ΗΜΥ 325: Επαναληπτικές Μέθοδοι

Διδάσκων: Χρίστος Παναγιώτου

ΗΜΥ 325: Επαναληπτικές Μέθοδοι.

- A. Levitin, “Introduction to the Design and Analysis of Algorithms”, 2nd Ed.
- Περίληψη μαθήματος
- Επιπρόσθετες Πληροφορίες
  - www.eng.ucy.ac.cy/christos/courses/ECE325

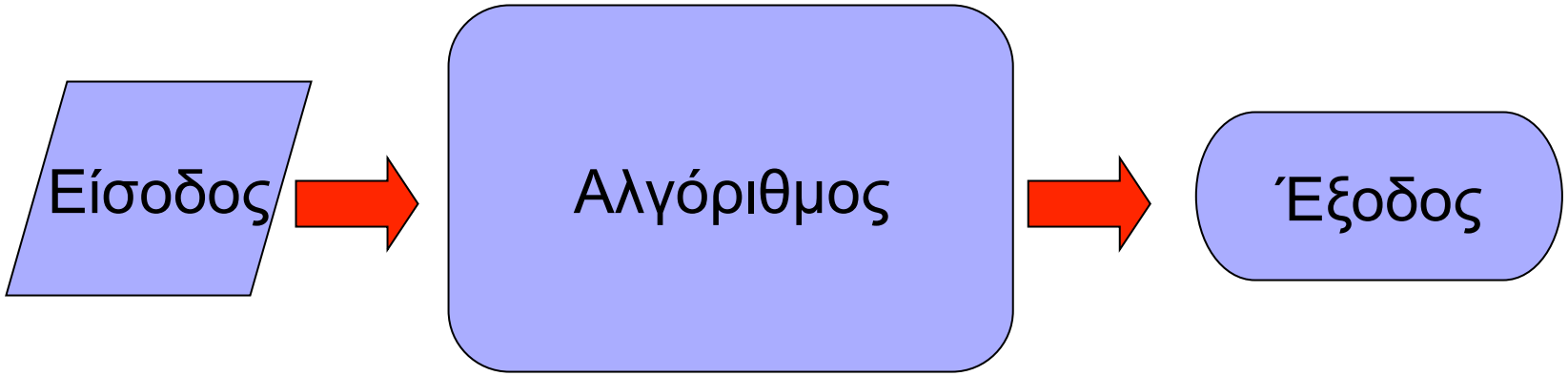
Αθροίσματα

- Υπολογίστε το άθροισμα:

$$
\begin{array}{cccccc}
0 & 0 & 1 & 0 & 1 & \\
2 & 4 & 6 & 8 & 3 & 8 \\
6 & 4 & 2 & 3 & 2 & 4 \\
\hline
8 & 8 & 9 & 1 & 6 & 2
\end{array}
$$

Επαναληπτικές Μέθοδοι

- Τι είναι αλγόριθμος;
- Αλγόριθμος είναι μια σειρά από σαφή (unambiguous) βήματα τα οποία λύνουν ένα πρόβλημα.



Αλγόριθμος

- **Σαφή** βήματα (εντολές)
- Είσοδος: Το είδος και εύρος των δεδομένων εισόδου θα πρέπει επίσης να είναι σαφή
- Ο αλγόριθμος μπορεί να περιγραφεί με πολλούς τρόπους (διαγραμματικά, ψευδο-κώδικας)
- Μπορεί να υπάρχουν πολλοί αλγόριθμοι που λύνουν ένα πρόβλημα
  - Μπορεί να βασίζονται σε πολύ διαφορετικές ιδέες
  - Μπορεί να έχουν πολύ διαφορετική απόδοση (ταχύτητα επίλυσης ή απαιτήσεις μνήμης)
  - Διαφορετική δυσκολία κατανόησης

Αλγόριθμος Αθροίσματος

- Είσοδος: *x[.], y[.]*
- Έξοδος: *z=x+y*

```
c= 0;
for i=1 to n
    s= x[i]+y[i]+c;
    if s>=10
        c=1; s=s-10;
    else
        c=0;
    z[i]=s;
return z;
```

Μέγεθος των x[], y[]

	0	0	1	0	1	
	2	4	6	8	3	8
	6	4	2	3	2	4
	8	8	9	1	6	2

- Είναι ο αλγόριθμος ορθός;
- Είναι αποδοτικός;

- **Δεν** Υπάρχει (και δεν μπορεί να υπάρξει) **ένας** αλγόριθμος που να λύνει όλα τα προβλήματα!

