Εισαγωγή στην Επιστήμη των Υπολογιστών

4ο εξάμηνο Σ.Η.Μ.Μ.Υ. & Σ.Ε.Μ.Φ.Ε.

http://www.corelab.ece.ntua.gr/courses/

1η ενότητα: Εισαγωγή, Αλγόριθμοι

Στάθης Ζάχος
Άρης Παγουρτζής

Επιμέλεια: Πάνος Χείλαρης, Βαγγέλης Μπαμπάς, Γεωργία Καούρη

Περιεχόμενα

1 Εισαγωγή **1**

1.1 Κλάδοι Επιστήμης των Υπολογιστών 2

1.2 Επανάληψη, επαγωγή, αναδρομή 3

1.2.1 Επανάληψη (Iteration) 3

1.2.2 Αναδρομή (Recursion) 3

1.2.3 Επαγωγή (Induction) 5

1.2.4 Μερική και ολική ορθότητα 5

1.3 Δομημένος προγραμματισμός και modularity 6

1.4 Παράλληλες, ταυτόχρονες, κατανεμημένες διεργασίες 7

1.4.1 Δίκτυα ταξινόμησης 8

1.5 Παράδειγμα: ταξινόμηση treesort με binary search tree 8

1.6 Το θεώρημα τεσσάρων χρωμάτων (four color theorem) 9

2 Αλγόριθμοι **13**

2.1 Μαθηματικοί Συμβολισμοί 17

2.2 Αλγόριθμοι και πολυπλοκότητα 20

2.3 Εύρεση μέγιστου κοινού διαιρέτη 20

2.3.1 Ένας απλός αλγόριθμος για το gcd 20

2.3.2 Αλγόριθμος με αφαιρέσεις για το gcd 21

2.3.3 Αλγόριθμος του Ευκλείδη 21

Περιεχόμενα

3 Αλγόριθμοι γράφων . . . 23

3.1 Γραφήματα . . . 23

3.1.1 Γενικά . . . 23

3.1.2 Υπογράφος . . . 25

3.1.3 Βαθμός κορυφής . . . 26

3.1.4 Δρόμος - Μονοπάτι - Κύκλος . . . 27

3.1.5 Παράσταση Γράφου . . . 28

3.1.6 Προσανατολισμένος Γράφος . . . 30

3.1.7 Συνεκτικός Γράφος . . . 31

3.2 Δέντρα . . . 33

3.2.1 Γενικά . . . 33

3.3 Ελάχιστο συνδετικό δέντρο (MST) . . . 36

3.4 Το πρόβλημα των συντομότερων μονοπατιών . . . 41

3.5 Διάσχιση γράφων . . . 44

3.5.1 Γενικά . . . 44

3.5.2 Αναζήτηση κατά πλάτος (Breadth First Search) . . . 44

3.5.3 Αναζήτηση κατά βάθος (Depth First Search) . . . 46

3.6 Πολυπλοκότητα γραφοθεωρητικών προβλημάτων . . . 47

Περιεχόμενα

4 Αυτόματα και τυπικές γλώσσες **49**

4.1 Εισαγωγή . . . 49

4.2 Η ιεραρχία γραμματικών κατά Chomsky . . . 51

4.3 Πεπερασμένα αυτόματα και κανονικές παραστάσεις . . . 52

4.3.1 Ντετερμινιστικά πεπερασμένα αυτόματα . . . 52

4.3.2 Μη ντετερμινιστικά πεπερασμένα αυτόματα . . . 55

4.3.3 Κανονικές παραστάσεις . . . 59

4.3.4 Παρόμοια αυτόματα . . . 61

4.3.5 Ελαχιστοποίηση DFA . . . 61

4.3.6 Pumping Lemma για κανονικά σύνολα . . . 63

4.4 Γραμματικές χωρίς συμφραζόμενα και αυτόματα στοίβας . . . 65

4.4.1 Συντακτικά δένδρα . . . 65

4.4.2 Απλοποίηση και κανονικές μορφές . . . 68

4.4.3 Αυτόματα στοίβας . . . 69

4.5 Γενικές γραμματικές . . . 71

4.6 Γραμματικές με συμφραζόμενα . . . 72

4.7 Κανονικές Γραμματικές . . . 73

Περιεχόμενα

5 Λογική στην Επιστήμη των Υπολογιστών **77**

5.1 Προτασιακή Λογική 77

5.2 Κατηγορηματικός Λογισμός 78

5.3 Πρωτοβάθμια Λογική 79

6 Υπολογιστικά μοντέλα **83**

6.1 Μηχανές Turing 83

6.2 Μηχανή τυχαίας προσπέλασης (Random Access Machine) . . . 86

7 Υπολογισιμότητα (Computability) **89**

7.1 Ιστορία - Εισαγωγή 89

7.2 Υπολογιστικά μοντέλα 92

8 Υπολογιστική Πολυπλοκότητα **95**

9 Λογικός Προγραμματισμός **97**

10 Συναρτησιακός Προγραμματισμός **135**

Περιεχόμενα

11 Αντικειμενοστρεφής Προγραμματισμός **145**
11.1 Object Oriented Programming (Java) 145
11.2 Java 151

12 Συστήματα Αρίθμησης - Δυαδική Παράσταση Αριθμών **187**
12.1 Αριθμητικά Συστήματα 187
12.2 Μετατροπή αριθμών από ένα αριθμητικό σύστημα σε άλλο . . . 189
12.3 Πράξεις θετικών ακεραίων αριθμών 193
12.4 Παράσταση ακεραίων αριθμών 194
12.5 Πρόσθεση προσημασμένων ακεραίων αριθμών 198
12.6 Παράσταση πραγματικών αριθμών 200
12.7 Ασκήσεις 211

13 Δομή και Λειτουργία ενός Απλού Υπολογιστή **223**
13.1 Εισαγωγή 223
13.2 Δομικά στοιχεία του υπολογιστή 224
13.3 Λειτουργία του Υπολογιστή 228

