβων με τον G, έστω n, αλλά είναι πλήρης. Σε κάθε ακμή του G' που υπήρχε και στον αρχικό γράφο G δίνουμε βάρος 1, ενώ στις υπόλοιπες δίνουμε βάρος 2. Θέτουμε σαν όριο κόστους $B = n$.

Απόδειξη ορθότητας: Θα δείξουμε ότι υπάρχει κύκλος Hamilton στον αρχικό γράφο G **αν και μόνο αν** υπάρχει κύκλος Hamilton στον G' με κόστος $\leq n$:
'μόνο αν': αν υπάρχει κύκλος Hamilton στον G τότε ο ίδιος κύκλος στον G' αποτελείται από ακμές βάρους 1, οπότε το συνολικό κόστος του κύκλου είναι n.
'αν': αν υπάρχει κύκλος Hamilton στον G' με κόστος $\leq n$, αυτός θα πρέπει να αποτελείται αποκλειστικά από ακμές βάρους 1, αφού κάθε κύκλος Hamilton έχει n ακμές. Επομένως αυτός ο κύκλος υπάρχει και στον G.

Χρειάζεται επιπλέον να δείξουμε ότι η αναγωγή γίνεται σε πολυωνυμικό χρόνο. Πράγματι, ένα πέρασμα του πίνακα γειτνίασης (ή των λιστών γειτνίασης) και αντικατάσταση των (μη διαγωνίων) 0 με 2 αρκεί, επομένως ο απαιτούμενος χρόνος είναι $O(n^2)$ ($O(m)$ με λίστες), δηλαδή πολυωνυμικός ως προς το μέγεθος της εισόδου.

11.4.2 Αναγωγές: δύο τρόποι χρήσης

Όπως είδαμε πιο πάνω, μια ορθή αναγωγή έχει την ιδιότητα οι απαντήσεις για τα δύο στιγμιότυπα (I_A του προβλήματος Π_A και I_B του προβλήματος Π_B) να ταυτίζονται. Αυτό σημαίνει ότι αν έχουμε έναν αποδοτικό αλγόριθμο επίλυσης του Π_B και μια αποδοτική αναγωγή από το Π_A στο Π_B τότε παίρνουμε έναν αποδοτικό αλγόριθμο επίλυσης του Π_A, όπως φαίνεται και στο Σχ. 11.1.

Η ιδιότητα αυτή των αναγωγών μπορεί να χρησιμοποιηθεί με δύο τρόπους, είτε με "θετικό" τρόπο, **προς επίλυση προβλήματος**, είτε με "αρνητικό", προς **απόδειξη δυσκολίας προβλήματος**.

Πιο συγκεκριμένα, μια αναγωγή πολυωνυμικού χρόνου του Π_A στο Π_B μπορεί να χρησιμοποιηθεί είτε για την εύρεση αποδοτικού αλγορίθμου για το πρόβλημα Π_A (εάν γνωρίζουμε αλγόριθμο για το Π_B) είτε για την απόδειξη δυσκολίας του Π_B (εάν διαθέτουμε απόδειξη δυσκολίας για το Π_A). Ο λόγος για το δεύτερο είναι ότι αν γνωρίζουμε (με βεβαιότητα, ή κάτω από βάσιμες

υποθέσεις) ότι το Π_A δεν διαθέτει πολυωνυμικό αλγόριθμο, τότε συνάγεται (με (με βεβαιότητα, ή κάτω από τις ίδιες υποθέσεις) ότι και το Π_B δεν διαθέτει πολυωνυμικό αλγόριθμο (αλλιώς θα διέθετε και το Π_A (αντίφαση).

Παραδείγματα:

- Η επίλυση του Προβλήματος του Βαρκάρη με αναγωγή στο πρόβλημα Reachability (βλ. Ασκήσεις) υπάγεται στην πρώτη περίπτωση.
- Οι αποδείξεις **NP**-πληρότητας υπάγονται στη δεύτερη περίπτωση και παρουσιάζονται πιο αναλυτικά παρακάτω.