Παράδειγμα

- Βρείτε το μέγιστο κοινό διαιρέτη (greatest common divisor) μεταξύ δύο ακέραιων αριθμών m, n.
- Είσοδος: m, n
- Έξοδος: z // μέγιστος κοινός διαιρέτης

Αλγόριθμος 1

1. Find the prime factors of m
2. Find the prime factors of n
3. Identify all common factors that appear in both
4. Compute the product of the common factors

- Παράδειγμα:

60= 2 x 2 x 3 x 5

24= 2 x 2 x 2 x 3

Z = 2 x 2 x 3 = 12

Αλγόριθμος 1

Πώς βρίσκουμε όλους τους πρώτους αριθμούς στο διάστημα $[0, max\{n, m\}]$;

1. Find the prime factors of m
2. Find the prime factors of n
3. Identify all common factors that appear in both
4. Compute the product of the common factors

Πώς βρίσκουμε όλους τους κοινούς αριθμούς σε 2 λίστες;

Αλγόριθμος 1 (παρένθεση)

- Πως αναγνωρίζουμε αν ένας αριθμός είναι πρώτος;

```
isPrime(n)
    for i=2 to n-1
       if n mod i == 0 return false;
    return true
```

```
isPrime(n)
    for i=2 to floor(n/2)
       if n mod i == 0 return false;
    return true
```

- **Fermat's Little Theorem**

Fermat's Little Theorem

- *Εάν p είναι πρώτος, τότε* ***γ α κάθε*** *$1 \le \alpha < p$ ισχύει*

$$a^{p-1} \equiv 1 \pmod{p}$$

- **Παράδειγμα**: Το p=5 είναι πρώτος αριθμός.

$$1^{5-1} \bmod 5 = 1^4 \bmod 5 = 1 \bmod 5 = 1$$

$$2^{5-1} \bmod 5 = 2^4 \bmod 5 = 16 \bmod 5 = 1$$

$$3^{5-1} \bmod 5 = 3^4 \bmod 5 = 81 \bmod 5 = 1$$

$$4^{5-1} \bmod 5 = 4^4 \bmod 5 = 256 \bmod 5 = 1$$

- **Προσοχή**: Το θεώρημα **δεν** είναι *εάν και μόνον άν*!!!

Αλγόριθμος 2

1. t= min{m, n}
2. If m mod t= 0 Goto 3, Else Goto to 4;
3. If n mod t= 0 return t;
4. Set t= t-1 and Goto 2.

Τι γίνεται εάν m=0 ή n=0;

- Παράδειγμα: m=60, n= 24
- t= min{60, 24}= 24
- 60 mod 24= 12 ≠ 0
- t= 23
- 60 mod 23= 14 ≠ 0
- t= 22
- 60 mod 22= 16 ≠ 0
- t= … t= 12
- 60 mod 12= 0
- 24 mod 12= 0
- Return 12

Αλγόριθμος 3 (Αλγόριθμος του Ευκλείδη)

```
gcd(m, n)
    If n=0 Return m
    gcd(n,m mod n)
```

- Παράδειγμα: m=60, n= 24

gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12

Αρχές Επίλυσης Προβλημάτων με Αλγοριθμικές Μεθόδους

- Κατανόηση του προβλήματος
  - Είναι αδύνατο να λύσετε ένα πρόβλημα εάν δεν το καταλάβετε!
  - Πρέπει να οριστεί επακριβώς το σύνολο όλων των πιθανών εισόδων.
- Δυνατότητες της «μηχανής» που θα λύσει το πρόβλημα
  - Εάν θα λυθεί στο «χέρι» τότε δεν μπορεί να έχει περισσότερα από μερικές δεκάδες βήματα.
  - Ένας υπολογιστής δεν έχει «άπειρη» υπολογιστική ικανότητα!

