Κεφάλαιο 1

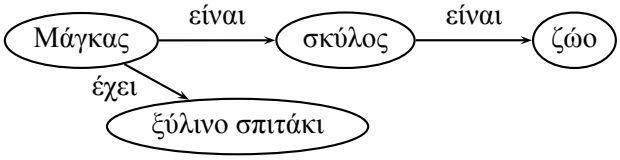
Εισαγωγή

Ο όρος *Επιστήμη των Υπολογιστών* (Computer Science) χρησιμοποιείται περισσότερο στην Αμερική. Στην Ευρώπη χρησιμοποιούμε τον όρο *Πληροφορική* (Informatics), ενώ στην Μεγάλη Βρετανία λένε συχνά *Επιστήμη των Υπολογισμών* (Computing Science).

Όπως είπε και ο διαπρεπής E. Dijkstra, η Επιστήμη των Υπολογιστών έχει την ίδια σχέση με τους υπολογιστές που έχει η Αστρονομία με τα τηλεσκόπια.

Σήμερα η Επιστήμη των Υπολογιστών άπτεται σχεδόν οποιασδήποτε ανθρώπινης δραστηριότητας. Η Πληροφορική είναι η συστηματική μελέτη αλγοριθμικών διαδικασιών που περιγράφουν και επεξεργάζονται πληροφορίες. Πιο συγκεκριμένα, η Επιστήμη των Υπολογισμών περιλαμβάνει: θεωρία, ανάλυση, σχεδίαση, αποδοτικότητα, υλοποίηση και εφαρμογή αλγοριθμικών διαδικασιών. Το βασικό ερώτημα με το οποίο ασχολείται η Επιστήμη των Υπολογιστών είναι τι μπορεί να *μηχανοποιηθεί* και μάλιστα με αποδοτικό τρόπο. Είναι κατά βάση *αφαιρετική* επιστήμη: Κατασκευάζουμε το σωστό μοντέλο για ένα πρόβλημα και επινοούμε τις κατάλληλες μηχανιστικές τεχνικές για την επίλυσή του.

Συχνά η εύρεση μίας καλής αφαίρεσης για ένα πρόβλημα μπορεί να είναι δύσκολη, λόγω των θεμελιωδών περιορισμών ως προς τις διαδικασίες (tasks) που μπορούν να επιτελέσουν οι υπολογιστές και την ταχύτητα με την οποία εκτελούν οι υπολογιστές αυτές τις διαδικασίες. Ένα δύσκολο τέτοιο πρόβλημα είναι η *αναπαράσταση γνώσης* (knowledge representation). Παρ' όλα αυτά έχουμε κάνει πρόοδο επινοώντας διάφορες αφαιρέσεις οι οποίες μας βοηθούν να κατασκευάσουμε προγράμματα που κάνουν ορισμένα είδη συλλογισμών. Μία τέτοια αφαίρεση είναι ο κατευθυνόμενος γράφος (directed graph· βλέπε σχήμα 1.1).



Σχήμα 1.1: Γράφος για την αναπαράσταση γνώσης.

'Αλλη μία χρήσιμη αφαίρεση είναι η *τυπική λογική*, η οποία μας επιτρέπει να χειριζόμαστε *γεγονότα* (facts) μέσω της εφαρμογής *συμπερασματικών κανόνων* (rules of inference).

Για την επίλυση προβλημάτων χρησιμοποιούνται τα παρακάτω εργαλεία:

1. *μοντέλα δεδομένων (data models)*, που είναι οι αφαιρέσεις που περιγράφουν το πρόβλημα, για παράδειγμα, όπως αναφέραμε προηγουμένως, η λογική και οι γράφοι·
2. *δομές δεδομένων (data structures)*, που είναι οι δομές των γλωσσών προγραμματισμού που παριστάνουν τα μοντέλα δεδομένων. Για παράδειγμα στην Pascal, οι πίνακες, οι εγγραφές, οι δείκτες μας επιτρέπουν να κατασκευάσουμε δομές δεδομένων που παριστάνουν πιο πολύπλοκες αφαιρέσεις, όπως οι γράφοι·
3. *αλγόριθμοι*, που είναι οι τεχνικές με τις οποίες παίρνουμε λύσεις για τα προβλήματα μέσω της επεξεργασίας των αντίστοιχων δομών δεδομένων.

