ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Δύο ιδέες λάμπουν στο βελούδο του κοσμηματοπώλη. Η πρώτη είναι ο απειροστικός λογισμός και η δεύτερη ο αλγόριθμος. Ο απειροστικός λογισμός και το πλούσιο πεδίο της μαθηματικής ανάλυσης που αυτός γέννησε, έκαναν δυνατή την εμφάνιση της μοντέρνας επιστήμης: όμως, ο αλγόριθμος είναι αυτός που διαμόρφωσε τον σύγχρονο κόσμο.

— *David Berlinski, "The Advent of the Algorithm", 2000*

Γιατί θα πρέπει να μελετήσετε τους αλγόριθμους; Εάν πρόκειται να σταδιοδρομήσετε ως επαγγελματίες της πληροφορικής, υπάρχουν πρακτικοί και θεωρητικοί λόγοι που επιβάλλουν τη μελέτη τους. Από την πρακτική σκοπιά, θα πρέπει να γνωρίζετε ένα τυποποιημένο σύνολο σημαντικών αλγορίθμων που να καλύπτουν διάφορες περιοχές της πληροφορικής. Επιπρόσθετα, θα πρέπει να είστε σε θέση να σχεδιάζετε νέους αλγόριθμους και να αναλύετε την αποδοτικότητά τους. Απο τη θεωρητική σκοπιά, η μελέτη των αλγορίθμων, η οποία αναφέρεται και ως ***αλγοριθμική*** (*algorithmics*), έχει πλέον αναγνωριστεί ως ο θεμέλιος λίθος της επιστήμης των υπολογιστών. Ο David Harel, στο εξαίσιο βιβλίο του που φέρει τον εύστοχο τίτλο «*Αλγοριθμική: το Πνεύμα της Επιστήμης των Υπολογιστών*» (*Algorithmics: the Spirit of Computing*), το θέτει ως εξής:

> Η Αλγοριθμική είναι κάτι παραπάνω από ένας απλός κλάδος της επιστήμης της πληροφορικής. Είναι ο πυρήνας της και, εάν θέλουμε να είμαστε δίκαιοι, μπορούμε να πούμε ότι άπτεται των περισσοτέρων τομέων της επιστήμης, της επιχειρηματικότητας και της τεχνολογίας. [3] σελ. 6.

Ακόμη και εάν δεν είστε φοιτητής σε ένα σχετικό με υπολογιστές πρόγραμμα σπουδών, υπάρχουν πειστικοί λόγοι για να μελετήσετε τους αλγόριθμους. Για να το πούμε

απλά, τα προγράμματα των υπολογιστών δεν θα ήταν δυνατό να υπάρξουν χωρίς τους αλγόριθμους. Και επειδή οι υπολογιστικές εφαρμογές καθίστανται απαραίτητες πλέον σχεδόν σε όλους τους τομείς της επαγγελματικής και της προσωπικής μας ζωής, η μελέτη των αλγορίθμων είναι αναγκαία για όλο και περισσότερους ανθρώπους.

Ένας άλλος λόγος που επιβάλλει τη μελέτη των αλγορίθμων είναι η χρησιμότητά τους στην ανάπτυξη δεξιοτήτων ανάλυσης. Στο κάτω–κάτω, οι αλγόριθμοι μπορούν να ειδωθούν ως ειδικές κατηγορίες λύσεων σε προβλήματα—όχι ως απαντήσεις αυτές καθεαυτές, αλλά επακριβώς ορισμένες διαδικασίες λήψης απαντήσεων. Συνεπώς, συγκεκριμένες τεχνικές σχεδίασης αλγορίθμων μπορούν να ερμηνευτούν ως στρατηγικές επίλυσης προβλημάτων, οι οποίες είναι χρήσιμες, ανεξάρτητα από το εάν στο προκείμενο πρόβλημα εμπλέκεται ή όχι ένας υπολογιστής. Φυσικά, η ακρίβεια που επιβάλλεται από τον αλγοριθμικό τρόπο σκέψης, περιορίζει τα είδη των προβλημάτων που μπορούν να επιλυθούν με έναν συγκεκριμένο αλγόριθμο. Δεν θα βρείτε, για παράδειγμα, έναν αλγόριθμο για το πώς να διάγετε μια ευτυχισμένη ζωή, ή για το πώς θα γίνετε πλούσιοι και διάσημοι. Από την άλλη μεριά, αυτή η απαιτούμενη ακρίβεια διαθέτει ένα σημαντικό παιδαγωγικό χαρακτηριστικό. Ο Donald Knuth, ένας από τους πλέον εξέχοντες επιστήμονες της πληροφορικής σε όλη τη μέχρι τώρα ιστορία της αλγοριθμικής, το θέτει ως εξής:

> Ένας άνθρωπος καλά εκπαιδευμένος στην επιστήμη των υπολογιστών, γνωρίζει πώς να αντιμετωπίζει τους αλγόριθμους: πώς να τους κατασκευάζει, να τους χειρίζεται, να τους καταλαβαίνει, να τους αναλύει. Αυτή η γνώση αποτελεί υπόβαθρο για κάτι πολύ περισσότερο από τη συγγραφή καλών προγραμμάτων: είναι ένα νοητικό εργαλείο γενικής χρήσης το οποίο είναι καθοριστικής σημασίας στην κατανόηση άλλων γνωστικών αντικειμένων, είτε αυτά έχουν να κάνουν με τη χημεία, τη γλωσσολογία, τη μουσική κλπ. Το γιατί συμβαίνει αυτό, μπορεί να γίνει κατανοητό με τον εξής τρόπο: Έχει ειπωθεί ότι κάποιος δεν καταλαβαίνει στην ουσία ένα πράγμα, μέχρις ότου να το διδάξει σε κάποιον άλλο. Στην πραγματικότητα, κάποιος δεν καταλαβαίνει ένα πράγμα, παρά μόνο εφόσον το έχει διδάξει σε έναν *υπολογιστή*, δηλαδή αφού το έχει εκφράσει ως έναν αλγόριθμο. Η προσπάθεια τυποποίησης της γνώσης με την μορφή αλγορίθμων οδηγεί σε πολύ βαθύτερη κατανόηση, από ό,τι μια απλή προσπάθεια αντίληψης των πραγμάτων, βασισμένη στον παραδοσιακό τρόπο. [4], σελ. 9.

Εισάγουμε και συζητούμε την έννοια του αλγορίθμου στην Ενότητα 1.1. Ως παραδείγματα χρησιμοποιούμε τρεις αλγόριθμους για το ίδιο πρόβλημα: τον υπολογισμό του μέγιστου κοινού διαιρέτη (ΜΚΔ). Υπάρχουν αρκετοί λόγοι που επιβάλλουν αυτήν την επιλογή. Κατά πρώτον, ασχολούμαστε με ένα οικείο σε όλους πρόβλημα, από τα σχολικά μας χρόνια. Κατά δεύτερον, γίνεται έτσι φανερό ένα σημαντικό σημείο, ότι

δηλαδή το ίδιο πρόβλημα μπορεί να επιλυθεί με περισσότερους του ενός αλγόριθμους. Η συγκεκριμένη είναι μια τυπική περίπτωση, όπου οι διάφοροι αλγόριθμοι διαφέρουν ως προς την κεντρική τους ιδέα, το επίπεδο πολυπλοκότητάς τους και την απόδοσή τους. Και τρίτον, ο ένας από τους αλγόριθμους αυτούς αξίζει να παρουσιαστεί πρώτος, τόσο λόγω της ηλικίας του—εμφανίστηκε στη διάσημη πραγματεία των Στοιχείων του Ευκλείδη, περισσότερο από δύο χιλιάδες χρόνια πριν, όσο και λόγω της διαρκούσης δύναμης και σημασίας του. Τέλος, η οικεία διαδικασία υπολογισμού του, από τα μαθητικά μας χρόνια, μας επιτρέπει να τονίσουμε μια καίρια απαίτηση, την οποία θα πρέπει να ικανοποιεί κάθε αλγόριθμος.

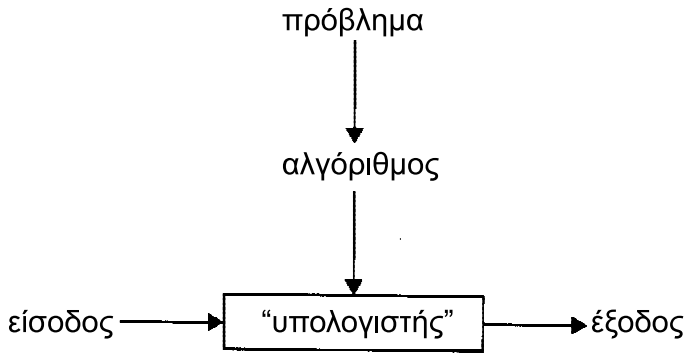
Η Ενότητα 1.2 ασχολείται με την αλγοριθμική επίλυση προβλημάτων. Εκεί συζητούμε διάφορα σημαντικά θέματα που σχετίζονται με τη σχεδίαση και την ανάλυση των αλγορίθμων. Οι διάφορες όψεις της αλγοριθμικής επίλυσης προβλημάτων ποικίλλουν από την ανάλυση του προβλήματος και τα μέσα που χρησιμοποιούμε για να εκφράσουμε έναν αλγόριθμο, μέχρι τη θεμελίωση της ορθότητάς του και την ανάλυση της απόδοσής του. Η Ενότητα αυτή δεν περιέχει κάποια μαγική συνταγή κατασκευής ενός αλγορίθμου για ένα αυθαίρετο πρόβλημα. Είναι κοινά αποδεκτό ότι δεν υπάρχει κάποια τέτοια συνταγή. Παρόλα αυτά, το υλικό της Ενότητας 1.2 θα είναι χρήσιμο για την οργάνωση της δουλειάς σας, όσον αφορά τη σχεδίαση και την ανάλυση αλγορίθμων.

Η Ενότητα 1.3 είναι αφιερωμένη σε μερικούς τύπους προβλημάτων, οι οποίοι έχουν αποδειχθεί ιδιαίτερα σημαντικοί στη μελέτη των αλγορίθμων, καθώς και των εφαρμογών τους. Στην πραγματικότητα, υπάρχουν διδακτικά εγχειρίδια (π.χ. [5]) που είναι οργανωμένα με βάση αυτούς τους τύπους προβλημάτων. Τείνω να συμμεριστώ την άποψη (η οποία υποστηρίζεται και από πολλούς άλλους) ότι μια οργάνωση με βάση τις τεχνικές σχεδίασης των αλγορίθμων, υπερτερεί της προηγούμενης. Σε κάθε περίπτωση όμως, είναι πολύ σημαντικό να είμαστε γνώστες των κύριων τύπων προβλημάτων. Αυτοί, δεν είναι μόνο οι πλέον κοινά απαντώμενοι τύποι προβλημάτων σε ρεαλιστικές εφαρμογές, αλλά χρησιμοποιούνται και σε όλη την έκταση του βιβλίου, για την επίδειξη των συγκεκριμένων κάθε φορά τεχνικών σχεδίασης αλγορίθμων.

Η Ενότητα 1.4 περιέχει μια ανασκόπηση των θεμελιωδών δομών δεδομένων. Προορίζεται να χρησιμεύσει κυρίως ως αναφορά, αντί μιας διεξοδικής συζήτησης στο θέμα αυτό. Εάν χρειάζεστε μια αναλυτικότερη και σε μεγαλύτερο βάθος αντιμετώπιση, υπάρχει μια πληθώρα καλών βιβλίων για το θέμα αυτό, με τα περισσότερα από αυτά να επικεντρώνονται σε κάποια συγκεκριμένη γλώσσα προγραμματισμού.

1.1 Τι Είναι Ένας Αλγόριθμος;

Αν και δεν υπάρχει κάποια κοινά αποδεκτή φρασεολογία για να περιγράψουμε τη συγκεκριμένη έννοια, υπάρχει εν τούτοις γενική συμφωνία για το τί *σημαίνει* αυτή η έννοια:



Σχήμα 1.1: Η έννοια του αλγορίθμου.

> Ένας ***Αλγόριθμος*** είναι μια ακολουθία ξεκάθαρων, ρητών εντολών για την επίλυση ενός προβλήματος, δηλαδή για την παραγωγή της απαιτούμενης εξόδου για κάθε αποδεκτή είσοδο, σε πεπερασμένο χρόνο.

Αυτός ο ορισμός μπορεί να απεικονιστεί με ένα απλό διάγραμμα (Σχήμα 1.1).

Η αναφορά σε «εντολές» (instructions) στον ορισμό αυτό υπονοεί την ύπαρξη κάποιου (ανθρώπου ή μηχανισμού), ο οποίος είναι ικανός να καταλάβει και να ακολουθήσει τις συγκεκριμένες εντολές. Αυτή την οντότητα την ονομάζουμε «υπολογιστή», έχοντας στο μυαλό μας ότι, στην προ των ηλεκτρονικών υπολογιστών εποχή, ο όρος «υπολογιστής» σήμαινε κάποιον άνθρωπο που επιτελούσε αριθμητικούς υπολογισμούς. Σήμερα, φυσικά, οι «υπολογιστές» είναι αφανείς (ubiquitous) ηλεκτρονικές συσκευές που έχουν καταστεί αναπόσπαστο μέρος σχεδόν οποιασδήποτε ενέργειάς μας. Παρατηρήστε όμως ότι, αν και η πλειοψηφία των αλγορίθμων όντως προορίζεται για υλοποίηση σε έναν υπολογιστή, η έννοια του αλγορίθμου δεν εξαρτάται από αυτή την προοπτική.

