Διορθώσεις σελ. 73 - 74

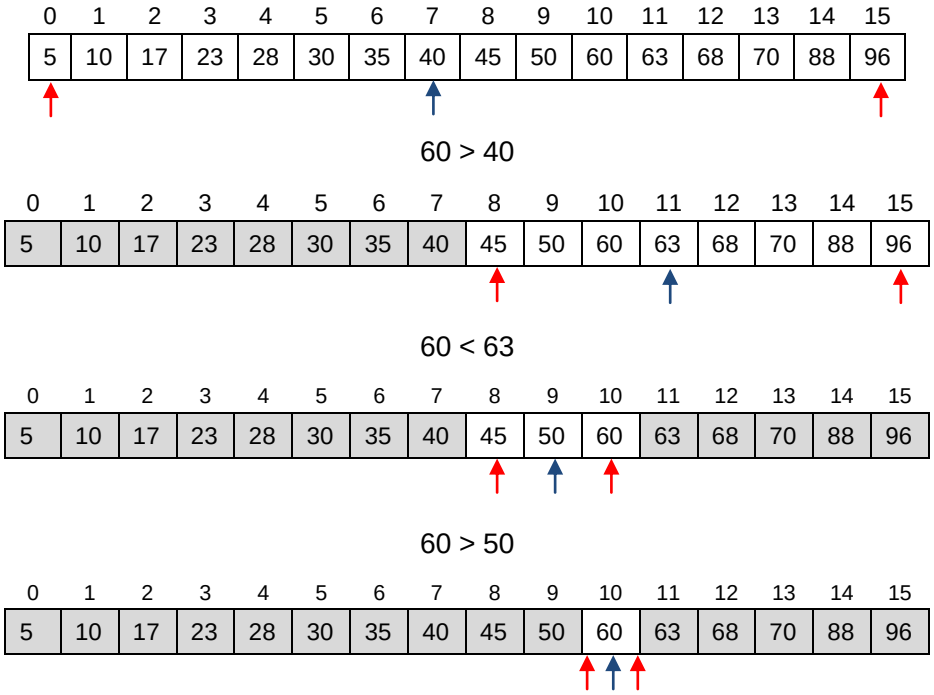
```
# Τώρα ο άνθρωπος σκέφτεται έναν αριθμό από 1 έως 1000
N = 1000
print “Σκέψου έναν αριθμό από το 1 έως το  ”, N

guesses = 0
found = False
first = 1
last = N
while  not  found  and guesses < 10  :
   mid = ( first + last ) / 2
   answer = raw_input(“Είναι ο αριθμός ο   ” + str(mid) + “ ? (N/O)” )
   guesses = guesses + 1
   if   answer == “N”  :
     found = True
   else :
      answer = raw_input(“Είναι μικρότερος του ” + str(mid) + “ ? (N/O)” )
      if answer ==“N”  :
        last = mid – 1
      else :
        first = mid + 1
  if   found == True  :
     print “Κέρδισα!!! Το βρήκα με ”, guesses, “ προσπάθειες”
 else :
     print “Κέρδισες”
```

Στο παραπάνω πρόγραμμα στις μεταβλητές first και last καταχωρούμε τις θέσεις των δυο άκρων του διαστήματος και στην mid το μέσο του διαστήματος.

Δυαδική αναζήτηση σε λίστα

Προηγουμένως χρησιμοποιήσαμε τον αλγόριθμο της δυαδικής αναζήτησης για να βρούμε έναν μυστικό αριθμό μέσα σε κάποια όρια. Ο αλγόριθμος βασίζεται στο γεγονός ότι οι αριθμοί είναι διατεταγμένοι κατά αύξουσα σειρά. Για να εφαρμοστεί ο ίδιος αλγόριθμος και σε άλλα δεδομένα, όπως για παράδειγμα σε ονόματα, θα πρέπει αυτά να είναι διατεταγμένα επίσης σε κάποιου είδους σειρά, όπως σε αλφαβητική. Για παράδειγμα με χρήση της δυαδικής αναζήτησης, μπορούμε να βρούμε ένα όνομα στον τηλεφωνικό κατάλογο, όπου όλα τα ονόματα είναι σε αλφαβητική σειρά. Έτσι μπορούμε να εφαρμόσουμε τη δυαδική αναζήτηση στα στοιχεία μιας λίστας τα οποία βρίσκονται σε κάποια λογική διάταξη, είτε αυτά είναι αριθμοί είτε αλφαριθμητικά.

