Κεφάλαιο 1
Διαδικαστικός και δηλωτικός προγραμματισμός

Σύνοψη

Στο κεφάλαιο αυτό γίνεται συγκριτική παρουσίαση, κυρίως μέσω απλών παραδειγμάτων, του διαδικαστικού και του δηλωτικού προγραμματισμού, δύο μεθοδολογιών για την αντιμετώπιση προβλημάτων στην Επιστήμη των Υπολογιστών.

Με τον διαδικαστικό προγραμματισμό μπορούμε να επιλύσουμε ένα πρόβλημα διατυπώνοντας τα βήματα ενός αλγορίθμου, δηλαδή ακολουθώντας μια διαδικασία που περιγράφει πώς πρέπει να λυθεί το πρόβλημα και να υλοποιηθεί ο αλγόριθμος, με τη βοήθεια μιας κατάλληλης γλώσσας διαδικαστικού προγραμματισμού.

Αντίθετα, στον δηλωτικό προγραμματισμό διατυπώνουμε αξιώματα που περιγράφουν τι ισχύει στον κόσμο του προβλήματος, τα οποία εκφράζουμε με τη βοήθεια μιας κατάλληλης γλώσσας δηλωτικού προγραμματισμού.

Πολλές φορές, η επίλυση ενός προβλήματος με αλγοριθμικό τρόπο οδηγεί σε προγράμματα πολύπλοκα, δυσανάγνωστα και, συνεπώς, δύσκολα συντηρήσιμα. Όταν έχουμε ένα πρόβλημα που ταιριάζει στη δηλωτική φιλοσοφία, δηλαδή διέπεται από απλά αξιώματα, είναι προτιμότερο να επιλέγουμε αυτόν τον τρόπο για την αντιμετώπισή του. Η μόνη επιφύλαξη που μπορούμε, ίσως, να διατυπώσουμε αφορά την απόδοσή του. Σε γενικές γραμμές, τα συστήματα δηλωτικού προγραμματισμού είναι λιγότερο αποδοτικά από αυτά του διαδικαστικού προγραμματισμού, χωρίς κάτι τέτοιο να σημαίνει ότι δεν υπάρχουν αρκετές δηλωτικές γλώσσες πολύ πιο «γρήγορες» από πολλές διαδικαστικές. Έτσι, αν μας ενδιαφέρει η απόδοση του προγράμματος που θα γράψουμε, πρέπει να ζυγίσουμε προσεκτικά όλες τις εμπλεκόμενες παραμέτρους, για να κάνουμε τη σωστή επιλογή.

Προαπαιτούμενη γνώση

Για την κατανόηση του κεφαλαίου, ο αναγνώστης απαιτείται να έχει στοιχειώδεις γνώσεις απλών αλγορίθμων και διατύπωσής τους σε ψευδογλώσσα.

1.1 Αλγόριθμοι και αξιώματα

Στην Επιστήμη των Υπολογιστών, γνωρίζουμε την έννοια του αλγορίθμου [1]. Χρησιμοποιώντας μια σχετικά απλουστευμένη διατύπωση, μπορούμε να δώσουμε τον εξής ορισμό:

> ***Ένας αλγόριθμος περιγράφει, μέσω μιας ακολουθίας στοιχειωδών εκτελέσιμων λειτουργιών, τη μέθοδο επίλυσης ενός συγκεκριμένου προβλήματος.***

Με άλλα λόγια, ένας αλγόριθμος περιγράφει **πώς** λύνεται ένα πρόβλημα, βήμα προς βήμα, έτσι ώστε η επίλυσή του να είναι εύκολα υλοποιήσιμη, με τη βοήθεια μιας γλώσσας προγραμματισμού. Υπάρχουν διάφορες γλώσσες προγραμματισμού που υποστηρίζουν τη διατύπωση αλγορίθμων για την επίλυση προβλημάτων, όπως η Basic, η Fortran, η Pascal, η C, η C++, η Java κ.ά. [2]. Στις γλώσσες αυτές, που ονομάζονται διαδικαστικές, βρίσκουμε εντολές που αντιστοιχούν άμεσα στις στοιχειώδεις λειτουργίες ενός αλγορίθμου, όπως οι εντολές ανάθεσης τιμών σε μεταβλητές μέσω μαθηματικών εκφράσεων, οι εντολές επανάληψης και οι εντολές ελέγχου της ροής μιας διαδικασίας. Με αυτόν τον τρόπο, εισάγουμε την έννοια του **διαδικαστικού προγραμματισμού** ως εξής:

> ***Διαδικαστικός προγραμματισμός ονομάζεται η υλοποίηση ενός αλγορίθμου σε μια διαδικαστική γλώσσα προγραμματισμού.***