1.1 Κλάδοι Επιστήμης των Υπολογιστών

Ακολουθεί μία πρόχειρη απαρίθμηση διαφόρων κλάδων της Επιστήμης των Υπολογιστών. Κοινό χαρακτηριστικό όλων τους είναι το περιεχόμενό τους: το θεωρητικό κομμάτι, η δημιουργία μοντέλων και το σχεδιαστικό – κατασκευαστικό τους κομμάτι:

1. Υπολογισιμότητα και Πολυπλοκότητα – Μοντέλα Υπολογισμού
2. Θεωρία Αυτομάτων και Τυπικών Γλωσσών
3. Αλγόριθμοι και Δομές Δεδομένων
4. Γλώσσες Προγραμματισμού και Μεταγλωττιστές
5. Αρχιτεκτονική Υπολογιστών και Δικτύων (hardware)
6. Αριθμητικοί και Συμβολικοί Υπολογισμοί
7. Λειτουργικά - Παράλληλα - Κατανεμημένα Συστήματα
8. Μεθοδολογία - Τεχνολογία Λογισμικού (software)
9. Βάσεις Δεδομένων και Διαχείριση Πληροφοριών
10. Τεχνητή Νοημοσύνη και Ρομποτική
11. Επικοινωνία ανθρώπου - υπολογιστή. Πολυμέσα
12. Κρυπτογραφία και ασφάλεια
13. Δίκτυα Επικοινωνιών - Ευφυή Δίκτυα - Διαδίκτυο
14. Υπολογιστική βιολογία

1.2 Επανάληψη, επαγωγή, αναδρομή

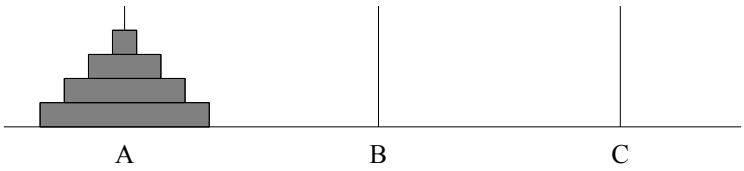
Θα συζητήσουμε διάφορες αρχές πάνω στις οποίες βασίζονται πολλές κατασκευές: π.χ. αλγεβρικές δομές όπως ’Αλγεβρες Boole και η Αναδρομική Μέθοδος.

1.2.1 Επανάληψη (Iteration)

Η ισχύς των υπολογιστών έγκειται στην δυνατότητά τους να εκτελούν με μεγάλη ταχύτητα *επαναληπτικά* την ίδια διαδικασία (πιθανώς παραμετροποιημένη). Ο απλούστερος τρόπος για να εκτελέσουμε μία διαδικασία επαναληπτικά είναι με μία δομή ελέγχου όπως η **for** της Pascal.

1.2.2 Αναδρομή (Recursion)

Αρκετά προβλήματα λύνονται εύκολα με την βοήθεια *αναδρομικών* διαδικασιών ή συναρτήσεων. Αναδρομικές είναι οι συναρτήσεις ή διαδικασίες που καλούν τον εαυτό τους μία ή περισσότερες φορές προκειμένου να λύσουν σχετικά υποπροβλήματα. Στον ορισμό της αναδρομικής διαδικασίας διαπιστώνουμε ότι το αρχικό πρόβλημα διασπάται σε μικρότερα προβλήματα του ίδιου τύπου με το αρχικό.



Σχήμα 1.2: Πύργοι του Ανόι ($n = 4$).

Ένα πρόβλημα για την λύση του οποίου είναι πιο εύκολο να κατασκευάσουμε έναν αναδρομικό από ότι έναν επαναληπτικό αλγόριθμο είναι οι “Πύργοι του Ανόι”: Έχουμε τρεις πασάλους, έστω A, B, C, και n δίσκους, που είναι όλοι διαφορετικοί σε μέγεθος μεταξύ τους. Αρχικά όλοι οι δίσκοι, οι οποίοι έχουν μία τρύπα στην μέση, ούτως ώστε να περνάνε στους πασάλους, βρίσκονται περασμένοι στον πάσαλο A, έτσι που να μην υπάρχει μικρότερος δίσκος κάτω από κάποιον μεγαλύτερό του (βλέπε σχήμα 1.2). Σκοπός είναι να μετακινήσουμε όλους τους δίσκους από τον πάσαλο A στον πάσαλο B, έναν κάθε φορά έτσι ώστε ποτέ να μην βρεθεί (στον ίδιο πάσαλο) μικρότερος δίσκος κάτω από μεγαλύτερο, χρησιμοποιώντας τον (βοηθητικό) πάσαλο C.

