ΣΗΜΕΙΩΣΕΙΣ ΣΤΟΥΣ ΠΙΝΑΚΕΣ

ΤΑΞΙΝΟΜΗΣΗ

Να δώσετε τον ορισμό της ταξινόμησης

Δοθέντων των στοιχείων $\alpha_1,\alpha_2,\ldots\alpha_N$, η ταξινόμηση συνίσταται στη μετάθεση (permutation) της θέσης των στοιχείων, ώστε να τοποθετηθούν σε μία σειρά $\alpha_{K1},\alpha_{K2}\ldots,\alpha_{KN}$ έτσι ώστε δοθείσης μιας συνάρτησης διάταξης f να ισχύει

$f(\alpha_{K1})\leq f(\alpha_{K2})\leq\ldots\leq f(\alpha_{KN})$ - ταξινόμηση σε αύξουσα τάξη ή

$f(\alpha_{K1})\geq f(\alpha_{K2})\geq\ldots\geq f(\alpha_{KN})$ - ταξινόμηση σε φθίνουσα τάξη

Επεξήγηση του αλγορίθμου με τη μέθοδο της Ευθείας Ανταλλαγής (Bubble Sort)

Με τη μέθοδο αυτή επεξεργαζόμαστε τα στοιχεία του πίνακα (φυσαλίδες) σε διαδοχικά περάσματα και τα ανασυντάσσουμε με τέτοιο τρόπο, ώστε στο τέλος ο πίνακας που θα προκύψει να είναι ταξινομημένος. Σ' ένα πρώτο πέρασμα η πιο ελαφριά φυσαλίδα να είναι πρώτη και οι υπόλοιπες αταξινόμητες, στο δεύτερο πέρασμα η δεύτερη πιο ελαφριά φυσαλίδα να έρθει σα δεύτερη γιατί δεν μπορεί να ξεπεράσει την πρώτη που είναι η πιο ελαφριά κ.ο.κ. Τελικά σε έναν πίνακα με Ν στοιχεία απαιτούνται Ν-1 περάσματα

Όταν χρησιμοποιούνται δυο δομές επανάληψης **Για** την επεξεργασία ενός πίνακα δεν σημαίνει απαραίτητα πως ο πίνακας αυτός είναι δισδιάστατος. Το είδος του πίνακα καθορίζεται αν θέλεις από το πλήθος των δεικτών που χρησιμοποιούνται εντός των αγκύλων !!

Κατ' αρχάς να ξεκαθαρίσουμε ότι ταξινόμηση μπορεί να γίνει μόνο σε μονοδιάστατο πίνακα (ή σε κάποια στήλη/γραμμή δισδιάστατου που όμως αποτελεί μονοδιάστατο πίνακα) !!

Στον αλγόριθμο ταξινόμησης της φυσαλίδας χρησιμοποιούνται δυο δομές επανάληψης Για (δυο δείκτες) με το εξής σκεπτικό:

- Ο δείκτης i σαρώνει τον πίνακα από πάνω προς τα κάτω και ο j από κάτω προς τα πάνω.
- Ο δείκτης j εξασφαλίζει την ανά δυο σύγκριση των στοιχείων του πίνακα με στόχο το μικρότερο σε κάθε σύγκριση στοιχείο ανεβαίνει προς τα πάνω (όπως η φυσαλίδα στο νερό)
- Ο δείκτης i δείχνει μέχρι πιο σημείο του πίνακα έχει ταξινομηθεί

	i	j	αύξουσα	Π(1)	π(2)	Π(3)	π(4)	Π(5)
			>	8	3	6	4	1
1	2	5	π(4) π(5)	8	3	6	1	4
2	2	4	π(3) π(4)	8	3	1	6	4
3	2	3	π(2) π(3)	8	1	3	6	4
4	2	2	π(1) π(2)	1	8	3	6	4
5	3	5	π(4) π(5)	1	8	3	4	6
6	3	4	π(3) π(4)	1	8	3	4	6
7	3	3	π(2) π(3)	1	3	8	4	6
8	4	5	π(4) π(5)	1	3	8	4	6
9	4	4	π(3) π(4)	1	3	4	8	6
10	5	5	π(4) π(5)	1	3	4	6	8
				1	3	4	6	8

```
ΠΡΟΓΡΑΜΜΑ taxinomisi
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: i, j, A[5], temp, k
ΑΡΧΗ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    ΔΙΑΒΑΣΕ A[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ 'TAXINOMISI'
  ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ 5
    ΓΙΑ j ΑΠΟ 5 ΜΕΧΡΙ i ΜΕ ΒΗΜΑ -1
      ΑΝ A[j-1] > A[j] ΤΟΤΕ
        temp <- A[j - 1]
        A[j - 1] <- A[j]
        A[j] <- temp
      ΤΕΛΟΣ_ΑΝ
      ΓΡΑΨΕ 'j=', j
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'ΒΗΜΑ ', i
    ΓΙΑ k ΑΠΟ 1 ΜΕΧΡΙ 5
      ΓΡΑΨΕ A[k]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

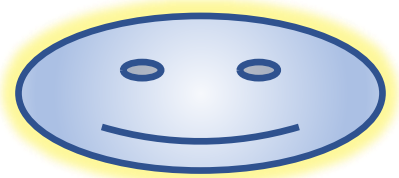
Επεξήγηση του αλγορίθμου με τη μέθοδο ταξινόμησης της έξυπνης φυσαλίδας και να περιγράψετε που επιτυγχάνεται η συγκεκριμένη βελτίωση

Πριν την έναρξη του εσωτερικού βρόχου ορίζουμε μια λογική μεταβλητή και της δίνουμε την τιμή **Αληθής**.

Αν κατά την εκτέλεση του εσωτερικού βρόχου συμβεί μια αντιμετάθεση στοιχείων του πίνακα, τότε η λογική μεταβλητή παίρνει την τιμή **Ψευδής**, αν παραμείνει **Αληθής** σημαίνει ότι δε χρειάστηκε να γίνει αντιμετάθεση άρα ο πίνακας είναι ήδη ταξινομημένος.

Η εξωτερική **Για** της απλής φυσαλίδας, αντικαθίσταται από μια **Οσο** , η οποία έχει μια σύνθετη συνθήκη η οποία εκτός του ότι ελέγχει αν έχουν εκτελεστεί όλες οι επαναλήψεις ελέγχει και την τιμή της λογικής μεταβλητής.

Όταν αυτή διατηρήσει την τιμή **Αληθής** αυτό σημαίνει ότι καμία αντιμετάθεση στοιχείων δεν έγινε συνεπώς ο πίνακας έχει ταξινομηθεί άρα σταματά η ταξινόμηση



```
ΠΡΟΓΡΑΜΜΑ taxinomisiBest
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Μ[5], κ, λ, προσ, μετρ, i
  ΛΟΓΙΚΕΣ: Ταξ
ΑΡΧΗ
  μετρ <- 0
  Μ[1] <- 1
  Μ[2] <- 3
  Μ[3] <- 8
  Μ[4] <- 4
  Μ[5] <- 6
  Ταξ <- ΨΕΥΔΗΣ
  κ <- 2
  ΟΣΟ κ <= 5 ΚΑΙ ΟΧΙ Ταξ ΕΠΑΝΑΛΑΒΕ
    Ταξ <- ΑΛΗΘΗΣ
    ΓΙΑ λ ΑΠΟ 5 ΜΕΧΡΙ κ ΜΕ_ΒΗΜΑ -1
      ΑΝ Μ[λ - 1] > Μ[λ] ΤΟΤΕ
        προσ <- Μ[λ]
        Μ[λ] <- Μ[λ - 1]
        Μ[λ - 1] <- προσ
        Ταξ <- ΨΕΥΔΗΣ
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    κ <- κ + 1
    μετρ <- μετρ + 1
    ΓΡΑΨΕ "_____"
    ΓΡΑΨΕ "μετρητής:", μετρ
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
      ΓΡΑΨΕ Μ[i]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ Ταξ
  ΓΡΑΨΕ "μετρητής:", μετρ
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 5
    ΓΡΑΨΕ Μ[κ]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
1
2 μετρητής:1
3 1
4 3
5 4
6 8
7 6
8
9 μετρητής:2
10 1
11 3
12 4
13 6
14 8
15
16 μετρητής:3
17 1
18 3
19 4
20 6
21 8
22 ΑΛΗΘΗΣ
23 μετρητής:3
24 1
25 3
26 4
27 6
28 8
29
```

ΠΟΛΥ ΣΗΜΑΝΤΙΚΟ ΘΕΜΑ – Παράλληλη σε συνδυασμό με Διπλή Ταξινόμηση

Να γραφεί πρόγραμμα το οποίο δέχεται από το πληκτρολόγιο για κάθε ομάδα ποδοσφαίρου, το όνομα, τις νίκες, τις ισοπαλίες και τις ήττες της. Ο αριθμός των ομάδων είναι 16 και οι αγώνες του πρωταθλήματος είναι 30. Για αυτό πρέπει να γίνεται έλεγχος επικύρωσης δεδομένων. Τα δεδομένα καταχωρούνται σε πίνακες που είναι αρχικά κενές.

Στον πίνακα POmades τα ονόματα και στους πίνακες PNikes, PHttes, PIsop για τον αριθμό νικών, ηττών και ισοπαλιών αντίστοιχα. Επίσης θα δημιουργείται ένας πίνακας PBat όπου θα καταχωρούνται οι βαθμοί για κάθε ομάδα, αν είναι γνωστό ότι η νίκη έχει 3 βαθμούς, η ισοπαλία 1 βαθμό και η ήττα 0 (μηδέν) βαθμούς.

Το πρόγραμμα χρησιμοποιεί τον αλγόριθμο ταξινόμησης της **ευθείας ανταλλαγής** (φυσαλίδα-bubble sort) και εμφανίζει όλα τα παραπάνω στοιχεία των ομάδων.

Συγκεκριμένα σε κάθε γραμμή εμφανίζει τα στοιχεία μιας ομάδας.

Στην πρώτη γραμμή θα είναι τα στοιχεία της ομάδας με την μεγαλύτερη βαθμολογία και στην τελευταία τα στοιχεία της ομάδας με την μικρότερη βαθμολογία.

Όταν δυο ή περισσότερες ομάδες έχουν τους ίδιους βαθμούς πρώτα θα γράφονται τα στοιχεία της ομάδας με τις περισσότερες νίκες και στις επόμενες γραμμές τα στοιχεία των υπολοίπων ομάδων **της ίδιας βαθμολογίας**.

(Βοηθητική σημείωση: Κατά την ταξινόμηση πρέπει να γίνονται αντιμεταθέσεις στις λίστες που πρέπει, έτσι ώστε να διατηρείται η αντιστοίχιση της κάθε βαθμολογίας με το όνομα της ομάδας, με τις νίκες, τις ήττες και τις ισοπαλίες της.)

Κατόπιν το πρόγραμμα θα ταξινομεί κατά αύξουσα αλφαβητική σειρά τον πίνακα POmades κάνοντας ταυτόχρονα κατάλληλες αντιματαθέσεις και στον πίνακα PBat μόνο. (Από το σημείο αυτό μας ενδιαφέρει μόνο η αντιστοίχιση ονομάτων, βαθμολογίας).

Το πρόγραμμα θα δέχεται ένα όνομα από το πληκτρολόγιο για αναζήτηση στον πίνακα POmades. Η αναζήτηση θα γίνεται με τον αλγόριθμο της δυαδικής αναζήτησης (με χρήση κατάλληλης συνάρτησης).

Το πρόγραμμα εκμεταλλεύεται ότι ο πίνακας POmades **είναι ταξινομημένος, που είναι προϋπόθεση ώστε να λειτουργήσει ο αλγόριθμος δυαδικής αναζήτησης.**

Η συνάρτηση αυτή θα επιστρέφει στο κυρίως πρόγραμμα τον αριθμό της θέσης στην οποία βρέθηκε το όνομα της ομάδας μέσα στον πίνακα των ονομάτων, **αν αυτό το όνομα που δόθηκε υπάρχει στον πίνακα**. Ενώ αν δεν υπάρχει θα επιστρέφει την τιμή -1.

Το κυρίως πρόγραμμα εκμεταλλευόμενο την επιστρεφόμενη τιμή της συνάρτησης θα εμφανίζει το όνομα της και την βαθμολογία της ομάδας (που δόθηκε για αναζήτηση) και στην περίπτωση που δεν υπάρχει αυτή η ομάδα θα πρέπει να εμφανίζει: Η ομάδα αυτή δεν υπάρχει.