Παράδειγμα 1.1

Έστω ότι ενδιαφερόμαστε αρχικά να διατυπώσουμε έναν αλγόριθμο και στη συνέχεια να τον υλοποιήσουμε σε κάποια γλώσσα διαδικαστικού προγραμματισμού, με τον οποίο να μπορούμε να υπολογίσουμε το πλήθος των στοιχείων μιας απλά συνδεδεμένης λίστας, δηλαδή το μήκος της [3]. Ας θυμηθούμε πρώτα ότι μια τέτοια λίστα δεν είναι τίποτε άλλο παρά μια ακολουθία από στοιχεία, με την ταυτότητά της να αντιστοιχεί στη θέση/διεύθυνση του πρώτου της στοιχείου, που λέγεται **κεφαλή** της λίστας. Η κεφαλή «δείχνει», με κάποιον τρόπο,

στο επόμενο στοιχείο της λίστας ή, ισοδύναμα, στη λίστα που αρχίζει από το επόμενο της κεφαλής, η οποία ονομάζεται **ουρά** της αρχικής λίστας. Αυτό ισχύει για κάθε στοιχείο, εκτός φυσικά από το τελευταίο, το οποίο δεν «δείχνει» πουθενά ή, αλλιώς, «δείχνει» στην **κενή λίστα**. Ένας αλγόριθμος υπολογισμού του μήκους μιας λίστας, εκφρασμένος σε μια αυθαίρετη ψευδογλώσσα, θα μπορούσε να είναι ο εξής:

Υπολογισμός μήκους λίστας **list**
Βήμα 1: *Θέστε στο μετρητή* **count** *την τιμή 0*
Βήμα 2: *Ενόσω η λίστα* **list** *δεν είναι κενή*
Βήμα 3: *Αυξήστε τον* **count** *κατά 1*
Βήμα 4: *Κάντε τη* **list** *να δείχνει εκεί όπου δείχνει το πρώτο της στοιχείο, δηλαδή στην ουρά της*
Βήμα 5: *Επιστρέψτε την τιμή του* **count** *ως μήκος της λίστας*

Ο προηγούμενος αλγόριθμος μπορεί πολύ εύκολα να υλοποιηθεί σε μια γλώσσα διαδικαστικού προγραμματισμού. Στη γλώσσα C, για παράδειγμα, αν είχαμε δηλώσει τη δομή:

```
struct listnode {
      int value;
      struct listnode *next;
};
```

θα γράφαμε την εξής συνάρτηση, για τον υπολογισμό του μήκους μιας λίστας:

```
int length(struct listnode *list)
{     int count = 0;
      while (list != NULL)
            {count++;
            list = list->next;
      }
      return count;
}
```

Δεν δίνονται περισσότερες επεξηγήσεις για τη λειτουργία του προηγούμενου προγράμματος, δεδομένου ότι ο αναγνώστης έχει κάποια, στοιχειώδη έστω, εξοικείωση με τη γλώσσα C [4].

□

Αντί, όμως, να περιγράψουμε έναν αλγόριθμο, ο οποίος όταν ακολουθείται βήμα προς βήμα επιλύει ένα πρόβλημα, θα μπορούσαμε, εναλλακτικά, να περιγράψουμε τα αξιώματα που διέπουν το πρόβλημα. Ένας απλός ορισμός για το αξίωμα είναι ο εξής:

Αξίωμα είναι ένας ισχυρισμός του οποίου η αλήθεια ούτε αμφισβητείται, ούτε υπόκειται σε απόδειξη.

Με τη διατύπωση αξιωμάτων, αντί να εξετάσουμε πώς λύνεται ένα πρόβλημα, απλώς δηλώνουμε τι ισχύει στον κόσμο του προβλήματος. Όταν αυτό το κάνουμε με τη βοήθεια κάποιας γλώσσας προγραμματισμού, έχουμε την έννοια του **δηλωτικού προγραμματισμού**, ως εξής:

Δηλωτικός προγραμματισμός ονομάζεται η διατύπωση αξιωμάτων που ισχύουν στον κόσμο ενός προβλήματος σε μια κατάλληλη γλώσσα προγραμματισμού.

Μεταξύ άλλων, η γλώσσα λογικού προγραμματισμού **Prolog** [5] και η γλώσσα συναρτησιακού προγραμματισμού **Haskell** [6] είναι δύο τυπικοί εκπρόσωποι της φιλοσοφίας του δηλωτικού προγραμματισμού.

Στον δηλωτικό προγραμματισμό, όπως μαρτυρά και η ονομασία του, δηλώνουμε κάτι, αντί να περιγράφουμε

μια διαδικασία, όπως συμβαίνει στον διαδικαστικό προγραμματισμό. Βέβαια, δεν είναι δυνατόν να λύσουμε ένα πρόβλημα διατυπώνοντας απλώς κάποια αξιώματα. Πρέπει να φροντίσουμε να επεξεργαστούμε τα δηλωθέντα αξιώματα του προβλήματος, για να δώσουμε την επιθυμητή λύση. Ωστόσο, αυτή η επεξεργασία δεν αφορά τον προγραμματιστή, γιατί γίνεται από το σύστημα που υποστηρίζει τον προγραμματισμό αυτού του είδους, δηλαδή τον δηλωτικό προγραμματισμό. Έτσι, ο προγραμματιστής είναι αφοσιωμένος στη διατύπωση γνώσης, παρά διαδικασιών.