Ως παραδείγματα που καταδεικνύουν την έννοια του αλγορίθμου, θα θεωρήσουμε σε αυτή την Ενότητα τρεις μεθόδους για να επιλύσουμε ένα και το αυτό πρόβλημα: τον υπολογισμό του μέγιστου κοινού διαιρέτη (ΜΚΔ) δύο ακέραιων. Τα παραδείγματα αυτά θα μας βοηθήσουν να αντιληφθούμε μερικά σημαντικά σημεία:

- Η απαίτηση για ξεκάθαρη έκφραση του κάθε βήματος του αλγορίθμου, δεν γίνεται να μην υπάρχει.

- Το εύρος τιμών των εισόδων, για το οποίο ο αλγόριθμος μπορεί να εφαρμοστεί, θα πρέπει να καθοριστεί με προσοχή.

- Ο ίδιος αλγόριθμος μπορεί να αναπαρασταθεί με διάφορους, ισοδύναμους μεταξύ τους, τρόπους.
- Ενδέχεται να υπάρχουν περισσότεροι του ενός αλγόριθμοι, για την επίλυση του ίδιου προβλήματος.
- Αλγόριθμοι για το ίδιο πρόβλημα μπορεί να βασίζονται σε εντελώς διαφορετικές ιδέες και μπορεί να επιλύουν το πρόβλημα με εντελώς διαφορετικές ταχύτητες επίλυσης.

Θυμηθείτε ότι ο ***μέγιστος κοινός διαιρέτης*** (*greatest common divisor, GCD*) δύο μη-αρνητικών, μη-ταυτόχρονα-μηδενικών ακεραίων m και n, δηλούμενος ως $gcd(m, n)$, ορίζεται ως ο μεγαλύτερος ακέραιος που διαιρεί ακριβώς τους m και n, αφήνοντας δηλαδή υπόλοιπο μηδέν. Ο Ευκλείδης ο Αλεξανδρεύς (τρίτος αιώνας π.Χ.) περιέγραψε έναν αλγόριθμο για την επίλυση του προβλήματος αυτού, σε ένα από τα βιβλία των *Στοιχείων* του, τα οποία παρέμειναν γνωστά ανά τους αιώνες για τη συστηματική ανάπτυξη της γεωμετρίας που περιείχαν. Σε σύγχρονους όρους, ο ***αλγόριθμος του Ευκλείδη*** (*Euclid's algorithm*) βασίζεται στην επαναληπτική εφαρμογή της ταυτότητας

$$gcd(m, n) = gcd(n, m \ mod \ n)$$

(όπου $m \ mod \ n$ είναι το υπόλοιπο της διαίρεσης του m από τον n), μέχρις ότου το $m \ mod \ n$ γίνει μηδέν. Εφόσον $gcd(m, 0) = m$ (γιατί ισχύει αυτό;), η τελική τιμή του m είναι και ο μέγιστος κοινός διαιρέτης της αρχικής τιμής του m και του n.

Για παράδειγμα, ο $gcd(60, 24)$ μπορεί να υπολογιστεί ως ακολούθως:

$$gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12 \ .$$

Εάν δεν έχετε εντυπωσιαστεί από τον αλγόριθμο αυτό, προσπαθήστε να βρείτε το μέγιστο κοινό διαιρέτη μεγάλων αριθμών, όπως αυτών του Προβλήματος 5 των Ασκήσεων της Ενότητας 1.1.

Παραθέτουμε μια περισσότερο δομημένη περιγραφή του αλγορίθμου αυτού:

Αλγόριθμος του Ευκλείδη για τον υπολογισμό του $gcd(m, n)$

- **Βήμα 1** Εάν $n = 0$, επέστρεψε την τιμή του m ως απάντηση και σταμάτησε. Αλλιώς, προχώρα στο Βήμα 2.
- **Βήμα 2** Διαίρεσε τον m με τον n και ανάθεσε την τιμή του υπόλοιπου στον r.
- **Βήμα 3** Ανάθεσε την τιμή του n στον m και αυτήν του r στον n. Πήγαινε στο Βήμα 1.

Εναλλακτικά, μπορούμε να εκφράσουμε τον ίδιο αλγόριθμο σε μια μορφή ψευδοκώδικα:

Αλγόριθμος 1.1.1 : *Euclid*(m, n)

```
1 //Υπολογισμός του ΜΚΔ gcd(m,n) με τον αλγόριθμο του Ευκλείδη
2 //Είσοδος: δύο μη-αρνητικοί, μη-ταυτόχρονα-μηδενικοί ακέραιοι m και n
3 //Έξοδος: ο μέγιστος κοινός διαιρέτης των m και n
4 while n ≠ 0 do {
5     r ← m mod n
6     m ← n
7     n ← r
8 }
9 return m
```

Πώς ξέρουμε ότι ο αλγόριθμος του Ευκλείδη τερματίζεται σε πεπερασμένο χρόνο; Αυτό είναι επακόλουθο της παρατήρησης ότι ο δεύτερος αριθμός του (διατεταγμένου) ζεύγους (m, n) γίνεται συνεχώς μικρότερος, με κάθε νέα επανάληψη και από το ότι δεν μπορεί να γίνει αρνητικός. Πράγματι, η νέα τιμή του n στην επόμενη επανάληψη θα είναι $m \ mod \ n$, η οποία είναι πάντοτε μικρότερη από n. Επομένως, η τιμή του δεύτερου αριθμού του ζεύγους τελικά θα γίνει 0, οπότε ο αλγόριθμος περατώνεται.

Όπως συμβαίνει και με πολλά άλλα προβλήματα, υπάρχουν περισσότεροι του ενός αλγόριθμοι για τον υπολογισμό του μέγιστου κοινού διαιρέτη. Ας ρίξουμε μια ματιά στις άλλες δύο μεθόδους επίλυσης του προβλήματος αυτού. Η πρώτη βασίζεται απλά στον ορισμό του μέγιστου κοινού διαιρέτη των m και n, ως το μεγαλύτερο ακέραιο ο οποίος διαιρεί ακριβώς τους ακέραιους αυτούς. Προφανώς, ο κοινός αυτός διαιρέτης δεν μπορεί να είναι μεγαλύτερος από το μικρότερο από τους δύο αριθμούς, τον οποίο συμβολίζουμε $t = min(m, n)$. Επομένως, μπορούμε να ξεκινήσουμε ελέγχοντας εάν ο t διαιρεί ακριβώς και τον m και τον n: εάν αυτό ισχύει, τότε ο t είναι η ζητούμενη απάντηση. Εάν όχι, απλά μειώνουμε τον t κατά 1 και προσπαθούμε ξανά. (Πώς ξέρουμε ότι αυτή η διαδικασία θα τερματιστεί τελικά;) Για παράδειγμα, για τους αριθμούς 60 και 24, ο αλγόριθμος θα δοκιμάσει πρώτα το 24, μετά το 23 και ούτω καθεξής, μέχρις ότου φθάσει στο 12, οπότε και θα σταματήσει.

Αλγόριθμος ελέγχου διαδοχικών ακέραιων για τον υπολογισμό του $gcd(m, n)$

- **Βήμα 1** Ανάθεσε την τιμή $min(m, n)$ στον t.
- **Βήμα 2** Διαίρεσε τον m με τον t. Εάν το υπόλοιπο της διαίρεσης αυτής είναι 0, πήγαινε στο Βήμα 3. Αλλιώς, πήγαινε στο Βήμα 4.
- **Βήμα 3** Διαίρεσε τον n με τον t. Εάν το υπόλοιπο της διαίρεσης αυτής είναι 0, επέστρεψε την τιμή του t ως απάντηση και σταμάτησε. Αλλιώς, πήγαινε στο Βήμα 4.
- **Βήμα 4** Μείωσε την τιμή του t κατά 1. Πήγαινε στο Βήμα 2.

Παρατηρήστε ότι, σε αντίθεση με τον αλγόριθμο του Ευκλείδη, αυτός ο αλγόριθμος, με τη μορφή που παρουσιάστηκε εδώ, δε δουλεύει σωστά, εάν ένας από τους αριθμούς στην είσοδό του είναι μηδενικός. Το παράδειγμα αυτό καταδεικνύει πόσο σημαντικός είναι ο ξεκάθαρος και προσεκτικός καθορισμός του εύρους τιμών των εισόδων ενός αλγορίθμου.

Η τρίτη διαδικασία για την εύρεση του μέγιστου κοινού διαιρέτη θα πρέπει να μας είναι οικεία από τα μαθητικά μας χρόνια:

Μαθητικός αλγόριθμος για τον υπολογισμό του $gcd(m, n)$

- **Βήμα 1** Ανέλυσε τον m σε γινόμενο πρώτων παραγόντων.
- **Βήμα 2** Ανέλυσε τον n σε γινόμενο πρώτων παραγόντων.
- **Βήμα 3** Ταυτοποίησε όλους τους κοινούς παράγοντες στα γινόμενα των Βημάτων 1 και 2. Εάν ένας πρώτος παράγοντας επαναλαμβάνεται p_m και p_n φορές στα αναπτύγματα των m και n, αντίστοιχα, τότε αυτός θα πρέπει να επαναληφθεί $min(p_m, p_n)$ φορές.
- **Βήμα 4** Υπολόγισε το γινόμενο όλων των κοινών παραγόντων και επέστρεψε την τιμή αυτή ως το μέγιστο κοινό διαιρέτη των m και n.

Επομένως, για τους αριθμούς 60 και 24, έχουμε:

$$\begin{aligned} 60 &= 2\cdot 2\cdot 3\cdot 5 \\ 24 &= 2\cdot 2\cdot 2\cdot 3 \\ gcd(60,24) &= 2\cdot 2\cdot 3 = 12. \end{aligned}$$

Η νοσταλγία για την εποχή που μάθαμε αυτή τη μέθοδο, δεν θα πρέπει να μας εμποδίσει από το να παραδεχθούμε ότι αυτή η διαδικασία είναι πιο πολύπλοκη και πιο αργή από τον αλγόριθμο του Ευκλείδη. (Θα συζητήσουμε μεθόδους εύρεσης και σύγκρισης των χρόνων εκτέλεσης των αλγορίθμων στο επόμενο Κεφάλαιο.) Επιπλέον της κατώτερης αποδοτικότητας, η μαθητική διαδικασία δεν μπορεί να χαρακτηριστεί ως αλγόριθμος, σύμφωνα με τα κριτήρια που παραθέσαμε παραπάνω, στη μορφή που την παρουσιάσαμε. Γιατί; Ο λόγος είναι ότι τα βήματα που αναφέρονται στην κατασκευή των γινομένων πρώτων παραγόντων, δεν έχουν οριστεί με πλήρη σαφήνεια. Απαιτούν μια λίστα από πρώτους αριθμούς και έχω βάσιμες υποψίες ότι ο δάσκαλός σας ή ο καθηγητής σας δεν σας είχε διδάξει τον τρόπο να παράγετε μια τέτοια λίστα. Αναμφίβολα συμφωνείτε ότι αυτό δεν είναι απλά μια ασήμαντη λεπτομέρεια. Εάν δεν ξεκαθαρίσουμε το σημείο αυτό, δεν μπορούμε, για παράδειγμα, να γράψουμε ένα πρόγραμμα υλοποίησης του αλγορίθμου αυτού. (Παρεπιπτόντως, ούτε το Βήμα 3 είναι σαφώς ορισμένο. Όμως, η ασάφειά του αντιμετωπίζεται πολύ πιο εύκολα, σε σύγκριση με τα βήματα παραγοντοποίησης. Με ποιόν τρόπο θα βρίσκατε τα κοινά στοιχεία δύο ταξινομημένων λιστών;)

Ας εισάγουμε λοιπόν έναν απλό αλγόριθμο για την παραγωγή διαδοχικών πρώτων αριθμών, οι οποίοι δεν θα υπερβαίνουν έναν δεδομένο ακέραιο n. Ένας τέτοιος αλγόριθμος, ο οποίος εφευρέθη μάλλον στην αρχαία Ελλάδα, είναι γνωστός με το όνομα ***κόσκινο του Ερατοσθένη*** (περ. 200 π.Χ.). Ο αλγόριθμος ξεκινά αρχικοποιώντας μια λίστα υποψηφίων πρώτων αριθμών, με διαδοχικούς ακέραιους από το 2 έως το n. Κατόπιν, στην πρώτη επανάληψη, απαλείφει από τη λίστα όλα τα πολλαπλάσια του 2, δηλαδή τα 4, 6 κλπ. Κατόπιν, προχωρά στην επόμενη τιμή της λίστας, τον αριθμό 3, και απαλείφει τα πολλαπλάσιά του. (Σε αυτή την έκδοση άμεσης υλοποίησης (straightforward version), παρατηρείται μια καθυστέρηση, επειδή ορισμένοι αριθμοί, όπως π.χ. ο 6, απαλείφονται περισσότερες από μία φορές.) Δεν απαιτείται ξεχωριστό πέρασμα για τον επόμενο του 3 ακέραιο, τον 4: εφόσον ο 4 και όλα τα πολλαπλάσιά του είναι και πολλαπλάσια του 2, έχουν ήδη απαλοιφεί σε προηγούμενο πέρασμα. (Με παρόμοιους συλλογισμούς, δεν χρειάζεται να λάβουμε υπόψη μας κανένα πολλαπλάσιο ενός ήδη απαληφθέντος αριθμού.) Ο επόμενος μη-απαλειφθείς αριθμός στη λίστα, που θα χρησιμοποιηθεί για ένα νέο πέρασμα, είναι ο 5. Ο αλγόριθμος προχωρά με τον τρόπο αυτό, μέχρις ότου να μη μπορεί να γίνει άλλη απαλοιφή αριθμού από τη λίστα. Οι ακέραιοι που παρέμειναν στη λίστα, είναι οι ζητούμενοι πρώτοι αριθμοί.