Αρχές Επίλυσης Προβλημάτων με Αλγοριθμικές Μεθόδους

- Ακριβής λύση ή λύση κατά προσέγγιση (exact or approximate algorithm);
  - Μερικά προβλήματα δεν μπορούν να λυθούν ακριβώς
    - Υπολογισμός τετραγωνική ρίζας ή ορισμένου ολοκληρώματος.
  - Μια ακριβής λύση μπορεί να συνεπάγεται τεράστιο υπολογιστικό φόρτο σε αντίθεση με μια προσεγγιστική λύση η οποία μπορεί να είναι υπολογιστικά πολύ πιο αποδοτική
  - Ο συνδυασμός προσεγγιστικών λύσεων μπορεί να μας δώσει την ακριβή λύση

Αρχές Επίλυσης Προβλημάτων με Αλγοριθμικές Μεθόδους

- Μοντελοποίηση
  - Περιγραφή του προβλήματος με σύμβολα
  - Δομές δεδομένων
- Σχεδιασμός του αλγορίθμου επίλυσης του προβλήματος
- Περιγραφή του αλγορίθμου
  - Διαγράμματα ροής
  - Ψευδο-κώδικας (pseudocode)
- Ανάλυση της ορθότητας του αλγόριθμου
  - Για όλα τα πιθανά δεδομένα εισόδου
  - Μαθηματική επαγωγή (mathematical induction)

Αρχές Επίλυσης Προβλημάτων με Αλγοριθμικές Μεθόδους

- Ανάλυση της απόδοσης του αλγόριθμου
  - Χρονικές απαιτήσεις
  - Απαιτήσεις μνήμης
  - Απλότητα (simplicity)
    - Απλοί αλγόριθμοι κατανοούνται και υλοποιούνται ευκολότερα
  - Γενικότητα (generality)
    - Ένας αλγόριθμος που λαμβάνει υπόψη όλες τις πιθανές εισόδους μπορεί να είναι πολύ πιο πολύπλοκος από κάποιον άλλο ο οποίος μπορεί να αγνοεί σενάρια τα οποία να μην εμφανίζονται πολύ συχνά.
- Υλοποίηση
  - Έλεγχος ορθότητας

Μαθηματική Επαγωγή (Mathematical Induction)

- Μέθοδος απόδειξης σχέσεων/συνθηκών

- Αποδεικνύουμε πως η σχέση ισχύει για μια αρχική τιμή.
- Υποθέστε πως η σχέση ισχύει για την τιμή n και δείξετε πως η σχέση ισχύει επίσης για την τιμή $n+1$.

Μαθηματική Επαγωγή (Παράδειγμα)

- Δείξτε πως $\sum_{i=1}^{n} i = 1 + \cdots + n = \frac{n(n+1)}{2}$
- Για n=1: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$

Ισχύει

- Υποθέτουμε πως ισχύει για n τότε πρέπει να δείξουμε ότι ισχύει επίσης για n+1

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+2)}{2}$$

Ισχύει, οπόταν η απόδειξη έχει ολοκληρωθεί!

Loop Invariants

- Μέθοδος απόδειξης της ορθότητας ενός αλγορίθμου η οποία βασίζεται στην μαθηματική επαγωγή.
- **Αρχικοποίηση**: Αποδεικνύουμε πως ο αλγόριθμος είναι ορθός κατά την πρώτη επανάληψη.
- **Διατήρηση**: Υποθέτουμε πως ο αλγόριθμος είναι ορθός κατά την επανάληψη n και δείχνουμε ότι παραμένει ορθός και κατά την επόμενη επανάληψη $n+1$.
- **Αποπεράτωση**: Δείχνουμε πως κατά τον τερματισμό ο αλγόριθμος παραμένει ορθός.