Παράδειγμα 1.2

Αν επιστρέψουμε στο πρόβλημα του υπολογισμού του μήκους μιας απλά συνδεδεμένης λίστας και θυμηθούμε ότι μια μη κενή λίστα έχει κεφαλή (το πρώτο της στοιχείο) και ουρά (τη λίστα που ακολουθεί το πρώτο της στοιχείο), μπορούμε να διατυπώσουμε τα εξής αξιώματα:

Αξιώματα για το μήκος της λίστας
Αξίωμα 1: *Το μήκος της κενής λίστας είναι 0.*
Αξίωμα 2: *Το μήκος μιας μη κενής λίστας είναι ίσο με το μήκος της ουράς της αυξημένο κατά 1.*

Στη γλώσσα λογικού προγραμματισμού Prolog, τα προηγούμενα αξιώματα θα μπορούσαν να διατυπωθούν ως εξής:

```
length([], 0).
length([X|L], N) :- length(L, M), N is M+1.
```

και στη γλώσσα συναρτησιακού προγραμματισμού Haskell ως εξής:

```
length []    = 0
length (a:x) = 1 + length x
```

Φυσικά, θα μπορέσουμε να κατανοήσουμε καλύτερα τα προγράμματα Prolog και Haskell σε επόμενα κεφάλαια, στα οποία δίνονται λεπτομέρειες για τον τρόπο προγραμματισμού στις γλώσσες αυτές.

□

Στο Παράδειγμα 1.2 είναι σαφής η φιλοσοφία του δηλωτικού προγραμματισμού [7]. Πουθενά δεν περιγράψαμε μια διαδικασία για να βρούμε το μήκος μιας λίστας. Αντίθετα, ορίσαμε τι είναι μήκος μιας λίστας, με στόχο, όταν προκύψει κάποια στιγμή ένα πρόβλημα εύρεσης του μήκους συγκεκριμένης λίστας, να μπορέσουμε να το διατυπώσουμε σε ένα σύστημα λογικού ή συναρτησιακού προγραμματισμού, μαζί με τον κατάλληλο ορισμό, και να ζητήσουμε τον υπολογισμό της απάντησης σε αυτό. Όσον αφορά τον τρόπο με τον οποίο διατυπώνουμε προβλήματα σε κάθε περίπτωση, θα πάρουμε μια ιδέα στη συνέχεια, αλλά θα τον μελετήσουμε με περισσότερες λεπτομέρειες σε επόμενα κεφάλαια.

Ας εξετάσουμε τώρα άλλα δύο παραδείγματα, στα οποία θα είναι εμφανέστερη η αντιδιαστολή ανάμεσα στον διαδικαστικό και τον δηλωτικό προγραμματισμό.

Παράδειγμα 1.3

Έστω ότι έχουμε πληροφορίες για διάφορα άτομα και, συγκεκριμένα, για τα ονόματα των γονέων τους. Θα θέλαμε να μάθουμε αν ο παππούς του *Γιώργου* είναι αδελφός του *Νίκου*. Μια πιθανή διαδικαστική προσέγγιση για να λύσουμε αυτό το πρόβλημα είναι ο εξής αλγόριθμος:

Έλεγχος συγκεκριμένης συγγένειας μεταξύ Γιώργου και Νίκου
Βήμα 1: *Θέστε στο* **flag** *την τιμή 0*
Βήμα 2: *Θέστε στο* **x** *τη μητέρα του Γιώργου*
Βήμα 3: *Θέστε στο* **y** *τον πατέρα του* **x**
Βήμα 4: *Θέστε στο* **z** *τον πατέρα του* **y**
Βήμα 5: *Αν ο* **z** *είναι πατέρας του Νίκου, απαντήστε καταφατικά και τερματίστε*
Βήμα 6: *Θέστε στο* **z** *τη μητέρα του* **y**
Βήμα 7: *Αν η* **z** *είναι μητέρα του Νίκου, απαντήστε καταφατικά και τερματίστε*
Βήμα 8: *Αν το* **flag** *είναι 1, απαντήστε αρνητικά και τερματίστε*
Βήμα 9: *Θέστε στο* **flag** *την τιμή 1*
Βήμα 10: *Θέστε στο* **x** *τον πατέρα του Γιώργου*
Βήμα 11: *Πηγαίνετε στο βήμα 3*

Φυσικά, ο αλγόριθμος αυτός θα μπορούσε να διατυπωθεί με πιο γενικό τρόπο, αν στη θέση του *Γιώργου* και του *Νίκου* είχαμε δύο μεταβλητές **A** και **B**, και σε κάθε εκτέλεση του αλγορίθμου τα **A** και **B** ήταν δεδομένα εισόδου. Επί της ουσίας, όμως, δεν θα υπήρχε διαφορά.