```
ΠΡΟΓΡΑΜΜΑ foodball
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Pnikes[16], Pittes[16], Pisop[16], i, j, ttl_agon, Pbath[16], temp
  ΧΑΡΑΚΤΗΡΕΣ: Pomada[16], tempx
ΑΡΧΗ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 16
     !-------data import--------
    ttl_agon <- 0
    ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
      ΓΡΑΨΕ 'Δώσε όνομα ομάδας:'
      ΔΙΑΒΑΣΕ Pomada[i]
      ΓΡΑΨΕ 'Δώσε νίκες ομάδας:'
      ΔΙΑΒΑΣΕ Pnikes[i]
      ΓΡΑΨΕ 'Δώσε ήττες ομάδας:'
      ΔΙΑΒΑΣΕ Pittes[i]
      ΓΡΑΨΕ 'Δώσε ισοπαλίες ομάδας:'
      ΔΙΑΒΑΣΕ Pisop[i]
      ttl_agon <- Pnikes[i] + Pittes[i] + Pisop[i]
    ΜΕΧΡΙΣ_ΟΤΟΥ ttl_agon = 30
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 16
      !-------Grades calc----------
    Pbath[i] <- Pnikes[i]*5 + Pisop[i]*2 - Pittes[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ 16
    ΓΙΑ j ΑΠΟ 16 ΜΕΧΡΙ i ΜΕ_ΒΗΜΑ -1
      ΑΝ Pbath[j] < Pbath[j - 1] ΤΟΤΕ
      !--------Multi Table Sort Parallel--------
        temp <- Pbath[j - 1]
        Pbath[j - 1] <- Pbath[j]
        Pbath[j] <- temp

        temp <- Pnikes[j - 1]
        Pnikes[j - 1] <- Pnikes[j]
        Pnikes[j] <- temp

        temp <- Pittes[j - 1]
        Pittes[j - 1] <- Pittes[j]
        Pittes[j] <- temp

        temp <- Pisop[j - 1]
        Pisop[j - 1] <- Pisop[j]
        Pisop[j] <- temp

        tempx <- Pomada[j - 1]
        Pomada[j - 1] <- Pomada[j]
        Pomada[j] <- tempx
```

```
        ΑΛΛΙΩΣ_ΑΝ Pbath[j] = Pbath[j - 1] ΚΑΙ Pnikes[j] > Pnikes[j - 1] ΤΟΤΕ
          !------Second condition Sort
          temp <- Pnikes[j - 1]
          Pnikes[j - 1] <- Pnikes[j]
          Pnikes[j] <- temp

          temp <- Pittes[j - 1]
          Pittes[j - 1] <- Pittes[j]
          Pittes[j] <- temp

          temp <- Pisop[j - 1]
          Pisop[j - 1] <- Pisop[j]
          Pisop[j] <- temp

          tempx <- Pomada[j - 1]
          Pomada[j - 1] <- Pomada[j]
          Pomada[j] <- tempx
        ΤΕΛΟΣ_ΑΝ
      ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 16
      ΓΡΑΨΕ Pbath[i], '-', Pomada[i], '-', Pnikes[i], '-', Pittes[i], '-', Pisop[i]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Οθόνη εκτέλεσης

```
Δώσε όνομα ομάδας:
a
Δώσε νίκες ομάδας:
11
Δώσε ήττες ομάδας:
11
Δώσε ισοπαλίες ομάδας:
8
Δώσε όνομα ομάδας:
b
Δώσε νίκες ομάδας:
12
Δώσε ήττες ομάδας:
10
Δώσε ισοπαλίες ομάδας:
8
Δώσε όνομα ομάδας:
c
Δώσε νίκες ομάδας:
13
Δώσε ήττες ομάδας:
13
Δώσε ισοπαλίες ομάδας:
4
Δώσε όνομα ομάδας:
d
Δώσε νίκες ομάδας:
14
Δώσε ήττες ομάδας:
14
Δώσε ισοπαλίες ομάδας:
2
Δώσε όνομα ομάδας:
e
Δώσε νίκες ομάδας:
8
Δώσε ήττες ομάδας:
12
Δώσε ισοπαλίες ομάδας:
10
Δώσε όνομα ομάδας:
f
Δώσε νίκες ομάδας:
7
Δώσε ήττες ομάδας:
11
Δώσε ισοπαλίες ομάδας:
12
Δώσε όνομα ομάδας:
g
Δώσε νίκες ομάδας:
15
Δώσε ήττες ομάδας:
5
Δώσε ισοπαλίες ομάδας:
10
Δώσε όνομα ομάδας:
h
Δώσε νίκες ομάδας:
16
Δώσε ήττες ομάδας:
11
Δώσε ισοπαλίες ομάδας:
3
Δώσε όνομα ομάδας:
i
Δώσε νίκες ομάδας:
14
Δώσε ήττες ομάδας:
6
Δώσε ισοπαλίες ομάδας:
10
Δώσε όνομα ομάδας:
j
Δώσε νίκες ομάδας:
19
Δώσε ήττες ομάδας:
3
Δώσε ισοπαλίες ομάδας:
8
Δώσε όνομα ομάδας:
k
Δώσε νίκες ομάδας:
20
Δώσε ήττες ομάδας:
5
Δώσε ισοπαλίες ομάδας:
5
Δώσε όνομα ομάδας:
l
Δώσε νίκες ομάδας:
14
Δώσε ήττες ομάδας:
6
Δώσε ισοπαλίες ομάδας:
10
Δώσε όνομα ομάδας:
m
Δώσε νίκες ομάδας:
14
Δώσε ήττες ομάδας:
7
Δώσε ισοπαλίες ομάδας:
9
Δώσε όνομα ομάδας:
n
Δώσε νίκες ομάδας:
17
Δώσε ήττες ομάδας:
3
Δώσε ισοπαλίες ομάδας:
10
Δώσε όνομα ομάδας:
o
Δώσε νίκες ομάδας:
5
Δώσε ήττες ομάδας:
15
Δώσε ισοπαλίες ομάδας:
10
Δώσε όνομα ομάδας:
p
Δώσε νίκες ομάδας:
7
Δώσε ήττες ομάδας:
13
Δώσε ισοπαλίες ομάδας:
10
30-o-5-15-10
42-p-7-13-10
48-e-8-12-10
48-f-7-11-12
60-d-14-14-2
60-c-13-13-4
60-a-11-11-8
66-b-12-10-8
75-h-16-11-3
81-m-14-7-9
84-i-14-6-10
84-l-14-6-10
90-g-15-5-10
102-n-17-3-10
105-k-20-5-5
108-j-19-3-8
```

Ταξινόμηση δισδιάστατου πίνακα

Ας υποθέσουμε ότι μας δίνεται ένας πίνακας δύο διαστάσεων 3 γραμμών και 4 στηλών όπως ο επόμενος:

5	6	8	17
-7	22	15	3
18	1	21	9

Ζητούμενο είναι η δημιουργία αλγορίθμου που θα ταξινομεί τα στοιχεία του, σε αύξουσα σειρά ώστε το μικρότερο να βρίσκεται στην θέση 1,1 και το μεγαλύτερο στην θέση 3,4. Δηλαδή τα στοιχεία να διατάσσονται γραμμή-προς-γραμμή από το μικρότερο στο μεγαλύτερο. Ο επόμενος πίνακας δείχνει την διάταξη των στοιχείων.

-7	1	3	5
6	8	9	15
17	18	21	22

Δημιουργία ενός μονοδιάστατου πίνακα με όλα τα στοιχεία του δισδιάστατου, ταξινόμηση του μονοδιάστατου και στην συνέχεια αντιγραφή των στοιχείων από τον μονοδιάστατο στον δισδιάστατο.

```
Αλγόριθμος ταξινόμηση_διδιάστατου

   Δεδομένα // A //
   ! αντιγραφή των στοιχείων σε πίνακα μιας διάστασης (Β)
   k <- 1
   Για i από 1 μέχρι 3
      Για j από 1 μέχρι 4
         B[k] <- A[i,j]
         k <- k + 1
      Τέλος_επανάληψης
   Τέλος_επανάληψης

   ! ταξινόμηση του Β
   Για i από 2 μέχρι 12
      Για j από 12 μέχρι i με_βήμα -1
         Αν B[j] < B[j-1] τότε
            αντιμετάθεσε B[j], B[j-1]
         Τέλος_αν
      Τέλος_επανάληψης
   Τέλος_επανάληψης

   ! επανατοποθέτηση των στοιχείων στον πίνακα Α
   k <- 1
   Για i από 1 μέχρι 3
      Για j από 1 μέχρι 4
         A[i,j] <- B[k]
         k <- k + 1
      Τέλος_επανάληψης
   Τέλος_επανάληψης

Τέλος ταξινόμηση_διδιάστατου
```

Ολίσθηση

Ολίσθηση των στοιχείων κατά μία θέση προς τα δεξιά και τοποθέτηση του τελευταίου στοιχείου πρώτου

```
ΠΡΟΓΡΑΜΜΑ olisthisi
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: i, p[5], a
ΑΡΧΗ
  p[1] <- 18
  p[2] <- 20
  p[3] <- 14
  p[4] <- 11
  p[5] <- 15
!Ολίσθηση των στοιχείων κατά μία θέση προς τα δεξιά
!και τοποθέτηση του τελευταίου στοιχείου πρώτο
  a <- p[5]
  ΓΙΑ i ΑΠΟ 5 ΜΕΧΡΙ 2 ΜΕ_ΒΗΜΑ -1
    p[i] <- p[i - 1]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  p[1] <- a
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    ΓΡΑΨΕ p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

18	20	14	11	15
15	18	20	14	11

Ολίσθηση δυαδικού αριθμού

Όταν ο προγραμματιστής ορίζει ένα δεδομένο τότε αυτό αποθηκεύεται στα κυκλώματα του υπολογιστή σε δυαδική μορφή δηλαδή σε ακολουθίες από 0 και

Αν για παράδειγμα ο προγραμματιστής εκτελέσει την εντολή : χ←17 τότε μέσα στην μνήμη του υπολογιστή (η οποία ουσιαστικά είναι ένα κύκλωμα) αποθηκεύεται ο ισοδύναμος αριθμός του δυαδικού συστήματος, ο οποίος είναι το 00010001. Ολίσθηση είναι η μετακίνηση όλων των ψηφίων ενός αριθμού κατά μια θέση.

Διακρίνουμε δύο ολισθήσεις:

- **Ολίσθηση** προς τα **αριστερά**. Ισοδυναμεί με πολλαπλασιασμό επί δύο.
  Πώς γίνεται η ολίσθηση ενός δυαδικού ακεραίου προς τα αριστερά κατά μια θέση;
  1. Όλα τα ψηφία μετακινούνται προς τα αριστερά κατά μια θέση (το πρώτο ψηφίο χάνεται)
  2. Στο κενό που δημιουργείται από τη δεξιά πλευρά εισάγεται το ψηφίο 0.

0	**0**	**0**	**1**	**0**	**0**	**0**	**1**	
7	6	5	4	3	2	1	0	
2	2	2	2	2	2	2	2	
0	**0**	**0**	**16**	**0**	**0**	**0**	**1**	**17**

0	**0**	**1**	**0**	**0**	**0**	**1**	**0**	
7	6	5	4	3	2	1	0	
2	2	2	2	2	2	2	2	
0	**0**	**32**	**0**	**8**	**0**	**2**	**0**	**34**

- **Ολίσθηση** προς τα **δεξιά**. Ισοδυναμεί με την ακέραια διαίρεση δια δύο.

  Πώς γίνεται η ολίσθηση ενός δυαδικού ακεραίου προς τα δεξιά κατά μια θέση;

  1. Όλα τα ψηφία μετακινούνται προς τα δεξιά κατά μια θέση (το τελευταίο ψηφίο χάνεται)
  2. Στο κενό που δημιουργείται από την αριστερή πλευρά εισάγεται το ψηφίο 0.

0	**0**	**0**	**1**	**0**	**0**	**0**	**1**	
7	6	5	4	3	2	1	0	
2	2	2	2	2	2	2	2	
0	**0**	**0**	**16**	**0**	**0**	**0**	**1**	**17**

0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	
7	6	5	4	3	2	1	0	
2	2	2	2	2	2	2	2	
0	**0**	**0**	**0**	**8**	**0**	**0**	**0**	**8**

```
ΠΡΟΓΡΑΜΜΑ olisthisi_pol
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: i, p[8], a
  ΠΡΑΓΜΑΤΙΚΕΣ: gin
ΑΡΧΗ
  p[1] <- 1
  p[2] <- 0
  p[3] <- 0
  p[4] <- 0
  p[5] <- 1
  p[6] <- 0
  p[7] <- 0
  p[8] <- 0
  gin <- 0
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 8
    gin <- gin + p[i]*2^(i - 1)
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ gin
!ολίσθηση των στοιχείων κατά μία θέση προς τα αριστερά και
!τοποθέτηση στη θέση του στοιχείου με δύναμη το 0 το 0
  ΓΙΑ i ΑΠΟ 8 ΜΕΧΡΙ 2 ΜΕ_ΒΗΜΑ -1
    p[i] <- p[i - 1]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  p[1] <- 0
  gin <- 0
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 8
    gin <- gin + p[i]*2^(i - 1)
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ "_____"
  ΓΡΑΨΕ gin
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 8
    ΓΡΑΨΕ p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
ΠΡΟΓΡΑΜΜΑ olisthisi_diairesi
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: i, p[8], a
  ΠΡΑΓΜΑΤΙΚΕΣ: gin
ΑΡΧΗ
  p[1] <- 1
  p[2] <- 0
  p[3] <- 0
  p[4] <- 0
  p[5] <- 1
  p[6] <- 0
  p[7] <- 0
  p[8] <- 0
  gin <- 0
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 8
    gin <- gin + p[i]*2^(i - 1)
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ gin
!ολίσθηση των στοιχείων κατά μία θέση προς τα δεξιά και
!τοποθέτηση στο κενό που δημιουργείται από την αριστερή πλευρά το 0
  ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ 8
    p[i - 1] <- p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  p[1] <- 0
  gin <- 0
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 8
    gin <- gin + p[i]*2^(i - 1)
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ "_____"
  ΓΡΑΨΕ gin
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 8
    ΓΡΑΨΕ p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Εξήγηση – Ανάλυση δεκαδικού δυαδικού συστήματος αρίθμησης

Ως γνωστόν, ένας αριθμός στο δυαδικό σύστημα είναι μια ακολουθία από 0 και 1. Όταν θέλουμε να μετατρέψουμε ένα αριθμό από το δυαδικό σύστημα στο δεκαδικό, τον αναλύουμε σε δυνάμεις του 2. Το τελευταίο ψηφίο έχει βαρύτητα 0 και καθώς προχωράμε προς τα αριστερά η βαρύτητα αυξάνεται κατά 1 (η βαρύτητα είναι ο εκθέτης της δύναμης του 2 με την οποία πολλαπλασιάζεται το αντίστοιχο ψηφίο).

Π.χ. Ο δυαδικός ακέραιος 00011001 είναι ο :

1*1+0*2+0*4+1*8+1*16 = 25 στο δεκαδικό σύστημα αρίθμησης.

Π.χ. Αν ολισθήσουμε τον 00011001 προς τα δεξιά κατά μια θέση θα πάρουμε ως αποτέλεσμα τον δυαδικό ακέραιο 00001100. Το αποτέλεσμά

μας είναι ο αριθμός 0*1+0*2+1*4+1*8 = 12 στο δεκαδικό σύστημα αρίθμησης. Όμως ισχύει:

12 = 25 div 2

Δηλαδή, η ολίσθηση ενός δυαδικού ακέραιου προς τα δεξιά κατά μια θέση, προκαλεί τον υποδιπλασιασμό του.

Πώς γίνεται η ολίσθηση ενός δυαδικού ακεραίου προς τα αριστερά κατά μια θέση;

1. Όλα τα ψηφία μετακινούνται προς τα αριστερά κατά μια θέση (το πρώτο ψηφίο χάνεται)
2. **2.** Στο κενό που δημιουργείται από τη δεξιά πλευρά εισάγεται το ψηφίο 0.
3. Π.χ. Αν ολισθήσουμε τον 00011001 προς τα αριστερά κατά μια θέση θα πάρουμε ως αποτέλεσμα τον δυαδικό ακέραιο 00110010.
   Το αποτελεσμά μας είναι ο αριθμός 0*1+1*2+0*4+0*8+1*16+1*32 = 50 στο δεκαδικό σύστημα αρίθμησης.
   Όμως ισχύει: 50 = 25 * 2
   Δηλαδή, η ολίσθηση ενός δυαδικού ακέραιου προς τα αριστερά κατά μια θέση, προκαλεί τον διπλασιασμό του.