Ορθότητα Αλγόριθμου Αθροίσματος

- Είσοδος: *x[.], y[.]*
- Έξοδος: *z=x+y*

- `c= 0;`
- **`for`** `i=1` **`to`** `n`
  - `s= x[i]+y[i]+c;`
  - **`if`** `s>=10`
    - `c=1; s=s-10;`
  - **`else`**
    - `c=0;`
  - `z[i]=s;`
- **`return`** `z;`

- Είναι ο αλγόριθμος ορθός;
  - Αρχικοποίηση;

- Για `i=1`
  - `s= x[1]+y[1]+0;`
  - **`if`** `s>=10`
    - `c=1; s=s-10;`
  - **`else`**
    - `c=0;`
  - `Z[1]=s;`
- **Ορθό!**

Ορθότητα Αλγόριθμου Αθροίσματος

- Είσοδος: *x[.], y[.]*
- Έξοδος: *z=x+y*

```
c= 0;
for i=1 to n
    s= x[i]+y[i]+c;
    if s>=10
        c=1; s=s-10;
    else
        c=0;
    z[i]=s;
return z;
```

- Είναι ο αλγόριθμος ορθός;
  - Αρχικοποίηση;
  - Διατήρηση;
- Υποθέστε ορθός μέχρι `i=k`

```
    s= x[k+1]+y[k+1]+c;
    if s>=10
        c=1; s=s-10;
    else
        c=0;
    Z[k+1]=s;
```

- **Ορθό!**

Ορθότητα Αλγόριθμου Αθροίσματος

- Είσοδος: *x[.], y[.]*
- Έξοδος: *z=x+y*

```
c= 0;
for i=1 to n
    s= x[i]+y[i]+c;
    if s>=10
        c=1; s=s-10;
    else
        c=0;
    z[i]=s;
return z;
```

- Είναι ο αλγόριθμος ορθός;
  - Αρχικοποίηση;
  - Διατήρηση;
  - Τερματισμός;

```
c=x[n+1]=y[n+1]=0;
for i=1 to n+1
    s= x[i]+y[i]+c;
    if s>=10
        c=1; s=s-10;
    else
        c=0;
    z[i]=s;
return z;
```

Δεν τερματίζει ορθά!

Τύποι Προβλημάτων

- Προβλήματα Ταξινόμησης (sorting)
- Προβλήματα Αναζήτησης (searching)
- Επεξεργασία σειράς συμβόλων (string processing)
- Προβλήματα σε γράφους (graph problems)
- Συνδυαστικά Προβλήματα (combinatorial problems)
- Γεωμετρικά Προβλήματα (geometric problems)
- Αριθμητικά Προβλήματα (numerical problems)

Προβλήματα Ταξινόμησης (sorting)

- Δεδομένου ενός συνόλου από στοιχεία ταξινομήστε τα σε μια σειρά με βάση κάποιο **κλειδί** (**key**).
  - Το κλειδί θα πρέπει να ικανοποιεί μια σχέση που να επιτρέπει την ταξινόμηση
- Γιατί είναι σημαντικό να διατηρούμε ταξινομημένες λίστες;
  - Προσπαθήστε να βρείτε το τηλέφωνο του …. σε ένα κατάλογο που **δεν** είναι αλφαβητικά ταξινομημένος!
  - Υποβοηθητικό βήμα για την επίλυση άλλων προβλημάτων
- Υπάρχει ένας μεγάλος αριθμός από αλγόριθμους ταξινόμησης. Γιατί;
- Ιδιότητες
  - Ευσταθείς (stable) αλγόριθμοι ταξινόμησης
  - Αλγόριθμοι που δεν χρειάζονται επιπρόσθετη μνήμη

Προβλήματα Αναζήτησης (searching)

- Αναζήτηση ενός ή περισσοτέρων στοιχείων (αντικειμένων) σε ένα σύνολο σύμφωνα με το ***κλε δ*** αναζήτησης.
  - Πολλές φορές τα προβλήματα αναζήτησης μπορεί να συνδυαστούν με δύο άλλα προβλήματα
    - Αναζήτηση και διαγραφή (αφαίρεση) από το σύνολο
    - Αναζήτηση και προσθήκη (πρόσθεση) στο σύνολο
- Προβλήματα βελτιστοποίησης μπορούν να διατυπωθούν σαν προβλήματα αναζήτησης
  - Βρείτε τα στοιχεία του συνόλου για τα οποία μια συνάρτηση παίρνει τη βέλτιστη τιμή (ελάχιστη ή μέγιστη).

