ΠΛΗ20

ΕΝΟΤΗΤΑ 0: ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ

Μάθημα 0.4:
Αναδρομικοί και Επαναληπτικοί Αλγόριθμοι

Δημήτρης Ψούνης

ΠΕΡΙΕΧΟΜΕΝΑ

Α. Σκοπός του Μαθήματος

Επίπεδο Α

- (-)

Επίπεδο Β

- Στοιχειώδης κατανόηση ανάγνωσης ενός προγράμματος σε ψευδογλώσσα.

Επίπεδο Γ

- Γνώση του τρόπου εκτέλεσης μίας αναδρομικής διαδικασίας.

Β. Θεωρία

1. Αλγόριθμος

1. Ορισμός Αλγορίθμου

- Αλγόριθμος είναι ένας επιλύτης ενός προβλήματος. Είναι μία πεπερασμένη ακολουθίας σαφώς καθορισμένων βημάτων που παίρνει ως είσοδο ένα στιγμιότυπο του προβλήματος και παράγει την λύση του στιγμιοτύπου

- Ένα πρόβλημα μπορεί να λύνεται από διαφορετικούς αλγόριθμους. Για παράδειγμα:
  - Το πρόβλημα της εύρεσης ενός στοιχείου σε έναν πίνακα, λύνεται με την σειριακή αναζήτηση, αλλά και την δυαδική αναζήτηση.
  - Το πρόβλημα της ταξινόμησης μιας ακολουθίας αριθμών, λύνεται με τον αλγόριθμο φυσαλίδας, τον αλγόριθμο γρήγορης ταξινόμησης κ.α.
  - Το πρόβλημα της έυρεσης του Μ.Κ.Δ. ενός αριθμού λύνεται π.χ. με τον αλγόριθμο του Ευκλείδη.

Β. Θεωρία
1. Αλγόριθμος
1. Ορισμός Αλγορίθμου

- Παράδειγμα: Ο ακόλουθος αλγόριθμος υπολογίζει τον μέγιστο μιας ακολουθίας αριθμών

Αλγόριθμος Εύρεσης Μεγίστου Ακολουθίας
Είσοδος: Ακολουθία Αριθμών $A=a_1,a_2,\ldots,a_n$
Έξοδος: Ο μέγιστος αριθμός της ακολουθίας A

```
-----------------------------------------------------
procedure maximum(A)
  m=a_1
  for i=2 to n do
    if (a_i>m) then
      m=a_i
    end if
  end for
  return m
end maximum
```

Β. Θεωρία

1. Αλγόριθμος

1. Ορισμός Αλγορίθμου

Οι έννοιες πρόβλημα, αλγόριθμος και στιγμιότυπο στο παράδειγμα που κατασκευάσαμε

- **Το πρόβλημα**: Να βρεθεί το μέγιστο μιας ακολουθίας αριθμών

- **Ο αλγόριθμος**: Διέτρεξε την ακολουθία διατηρώντας στην μνήμη τον τρέχων μέγιστο αριθμό.
  - Η διατύπωση του αλγορίθμου γίνεται στην **ψευδογλώσσα**

- **Το στιγμιότυπο.** Π.χ.
  - [1,2,8,4,3,0]
  - [6,3,10,4,2,8,11]
  - [-12,44,1002,9499,994]

Β. Θεωρία
1. Αλγόριθμος
2. Χαρακτηριστικά Αλγορίθμου

Κατά D.Knuth (The Art of Computer Programming) ένας αλγόριθμος πρέπει να χαρακτηρίζεται από τα εξής:

- **Ακρίβεια:** Τα βήματα πρέπει να είναι σαφή.
- **Μοναδικότητα**: Τα ενδιάμεσα αποτελέσματα είναι μοναδικά για κάθε είσοδο (δεν μπορεί να προκύψουν διαφορετικά ενδιάμεσα αποτελέσματα για την ίδια είσοδο)
- **Αριθμός Βημάτων**: Πρέπει να είναι πεπερασμένα. Δηλαδή πρέπει ο αλγόριθμος κάποια στιγμή να τελειώνει, μετά από πεπερασμένο αριθμό βημάτων.
- **Γενικότητα**: Ο αλγόριθμος πρέπει να λειτουργεί για όλες τις εισόδους ενός συγκεκριμένου τύπου.
- **Είσοδος-Έξοδος:** Ο αλγόριθμος πρέπει να παίρνει κάποια είσοδο και να παράγει μία μοναδική έξοδο για κάθε είσοδο.