14 Συμβολική Γλώσσα (ASSEMBLY) του ΕΚΥ **229**
14.1 Εντολές γλώσσας ASSEMBLY του ΕΚΥ 229
14.2 Προγραμματισμός Η/Υ 241
14.3 Ασκήσεις 246

Computer Science

Informatics

Computing Science

Dijkstra

Τι μπορεί να μηχανοποιηθεί και μάλιστα αποδοτικά ;

Υπολογιστές: ταχύτητα - ακρίβεια

Knowledge representation

Modeling

Abstraction:

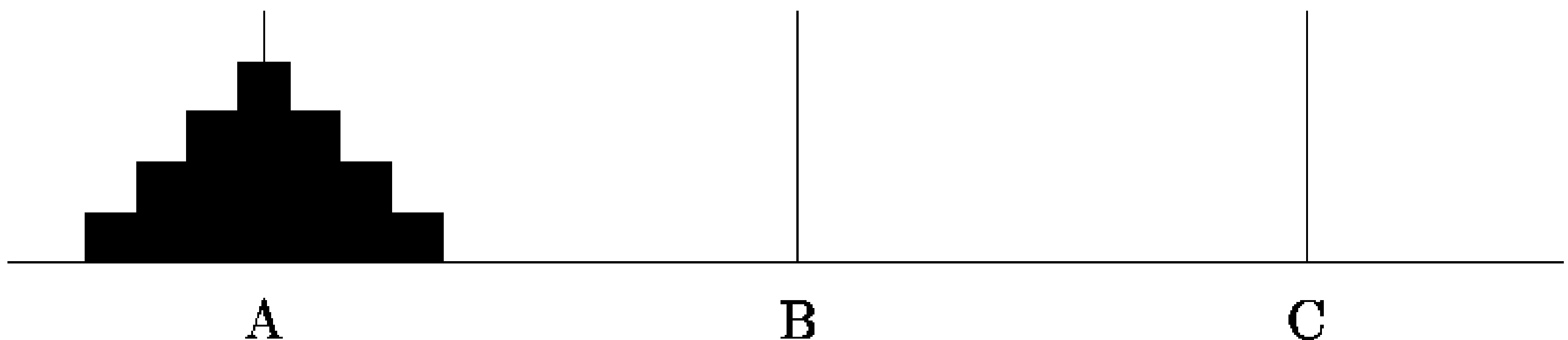
- (Directed) Graphs
- (Formal) Logic

- Data Models
- Data Structures
- Algorithms

Κλάδοι Επιστήμης Υπολογιστών

1. Αλγόριθμοι και Δομές Δεδομένων
2. Γλώσσες Προγραμματισμού και Μεταγλωττιστές
3. Αρχιτεκτονική Υπολογιστών και Δικτύων (hardware)
4. Αριθμητικοί και Συμβολικοί Υπολογισμοί
5. Λειτουργικά - Παράλληλα - Κατανεμημένα Συστήματα
6. Μεθοδολογία - Τεχνολογία Λογισμικού (software)
7. Βάσεις Δεδομένων και Διαχείριση Πληροφοριών
8. Τεχνητή Νοημοσύνη και Ρομποτική
9. Επικοινωνία ανθρώπου - υπολογιστή. Πολυμέσα
10. Δίκτυα Επικοινωνιών - Ευφυή Δίκτυα - Διαδίκτυο

Iteration-Recursion-Induction

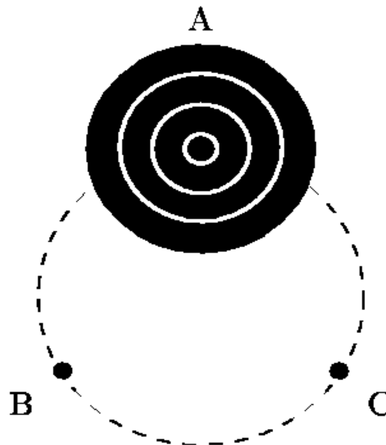


Σχήμα 1.2: Πύργοι του Ανόι $(n = 4)$.

Πύργοι Ανόι (Hanoi Towers)

```
procedure move_anoi(n from X to Y using Z)
begin
   if n = 1 then move top disk from X to Y
   else begin
            move_anoi(n−1 from X to Z using Y);
            move top disk from X to Y;
            move_anoi(n−1 from Z to Y using X)
         end
end
```

1. *μετακίνησε κατά την θετική φορά τον μικρότερο δίσκο·*
2. *κάνε την μοναδική επιτρεπτή κίνηση που δεν αφορά τον μικρότερο δίσκο.*



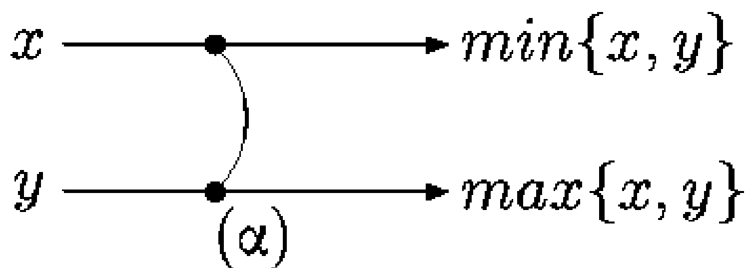
Μερική και Ολική Ορθότητα

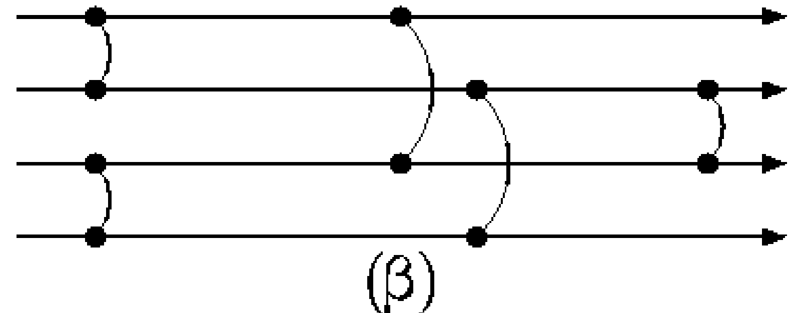
- **Λειτουργική σημασιολογία** (operational semantics). Περιγράφει την υπολογιστική ακολουθία που εκτελείται.
- **Δηλωτική σημασιολογία** (denotational semantics). Ορίζει μόνο τη συνάρτηση εισόδου-εξόδου.
- **Αξιωματική σημασιολογία** (axiomatic semantics). Περιγράφει τις σχετικές ιδιότητες που πρέπει απαραίτητα να ικανοποιούνται από την είσοδο και την έξοδο.