Προβλήματα σε γράφους (graph problems)

- G=(V,E)
  - Γράφος (Graph) G
  - Κόμβοι ή κορυφές (Vertices) V
  - Τόξα ή ακμές (Edges) E
- Χρησιμοποιούνται ευρέως για τη μοντελοποίηση πολλών προβλημάτων
- Προβλήματα βελτιστοποίησης μπορούν να διατυπωθούν σαν προβλήματα σε γράφους συνδέοντας τα τόξα με κάποιο κόστος ή βάρος (cost or weight).

Συνδυαστικά Προβλήματα (combinatorial problems)

- Προβλήματα που συνήθως ζητούν την εύρεση των συνδυασμών αντικειμένων που ικανοποιούν κάποιους περιορισμούς (constraints) και που βελτιστοποιούν κάποια συνάρτηση κόστους.
  - Συνάρτηση κόστους (cost function) ή αντικειμενική συνάρτηση (objective function).
- Παράδειγμα
  - Ένας βαρκάρης πρέπει να περάσει ένα λύκο, μια κατσίκα και ένα μαρούλι από την μια όχθη του ποταμού στην άλλη χρησιμοποιώντας μία βάρκα στην οποία μπορεί να χωρέσει μόνο ένα αντικείμενο!
  - Ποιοι οι περιορισμοί και πως μπορεί να το καταφέρει;

Γεωμετρικά Προβλήματα (geometric problems)

- Αλγοριθμική λύση γεωμετρικών προβλημάτων
- Τα προβλήματα αυτά εμφανίζονται συχνά σε διάφορες εφαρμογές ρομποτικής, γραφικών, τομογραφίας κλπ.
- Παραδείγματα
  - Ζευγάρι των πιο κοντινών σημείων (closest-pair problem)
  - Convex hull problem (ποιο το μικρότερο μήκος ενός φράκτη που να εμπερικλείει ένα σύνολο από σημεία)
  - Εντοπισμός θέσης.

Αριθμητικά Προβλήματα (Numerical problems)

- Συμπεριλαμβάνουν ένα μεγάλο εύρος προβλημάτων
  - Υπολογισμός ορισμένων ολοκληρωμάτων
  - Επίλυση συστήματος εξισώσεων
  - Μετασχηματισμοί Fourier
  - …
- Πολλά από αυτά τα προβλήματα λύνονται (σε υπολογιστές) κατά προσέγγιση αφού ένας υπολογιστής έχει περιορισμένη ακρίβεια ενώ οι άρρητοι αριθμοί χρειάζονται άπειρο αριθμό ψηφίων.
- Η αριθμητική προσέγγιση λόγω της στρογγυλοποίησης αριθμών μπορεί να δημιουργήσει πολύ μεγάλα σφάλματα ιδιαίτερα σε επαναληπτικούς (iterative) ή αναδρομικούς (recursive) αλγόριθμους!

Προβλήματα Αποφάσεων και Βελτιστοποίησης

- Προβλήματα Αποφάσεων (Decision Problems):
  - Προβλήματα του τύπου αυτού είναι διατυπωμένα με τέτοιο τρόπο ώστε η απάντηση παίρνει τη μορφή (ναι/όχι, yes/no, 1/0).
  - Παράδειγμα:
    - Το 1234567 είναι πρώτος (prime) αριθμός;
    - Παρουσιάζεται η λέξη algorithm στο κείμενο myDoc.txt
- «Γενικά» Προβλήματα (Function Problems)
  - Προβλήματα στα οποία η απάντηση δεν είναι της μορφής ναι/όχι
  - Παράδειγμα
    - Αναλύστε το 1234567 σαν γινόμενο πρώτων αριθμών.
  - Προβλήματα βελτιστοποίησης

Προβλήματα Αποφάσεων και Βελτιστοποίησης

- Προβλήματα βελτιστοποίησης:
  - Υποθέστε πως X αποτελεί το σύνολο όλων των πιθανών περιπτώσεων (στιγμιότυπων) ενός προβλήματος.
  - $C(X) \in X$ είναι υποσύνολο του X με όλα τα στοιχεία που ικανοποιούν όλους τους περιορισμούς του προβλήματος.
  - $f(x)$ είναι μια συνάρτηση

$$\max_{x \in C(X)} f(x)$$

$$\min_{x \in C(X)} f(x)$$