□

Από την άλλη πλευρά, θα μπορούσαμε να είχαμε επιλέξει μια δηλωτική αντιμετώπιση του προβλήματος του Παραδείγματος 1.3, όπως φαίνεται στη συνέχεια.

Παράδειγμα 1.4

Ο δηλωτικός τρόπος θεώρησης του προβλήματος, αν ο παππούς του *Γιώργου* είναι αδελφός του *Νίκου*, συνίσταται απλώς στη διατύπωση των αξιωμάτων τα οποία ορίζουν τις συγγένειες που μας ενδιαφέρουν:

Αξιώματα για συγγένειες
Αξίωμα 1: *Παππούς κάποιου είναι ο πατέρας του πατέρα του ή της μητέρας του.*
Αξίωμα 2: *Αδελφός κάποιου είναι ένας άνδρας με τον οποίο έχουν τον ίδιο πατέρα ή την ίδια μητέρα.*

Για άλλη μία φορά, βλέπουμε στο παράδειγμα ότι δεν περιγράφουμε πώς θα λύσουμε το πρόβλημα, αλλά δηλώνουμε τι ισχύει στον φυσικό κόσμο σχετικά με το πρόβλημα.

Αν θέλαμε να διατυπώσουμε τα προηγούμενα αξιώματα σε μια γλώσσα δηλωτικού προγραμματισμού, θα μπορούσαμε, χρησιμοποιώντας, για παράδειγμα, τη γλώσσα Prolog, να γράψουμε:

```
grandfather(X, Z) :- father(X, Y), (father(Y, Z) ; mother(Y, Z)).
brother(X, Y) :- male(X), ((father(Z, X), father(Z, Y)) ;
                           (mother(Z, X), mother(Z, Y))).
```

Υποτίθεται ότι αυτά τα αξιώματα είναι δυνατόν να χρησιμοποιηθούν από ένα σύστημα λογικού προγραμματισμού, εφόσον αυτό είναι εφοδιασμένο και με τη γνώση του κόσμου μας σχετικά με το ποιος είναι πατέρας ποιων (`father`), ποια είναι μητέρα ποιων (`mother`) και ποιοι είναι οι άνδρες (`male`). Αν υπάρχει αυτή η γνώση, τότε η επίλυση του προβλήματος, δηλαδή αν ο παππούς του *Γιώργου* (`george`) είναι αδελφός του *Νίκου* (`nick`), ανάγεται στη διατύπωση της εξής ερώτησης στο σύστημα Prolog:

```
?- grandfather(X, george), brother(X, nick).
```

Με την επεξεργασία των αξιωμάτων που του δόθηκαν, το σύστημα θα απαντήσει καταφατικά, στην περίπτωση που υπάρχει μεταξύ *Γιώργου* και *Νίκου* η συγκεκριμένη συγγένεια, αλλιώς θα απαντήσει αρνητικά.

□

Από τα παραδείγματα που προηγήθηκαν, πήραμε μια ιδέα για τις διαφορές ανάμεσα στη διαδικαστική και στη δηλωτική αντιμετώπιση προβλημάτων. Στον διαδικαστικό προγραμματισμό περιγράφουμε τη διαδικασία επίλυσης ενός προβλήματος. Σε ορισμένες περιπτώσεις, όπως στο Παράδειγμα 1.3, αυτό μπορεί να οδηγήσει στη διατύπωση πολύπλοκων και δυσνόητων αλγορίθμων, με σοβαρές επιπτώσεις στον τρόπο με τον οποίο μπορούμε εύκολα να ελέγξουμε την ορθότητα μεγάλων προγραμμάτων και, φυσικά, να τα συντηρήσουμε. Αντίθετα, με τον δηλωτικό προγραμματισμό, στον οποίο διατυπώνουμε τα αξιώματα του κόσμου μας που απαιτούνται για τη λύση του προβλήματός μας, η κατάσταση είναι απλούστερη. Θυμηθείτε το Παράδειγμα 1.4 και συγκρίνετε την προσπάθειά σας να το κατανοήσετε σε σχέση με αυτήν για το Παράδειγμα 1.3. Πιστεύουμε ότι η διαφορά είναι εμφανής. Έτσι, αν ακολουθήσουμε μια δηλωτική φιλοσοφία προγραμματισμού, η παραγωγικότητά μας θα αυξηθεί, όσον αφορά τόσο το χρόνο που απαιτείται για την αντιμετώπιση ενός προβλήματος, όσο και την ποιότητα της λύσης που επιτυγχάνεται.