Ένας αναδρομικός αλγόριθμος:

```
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)
```

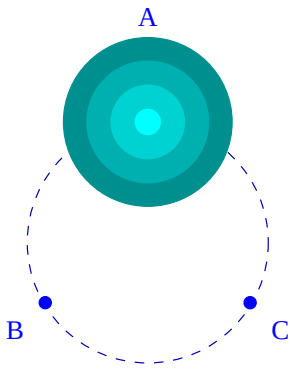
end
end

Για να λύσουμε το πρόβλημα του σχήματος 1.2 αρκεί να εκτελέσουμε την διαδικασία ως εξής: move_anoi(4 from A to B using C)

'Ασκηση: Δείξτε ότι ο ακόλουθος επαναληπτικός αλγόριθμος λύνει το πρόβλημα των πύργων του Ανόι:
Θεωρούμε ότι οι πάσαλοι είναι τοποθετημένοι επί κύκλου όπως στο σχήμα 1.3. Ο επαναληπτικός αλγόριθμος λειτουργεί ως εξής:

Επαναλάμβανε συνέχεια τα παρακάτω βήματα μέχρι να μην μπορείς να εκτελέσεις το δεύτερο βήμα:

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



Σχήμα 1.3: Πύργοι του Ανόι ($n = 4$): για τον επαναληπτικό αλγόριθμο.

1.2.3 Επαγωγή (Induction)

Η *επαγωγή* είναι μία σημαντική μέθοδος για την απόδειξη διαφόρων προτάσεων. Είναι πολύ χρήσιμη κατά την απόδειξη της ορθότητας των προγραμμάτων. Επίσης, η επαγωγή χρησιμοποιείται στον λεγόμενο επαγωγικό τρόπο ορισμού, π.χ.: "Μία λίστα είναι είτε η κενή λίστα είτε ένα στοιχείο ακολουθούμενο από μία λίστα".

1.2.4 Μερική και ολική ορθότητα

Στα προγράμματα συνήθως διακρίνουμε τρία επίπεδα ορθότητας: **συντακτική** ορθότητα (αντιστοιχεί στο context free κομμάτι της γλώσσας), **νοηματική** ορθότητα (αντιστοιχεί στο context sensitive κομμάτι της γλώσσας) και **σημασιολογική** ορθότητα. Στα προγράμματα ο compiler

ελέγχει τα 2 πρώτα επίπεδα ορθότητας. Η σημασιολογική ορθότητα δεν μπορεί φυσικά να ελεγχθεί από τον compiler και άρα ελέγχεται με δοκιμές (testing) ή με μαθηματική επαλήθευση (verification) σε σχέση με κάποιες προδιαγραφές (specifications). Γενικά, επιδιώκουμε ένα πρόγραμμα να έχει απλή δομή, δηλαδή να αποτελείται από δομικά στοιχεία (building blocks), επαναχρησιμοποιούμενες ενότητες (modules) για τις οποίες έχει ήδη γίνει αυστηρή επαλήθευση ορθότητας. Υπάρχουν πολλών ειδών σημασιολογίες προγραμμάτων (program semantics):

- **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.

Θα περιοριστούμε σε μια ελάχιστη περιγραφή της τρίτης. Διάφορες ιδιότητες βεβαιώνονται (are asserted) σε ορισμένα σημεία της ροής του προγράμματος. Αυτές οι συνθήκες που βεβαιώνονται, και πρέπει να αποδειχθεί ότι ικανοποιούνται, λέγονται **βεβαιώσεις** (assertions). Οι βεβαιώσεις βρόχου λέγονται **αναλλοίωτες του βρόχου** (loop invariants).