Ως παράδειγμα, ας θεωρήσουμε την εφαρμογή του αλγορίθμου στην εύρεση της λίστας των πρώτων αριθμών, οι οποίοι δεν υπερβαίνουν τον $n = 25$:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	3		5		7		9		11		13		15		17		19		21		23		25
2	**3**		5		7				11		13				17		19				23		25
2	3		**5**		7				11		13				17		19				23		

Για το συγκεκριμένο παράδειγμα, δεν απαιτούνται περισσότερα περάσματα, επειδή το μόνο που μπορεί να γίνει είναι να απαλοιφούν ξανά αριθμοί ήδη απορριφθέντες από τα προηγούμενα περάσματα του αλγορίθμου. Οι εναπομείναντες στη λίστα αριθμοί είναι οι διαδοχικοί πρώτοι αριθμοί που είναι μικρότεροι ή ίσοι του 25.

Στη γενική περίπτωση, ποιός είναι ο μεγαλύτερος αριθμός, p, του οποίου τα πολλαπλάσια ενδέχεται να παραμένουν στη λίστα; Πριν απαντήσουμε στην ερώτηση αυτή, ας σημειώσουμε πρώτα ότι εάν ο p είναι ο αριθμός του οποίου τα πολλαπλάσια απαλείφονται στο τρέχον πέρασμα, τότε το πρώτο πολλαπλάσιό του που χρειάζεται να θεωρήσουμε είναι το $p \cdot p$, επειδή όλα τα μικρότερα πολλαπλάσια, $2p, 3p, \ldots, (p-1)p$, έχουν ήδη απαλοιφεί, σε προηγούμενα περάσματα διαμέσου της λίστας. Αυτή η παρατήρηση βοηθά να αποφύγουμε την απαλοιφή του ίδιου αριθμού, περισσότερες από μια φορές. Προφανώς, ο $p \cdot p$ δεν θα πρέπει να είναι μεγαλύτερος του n και επομένως ο p δεν μπορεί να υπερβαίνει τον $\sqrt{n}$, στρογγυλοποιημένο στον προς τα κάτω ακέραιο (που τον συμβολίζουμε ως $\lfloor \sqrt{n} \rfloor$), χρησιμοποιώντας τη λεγόμενη συνάρτηση «πατώματος» (floor function). Στον ακόλουθο ψευδοκώδικα, υποθέτουμε ότι υπάρχει διαθέσιμη μια συνάρτηση υπολογισμού του $\lfloor \sqrt{n} \rfloor$: εναλλακτικά, μπορούμε να ελέγχουμε την ανισότητα $p \cdot p \leq n$, ως συνθήκη τερματισμού του βρόχου επανάληψης.

Αλγόριθμος 1.1.2 : *Sieve(n)*

```
1  //Υλοποίηση του κόσκινου του Ερατοσθένη
2  //Είσοδος: ένας ακέραιος n ≥ 2
3  //Έξοδος: πίνακας L όλων των πρώτων αριθμών, ≤ n
4  for p ← 2 to n do A[p] ← p
5  for p ← 2 to ⌊√n⌋ do {          //βλ. παρατήρηση στα αμέσως προηγούμενα
6      if A[p] ≠ 0 {               //ο p δεν έχει απαλοιφεί από προηγούμενα περάσματα
7          j ← p · p
8          while j ≤ n do {
9              A[j] ← 0
10             j ← j + p
11         }
12     }
13 }
14 //αντιγραφή των εναπομεινάντων στοιχείων του A
15 //      στον πίνακα L των πρώτων αριθμών
16 i ← 0
17 for p ← 2 to n do {
18     if A[p] ≠ 0 {
19         L[i] ← A[p]
20         i ← i + 1
21     }
22 }
23 return L
```

Τώρα πλέον μπορούμε να ενσωματώσουμε το κόσκινο του Ερατοσθένη στο μαθητικό αλγόριθμο, για να πάρουμε έναν αποδεκτό αλγόριθμο για τον υπολογισμό του μέγιστου κοινού διαιρέτη δύο θετικών ακεραίων. Παρατηρήστε ότι χρειάζεται ειδική προσοχή, για την περίπτωση όπου ένας ή και οι δύο ακέραιοι ισούνται με 1: επειδή οι μαθηματικοί δεν θεωρούν τη μονάδα ως πρώτο αριθμό, με την αυστηρή έννοια του ορισμού, η μέθοδος δεν δουλεύει για τέτοιες εισόδους.

Πριν ολοκληρώσουμε την παρούσα Ενότητα, σημειώνουμε ένα ακόμη σχόλιο. Οι περισσότεροι αλγόριθμοι που βρίσκονται σε χρήση σήμερα (εξαιρούνται τα παραδείγματα της Ενότητας αυτής)—ακόμη και αυτοί που είναι υλοποιημένοι σε προγράμματα ηλεκτρονικών υπολογιστών—δεν ασχολούνται με μαθηματικά προβλήματα. Μπορείτε (και είναι καλό να το κάνετε αυτό) να ψάξετε γύρω σας για αλγόριθμους που μας βοηθάνε στις καθημερινές μας εργασίες, τόσο στη δουλειά, όσο και στην προσωπική μας ζωή. Ας είναι αυτή η αφάνεια των αλγορίθμων στην εποχή που ζούμε, το έναυσμα για να θελήσετε να μάθετε περισσότερα σχετικά με αυτές τις μηχανές παραγωγής πληροφορίας στην εποχή της πληροφορικής.

Ασκήσεις 1.1

1. Αναζητήστε πληροφορίες για τον al-Khorezmi (αλλιώς al-Khwarizmi), τον άνθρωπο από το όνομα του οποίου προήλθε ο όρος «αλγόριθμος». Επίσης, προσπαθήστε να βρείτε τα κοινά σημεία των ριζών των λέξεων «αλγόριθμος» και «άλγεβρα».

2. Με δεδομένο ότι ο θεσμικός στόχος του Αμερικάνικου Συστήματος Ευρεσιτεχνίας (US Patent Sustem) είναι η προαγωγή και ανάπτυξη των «χρήσιμων τεχνών» ("useful arts"), πιστεύετε ότι οι αλγόριθμοι επιδέχονται δικαιώματα ευρεσιτεχνίας, στη χώρα αυτή; Πώς θα πρέπει να ρυθμιστεί, κατά τη γνώμη σας, το θέμα αυτό;

3. (α΄) Καταγράψτε οδηγίες πλοήγησης από το σχολείο/εκπαιδευτήριο/Πανεπιστήμιό σας στο σπίτι σας, με την απαιτούμενη ακρίβεια ενός αλγορίθμου.

   (β΄) Καταγράψτε μια συνταγή παρασκευής ενός αγαπημένου σας φαγητού, με την απαιτούμενη ακρίβεια ενός αλγορίθμου.

4. Σχεδιάστε έναν αλγόριθμο για τον υπολογισμό του $\lfloor \sqrt{n} \rfloor$, για κάθε θετικό ακέραιο n. Εκτός από την εντολή ανάθεσης τιμής και την εντολή σύγκρισης, ο αλγόριθμός σας επιτρέπεται να χρησιμοποιήσει μόνο τις 4 βασικές αριθμητικές πράξεις.

5. (α΄) Βρείτε τον $gcd(31415, 14142)$, με εφαρμογή του αλγορίθμου του Ευκλείδη.

   (β΄) Εκτιμήστε πόσες φορές γρηγορότερος είναι ο αλγόριθμος του Ευκλείδη, στην εύρεση του $gcd(31415, 14142)$, από τον αλγόριθμο που βασίζεται στον έλεγχο διαδοχικών ακέραιων, από τον $min(m, n)$ και προς τα κάτω, έως τον $gcd(m, n)$.

6. Να αποδείξετε την ταυτότητα $gcd(m, n) = gcd(n, m \ mod \ n)$, για κάθε ζεύγος θετικών ακέραιων αριθμών m και n.

7. Ποιά είναι η συμπεριφορά του αλγορίθμου του Ευκλείδη για ένα ζεύγος αριθμών, στο οποίο ο πρώτος αριθμός είναι μικρότερος του δεύτερου; Ποιός είναι ο μέγιστος αριθμός επαναλήψεων που αυτό είναι δυνατό να συμβεί, κατά τη διάρκεια εκτέλεσης του αλγορίθμου, με μια τέτοια αρχική είσοδο;

8. (α΄) Ποιός είναι ο ελάχιστος αριθμός διαιρέσεων που γίνονται από τον αλγόριθμο του Ευκλείδη, για κάθε δυνατή είσοδο στο διάστημα $1 \leq m, n \leq 10$;

   (β΄) Ποιός είναι ο μέγιστος αριθμός διαιρέσεων που γίνονται από τον αλγόριθμο του Ευκλείδη, για κάθε δυνατή είσοδο στο διάστημα $1 \leq m, n \leq 10$;

9. (α΄) Ο αλγόριθμος του Ευκλείδη, όπως παρουσιάζεται στα *Στοιχεία*, χρησιμοποιεί αφαιρέσεις, αντί της ακέραιας διαίρεσης. Γράψτε έναν ψευδοκώδικα για αυτή την έκδοση του αλγορίθμου του Ευκλείδη.

(β') ***Το παιχνίδι του Ευκλείδη*** (*Euclid's game*) (βλ. [6]) ξεκινά με την αναγραφή δύο διαφορετικών μεταξύ τους θετικών ακέραιων, ως κεφαλές σε δύο στήλες. Κάθε παίκτης παίζει με τη σειρά του. Σε κάθε κίνηση, ο παίκτης πρέπει να γράψει στη στήλη του έναν θετικό αριθμό, ίσο με τη διαφορά δύο ήδη υπαρχόντων στον πίνακα αριθμών. Αυτός ο αριθμός θα πρέπει να είναι καινούριος, να μην έχει ξαναεμφανιστεί στον πίνακα. Ο παίκτης που αδυνατεί να γράψει έναν νέο αριθμό, χάνει το παιχνίδι. Θα διαλέγατε την αρχική κίνηση σε αυτό το παιχνίδι, ή θα προτιμούσατε τη δεύτερη;

10. ***Ο επαυξημένος αλγόριθμος του Ευκλείδη*** (*extended Euclid's algorithm*) καθορίζει όχι μόνο τον μέγιστο κοινό διαιρέτη d δύο θετικών ακέραιων m και n, αλλά επίσης και δύο ακέραιους (όχι αναγκαστικά θετικούς), x και y, τέτοιους ώστε $mx + ny = d$.

    (α') Αναζητήστε μια περιγραφή του επαυξημένου αλγορίθμου του Ευκλείδη (βλ. π.χ. [7], σελ 13) και υλοποιήστε τον σε μια γλώσσα της αρεσκείας σας.

    (β') Τροποποιήστε το πρόγραμμά σας αυτό, ώστε να βρίσκει λύσεις στη Διοφαντική εξίσωση $\alpha x + \beta y = c$, για οποιοδήποτε σύνολο ακέραιων συντελεστών α, β και c.

11. ***Πόρτες ντουλαπιών*** (*locker doors*) Υπάρχουν n ντουλάπια σε έναν προθάλαμο, διαδοχικά αριθμημένα από 1 έως και n. Αρχικά οι πόρτες των ντουλαπιών είναι όλες κλειστές. Πραγματοποιούμε n περάσματα, ξεκινώντας κάθε φορά από το ντουλάπι $\#1$. Στο i-οστό πέρασμα, $i = 1, 2, \ldots, n$, αλλάζουμε την κατάσταση (toggle) κάθε i-οστής πόρτας ντουλαπιού: εάν η πόρτα είναι ανοικτή, την κλείνουμε. Εάν είναι κλειστή, την ανοίγουμε. Για παράδειγμα, μετά το πρώτο πέρασμα, κάθε πόρτα είναι ανοιχτή. Στο δεύτερο πέρασμα πειράζουμε μόνο τα άρτια αριθμημένα ντουλάπια ($\#2, \#4 \ldots$), οπότε μετά το δεύτερο πέρασμα οι άρτιες πόρτες είναι κλειστές και οι περιττές ανοιχτές. Την τρίτη φορά, κλείνουμε την πόρτα του ντουλαπιού $\#3$ (που είχε ανοίξει στο πρώτο πέρασμα), ανοίγουμε την $\#6$ (που είχαμε κλείσει στο δεύτερο πέρασμα) κλπ. Μετά το τελευταίο πέρασμα, ποιές και πόσες πόρτες είναι ανοιχτές ή κλειστές;

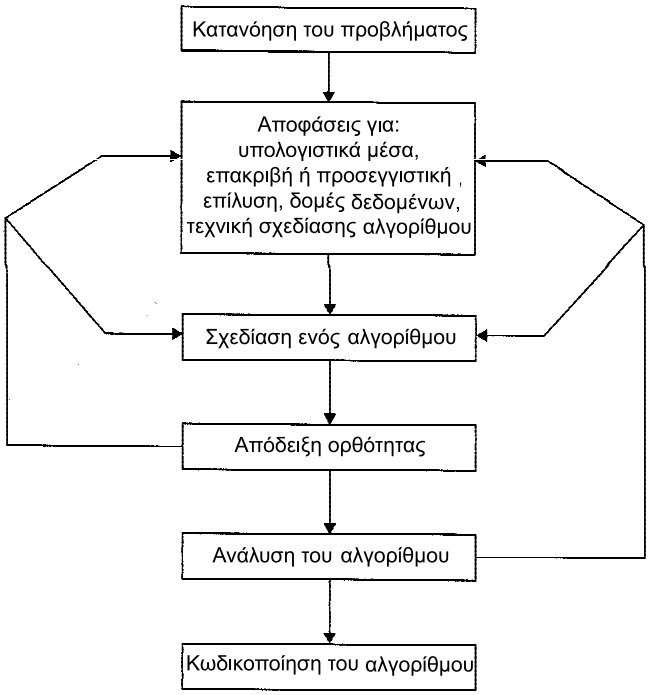
1.2 Βασικες Εννοιες Αλγοριθμικης Επιλυσης Προβληματων

Ας ξεκινήσουμε επαναλαμβάνοντας ένα σημαντικό σημείο, το οποίο αναφέραμε στην εισαγωγή του βιβλίου:

Μπορούμε να θεωρήσουμε τους αλγόριθμους ως διαδικαστικές λύσεις (procedural solutions) για διάφορα προβλήματα.

