Αλγόριθμοι & Δομές Δεδομένων

Ιωάννης Γαβιώτης
Ζ΄ εξάμηνο

Τμήμα Μηχανικών Σχεδίασης
Προϊόντων & Συστημάτων

1
Εισαγωγή

Αλγόριθμος

- Η ακριβής περιγραφή μιας αυστηρά καθορισμένης σειράς ενεργειών (βημάτων) για τη λύση ενός προβλήματος.
- Υπολογιστική συνταγή: ένας γενικός τρόπος να κάνω κάτι, που είναι τόσο συγκεκριμένος ώστε οποιοσδήποτε (συμπεριλαμβανομένων και των υπολογιστών) μπορεί να τον ακολουθήσει.

3

Δομή Δεδομένων

Data structure

- Παράσταση γεγονότων, εννοιών ή εντολών σε τυποποιημένη μορφή που είναι κατάλληλη για επικοινωνία, ερμηνεία ή επεξεργασία από τον άνθρωπο ή από αυτόματα μέσα.

  ΤΕ48 - ΕΛΟΤ

- Προγράμματα = Αλγόριθμοι
  \+
  Δομές Δεδομένων

  Niklaus Wirth

4

Τι μας Ενδιαφέρει σε ένα Πρόγραμμα

- Είναι σωστό;
- Είναι εύκολο να κατανοήσουμε τον κώδικα;
- Είναι τεκμηριωμένο;
- Είναι εύκολο να γίνουν αλλαγές;
- Πόση μνήμη και πόσο χρόνο απαιτεί;
- Πόσο γενικός είναι ο κώδικας;
  - Λύνει προβλήματα για μεγάλο εύρος εισόδων χωρίς τροποποιήσεις;
- Πόσο μεταφέρσιμος είναι ο κώδικας;
  - Μπορεί να μεταγλωττιστεί σε άλλους υπολογιστές χωρίς αλλαγές;

5

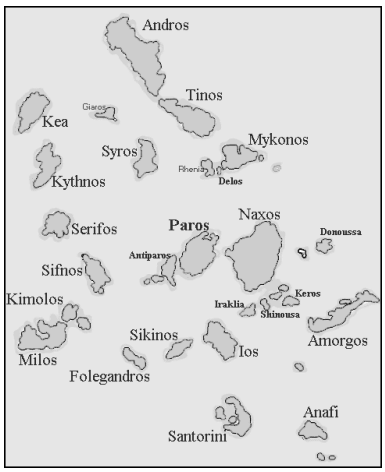
Χαρακτηριστικά Αλγορίθμου

- Μπορεί να έχει ή να μην έχει **εισόδους**.
- Έχει τουλάχιστον μια **έξοδο**.
- Κάθε εντολή του είναι **σαφής** και χωρίς διφορούμενα.
- Σε κάθε περίπτωση ο αλγόριθμος τερματίζει σε **πεπερασμένο** αριθμό βημάτων.
- Κάθε εντολή πρέπει να είναι αρκετά «**απλή**» ή να αναλύεται σε απλούστερες που προδιαγραφούν την μέθοδο εκτέλεσής τους.

6

Μικρότερη Διαδρομή

Φτιάξτε οικονομικά γέφυρες που να συνδέουν όλα τα νησιά



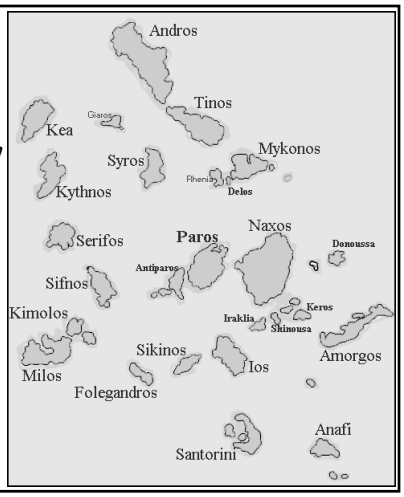
Υπάρχει γρήγορη λύση

7

Περιπλανόμενος Πωλητής

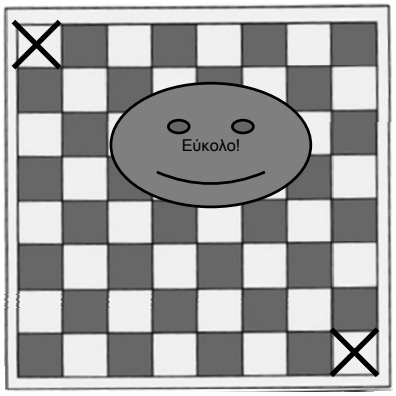
Traveling salesman

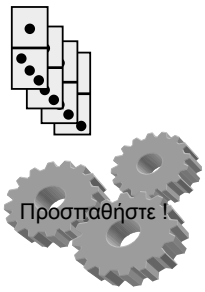
Φτιάξτε το συντομότερο δρομολόγιο πλοίου που περνάει από όλα τα νησιά



Δεν υπάρχει γρήγορη λύση

Ντόμινο στη Σκακιέρα



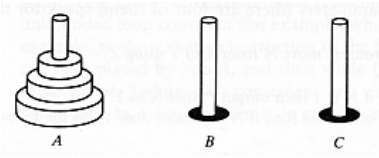


Αδύνατον !