Αντιδιαμετρική αντιμετάθεση στοιχείων πίνακα

```
ΠΡΟΓΡΑΜΜΑ antimeta
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: p[5], k, tmp, i
ΑΡΧΗ
  p[1] <- 18
  p[2] <- 20
  p[3] <- 14
  p[4] <- 11
  p[5] <- 15
  ΓΙΑ k ΑΠΟ 1 ΜΕΧΡΙ (5 div 2)
    tmp <- p[k]
    p[k] <- p[(5 + 1) - k]
    p[(5 + 1) - k] <- tmp
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    ΓΡΑΨΕ p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

18	20	14	11	15
15	11	14	20	18

Συγχώνευση πινάκων: οι πίνακες p1[10] και p2[5] συγχωνεύονται στον πίνακα p[15]

```
ΠΡΟΓΡΑΜΜΑ merge
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: p1[10], p2[5], p[15], tmp, i
ΑΡΧΗ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    p1[i] <- i
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  p2[1] <- 18
  p2[2] <- 20
  p2[3] <- 14
  p2[4] <- 11
  p2[5] <- 15
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    p[i] <- p1[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 11 ΜΕΧΡΙ 15
    p[i] <- p2[i - 10]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 15
    ΓΡΑΨΕ p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
Αλγόριθμος συγχώνευση
!2016 ΠΑΛΑΙΟ
!Β2. Δίνεται ο πίνακας αριθμών Χ[50], ταξινομημένος κατά φθίνουσα σειρά,
!και ο πίνακας Υ[100], ταξινομημένος κατά αύξουσα σειρά. Να θεωρήσετε
!ότι οι τιμές κάθε πίνακα είναι διαφορετικές μεταξύ τους και ότι οι δύο
!πίνακες δεν έχουν κοινές τιμές.
!Το παρακάτω ημιτελές τμήμα αλγορίθμου δημιουργεί ένα νέο πίνακα
!Ζ[10], ταξινομημένο σε φθίνουσα σειρά, με τις δέκα μεγαλύτερες τιμές
!από τις εκατόν πενήντα (150) τιμές των δύο πινάκων.
! i <- □(1)□
! j <- □(2)□
! Για k από 1 μέχρι 10
! Αν Χ[ i ] □(3)□ Υ[ j ] τότε
!    Ζ[ k ] <- Χ[ i ]
!    i <- i □(4)□ 1
! Αλλιώς
!    Ζ[ k ] <- Υ[ j ]
!    j <- j □(5)□ 1
! Τέλος_αν
! Τέλος_επανάληψης
!Να γράψετε στο τετράδιό σας τους αριθμούς (1) έως (5), που
!αντιστοιχούν στα κενά του αλγορίθμου, και, δίπλα σε κάθε αριθμό, ό,τι
!πρέπει να συμπληρωθεί, ώστε το τμήμα αλγορίθμου να επιτελεί τη
!λειτουργία που περιγράφεται.
!Μονάδες 10
Δεδομένα // Χ, Υ //

i ← 1
j ← 100
Για k από 1 μέχρι 10
  Αν Χ[ i] > Υ[ j] τότε
    Ζ[ k] ← Χ[ i]
    i ← i + 1
  αλλιώς
    Ζ[ k] ← Υ[ j]
    j ← j - 1
  Τέλος_αν
Τέλος_επανάληψης

Για i από 1 μέχρι 10
  Εμφάνισε Ζ[i]
Τέλος_επανάληψης
Τέλος συγχώνευση
```

Διαχωρισμός πινάκων

Αντίστροφη διαδικασία. Από τον p[15] στους p1[10] και p2[5].

```
ΠΡΟΓΡΑΜΜΑ diairesi
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: p1[10], p2[5], p[15], tmp, i
ΑΡΧΗ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    p[i] <- i
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  p[11] <- 18
  p[12] <- 20
  p[13] <- 14
  p[14] <- 11
  p[15] <- 15
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    p1[i] <- p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    p2[i] <- p[10 + i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    ΓΡΑΨΕ "p1[", i, "]=", p1[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    ΓΡΑΨΕ "p2[", i, "]=", p2[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Οθόνη εκτέλεσης

```
p1[1]=1
p1[2]=2
p1[3]=3
p1[4]=4
p1[5]=5
p1[6]=6
p1[7]=7
p1[8]=8
p1[9]=9
p1[10]=10
----------------
p2[1]=18
p2[2]=20
p2[3]=14
p2[4]=11
p2[5]=15
```

Οθόνη εκτέλεσης

```
1
2
3
4
5
6
7
8
9
10
18
20
14
11
15
```

Συγχώνευση με παράλληλη ταξινόμηση σε ήδη ταξινομημένους πίνακες.

Έστω οι ταξινομημένοι πίνακες ακεραίων p1[10] και p2[5] (σε αύξουσα σειρά).

Ζητούμενο είναι η δημιουργία ενός νέου ταξινομημένου πίνακα p[15].

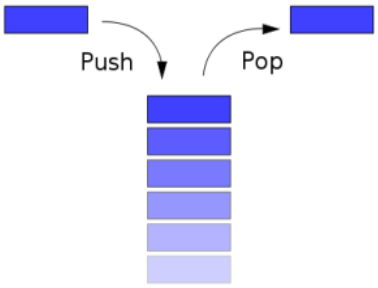
```
ΠΡΟΓΡΑΜΜΑ diairesi
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: p1[10], p2[5], p[15], i, k, l
ΑΡΧΗ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    p1[i] <- 2*i
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    p2[i] <- 2*i + 1
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    ΓΡΑΨΕ "p1[", i, "]=", p1[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ "--------------------"
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 5
    ΓΡΑΨΕ "p2[", i, "]=", p2[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ "--------------------"
  k <- 1
  l <- 1
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 15
    ΑΝ k <= 10 ΤΟΤΕ
      ΑΝ l <= 5 ΤΟΤΕ
        ΑΝ p1[k] < p2[l] ΤΟΤΕ
          p[i] <- p1[k]
          k <- k + 1
        ΑΛΛΙΩΣ
          p[i] <- p2[l]
          l <- l + 1
        ΤΕΛΟΣ_ΑΝ
      ΑΛΛΙΩΣ
        p[i] <- p1[k]
        k <- k + 1
      ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ
      p[i] <- p2[l]
      l <- l + 1
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 15
    ΓΡΑΨΕ "p[", i, "]=", p[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Οθόνη εκτέλεσης

```
p1[1]=2
p1[2]=4
p1[3]=6
p1[4]=8
p1[5]=10
p1[6]=12
p1[7]=14
p1[8]=16
p1[9]=18
p1[10]=20
--------------------
p2[1]=3
p2[2]=5
p2[3]=7
p2[4]=9
p2[5]=11
--------------------
p[1]=2
p[2]=3
p[3]=4
p[4]=5
p[5]=6
p[6]=7
p[7]=8
p[8]=9
p[9]=10
p[10]=11
p[11]=12
p[12]=14
p[13]=16
p[14]=18
p[15]=20
```

Στοίβα – Ουρά

οι Στοίβες και οι Ουρές υλοποιούνται με μονοδιάστατους πίνακες



Στοίβα

- Έχει ομοιότητες με μία στοίβα πιάτων. Κάθε νέο πιάτο μπαίνει στην κορυφή της στοίβας και όταν θέλουμε να αφαιρέσουμε ένα πιάτο το αφαιρούμε από την κορυφή (Last In First Out - LIFO).
- Για την υλοποίηση της στοίβας απαιτείται ένας πίνακας Π[ν] και μία μεταβλητή με όνομα κορυφή (top).
- Αρχικά ο πίνακας είναι άδειος και η μεταβλητή κορυφή έχει τιμή 0. Όταν θέλουμε να εισάγουμε μία νέα τιμή, την εισάγουμε στο κελί Π[κορυφή+1] και αυξάνουμε την τιμή της μεταβλητής κορυφή κατά 1. Η διαδικασία της εισαγωγής δεδομένων στην Στοίβα ονομάζεται Ώθηση (Push). Για να πάρουμε μία τιμή από την Στοίβα, πάμε στο κελί Π[κορυφή] και μειώνουμε την τιμή της μεταβλητής κορυφή κατά 1. Η διαδικασία της εξαγωγής δεδομένων από την Στοίβα ονομάζεται Απώθηση (Pop).
- Δεν πρέπει να επιχειρείται ώθηση όταν η στοίβα είναι γεμάτη διότι θα προκληθεί Υπερχείλιση (Overflow) αλλά ούτε και απώθηση σε άδεια στοίβα θα προκληθεί υπερχείλιση (Underflow).
- Στην Απώθηση δεν είναι απαραίτητο να διαγράψουμε την τιμή στο τελευταίο κελί. Απλά μειώνουμε κατά ένα την μεταβλητή κορυφή.

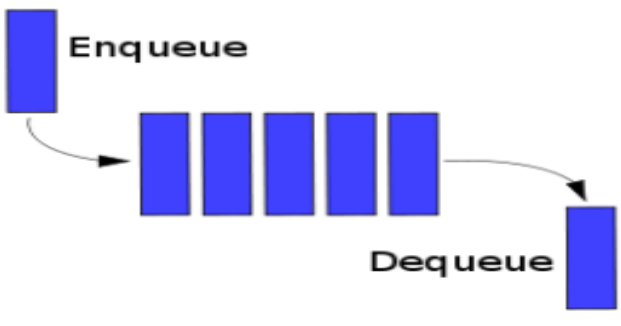
```
ΠΡΟΓΡΑΜΜΑ ΣΤΟΙΒΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: Α[10]
  ΑΚΕΡΑΙΕΣ: top, απάντηση, i
ΑΡΧΗ
  top <- 0
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Δώστε 1: ώθηση  2: απώθηση  3:εμφάνιση 9:έξοδος'
    ΔΙΑΒΑΣΕ απάντηση
    ΑΝ απάντηση = 1 ΤΟΤΕ
!  **********ώθηση**********
      ΑΝ top < 10 ΤΟΤΕ
        top <- top + 1
        ΓΡΑΨΕ 'Δώστε τιμή για ώθηση : '
        ΔΙΑΒΑΣΕ Α[top]
      ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'Γεμάτη στοίβα'
      ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ_ΑΝ απάντηση = 2 ΤΟΤΕ
!  ********** απώθηση**********
      ΑΝ top > 0 ΤΟΤΕ
        ΓΡΑΨΕ 'Απώθηση : ', Α[top]
        Α[top] <- ''
        top <- top - 1
      ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'Αδεια στοίβα'
      ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ_ΑΝ απάντηση = 3 ΤΟΤΕ
!  **********εμφάνιση**********
      ΑΝ top > 0 ΤΟΤΕ
        ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ top
          ΓΡΑΨΕ i, ' ', Α[i]
        ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
      ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'Αδεια στοίβα'
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ απάντηση = 9
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Ουρά

- Έχει ομοιότητες με μία ουρά ανθρώπων σε κάποιο ταμείο π.χ. τράπεζας. Κάθε νέος πελάτης μπαίνει στο τέλος της ουράς και ο επόμενος πελάτης που θα εξυπηρετηθεί είναι αυτός που είναι πρώτος στην ουρά. (First In First Out - FIFO).
- Για την υλοποίηση της στοίβας απαιτείται ένας πίνακας Π[ν] και δύο μεταβλητές με ονόματα εμπρός (front) που δείχνει τη **θέση του 1ου στοιχείου** της ουράς και την **rear** (ή πίσω) που δείχνει τη **θέση του τελευταίου στοιχείου**. και πίσω (rear).
- Αρχικά ο πίνακας είναι άδειος και οι μεταβλητές εμπρός και πίσω έχουν την τιμή 0. Τα νέα στοιχεία της ουράς **εισέρχονται** στο τέλος της (πίσω άκρο της ουράς) και η τιμή της μεταβλητής rear αλλάζει ως εξής: rear ← rear+1
  ενώ κάθε στοιχείο που θέλουμε να **εξάγουμε** αφαιρείται από την αρχή της ουράς (το εμπρός άκρο της ουράς) και η τιμή της μεταβλητής front αλλάζει ως εξής: front ← front +1.

  Η διαδικασία της εισαγωγής δεδομένων στην ουρά ονομάζεται Εισαγωγή (Enqueue). Η διαδικασία της εξαγωγής δεδομένων από την ουρά ονομάζεται Εξαγωγή (Dequeue). Δεν πρέπει να επιχειρείται Εισαγωγή όταν η ουρά είναι γεμάτη αλλά ούτε και Εξαγωγή σε άδεια στοίβα διότι θα προκληθεί Υπερχείλιση (Overflow/Underflow). Οι εισαγωγές και εξαγωγές δεδομένων έχει σαν αποτέλεσμα η μεταβλητή rear κάποια στιγμή να δείχνει στο τελευταίο κελί του πίνακα. Για να ξεπεράσουμε το πρόβλημα έχουμε δύο επιλογές: να θεωρήσουμε την ουρά ως **γραμμική** και ως **κυκλική**.
- Γραμμική υλοποίηση: Αν ο πίνακας δεν είναι γεμάτος τότε μετακινούμε όλα τα στοιχεία της ουράς στην αρχή.
- Κυκλική υλοποίηση : Αν ο πίνακας δεν είναι γεμάτος τότε το επόμενο στοιχείο εισάγεται στην αρχή του πίνακα.

```
ΠΡΟΓΡΑΜΜΑ ΟΥΡΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: Α[10]
  ΑΚΕΡΑΙΕΣ: f, r, απάντηση, i
ΑΡΧΗ
  f <- 0
  r <- 0
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Δώστε 1: εισαγωγή  2: εξαγωγή  3:εμφάνιση 9:έξοδος'
    ΔΙΑΒΑΣΕ απάντηση
    ΕΠΙΛΕΞΕ απάντηση
      ΠΕΡΙΠΤΩΣΗ 1
        ΑΝ r < 10 ΤΟΤΕ
          r <- r + 1
          ΓΡΑΨΕ 'Δώστε τιμή για εισαγωγή : '
          ΔΙΑΒΑΣΕ Α[r]
          ΑΝ r = 1 ΤΟΤΕ
            f <- 1
          ΤΕΛΟΣ_ΑΝ
        ΑΛΛΙΩΣ
          ΓΡΑΨΕ 'Γεμάτη ουρά'
        ΤΕΛΟΣ_ΑΝ
      ΠΕΡΙΠΤΩΣΗ 2
        ΑΝ f > 0 ΤΟΤΕ
          ΓΡΑΨΕ 'Εξαγωγή : ', Α[f]
          Α[f] <- ''
          f <- f + 1
          ΑΝ f > r ΤΟΤΕ
            f <- 0
            r <- 0
          ΤΕΛΟΣ_ΑΝ
        ΑΛΛΙΩΣ
          ΓΡΑΨΕ 'Αδεια ουρά'
        ΤΕΛΟΣ_ΑΝ
      ΠΕΡΙΠΤΩΣΗ 3
        ΑΝ f > 0 ΤΟΤΕ
          ΓΙΑ i ΑΠΟ f ΜΕΧΡΙ r
            ΓΡΑΨΕ i, ' ', Α[i]
          ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
        ΑΛΛΙΩΣ
          ΓΡΑΨΕ 'Αδεια ουρά'
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ απάντηση = 9
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```



1.2 Ουρά

Δυναμικές Δομές δεδομένων. FIFO

1.2.2 Ερωτήσεις - Ασκήσεις

Να γράψετε ΠΡΌΓΡΑΜΜΑ το οποίο να **υλοποιεί την εξαγωγή** ενός **στοιχείου** από μία **ουρά 10 θέσεων**.
Μετά από **κάθε εξαγωγή** να **μεταφέρονται** όλα τα στοιχεία της ουράς **μία θέση αριστερά** (δηλαδή η **θέση του μπροστινού δείκτη της ουράς να είναι πάντα 1**).
Η **μετακίνηση προς τα αριστερά** να γίνεται ανεξαρτήτως αν η ουρά είναι γεμάτη ή όχι.