Βέβαια, θα αναρωτιέστε εάν μια δηλωτική αντιμετώπιση κάποιου προβλήματος είναι πάντοτε προτιμότερη από μια διαδικαστική. Η απάντηση είναι αρνητική, γιατί, αφενός, δεν είναι βέβαιο ότι για κάθε πρόβλημα υπάρχει μια απλή και κατανοητή περιγραφή των αξιωμάτων που το διέπουν και, αφετέρου, τα συστήματα δηλωτικού προγραμματισμού οδηγούν, εν γένει, σε λιγότερο αποδοτικές υλοποιήσεις από αυτές που βασίζονται σε διαδικαστικές γλώσσες προγραμματισμού. Έτσι, αν είναι πολύ κρίσιμη η απόδοση ενός προγράμματος που θέλουμε να γράψουμε, μάλλον θα πρέπει να επιλέξουμε τη διατύπωση ενός αλγορίθμου σε κάποια «γρήγορη» γλώσσα διαδικαστικού προγραμματισμού αντί της δηλωτικής διατύπωσης αξιωμάτων. Αυτό δεν θα πρέπει να το δεχτούμε ως απαράβατο κανόνα, γιατί σήμερα πολλά συστήματα δηλωτικού προγραμματισμού είναι σε απόδοση σχεδόν εφάμιλλα των διαφόρων γλωσσών διαδικαστικού προγραμματισμού. Όσο η έρευνα στον τομέα του δηλωτικού προγραμματισμού προοδεύει, τόσο περισσότερα πλεονεκτήματα θα παρουσιάζει η πρακτική εφαρμογή του.

Θα τελειώσουμε αυτό το κεφάλαιο, στο οποίο επικεντρωθήκαμε στις διαφορές ανάμεσα στη διαδικαστική και στη δηλωτική αντιμετώπιση προβλημάτων, με μερικές ασκήσεις, με τις οποίες θα μπορέσετε να εμπεδώσετε καλύτερα το περιεχόμενό του. Ακολουθούν οι απαντήσεις στις ασκήσεις και δίνονται κάποια προβλήματα προς λύση.

Ασκήσεις

Άσκηση 1.1

Έστω ότι ενδιαφερόμαστε να έχουμε έναν τρόπο για να μπορούμε να υπολογίζουμε το παραγοντικό ενός μη αρνητικού ακέραιου αριθμού. Για το σκοπό αυτό, θα μπορούσαμε να χρησιμοποιήσουμε κάποια από τις παρακάτω μεθόδους:

1. Το 0! ισούται με 1 και το n! (για $n \neq 0$) ισούται με (n-1)! $\times$ n
2. Το 0! ισούται με 1 και, για να υπολογίσουμε το n! (για $n \neq 0$), πρέπει να πολλαπλασιάσουμε όλους τους ακέραιους αριθμούς από το 1 έως το n

Ποια από τις δύο μεθόδους θα χαρακτηρίζατε διαδικαστική και ποια δηλωτική;

Άσκηση 1.2

Έστω ότι θέλουμε να κατασκευάσουμε μια διαδικασία με την οποία να αντιστρέφεται μια λίστα **list**. Μια (ημιτελής) απάντηση σε αυτό το πρόβλημα θα μπορούσε να ήταν η εξής:

Αντιστροφή λίστας **list**
Βήμα 1: *Κάντε τη* **next** *να δείχνει εκεί όπου δείχνει η* **list**
Βήμα 2: *Θέστε στην* **prev** *την κενή λίστα*
Βήμα 3: *Ενόσω η λίστα* [**X**] *δεν είναι κενή*
Βήμα 4: *Κάντε τη* **list** *να δείχνει εκεί όπου δείχνει η* **next**
Βήμα 5: *Κάνε τη* **next** *να δείχνει εκεί όπου δείχνει το πρώτο της στοιχείο*
Βήμα 6: *Κάντε το πρώτο στοιχείο της* **list** *να δείχνει* [**Y**]
Βήμα 7: *Κάντε την* **prev** *να δείχνει εκεί όπου δείχνει η* **list**
Βήμα 8: *Επιστρέψτε τη* [**Z**] *ως τη ζητούμενη αντεστραμμένη λίστα*

Ποια επιλογή από τις παρακάτω θα πρέπει να συμπληρωθεί στις θέσεις **X**, **Y** και **Z**, για να είναι σωστός ο αλγόριθμος;

X: 1. **list**, 2. **next**, 3. **prev**
Y: 1. στην κενή λίστα, 2. στη **next**, 3. στην **prev**
Z: 1. **list**, 2. **next**, 3. **prev**

Άσκηση 1.3

Εξετάζοντας πάλι το πρόβλημα της αντιστροφής μιας λίστας, θα μπορούσαμε, αντί να περιγράψουμε έναν αλγόριθμο, όπως στην Άσκηση 1.2, να δώσουμε τα αξιώματα που ορίζουν πότε μια λίστα είναι η αντίστροφη μιας άλλης. Μια (ημιτελής) διατύπωση τέτοιων αξιωμάτων είναι η εξής:

Αξιώματα για αντίστροφη λίστας
Αξίωμα 1: *Η αντίστροφη της κενής λίστας είναι* **X** .
Αξίωμα 2: *Η αντίστροφη μιας μη κενής λίστας είναι η λίστα που προκύπτει από την τοποθέτηση* **Y** *στο τέλος* **Z** .