9

Το Τέλος του Κόσμου…

… θα έλθει όταν οι βουδιστές μοναχοί μετακινήσουν τους 64 δίσκους σε άλλο πύργο

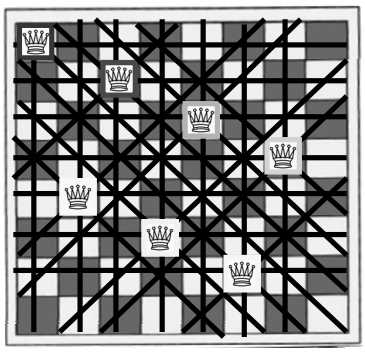


- Μετακινούν 1 δίσκο το δευτερόλεπτο

2^{64} δευτερόλεπτα, δηλ. 600.000.000 χιλιετίες

10

Οκτώ Βασίλισσες στη Σκακιέρα



Τοποθετήστε 8 βασίλισσες έτσι ώστε καμιά να μην απειλεί κάποια άλλη.

92 λύσεις !

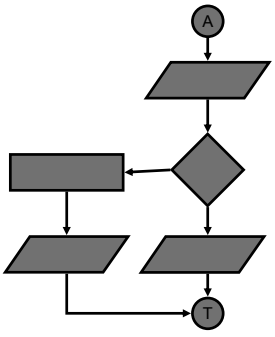
Οπισθοχώρηση

Δείτε στο http://students.ceid.upatras.gr/~papagel/project/kef5_8.htm

11

Παράσταση Αλγορίθμων

- Διαγράμματα ροής (flow charts)
- Ψευδο-κώδικας
- Απεικόνιση σε γλώσσα προγραμματισμού



12

Παράδειγμα Ψευδοκώδικα

```
function ΤαξινόμησηΜεΕπιλογή
  for i=1 to n
     βρες το ελάχιστο
     στοιχείο ψάχνοντας
     στις θέσεις i έως n

     αντάλλαξέ το με το
      στοιχείο στη θέση i
  end for i
end function
```

Πώς θα βρεις το ελάχιστο; Χρειάζεται να περιγράψεις τη μέθοδο.

3 εντολές ανάθεσης

13

Επεξεργασία Πληροφορίας

- Παλιότερα οι υπολογιστές χρησιμοποιούνταν ως **γρήγορες αριθμομηχανές**.
  - Έμφαση σε αριθμητικές πράξεις
- Τρέχουσα χρήση:
  επεξεργαστές πληροφορίας
  - Μεγάλη ποσότητα δεδομένων
  - Σημαντική η οργάνωση των δεδομένων

14

Τοποθεσία Δεδομένων

- Εξαρτάται:
  - από την ποσότητα των δεδομένων
  - Από άλλα χαρακτηριστικά, πχ μεταβλητότητα
- Για κύρια μνήμη, πχ RAM
  - Τυχαία προσπέλαση
  - Μικρό μέγεθος – υψηλή ταχύτητα
- Για δευτερεύουσα μνήμη
  - Προσπέλαση εξαρτάται από μέσον αποθήκευσης, πχ σκληρός δίσκος, ταινία
  - Μεγάλο μέγεθος
- Το ίδιο πρόβλημα μπορεί να απαιτεί διαφορετικούς αλγόριθμους όταν αλλάζει το μέγεθος των δεδομένων (και άρα η τοποθεσία τους).
  - Παράδειγμα: ΤαξινόμησηΦυσαλίδων έναντι ΤαξινόμησηςΜεΕπιλογή

15

Μελέτη Δεδομένων

- Γλώσσα προγραμματισμού που θα «ζήσουν» τα δεδομένα
  - Δήλωση και αναπαράσταση δεδομένων
  - Χειρισμός και λειτουργίες
- Η σωστή επιλογή δομής δεδομένων βελτιώνει κατά πολύ τις επιδόσεις του προγράμματος.