Αποδείξεις NP-πληρότητας με χρήση αναγωγής πολυωνυμικού χρόνου

Ορισμός. Ένα πρόβλημα απόφασης Π λέγεται **NP-δύσκολο (NP-hard)** αν για κάθε πρόβλημα Π' $\in$ **NP** ισχύει Π' $\leq^p$ Π. Αν επιπλέον Π $\in$ **NP** τότε το Π λέγεται **NP-πλήρες (NP-complete)**.

Ισχύει ότι αν ένα πρόβλημα Π είναι **NP**-δύσκολο τότε αν υπήρχε πολυωνυμικός αλγόριθμος S για το πρόβλημα αυτό θα συνεπαγόταν ότι **P**=**NP** (γιατί θα μπορούσαμε να 'συνθέσουμε' τον αλγόριθμο της αναγωγής Π' $\leq^p$ Π με τον υποθετικό αλγόριθμο S και να πάρουμε πολυωνυμικό αλγόριθμο για το Π', όπου Π' οποιοδήποτε πρόβλημα στην κλάση **NP**). Παρ'ότι αυτό είναι ακόμη ανοιχτό πρόβλημα, θεωρείται εξαιρετικά απίθανο να συμβαίνει, οπότε αν γνωρίζουμε ότι ένα πρόβλημα είναι **NP**-δύσκολο συμπεραίνουμε ότι είναι μάλλον αδύνατο να έχει πολυωνυμικό αλγόριθμο. Τα ίδια ισχύουν βέβαια αν ένα πρόβλημα είναι **NP**-πλήρες (αφού **NP**-πλήρες είναι ειδική περίπτωση **NP**-δύσκολου).

Το παρακάτω θεώρημα δίνει τον (συνήθη) τρόπο απόδειξης ότι ένα πρόβλημα είναι **NP**-δύσκολο (ή και **NP**-πλήρες).

Θεώρημα. Αν $\Pi_A \leq^p \Pi_B$ και το Π_A είναι **NP**-πλήρες τότε το Π_B είναι **NP**-δύσκολο. Αν επιπλέον $\Pi_B \in$ **NP** τότε το Π_B είναι **NP**-πλήρες.

Η ιδέα της απόδειξης (δεν θα την παρουσιάσουμε αναλυτικά εδώ) είναι ότι η σύνθεση αναγωγών πολυωνυμικού χρόνου είναι αναγωγή πολυωνυμικού χρόνου, επομένως ισχύει ότι:

$$\Pi \leq^p \Pi_A \text{ και } \Pi_A \leq^p \Pi_B \Longrightarrow \Pi \leq^p \Pi_B$$

Παράδειγμα: η αναγωγή που δώσαμε παραπάνω Hamilton Cycle $\leq^p$ TSP, επειδή γνωρίζουμε ότι το Hamilton Cycle είναι **NP**-πλήρες οδηγεί στο συμπέρασμα ότι και το πρόβλημα απόφασης TSP είναι **NP**-δύσκολο. Επειδή επιπλέον το πρόβλημα απόφασης TSP ανήκει στην κλάση **NP** (αποδεικνύεται σχετικά εύκολα: μια πιθανή λύση επαληθεύεται σε πολυωνυμικό χρόνο) συμπεραίνουμε ότι το πρόβλημα απόφασης TSP είναι **NP**-πλήρες.

11.5 Διαδραστικό υλικό - Σύνδεσμοι

- Οι κλάσεις πολυπλοκότητας που υπάρχουν σήμερα, είναι πλέον τόσο πολλές που χρειάζονται μια γενικευμένη κατηγοριοποίηση. Εδώ μπορείτε να βρείτε έναν "ζωολογικό κήπο"

με τις περισσότερες κλάσεις πολυπλοκότητας που υπάρχουν.

11.6 Ασκήσεις

1. *Κάλυψη κόμβων* σε έναν γράφο $G(V, E)$ λέγεται ένα υποσύνολο V' των κόμβων του γράφου τέτοιο ώστε κάθε ακμή του γράφου έχει έναν τουλάχιστον κόμβο της στο σύνολο, δηλαδή $\forall\{u, v\} \in E : u \in V' \vee v \in V'$. *Ανεξάρτητο σύνολο* σε έναν γράφο $G(V, E)$ λέγεται ένα υποσύνολο κόμβων του γράφου V' που δεν έχουν καμμία ακμή μεταξύ τους, δηλαδή $\forall u, v \in V' : \{u, v\} \notin E$.