```
ΠΡΟΓΡΑΜΜΑ      ΚΥΚΛΙΚΗ_ΟΥΡΑ
  ΑΚΕΡΑΙΕΣ: αρχή,τέλος,i ,πλήθος
  ΧΑΡΑΚΤΗΡΕΣ: στοιχειο,  ΟΥΡΑ[10]],απάντηση

ΑΡΧΗ
    αρχή <- 0
    τέλος <- 0
    ΑΝ   (αρχή=0 ΚΑΙ τέλος=0)   ΤΟΤΕ
         αρχή <- 1
         τέλος <- 1
         ΓΡΑΨΕ 'ΔΩΣΕ ΣΤΟΙΧΕΙΟ ΓΙΑ ΕΙΣΑΓΩΓΗ 1ΟΥ ΣΤΟΙΧΕΙΟΥ ΣΤΗΝ ΟΥΡΑ'
         ΔΙΑΒΑΣΕ  στοιχείο
        ΟΥΡΑ [ τέλος ] <-  στοιχείο
    ΤΕΛΟΣ_ΑΝ
    ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΑΝ      αρχή>0 ΚΑΙ τέλος<10    ΤΟΤΕ
          ΓΡΑΨΕ   ' ΔΩΣΕ ΣΤΟΙΧΕΙΟ ΓΙΑ ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΟΥΡΑ '
          ΔΙΑΒΑΣΕ   στοιχείο
          τέλος <-  τέλος+1
         ΟΥΡΑ [ τέλος ] <-  στοιχείο
    ΑΛΛΙΩΣ       !τέλος=10
          ΓΡΑΨΕ   'Δεν μπορεί να εισέλθει άλλο στοιχείο '
    ΤΕΛΟΣ_ΑΝ
    ΓΡΑΨΕ    ' Επιθυμείτε να εισάγετε  και άλλο στοιχείο στην ουρά;(Ν ή Ο) '
    ΔΙΑΒΑΣΕ     απάντηση
    ΜΕΧΡΙΣ_ΟΤΟΥ    απάντηση='Ο' Η τέλος=10
```

```
! ΕΞΑΓΩΓΗ ΑΠΟ ΤΗ ΟΥΡΑ  ΚΑΙ ΜΕΤΑΤΟΠΙΣΗ ΠΡΟΣ ΤΑ ΑΡΙΣΤΕΡΑ
ΑΝ  αρχή <= τέλος  ΚΑΙ    τέλος >0   ΤΟΤΕ
 ! Για να σιγουρέψουμε ότι υπάρχει ένα τουλάχιστον στοιχείο
     αρχή   <-   αρχή + 1
     στοιχείο <- Ουρά[αρχή]  ! Στοιχείο που εξέρχεται
 ΓΡΑΨΕ ' ΕΞΕΡΧΕΤΑΙ ΤΟ 1ο ΣΤΟΙΧΕΙΟ ΤΗΣ ΟΥΡΑΣ ΜΕ ΤΙΜΗ ', στοιχείο
  ! Μετακίνηση όλων των στοιχείων της ουράς μία θέση αριστερά
     Αν   αρχή <= τέλος   ΤΟΤΕ
  ! Για να σιγουρέψουμε ότι η ουρά δεν είναι άδεια
  πλήθος   <-   τέλος-αρχή + 1
     Για    i ΑΠΟ 1 ΜΕΧΡΙ πλήθος
              Ουρά[i] ← Ουρά[αρχή-1+i]
     ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
                  αρχή  <- αρχή - 1
                  τέλος <- τέλος - 1
   ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ     ΚΥΚΛΙΚΗ_ΟΥΡΑ
```

Συνήθως την ΟΥΡΑ την "γεμίζουμε" με μια επαναληπτική δομή

Δυναμικές Δομές δεδομένων. FIFO

Σε ένα ταχυδρομικό κατάστημα, οι πελάτες εξυπηρετούνται με βάση τη σειρά άφιξής τους σε αυτό. Το ταχυδρομικό κατάστημα έχει ένα ταμείο και ο μέσος χρόνος εξυπηρέτησης κάθε πελάτη είναι 3 λεπτά.

Να αναπτύξετε πρόγραμμα σε ΓΛΩΣΣΑ το οποίο:

1. Να δέχεται σαν είσοδο από τον χρήστη μία εκ των δύο τιμών εισαγωγής: «1.ΕΙΣΑΓΩΓΗ» ή «2.ΕΠΟΜΕΝΟΣ» «3.ΕΞΟΔΟΣ» (με έλεγχο εγκυρότητας).
2. Αν δοθεί η τιμή «1.ΕΙΣΑΓΩΓΗ», τότε το πρόγραμμα να διαβάζει το ονοματεπώνυμο του πελάτη και αμέσως μετά να εμφανίζει το πλήθος των ατόμων που περιμένουν πριν από αυτόν, εκτός αν η ουρά αναμονής είναι γεμάτη, οπότε να εμφανίζει το μήνυμα «Το κατάστημα γέμισε. Παρακαλούμε ελάτε άλλη φορά».

3. Αν δοθεί η τιμή «2.ΕΠΟΜΕΝΟΣ», τότε το πρόγραμμα να εμφανίζει το ονοματεπώνυμο του πελάτη προς εξυπηρέτηση.
4. Η παραπάνω διαδικασία να επαναλαμβάνεται μέχρι να εξυπηρετηθούν όλοι οι πελάτες.
5. 5.Στο τέλος το πρόγραμμα να εμφανίζει το πλήθος των ατόμων που εξυπηρετήθηκαν, καθώς και τον μέσο χρόνο αναμονής των πελατών.

```
ΠΡΟΓΡΑΜΜΑ postOffice
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: f, r, max_cust, hours, epilogi, pl_cust, pl_queue, i, pl_srv, wait_time, ttl_wait_time
  ΧΑΡΑΚΤΗΡΕΣ: q_ep[30], epitheto
ΑΡΧΗ
  f <- 0
  r <- 0
  pl_cust <- 0
  pl_queue <- 0
  ttl_wait_time <- 0
  pl_srv <- 0
  ΓΡΑΨΕ 'Δώσε ωρες λειτουργίας:'
  ΔΙΑΒΑΣΕ hours
  max_cust <- (hours*60) div 3
  ΓΡΑΨΕ 'Μεγιστος αριθμός πελατών: ', max_cust
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ '1.ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΟΥΡΑ'
    ΓΡΑΨΕ '2.ΕΠΟΜΕΝΟΣ ΓΙΑ ΕΞΥΠΗΡΕΤΗΣΗ'
    ΓΡΑΨΕ '0.ΕΞΟΔΟΣ'
    ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
      ΔΙΑΒΑΣΕ epilogi
      ΑΝ epilogi < 0 Η epilogi > 2 ΤΟΤΕ
        ΓΡΑΨΕ 'ΛΑΘΟΣ ΕΠΙΛΟΓΗ!!! ΞΑΝΑΠΡΟΣΠΑΘΗΣΕ'
      ΤΕΛΟΣ_ΑΝ
    ΜΕΧΡΙΣ_ΟΤΟΥ epilogi >= 0 ΚΑΙ epilogi < 3
    ΑΝ epilogi = 1 ΤΟΤΕ
      ΑΝ pl_cust = max_cust ΤΟΤΕ
        ΓΡΑΨΕ 'ΤΟ ταχυδρομείο δεν μπορεί να εξυπηρετήσει άλλους!!'
        ΓΡΑΨΕ 'Ελάτε άλλη μέρα!!'
      ΑΛΛΙΩΣ_ΑΝ pl_queue = 30 ΤΟΤΕ
        ΓΡΑΨΕ 'Η ουρά είναι πλήρης!!'
        ΓΡΑΨΕ 'Ελάτε αργότερα!!'
      ΑΛΛΙΩΣ_ΑΝ r < 30 ΤΟΤΕ
        r <- r + 1
        ΓΡΑΨΕ 'Πληκτρολόγησε επίθετο:'
        ΔΙΑΒΑΣΕ epitheto
        q_ep[r] <- epitheto
        ΑΝ r = 1 ΤΟΤΕ
          f <- 1
        ΤΕΛΟΣ_ΑΝ
        pl_cust <- pl_cust + 1
        pl_queue <- r - f
        wait_time <- pl_queue*3
        ttl_wait_time <- ttl_wait_time + wait_time
        ΓΡΑΨΕ 'κ.', q_ep[r], ' χρονος αναμονής-->>', wait_time
        ΓΡΑΨΕ ' προηγούνται ', pl_queue, ' πελατες '
        ΓΡΑΨΕ
        ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ r
```

```
          ΓΡΑΨΕ 'ΟΥΡΑ[', i, ']=', q_ep[i]
        ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
      ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ_ΑΝ epilogi = 2 ΤΟΤΕ
      ΑΝ f > 0 ΤΟΤΕ
        ΓΡΑΨΕ 'Ο κ. ', q_ep[f], ' καλείται για εξυπηρέτηση!!'
        q_ep[f] <- ''
        f <- f + 1
        pl_queue <- pl_queue - 1
        pl_srv <- pl_srv + 1
        ΑΝ f > 1 ΤΟΤΕ
          ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ r
            q_ep[i - 1] <- q_ep[i]
            ΓΡΑΨΕ 'q_ep[', i - 1, ']=', q_ep[i - 1]
          ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
          q_ep[r] <- ''
          r <- r - 1
          f <- f - 1
        ΤΕΛΟΣ_ΑΝ
        ΑΝ f > r ΤΟΤΕ
          f <- 0
          r <- 0
        ΤΕΛΟΣ_ΑΝ
      ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΟΥΡΑ ΑΔΕΙΑ'
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ epilogi = 0
  ΓΡΑΨΕ ' Οι πελάτες που εξυπηρετήθηκαν είναι:', pl_srv
  ΓΡΑΨΕ 'MEAN TIME', ttl_wait_time/ pl_cust
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
ΠΡΟΓΡΑΜΜΑ ΤΑΧΥΔΡΟΜΕΙΟ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ:  αρχή, τέλος, επ, ώρες,μεγ_πληθος,πλήθ,πλήθολ
  ΑΚΕΡΑΙΕΣ: χρόνος_αναμ
  ΧΑΡΑΚΤΗΡΕΣ: επίθετο,  Λίστα_επίθετων[30]
ΑΡΧΗ
    αρχή <- 0
    τέλος <- 0
    πλήθ<- 0
    πλήθολ<- 0
    χρόνος_αναμ<- 0
    πλ_εξυπ <- 0
    μεγ_πληθος <- 0
ΓΡΑΨΕ  ' Πόσες ώρες λειτουργεί το κατάστημα; '
ΔΙΑΒΑΣΕ   ώρες
  μεγ_πληθος <- (ώρες* 60) div 3
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ          ' (1) ΕΙΣΑΓΩΓΗ' '
    ΓΡΑΨΕ          ' (2) ΕΠΟΜΕΝΟΣ' '
    ΓΡΑΨΕ          ' (3) ΕΞΟΔΟΣ'
    ΓΡΑΨΕ          ' Δώσε επιλογή: '
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
      ΔΙΑΒΑΣΕ  επ
   ΑΝ  επ<>1 ΚΑΙ επ<>2 ΚΑΙ επ<>3      ΤΟΤΕ
        ΓΡΑΨΕ  ' Λάθος επιλογή. Ξαναπροσπάθησε!!! '
  ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ επ=1 Η επ=2 Η επ=3
ΑΝ  επ=1   ΤΟΤΕ
 ΑΝ   πλήθολ =μεγ_πληθος   ΤΟΤΕ
ΓΡΑΨΕ ' Το ταχυδρομείο δεν μπορεί να εξυπηρετήσει άλλους!'
ΓΡΑΨΕ ' Παρακαλούμε ελάτε άλλη φορά.'
```