Οι λύσεις αυτές δεν αποτελούν απαντήσεις, αλλά ειδικές εντολές για τη λήψη απαντήσεων. Αυτή η έμφαση στις επακριβώς καθορισμένες κατασκευαστικές διαδικασίες, είναι που κάνει την επιστήμη των υπολογιστών να ξεχωρίζει από όλες τις άλλες τέχνες και επιστήμες. Συγκεκριμένα, έτσι ξεχωρίζει από τα θεωρητικά μαθηματικά, όπου οι θεωρητικοί μαθηματικοί είναι τυπικά εντάξει εάν αποδείξουν απλά την ύπαρξη μιας λύσης σε ένα πρόβλημα και, πιθανώς, εξερευνήσουν τις ιδιότητες της λύσης αυτής.

Παραθέτουμε τώρα (και συζητούμε εν συντομία) την ακολουθία των βημάτων που συνήθως υλοποιούμε, κατά τη σχεδίαση και την ανάλυση ενός αλγορίθμου (βλ. Σχήμα 1.2).



Σχήμα 1.2: Διαδικασία σχεδίασης και ανάλυσης ενός αλγορίθμου.

1.2.1 Κατανόηση του Προβλήματος

Από πρακτική σκοπιά, το πρώτο πράγμα που πρέπει να κάνετε, πριν σχεδιάσετε έναν αλγόριθμο, είναι να κατανοήσετε πλήρως το δεδομένο πρόβλημα. Διαβάστε την περι-

γραφή του προβλήματος προσεκτικά και ζητήστε διευκρινίσεις, εάν υπάρχουν απορίες σχετικά με το πρόβλημα, δουλέψτε χειροκίνητα μερικά μικρά παραδείγματά του, σκεφτείτε κάποιες ειδικές περιπτώσεις και ρωτήστε πάλι, εάν κάτι δεν είναι πλήρως κατανοητό.

Υπάρχουν μερικοί τύποι προβλημάτων, τα οποία εμφανίζονται αρκετά συχνά σε υπολογιστικές εφαρμογές. Τους συζητούμε στην επόμενη Ενότητα. Εάν το συγκεκριμένο πρόβλημα ανήκει σε έναν από αυτούς, ενδέχεται να είστε σε θέση να χρησιμοποιήσετε έναν γνωστό αλγόριθμο για να το επιλύσετε. Προφανώς, είναι χρήσιμο να γνωρίζετε το πώς ακριβώς δουλεύει ένας τέτοιος αλγόριθμος, ποιά είναι τα δυνατά και τα αδύνατά του σημεία, ιδιαίτερα εάν θα πρέπει να επιλέξετε μεταξύ εναλλακτικών αλγορίθμων. Πολλές φορές όμως, δεν θα βρείτε έναν άμεσα διαθέσιμο αλγόριθμο και θα πρέπει να σχεδιάσετε έναν δικό σας. Η ακολουθία των βημάτων που περιγράφεται στην Ενότητα αυτή θα σας βοηθήσει σε αυτό το συναρπαστικό, αλλά όχι πάντα εύκολο, έργο.

Μια είσοδος σε έναν αλγόριθμο καθορίζει μια ***έκφανση*** (*instance*) του προβλήματος που προσπαθεί να λύσει ο αλγόριθμος. Ο επακριβής καθορισμός του εύρους εκφάνσεων που απαιτούμε από τον αλγόριθμο να χειριστεί, είναι ζωτικής σημασίας. (Ως παράδειγμα αυτού, θυμηθείτε τις διαφορές στο εύρος των εκφάνσεων για τους τρεις αλγόριθμους υπολογισμού του μέγιστου κοινού διαιρέτη, τους οποίους συζητήσαμε στην προηγούμενη Ενότητα.) Εάν δεν γίνει επακριβής καθορισμός, ο αλγόριθμος ενδέχεται να δουλεύει σωστά για την πλειονότητα των εισόδων του, αλλά να παρουσιάζει πρόβλημα (crash) σε ορισμένες «οριακές» περιπτώσεις. Θυμηθείτε ότι σωστός αλγόριθμος δεν είναι αυτός που δουλεύει καλά στις περισσότερες των περιπτώσεων, αλλά αυτός που δουλεύει καλά για *όλες* τις αποδεκτές εισόδους.

Μην παραβλέπετε αυτό το πρώτο βήμα της διαδικασίας αλγοριθμικής επίλυσης ενός προβλήματος: εάν το κάνετε, διατρέχετε τον κίνδυνο να ξαναδουλέψετε από την αρχή τον αλγόριθμό σας.

1.2.2 Καθορίζοντας τις Δυνατότητες μιας Υπολογιστικής Συσκευής

Από τη στιγμή που κατανοήσετε πλήρως ένα πρόβλημα, θα πρέπει να καθορίσετε τις δυνατότητες της υπολογιστικής συσκευής, για την οποία προορίζεται ο αλγόριθμος. Η συντριπτική πλειοψηφία των αλγορίθμων που βρίσκονται σήμερα σε χρήση εξακολουθούν να είναι σχεδιασμένοι για προγραμματισμό και εκτέλεση σε έναν υπολογιστή, ο οποίος λειτουργεί ως μια μηχανή von Neumann—ένας τύπος αρχιτεκτονικής υπολογιστών εμπνευσμένη από τον μεγάλο Ουγγρο-Αμερικάνο μαθηματικό John von Neumann (1903–1957), σε συνεργασία με τους A.Burks και H. Goldstine, στα 1946. Η ουσία της αρχιτεκτονικής αυτής αναφέρεται στη λεγόμενη ***μηχανή τυχαίας προσπέλασης*** (*random-access machine, RAM*). Η κεντρική της υπόθεση είναι ότι οι εντολές εκτελούνται η μια μετά την άλλη, επιτελώντας μία λειτουργία κάθε φορά. Ως επακόλουθο, οι αλγόριθμοι που σχεδιάζονται για εκτέλεση σε τέτοιου είδους μηχανές ονομάζονται ***ακολουθιακοί αλγόριθμοι*** (*sequential algorithms*).

Η κεντρική υπόθεση του μοντέλου RAM δεν ισχύει για μερικούς νεότερους υπολογιστές, οι οποίοι εκτελούν παράλληλα διάφορες λειτουργίες. Οι αλγόριθμοι που εκμεταλλεύονται αυτή τη δυνατότητα, ονομάζονται ***παράλληλοι αλγόριθμοι*** (*parallel algorithms*). Παρόλα αυτά, η μελέτη των κλασικών τεχνικών σχεδίασης και ανάλυσης αλγορίθμων, σύμφωνα με το μοντέλο RAM, παραμένει ο ακρογωνιαίος λίθος της αλγοριθμικής, τουλάχιστον για το ορατό μέλλον.

Θα πρέπει να αποτελεί αντικείμενο προβληματισμού σας η ταχύτητα και το μέγεθος της μνήμης του υπολογιστή που έχετε στη διάθεσή σας; Εάν σχεδιάζετε έναν αλγόριθμο ως επιστημονική άσκηση, η απάντηση είναι ένα ξεκάθαρο «όχι». Όπως θα δείτε στην Ενότητα 2.1, οι περισσότεροι επιστήμονες της πληροφορικής προτιμούν να μελετούν αλγόριθμους με όρους ανεξάρτητους του καθορισμού διαφόρων παραμέτρων για κάποιους συγκεκριμένους υπολογιστές. Εάν σχεδιάζετε έναν αλγόριθμο ως ένα πρακτικό εργαλείο, η απάντηση ενδέχεται να εξαρτάται από το πρόβλημα που έχετε να λύσετε. Ακόμη και οι «αργοί» σύγχρονοι υπολογιστές, είναι απίθανα γρήγοροι. Συνεπώς, σε πολλές περιπτώσεις, δεν χρειάζεται να ανησυχείτε για το εάν ένας υπολογιστής είναι αργός για τη συγκεκριμένη εργασία. Υπάρχουν όμως και σημαντικά προβλήματα, τα οποία εκ φύσεως είναι πολύ περίπλοκα, πρέπει να επεξεργαστούν τεράστιο όγκο δεδομένων, ή ασχολούνται με εφαρμογές όπου ο χρόνος είναι ο κρίσιμος παράγων. Σε τέτοιες περιπτώσεις, είναι βεβαίως αναγκαίο να γνωρίζουμε τις επιδόσεις σε ταχύτητα και μνήμη του συγκεκριμένου υπολογιστικού συστήματος που θα χρησιμοποιήσουμε.

1.2.3 Επιλογή Μεταξύ Ακριβούς και Προσεγγιστικής Επίλυσης του Προβλήματος

Η επόμενη ουσιώδης επιλογή είναι να αποφασίσουμε εάν θα λύσουμε επακριβώς το πρόβλημα, ή θα βρούμε κατά προσέγγιση τη λύση του. Στην πρώτη περίπτωση, ο αλγόριθμος καλείται ***ακριβής αλγόριθμος*** (*exact algorithm*). Στη δεύτερη, καλείται ***προσεγγιστικός αλγόριθμος*** (*approximation algorithm*). Για ποιό λόγο θα αποφάσιζε κάποιος υπέρ ενός προσεγγιστικού αλγορίθμου; Κατά πρώτον, υπάρχουν σημαντικά προβλήματα που απλά δεν μπορούν να λυθούν επακριβώς, για την πλειονότητα των εκφάνσεών τους: τέτοια παραδείγματα αποτελούν η εξαγωγή τετραγωνικών ριζών, η επίλυση μη-γραμμικών εξισώσεων, καθώς και ο υπολογισμός ορισμένων ολοκληρωμάτων (definite integrals). Κατά δεύτερον, οι υπάρχοντες αλγόριθμοι για την επακριβή επίλυση ενός προβλήματος ενδέχεται να είναι απαράδεκτα αργοί, εξαιτίας της ενδογενούς πολυπλοκότητας του προβλήματος. Αυτό συμβαίνει, συγκεκριμένα, σε προβλήματα τα οποία εμπλέκουν έναν πολύ μεγάλο αριθμό επιλογών: θα συναντήσουμε παραδείγματα τέτοιων προβλημάτων στα Κεφάλαια 3, 11 και 12. Και τρίτον, ένας προσεγγιστικός αλγόριθμος μπορεί να αποτελεί μέρος ενός πιο εξεζητημένου αλγορίθμου, ο οποίος επιλύει επακριβώς το πρόβλημα.

1.2.4 Επιλογή των Κατάλληλων Δομών Δεδομένων

Ορισμένοι αλγόριθμοι δεν απαιτούν ιδιαίτερο προβληματισμό, όσον αφορά την αναπαράσταση των εισόδων τους. Ορισμένοι άλλοι, όμως, οφείλουν την ίδια τους την ύπαρξη σε μια τέτοια έξυπνη χρήση συγκεκριμένων δομών δεδομένων. Επιπλέον, μερικές από τις τεχνικές σχεδίασης αλγορίθμων που θα συζητήσουμε στα Κεφάλαια 6 και 7, εξαρτώνται σε μεγάλο βαθμό από τη δόμηση (structuring) ή την αναδόμηση (restructuring) των δεδομένων που καθορίζουν μια έκφανση ενός προβλήματος. Αρκετά χρόνια πριν, ένα βιβλίο που άσκησε μεγάλη επιρροή, δήλωνε εμφατικά αυτή τη θεμελειώδη σημασία των αλγορίθμων και των δομών δεδομένων, για τον προγραμματισμό των υπολογιστών, ακόμη και στον τίτλο του: *Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα* [8]. Στη σύγχρονη εποχή του αντικειμενοστραφούς προγραμματισμού, οι δομές δεδομένων παραμένουν ζωτικής σημασίας για την ανάλυση και τη σχεδίαση αλγορίθμων. Αναφέρουμε συνοπτικά τις βασικές δομές δεδομένων στην Ενότητα 1.4.