Στην περίπτωση που, αντί για κατηγορηματικό λογισμό και φυσικούς αριθμούς, χρησιμοποιούμε πράξεις και ιδιότητες (αξιώματα) κάποιας άλλης συγκεκριμένης αλγεβρικής δομής η αξιωματική σημασιολογία ονομάζεται συνήθως **αλγεβρική σημασιολογία** (algebraic semantics).

- Structured Programming
- Modularity
- Parallel Systems
- Concurrent Systems
- Distributed Systems

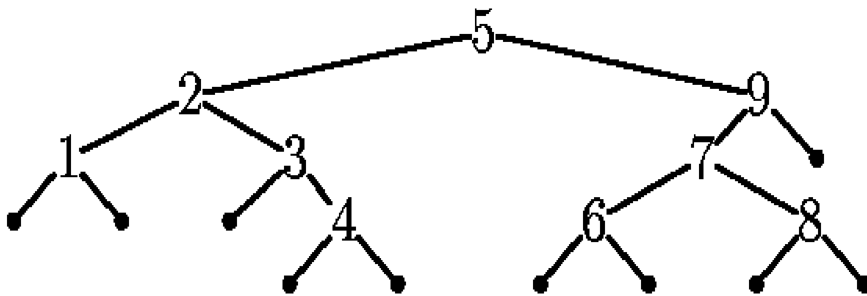
Δίκτυα Ταξινόμησης (Sorting Networks)





Σχήμα 1.4: (α) Συγκριτής (β) Δίκτυο ταξινόμησης 4 εισόδων

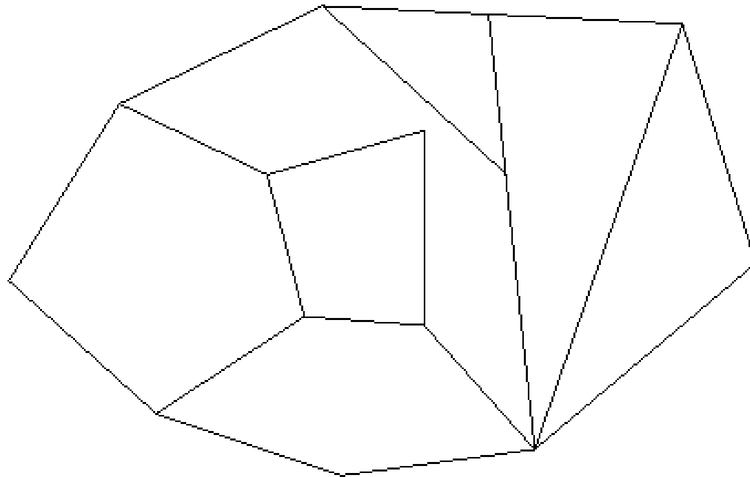
Treesort με χρήση Binary Search Tree




```
procedure inorder(t: treenode)
begin
    if t is not empty then
    begin
        inorder(left branch of t);
        write(element at t);
        inorder(right branch of t)
    end
end
```

Four Color Theorem (1852-1977)

Πόσα χρώματα απαιτούνται για τον χρωματισμό όλων των χωρών, ούτως ώστε χώρες που συνορεύουν (δηλαδή έχουν γραμμή, όχι απλώς σημείο για κοινό σύνορο) να έχουν διαφορετικό χρώμα;



Σχήμα 1.6: Επίπεδος χάρτης

Appel

Haken

Αλγόριθμοι

Η έννοια αλγόριθμος είναι πρωταρχική έννοια της θεωρίας αυτής. Γι' αυτό δεν ορίζεται. Εδώ δίνουμε μία άτυπη εξήγηση.

Αλγόριθμος είναι ένα πεπερασμένο σύνολο κανόνων, οι οποίοι περιγράφουν μία μέθοδο (που αποτελείται από μία σειρά υπολογιστικών διεργασιών) για να λυθεί ένα συγκεκριμένο πρόβλημα. Τα αντικείμενα πάνω στα οποία επενεργούν αυτές οι διεργασίες λέγονται **δεδομένα** (*data*).

Ο αλγόριθμος χαρακτηρίζεται από τα παρακάτω πέντε στοιχεία:

- Κάθε εκτέλεση είναι *πεπερασμένη*, δηλαδή τελειώνει ύστερα από έναν πεπερασμένο αριθμό διεργασιών ή βημάτων (*finiteness*).
- Κάθε κανόνας του ορίζεται επακριβώς και η αντίστοιχη διεργασία είναι συγκεκριμένη (*definiteness*).
- Έχει μηδέν ή περισσότερα μεγέθη *εισόδου* που δίδονται εξαρχής, πριν αρχίσει να εκτελείται ο αλγόριθμος (*input*).
- Δίδει τουλάχιστον ένα μέγεθος σαν αποτέλεσμα (*έξοδο-output*) που εξαρτάται κατά κάποιο τρόπο απ'τις αρχικές εισόδους.
- Είναι *μηχανιστικά* αποτελεσματικός, δηλαδή όλες οι διαδικασίες που περιλαμβάνει μπορούν να πραγματοποιηθούν με ακρίβεια και σε πεπερασμένο χρόνο «με μολύβι και χαρτί» (*effectiveness*).