```
ΑΛΛΙΩΣ_ΑΝ   πλήθ =30 ΤΟΤΕ
  ΓΡΑΨΕ  ' Η ουρά αναμονής είναι πλήρης.'
  ΓΡΑΨΕ  ' Παρακαλούμε ελάτε αργότερα.'
ΑΛΛΙΩΣ
  ΑΝ   αρχή>1 ΚΑΙ τέλος=30   ΤΟΤΕ
   ΓΙΑ  i ΑΠΟ 1 ΜΕΧΡΙ πλήθ
    λίστα_επίθετων[i]  <- λίστα_επίθετων[αρχή – 1+i]
    ! Ολίσθηση(SHIFT) προς την πρώτη θέση
   ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
   ΓΙΑ  i ΑΠΟ πλήθ+1 ΜΕΧΡΙ τέλος
    λίστα_επίθετων[i]  <- "    "
   ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
     αρχή <- 1
     τέλος <- πλήθ
  ΤΕΛΟΣ_ΑΝ
  ΓΡΑΨΕ  ' Πληκτρολογήστε το επίθετό σας .'
  ΔΙΑΒΑΣΕ    επίθετο
     πλήθ<- πλήθ +1
     πλήθολ<- πλήθολ+1
  ΑΝ   αρχή=0 ΚΑΙ τέλος=0       ΤΟΤΕ
       αρχή <- 1
       τέλος <- 1
  ΑΛΛΙΩΣ
       τέλος <- τέλος + 1
ΤΕΛΟΣ_ΑΝ
   χρόνος_αναμ<- χρόνος_αναμ + (πλήθ-1)*3
    λίστα_επίθετων[τέλος ]  <- επίθετο
  ΓΡΑΨΕ  ' κ. ', λίστα_επίθετων[τελος]
  ΓΡΑΨΕ
  ΓΡΑΨΕ  πλήθ - 1,' πελάτες'
ΤΕΛΟΣ_ΑΝ
```

```
!  ΚΛΗΣΗ ΕΠΟΜΕΝΟΥ ΠΕΛΑΤΗ ΚΑΙ
!  ΔΙΑΓΡΑΦΗ ΤΟΥ ΠΡΟΗΓΟΥΜΕΝΟΥ ΑΠΟ ΤΗΝ ΟΥΡΑ
ΑΛΛΙΩΣ_ΑΝ   επ=2   ΤΟΤΕ
   ΑΝ   (αρχή=0 ΚΑΙ τέλος=0 ) Η αρχή>τέλος   ΤΟΤΕ
      ΓΡΑΨΕ  ' Η ουρά είναι άδεια '
      ΓΡΑΨΕ  ' Δεν υπάρχει πελάτης να εξυπηρετηθεί '
   ΑΛΛΙΩΣ_ΑΝ  αρχή<= τέλος   ΤΟΤΕ
      πλήθ <- πλήθ -1
      ΓΡΑΨΕ  ' Καλείται ο/η κ. ', λίστα_επίθετων[αρχή]
      λίστα_επίθετων[αρχή]  <- " "
      αρχ <- αρχ+1
      πλ_εξυπ <- πλ_εξυπ+1
      ΑΝ  αρχή> τέλος    ΤΟΤΕ
            αρχή <- 0
            τέλος <- 0
      ΤΕΛΟΣ_ΑΝ
   ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
ΜΕΧΡΙΣ_ΟΤΟΥ επ=3  Η (πλ_εξυπ = μεγ_πληθος)
ΓΡΑΨΕ  'Οι πελάτες που εξυπηρετήθηκαν είναι ', πλ_εξυπ
   ΑΝ  πλ_εξυπ <> 0    ΤΟΤΕ
   ΓΡΑΨΕ  'Ο μέσος χρόνος αναμονής είναι '
ΑΝ  πλ_εξυπ <> 0    ΤΟΤΕ
ΓΡΑΨΕ  'Ο μέσος χρόνος αναμονής είναι '
ΓΡΑΨΕ  χρονος_αναμ/ πλ_εξυπ
ΑΛΛΙΩΣ
ΓΡΑΨΕ  'Ο μέσος χρόνος αναμονής είναι  0'
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ ΤΑΧΥΔΡΟΜΕΙΟ
```

Παρόμοια:

Προκειμένου ένας πολίτης να πληρώσει τον λογαριασμό του στη ΔΕΗ λαμβάνει αυτόματα έναν αύξοντα διαδοχικό αριθμό προτεραιότητας που καθορίζει τη σειρά του στην ουρά του ταμείου η οποία δε μπορεί να ξεπερνάει τα 30 άτομα. Ο ταμίας εξυπηρετεί τον πρώτο πελάτη που βρίσκεται στην ουρά και η εξυπηρέτηση διαρκεί 3 λεπτά κατά μέσο όρο. Επομένως κάθε πολίτης που περιμένει στην ουρά περιμένει κατά μέσο όρο 3 λεπτά για κάθε έναν που βρίσκεται στην ουρά πριν απ' αυτόν. Να αναπτύξετε προγραμμα σε ΓΛΩΣΣΑ το οποίο:

1) Εμφανίζει ένα μενού με τις επιλογές «ΕΙΣΑΓΩΓΗ», «ΕΞΑΓΩΓΗ», «ΤΕΡΜΑΤΙΣΜΟΣ» ως πιθανές ενέργειες.
   - Αν δοθεί η ενέργεια «ΕΙΣΑΓΩΓΗ» Πότε διαβάζει το ονοματεπώνυμό του πελάτη και αν η ουρά έχει γεμίσει εμφανίζει το μήνυμα «Ελάτε σε 3 ώρες», Διαφορετικά τον τοποθετεί σε ουρά ΟΝ 30 θέσεων και εμφανίζει το πλήθος των ατόμων που περιμένουν στην ουρά πριν από αυτόν καθώς και τον εκτιμώμενο χρόνο αναμονής.
   - Αν δοθεί η ενέργεια «ΕΞΑΓΩΓΗ» Εμφανίζει το ονοματεπώνυμο του πολίτη που θα εξυπηρετηθεί. (Η διαδικασία επαναλαμβάνεται μέχρι να δοθεί η επιλογή «ΤΕΡΜΑΤΙΣΜΟΣ»)
2) Μετά το τέλος την εξυπηρέτησης εμφανίζει:
   •Το πλήθος των πολιτών που εξυπηρετήθηκαν.
   •Τον μέσο χρόνο αναμονής των πολιτών.
   (Θεωρήστε ότι θα εισέλθει τουλάχιστον ένας πελάτης στη ΔΕΗ)

```
ΠΡΟΓΡΑΜΜΑ ΔΕΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: ΟΥΡΑ[30], ΟΝΟΜΑ
  ΑΚΕΡΑΙΕΣ: choice, front, rear, π, Ι, Σ
  ΠΡΑΓΜΑΤΙΚΕΣ: Μέσος_χρόνος
  ΛΟΓΙΚΕΣ: done
ΑΡΧΗ
! Η επεξεργασία της ουράς γίνεται χωρίς ολίσθηση των στοιχείων της
! κατά μιά θέση μπροστά μετά από κάθε εξαγωγή (όπως γίνεται πραγματικά)
  front <- 0
  rear <- 0
  π <- 0
  ΓΡΑΨΕ 'ΔΩΣΕ 1 ΓΙΑ ΕΙΣΑΓΩΓΗ, 2 ΓΙΑ ΕΞΑΓΩΓΗ, 3 ΓΙΑ ΤΕΡΜΑΤΙΣΜΟ'
  ΔΙΑΒΑΣΕ choice
  ΟΣΟ choice <> 3 ΕΠΑΝΑΛΑΒΕ
    ΑΝ choice = 1 ΤΟΤΕ
      ΔΙΑΒΑΣΕ ΟΝΟΜΑ
      ΚΑΛΕΣΕ ΕΙΣΑΓΩΓΗ(ΟΥΡΑ, ΟΝΟΜΑ, front, rear, done)
      ΑΝ done = ΨΕΥΔΗΣ ΤΟΤΕ
        ΓΡΑΨΕ 'Ελάτε σε 3 ώρες'
      ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'Προηγούνται ', rear - front, 'άτομα'
        ΓΡΑΨΕ 'Χρόνος αναμονής ', (rear - front)*3, 'λεπτά'
      ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ_ΑΝ choice = 2 ΤΟΤΕ
      ΚΑΛΕΣΕ ΕΞΑΓΩΓΗ(ΟΥΡΑ, ΟΝΟΜΑ, front, rear, done)
      ΑΝ done = ΑΛΗΘΗΣ ΤΟΤΕ
        ΓΡΑΨΕ ΟΝΟΜΑ
        π <- π + 1
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
    ΓΡΑΨΕ 'ΔΩΣΕ 1 ΓΙΑ ΕΙΣΑΓΩΓΗ, 2 ΓΙΑ ΕΞΑΓΩΓΗ, 3 ΓΙΑ ΤΕΡΜΑΤΙΣΜΟ'
```

```
    ΔΙΑΒΑΣΕ choice
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ π
  Σ <- 0
  ΓΙΑ Ι ΑΠΟ 1 ΜΕΧΡΙ π
    Σ <- Σ + Ι*3
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
! π<>0 (από εκφώνηση)
  Μέσος_χρόνος <- Σ/π
  ΓΡΑΨΕ Μέσος_χρόνος
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
ΔΙΑΔΙΚΑΣΙΑ ΕΙΣΑΓΩΓΗ(ΟΥΡΑ, ΟΝΟΜΑ, front, rear, done)
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: ΟΥΡΑ[30], ΟΝΟΜΑ
  ΑΚΕΡΑΙΕΣ: front, rear
  ΛΟΓΙΚΕΣ: done
ΑΡΧΗ
  ΑΝ rear = 30 ΤΟΤΕ
    done <- ΨΕΥΔΗΣ
  ΑΛΛΙΩΣ_ΑΝ (front = 0 ΚΑΙ rear = 0) ΤΟΤΕ
    front <- 1
    rear <- 1
    ΟΥΡΑ[rear] <- ΟΝΟΜΑ
    done <- ΑΛΗΘΗΣ
  ΑΛΛΙΩΣ
    rear <- rear + 1
    ΟΥΡΑ[rear] <- ΟΝΟΜΑ
    done <- ΑΛΗΘΗΣ
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
ΔΙΑΔΙΚΑΣΙΑ ΕΞΑΓΩΓΗ(ΟΥΡΑ, ΟΝΟΜΑ, front, rear, done)
ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: ΟΥΡΑ[30], ΟΝΟΜΑ
  ΑΚΕΡΑΙΕΣ: front, rear
  ΛΟΓΙΚΕΣ: done
ΑΡΧΗ
  ΑΝ (front = 0 ΚΑΙ rear = 0) ΤΟΤΕ
    done <- ΨΕΥΔΗΣ
  ΑΛΛΙΩΣ_ΑΝ (front = rear) ΤΟΤΕ
    ΟΝΟΜΑ <- ΟΥΡΑ[front]
    front <- 0
    rear <- 0
    done <- ΑΛΗΘΗΣ
  ΑΛΛΙΩΣ
    ΟΝΟΜΑ <- ΟΥΡΑ[front]
    front <- front + 1
    done <- ΑΛΗΘΗΣ
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Αναζήτηση

Αν και υπάρχουν αρκετοί αλγόριθμοι αναζήτησης ενός στοιχείου σε πίνακα, στην ύλη του μαθήματος συμπεριλαμβάνονται μόνο δύο :

• **Γραμμική ή Σειριακή (linear ή sequential).** Η πιο απλή περίπτωση, προσπελαύνουμε το πρώτο στοιχείο, μετά το δεύτερο κοκ μέχρι να βρούμε αυτό που αναζητούμε ή να τελειώσουν τα στοιχεία. Πρέπει να χρησιμοποιείται όταν:

- ο πίνακας είναι μη ταξινομημένος
- μικρού μεγέθους(<21) και
- η αναζήτηση στον πίνακα γίνεται σπάνια. (παράγραφος 3.6 σχολ. βιβλίου)

```
ΠΡΟΓΡΑΜΜΑ Σειριακή
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: i, Α[10], αριθμός
  ΛΟΓΙΚΕΣ: βρέθηκε
ΑΡΧΗ
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    ΓΡΑΨΕ 'ΔΩΣΕ ΣΤΟΙΧΕΙΟ ', i
    ΔΙΑΒΑΣΕ Α[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ 'ΔΩΣΕ ΑΡΙΘΜΟ: '
  ΔΙΑΒΑΣΕ αριθμός
  βρέθηκε <- ΨΕΥΔΗΣ
  i <- 1
  ΟΣΟ (βρέθηκε = ΨΕΥΔΗΣ) ΚΑΙ (i <= 10) ΕΠΑΝΑΛΑΒΕ
    ΑΝ (Α[i] = αριθμός) ΤΟΤΕ
      βρέθηκε <- ΑΛΗΘΗΣ
    ΑΛΛΙΩΣ
      i <- i + 1
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΑΝ (βρέθηκε = ΑΛΗΘΗΣ) ΤΟΤΕ
    ΓΡΑΨΕ 'Το στοιχείο βρέθηκε στον πίνακα'
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Το στοιχείο δεν βρέθηκε στον πίνακα'
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

• **Δυαδική (Binary)**. Για να εφαρμόσουμε τη δυαδική μέθοδο αναζήτησης σε έναν πίνακα θα πρέπει υποχρεωτικά ο πίνακας να είναι ταξινομημένος. Η βασική ιδέα της δυαδικής αναζήτησης είναι ότι σε κάθε βήμα της αποκλείουμε ολόκληρα τμήματα του πίνακα (στα οποία δεν είναι δυνατόν να υπάρχει το στοιχείο που ψάχνουμε αφού ο πίνακές είναι ταξινομημένος). Όταν θέλουμε να εξετάσουμε αν κάποιο στοιχείο του πίνακα έχει μια συγκεκριμένη τιμή, τότε μπορεί να συμβαίνει ένα από τα παρακάτω :

- Το στοιχείο που ζητάμε να βρίσκεται στη μεσαία θέση του πίνακα
- Το στοιχείο που ζητάμε να βρίσκεται στο πρώτο μισό του πίνακα.
- Το στοιχείο που ζητάμε να βρίσκεται στο δεύτερο μισό του πίνακα.
- Το στοιχείο που ζητάμε να μην υπάρχει καθόλου στον πίνακα.

Αρχικά εξετάζουμε το μεσαίο στοιχείο του πίνακα .Δεδομένου ότι ο πίνακας που εξετάζουμε είναι ταξινομημένος κατά αύξουσα σειρά, αν το στοιχείο που ζητάμε είναι μεγαλύτερο από το μεσαίο στοιχείο τότε μπορούμε να πούμε με σιγουριά ότι αν υπάρχει το στοιχείο που ζητάμε βρίσκεται στο αριστερό κομμάτι του πίνακα, ενώ αν είναι μικρότερο στο δεξί κομμάτι του. Συνεχίζοντας την ίδια διαδικασία για τον 'υποπίνακα' που προκύπτει καταλήγουμε μετά από κάποιο αριθμό βημάτων στο τελικό αποτέλεσμα. Όσον αφορά τον αλγόριθμο αυτό επιτυγχάνεται με τη χρήση τριών μεταβλητών L, R, M που μας βοηθούν να κάνουμε τους ελέγχους που προαναφέραμε. Στην μεταβλητή L καταχωρείται ο δείκτης του αριστερότερου στοιχείου του προς εξέταση πίνακα, στο R το δεξιότερο ενώ στο M καταχωρείται ο δείκτης του μεσαίου στοιχείου του πίνακα (Συγκεκριμένα η τιμή του M προκύπτει από τις αντίστοιχες τιμές των L και R με τον ακόλουθο τρόπο (L+R)div2). Ακολουθεί ο αλγόριθμος της δυαδικής αναζήτησης. Όπου x είναι η ζητούμενη τιμή (η τιμή που ψάχνουμε αν υπάρχει στον πίνακα).

Παράδειγμα

Δίνεται ο πίνακας

1	2	5	8	9	15	22	27	35	37	38	40	43	45	47

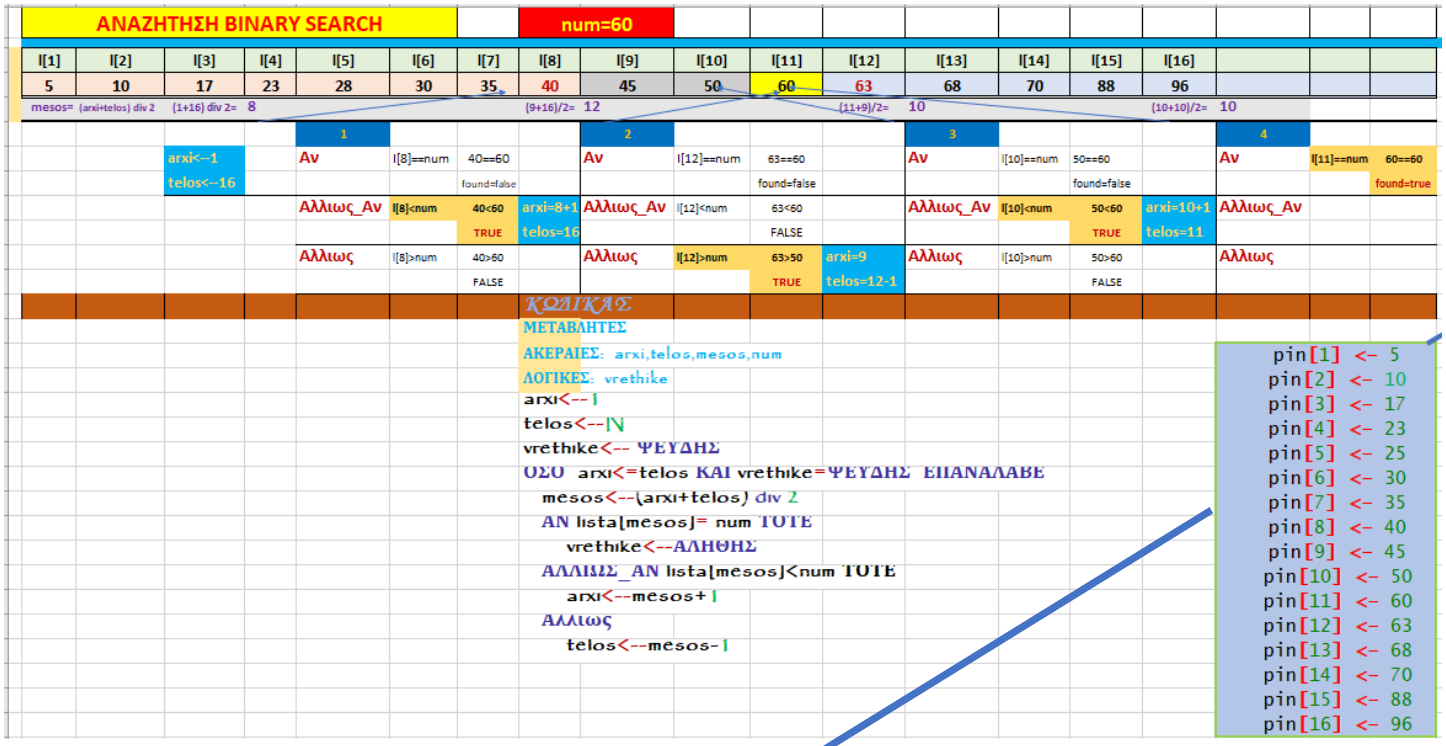
Αναζήτηση του στοιχείου 38 (υπάρχει στον πίνακα)

Βήμα 1	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 2	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 3	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 4	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47

Με κίτρινο σημειώνεται το στοιχείο του πίνακα που εξετάζεται (στο μέσον)
Με πράσινο σημειώνεται το τμήμα του πίνακα που απομένει για αναζήτηση
Με κόκκινο σημειώνεται το τμήμα του πίνακα που έχει αποκλειστεί
(Τα ίδια χρώματα χρησιμοποιούνται και στο επόμενο παράδειγμα)

Αναζήτηση του στοιχείου 39 (δεν υπάρχει στον πίνακα)

Βήμα 1	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 2	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 3	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 4	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47
Βήμα 5	1	2	5	8	9	15	22	27	35	37	38	40	43	45	47




```
ΚΩΔΙΚΑΣ
ΠΡΟΓΡΑΜΜΑ binary
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: arxi, telos, mesos, num, thesi, pin[16], step
  ΛΟΓΙΚΕΣ: vrethike
ΑΡΧΗ
  ΓΡΑΨΕ 'Δώσε τιμή:'
  ΔΙΑΒΑΣΕ num

  arxi <- 1
  telos <- 16
  vrethike <- ΨΕΥΔΗΣ
  step <- 0
  ΟΣΟ arxi <= telos ΚΑΙ vrethike = ΨΕΥΔΗΣ ΕΠΑΝΑΛΑΒΕ
    step <- step + 1
    ΓΡΑΨΕ 'step=', step
    mesos <- (arxi + telos) div 2
    ΓΡΑΨΕ 'mesos=', mesos
    ΑΝ pin[mesos] = num ΤΟΤΕ
      vrethike <- ΑΛΗΘΗΣ
      thesi <- mesos
    ΑΛΛΙΩΣ_ΑΝ pin[mesos] < num ΤΟΤΕ
      arxi <- mesos + 1
    ΑΛΛΙΩΣ
      telos <- mesos - 1
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΑΝ vrethike ΤΟΤΕ
    ΓΡΑΨΕ 'pin[', thesi, ']=', pin[thesi]
    ΓΡΑΨΕ step
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Δεν βρέθηκε!!'
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
ΠΡΟΓΡΑΜΜΑ binary
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: arxi, telos, mesos, num, i, thesi, pin[16], step
  ΛΟΓΙΚΕΣ: vrethike
ΑΡΧΗ
  ΓΡΑΨΕ 'Δώσε τιμή:'
  ΔΙΑΒΑΣΕ num
  pin[1] <- 5
  pin[2] <- 10
  pin[3] <- 17
  pin[4] <- 23
  pin[5] <- 25
  pin[6] <- 30
  pin[7] <- 35
  pin[8] <- 40
  pin[9] <- 45
  pin[10] <- 50
  pin[11] <- 60
  pin[12] <- 63
  pin[13] <- 68
  pin[14] <- 70
  pin[15] <- 88
  pin[16] <- 96
  arxi <- 1
  telos <- 16
  vrethike <- ΨΕΥΔΗΣ
  step <- 0
  ΟΣΟ arxi <= telos ΚΑΙ vrethike = ΨΕΥΔΗΣ ΕΠΑΝΑΛΑΒΕ
    step <- step + 1
    ΓΡΑΨΕ 'step=', step
    mesos <- (arxi + telos) div 2
    ΓΡΑΨΕ 'mesos=', mesos
    ΑΝ pin[mesos] = num ΤΟΤΕ
      vrethike <- ΑΛΗΘΗΣ
      thesi <- mesos
    ΑΛΛΙΩΣ_ΑΝ pin[mesos] < num ΤΟΤΕ
      arxi <- mesos + 1
    ΑΛΛΙΩΣ
      telos <- mesos - 1
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΑΝ vrethike ΤΟΤΕ
    ΓΡΑΨΕ 'pin[', thesi, ']=', pin[thesi]
    ΓΡΑΨΕ step
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Δεν βρέθηκε!!'
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
ΠΡΟΓΡΑΜΜΑ δυαδική_αναζήτηση
! α) Να γραφτεί πρόγραμμα το οποίο να διαβάζει 10 ονόματα
!    και τηλέφωνα
! β) Να εμφανίζει τα ονόματα και τα τηλέφωνα αλφαβητικά
! γ) Να διαβάζει ένα όνομα για αναζήτηση του τηλεφώνου του.
! δ) Να αναζητά το όνομα σειριακά.  Αν βρεθεί να εμφανίζει
!    το τηλέφωνό του διαφορετικά μήνυμα ότι δεν βρέθηκε.
!    Να εμφανίζει από το πλήθος των αναζητήσεων που έκανε.
!    (Επειδή ο πίνακας είναι ταξινομημένος να σταματά όταν ξεπεράσει
!    το ζητούμενο όνομα)
! ε) Να αναζητά το όνομα δυαδικά.  Αν βρεθεί να εμφανίζει
!    το τηλέφωνό του διαφορετικά μήνυμα ότι δεν βρέθηκε.
!    Να εμφανίζει από το πλήθος των αναζητήσεων που έκανε.

ΣΤΑΘΕΡΕΣ
  N = 10

ΜΕΤΑΒΛΗΤΕΣ
  ΧΑΡΑΚΤΗΡΕΣ: Ον[N], Τηλ[N], temp, ζητούμενο
  ΑΚΕΡΑΙΕΣ: i, j, αρχικό, τελικό, μεσαίο, θέση, βήματα
  ΛΟΓΙΚΕΣ: τελείωσε, βρέθηκε

ΑΡΧΗ
! α. διάβασμα πινάκων
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ N
    ΓΡΑΨΕ i, « Δώστε όνομα και τηλέφωνο : «
    ΔΙΑΒΑΣΕ Ον[i], Τηλ[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

! β. ταξινόμηση πινάκων
  ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ N
    ΓΙΑ j ΑΠΟ N ΜΕΧΡΙ i ΜΕ ΒΗΜΑ -1
      ΑΝ Ον[j – 1] > Ον[j] ΤΟΤΕ
        temp ← Ον[j – 1]
        Ον[j – 1] ← Ον[j]
        Ον[j] ← temp

        temp ← Τηλ[j – 1]
        Τηλ[j – 1] ← Τηλ[j]
        Τηλ[j] ← temp
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

! εμφάνιση πίνακα
```

```
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ Ν
      ΓΡΑΨΕ Ον[i], Τηλ[i]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

! γ.
  ΓΡΑΨΕ "Δώστε όνομα για αναζήτηση"
  ΔΙΑΒΑΣΕ ζητούμενο

!δ. σειριακή αναζήτηση
  i <-- 1
  τελείωσε <-- ΨΕΥΔΗΣ
  βρέθηκε <-- ΨΕΥΔΗΣ
  βήματα <-- 0

  ΟΣΟ τελείωσε = ΨΕΥΔΗΣ ΚΑΙ i <= Ν ΕΠΑΝΑΛΑΒΕ
    βήματα <-- βήματα + 1
    ΑΝ Ον[i] = ζητούμενο ΤΟΤΕ
      τελείωσε <-- ΑΛΗΘΗΣ
      βρέθηκε <-- ΑΛΗΘΗΣ
      θέση <-- i
    ΑΛΛΙΩΣ
      ΑΝ Ον[i] > ζητούμενο ΤΟΤΕ
        τελείωσε <-- ΑΛΗΘΗΣ
      ΑΛΛΙΩΣ
        i <-- i + 1
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΓΡΑΨΕ βήματα, "αναζητήσεις"
  ΑΝ βρέθηκε = ΑΛΗΘΗΣ ΤΟΤΕ
    ΓΡΑΨΕ Ον[θέση], Τηλ[θέση]
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ "Δεν βρέθηκε"
  ΤΕΛΟΣ_ΑΝ


! ε. δυαδική αναζήτηση
  αρχικό <-- 1
  τελικό <-- Ν
  τελείωσε <-- ΨΕΥΔΗΣ
  βρέθηκε <-- ΨΕΥΔΗΣ
  βήματα <-- 0
  ΟΣΟ τελείωσε = ΨΕΥΔΗΣ ΚΑΙ αρχικό <= τελικό ΕΠΑΝΑΛΑΒΕ
    βήματα <-- βήματα + 1
    μεσαίο <-- (αρχικό + τελικό) div 2
```

```
    ΑΝ Ον[μεσαίο] = ζητούμενο ΤΟΤΕ
      τελείωσε <-- ΑΛΗΘΗΣ
      βρέθηκε <-- ΑΛΗΘΗΣ
      θέση <-- μεσαίο
    ΑΛΛΙΩΣ
      ΑΝ Ον[μεσαίο] > ζητούμενο ΤΟΤΕ
        τελικό <-- μεσαίο - 1
      ΑΛΛΙΩΣ
        αρχικό <-- μεσαίο + 1
      ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΓΡΑΨΕ βήματα, "αναζητήσεις"
  ΑΝ βρέθηκε = ΑΛΗΘΗΣ ΤΟΤΕ
    ΓΡΑΨΕ Ον[θέση], Τηλ[θέση]
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ "Δεν βρέθηκε"
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

```
Αλγόριθμος Sequential_Search
Δεδομένα // n, table, key //
done ← ψευδής
position ← 0
i ← 1
Όσο (done=ψευδής) και (i<=n) επανάλαβε
      Αν table[i]=key τότε
          done ← αληθής
          position ← i
      αλλιώς
          i ← i+1
      Τέλος_αν
Τέλος_επανάληψης
Αποτελέσματα //done, position //
Τέλος Sequential_Search
```

```
Αλγόριθμος δυαδική_αναζήτηση
Δεδομένα // n, table, key //
L←1
R←n
D←ΨΕΥΔΗΣ
Όσο D=ΨΕΥΔΗΣ και L<=R επανάλαβε
          M←(L+R) DIV 2
          Αν table[M] = key τότε
                D← ΑΛΗΘΗΣ
          Αλλιώς_αν table[M]<key τότε
                L←M+1
          Αλλιώς
                R←M-1
          Τέλος_αν
Τέλος_επανάληψης
Αν D = ΑΛΗΘΗΣ τότε
          Εμφάνισε 'Βρέθηκε το στοιχείο στη θέση:',M
Αλλιώς
          Εμφάνισε 'Δεν βρέθηκε το στοιχείο'
Τέλος_αν
Τέλος δυαδική_αναζήτηση
```

Γενικά

Τι γίνεται αν δεν υπάρχει το υπό αναζήτηση στοιχείο στον πίνακα;

Ακόμη και στην περίπτωση που το υπό αναζήτηση στοιχείο δεν υπάρχει στον πίνακα, η σάρωση του πίνακα γίνεται πολύ γρήγορα. Για παράδειγμα σε ένα πίνακα με 200 θέσεις αν δεν υπάρχει το υπό αναζήτηση στοιχείο η σειριακή αναζήτηση θα πραγματοποιήσει 200 επαναλήψεις για να καταλήξει στο αποτέλεσμα, αντίστοιχα η δυαδική δεν θα χρειαστεί περισσότερες από 7 με 8 επαναλήψεις!