1.2.5 Τεχνικές Σχεδίασης Αλγορίθμων

Τώρα πλέον, με όλα τα απαραίτητα εφόδια για την αλγοριθμική επίλυση προβλημάτων στα χέρια σας, πώς μπορείτε να σχεδιάσετε έναν αλγόριθμο, για να λύσετε ένα δεδομένο πρόβλημα; Αυτή είναι η κεντρική ερώτηση, στην οποία αποσκοπεί να απαντήσει το βιβλίο αυτό, προσπαθώντας να σας διδάξει μερικές γενικές τεχνικές σχεδίασης.

Τί είναι λοιπόν μια τεχνική σχεδίασης αλγορίθμων;

> Μια ***τεχνική σχεδίασης αλγορίθμων*** (*algorithm design technique*), ή «στρατηγική», ή «υπόδειγμα», είναι μια γενική προσέγγιση επίλυσης προβλημάτων με αλγοριθμικό τρόπο, η οποία εφαρμόζεται σε μια ευρεία ποικιλία προβλημάτων που ανακύπτουν σε διάφορους τομείς της επιστήμης της πληροφορικής.

Μια ματιά στον πίνακα περιεχομένων του βιβλίου αυτού θα σας πείσει ότι στην πλειοψηφία τους τα Κεφάλαιά του είναι αφιερωμένα σε ξεχωριστές, επιμέρους τεχνικές σχεδίασης. Καθεμιά τους είναι το απόσταγμα ορισμένων βασικών ιδεών που αποδείχθηκαν χρήσιμες στη σχεδίαση αλγορίθμων. Η εκμάθηση των τεχνικών αυτών είναι κεφαλαιώδους σημασίας, για τους εξής λόγους:

Κατά πρώτον, παρέχουν καθοδήγηση στη σχεδίαση αλγορίθμων για καινούρια προβλήματα, δηλαδή για τα οποία δεν υπάρχει κανένας γνωστός ικανοποιητικός αλγόριθμος. Επομένως—για να παραφράσουμε κάποια σοφά λόγια—η εκμάθηση τέτοιων τεχνικών είναι σα να μαθαίνουμε να ψαρεύουμε, αντί να αρκεστούμε στα ψάρια που κάποιος μας έδωσε. Δεν περιμένουμε, φυσικά, ότι καθεμιά από αυτές τις γενικές τεχνικές θα μπορεί να εφαρμοστεί επιτυχώς σε κάθε πρόβλημα που θα συναντήσετε. Αυτές οι τεχνικές όμως, θεωρούμενες όλες μαζί, αποτελούν όντως μια ισχυρή συλλογή εργαλείων, των οποίων θα εκτιμήσετε την αξία και θα τις βρείτε ιδιαίτερα χρήσιμες, στη

διάρκεια των σπουδών σας και στις δουλειές σας, αργότερα.

Κατά δεύτερον, οι αλγόριθμοι αποτελούν τον ακρογωνιαίο λίθο της επιστήμης της πληροφορικής. Κάθε επιστήμη επιδιώκει να κατηγοριοποιήσει τα κύρια γνωστικά της αντικείμενα και η πληροφορική δεν αποτελεί εξαίρεση. Οι τεχνικές σχεδίασης αλγορίθμων καθιστούν δυνατή την κατηγοριοποίηση των αλγορίθμων, ανάλογα με τις ενυπάρχουσες σχεδιαστικές ιδέες: μπορούν επομένως με φυσιολογικό τρόπο να χρησιμοποιηθούν τόσο για κατηγοριοποίηση, όσο και για μια σε βάθος μελέτη των αλγορίθμων.

1.2.6 Μέθοδοι Καθορισμού Ενός Αλγορίθμου

Έχοντας σχεδιάσει έναν αλγόριθμο, θα πρέπει να τον καθορίσετε με κάποιο τρόπο. Στην Ενότητα 1.1, για παράδειγμα, περιγράψαμε τον αλγόριθμο του Ευκλείδη με λόγια (σε μια ελεύθερη διατύπωση και σε μια μορφή βήμα-προς-βήμα), καθώς και με ψευδοκώδικα. Αυτές είναι και οι πλέον συνήθεις επιλογές σήμερα για τον καθορισμό αλγορίθμων.

Η χρήση της φυσικής γλώσσας είναι ιδιαίτερα βολική, σε πολλές περιπτώσεις: όμως, η ενδογενής αμφισημία οποιασδήποτε φυσικής γλώσσας καθιστά την ξεκάθαρη και επακριβή περιγραφή των αλγορίθμων ένα εκπληκτικά δύσκολο έργο. Παρόλα αυτά, το να μπορούμε να περιγράψουμε έναν αλγόριθμο, έστω και σε μια φυσική γλώσσα, είναι μια σημαντική δεξιότητα, την οποία θα πρέπει να επιδιώξετε να αναπτύξετε, κατά τη διαδικασία της εκμάθησης αλγορίθμων.

Ο ***ψευδοκώδικας*** (*pseudocode*) είναι ένα μίγμα φυσικής γλώσσας και δομών έκφρασης που συναντάμε στις γλώσσες προγραμματισμού. Ένας ψευδοκώδικας είναι συνήθως σαφέστερος από την αντίστοιχη περιγραφή σε φυσική γλώσσα και η χρήση του οδηγεί πολλές φορές σε ευκρινέστερη διατύπωση των αλγοριθμικών περιγραφών. Κάτι που προκαλεί έκπληξη, είναι το γεγονός ότι οι πληροφορικοί δεν έχουν συμφωνήσει σε μια αποδεκτή μορφή ψευδοκώδικα, με αποτέλεσμα οι συγγραφείς διδακτικών εγχειριδίων να σχεδιάζουν τις δικές τους «διαλέκτους». Ευτυχώς όμως, αυτές οι διάλεκτοι είναι τόσο κοντά μεταξύ τους, που η εξοικείωση με μια οποιαδήποτε σύγχρονη γλώσσα προγραμματισμού επιτρέπει την κατανόηση οποιασδήποτε από αυτές.

Η διάλεκτος του βιβλίου επιλέχθηκε με κριτήριο την ελαχιστοποίηση της δυσκολίας κατανόησης από τον αναγνώστη. Για λόγους απλότητας, παραλείπουμε τις δηλώσεις των μεταβλητών και χρησιμοποιούμε τη στοίχιση του κώδικα (indentation) για να δείξουμε την εμβέλεια των μεταβλητών, σε δομές όπως οι **for, if** και **while**. Όπως παρατηρήσατε στην προηγούμενη Ενότητα, χρησιμοποιούμε το αριστερό βέλος (←) για τη λειτουργία ανάθεσης τιμής και τις δύο καθέτους (//) για σχόλια της μιας γραμμής κώδικα.

Στην πρώιμη εποχή των υπολογιστών, το κυρίαρχο μέσο για τον καθορισμό των αλγορίθμων ήταν το ***διάγραμμα ροής*** (*flowchart*), μια μέθοδος έκφρασης του αλγορίθμου μέσω μιας συλλογής συνδεδεμένων γεωμετρικών σχημάτων, τα οποία περιείχαν περιγραφές των βημάτων του αλγορίθμου. Αυτή η τεχνική αναπαράστασης αποδείχθηκε άβολη για το σύνολο σχεδόν των αλγορίθμων, εκτός των πλέον απλών. Σήμερα, τα

διαγράμματα ροής τα συναντούμε μόνο σε παλαιά βιβλία αλγορίθμων.

Η τρέχουσα τεχνολογία αιχμής στους υπολογιστές δεν έχει φτάσει ακόμη στο σημείο όπου να είμαστε σε θέση να τροφοδοτήσουμε μια περιγραφή ενός αλγορίθμου (είτε σε φυσική γλώσσα, είτε σε ψευδοκώδικα), απευθείας σε έναν ηλεκτρονικό υπολογιστή. Αντί για αυτό, απαιτείται η μετατροπή σε ένα πρόγραμμα υπολογιστή, γραμμένο σε κάποια καθορισμένη γλώσσα προγραμματισμού. Μπορούμε να θεωρήσουμε ένα τέτοιο πρόγραμμα ως μια ακόμη μέθοδο καθορισμού του αλγορίθμου, αν και είναι προτιμότερο να το θεωρήσουμε ως μια υλοποίηση του αλγορίθμου αυτού.

1.2.7 Απόδειξη της Ορθότητας ενός Αλγορίθμου

Από τη στιγμή που ένας αλγόριθμος έχει καθοριστεί, θα πρέπει να αποδείξετε την ***ορθότητά*** *του* (*correctness*). Δηλαδή, θα πρέπει να αποδείξετε ότι ο αλγόριθμος παράγει το απαιτούμενο αποτέλεσμα, για κάθε αποδεκτή είσοδο, σε πεπερασμένο χρόνο. Για παράδειγμα, η ορθότητα του αλγορίθμου του Ευκλείδη για τον υπολογισμό του μέγιστου κοινού διαιρέτη, έπεται από την ορθότητα της ταυτότητας $gcd(m, n) = gcd(n, m \ mod \ n)$ (η οποία, με τη σειρά της, χρειάζεται να αποδειχθεί· βλ. Πρόβλημα 6 στις Ασκήσεις 1.1), την απλή παρατήρηση ότι ο δεύτερος από αυτούς τους αριθμούς γίνεται μικρότερος σε κάθε επανάληψη του αλγορίθμου και από το γεγονός ότι ο αλγόριθμος σταματά, όταν ο δεύτερος αριθμός γίνει μηδέν.

Για μερικούς αλγόριθμους, η απόδειξη της ορθότητάς τους είναι αρκετά εύκολη· για άλλους, ενδέχεται να είναι υπερβολικά πολύπλοκη. Μια συνηθισμένη μέθοδος απόδειξης της ορθότητας είναι η χρήση της μαθηματικής επαγωγής, επειδή οι επαναλήψεις μιας διαδικασίας του αλγορίθμου μπορούν με φυσιολογικό τρόπο να θεωρηθούν ως μια ακολουθία βημάτων, όπως αυτά που απαιτούνται από την επαγωγή. Αξίζει να σημειώσουμε ότι ενώ η παρακολούθηση βήμα-προς-βήμα (tracing) της απόδοσης ενός αλγορίθμου για κάποιες συγκεκριμένες εισόδους μπορεί να είναι μια διαδικασία από την οποία θα αποκομίσουμε πολλά οφέλη, αυτή εν τούτοις δεν αποδεικνύει οριστικά την ορθότητα του αλγορίθμου. Από την άλλη μεριά, για να δείξετε ότι ένας αλγόριθμος είναι λανθασμένος, αρκεί να βρείτε μία μόνο έκφανση της εισόδου του, για την οποία ο αλγόριθμος αποτυγχάνει. Εάν ο αλγόριθμος αποδειχθεί λανθασμένος, θα πρέπει είτε να τον ξανασχεδιάσετε, χρησιμοποιώντας και πάλι ως βάση τις ήδη ληφθείσες αποφάσεις, όσον αφορά τις δομές δεδομένων, την τεχνική σχεδίασης κλπ., είτε, σε μια πιο δραματική τροπή, να αναθεωρήσετε μία ή περισσότερες από τις αποφάσεις αυτές (βλ. Σχήμα 1.2).

Η έννοια της ορθότητας των προσεγγιστικών αλγορίθμων είναι λιγότερο άμεση, από αυτή για τους επακριβείς αλγόριθμους. Για έναν προσεγγιστικό αλγόριθμο, μας αρκεί συνήθως να είμαστε σε θέση να αποδείξουμε ότι το λάθος που παρήχθη από τον αλγόριθμο, δεν υπερβαίνει μια προκαθορισμένη τιμή. Παραδείγματα τέτοιας μελέτης θα βρείτε στο Κεφάλαιο 12.

1.2.8 Ανάλυση ενός Αλγορίθμου

Συνήθως επιδιώκουμε οι αλγόριθμοί μας να διαθέτουν ορισμένα χαρακτηριστικά. Μετά την ορθότητα, το πλέον σημαντικό από αυτά είναι η αποδοτικότητά του. Στην πραγματικότητα, υπάρχουν δύο είδη αποδοτικότητας: η χρονική και η χωρική. Η ***χρονική αποδοτικότητα*** (*time efficiency*) δείχνει το πόσο γρήγορα «τρέχει» ένας αλγόριθμος. Η ***χωρική αποδοτικότητα*** (*space efficiency*) δείχνει πόσο επιπλέον μνήμη απαιτεί ο αλγόριθμος. Στο Κεφάλαιο 2 αναπτύσσουμε ένα γενικό πλαίσιο, καθώς και επιμέρους τεχνικές, για την ανάλυση της αποδοτικότητας ενός αλγορίθμου.

Ένα άλλο επιθυμητό χαρακτηριστικό ενός αλγορίθμου είναι η ***απλότητά*** *του* (*simplicity*). Σε αντίθεση με την αποδοτικότητα, η οποία μπορεί να οριστεί επακριβώς και να διερευνηθεί με την ανάλογη μαθηματική αυστηρότητα, η απλότητα, όπως η ομορφιά, είναι σε σημαντικό βαθμό θέμα υποκειμενικής θεώρησης. Για παράδειγμα, οι περισσότεροι άνθρωποι θα συμφωνούσαν ότι ο αλγόριθμος του Ευκλείδη είναι απλούστερος από τη μαθητική διαδικασία για τον υπολογισμό του $gcd(m, n)$, αλλά δεν είναι ξεκάθαρο εάν ο αλγόριθμος του Ευκλείδη είναι απλούστερος και από τον αλγόριθμο διαδοχικού ελέγχου ακεραίων. Παρόλα αυτά, η απλότητα είναι ένα σημαντικό χαρακτηριστικό και αξίζει να προσπαθήσουμε για την ύπαρξή του στον αλγόριθμό μας. Γιατί; Επειδή οι απλοί αλγόριθμοι είναι ευκολότεροι στην κατανόησή τους και στον προγραμματισμό τους. Συνεπώς, τα προκύπτοντα προγράμματα περιέχουν συνήθως λιγότερα σφάλματα (bugs). Υπάρχει επίσης και η αδιαφιλονίκητη αισθητική της απλότητας. Μερικές φορές, οι απλούστεροι αλγόριθμοι είναι και πιο αποδοτικοί, σε σχέση με πιο σύνθετες εναλλακτικές επιλογές. Δυστυχώς όμως, αυτό δεν ισχύει πάντοτε, οπότε στην περίπτωση αυτή θα πρέπει να γίνει ένας προσεκτικός συμβιβασμός.