   (α) Αποδείξτε ότι ένα υποσύνολο V' των κόμβων του γράφου είναι κάλυψη κόμβων αν και μόνο αν το σύνολο $V \setminus V'$ είναι ανεξάρτητο.

   (β) *Κλίκα* σε έναν γράφο $G(V, E)$ λέγεται ένα υποσύνολο κόμβων V' του γράφου που συνδέονται όλοι ανά δύο μεταξύ τους, δηλαδή $\forall u, v \in V' : (u, v) \in E$.

   Περιγράψτε αναγωγή από το πρόβλημα Vertex Cover (δίνεται γράφος και αριθμός k, υπάρχει κάλυψη κόμβων μεγέθους το πολύ k στο γράφο;) στο πρόβλημα Clique (δίνεται γράφος και αριθμός m, υπάρχει κλίκα μεγέθους τουλάχιστον m στο γράφο;). Τι συμπέρασμα προκύπτει εάν γνωρίζουμε ότι το πρόβλημα Vertex Cover είναι NP-πλήρες; Τι έχετε να πείτε για το πρόβλημα Independent Set (δίνεται γράφος και αριθμός t, υπάρχει ανεξάρτητο σύνολο μεγέθους τουλάχιστον t στο γράφο;);

2. Δώστε αναγωγή από το Vertex Cover (βλ. άσκηση 1) στο Set Cover: "Δίνεται ένα πεπερασμένο σύνολο U και ένα σύνολο $S = \{S_1, \ldots, S_m\}$ υποσυνόλων του U. Ζητείται το μικρότερο δυνατό υποσύνολο C του S, τέτοιο ώστε το C να καλύπτει όλα τα στοιχεία του U, δηλαδή $\cup_{S_i \in C} S_i = U$.

3. Γενικεύοντας το Πρόβλημα του Βαρκάρη (βλ. και ασκήσεις γράφων) μας δίνονται n αντικείμενα και χωρητικότητα βάρκας k (εκτός του βαρκάρη). Οι ασυμβατότητες μεταξύ των αντικειμένων δίνονται από μια σχέση R: $R(a_i, a_j)$ σημαίνει ότι απαγορεύεται τα a_i και a_j να βρίσκονται στην ίδια όχθη αφύλακτα (επιτρέπεται όμως να είναι στην ίδια όχθη με τον βαρκάρη παρόντα, ή μέσα στη βάρκα πηγαίνοντας από την μια όχθη στην άλλη).

   (α) Έχει πάντοτε λύση το Πρόβλημα του Βαρκάρη;

   (β) Επεκτείνετε την αναγωγή που περιγράψαμε στο μάθημα σε μια αναγωγή του Γενικευμένου Προβλήματος του Βαρκάρη στο Πρόβλημα Προσβασιμότητας σε γράφο (Reachability).

   (γ) Διατυπώστε έναν αλγόριθμο επίλυσης του Γενικευμένου Προβλήματος του Βαρκάρη. Ποια είναι η πολυπλοκότητα του αλγορίθμου σας;

4. Δείξτε ότι η αναγωγή πολυωνυμικού χρόνου είναι μεταβατική σχέση, δηλαδή αν $\Pi_A \leq^p \Pi_B$ και $\Pi_B \leq^p \Pi_\Gamma$, τότε και $\Pi_A \leq^p \Pi_\Gamma$.

5. Ορίζουμε το πρόβλημα της Διαδρομής Ίππου ως εξής: είσοδος είναι οι διαστάσεις ορθογώνιας σκακιέρας n, m, καθώς και κάποια απαγορευμένα τετράγωνα που δίνονται σαν ζεύγη (i, j). Θέλουμε να βρούμε αν ένα άλογο (ίππος) που ξεκινά από το τετράγωνο (x_1, y_1)

μπορεί να φτάσει στο τετράγωνο (x_2, y_2) χωρίς να πατήσει πάνω σε απαγορευμένα τετράγωνα, και αν ναι με ποιον τρόπο.