Δεν αρκεί να αποδείξουμε την ορθότητα σε περίπτωση τερματισμού. Η απόδειξη ορθότητας σε περίπτωση τερματισμού λέγεται **μερική ορθότητα** (partial correctness). Για την **ολική ορθότητα** (total correctness) χρειάζεται και απόδειξη τερματισμού. **Συνθήκη τερματισμού** (termination condition) λέγεται μια γνησίως φθίνουσα θετική συνάρτηση $f(t)$ που εγγυάται τον τερματισμό όταν $f(t) = 0$ (όπου t ο χρόνος, ο αριθμός των βημάτων που έχουν εκτελεστεί).

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).

1.3 Δομημένος προγραμματισμός και modularity

Η ιδεολογία του δομημένου προγραμματισμού υποστηρίζει ότι για κάθε ανεξάρτητη συγκεκριμένη λειτουργία πρέπει να γράφεται μια ανεξάρτητη διαδικασία. Θα πρέπει λοιπόν να αναλύεται όσο γίνεται το πρόβλημα σε ανεξάρτητα υποπροβλήματα και για το καθένα από αυτά να γράφεται κάποιο υποπρόγραμμα (διαδικασία ή συνάρτηση). Επίσης, πρέπει να αποφεύγεται η χρήση παρενεργειών. Αυτό σημαίνει ότι κάθε υποπρόγραμμα πρέπει να κάνει μια συγκεκριμένη λειτουργία, χωρίς να επηρεάζει το κυρίως πρόγραμμα ή άλλα υποπρογράμματα. Η επικοινωνία των επιμέρους μονάδων (πέρασμα τιμών κ.τ.λ.) γίνεται με χρήση παραμέτρων (βλέπε παρακάτω). Έτσι, το πρόγραμμα γίνεται ευανάγνωστο και είναι εύκολη η μεταφορά και η χρήση των υποπρογραμμάτων σε άλλα προγράμματα .

- Είναι ευκολότερο να γράψουμε ένα δομημένο πρόγραμμα, επειδή πολύπλοκα προβλήματα διασπώνται σε έναν αριθμό μικρότερων, απλούστερων εργασιών.

- Είναι ευκολότερο να ανιχνεύουμε λάθη σε δομημένο πρόγραμμα.
- Ένα σχετικό πλεονέκτημα είναι ο χρόνος που εξοικονομείται με δομημένα προγράμματα. Μπορούν να διορθωθούν ή να τροποποιηθούν πιο εύκολα. Όταν έχουμε κατασκευάσει μια διαδικασία που κάνει κάτι συγκεκριμένο, θα μπορούμε να τη χρησιμοποιούμε σε άλλα προγράμματα δίχως να σπαταλάμε χρόνο σε συγγραφή νέων υποπρογραμμάτων.

Σε ορισμένες γλώσσες, όπως η Modula-2, αλλά και στις περισσότερες σύγχρονες γλώσσες, υποστηρίζεται η διάσπαση του κώδικα σε διαφορετικές αλληλοσυνεργαζόμενες **ενότητες** οι οποίες ονομάζονται **modules**. Κάθε μια από τις ενότητες αυτές περιέχει τα δικά της αντικείμενα (σταθερές, τύπους, μεταβλητές, υποπρογράμματα). Μια τέτοια ενότητα μπορεί να αναφέρεται σε (ή να καλεί) αντικείμενα που είτε είναι εσωτερικά σ' αυτήν είτε έχουν εισαχθεί, με αυστηρό τρόπο, από άλλες ενότητες. Βέβαια για να γίνει δυνατή η εισαγωγή τέτοιων αντικειμένων θα πρέπει να έχει γίνει επιτρεπτή η εξαγωγή τους στην ενότητα που έχουν οριστεί. Τελικά, ένα πρόγραμμα αποτελείται από ένα σύνολο ενοτήτων, εντελώς ανεξαρτήτων ως προς την υλοποίηση, οι οποίες όμως επικοινωνούν μεταξύ τους με αυστηρά καθορισμένο τρόπο. Έτσι είναι δυνατή η δημιουργία μιας "**βιβλιοθήκης**" **προγραμμάτων**, οριζόμενων από το χρήστη ώστε κώδικας που έχει γραφτεί μια φορά να μπορεί να χρησιμοποιείται από πολλά προγράμματα που δυνατόν να χρειάζονται τις ίδιες λειτουργίες. Ο βασικός λόγος για αυτή τη δόμηση *είναι η επιθυμία μας για την υλοποίηση μιας ιεραρχίας αφαίρεσης (abstraction) στην οποία modules που βρίσκονται σε υψηλότερο επίπεδο από κάποια άλλα να μπορούν να χρησιμοποιούν αντικείμενα που έχουν οριστεί σε αυτά χωρίς να χρειάζονται να γνωρίζουν τίποτα για τον τρόπο υλοποίησής τους (implementation hiding)*.