Ένα ακόμη επιθυμητό χαρακτηριστικό ενός αλγορίθμου είναι η ***γενικότητά*** *του* (*generality*). Υπάρχουν, στην πραγματικότητα, δύο θέματα εδώ: η γενικότητα του προβλήματος που επιλύει ο αλγόριθμος και το εύρος εισόδων που αυτός επιδέχεται. Για το πρώτο, παρατηρήστε ότι είναι μερικές φορές ευκολότερη η σχεδίαση ενός αλγορίθμου για ένα πρόβλημα, το οποίο τίθεται με γενικότερους όρους. Θεωρήστε, για παράδειγμα, το πρόβλημα του καθορισμού εάν δύο ακέραιοι είναι πρώτοι μεταξύ τους, δηλαδή ο μόνος κοινός τους διαιρέτης είναι η μονάδα. Είναι ευκολότερη η σχεδίαση ενός αλγορίθμου για ένα πιο γενικό πρόβλημα, αυτό του υπολογισμού του μέγιστου κοινού διαιρέτη δύο ακεραίων και, για την επίλυση του αρχικού προβλήματος, να ελέγξουμε απλά εάν ο ΜΚΔ είναι 1 ή όχι. Υπάρχουν όμως και περιπτώσεις, όπου η σχεδίαση ενός γενικότερου αλγορίθμου είναι περιττή, δύσκολη, ή και αδύνατη. Για παράδειγμα, είναι περιττή η ταξινόμηση μιας λίστας n ακεραίων για την εύρεση του μεσαίου όρου τους, ο οποίος είναι το $\lceil n/2 \rceil$-οστό μικρότερο στοιχείο της. Ένα άλλο παράδειγμα, είναι ο τύπος εύρεσης των ριζών ενός τριωνύμου: αυτός δεν μπορεί να γενικευθεί για την εύρεση των ριζών πολυωνύμων αυθαίρετου βαθμού.

Όσον αφορά το εύρος τιμών των εισόδων, το κύριο μέλημά σας θα πρέπει να είναι η σχεδίαση ενός αλγορίθμου που να μπορεί να χειριστεί το εύρος τιμών εισόδου, το οποίο

είναι φυσιολογικό για το συγκεκριμένο πρόβλημα. Για παράδειγμα, ο αποκλεισμός ακεραίων ίσων με 1 ως πιθανής εισόδου για έναν αλγόριθμο εύρεσης του μέγιστου κοινού διαιρέτη θα ήταν μη φυσιολογικός. Από την άλλη μεριά, αν και ο αναλυτικός τύπος εύρεσης των ριζών ενός τριωνύμου ισχύει και για μιγαδικούς συντελεστές, δεν υλοποιούμε αυτή τη δυνατότητα (στο παρόν επίπεδο γενικότητας, τουλάχιστον), εκτός εάν αυτό απαιτηθεί ρητά.

Εάν δεν είσαστε ικανοποιημένοι με την αποδοτικότητα, την απλότητα, ή τη γενικότητα του αλγορίθμου σας, θα πρέπει να «γυρίσετε στο σχεδιαστικό τραπέζι» και να επανασχεδιάσετε τον αλγόριθμο. Στην πραγματικότητα, ακόμη και εάν η αποτίμησή σας είναι θετική, η αναζήτηση και άλλων αλγοριθμικών τρόπων επίλυσης, αξίζει τον κόπο. Θυμηθείτε τους τρεις διαφορετικούς αλγόριθμους της προηγούμενης Ενότητας, για τον υπολογισμό του μέγιστου κοινού διαιρέτη: γενικά, δε θα πρέπει να περιμένετε να βρείτε τον καλύτερο αλγόριθμο με την πρώτη προσπάθεια. Θα πρέπει, τουλάχιστον, να προσπαθήσετε να βελτιώσετε τις λεπτομέρειες (fine-tune) του αλγορίθμου που ήδη έχετε. Για παράδειγμα, κάναμε μερικές βελτιώσεις στην υλοποίησή μας του κόσκινου του Ερατοσθένη, σε σύγκριση με τις αρχικές του προδιαγραφές, στην Ενότητα 1.1. (Μπορείτε να τις εντοπίσετε;) Καλό θα είναι να έχετε στο μυαλό σας την ακόλουθη παρατήρηση του Antoine de Saint-Exupéry, Γάλλου συγγραφέα, πιλότου και σχεδιαστή αεροπλάνων: «Ο σχεδιαστής καταλαβαίνει ότι άγγιξε την τελειότητα, όχι όταν δεν υπάρχει τίποτα πλέον να προσθέσει, αλλά όταν δεν υπάρχει τίποτα να αφαιρέσει».[1]

1.2.9 Κωδικοποίηση ενός Αλγορίθμου

Οι περισσότεροι αλγόριθμοι προορίζονται να υλοποιηθούν τελικά ως προγράμματα υπολογιστών. Ο προγραμματισμός ενός αλγορίθμου παρουσιάζει κινδύνους και ευκαιρίες. Ο κίνδυνος έγκειται στην πιθανότητα η μετάβαση από τον αλγόριθμο στο πρόγραμμα να γίνει είτε λανθασμένα, είτε μη αποδοτικά. Μερικοί επιστήμονες των υπολογιστών με επιρροή πιστεύουν ότι μέχρις ότου η ορθότητα ενός προγράμματος αποδειχθεί με μαθηματική αυστηρότητα, το πρόγραμμα δεν μπορεί να θεωρηθεί σωστό. Αυτοί έχουν αναπτύξει ειδικές τεχνικές για την επίτευξη τέτοιων αποδείξεων (βλ. [10]), όμως η καταλληλότητα εφαρμογής τέτοιων τεχνικών τυπικής επαλήθευσης της ορθότητας περιορίζεται μέχρι τώρα μόνο σε πολύ μικρά προγράμματα. Πρακτικά, η ορθότητα και εγκυρότητα των προγραμμάτων εξακολουθεί να διαπιστώνεται μετά από έλεγχο (testing). Ο έλεγχος των προγραμμάτων των υπολογιστών είναι μια τέχνη, όχι μια επιστήμη, αυτό όμως δε σημαίνει ότι δεν μπορούμε να μάθουμε τίποτε από αυτή. Ανατρέξτε σε βιβλία αφιερωμένα στον έλεγχο και την αποσφαλμάτωση (debugging) προγραμμάτων: ακόμη πιο σημαντικό, ελέγξτε και αποσφαλματώστε εσείς οι ίδιοι τα

[1] Βρήκα αυτή την επίκληση για απλότητα σε μια συλλογή δοκιμίων του Jon Bentley [9]: τα δοκίμια πραγματεύονται μια ποικιλία θεμάτων σχεδίασης και υλοποίησης αλγορίθμων, με τον εύστοχο τίτλο *Προγραμματιστικά Μαργαριτάρια* (*Programming Pearls*). Συνιστώ ανεπιφύλακτα τα γραπτά των Jon Bentley και Antoine de Saint-Exupéry.

προγράμματά σας, στο μεγαλύτερο δυνατό βαθμό, οποτεδήποτε υλοποιείτε έναν αλγόριθμο.

Σημειώστε επίσης ότι, σε όλη την έκταση του βιβλίου, υποθέτουμε ότι οι είσοδοι των αλγορίθμων εμπίπτουν στα προκαθορισμένα εύρη τιμών και επομένως δε χρειάζεται κάποια σχετική επαλήθευση για αυτό. Κατά την κατασκευή προγραμμάτων υλοποίησης αλγόριθμων προς χρήση σε πραγματικές εφαρμογές, θα πρέπει φυσικά να κάνετε τους ανάλογους ελέγχους.

Φυσικά, η ορθή υλοποίηση ενός αλγορίθμου είναι αναγκαία, αλλά όχι και ικανή συνθήκη: δε θα θέλατε να εκμηδενίσετε την ισχύ του αλγορίθμου σας, λόγω μιας μη-αποδοτικής υλοποίησης. Οι σύγχρονοι μεταγλωττιστές παρέχουν (ανάμεσα στις άλλες λειτουργίες τους) ένα «δίχτυ ασφαλείας» για το σκοπό αυτό, ειδικά όταν αυτοί λειτουργούν στην κατάσταση βελτιστοποίησης κώδικα. Όπως και να είναι όμως, θα πρέπει να είστε ενήμεροι σχετικά με τα τυποποιημένα τεχνάσματα, όπως ο υπολογισμός των αναλλοίωτων ποσοτήτων ενός βρόχου (εκφράσεων των οποίων δε μεταβάλλεται η τιμή) έξω από αυτόν, η ομαδοποίηση κοινών υπο-εκφράσεων, η αντικατάσταση υψηλών σε κόστος πράξεων με άλλες φθηνότερες κλπ. (Βλ. [11] και [9] για μια διδακτική συζήτηση, όσον αφορά τη βελτιστοποίηση κώδικα και άλλα θέματα σχετικά με τον προγραμματισμό αλγορίθμων.) Τυπικά, τέτοιες βελτιώσεις επιταχύνουν ένα πρόγραμμα κατά ένα σταθερό παράγοντα και μόνο, ενώ ένας καλύτερος αλγόριθμος μπορεί να δώσει μια διαφορά πολλών τάξεων μεγέθους στο χρόνο εκτέλεσης. Από τη στιγμή που επιλεγεί ένας συγκεκριμένος αλγόριθμος, μια βελτίωση της ταχύτητας (speedup) κατά 10-50% είναι βέβαια πολύτιμη.

Ένα πρόγραμμα που δουλεύει σωστά παρέχει μια επιπλέον ευκαιρία, στο ότι επιτρέπει μια εμπειρική ανάλυση του ενυπάρχοντος σε αυτό αλγορίθμου. Η ανάλυση αυτή βασίζεται στη χρονομέτρηση του προγράμματος για διάφορες εισόδους και στην κατοπινή ανάλυση των αποτελεσμάτων της χρονομέτρησης. Συζητούμε τα πλεονεκτήματα και τα μειονεκτήματα αυτής της προσέγγισης στην ανάλυση αλγορίθμων στην Ενότητα 2.6.

Ανακεφαλαιώνοντας, ας επαναλάβουμε επιγραμματικά το κύριο μάθημα που αποκομίσαμε από τη διαδικασία που απεικονίζεται στο Σχήμα 1.2:

Ως γενικός κανόνας, ένας καλός αλγόριθμος είναι το αποτέλεσμα επαναληπτικών προσπαθειών και ξαναδουλέματος.

Ακόμη και εάν βρεθείτε στην ευτυχή θέση να έχετε μια αλγοριθμική ιδέα, η οποία να φαντάζει πολύ ωραία, αυτό δε σημαίνει ότι δεν θα πρέπει να επιδιώκετε τη βελτίωσή της.

Στην πραγματικότητα, αυτά είναι καλά νέα, με την έννοια ότι το τελικό αποτέλεσμα γίνεται πολύ πιο απολαυστικό. (Ναι, εξέτασα το ενδεχόμενο να ονομάσω το βιβλίο *«Η Απόλαυση των Αλγορίθμων»* (*The Joy of Algorithms*).) Από την άλλη μεριά, πώς ξέρει κανείς πότε πρέπει να σταματήσει; Σε πραγματικές συνθήκες, το χρονοδιάγραμμα

του ερευνητικού σας προγράμματος ή το αφεντικό σας μπορούν να σας σταματήσουν. Και έτσι θα έπρεπε να είναι: η τελειότητα είναι ακριβή και, στην πραγματικότητα, δεν είναι πάντοτε απαιτητή. Η σχεδίαση ενός αλγορίθμου είναι μια δραστηριότητα που προσιδιάζει αυτή ενός μηχανικού, στην οποία απαιτούνται συμβιβασμοί μεταξύ αντιμαχόμενων τελικών στόχων, υπό τον περιορισμό των διαθέσιμων πόρων, με το χρόνο που μπορεί να αφιερώσει ο σχεδιαστής να αποτελεί έναν (σημαντικό) από αυτούς.