16

Σχετικά με το Μάθημα

- Διδάσκοντες
  - Θεωρία: gaviotis@aegean.gr
  - Εργαστήριο: evlach@aegean.gr
- Διαλέξεις
  - Τρίτη 16:00-19:00 @ Αμφιθέατρο
- Εργαστήριο (υποχρεωτικό)
  - Τετάρτη 09:00-11:00 @ Εργαστήριο υπολογιστών

17

Προαπαιτούμενες Γνώσεις

- Γλώσσα προγραμματισμού (VB, C, Java)
  - Μεταβλητές & παραστάσεις
  - Μέθοδοι (συναρτήσεις / διαδικασίες)
  - Δομές ελέγχου (**`if`**, **`switch`**)
  - Επανάληψη (**`for`**, **`while`**)
  - Τεκμηρίωση, αποσφαλμάτωση, κλάσεις/αντικείμενα
- Θα γίνει επανάληψη στο εργαστήριο

18

Τι θα Μάθουμε

- Μια μικρή, αλλά ενδεικτική γκάμα υπολογιστικών λύσεων
  - Προφανείς, κομψές, «έξυπνες»
- Τεχνικές για να σχεδιάζουμε δικές μας λύσεις σε προβλήματα
  - Διαίρει και βασίλευε (divide & conquer)
  - Ωμή βία (brute force)
  - Οπισθοχώρηση (backtracking)

19

Τι Άλλο Θα Μάθουμε

- Να εκτιμούμε την απόδοση των αλγορίθμων
  - Ταχύτητα, απαιτήσεις μνήμης
  - Υπάρχει πιο καλή λύση στο ίδιο πρόβλημα
    - … μήπως αν «πειράξω» λίγο το πρόβλημα
- Οργανώσεις δεδομένων που «διευκολύνουν» τη λύση προβλημάτων
- Στόχος: να δούμε πολλά προβλήματα και πολλές λύσεις τους

20

Ποιος Κάνει Τι

	Αλγόριθμος	Εκτέλεση
Σενάριο Α	Δάσκαλος	Φοιτητής
Σενάριο Β	Φοιτητής	Υπολογιστής

21

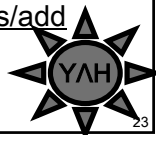
Στο Εργαστήριο

- Εξάσκηση στη Java (3 πρώτα εργαστήρια)
- Παραδείγματα εκτέλεσης των αλγορίθμων
  - Εκτέλεση βήμα-προς-βήμα
  - Δοκιμή επιδόσεων (benchmark)
- Οπτικοποίηση των δομών δεδομένων
  - Πώς εξελίσσονται στη ζωή τους
- Εξηγήσεις, υποδείξεις & συμβουλές για τις ασκήσεις

22

Διδακτικά Εγχειρίδια - Σημειώσεις

- 2 βιβλία
  - Gregory Rawlins, *Αλγόριθμοι Ανάλυση και Σύγκριση*, Κριτική, 2004
  - Γεώργιος Γεωργακόπουλος, *Δομές Δεδομένων*, Πανεπιστημιακές Εκδόσεις Κρήτης, 960-524-125-0, 2002
- Υλικό στην ιστοσελίδα του μαθήματος www.syros.aegean.gr/users/gaviotis/add
  - Κανονισμός μαθήματος, Ασκήσεις, κά
  - Διαφάνειες



23

Παραπομπές

- M. Goodrich, R. Tamassia, *Data Structures & Algorithms in Java*, 2004, J Wiley
  - http://java.datastructures.net
- W. Collins, *Data Structures and the Java Collection Framework, 2005,* McGraw-Hill
  - http://www.mhhe.com/collins
- Sahni, *Data Structures, Algorithms and Applications in Java,* 2000, McGraw-Hill
  - http://www.mhhe.com/sahnijava

24

Αξιολόγηση με Ασκήσεις

- Ενδεικτικά πέρυσι 6+2 ασκήσεις
  - προγράμματα, τροποποιήσεις προγραμμάτων
  - αναφορά (χαρτί)
- Συνεισφορά από 5% (οι εύκολες και γρήγορες) έως 15%
- Άθροισμα 115%
- Όχι γραπτή εξέταση
- Οι κανόνες θα οριστικοποιηθούν με την ολοκλήρωση των εγγραφών

25

2
Ανάλυση Επιδόσεων Αλγορίθμων

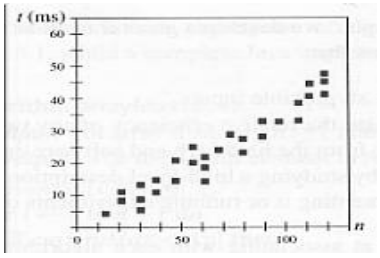
Επιδόσεις Αλγορίθμου

- Πόσος χρόνος χρειάζεται για την ολοκλήρωσή του (πειραματικά);
  - Μετρήσεις για διάφορα μεγέθη εισόδων
  - >> για διαφορετικές εισόδους ίδιου μεγέθους
- Εξάρτηση από:
  - Χαρακτηριστικά υπολογιστή
  - Μεταγλωττιστή
  - Πολυεπεξεργασία