Απαγορεύοντας έτσι την πρόσβαση στο εσωτερικό της κάθε ενότητας μπορούμε να φτιάξουμε ενότητες οι οποίες (είμαστε σίγουροι ότι) λειτουργούν σωστά και έτσι, σε μετέπειτα στάδιο κατά την χρησιμοποίησή τους από άλλη ενότητα, θα έχουμε μικρύνει κατά πολύ την περιοχή αναζήτησης λαθών.

Εφόσον όμως κάθε ενότητα μπορεί να εισαγάγει αντικείμενα από διάφορες άλλες, ο μεταφραστής (compiler) θα πρέπει να έχει πρόσβαση στην περιγραφή της δομής των δεδομένων έτσι ώστε να εξασφαλίζεται η σωστή επικοινωνία. Οδηγούμαστε έτσι στην *περαιτέρω διάσπαση ενότητας σε μέρος ορισμού (definition part) και μέρος υλοποίησης (implementation part) τα οποία μπορούν να δημιουργηθούν ανεξάρτητα*.

1.4 Παράλληλες, ταυτόχρονες, κατανεμημένες διεργασίες

Γενικά, οι αλγόριθμοι στους οποίους αναφερθήκαμε ως τώρα εκτελούνται *σειριακά*, όπως λέμε, σε σε έναν υπολογιστή. Παρ' όλα αυτά υπάρχουν ορισμένα προβλήματα, τα οποία μπορούμε να λύσουμε, κατανέμοντας το υπολογιστικό φορτίο σε περισσότερους από έναν υπολογιστές/επεξεργαστές που λειτουργούν *παράλληλα*. Χάρη στον *ταυτοχρονισμό* (concurrency), μειώνεται σημαντικά ο χρόνος που παίρνουμε απάντηση στο πρόβλημα.

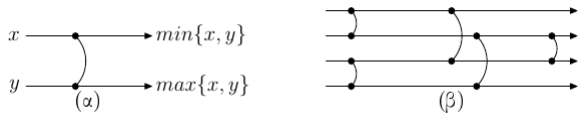
Πολλά προβλήματα σε αυτήν την περιοχή προκύπτουν από τις λεγόμενες κατανεμημένες (distributed) διεργασίες. Αυτές είναι διεργασίες που από την φύση τους εκτελούνται παράλληλα, για παράδειγμα οι συνδιαλέξεις σε ένα τηλεφωνικό δίκτυο.

Το μεγάλο πρόβλημα, στα παράλληλα συστήματα είναι πώς θα *παραλληλοποιήσουμε* έναν σειριακό αλγόριθμο. Για πολλούς αλγορίθμους, αυτό είναι εφικτό με σημαντική επιτυχία.

Παρ'όλα αυτά υπάρχουν και διεργασίες που είναι όπως λέμε *εγγενώς σειριακές* (inherently sequential). Σε αυτές, είναι τόσο έντονη η εξάρτηση ενός βήματος από τα προηγούμενα που δεν είναι δυνατόν τα βήματα να εκτελεστούν παράλληλα και ανεξάρτητα μεταξύ τους.