Στην ακαδημαϊκή κοινότητα, αυτή η ερώτηση οδηγεί σε μια ενδιαφέρουσα αλλά και συνήθως δύσκολη διερεύνηση της λεγόμενης ***βελτιστότητας*** (*optimality*) του αλγορίθμου. Στην πραγματικότητα, η ερώτηση δεν αφορά την αποδοτικότητα ενός αλγορίθμου, αλλά την πολυπλοκότητα του προβλήματος που αυτός επιλύει: ποιά είναι η ελάχιστη απαιτούμενη προσπάθεια που χρειάζεται να καταβάλλει *οποιοσδήποτε* αλγόριθμος, για να επιλύσει το συγκεκριμένο πρόβλημα; Για ορισμένα προβλήματα, η απάντηση στο ερώτημα αυτό είναι γνωστή. Για παράδειγμα, οποιοσδήποτε αλγόριθμος που ταξινομεί έναν πίνακα συγκρίνοντας τις τιμές των στοιχείων του, χρειάζεται περίπου $n \log_2 n$ συγκρίσεις, για πίνακες μεγέθους n (βλ. Ενότητα 11.2). Για πολλά φαινομενικά απλά προβλήματα όμως, όπως π.χ. για τον πολλαπλασιασμό πινάκων, οι επιστήμονες της πληροφορικής δεν έχουν καταφέρει να δώσουν μια τελική απάντηση.

Ένα άλλο σημαντικό θέμα στην αλγοριθμική επίλυση προληματων, είναι το ερώτημα του εάν και κατά πόσο κάθε πρόβλημα μπορεί να επιλυθεί με έναν αλγόριθμο. Δεν συζητάμε εδώ για προβλήματα τα οποία δεν επιδέχονται λύση, όπως π.χ. η προσπάθεια εύρεσης πραγματικών ριζών ενός τριωνύμου με αρνητική διακρίνουσα. Σε τέτοιες περιπτώσεις, ένα διευκρινιστικό μήνυμα, το οποίο αναφέρει ότι το συγκεκριμένο πρόβλημα δεν έχει λύση, είναι ό,τι μπορούμε (και θα έπρεπε) να περιμένουμε, από έναν αλγόριθμο. Ούτε αναφερόμαστε επίσης σε ασαφώς τεθέντα προβλήματα. Ακόμη και μερικά επακριβώς καθορισμένα, απαλλαγμένα ασαφειών προβλήματα, τα οποία θα περιμέναμε να έχουν μια απλή καταφατική ή αρνητική απάντηση, χαρακτηρίζονται ως «μη-αποφασίζοντα» (undecidable), δηλαδή μη δυνάμενα να λυθούν, από οποιοδήποτε αλγόριθμο. Ένα σημαντικό παράδειγμα του τύπου αυτού αναφέρεται στην Ενότητα 11.3. Ευτυχώς, η μεγάλη πλειοψηφία των προβλημάτων της πρακτικής πληροφορικής, επιδέχεται αλγοριθμικές λύσεις.

Προτού αφήσουμε την Ενότητα αυτή, ας σιγουρευτούμε ότι δεν σας έχει δημιουργηθεί η εσφαλμένη εντύπωση—η οποία πιθανώς να προκλήθηκε από την κάπως μηχανιστική απεικόνιση του Σχήματος 1.2—ότι η σχεδίαση ενός αλγορίθμου είναι μια ανιαρή διαδικασία. Τίποτε δεν είναι πιο λάθος από αυτό: η εφεύρεση (μήπως ανακάλυψη;) αλγορίθμων είναι μια πλέον δημιουργική διαδικασία, με πολλές ανταμοιβές. Το βιβλίο αυτό σχεδιάστηκε με σκοπό να σας πείσει ότι αυτό όντως είναι αληθές.

Ασκήσεις 1.2

1. ***Ο γρίφος του Παλαιού Κόσμου*** *(Old World puzzle)* Ένας χωρικός βρίσκεται στην ακροποταμιά με ένα λύκο, μια κατσίκα και ένα λάχανο. Πρέπει να περάσει απέναντι και τα τρία αυτά υπάρχοντά του, με τη βάρκα του. Η βάρκα όμως

χωράει μόνο αυτόν και ένα από τα υπάρχοντα (είτε το λύκο, είτε την κατσίκα, είτε το λάχανο). Εάν ο χωρικός βρίσκεται μακριά, ο λύκος θα φάει την κατσίκα και η κατσίκα, σε ανάλογη περίπτωση, θα φάει το λάχανο. Επιλύστε το πρόβλημα που αντιμετωπίζει ο χωρικός, ή αποδείξτε ότι είναι άλυτο. (Παρατήρηση: ο χωρικός είναι χορτοφάγος, αλλά δεν του αρέσει το λάχανο και επομένως δεν μπορεί να φάει ούτε την κατσίκα, ούτε το λάχανο, για να λύσει το πρόβλημα. Και, φυσικά, ο λύκος είναι προστατευόμενο είδος.)

2. ***Ο γρίφος του Νέου Κόσμου*** *(New World puzzle)* Τέσσερεις άνθρωποι θέλουν να διασχίσουν μια γέφυρα: αρχικά, βρίσκονται όλοι στην ίδια πλευρά της. Υπάρχει ένα όριο των 17 λεπτών, μετά το οποίο όλοι θα πρέπει να βρίσκονται στην άλλη άκρη της γέφυρας. Είναι νύχτα και διαθέτουν μόνο ένα φακό. Το πολύ δύο άτομα μπορούν να διασχίσουν ταυτόχρονα τη γέφυρα, σε κάθε χρονική στιγμή. Όποια ομάδα διασχίζει τη γέφυρα, είτε του ενός είτε των δύο ατόμων, θα πρέπει να έχει αυτή το φακό. Ο φακός μεταφέρεται από χέρι σε χέρι: δεν μπορεί π.χ. να πεταχθεί από κάποιον προς κάποιον άλλο. Ο άνθρωπος 1 χρειάζεται 1 λεπτό για να διασχίσει τη γέφυρα, ο 2 χρειάζεται 2 λεπτά, ο 3 θέλει 5 λεπτά και ο 4 χρειάζεται 10 λεπτά. Δύο άνθρωποι μαζί, κινούνται με την ταχύτητα του βραδύτερου από αυτούς. Για παράδειγμα, εάν ο 1 και ο 4 ξεκινήσουν μαζί, θα περάσουν 10 λεπτά μέχρι να φθάσουν στο άλλο άκρο της γέφυρας. Εάν ο 4 είναι αυτός που θα επιστρέψει το φακό, θα περάσουν συνολικά 20 λεπτά και η όλη αποστολή θα αποτύχει. (Σημείωση: σύμφωνα με φήμες που κυκλοφόρησαν στο Internet, οι συνεντευξιάζοντες μιας γνωστής εταιρείας παραγωγής λογισμικού, με έδρα κοντά στο Seattle, έθεσαν το πρόβλημα αυτό στους συνεντευξιαζόμενους.)

3. Ποιός από τους ακόλουθους τύπους μπορεί να θεωρηθεί ως αλγόριθμος για τον υπολογισμό του εμβαδού ενός τριγώνου, του οποίου οι πλευρές είναι δοσμένοι θετικοί αριθμοί a, b και c;

   (α΄) $S = \sqrt{p(p-a)(p-b)(p-c)}$, όπου $p = (a+b+c)/2$

   (β΄) $S = \frac{1}{2}bc\sin A$, όπου A είναι η γωνία μεταξύ των πλευρών b και c

   (γ΄) $S = \frac{1}{2}ah_a$, όπου h_a είναι το ύψος προς τη βάση a

4. Γράψτε ένα ψευδοκώδικα για έναν αλγόριθμο εύρεσης πραγματικών ριζών της εξίσωσης $ax^2 + bx + c = 0$, για αυθαίρετους πραγματικούς συντελεστές a, b και c. (Μπορείτε να υποθέσετε ότι υπάρχει διαθέσιμη η συνάρτηση τετραγωνικής ρίζας, $sqrt(x)$.)

5. Περιγράψτε τον τυποποιημένο αλγόριθμο εύρεσης της δυαδικής αναπαράστασης ενός θετικού δεκαδικού ακέραιου,

(α') στα Ελληνικά.

(β') σε ψευδοκώδικα.

6. Περιγράψτε τον αλγόριθμο που ακολουθεί το ΑΤΜ της τράπεζας με την οποία συνεργάζεστε, για την παροχή χαρτονομισμάτων. (Δώστε αυτή την περιγραφή είτε σε φυσική γλώσσα, είτε σε ψευδοκώδικα, όπως εσείς κρίνετε βολικό.)

7. (α') Μπορεί το πρόβλημα του υπολογισμού του π να επιλυθεί ακριβώς;

   (β') Πόσες εκφάνσεις διαθέτει το πρόβλημα αυτό;

   (γ') Ανατρέξτε στο Internet και βρείτε έναν τουλάχιστον αλγόριθμο για το πρόβλημα αυτό.

8. Δώστε ένα παράδειγμα προβλήματος, εκτός του μέγιστου κοινού διαιρέτη, για το οποίο γνωρίζετε περισσότερους από έναν αλγόριθμους επίλυσής του. Ποιός από αυτούς είναι απλούστερος; Ποιός είναι αποδοτικότερος;

9. Θεωρήστε τον ακόλουθο αλγόριθμο εύρεσης της ελάχιστης απόστασης μεταξύ των στοιχείων σε έναν πίνακα αριθμών:

Αλγόριθμος 1.2.1 : *MinDistance(*$A[0..n-1]$*)*

```
1   //Είσοδος: Πίνακας αριθμών A[0..n-1]
2   //Έξοδος: η ελάχιστη απόσταση μεταξύ δύο στοιχείων του πίνακα
3   dmin ← ∞
4   for i ← 0 to n − 1 do
5       for j ← 0 to n − 1 do
6           if i ≠ j and A[i]-A[j] < dmin
7               dmin ← A[i]-A[j]
8   return dmin
```

Μπορείτε ελεύθερα να επεμβείτε στον αλγόριθμο αυτόν και να κάνετε όποιες βελτιώσεις θέλετε, για καλύτερη αλγοριθμική επίλυση του προβλήματος. (Εάν θέλετε μάλιστα, αλλάξτε εντελώς τον αλγόριθμο αυτόν. Εάν όχι, βελτιώστε τη δοσμένη υλοποίησή του.)

10. Ένα βιβλίο που άσκησε μεγάλη επιρροή στην επίλυση προβλημάτων, με τίτλο *Πώς να Το Λύσετε* (*How to Solve It*), έχει γραφεί από τον Ουγγρο-Αμερικανό μαθηματικό George Polya (1887–1985). Ο Polya συνόψισε τις ιδέες του επιγραμματικά σε 4 σημεία. Αναζητήστε τα στο Internet, ή, ακόμη καλύτερα, στο ίδιο το βιβλίο του και συγκρίνετέ τα με το σχέδιο που σκιαγραφήσαμε στην Ενότητα 1.2. Ποιά είναι τα κοινά τους σημεία; Σε τί διαφέρουν;

1.3 Μερικοι Σημαντικοι Τυποι Προβληματων

Στην απέραντη θάλασσα των προβλημάτων που συναντά κανείς στην πληροφορική, υπάρχουν διάσπαρτες μερικές περιοχές, οι οποίες έχουν τραβήξει το ιδιαίτερο ενδιαφέρον των ερευνητών. Σε αδρές γραμμές, το ενδιαφέρον τους υποκινείται είτε από την πρακτική σημασία του προβλήματος, είτε από κάποια ιδιάζοντα χαρακτηριστικά που κάνουν το πρόβλημα να έχει ερευνητικό ενδιαφέρον. Ευτυχώς, αυτά τα δύο κίνητρα συχνά ενισχύουν το ένα το άλλο, στις περισσότερες των περιπτώσεων.

Στην Ενότητα αυτή, θα συζητήσουμε τους πλέον σημαντικούς τύπους προβλημάτων:

- Ταξινόμηση (sorting)
- Αναζήτηση (searching)
- Επεξεργασία συμβολοσειρών (string processing)
- Προβλήματα γραφημάτων (graph problems)
- Προβλήματα συνδυαστικής (combinatorial problems)
- Γεωμετρικά προβλήματα
- Αριθμητικά προβλήματα

Τα προβλήματα αυτά χρησιμοποιούνται σε επόμενα Κεφάλαια του βιβλίου για να καταδείξουν τις διάφορες τεχνικές σχεδίασης αλγορίθμων, καθώς και μεθόδους της αλγοριθμικής ανάλυσης.

1.3.1 Ταξινόμηση

Το ***πρόβλημα της ταξινόμησης*** (*sorting problem*) αναφέρεται στην αναδιάταξη των στοιχείων μιας δεδομένης λίστας, ώστε αυτά να βρεθούν σε αύξουσα σειρά. Φυσικά, για να έχει νόημα το πρόβλημα αυτό, η φύση των στοιχείων της λίστας θα πρέπει να είναι τέτοια, ώστε αυτά να επιδέχονται μια τέτοια διάταξη. (Ένας μαθηματικός θα έλεγε εδώ ότι θα πρέπει να υπάρχει μια σχέση ολικής διάταξης.) Σε πρακτικές περιπτώσεις, χρειάζεται να ταξινομήσουμε λίστες αριθμών, χαρακτήρων που ανήκουν σε ένα προκαθορισμένο αλφάβητο, συμβολοσειρών και, το πιο σημαντικό, εγγραφών (records) παρόμοιων με αυτές που διατηρούνται από σχολεία και εκπαιδευτήρια, σχετικά με τους μαθητές τους, με αυτές των βιβλιοθηκών σχετικά με τα βιβλία τους και με αυτές των εταιρειών σχετικά με τους υπαλλήλους τους. Στην περίπτωση των εγγραφών, θα πρέπει να επιλέξουμε κάποια συγκεκριμένη πληροφορία, η οποία θα καθοδηγήσει την ταξινόμηση. Για παράδειγμα, μπορούμε να επιλέξουμε την αλφαβητική ταξινόμηση των εγγραφών των φοιτητών κατά όνομα, ή κατά αριθμό μητρώου, ή κατά συνολικό

μέσο βαθμό επίδοσης στα μαθήματα. Μια τέτοια, ειδικά επιλεγμένη ποσότητα πληροφορίας, ονομάζεται ***κλειδί*** (*key*). Οι επιστήμονες της πληροφορικής συχνά μιλούν για την ταξινόμηση μιας λίστας κλειδιών, ακόμη και όταν τα στοιχεία της λίστας δεν είναι εγγραφές, παρά μόνο, ας πούμε, ακέραιοι αριθμοί.