Παρακάτω φαίνεται η εκτέλεση του αλγόριθμου δυαδικής αναζήτησης σε μια λίστα 16 αριθμών, για την αναζήτηση του αριθμού 60. Η λογική εδώ είναι ακριβώς ίδια με το παιχνίδι “Βρες τον αριθμό”.



Διόρθωση σελ. 79

Οι οριακές τιμές της εσωτερικής επανάληψης από (N–1, 1, –1) να γίνουν (N–1, 0, –1)

```
N = len( array )
for i in range(1 , N, 1):
    for j in range(N–1 , 0, –1):
        if  array[ j ] < array[ j–1 ]  :
            array [ j ] , array [ j–1 ] = array[ j–1 ] , array[ j ]
```

Διόρθωση σελ. 84

Αλγόριθμος Ταξινόμησης με Εισαγωγή (Insertion Sort)

```
def insertionSort( array ) :
    for i in range(1, len( array ) ) :
        value = array[ i ]
        pos = i
        # αναζήτηση
        while pos > 0 and array[ pos–1 ] > value :
            pos = pos – 1
        # μετακίνηση των στοιχείων μια θέση δεξιά
        for j in range(i–1 , pos–1, –1) :
            array[ j+1 ] = array[ j ]
        # το στοιχείο τοποθετείται στη θέση pos.
        array[ pos ] = value
```

Διορθώσεις σελ. 143 - 145

Η δομή της στοίβας μπορεί να υλοποιηθεί στην Python με μια λίστα στην οποία οι εισαγωγές και οι εξαγωγές στοιχείων γίνονται μόνο από το ένα άκρο. Παρακάτω δίνονται δυο υλοποιήσεις της στοίβας. Στην πρώτη περίπτωση, οι εισαγωγές/διαγραφές στοιχείων γίνονται στο πίσω μέρος της λίστας, ενώ στη δεύτερη, από το εμπρός μέρος.

Υλοποίηση Στοίβας σε Python με δύο τρόπους

```
def push(stack, item) :
    stack.append( item )
def pop(stack) :
    return stack.pop( )
def isEmpty(stack) :
    return len(stack) == 0
def createStack( ) :
    return [ ]
```

```
def push(stack, item) :
    stack.insert(0, item)
def pop(stack) :
    return stack.pop( 0 )
def isEmpty(stack) :
    return len(stack) == 0
def createStack( ) :
    return [ ]
```

Οι δύο υλοποιήσεις που δίνονται παραπάνω, διαφέρουν μόνο ως προς το σημείο της λίστας στο οποίο γίνονται οι εισαγωγές/διαγραφές των στοιχείων.

Εφαρμογή Στοίβας : Αντιστροφή αριθμών

Το παρακάτω πρόγραμμα δέχεται από τον χρήστη αριθμούς μέχρι να δοθεί το 0 και τους εμφανίζει σε αντίστροφη σειρά από αυτή με την οποία δόθηκαν. Θέλουμε κάθε φορά να εμφανίσουμε πρώτο τον αριθμό που δόθηκε τελευταίος. Χρειαζόμαστε μια δομή δεδομένων που να υποστηρίζει τη λειτουργία LIFO, όπως η στοίβα.

```
stack = createStack()        # Δημιουργία της στοίβας stack
number = int( raw_input( ) )   # Διάβασε τον πρώτο αριθμό
while number != 0 :                  # Όσο δεν δίνεται 0
    push(stack, number)    # Σπρώξε τον αριθμό στη στοίβα
    number = int( raw_input( ) )  # Διάβασε τον επόμενο αριθμό

while not  isEmpty( stack ) :        # Μέχρι να αδειάσει η στοίβα
    number = pop(stack)          # βγάλε τον επόμενο αριθμό
    print number                 # και εκτύπωσέ τον
```

Παρατηρήστε ότι στο παραπάνω πρόγραμμα δεν φαίνεται ποια υλοποίηση χρησιμοποιείται. Αν τροποποιήσουμε την υλοποίηση της ώθησης, δε θα χρειαστεί να αλλάξουμε τίποτα στο πρόγραμμα. Αυτό είναι γνωστό ως διαχωρισμός **διεπαφής (διασύνδεσης) – υλοποίησης (interface – implementation)**, αφού οι εφαρμογές που χρησιμοποιούν δομές δεδομένων όπως η στοίβα, είναι απολύτως ανεξάρτητες από την υλοποίηση.