Β. Θεωρία
1. Αλγόριθμος
3. Ψευδογλώσσα

Ένας αλγόριθμος θα διατυπώνεται στην ψευδογλώσσα:

- Η ψευδογλώσσα παρέχει όλες τις προγραμματιστικές ευκολίες που παρέχει μία συναρτησιακή γλώσσα προγραμματισμού (όπως π.χ. η C ή η Pascal αλλά σε ένα υψηλότερο επίπεδο αφαίρεσης
- Δηλαδή δεν μας ενδιαφέρουν οι τεχνικές λεπτομέρειες της υλοποίησης σε κάποια πραγματική γλώσσα προγραμματισμού, αλλά η ουσία των εντολών
- Δεν γινόμαστε δηλαδή «αφόρητα» τυπικοί όσον αφορά τη λεπτομέρεια της υλοποίησης.

Β. Θεωρία
1. Αλγόριθμος
3. Ψευδογλώσσα

Τα δομικά στοιχεία που θα συναντήσουμε στην ψευδογλώσσα είναι τα ακόλουθα:

- **Ορισμός διαδικασίας:**

```
procedure Ονομα-Διαδικασίας(ορίσματα)
    …
    Εντολές
    …
end όνομα-διαδικασίας
```

- **Δομή Συνθήκης**

```
if (συνθήκη) then
    …
    Εντολές
    …
end if
```

ή

```
if (συνθήκη) then
    …Εντολές…
else
    …Εντολές…
end if
```

Β. Θεωρία
1. Αλγόριθμος
3. Ψευδογλώσσα

- **Δομές Επανάληψης**
  - **Επανάληψη while…end while**

```
while(συνθήκη) then
    …Εντολές…
end while
```

  - **Επανάληψη repeat…until**

```
repeat
   …Εντολές…
until(συνθήκη)
```

  - **Επανάληψη for..end for**

```
for μεταβλητή=αρχ.τιμή to τελ.τιμή do
   …Εντολές…
end for
```

Β. Θεωρία
1. Αλγόριθμος
3. Ψευδογλώσσα

- Στα παραπάνω
  - **Εντολές είναι:**
    - Απλές εντολές π.χ. Εντολή Ανάθεσης: Θέσε x=5
    - Σύνθετες Εντολές π.χ.
      - Θέσε x=ο αριθμός των γειτόνων της κορυφής v
      - Θέσε x_1,x_2= οι ρίζες μιας δευτεροβάθμια εξίσωσης.
  - **Συνθήκες είναι**
    - Απλές Συνθήκες π.χ. (x>1)
    - Σύνθετες Συνθήκε π.χ. (x==1 και το z είναι άρτιος αριθμός)

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

1. Ορισμός Αναδρομικού Αλγορίθμου

- Οι περισσότεροι αλγόριθμοι, χρησιμοποιούν σαν δομικά στοιχεία τους την επανάληψη και την συνθήκη:
  - Χαρακτηρίζονται ως επαναληπτικοί ή διαδικαστικοί αλγόριθμοι

- Αντίθετα αν μια διαδικασία κατά τη διάρκεια εκτέλεσής της καλεί τον εαυτό της, τότε λέγεται **αναδρομική διαδικασία.**
  - Ένας αλγόριθμος που υλοποιείται από μία αναδρομική διαδικασία, θα λέγεται **αναδρομικός αλγόριθμος**.

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

2. Παράδειγμα: Η ακολουθία Fibonacci

- ΠΡΟΒΛΗΜΑ: Δίνεται ένας φυσικός αριθμός n. Να υπολογιστεί ο n-ός αριθμός Fibonacci.
- Υπενθύμιση: Οι δύο πρώτοι αριθμοί Fibonacci είναι 1 και κάθε επόμενος αριθμός Fibonacci, ορίζεται ως το άθροισμα των δύο προηγουμένων αριθμών Fibonacci:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
1	1	2	3	5	8	13	21	34	55	89	144	233	377	…

- Και τυπικά η ακολουθία ορίζεται μέσω της αναδρομικής σχέσης:

$$f_n = \begin{cases} 1, & n=1 \ \acute{\eta} \ n=2 \\ f_{n-1} + f_{n-2}, & n>2 \end{cases}$$

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

2. Παράδειγμα: Η ακολουθία Fibonacci

- Ένας αναδρομικός αλγόριθμος που υλοποιεί την παραπάνω διαδικασία είναι ο εξής:

Αλγόριθμος Εύρεσης n-οστού αριθμού Fibonacci
Είσοδος: Φυσικός Αριθμός n
Έξοδος: Η τιμή του n-οστού αριθμού Fibonacci.

```
procedure FibRec(n)
   if n=1 or n=2 then
      return 1
   else
      a=FibRec(n-1)
      b=FibRec(n-2)
      c=a+b
      return c
   end if
end procedure
```

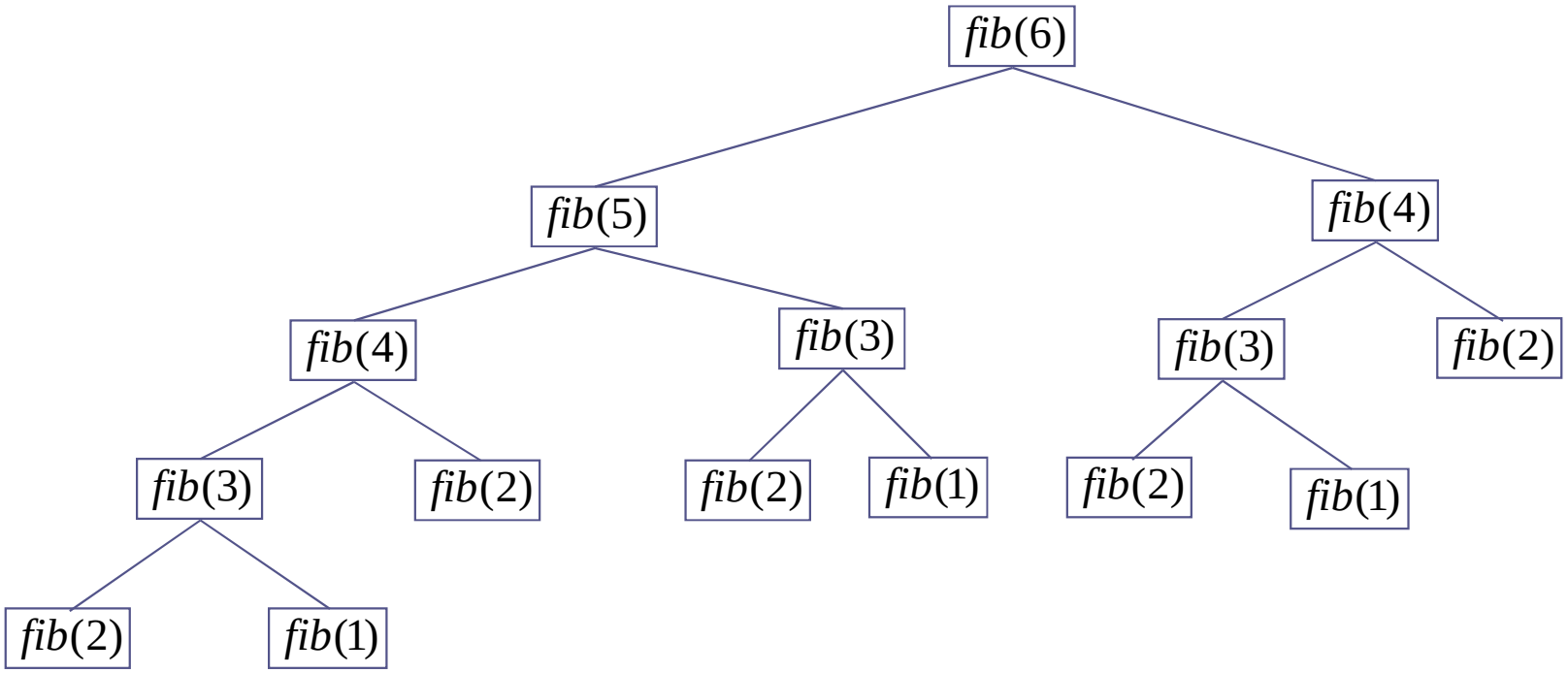
- Ο αλγόριθμος είναι αναδρομικός. Για τον υπολογισμό γίνεται κλήση της ίδιας διαδικασίας.