Για ποιό λόγο επιθυμούμε την ταξινόμηση μιας λίστας; Η ταξινόμηση διευκολύνει κατά πολύ την απάντηση πολλών ερωτήσεων σχετικά με τη συγκεκριμένη λίστα. Η σημαντικότερη από αυτές αφορά την αναζήτηση: αυτός είναι και ο λόγος για τον οποίο τα λεξικά, οι τηλεφωνικοί κατάλογοι, τα μαθητολόγια κλπ. είναι ταξινομημένα. Θα δείτε και άλλες περιπτώσεις χρησιμότητας της λεγόμενης προ-ταξινόμησης μιας λίστας (list presorting), στην Ενότητα 6.1. Παρόμοια, η ταξινόμηση χρησιμοποιείται ως ένα βοηθητικό βήμα σε μερικούς σημαντικούς αλγόριθμους άλλων περιοχών, όπως π.χ. στους γεωμετρικούς αλγόριθμους.

Μέχρι τώρα, οι επιστήμονες της πληροφορικής έχουν ανακαλύψει *δεκάδες* αλγορίθμων ταξινόμησης. Στην πραγματικότητα, η εφεύρεση ενός νέου αλγορίθμου ταξινόμησης παρομοιάζεται με τη «σχεδίαση της ποντικοπαγίδας». Και βρίσκομαι στην ευχάριστη θέση να σας αναφέρω ότι το κυνήγι για μια καλύτερη ποντικοπαγίδα ταξινόμησης, συνεχίζεται με αμείωτο ενδιαφέρον. Αυτή η επιμονή αξίζει να επισημανθεί, ιδιαίτερα εάν αναλογιστούμε τα παρακάτω γεγονότα: Από τη μια μεριά, υπάρχουν μερικοί καλοί αλγόριθμοι ταξινόμησης, οι οποίοι ταξινομούν έναν πίνακα μεγέθους n, χρησιμοποιώντας συγκρίσεις της τάξης του $n \log_2 n$. Από την άλλη, κανένας αλγόριθμος που ταξινομεί με συγκρίσεις κλειδιών (σε αντίθεση με, ας πούμε, τη σύγκριση μικρών κομματιών των κλειδιών) δεν μπορεί να παράσχει σημαντική βελτίωση αυτής της απόδοσης.

Υπάρχει λόγος για την ύπαρξη αυτής της πληθώρας αλγορίθμων, στη «γη» της ταξινόμησης. Αν και μερικοί αλγόριθμοι είναι ομολογουμένως καλύτεροι από κάποιους άλλους, δεν υπάρχει ο ένας και μοναδικός αλγόριθμος, που να αντιπροσωπεύει τη βέλτιστη λύση σε όλες τις περιπτώσεις. Μερικοί από τους αλγόριθμους αυτούς είναι απλοί αλλά σχετικά αργοί και άλλοι είναι μεν γρηγορότεροι, αλλά και πιο περίπλοκοι· μερικοί αποδίδουν καλύτερα με τυχαία διατεταγμένες εισόδους, ενώ άλλοι δίνουν καλύτερα αποτελέσματα για σχεδόν ταξινομημένες λίστες. Ορισμένοι είναι κατάλληλοι μόνο για λίστες που βρίσκονται εξ' ολοκλήρου στη (γρήγορη) μνήμη του υπολογιστή, ενώ άλλοι μπορούν να προσαρμοστούν για την ταξινόμηση μεγάλων αρχείων που βρίσκονται στο δίσκο. Αυτή η κατηγοριοποίηση μπορεί να συνεχιστεί πολύ παραπέρα.

Δύο ιδιότητες των αλγορίθμων ταξινόμησης αξίζουν να συζητηθούν στο σημείο αυτό. Ένας αλγόριθμος ταξινόμησης καλείται ***ευσταθής*** (*stable*), εάν διατηρεί τη σχετική διάταξη δύο οποιωνδήποτε ίσων στοιχείων στην είσοδό του. Με άλλα λόγια, εάν μια λίστα εισόδου περιέχει δύο ίσα στοιχεία στις θέσεις i και j, με $i < j$, τότε αυτά θα βρίσκονται στην ταξινομημένη λίστα στις θέσεις i' και j', αντίστοιχα, με $i' < j'$. Αυτή η ιδιότητα είναι επιθυμητή, εάν, για παράδειγμα, διαθέτουμε μια λίστα από φοιτητές, η οποία είναι αλφαβητικά ταξινομημένη και θέλουμε να την ταξινομήσουμε και κατά

μέσο όρο επιδόσεων των φοιτητών: ένας ευσταθής αλγόριθμος θα παράξει μια λίστα, στην οποία οι φοιτητές με τον ίδιο συνολικό βαθμό θα είναι ταξινομημένοι αλφαβητικά. Μιλώντας γενικά, οι αλγόριθμοι που είναι ικανοί να εναλλάσσουν κλειδιά τα οποία βρίσκονται αρκετά μακριά μεταξύ τους, δεν είναι ευσταθείς, αλλά εκτελούνται συνήθως γρηγορότερα· θα δείτε αυτή τη γενική παρατήρηση να επαληθεύεται σε σημαντικούς αλγόριθμους ταξινόμησης, αργότερα στο βιβλίο.

Η δεύτερη αξιοσημείωτη ιδιότητα ενός αλγορίθμου ταξινόμησης, είναι η επιπλέον ποσότητα μνήμης που αυτός απαιτεί. Χαρακτηρίζουμε έναν αλγόριθμο ως ***επιτόπιο*** (*in place*), εάν αυτός δεν απαιτεί επιπλέον μνήμη, εκτός ίσως από ένα πολύ μικρό αριθμό μονάδων μνήμης. Ανάμεσα στους σημαντικούς αλγόριθμους ταξινόμησης, υπάρχουν και επιτόπιοι και άλλοι που δεν είναι τέτοιοι.

1.3.2 Αναζήτηση

Το ***πρόβλημα της αναζήτησης*** (*searching problem*) ασχολείται με την εύρεση μιας συγκεκριμένης τιμής, που ονομάζεται ***κλειδί της αναζήτησης*** (*search key*) εντός ενός δεδομένου συνόλου (ή πολυ-συνόλου (multiset), κάτι που επιτρέπει σε περισσότερα του ενός στοιχεία να έχουν την ίδια τιμή). Και εδώ υπάρχει μια πληθώρα αλγορίθμων αναζήτησης. Ποικίλλουν από την απλοϊκή σειριακή αναζήτηση, μέχρι τη θεαματικά αποδοτική, αλλά περιορισμένης εφαρμογής δυαδική αναζήτηση και τους εξειδικευμένους αλγόριθμους που βασίζονται στην αναπαράσταση του ενυπάρχοντος συνόλου σε μια μορφή περισσότερο κατάλληλη για τους σκοπούς της αναζήτησης. Αυτοί οι τελευταίοι έχουν ιδιαίτερη σημασία για κάποιες ρεαλιστικές εφαρμογές, όπου και αποδεικνύονται ανεκτίμητοι κατά την αποθήκευση και επανάκτηση πληροφοριών από μεγάλες βάσεις δεδομένων.

Και για την αναζήτηση, επίσης, δεν υπάρχει ο ένας και μοναδικός αλγόριθμος που να είναι ο καλύτερος σε κάθε περίπτωση. Μερικοί αλγόριθμοι είναι γρηγορότεροι από άλλους, αλλά απαιτούν περισσότερη μνήμη. Ορισμένοι είναι πολύ γρήγοροι, αλλά εφαρμόζονται μόνο σε ταξινομημένους πίνακες κλπ. Σε αντίθεση με τους αλγόριθμους ταξινόμησης, εδώ δεν υπάρχει το πρόβλημα της ευστάθειας, εγείρονται όμως άλλα θέματα. Συγκεκριμένα, σε εφαρμογές όπου τα ενυπάρχοντα δεδομένα αλλάζουν με μεγάλη συχνότητα, σχετικά με τον αριθμό των αναζητήσεων, η αναζήτηση θα πρέπει να θεωρηθεί σε συνδυασμό με δύο άλλες λειτουργίες: πρόσθεση στο (και αφαίρεση από το) σύνολο δεδομένων ενός στοιχείου. Σε τέτοιες περιπτώσεις, οι δομές δεδομένων και οι αλγόριθμοι θα πρέπει να επιλεγούν έτσι ώστε να επιτευχθεί μια ισορροπία μεταξύ των απαιτήσεων κάθε λειτουργίας. Επίσης, η οργάνωση πολύ μεγάλων συνόλων δεδομένων προς αποδοτική αναζήτηση, θέτει τις δικές της ειδικές προκλήσεις, με σημαντικές επιπτώσεις σε ρεαλιστικές εφαρμογές.

1.3.3 Επεξεργασία Συμβολοσειρών

Τα τελευταία χρόνια η μαζική εμφάνιση εφαρμογών που ασχολούνται με μη-αριθμητικά δεδομένα, έχει ενισχύσει το ενδιαφέρον των ερευνητών και αυτών που ασχολούνται πρακτικά με την πληροφορική, όσον αφορά τους αλγόριθμους χειρισμού συμβολοσειρών. Μια ***συμβολοσειρά*** (*string*) είναι μια ακολουθία χαρακτήρων, οι οποίοι ανήκουν σε ένα αλφάβητο. Συμβολοσειρές ιδιαίτερου ενδιαφέροντος είναι οι συμβολοσειρές χαρακτήρων κειμένου (text strings), οι οποίες αποτελούνται από γράμματα, αριθμούς και ειδικούς χαρακτήρες στίξης· οι συμβολοσειρές bits, που αποτελούνται από μηδενικά και μονάδες και οι ακολουθίες βιολογικών γονιδίων (gene sequences), οι οποίες μπορούν να μοντελοποιηθούν από συμβολοσειρές χαρακτήρων, από το αλφάβητο των τεσσάρων χαρακτήρων {A,C,G,T}. Θα πρέπει να τονίσουμε, όμως, ότι οι αλγόριθμοι επεξεργασίας συμβολοσειρών αποτελούσαν ήδη παλαιότερα ένα σημαντικό θέμα για την επιστήμη τς πληροφορικής, σε συνδυασμό με τις γλώσσες των υπολογιστών και την κατασκευή μεταγλωττιστών.

Ένα συγκεκριμένο πρόβλημα—αυτό της αναζήτησης μιας δεδομένης λέξης εντός ενός κειμένου—έχει τραβήξει το ιδιαίτερο ενδιαφέρον πολλών ερευνητών. Το αποκαλούμε ***ταίριασμα συμβολοσειρών*** (*string matching*). Έχουν εφευρεθεί μερικοί αλγόριθμοι, οι οποίοι εκμεταλλεύονται την ειδική φύση αυτού του τύπου αναζήτησης. Εισάγουμε έναν πολύ απλό τέτοιο αλγόριθμο, στο Κεφάλαιο 3 και συζητούμε δύο άλλους αλγόριθμους που βασίζονται σε μια αξιοσημείωτη ιδέα των R. Boyer και J. Moore, στο Κεφάλαιο 7.

1.3.4 Προβλήματα Γραφημάτων

Μια από τις παλαιότερες και πλέον ενδιαφέρουσες περιοχές της αλγοριθμικής, είναι οι αλγόριθμοι γραφημάτων. Σε μη αυστηρούς όρους, μπορούμε να σκεφτούμε ένα ***γράφημα*** (*graph*) ως μια συλλογή σημείων, τα οποία ονομάζονται κορυφές, μερικές από τις οποίες ενώνονται μεταξύ τους με γραμμές, οι οποίες ονομάζονται ακμές. (Ένας αυστηρότερος, τυπικός ορισμός, δίνεται στην επόμενη Ενότητα.) Τα γραφήματα αποτελούν ένα ενδιαφέρον αντικείμενο μελέτης, τόσο για θεωρητικούς, όσο και για πρακτικούς λόγους. Τα γραφήματα μπορούν να χρησιμοποιηθούν για τη μοντελοποίηση μιας μεγάλης ποικιλίας ρεαλιστικών εφαρμογών, συμπεριλαμβανομένων των δικτύων μεταφορών και επικοινωνιών, τη χρονοδρομολόγηση εργασιών (project scheduling), καθώς και μερικών παιχνιδιών. Μια αξιοσημείωτη πρόσφατη εφαρμογή είναι μια εκτίμηση της διαμέτρου του Παγκόσμιου Ιστού, η οποία ορίζεται ως ο μέγιστος αριθμός των συνδέσμων (links) που χρειάζεται κάποιος να διανύσει, έτσι ώστε να φτάσει από μια συγκεκριμένη ιστοσελίδα σε κάποια άλλη, ακολουθώντας την αμεσότερη διαδρομή μεταξύ των σελίδων αυτών.[2]

[2]Ο αριθμός αυτός, σύμφωνα με την εκτίμηση μιας ομάδας ερευνητών του Πανεπιστημίου της Notre Dame, είναι μόνο 19.