Τι θα πρέπει να συμπληρώσουμε στις θέσεις **X**, **Y** και **Z**, για να είναι τα προηγούμενα αξιώματα σωστά;

Άσκηση 1.4

Δείτε πάλι την Άσκηση 1.1 και διατυπώστε στη μορφή που μάθατε σε αυτό το κεφάλαιο τόσο τα αξιώματα για τον δηλωτικό ορισμό του παραγοντικού, όσο και τα βήματα του αλγορίθμου για τον υπολογισμό του παραγοντικού.

Άσκηση 1.5

Θεωρήστε ότι έχετε στη διάθεσή σας έναν κατευθυνόμενο γράφο χωρίς κύκλους. Θυμηθείτε ότι ένας κατευθυνόμενος γράφος αποτελείται από ένα σύνολο κόμβων και ένα σύνολο ακμών (με κατεύθυνση) μεταξύ κόμβων. Ο γράφος δεν έχει κύκλους, αν δεν υπάρχει περίπτωση να ξεκινήσουμε από κάποιον κόμβο και, ακολουθώντας κατευθυνόμενες ακμές, να επιστρέψουμε στον κόμβο αυτόν. Διατυπώστε τα βήματα ενός αλγορίθμου ο οποίος να ελέγχει αν για δύο κόμβους του γράφου υπάρχει μονοπάτι που να συνδέει τον πρώτο με τον δεύτερο.

Άσκηση 1.6

Δώστε μια δηλωτική προσέγγιση για το πρόβλημα της Άσκησης 1.5, δηλαδή τον έλεγχο της σύνδεσης μεταξύ δύο κόμβων σε έναν κατευθυνόμενο γράφο χωρίς κύκλους.

Απαντήσεις ασκήσεων

Απάντηση άσκησης 1.1

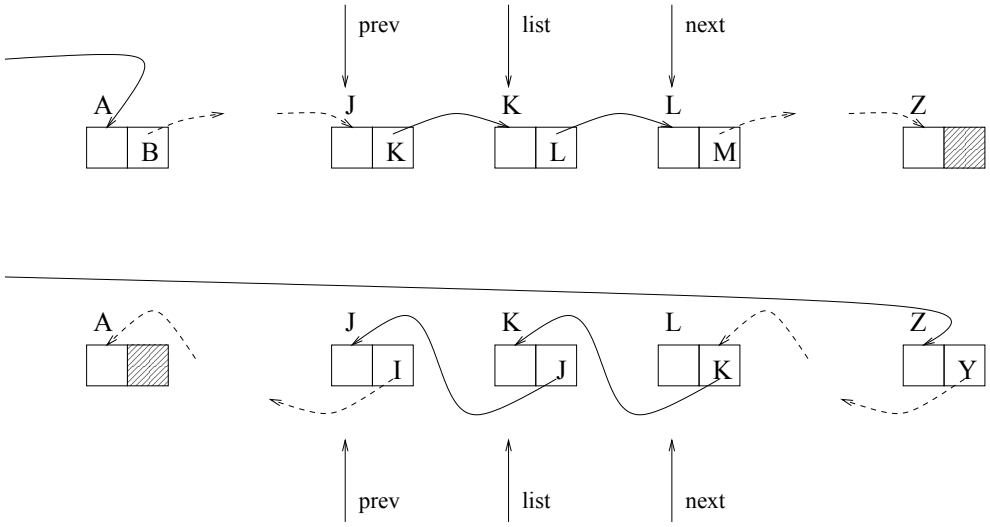
Η μέθοδος 1 είναι δηλωτική, επειδή δίνει τον ορισμό του παραγοντικού, δηλαδή δηλώνει κάτι που ισχύει στα μαθηματικά. Η μέθοδος 2 είναι διαδικαστική, επειδή περιγράφει μια διαδικασία (αλγόριθμο) με την οποία μπορεί κάποιος να υπολογίσει το παραγοντικό ενός αριθμού.