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

2. Παράδειγμα: Η ακολουθία Fibonacci

- Είναι πολύ αργός αλγόριθμος!!
  - Λόγω του τρόπου εκτέλεσης της αναδρομικής διαδικασίας. Π.χ. για n=6:



Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

2. Παράδειγμα: Η ακολουθία Fibonacci

- Παρατηρούμε ότι ο αναδρομικός αλγόριθμος είναι πολύ χρονοβόρος, διότι:
  - Γίνεται πολλές φορές υπολογισμός του ίδιου αριθμού
    - Π.χ. γίνεται δύο φορές κλήση της fibRec(4)
- Ο ακόλουθος αλγόριθμος υπολογίζει τον n-οστό αριθμό Fibonacci επαναληπτικά:

Αλγόριθμος Εύρεσης n-οστού αριθμού Fibonacci
Είσοδος: Φυσικός Αριθμός n
Έξοδος: Η τιμή του n-οστού αριθμού Fibonacci.

```
procedure FibSeq(n)
   A[1]=1
   A[2]=1
   for i=3 to n
      A[i]=A[i-1]+A[i-2]
   end for
   return A[n]
end procedure
```

- Είναι αποδοτικότερος! Κάθε τιμή υπολογίζεται μία φορά!

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

3. Παράδειγμα: Υπολογισμός Παραγοντικού

- ΠΡΟΒΛΗΜΑ: Δίνεται ένας φυσικός αριθμός n. Να υπολογιστεί το n!
- Υπενθύμιση: $n! = n \times (n-1) \times \cdots \times 2 \times 1$
- Και τυπικά η ακολουθία ορίζεται μέσω της αναδρομικής σχέσης:

$$n! = \begin{cases} 1, & n = 1 \\ n \cdot (n-1)!, & n > 1 \end{cases}$$

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

3. Παράδειγμα: Υπολογισμός Παραγοντικού

- Ένας αναδρομικός αλγόριθμος που υλοποιεί την παραπάνω διαδικασία είναι ο εξής:

Αλγόριθμος Εύρεσης Παραγοντικού
Είσοδος: Φυσικός Αριθμός n
Έξοδος: Η τιμή του n!

```
-------------------------------------------------------------
procedure FactRec(n)
   if n=1 then
      return 1
   else
      a=FactRec(n-1)
      c=n*a
      return c
   end if
end procedure
```

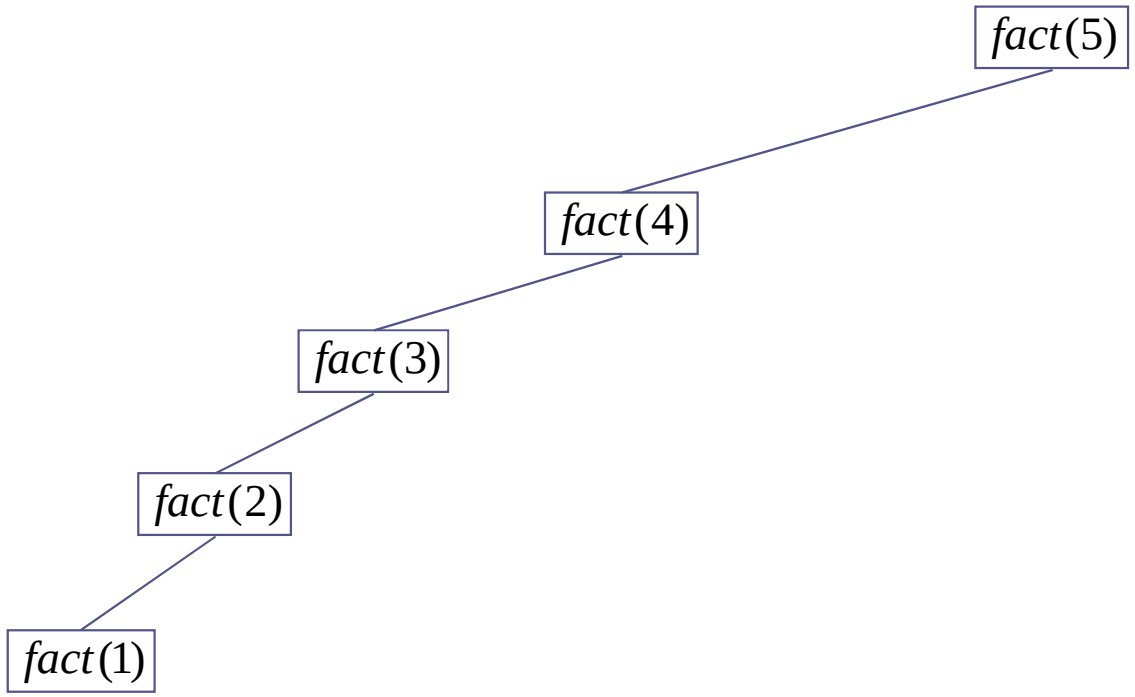
- Ο αλγόριθμος είναι αναδρομικός. Για τον υπολογισμό γίνεται κλήση της ίδιας διαδικασίας.

Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

3. Παράδειγμα: Υπολογισμός Παραγοντικού

- Είναι ικανοποιητικός αλγόριθμος:
  - Λόγω του τρόπου εκτέλεσης της αναδρομικής διαδικασίας, όπου εδώ κάθε κλήση γίνεται 1 φορά:



Β. Θεωρία

2. Αναδρομικοί Αλγόριθμοι

3. Παράδειγμα: Υπολογισμός Παραγοντικού

- Γενικά ένας αλγόριθμος που υλοποιείται από μία αναδρομική διαδικασία, μπορεί να υλοποιηθεί και από επαναληπτικό αλγόριθμο.
- Η κατασκευή αναδρομικών αλγορίθμων απαιτεί εμπειρία (βλέπε ΠΛΗ30)
- Ας ρίξουμε μια ματιά και σε μια υλοποίηση με επαναληπτικό αλγόριθμο της παραπάνω διαδικασίας:

Αλγόριθμος Εύρεσης n-οστού αριθμού Fibonacci
Είσοδος: Φυσικός Αριθμός n
Έξοδος: Η τιμή του n-οστού αριθμού Fibonacci.

```
procedure FactSeq(n)
   A[1]=1
   for i=2 to n
      A[i]=i*A[i-1]
   end for
   return A[n]
end procedure
```

Γ. Ασκήσεις

Εφαρμογή 1

Δίνεται ο παρακάτω αναδρομικός αλγόριθμος:

```
procedure rec(A, left, right)
    if left = right then
        return(A[left]);
    x := rec(A, left+1, right);
    if A[left] ≤ x then
        result:= A[left];
    else result:= x;
        return(result);
end procedure
```

Η διαδικασία *rec*(*A*, *left*, *right*) δέχεται ως παραμέτρους τον πίνακα ακεραίων *A* και τους φυσικούς αριθμούς *left* και *right*. Υποθέτουμε ότι ισχύει πάντοτε ότι *left* ≤ *right*. Αν ο πίνακας *A* έχει *n* στοιχεία, η αρχική κλήση είναι *rec*(*A*, 1, *n*). Ο συμβολισμός *A*[*left*] δηλώνει το στοιχείο του πίνακα *A* στη θέση *left*.

Έστω ότι *A* = [6, 1, 3, 2, 7, 3, 5, 12, 2, 8].

Να εκτελεστούν όλα τα βήματα της κλήσης *rec*(*A*, 1, 10) με είσοδο τον πίνακα *A* και να βρεθεί τι κάνει η διαδικασία rec.

Γ. Ασκήσεις

Εφαρμογή 2

Δίνεται ο παρακάτω αναδρομικός αλγόριθμος:

```
procedure fun(A, left, right, k)
   if left ≥ right then
      if k = A[left] then return(1);
      else return(0);
   mid := [(left + right) / 2];
   x := fun(A, left, mid, k);
   y := fun(A, mid+1, right, k);
   return(x + y);
```

Η διαδικασία *fun*(*A*, *left*, *right*, *k*) δέχεται σαν είσοδο τον πίνακα ακεραίων *A* και τις ακέραιες μεταβλητές *left*, *right* και *k*, και επιστρέφει σαν έξοδο έναν ακέραιο αριθμό. Αν ο πίνακας *A* έχει *n* στοιχεία, η αρχική κλήση είναι *fun* (*A*, 1, *n*, *k*). Ο συμβολισμός *A*[*left*] δηλώνει το στοιχείο του πίνακα *A* στη θέση *left*. Η παράσταση [(*left* + *right*) / 2] δηλώνει το **κάτω ακέραιο** μέρος της διαίρεσης, π.χ. [(1+8) / 2] = 4.

Ποια λειτουργία επιτελεί η διαδικασία *fun*(*A*, 1, *n*, *k*) (δηλαδή, ποια είναι η ιδιότητα της τιμής που επιστρέφει η *fun* σε σχέση με τα στοιχεία του πίνακα *A* και τον αριθμό *k*); Επαληθεύστε το με την εκτέλεση: fun([1 2 8 4 3 2 8 4 2 3], 1, 10, 2)