Ουσιαστικά δε μας ενδιαφέρει πώς έχει υλοποιηθεί η στοίβα, αφού εμείς θέλουμε μόνο να χρησιμοποιήσουμε τις συναρτήσεις που έχουμε ορίσει παραπάνω.

Το σύνολο των επικεφαλίδων των συναρτήσεων το οποίο είναι διαθέσιμο στον προγραμματιστή που χρησιμοποιεί τη στοίβα το ονομάζουμε διεπαφή ή διασύνδεση (interface) της δομής αυτής. Η διεπαφή μιας δομής ορίζει τι μπορεί να κάνει η δομή και όχι τον τρόπο με τον οποίο το κάνει. Το τελευταίο είναι ο ρόλος της υλοποίησης.

Διεπαφή της Δομής Δεδομένων Στοίβα

```
def  push(stack, item)
def  pop(stack)
def  isEmpty(stack)
def  createStack( )
```

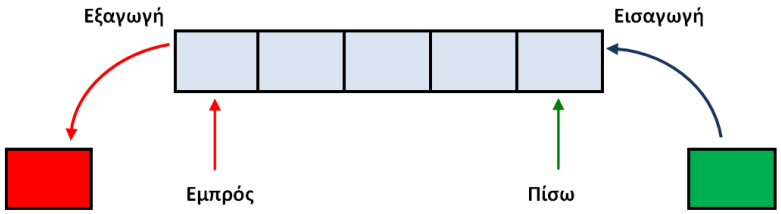
1.1 Ουρά

Μια δομή δεδομένων που χρησιμοποιείται για την μοντελοποίηση και προσομοίωση πραγματικών φαινομένων, είναι η δομή της ουράς. Τα φαινόμενα που μοντελοποιούνται αναφέρονται στην εξυπηρέτηση ανθρώπων, αντικειμένων ή προγραμμάτων. Τέτοια παραδείγματα ουρών είναι:

- Οι ουρές στις τράπεζες και τα σούπερ-μάρκετ.
- Η ουρά των προγραμμάτων που περιμένουν να εξυπηρετηθούν από τον επεξεργαστή του υπολογιστή σας.
- Η ουρά των αιτήσεων προς το διακομιστή ιστού (web server) ενός δικτυακού τόπου.

Τα φαινόμενα αυτά μελετώνται από διάφορους κλάδους των Μαθηματικών και της Πληροφορικής, όπως είναι η θεωρία ουρών (Queueing theory) και η Επιχειρησιακή έρευνα (Operations research).

Σε αντίθεση με τη στοίβα, που η λειτουργία της χαρακτηρίζεται ως LIFO (**L**ast **I**n **F**irst **O**ut), η λειτουργία της ουράς είναι γνωστή στη βιβλιογραφία ως FIFO (First In First Out), αφού το κάθε στοιχείο της ουράς εξυπηρετείται με τη σειρά που έφτασε στην ουρά.



Εικόνα 8-2

Δύο είναι οι βασικές λειτουργίες μιας ουράς:

- Εισαγωγή στοιχείου, η οποία γίνεται στο πίσω μέρος της ουράς.
- Εξαγωγή στοιχείου, η οποία γίνεται από το εμπρός μέρος της ουράς.

Υλοποίηση Ουράς σε Python

```
def  enqueue(queue, item) :
    queue = queue.append( item )
def  dequeue(queue) :
    return queue.pop( 0 )
def  isEmpty(queue) :
    return len(queue) == 0
def  createQueue( ) :
    return [ ]
```

Διόρθωση σελ. 139

Η επανάληψη να γίνει μέχρι 101 και όχι μέχρι 100, ώστε να συμπεριλαμβάνει και την τιμή 100

```
>>> sum = 0
>>> for i in range(101) :
        sum = sum + i
>>> print sum
5050
```

```
>>> for index in range(2,11,2) :
        print index ,
2  4  6 8 10
```