Τι γίνεται αν το υπό αναζήτηση στοιχείο είναι σε μια από τις πρώτες θέσεις του πίνακα;

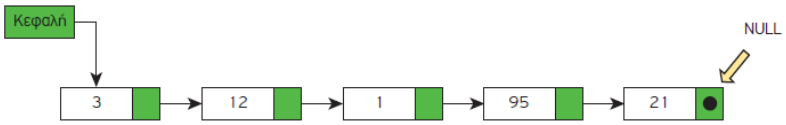
Όλοι οι κανόνες έχουν και εξαιρέσεις! Εδώ λοιπόν, αν το υπό αναζήτηση στοιχείο βρίσκεται σε μια από τις πρώτες θέσεις του πίνακα , τότε η σειριακή αναζήτηση είναι γρηγορότερη από τη δυαδική ! Είναι η μόνη περίπτωση στην οποία ¨κερδίζει¨ τη μάχη της αναζήτησης η σειριακή ! Σε όλες τις υπόλοιπες η δυαδική είναι σαφώς πιο γρήγορη, εφ όσων μιλάμε για ταξινομημένο πίνακα, μη το ξεχνάμε!

<u>Δυναμικές Δομές Δεδομένων Λίστα – Δένδρα - Γράφοι</u>

1. Τι ονομάζεται συνδεδεμένη λίστα;

Μία (απλά) **συνδεδεμένη λίστα** (linked list) είναι ένα σύνολο κόμβων διατεταγμένων γραμμικά (ο ένας μετά τον άλλο). Κάθε κόμβος περιέχει εκτός από τα **δεδομένα** του και έναν **δείκτη** που δείχνει προς τον **επόμενο** κόμβο.

Ο **δείκτης** του **τελευταίου** κόμβου δε δείχνει σε κάποιον κόμβο (δείκτης στο κενό). Για να το δηλώσουμε αυτό λέμε ότι το πεδίο δείκτη του τελευταίου κόμβου έχει την τιμή **NULL**.



2. Πώς γίνεται η **προσπέλαση** σε μια συνδεδεμένη λίστα;

Για να προσπελάσουμε τους κόμβους της λίστας χρειάζεται να γνωρίζουμε τη διεύθυνση (θέση στη μνήμη) του **πρώτου** κόμβου της λίστας. Η διεύθυνση αυτή αποθηκεύεται σε μία ειδική μεταβλητή που την ονομάζουμε συνήθως **Κεφαλή** (Head).

Οι κόμβοι μιας (απλά) συνδεδεμένης λίστας είναι διατεταγμένοι σε μια συγκεκριμένη σειρά, χωρίς αυτό να σημαίνει ότι αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη. Αντίθετα, είναι **διασκορπισμένοι σε όλη τη μνήμη** και η σύνδεση μεταξύ τους γίνεται μέσω των δεικτών.

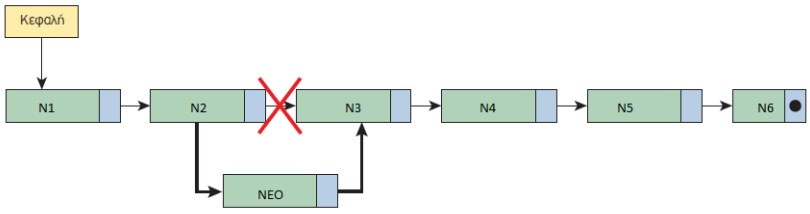
Έχουμε άμεση πρόσβαση μόνο στον πρώτο κόμβο της λίστας. Επομένως, για να εντοπίσουμε κάποιον από τους ενδιάμεσους κόμβους, πρέπει να ξεκινήσουμε από τον πρώτο κόμβο της λίστας και να ακολουθήσουμε τους δείκτες με τη σειρά, μέχρι να φτάσουμε στον επιθυμητό κόμβο.

3. Πώς **προσθέτω** κόμβο σε μια συνδεδεμένη **λίστα**;

Οι απαιτούμενες ενέργειες για την εισαγωγή (παρεμβολή) του νέου κόμβου είναι :

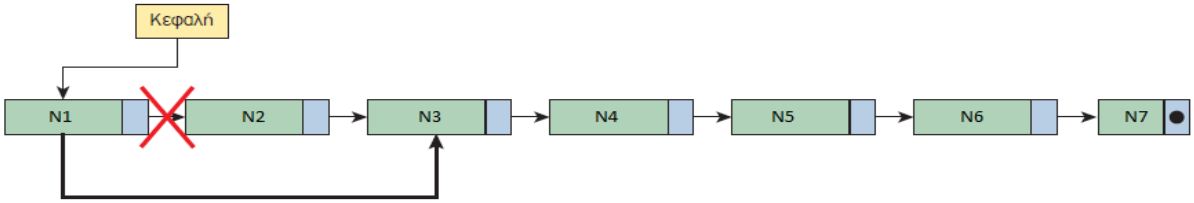
- ο δείκτης του κόμβου μετά τον οποίο θα γίνει η παρεμβολή (N2), να δείχνει τον νέο κόμβο (NEO)
- και ο δείκτης του νέου κόμβου (NEO), να δείχνει αυτό που έδειχνε ο προηγούμενος (στο N3) (δηλαδή, να πάρει την τιμή που είχε πριν την εισαγωγή ο δείκτης του N2).

Με αυτόν τον τρόπο, οι κόμβοι της λίστας διατηρούν τη λογική τους σειρά, αλλά οι φυσικές θέσεις στη μνήμη μπορεί να είναι τελείως διαφορετικές.



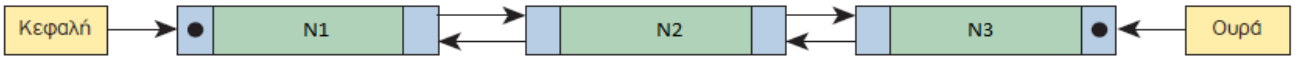
4. Πώς **διαγράφω** κόμβο από μια συνδεδεμένη **λίστα**;

Για τη διαγραφή ενός κόμβου αρκεί ν’ αλλάξει τιμή ο δείκτης του προηγούμενου κόμβου και να δείχνει πλέον στον



επόμενο αυτού που διαγράφεται. Ο κόμβος που διαγράφηκε (ο δεύτερος) αποτελεί «άχρηστο δεδομένο» και ο χώρος μνήμης που καταλάμβανε, παραχωρείται για άλλη χρήση.

5. Τι είναι η **διπλά συνδεδεμένη** λίστα (doubly linked list);
Η διπλά συνδεδεμένη λίστα είναι μια λίστα στην οποία μπορούμε να τη διατρέξουμε και προς τις δύο κατευθύνσεις. Αυτό επιτυχάνεται με τη χρήση του δεύτερου δείκτη που δείχει τον προηγούμενο κόμβο. Προσφέρει έτσι τη δυνατότητα ξεκινώντας από οποιοδήποτε κόμβο της λίστας να μπορούμε να διατρέξουμε τη λίστα και προς τις δυο κατευθύνσεις.



6. Γιατί **χρησιμοποιούνται** οι συνδεδεμένες **λίστες** στην υλοποίηση της **στοίβας** και της **ουράς**
Οι συνδεδεμένες λίστες αξιοποιούνται για την υλοποίηση της στοίβας και της ουράς, λόγω της **δυνατότητάς αυξομείωσης του μεγέθους τους**.

7. Πώς μια **στοίβα** μπορεί να υλοποιηθεί με μία απλά συνδεδεμένη λίστα;
Οι κόμβοι (στοιχεία) εισέρχονται από την αρχή της απλά συνδεδεμένης λίστας και εξέρχονται πάλι από την αρχή της λίστας τότε ο κόμβος (στοιχείο) που εισήλθε τελευταίος εξέρχεται και πρώτος (LIFO). Επομένως η στοίβα μπορεί να υλοποιηθεί με μία απλά συνδεδεμένη λίστα αφού μπορεί να γίνει **εισαγωγή κόμβου (στοιχείου) στην αρχή της λίστας και διαγραφή κόμβου (στοιχείου) από την αρχή της λίστας** .

8. **Πότε** θα χρησιμοποιήσουμε **λίστα** και πότε **πίνακα** για την υλοποίηση μιας **στοίβας**;
Καλύτερη είναι η υλοποίηση στοίβας με συνδεδεμένη λίστα εκτός αν δεν **γνωρίζουμε** εκ των προτέρων τον **μέγιστο αριθμό** στοιχείων και θέλουμε μια εύκολη και γρήγορη λύση, οπότε επιλέγουμε την υλοποίηση με πίνακα.

9. Πώς μια **ουρά** μπορεί να υλοποιηθεί με μία **διπλά συνδεδεμένη** λίστα.
Στην ουρά έχουμε τις εξής δύο κύριες λειτουργίες. Εισαγωγή στοιχείου στο πίσω άκρο της ουράς και εξαγωγή στοιχείου από το εμπρός άκρο της ουράς. Η ουρά μπορεί να υλοποιηθεί με μία διπλά συνδεδεμένη λίστα, αφού μπορούμε να κάνουμε **εισαγωγή** κόμβου στο **τέλος της διπλά συνδεδεμένης λίστας**. Επίσης μπορούμε να κάνουμε **διαγραφή** κόμβου στην **αρχή** της διπλά συνδεδεμένης λίστας (εξαγωγή στοιχείου από το εμπρός άκρο της ουράς).

10. Ποιες οι διαφορές ανάμεσα σε λίστες και πίνακες;
- Ο **πίνακας** θεωρείται μια δομή **τυχαίας προσπέλασης**, σε αντίθεση με μια **λίστα** που είναι στην ουσία μια δομή ακολουθιακής ή **σειριακής** προσπέλασης. Για να φθάσουμε, δηλαδή, σ' έναν κόμβο μιας λίστας πρέπει να περάσουμε από όλους τους προηγούμενους ξεκινώντας από τον πρώτο.
- Ο **πίνακας** έχει **σταθερό** μέγεθος, το οποίο δηλώνεται εξαρχής κατά την υλοποίηση. Αυτό γίνεται, διότι ο πίνακας είναι στατική δομή δεδομένων σε αντίθεση με τη **λίστα** που είναι **δυναμική** δομή και το μέγεθός της μπορεί να μεταβάλλεται καθώς εισέρχονται νέοι κόμβοι στη λίστα ή διαγράφονται κάποιοι άλλοι.
- Οι κόμβοι της **λίστας** αποθηκεύονται σε **μη συνεχόμενες θέσεις** μνήμης σε αντιδιαστολή με τους **πίνακες**, όπου τα στοιχεία αποθηκεύονται σε **συνεχόμενες** θέσεις μνήμης.

11. Ποια είναι τα πλεονεκτήματα των λιστών (έναντι των πινάκων) ;
- Το **δυναμικό** τους μέγεθος,
- η ευκολία **εισαγωγής** και **διαγραφής** από οποιοδήποτε μέρος της λίστας, καθώς και
- η **μη αναγκαιότητα δήλωσης του μεγέθους** τους.

12. Ποια τα μειονεκτήματα των λιστών (έναντι των πινάκων);

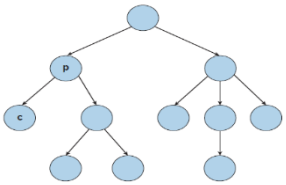
- Η **τυχαία πρόσβαση** στη λίστα **δεν επιτρέπεται**. Είναι αδύνατο να φτάσετε στον n-οστό κόμβο μιας απλά συνδεδεμένης λίστας χωρίς πρώτα να περάσετε από όλους τους κόμβους διαδοχικά μέχρι τον συγκεκριμένο κόμβο ξεκινώντας από τον πρώτο κόμβο. Εναλλακτικά, στην περίπτωση της διπλά συνδεμένης λίστας μπορείτε να ξεκινήσετε και από τον τελευταίο κόμβο. Επομένως, **δεν μπορούμε** να πραγματοποιήσουμε με αποτελεσματικό τρόπο **δυαδική αναζήτηση** σε συνδεδεμένες λίστες.
- Οι συνδεδεμένες λίστες έχουν **πολύ μεγαλύτερη επιβάρυνση** από τους πίνακες, αφού οι συνδεδεμένοι κόμβοι της λίστας είναι δυναμικά κατανεμημένοι (οι οποίοι είναι λιγότερο αποτελεσματικοί στη **χρήση της μνήμης**) και κάθε κόμβος στη λίστα πρέπει, επιπλέον, να **αποθηκεύσει έναν πρόσθετο δείκτη** που θα δείχνει στον επόμενο κόμβο. Στην περίπτωση των διπλά συνδεδεμένων λιστών χρειαζόμαστε επιπλέον έναν δεύτερο δείκτη που θα δείχνει στον προηγούμενο κόμβο.

13. Ποιες είναι οι βασικές πράξεις των συνδεδεμένων λιστών;

- **Εισαγωγή** κόμβου στη λίστα (εισαγωγή κόμβου στην αρχή, στο τέλος της λίστας ή ενδιάμεσα).
- **Διαγραφή** κόμβου από τη λίστα (διαγραφή από την αρχή, το τέλος της λίστας ή ενδιάμεσα).
- **Έλεγχος** για το αν η λίστα είναι **κενή**.
- **Αναζήτηση** κόμβου για την εύρεση συγκεκριμένου στοιχείου.
- **Διάσχιση** της λίστας και προσπέλαση των στοιχείων της (π.χ. εκτύπωση των δεδομένων που περιέχονται σε όλους τους κόμβους της λίστας).