Πολυπλοκότητα

Στην πράξη, το ενδιαφέρον δεν σταματά στο να βρεθεί ένας αλγόριθμος που επιλύει ένα πρόβλημα, αλλά προχωρά στη μελέτη των μετρήσιμων ιδιοτήτων που χαρακτηρίζουν την **αποδοτικότητα** μιας υπολογιστικής μεθόδου. Αυτά τα μεγέθη (αγαθά-*resources*) είναι π.χ. ο χρόνος υπολογισμού, ο χώρος σε μνήμη υπολογιστή, ο αριθμός προκαταρκτικών διαδικασιών που προαπαιτούνται και είναι αυτά που ορίζουν την **πολυπλοκότητα** (*complexity*) του αλγορίθμου.

Ονομάζουμε **πολυπλοκότητα ενός προβλήματος** την πολυπλοκότητα ενός **βέλτιστου** (*optimal*) αλγορίθμου που λύνει το πρόβλημα.

Ο τρόπος που προσεγγίζει κανείς την πολυπλοκότητα οδηγεί σ'έναν αρχικό διαχωρισμό της έννοιας **πολυπλοκότητα**, σε **συγκεκριμένη** (*concrete*) πολυπλοκότητα και **μη συγκεκριμένη, περισσότερο θεωρητική** (*abstract*) **πολυπλοκότητα**.

Ο κλάδος της συγκεκριμένης (*concrete*) πολυπλοκότητας ασχολείται με την περιγραφή συστηματικών τεχνικών αξιολόγησης των μετρήσιμων αγαθών (*resources*) που χαρακτηρίζουν την αποδοτικότητα ενός συγκεκριμένου αλγορίθμου (κυρίως του χρόνου και του χώρου που απαιτούνται απ'τον αλγόριθμο) σ'ένα συγκεκριμένο **υπολογιστικό μοντέλο**.

Είδη πολυπλοκότητας

Η συμπεριφορά του αλγορίθμου μελετάται κυρίως σε δύο περιπτώσεις. Στην **χειρότερη** (*worst case*) και στην **μέση** (*average case*), μιας δεδομένης κατανομής πιθανών **στιγμιοτύπων** (*instances*) του προβλήματος. Μια άλλη ανάλυση ενδιαφέρεται για την **μακροπρόθεσμη απόσβεση** (*amortization*) επαναληπτικής χρήσης ενός αλγορίθμου. Η μελέτη της πολυπλοκότητας ενός αλγορίθμου μας επιτρέπει πολλές φορές να αποφανθούμε αν αυτός είναι **βέλτιστος** (*optimal*) για το συγκεκριμένο πρόβλημα. Αυτό προϋποθέτει ότι έχουμε **τα άνω (με αλγόριθμο)** και **κάτω (με απόδειξη) φράγματα του χρόνου** (ή και **του χώρου**) που επαρκούν και απαιτούνται για την επίλυση ενός προβλήματος και επίσης προϋποθέτει ότι αυτά ταυτίζονται.

Πολυπλοκότητα: κόστος αλγορίθμου / κόστος προβλήματος

Το κόστος ενός αλγορίθμου ορίζεται με τη βοήθεια της παρακάτω συνάρτησης:

$$\text{κόστος αλγορίθμου}(n) = \max_{\substack{\text{για όλες τις δυνα-}\\\text{τές εισόδους με-}\\\text{γέθους } n}} \{\text{κόστος αλγορίθμου για είσοδο } x\}$$

Και το κόστος ενός προβλήματος, με τη βοήθεια της συνάρτησης:

$$\text{κόστος προβλήματος } (n) = \min_{\substack{\text{για όλους τους}\\\text{αλγόριθμους A}\\\text{που επιλύουν το}\\\text{πρόβλημα}}} \{\text{κόστος του αλγορίθμου } A(n)\}$$

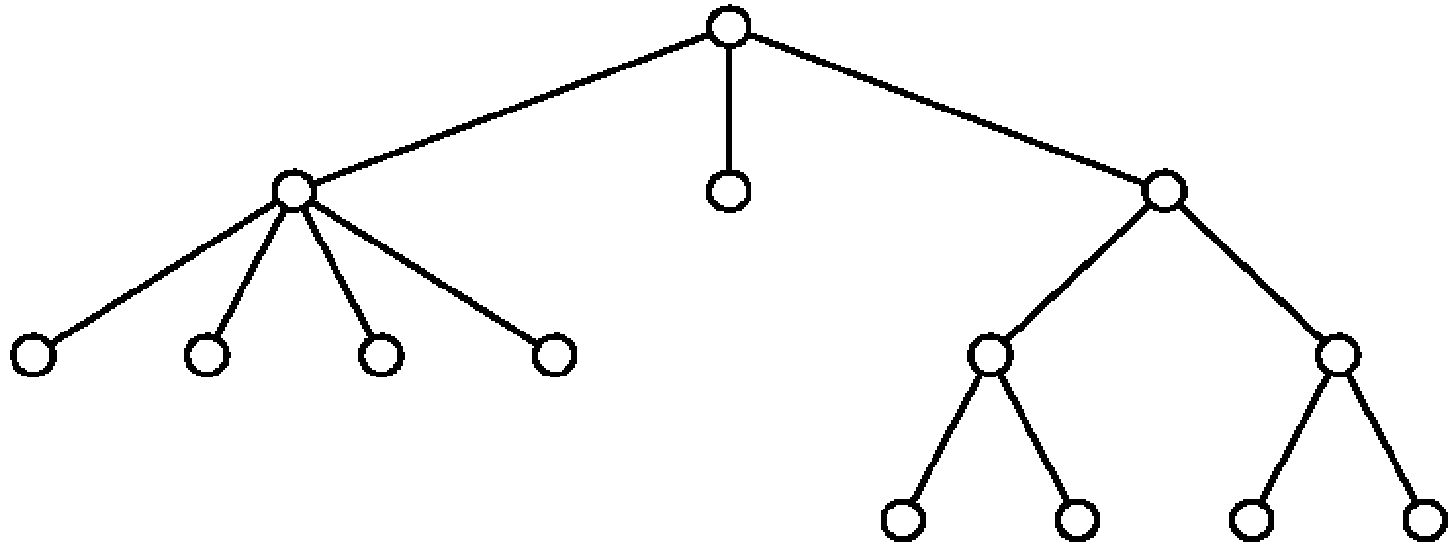
Determinism

Ένας αλγόριθμος είναι **ντετερμινιστικός** (*deterministic*) ή **μη ντετερμινιστικός** (*nondeterministic*). Ο ντετερμινιστικός αλγόριθμος διακρίνεται από τα παρακάτω στοιχεία:

- Ο υπολογισμός που προτείνει είναι γραμμικός. Για κάθε υπολογιστική διαμόρφωση (configuration) υπάρχει ακριβώς μία νόμιμη επόμενη διαμόρφωση.
- Η υπολογιστική διαδικασία προχωρεί βήμα προς βήμα και είναι σε θέση να σταματήσει για οποιαδήποτε δυνατή είσοδο.

Σχήμα 2.1: Ντετερμινιστικός αλγόριθμος

Nondeterminism



Σχήμα 2.2: Μη ντετερμινιστικός αλγόριθμος

Ταξινόμηση αλγορίθμων

Ανάλογα με:

- τις δομές δεδομένων που χρησιμοποιούν
- το είδος των δεδομένων που χρησιμοποιούν (πραγματικούς αριθμούς– αριθμητική ανάλυση, γράφους, κ.τ.λ.)
- την πολυπλοκότητά τους
- την στρατηγική σχεδιασμού τους (π.χ. divide and conquer, greedy, dynamic programming, backtracking κ.τ.λ.)

Μοντέλα Υπολογισμού

Θέλοντας να τυποποιήσουμε δηλαδή να ορίσουμε αυστηρά την έννοια του αλγορίθμου, είναι απαραίτητο να ορίσουμε ένα συγκεκριμένο υπολογιστικό μοντέλο. Πολλοί επιστήμονες, όπως οι *A. Turing*, *A. Church*, *S. Kleene*, *E. Post*, *R. Markov* κ.α., ασχολήθηκαν με το θέμα αυτό και όρισαν διάφορα υπολογιστικά μοντέλα.

Το υπολογιστικό μοντέλο που αντιστοιχεί στον πιο «φυσικό» και διαισθητικό ορισμό του αλγορίθμου είναι η **μηχανή Turing**. Σύμφωνα με την αξιωματική «**θέση του Church**»:

> *«Κάθε αλγόριθμος μπορεί να περιγραφεί με τη βοήθεια μιας μηχανής Turing»*

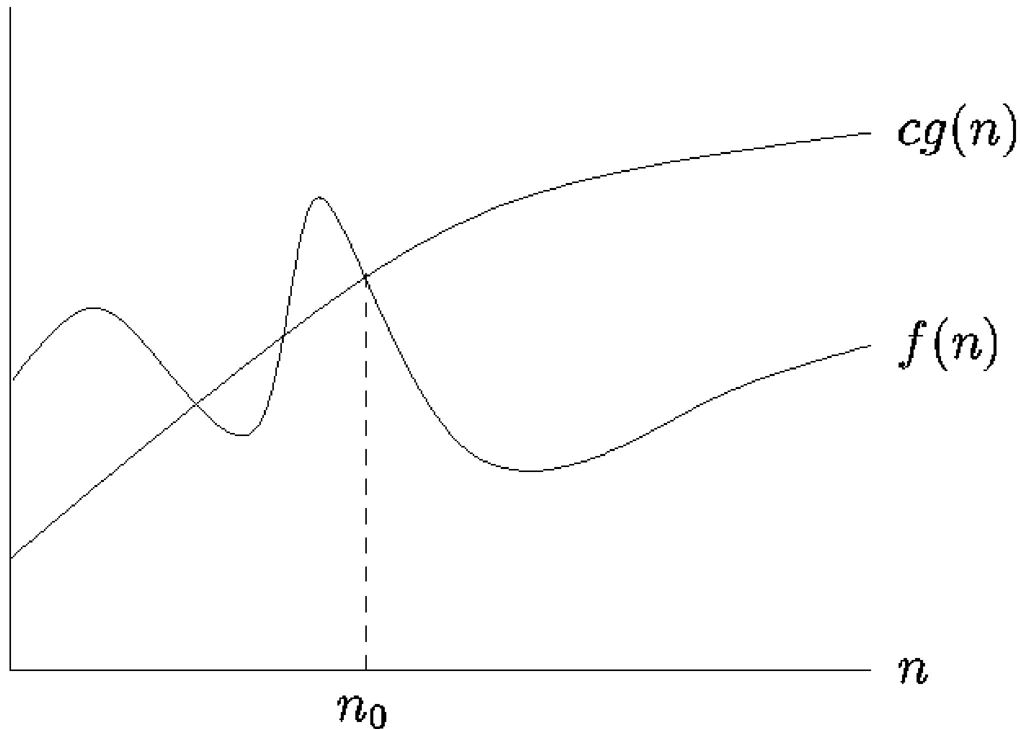
Θέση των Church-Turing (ισοδύναμη διατύπωση)

«Όλα τα γνωστά και άγνωστα υπολογιστικά μοντέλα είναι μηχανιστικά ισοδύναμα»

δηλαδή:

«Για μια συγκεκριμένη συνάρτηση f, δοθέντος ενός αλγορίθμου σ'ένα υπολογιστικό μοντέλο μπορούμε με τη βοήθεια μηχανής (ή προγράμματος: compiler) να κατασκευάσουμε, για την ίδια συνάρτηση f, αλγόριθμο σ'ένα άλλο υπολογιστικό μοντέλο».