Απάντηση άσκησης 1.2

Οι σωστές απαντήσεις είναι:

X: 2. **next**
Y: 3. στην **prev**
Z: 1. **list**

Στον αλγόριθμο, η λίστα διασχίζεται στοιχείο προς στοιχείο. Σε κάθε επανάληψη κάνουμε το τρέχον στοιχείο να δείχνει στο προηγούμενό του, αντί για το επόμενό του. Βάζουμε το πρώτο στοιχείο της αρχικής λίστας να δείχνει στην κενή λίστα, ενώ το τελευταίο στοιχείο της αρχικής λίστας δεν είναι τίποτε άλλο παρά το πρώτο στοιχείο της νέας αντεστραμμένης λίστας. Σε κάθε επανάληψη του αλγορίθμου, **list** είναι η διεύθυνση του τρέχοντος στοιχείου, **prev** η διεύθυνση του προηγούμενού του και **next** η διεύθυνση του επόμενού του. Πιο παραστατικά, αυτά φαίνονται στο Σχήμα 1.1.



Σχήμα 1.1: *Αντιστροφή λίστας.*

Στη θέση **X** του αλγορίθμου πρέπει να βάλουμε το «**next**», επειδή, όταν το τρέχον στοιχείο δεν έχει επόμενο (δείχνει στην κενή λίστα), τότε έχουμε φτάσει στο τέλος της αρχικής λίστας. Στη θέση **Y** πρέπει να βάλουμε το «στην **prev**», επειδή, για να επιτευχθεί η αντιστροφή, πρέπει το τρέχον στοιχείο να δείξει στο προηγούμενό του. Τέλος, στη θέση **Z** του αλγορίθμου πρέπει να βάλουμε το «**list**», επειδή, όταν έχει τελειώσει η επαναληπτική διαδικασία του αλγορίθμου, η διεύθυνση του τρέχοντος στοιχείου είναι, ουσιαστικά, η διεύθυνση της αντεστραμμένης λίστας.

Απάντηση άσκησης 1.3

Οι σωστές απαντήσεις είναι:

X: η κενή λίστα
Y: της κεφαλής της
Z: της αντίστροφης της ουράς της

Όσον αφορά την απάντηση για το **X**, δεν θα μπορούσε η αντίστροφη μιας κενής λίστας να είναι τίποτε άλλο παρά η κενή λίστα (θυμηθείτε τις λεγόμενες οριακές περιπτώσεις στα μαθηματικά). Όσο για τα **Y** και **Z**, προκειμένου να διαπιστώσετε ότι, με τις παραπάνω αντικαταστάσεις, το δεύτερο αξίωμα ορίζει σωστά την αντίστροφη μιας μη κενής λίστας, δείτε το εξής παράδειγμα: «*Για να βρούμε την αντίστροφη της λίστας 1, 2, 3, 4, αρκεί να πάρουμε την ουρά της (2, 3, 4) και να τοποθετήσουμε στο τέλος της αντίστροφής της (4, 3, 2) την κεφαλή της αρχικής λίστας (1), παίρνοντας έτσι τη λίστα 4, 3, 2, 1.*».

Επειδή, βέβαια, ο δηλωτικός ορισμός για την αντίστροφη μιας λίστας, που δόθηκε στην άσκηση, περιλαμβάνει και την τοποθέτηση ενός στοιχείου στο τέλος μιας λίστας, για λόγους πληρότητας, δίνουμε τα κατάλληλα αξιώματα και γι' αυτό το πρόβλημα.

Αξιώματα για τοποθέτηση στοιχείου στο τέλος λίστας

Αξίωμα 1: *Η λίστα που προκύπτει από την τοποθέτηση ενός στοιχείου στο τέλος της κενής λίστας είναι μια λίστα που περιλαμβάνει μόνο το στοιχείο αυτό.*

Αξίωμα 2: *Η λίστα που προκύπτει από την τοποθέτηση ενός στοιχείου στο τέλος μιας μη κενής λίστας έχει κεφαλή την κεφαλή της αρχικής και ουρά το αποτέλεσμα της τοποθέτησης στο τέλος της ουράς της αρχικής του στοιχείου αυτού.*

Απάντηση άσκησης 1.4

Για τη μέθοδο 1, που είναι δηλωτική, θα μπορούσαμε να γράψουμε τα εξής αξιώματα:

Αξιώματα για ορισμό παραγοντικού

Αξίωμα 1: *Το παραγοντικό του 0 ισούται με 1.*

Αξίωμα 2: *Το παραγοντικό ενός θετικού ακέραιου αριθμού ισούται με το γινόμενο του παραγοντικού του προηγούμενου αριθμού επί τον ίδιο τον αριθμό.*

Για τη διαδικαστική μέθοδο 2, ένας αλγόριθμος είναι ο εξής:

Υπολογισμός παραγοντικού του **n**

Βήμα 1: *Αν το* **n** *ισούται με 0, επιστρέψτε το 1 σαν τιμή του παραγοντικού*

Βήμα 2: *Θέστε στο* **fact** *την τιμή 1*

Βήμα 3: *Για κάθε* **i** *από το 1 έως το* **n** *με βήμα 1*

Βήμα 4: *Θέστε στο* **fact** *το* **fact * i**

Βήμα 5: *Επιστρέψτε το* **fact** *ως τιμή του παραγοντικού*

Απάντηση άσκησης 1.5

Ένας αλγόριθμος που ελέγχει αν υπάρχει μονοπάτι που να συνδέει δύο κόμβους σε έναν κατευθυνόμενο γράφο χωρίς κύκλους είναι ο εξής:

Έλεγχος σύνδεσης κόμβων **A** *και* **B** *σε έναν κατευθυνόμενο γράφο χωρίς κύκλους*

Βήμα 1: *Αρχικοποιήστε μια στοίβα* **stack**

Βήμα 2: *Βάλτε στη* **stack** *τον κόμβο* **A**

Βήμα 3: *Αν η* **stack** *είναι άδεια, απαντήστε αρνητικά και τερματίστε*

Βήμα 4: *Βγάλτε από τη* **stack** *το πρώτο της στοιχείο και θέστε το* **current** *ίσο με αυτό*

Βήμα 5: *Αν το* **current** *είναι το* **B***, απαντήστε καταφατικά και τερματίστε*

Βήμα 6: *Βάλτε στη* **stack** *τους κόμβους που είναι επόμενοι του* **current***, αν υπάρχουν τέτοιοι*

Βήμα 7: *Πηγαίνετε στο βήμα 3*

Στον αλγόριθμο αυτόν χρησιμοποιείται μια στοίβα. Θυμηθείτε ότι σε μια στοίβα εισάγουμε στοιχεία στο άκρο της και εξάγουμε στοιχεία από αυτό, με αποτέλεσμα κάθε φορά να εξάγουμε το πιο πρόσφατα εισαχθέν στοιχείο. Στη στοίβα αυτή, ο αλγόριθμος εισάγει κόμβους του γράφου (αρχίζοντας από τον κόμβο **A**), που πρέπει να εξεταστούν αν οδηγούν στον τελικό κόμβο **B**. Σε κάθε επανάληψη, εξάγεται ένας κόμβος από τη στοίβα και, αν δεν είναι ο **B**, εισάγονται στη στοίβα εκείνοι οι κόμβοι στους οποίους μπορούμε να μεταβούμε από αυτόν.

Απάντηση άσκησης 1.6

Για τη δηλωτική αντιμετώπιση του ελέγχου της σύνδεσης δύο κόμβων σε έναν κατευθυνόμενο γράφο χωρίς κύκλους, μπορούμε να διατυπώσουμε τα εξής αξιώματα:

Αξιώματα για σύνδεση δύο κόμβων σε έναν κατευθυνόμενο γράφο χωρίς κύκλους

Αξίωμα 1: *Ένας κόμβος συνδέεται με κάποιον άλλο, αν υπάρχει απευθείας σύνδεση από τον πρώτο στον δεύτερο.*

Αξίωμα 2: *Ένας κόμβος συνδέεται με κάποιον άλλο, αν υπάρχει ένας ενδιάμεσος κόμβος, τέτοιος ώστε να συνδέεται απευθείας ο πρώτος κόμβος με τον ενδιάμεσο και ο ενδιάμεσος να συνδέεται τελικά με τον δεύτερο.*

Προβλήματα

Πρόβλημα 1.1

Δώστε δηλωτικό ορισμό για το άθροισμα των ακέραιων αριθμών από το 1 μέχρι δεδομένο ακέραιο αριθμό *n*.

Πρόβλημα 1.2

Διατυπώστε τα βήματα αλγορίθμου για τον υπολογισμό του αθροίσματος των ακέραιων αριθμών από το 1 μέχρι δεδομένο ακέραιο αριθμό *n*.

Πρόβλημα 1.3

Διατυπώστε δηλωτικό ορισμό για τη συνένωση δύο λιστών, δηλαδή την προσθήκη στο τέλος της πρώτης λίστας των στοιχείων της δεύτερης κατά σειρά.

Πρόβλημα 1.4

Ζητείται αλγόριθμος για την εύρεση του *n*-οστού στοιχείου μιας λίστας, για δεδομένο θετικό ακέραιο *n*.

Βιβλιογραφικές αναφορές

[1] L. Goldschlager and A. Lister, *Εισαγωγή στη Σύγχρονη Επιστήμη των Υπολογιστών*, Δίαυλος, 1994.

[2] T. Pratt and M. Zelkowitz, *Programming Languages: Design and Implementation*, Prentice Hall, 2000.

[3] J. Martin, *Data Types and Data Structures*, Prentice Hall, 1986.

[4] B. Kernighan and D. Ritchie, *Η Γλώσσα Προγραμματισμού C*, Κλειδάριθμος, 1990.

[5] L. Sterling and E. Shapiro, *The Art of Prolog*, The MIT Press, 1994.

[6] S. Thompson, *Haskell: The Craft of Functional Programming*, Addison Wesley (3rd Edition), 2011.

[7] A. Fischer and F. Grodzinsky, *The Anatomy of Programming Languages*, Prentice Hall, 1993.