1.4.1 Δίκτυα ταξινόμησης

Σκοπός των δικτύων ταξινόμησης είναι η ταξινόμηση μίας ακολουθίας αριθμών χρησιμοποιώντας μόνο συγκρίσεις, με όσο το δυνατό μεγαλύτερη παραλληλία. Τα δίκτυα ταξινόμησης αναπαρίστανται με οριζόντιες γραμμές, μία για κάθε είσοδο. Οι συγκρίσεις μεταξύ δύο αριθμών αναπαρίστανται με κατακόρυφες συνδέσεις δύο γραμμών. Κάθε σύγκριση θεωρείται ότι "κοστίζει" μία χρονική μονάδα και η συνολική καθυστέρηση αντιστοιχεί στον αριθμό των κόμβων σε μια οριζόντια γραμμή. Η έξοδος είναι ταξινομημένη από "πάνω" προς τα "κάτω". Στο σχήμα 1.4(α) παρουσιάζεται ένας συγκριτής ενώ στο σχήμα 1.4(β) δίνεται ένα παράδειγμα δικτύου ταξινόμησης τεσσάρων εισόδων.



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

Η ποιότητα των δικτύων ταξινόμησης χαρακτηρίζεται από το χρόνο που χρειάζονται για να ταξινομήσουν τις εισόδους τους (*βάθος* δικτύου) και το πλήθος των συγκριτών που χρησιμοποιούν (*μέγεθος* δικτύου). Το βάθος του δικτύου ορίζεται ως το μέγιστο βάθος των γραμμών του, ενώ το βάθος μίας γραμμής είναι το πλήθος των συγκρίσεων που πραγματοποιούνται σ' αυτή. Για παράδειγμα, το βάθος και το μέγεθος του δικτύου του σχήματος 1.4(β) είναι 3 και 5 αντίστοιχα. Το μέγεθος αντιστοιχεί στον σειριακό χρόνο ταξινόμησης και το βάθος στον παράλληλο.

1.5 Ταξινόμηση treesort με δένδρο δυαδικής αναζήτησης

Θα δώσουμε έναν αλγόριθμο ταξινόμησης που βασίζεται στα δένδρα δυαδικής αναζήτησης (binary search tree).

Δίνεται μία λίστα αριθμών.

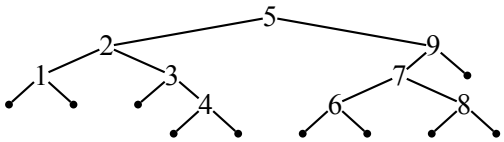
Παίρνουμε ένα ένα τα στοιχεία της λίστας με την σειρά και κατασκευάζουμε ένα δυαδικό δένδρο αναζήτησης. Αυτό κατασκευάζεται ως εξής: Το πρώτο στοιχείο της λίστας τοποθετείται στην ρίζα του δένδρου. Τα υπόλοιπα στοιχεία διασχίζουν το δένδρο μέσω των κόμβων μέχρι να βρουν την θέση τους σε κάποιο φύλλο του δένδρου ως εξής: αν το ήδη τοποθετημένο στοιχείο στον κόμβο είναι μεγαλύτερο από το στοιχείο που επιδιώκουμε να τοποθετήσουμε το προωθούμε στο δεξιό υποδένδρο του κόμβου, αλλιώς στο αριστερό. Επίσης, σε κάθε νέο στοιχείο

που τοποθετούμε στο δένδρο φροντίζουμε να δημιουργούμε δύο κόμβους παιδιά οι οποίοι είναι κενοί (empty).

Μετά διασχίζουμε το δένδρο με την λεγόμενη ενδοδιατεταγμένη (inorder) σειρά, οπότε προκύπτουν τα στοιχεία ταξινομημένα. Μία αναδρομική διαδικασία που τυπώνει τα στοιχεία του δένδρου σε ενδοδιατεταγμένη σειρά είναι η εξής:

```
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
```

Το δυαδικό δένδρο που προκύπτει από την εκτέλεση του αλγορίθμου για είσοδο: 5, 2, 3, 9, 7, 1, 8, 4, 6 φαίνεται στο σχήμα 1.5. Εκτελέστε ενδοδιατεταγμένη διάσχιση για να δείτε ότι όντως τα στοιχεία τυπώνονται ταξινομημένα.

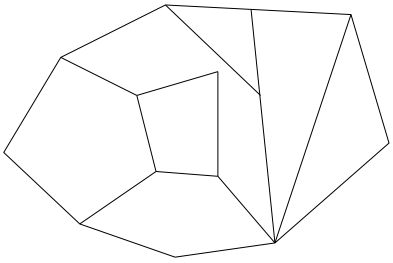


Σχήμα 1.5: Δένδρο αναζήτησης

Ο αλγόριθμος δεν είναι βέλτιστος. Η κατασκευή του δυαδικού δένδρου, αν αυτό δεν προκύψει ισορροπημένο, μπορεί να πάρει $O(n^2)$ βήματα.