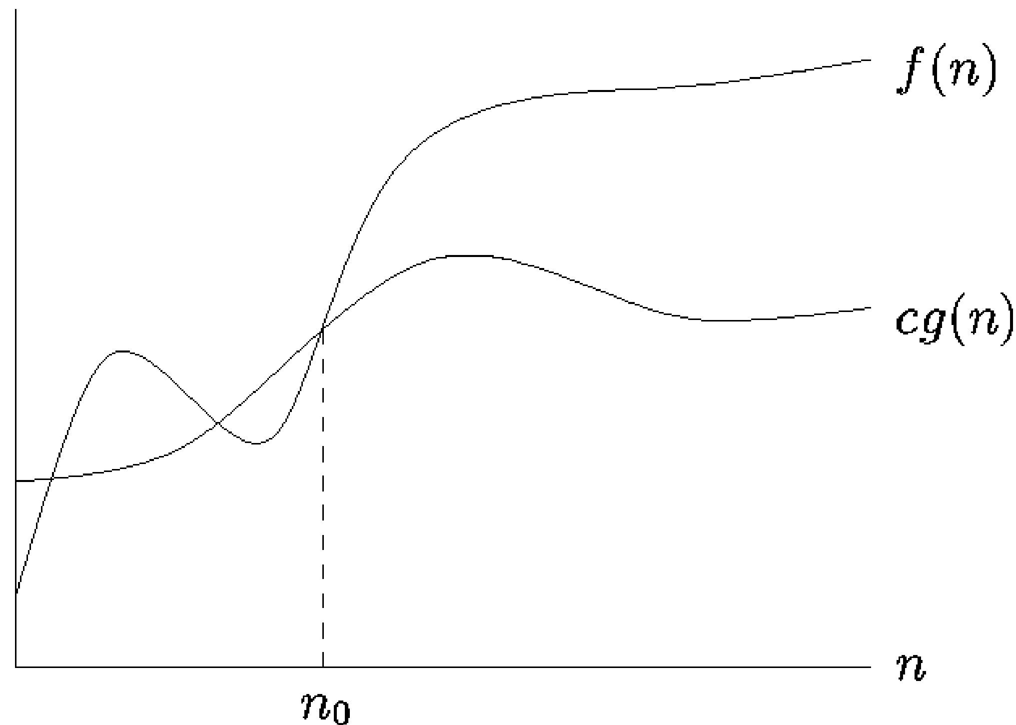
Μαθηματικοί Συμβολισμοί



Σχήμα 2.3: $f = O(g)$

$$O(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \ \ f(n) \leq cg(n)\}$$

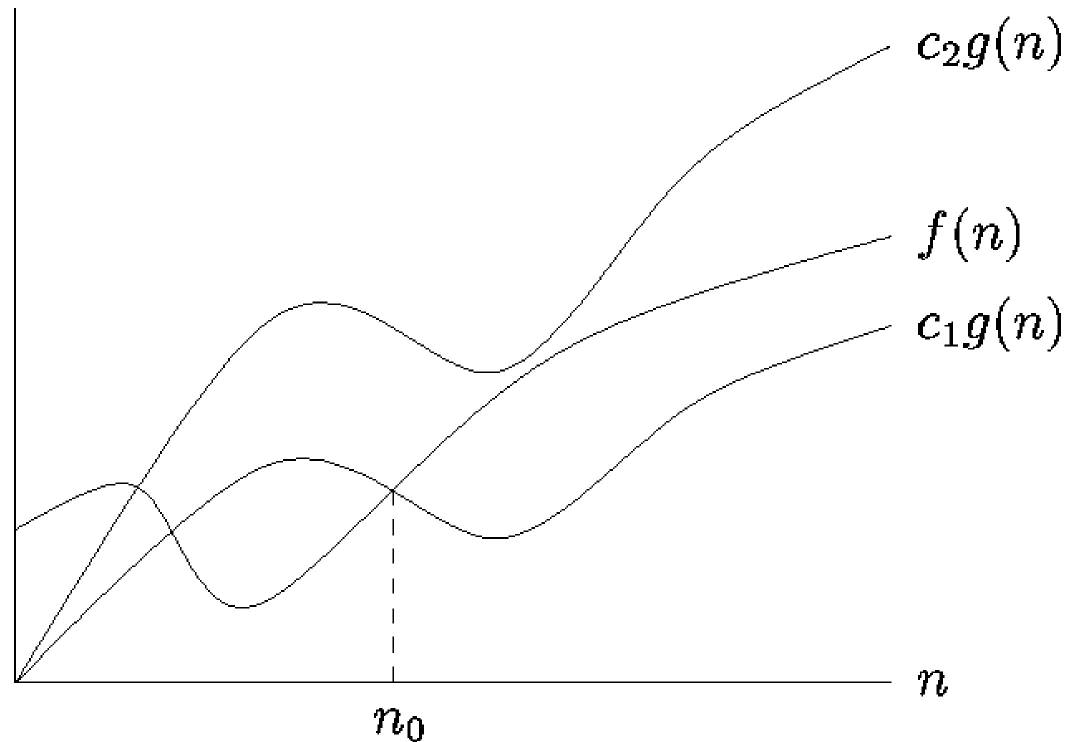
Μαθηματικοί Συμβολισμοί



Σχήμα 2.4: $f = \Omega(g)$

$$\Omega(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \ \ f(n) \geq cg(n)\}$$

Μαθηματικοί Συμβολισμοί



Σχήμα 2.5: $f = \Theta(g)$

$$\Theta(g) = \{f \mid \exists c_1 > 0,\ \exists c_2 > 0,\ \exists n_0 : \forall n > n_0\ \ c_1 \leq \frac{f(n)}{g(n)} \leq c_2\}$$

Ισχύουν: $\Theta(f) = O(f) \bigcap \Omega(f)$ και $f \in \Theta(g) \iff (f \in O(g)$ και $g \in O(f))$.

Αν $p = c_k n^k + c_{k-1} n^{k-1} + \cdots + c_0$, δηλαδή πολυώνυμο βαθμού k, τότε $p \in \mathrm{O}(n^k)$ ή $p(n) = \mathrm{O}(n^k)$. Επίσης $p \in \Omega(n^k)$ ή $p(n) = \Omega(n^k)$. Συνεπώς $p(n) = \Theta(n^k)$.