14. Τι ονομάζεται δέντρο;

Ένα **δένδρο** (tree) είναι μία **δομή** που αποτελείται από ένα **σύνολο κόμβων** και ένα **σύνολο ακμών** μεταξύ των κόμβων με βάση τους εξής **κανόνες**:



- Υπάρχει ένας ξεχωριστός κόμβος που ονομάζεται **ρίζα**. Αυτός είναι ένας κόμβος **χωρίς γονέα**.
- Για κάθε κόμβο c, εκτός από τη ρίζα, υπάρχει **μόνο μια ακμή** που καταλήγει στον κόμβο αυτόν ξεκινώντας από κάποιον άλλον κόμβο p. Ο κόμβος **p ονομάζεται γονέας** του c και ο κόμβος **c παιδί** του p.
- Για κάθε κόμβο υπάρχει μία **μοναδική διαδρομή**, δηλαδή, μια ακολουθία διαδοχικών ακμών, που ξεκινάει από τη **ρίζα** και τερματίζει σε αυτόν τον **κόμβο**.

Δένδρο θεωρούμε και το κενό δένδρο, δηλαδή το δένδρο που δεν έχει ούτε κόμβους, ούτε ακμές. Το κενό δένδρο είναι το μόνο δένδρο χωρίς ρίζα.

15. Σε ένα δέντρο ποιοι κόμβοι ονομάζονται φύλλα και ποιοι αδέλφια;

Οι κόμβοι χωρίς παιδιά ονομάζονται «**φύλλα**»

Κόμβοι με τον ίδιο γονέα ονομάζονται «**αδέλφια**».

16. Ποια δέντρα ονομάζονται διατεταγμένα;

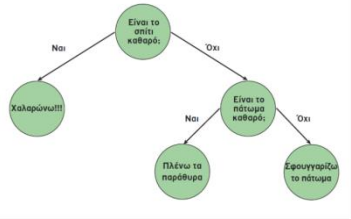
Τα δέντρα στα οποία για κάθε κόμβο του υπάρχει μία **γραμμική σχέση μεταξύ των παιδιών του κόμβου αυτού**, αναφερόμαστε σε ένα διατεταγμένο δένδρο.

17. Σε ποιους **τομείς** της επιστήμης χρησιμοποιούνται τα δέντρα;
Τα δένδρα είναι μία μη-γραμμική ευέλικτη δομή δεδομένων που χρησιμοποιούνται σε πολλούς τομείς της επιστήμης των υπολογιστών, συμπεριλαμβανομένων των λειτουργικών συστημάτων, των γραφικών, των συστημάτων βάσεων δεδομένων, των παιχνιδιών, της τεχνητής νοημοσύνης και της δικτύωσης υπολογιστών.

18. Δώστε μερικά **παραδείγματα** χρήσης δέντρων.
- το σύστημα αρχείων του υπολογιστή
- το οικογενειακό δένδρο
- δομή ενός οργανισμού
- πίνακας περιεχομένων ενός βιβλίου
- μέρη που απαρτίζουν την μηχανή ενός αυτοκινήτου

19. Τι ονομάζεται δένδρο του παιχνιδιού (game tree);
Στα παιχνίδια ο υπολογιστής χρησιμοποιεί ένα ειδικό δένδρο, που ονομάζεται **δένδρο του παιχνιδιού** (game tree), το οποίο **μοντελοποιεί όλες τις πιθανές κινήσεις των παικτών** για να νικήσει. Κάθε **κόμβος** στο δένδρο, αντιπροσωπεύει μία συγκεκριμένη κατάσταση παιχνιδιού και περιέχει **πληροφορίες** σχετικά με το ποιος παίκτης έχει τη **μεγαλύτερη πιθανότητα** να κερδίσει από οποιαδήποτε πιθανή **κίνηση**.

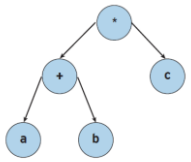
20. Τι ονομάζεται **δένδρο απόφασης**; Δώστε παράδειγμα.
Τα δένδρα απόφασης, είναι δένδρα στα οποία :



- κάθε κόμβος αντιπροσωπεύει ένα χαρακτηριστικό (ιδιότητα),
- κάθε ακμή αντιπροσωπεύει μια απόφαση (κανόνα) και
- κάθε φύλλο αντιπροσωπεύει ένα αποτέλεσμα.

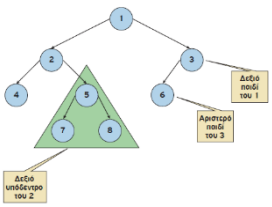
Ένα απλό πρόβλημα λήψης απόφασης μπορεί να είναι αυτό της καθαριότητας του φοιτητικού σπιτιού σας.

21. Πώς χρησιμοποιούνται τα δέντρα στον **υπολογισμό αριθμητικών εκφράσεων.**
Για τον υπολογισμό αριθμητικών εκφράσεων στα φύλλα του δέντρου μπαίνουν οι τελεστέοι, ενώ στους εσωτερικούς κόμβους οι τελεστές.



Εικόνα 1.3.20. (a+b)*c

22. Για ποιους λόγους τα δέντρα θεωρούνται ισχυρές δομές.
- Ο πρώτος λόγος αναφέρεται στη **δυναμικότητα** των δένδρων. Είναι πολύ εύκολο να **προσθέσετε**, να **αφαιρέσετε** ή να αναζητήσετε ένα στοιχείο σε ένα δένδρο.
- Ο δεύτερος βασικός λόγος είναι ότι η **δομή** των δένδρων μεταφέρει **πληροφορίες**.

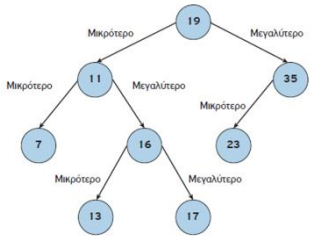


23. Τι ονομάζεται δυαδικό δέντρο;
Ένα δυαδικό δένδρο (binary tree) είναι ένα **διατεταγμένο** δένδρο, στο οποίο κάθε κόμβος έχει **το πολύ δύο** παιδιά, το αριστερό και το δεξί παιδί.

Μπορούμε, συνεπώς, να μιλάμε για αριστερό και δεξιό υποδένδρο ενός κόμβου.

24. Τι ονομάζεται δυαδικό δέντρο αναζήτησης;

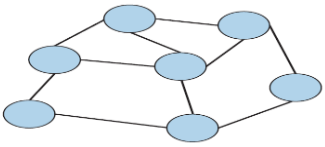
Ένα δυαδικό δένδρο αναζήτησης (binary search tree) είναι ένα **δυαδικό δένδρο**, όπου για **κάθε κόμβο** u, **όλοι οι κόμβοι του αριστερού υποδένδρου** έχουν τιμές **μικρότερες** της τιμής του κόμβου **u** και όλοι οι κόμβοι του **δεξιού** υποδένδρου έχουν τιμές **μεγαλύτερες** (ή ίσες) της τιμής του κόμβου u.



25. Τι ονομάζεται γράφος

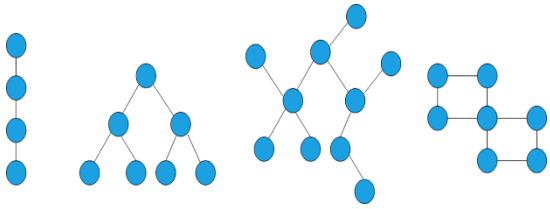
Ένας γράφος (graph) είναι μία δομή που αποτελείται :

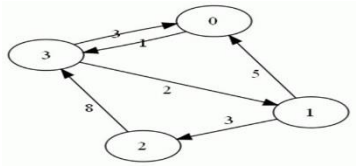
- από ένα **σύνολο κόμβων** (ή σημείων ή κορυφών)
- και ένα σύνολο **γραμμών** (ή ακμών ή τόξων) που **ενώνουν** μερικούς ή όλους τους **κόμβους**.



26. Ποια η συσχέτιση του γράφου με τις λίστες και τα δέντρα;

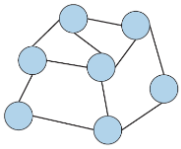
Ο γράφος αποτελεί την πιο γενική δομή δεδομένων, με την έννοια ότι **όλες οι προηγούμενες δομές** που παρουσιάστηκαν μπορούν να θεωρηθούν περιπτώσεις **γράφων**.



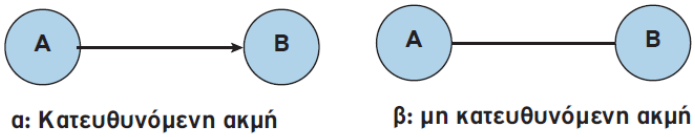


27. Ποιοι οι τύποι του γράφου;

- Εάν όλες οι **ακμές** σε έναν γράφο έχουν **κατεύθυνση**, ο γράφος ονομάζεται **κατευθυνόμενος γράφος** (directed graph).



- Εάν όλες οι ακμές σε έναν γράφο δεν έχουν κατεύθυνση, ο γράφος ονομάζεται **μη κατευθυνόμενος** γράφος (undirected graph) και η διαδρομή μεταξύ των δύο κόμβων είναι αμφίδρομη.

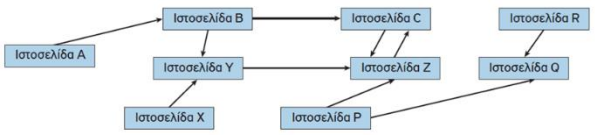


α: Κατευθυνόμενη ακμή **β: μη κατευθυνόμενη ακμή**

28. Τι τύπου γράφοι είναι τα κοινωνικά δίκτυα facebook, twitter και τι ο παγκόσμιος ιστός;

Στον παγκόσμιος Ιστός (WWW), που είναι ένας τεράστιος γράφος,

- έχουμε κάποιες φορές ακμές που είναι :μη κατευθυνόμενες - μπορούμε να προχωρήσουμε από μια ιστοσελίδα σε μια άλλη και το αντίστροφο
- και άλλες φορές έχουμε ακμές που κατευθύνονται - μπορούμε να πάμε μόνο από την ιστοσελίδα Α στην ιστοσελίδα Β, και ποτέ το αντίστροφο.φορές



Το Facebook είναι **μη κατευθυνόμενος γράφος** επειδή η σχέση μεταξύ δύο χρηστών (κόμβοι), είναι αμφίδρομη. Δεν υπάρχει η έννοια κόμβου «προέλευσης» και «προορισμού» - αντ' αυτού, είσαι φίλος μου και είμαι δικός σου.

Το Twitter είναι **κατευθυνόμενος γράφος** επειδή μπορώ να σε ακολουθήσω, αλλά δεν απαιτείται να με ακολουθήσεις. Κάθε ακμή που δημιουργούμε αντιπροσωπεύει μια μονόδρομη σχέση.

- Όταν **με ακολουθείς** στο Twitter, δημιουργείται μια ακμή στον γράφο με τον λογαριασμό σου ως κόμβο προέλευσης και τον λογαριασμό μου ως τον κόμβο προορισμού.
- Όταν **σε ακολουθώ** στο Twitter, δημιουργώ μια δεύτερη ακμή, με τον λογαριασμό μου σαν κόμβο προέλευσης και τον δικό σου σαν κόμβο προορισμού

Οι διαφορες του Αλγοριθμου με το Προγραμμα

- Στον Αλγόριθμο ξεκινούμε με:
  Αλγόριθμος <Όνομα_αλγορίθμου>
  και τελειώνουμε με:
  Τέλος <Όνομα_αλγορίθμου>

  Στο Πρόγραμμα ξεκινούμε με:
  ΠΡΟΓΡΑΜΜΑ <Όνομα_προγράμματος>
  και τελειώνουμε με:
  ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
  ή
  ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ <Όνομα_προγράμματος>

- Στο Πρόγραμμα γράφουμε με κεφαλαία γράμματα. Στον Αλγόριθμο αντίθετα, όλα είναι μικρά εκτός από τον πρώτο χαρακτήρα της κάθε γραμμής(εξαιρείται το **αλλιώς** και το **αλλιώς_αν**).

- Στον Αλγόριθμο χρησιμοποιούμε: **με_βήμα** (με κάτω παύλα)
  ενώ στο Πρόγραμμα: **ΜΕ ΒΗΜΑ** (χωρίς κάτω παύλα)

- Στον Αλγόριθμο έχουμε τους συμβολισμούς: ≤ , ≥ και ≠
  στο Πρόγραμμα οι αντίστοιχοι είναι: <=, >= και <>

- Στον Αλγόριθμο έχουμε τις εντολές: **Εμφάνισε**, **Εκτύπωσε** κ.τ.λ.
  ενώ στο Πρόγραμμα μόνο: **ΓΡΑΨΕ**

- Στο Πρόγραμμα δηλώνουμε πάντοτε τις μεταβλητές, στον Αλγόριθμο δεν είναι απαραίτητο. Όταν η εκφώνηση της άσκησης αναφέρει την ανάγνωση των δεδομένων από το χρήστη τότε χρησιμοποιείται η εντολή **Διάβασε**. Σε διαφορετική περίπτωση, π.χ. όταν η εκφώνηση αναφέρει "*έστω μεταβλητή Ν που περιέχει το πλήθος*", εισάγουμε την τιμή της μεταβλητής Ν στον αλγόριθμό με την εντολή **Δεδομένα**.

- Στο Πρόγραμμα δεν χρησιμοποιούμε ποτέ την εντολή **αντιμετάθεσε** αλλά κάνουμε αντιμετάθεση με χρήση βοηθητικής μεταβλητής.

  temp← A[j-1]
  A[j-1]← A[j]
  A[j]← temp
  } **αντιμετάθεσε** A[j],A[j-1]

  Στον Αλγόριθμο μπορούμε να βάλουμε και τα δύο.

- Στο Πρόγραμμα κάθε εντολή γράφεται σε ξεχωριστή γραμμή. Αν μία εντολή πρέπει να συνεχιστεί και στην επόμενη γραμμή, τότε ο πρώτος χαρακτήρας αυτής της γραμμής πρέπει να είναι ο χαρακτήρας **&**.

- Στο Πρόγραμμα αν ο πρώτος χαρακτήρας είναι θαυμαστικό(!) τότε αυτή η εντολή περιέχει σχόλια και όχι εκτελέσιμες εντολές.

- Στο Πρόγραμμα πριν από κάθε εντολή **ΔΙΑΒΑΣΕ**, δίνουμε μια εντολή **ΓΡΑΨΕ** που επεξηγεί τι πρόκειται να διαβάσει.