1.6 Το θεώρημα τεσσάρων χρωμάτων (four color theorem)

Αναφερόμαστε στους γνωστούς επίπεδους χάρτες, όπως αυτός του σχήματος 1.6, όπου οι φραγμένες περιοχές αντιπροσωπεύουν χώρες.



Σχήμα 1.6: Επίπεδος χάρτης

Το ερώτημα είναι (δεδομένου ενός τέτοιου χάρτη):

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

Με την πρώτη ματιά, φαίνεται ότι το πλήθος των απαιτούμενων χρωμάτων μπορεί να γίνει πολύ μεγάλο αν δοθεί κάποιος ιδιαίτερα περίπλοκος χάρτης. Παρ' όλα αυτά, *είναι αρκετά τέσσερα χρώματα, σε κάθε περίπτωση*. Η παραπάνω πρόταση είναι γνωστή ως *θεώρημα τεσσάρων χρωμάτων*. Δοκιμάστε, για παράδειγμα, να χρωματίσετε τον χάρτη του σχήματος 1.6 με τέσσερα χρώματα.

Το πρόβλημα (αν αρκούν τέσσερα χρώματα) τέθηκε για πρώτη φορά το 1852 από τον Guthrie και αργότερα από τον Cayley το 1878. Ένα χρόνο μετά, ο Kempe παρουσίασε μία απόδειξη, η οποία όμως ήταν λανθασμένη. Σημειωτέον ότι η απόδειξη ότι αρκούν πέντε χρώματα είναι σχετικά εύκολη και την έδωσε ο Heawood το 1890, μεταβάλλοντας κάπως την λανθασμένη απόδειξη του Kempe. Από τότε, πολλοί επιδίωξαν να αποδείξουν το θεώρημα των τεσσάρων χρωμάτων.

Η πρώτη γενικά αποδεκτή απόδειξη δημοσιεύτηκε από τους Appel και Haken το 1977. Σε γενικές γραμμές, η απόδειξη λέει ότι αν είναι δυνατόν να χρωματιστεί κάθε ένας χάρτης από ένα πεπερασμένο σύνολο περιπτώσεων, τότε μπορεί να χρωματιστεί οποιοσδήποτε χάρτης. Το πρόβλημα είναι ότι πρέπει να αποδειχθεί για κάθε χάρτη από το πεπερασμένο σύνολο περιπτώσεων ότι αρκούν τέσσερα χρώματα και στην απόδειξη των Appel και Haken οι περιπτώσεις αυτές είναι περίπου 1700 το πλήθος και ορισμένες από αυτές αρκετά περίπλοκες. Λόγω του μεγάλου πλήθους των περιπτώσεων, ο χρωματισμός των περιπτώσεων γίνεται με την βοήθεια υπολογιστή. Επιπλέον και οι 1700 περίπου περιπτώσεις έχουν προκύψει με την βοήθεια υπολογιστή. Με λίγα λόγια, δεν φαίνεται να είναι δυνατόν να ελέγξει την απόδειξη κάποιος άνθρωπος χωρίς την βοήθεια του υπολογιστή και για αυτόν τον λόγο η απόδειξη των Appel και Haken δέχθηκε αρκετή κριτική. Η απόδειξη βασίζεται στην ορθότητα (βλέπε ενότητα 1.2.4) των χρησιμοποιηθέντων προγραμμάτων. Όμως, λόγω της περιπλοκότητας των προγραμμάτων, η ορθότητα τους δεν έχει πλήρως αποδειχθεί. Επιπλέον, αν θέλουμε να είμαστε απόλυτα αυστηροί, ούτε για τους μεταγλωττιστές με τους οποίους μεταφράστηκαν τα προγράμματα έχουμε απόδειξη ορθότητας.