Ορίζουμε $\mathrm{O}(poly) = \bigcup \mathrm{O}(n^k)$.

Γενικά ισχύει ότι: $O(1) < O(\alpha(n)) < O(\log^* n) < O(\log(n)) < O(\sqrt{n}) < O(n) < O(n \log(n)) < O(n^2) < \ldots < O(poly) < O(2^n) < O(n!) < O(n^n) < O(A(n))$

Σημείωση: γράφουμε «$<$» αντί «$\subset$».

Ορισμός 2.1.1 (*Πολυλογαριθμική συνάρτηση*). $\log^* n$ δίνει πόσες φορές πρέπει να λογαριθμήσουμε το n για να πάρουμε 1.

Ορισμός 2.1.2 (*Συνάρτηση Ackermann*). Ορίζουμε μία συνάρτηση ως εξής:

$$A(x,y) = \begin{cases} 1, & \text{όταν } x=0, y \geq 0 \\ 2, & \text{όταν } x=1, y=0 \\ x+2, & \text{όταν } x \geq 2, y=0 \\ A(A(x-1,y),y-1), & \text{όταν } x,y \geq 1 \end{cases}$$

Μπορούμε εύκολα να παρατηρήσουμε ότι:

$$\begin{aligned} A(x,0) &= 2+x \\ A(x,1) &= 2x \\ A(x,2) &= 2^x \\ A(x,3) &= \left.2^{2^{\cdot^{\cdot^{2}}}}\right\}x \end{aligned}$$

Η συνάρτηση Ackermann είναι μια συνάρτηση η οποία αυξάνεται πολύ γρήγορα. Η συνάρτηση Ackermann με ένα όρισμα μπορεί να οριστεί ως εξής: $A(n) = A(n,n)$.

Polynomially related functions

Ακόμη μεγαλύτερη αφαίρεση: δύο συναρτήσεις θεωρούνται ισοδύναμες όχι μόνο αν διαφέρουν κατά μία σταθερά (όπως πχ στο συμβολισμό Θ(...)) αλλά ακόμη και αν η διαφορά τους είναι πολυωνυμική.

Ορισμός 2.1.3. Δύο συναρτήσεις f_1, f_2 έχουν πολυωνυμική σχέση μεταξύ τους (*polynomially related*) ανν υπάρχουν πολυώνυμα $p_1(x)$, $p_2(x)$ τέτοια ώστε

$$\forall n : f_1(n) \leq p_1(f_2(n)) \bigwedge f_2(n) \leq p_2(f_1(n))$$

Παράδειγμα 2.1.4. Οι συναρτήσεις $f_1(n) = n^3$ και $f_2(n) = n^{17}$ είναι polynomially related, ενώ οι n^3 και 2^n δεν είναι.

Εύρεση Μέγιστου Κοινού Διαιρέτη (gcd)

Δεν είναι λογικό να ανάγεται στο πρόβλημα εύρεσης πρώτων παραγόντων γιατί αυτό δεν λύνεται αποδοτικά.

Απλός αλγόριθμος: *O(min(α,b))*

$$z := min(a,b);$$
$$\textbf{while}\ (a \textbf{ mod } z \neq 0) \textbf{ or } (a \textbf{ mod } z \neq 0) \textbf{ do } z := z - 1;$$

Αλγόριθμος με αφαιρέσεις: *O(max(α,b))*

$$i := a;\ j := b;$$
$$\textbf{while}\ i \neq j \textbf{ do if } i > j \textbf{ then } i := i - j \textbf{ else } j := j - i;$$
$$\textbf{return}\ (i)$$

Αλγόριθμος του Ευκλείδη: *O(*log*(α+b))*

$$i := a;\ j := b;$$
$$\textbf{while}\ (i > 0) \textbf{ and } (j > 0) \textbf{ do}$$
$$\quad \textbf{if } i > j \textbf{ then } i := i \textbf{ mod } j \textbf{ else } j := j \textbf{ mod } i;$$
$$\textbf{return}\ (i + j)$$

Πολ/τητα Αλγ/μου Ευκλείδη

Μάλιστα, την χειρότερη απόδοση ο αλγόριθμος του Ευκλείδη την παρουσιάζει αν του δοθούν ως είσοδος δύο διαδοχικοί αριθμοί της ακολουθίας Fibonacci:

$$F_k = \begin{cases} 0 & \text{για } k = 0 \\ 1 & \text{για } k = 1 \\ F_{k-1} + F_{k-2} & \text{για } k \geq 2 \end{cases}$$

Ισχύει $F_k = (\phi^k - \hat{\phi}^k)/\sqrt{5}$, όπου $\phi = (1+\sqrt{5})/2 \approx 1.618$ (γνωστή και ως η *χρυσή τομή*) και $\hat{\phi} = (1-\sqrt{5})/2$. Ας σημειώσουμε ότι επειδή $|\hat{\phi}| < 1$, το F_k είναι ίσο με το $\phi^k/\sqrt{5}$ στρογγυλοποιημένο στον πλησιέστερο ακέραιο, οπότε προκύπτει το παρακάτω πόρισμα:

Πόρισμα 2.3.1. $\log_\phi(F_k) + 1 \leq k \leq \log_\phi(F_k) + 2$

Έχουμε τα παρακάτω αποτελέσματα σχετικά με την πολυπλοκότητα:

Λήμμα 2.3.2. *Ο αλγόριθμος του Ευκλείδη για $a = F_{k+1}$ και $b = F_k$ έχει χρονική πολυπλοκότητα $O(k)$.*

Πόρισμα 2.3.3. *Η χρονική πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι $\Omega(\log(a+b))$.*

Λήμμα 2.3.4. *Τα ζεύγη διαδοχικών αριθμών Fibonacci είναι η χειρότερη περίπτωση από άποψη χρονικής πολυπλοκότητας για τον αλγόριθμο του Ευκλείδη.*

Πόρισμα 2.3.5. *Η χρονική πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι $O(\log(a+b))$.*

Θεώρημα 2.3.6. *Η χρονική πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι $\Theta(\log(a+b))$.*

Ύψωση σε δύναμη

```
power(a, n)
    result := 1;
    for i := 1 to n do
        result := result*a;
    return result
```

Πολυπλοκότητα: $O(n)$ - εκθετική!