Σημείωση: η κίνηση του αλόγου είναι 2 τετράγωνα προς μία κατεύθυνση, οριζόντια ή κάθετα, και μετά 1 τετράγωνο αριστερά ή δεξιά, κάθετα στην αρχική κατεύθυνση. Το άλογο επιτρέπεται να περάσει πάνω από απαγορευμένα τετράγωνα, αλλά όχι να σταματήσει σε αυτά.

(α) Έχει λύση το πρόβλημα για τη σκακιέρα 4×5 με απαγορευμένο τετράγωνο το $(2, 3)$, αρχικό τετράγωνο $(1, 1)$ και τελικό τετράγωνο $(3, 1)$; Αν ναι, ζωγραφίστε τη διαδρομή στο παρακάτω σχήμα. Αν όχι εξηγήστε γιατί.

	1	2	3	4	5
1	♞				
2			×		
3					
4					

(β) Περιγράψτε με σαφήνεια έναν όσο το δυνατόν πιο αποδοτικό αλγόριθμο που να επιλύει το πρόβλημα της Διαδρομής Ίππου στη γενική περίπτωση (δηλ. για οποιαδήποτε σκακιέρα δοθεί). Ποια είναι η πολυπλοκότητα του αλγορίθμου σας σε σχέση με τα n, m και το πλήθος k των απαγορευμένων τετραγώνων;

(γ) Θεωρήστε επιπλέον την παραλλαγή όπου δίνεται επιπλέον ένας ακέραιος k και ζητείται να βρείτε εάν υπάρχει διαδρομή από το αρχικό στο τελικό τετράγωνο με k το πολύ κινήσεις. Βρείτε την απάντηση για την παραπάνω σκακιέρα, για $k = 3$.

Πώς πρέπει να τροποποιήσετε / συμπληρώσετε τον αλγόριθμό σας ώστε να επιλύει αυτή την εκδοχή στη γενική περίπτωση; Αλλάζει η πολυπλοκότητα του αλγορίθμου σας; Εξηγήστε.

6. Δείξτε ότι **P** $\subseteq$ **PSPACE**.

7. (α) Ορίστε τις κλάσεις Reg (Regular), P (Polynomial Time), CF (Context Free), NP (Nondeterministic Polynomial Time), REC (Recursive), RE (Recursively Enumerable).

   (β) Σχεδιάστε τις παραπάνω κλάσεις σε διάγραμμα Hasse, με σύντομη αιτιολόγηση.

   (γ) Ορίστε το πρόβλημα Κύκλου Hamilton. Σε ποια από τις παραπάνω κλάσεις ανήκει και γιατί;

   (δ) Αντιστοιχήστε προβλήματα με κλάσεις πολυπλοκότητας (με σύντομη αιτιολόγηση):

   (i) Προβλήματα:

   - Ελάχιστο Συνδετικό Δένδρο
   - $L_1 = \{w \in \{a, b\}^* \mid w$ περιέχει ίσο αριθμό a και $b\}$
   - Ισοδυναμία αυτομάτων

- $L_2 = \{ww \mid w \in \{a, b, c, d\}^*\}$
- Satisfiability (SAT)
- $L_3 = \{w \in \{0, 1\}^* \mid w$ περιέχει ακριβώς τρία $'0'\}$
- Halting Problem

(ii) Κλάσεις: NP, REC, Context Free, Regular, RE, Context Sensitive, P

Βιβλιογραφία

[1] Christos Papadimitriou. “Computational Complexity”. Addison Wesley, 1994

[2] Sanjeev Arora, Boaz Barak. “Computational Complexity: A Modern Approach”. Cambridge University Press, 2009

[3] Oded Goldreich. “Computational Complexity: A Conceptual Perspective”, Cambridge University Press, 2008

[4] Martin D. Davis, Ron Sigal, Elaine J. Wayuker, “Computability, Complexity, and Languages”, 2nd edition, Academic Press Professional, Inc. San Diego, CA, USA, 1994.

[5] J.E. Hopcroft and J.D. Ullman. “Introduction to Automata Theory, Languages and Computation”, Addison Wesley Longman, 2007.

[6] M. Sipser. “Introduction to the Theory of Computation”, International Thomson Publishing, 1996.

[7] D. C. Kozen. “Automata and Computability” (Undergraduate Texts in Computer Science), Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1997.

[8]