Πάντως, η απόδειξη τελικά έγινε αποδεκτή, ειδικά επειδή επιπλέον έχουν εμφανιστεί και νεότερες αποδείξεις, όπως για παράδειγμα αυτή των Robertson, Sanders, Seymour, Thomas, οι οποίες μειώνουν κάπως τον αριθμό των εξεταζόμενων περιπτώσεων, αλλά και πάλι ο έλεγχος αν αρκούν τέσσερα χρώματα βασίζεται σε υπολογιστή.

1.7 Διαδραστικό υλικό - Σύνδεσμοι

- Η ιστοσελίδα http://towersofhanoi.info περιέχει animations και προσομοιώσεις για τον πρόβλημα των Πύργων του Hanoi.

1.8 Ασκήσεις

1. Αναδρομή – Επανάληψη – Επαγωγή:

   (α) Εκφράστε τον αριθμό μετακινήσεων δίσκων που κάνει ο αναδρομικός αλγόριθμος για τους πύργους του Ανόι, σαν συνάρτηση του αριθμού των δίσκων n.

   (β) Βρείτε τη σχέση ανάμεσα στον αριθμό των μετακινήσεων του αναδρομικού και τον αριθμό των μετακινήσεων του επαναληπτικού αλγορίθμου.

   (γ) Δείξτε ότι ο αριθμός των μετακινήσεων του αναδρομικού αλγορίθμου είναι ο ελάχιστος μεταξύ όλων των δυνατών αλγορίθμων για το πρόβλημα αυτό.

2. Δυαδική αναζήτηση:

   (α) Περιγράψτε (σε ψευδοκώδικα) διαδικασία / συνάρτηση treeinsert(T, k), που να δέχεται ένα δένδρο δυαδικής αναζήτησης T και έναν αριθμό k, και να εισάγει τον αριθμό k στην σωστή θέση του δένδρου, ώστε αυτό να διατηρεί την ιδιότητα του δένδρου δυαδικής αναζήτησης. Υποθέστε ότι, δεδομένου του T, έχετε πρόσβαση στο 'αριστερό' / 'δεξί' παιδί του T (π.χ. μέσω κάποιας συνάρτησης leftchild(T) / rightchild(T)).

   (β) Ποια είναι η πολυπλοκότητα της συνάρτησής σας, όταν εκτελεστεί διαδοχικά για n αριθμούς; Μελετήστε την πολυπλοκότητα χειρότερης περίπτωσης και, προαιρετικά, την πολυπλοκότητα της μέσης περίπτωσης (θεωρώντας κάθε διάταξη των n αριθμών εισόδου ισοπίθανη).

   (γ) Υλοποιήστε την συνάρτησή σας σε γλώσσα προγραμματισμού της επιλογής σας, και συνδυάστε την με κατάλληλη διάσχιση ώστε να φτιάξετε μια συνάρτηση treesort που να δέχεται λίστα ακεραίων και να την επιστρέφει ταξινομημένη.

3. Θεώρημα τεσσάρων χρωμάτων:

   (α) Αποδείξτε ότι υπάρχουν περιπτώσεις χαρτών που δεν μπορούν να χρωματιστούν με 3 χρώματα.

   (β) Αποδείξτε ότι 5 χρώματα αρκούν για χρωματισμό οποιουδήποτε χάρτη.

4. Δίκτυα ταξινόμησης:

   (α) Σχεδιάστε ένα δίκτυο ταξινόμησης οκτώ εισόδων. Τι βάθος και τι μέγεθος έχει το δίκτυό σας; Τι σημαίνει αυτό για τον χρόνο σειριακής και τι για τον χρόνο παράλληλης εκτέλεσης του αντίστοιχου αλγορίθμου ή κυκλώματος;

   (β) Μπορείτε να σχεδιάσετε καλύτερη μέθοδο ταξινόμησης οκτώ αριθμών με συγκρίσεις; Ποιος είναι ο αριθμός συγκρίσεων της μεθόδου σας; Είναι ελάχιστος; Αποδείξτε τους ισχυρισμούς σας.

Βιβλιογραφία

[1] Thomas Cormen, Charles Leiserson, Ronald Rivest and Cliff Stein, “Introduction to Algorithms”, 3rd edition, MIT Press, 2009.

[2] J. Edmonds, “How to Think About Algorithms”, Cambridge University Press, 2008.