Ύψωση σε δύναμη με *επαναλαμβανόμενο τετραγωνισμό*

```
fastpower(a, n)
    result := 1;
    while n>0 do {
        if odd(n) then result:=result*a;
        n := n div 2;
        a := a*a
    }
    return result
```

Ιδέα: $a^{13} = a^{1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0}$

Πολυπλοκότητα: $O(\log n)$ - πολυωνυμική

Ευχαριστίες

Οι διαφάνειες που ακολουθούν έχουν βασιστεί σε διαφάνειες ομιλίας του καθηγητή Ηλία Κουτσουπιά (Τμήμα Πληροφορικής και Τηλεπικοινωνιών, ΕΚΠΑ)

Αριθμοί Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

$F_0 = 0, \; F_1 = 1,$

$F_n = F_{n-1} + F_{n-2}, \; n >= 2$

Πρόβλημα: Δίνεται n, να υπολογιστεί το F_n

Πόσο γρήγορο μπορεί να είναι το πρόγραμμά μας;

Αριθμοί Fibonacci – αναδρομικός αλγόριθμος

- ```
  F(n)
      if (n<2) then return n
      else return F(n-1)+F(n-2);
  ```

- *Πολυπλοκότητα*: T(n) = T(n-1) + T(n-2) + c, δηλ. η T(n) ορίζεται όπως η F(n) (+ κάτι μικρό), οπότε:

$$T(n) > F(n) = \Omega(1.62^n)$$
  ```

Αριθμοί Fibonacci – καλύτερος αλγόριθμος

- F(n)

```
    a:=0; b:=1;
    for i:=2 to n do
        c:=b; b:=a+b; a:=c;
    return b;
```

- Πολυπλοκότητα: $O(n)$

Χρόνος εκτέλεσης αλγορίθμων

- Θεωρήστε 4 προγράμματα με αριθμό βημάτων $O(2^n)$, $O(n^2)$, $O(n)$, και $O(\log n)$ που το καθένα χρειάζεται 1 δευτερόλεπτο για να υπολογίσει το F(100).
- Πόσα δευτερόλεπτα θα χρειαστούν για να υπολογίσουν το F(n);

	$c \cdot 2^n$	$c \cdot n^2$	$c \cdot n$	$c \cdot \log n$
F(100)	1	1	1	1
F(101)	2	1.02	1.01	1.002
F(110)	1024	1.21	1.1	1.02
F(200)	??????	4	2	1.15

Αριθμοί Fibonacci – ακόμα καλύτερος αλγόριθμος

Μπορούμε να γράψουμε τον υπολογισμό σε μορφή πινάκων:

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

Από αυτό συμπεραίνουμε

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Και ο αριθμός των **αριθμητικών πράξεων** μειώνεται στο $O(\log n)$.

Προβλήματα πρώτων αριθμών

- Primality testing: Δίνεται ακέραιος *n*. Είναι πρώτος;
  - Σχετικά εύκολο. Ανήκει στο P όπως έδειξαν πρόσφατα κάποιο προπτυχιακοί Ινδοί φοιτητές.
- Factoring: Δίνεται ακέραιος *n*. Να βρεθούν οι πρώτοι παράγοντες του.
  - Δεν ξέρουμε αν είναι εύκολο ή δύσκολο. Πιστεύουμε ότι δεν είναι στο P, αλλά ούτε ότι είναι τόσο δύσκολο όσο τα NP-complete προβλήματα.
  - Για κβαντικούς υπολογιστές (που δεν έχουμε ακόμα καταφέρει να κατασκευάσουμε) ανήκει στο P.

Factoring και κρυπτογραφία

- RSA: Κρυπτογραφικό σχήμα δημοσίου κλειδιού για να στείλει η A (Alice) στον B (Bob) ένα μήνυμα m.
- O B διαλέγει 2 μεγάλους πρώτους αριθμούς p και q, υπολογίζει το γινόμενο n=pq, και διαλέγει ακέραιο e σχετικά πρώτο με το φ(n)=(p-1)(q-1).
- O B στέλνει στην A τα n και e.
- H A στέλνει στον B τον αριθμό $c=m^e (\text{mod } n)$.
- O B υπολογίζει το m: $m=c^d (\text{mod } n)$, όπου το $d=e^{-1} (\text{mod } (p-1)(q-1))$.
- Παράδειγμα: p=11, q=17, n=187, e=21, d=61, m=42, c=9

- Η ασφάλεια του RSA στηρίζεται στην (εκτιμώμενη) δυσκολία του factoring.

- Για την υλοποίηση του RSA χρησιμοποιούνται, μεταξύ άλλων, ο αλγόριθμος επαναλαμβανόμενου τετραγωνισμού και ο επεκτεταμένος Ευκλείδειος αλγόριθμος (που επιπλέον εκφράζει τον gcd(a,b) σαν γραμμικό συνδυασμό των a και b).

Πολυπλοκότητα: ανοικτά ερωτήματα

- Εκτός από κάποιες ειδικές περιπτώσεις, για κανένα πρόβλημα δεν γνωρίζουμε πόσο γρήγορα μπορεί να λυθεί.
- Ακόμα και για τον πολλαπλασιασμό αριθμών δεν γνωρίζουμε τον ταχύτερο αλγόριθμο.
- Ο σχολικός τρόπος πολλαπλασιασμού αριθμών με n ψηφία χρειάζεται $O(n^2)$ βήματα.
- Υπάρχουν καλύτεροι αλγόριθμοι που χρειάζονται περίπου $O(n \log n)$ βήματα.
- Υπάρχει αλγόριθμος που χρειάζεται μόνο $O(n)$ βήματα; Αυτό είναι ανοικτό ερώτημα.