27

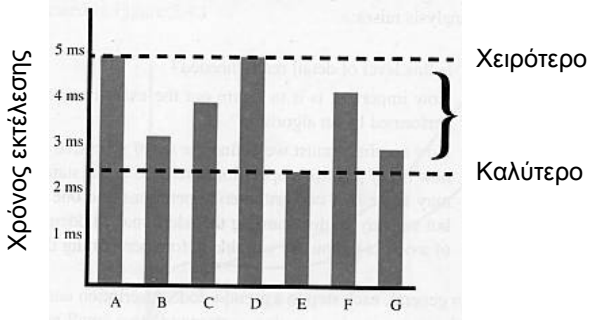
Χρονομέτρηση Προγράμματος

- Χρησιμοποιούμε το ρολόι του υπολογιστή
  - `System.currentTimeMillis()`
  - `Date()`
- Μετρούμε για διάφορες εισόδους ίδιου μεγέθους και διαφορετικά μεγέθη.



28

Βέλτιστο, Μέσο, Χείριστο

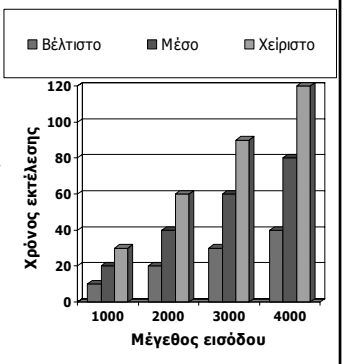


Σύνολα εισόδου του ίδιου μεγέθους

29

Μειονεκτήματα Μετρήσεων

- Πρέπει να υλοποιηθεί ο αλγόριθμος.
- Για να προσδιοριστούν βέλτιστα, μέσα, χείριστα πρέπει να εξαντληθούν όλες οι παραλλαγές των εισόδων συγκεκριμένου μεγέθους.
- Για να συγκριθούν αλγόριθμοι πρέπει να εκτελούνται στο ίδιο υλικό.



30

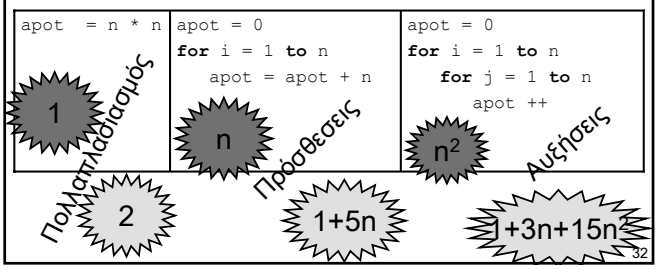
Ανάλυση Επιδόσεων

- Σε πόσα βήματα τερματίζει σε σχέση με το μέγεθος της εισόδου;
- Πολυπλοκότητα χρόνου
  - Βασικές πράξεις σταθερού κόστους:
    - Ανάθεση, κλήση / επιστροφή ρουτίνας, αριθμητική πράξη (+, -, *, /), σύγκριση δύο αριθμών, προσπέλαση στοιχείου πίνακα
- Πολυπλοκότητα χώρου
  - Πόση μνήμη απαιτείται για την αποθήκευση της δομής των δεδομένων;

31

Ποιες Είναι Βασικές Πράξεις;

Οι επιδόσεις εξαρτώνται από το υπολογιστικό μοντέλο της πλατφόρμας όπου εκτελείται ο αλγόριθμος



32

Άσκηση Επιδόσεων

Υπολογίστε πόσες φορές θα εκτελεστεί η εντολή S στο πρόγραμμα:

```
for i = 1 to n
  for j = 1 to i
     S
```

$$\sum_{i=1}^{n}\sum_{j=1}^{i}1=\sum_{i=1}^{n}i=\frac{n(n+1)}{2}\approx 0{,}5n^2$$

33

Μέτρηση Επιδόσεων

- Χειρότερη περίπτωση
  - Για δεδομένο μέγεθος εισόδου n επιλέγεται εκείνη η είσοδος που απαιτεί τις περισσότερες πράξεις.
  - Ασφαλές αλλά όχι ενδεικτικό όριο
- Μέσος όρος
  - Όλες οι δυνατές είσοδοι συγκεκριμένου μεγέθους θεωρούνται ισοπίθανες, υπολογίζεται το κόστος καθεμιάς και βγαίνει ο μέσος όρος.
  - Δυσκολότερο να υπολογιστεί

34

Εύρεση Μέγιστου

```
function μέγιστοςΔιανύσματος
  Είσοδος: πίνακας A[0..n], μη ταξινομημένος
  Εξοδος: το μεγαλύτερο στοιχείο του πίνακα

  t = A[0] προσωρινά μέγιστος είναι ο πρώτος
  for i = 1 to n
     if A[i] > t then
          t = A[i]
     end if
  end for
  return t
end function
```

Βήματα
n
0...n

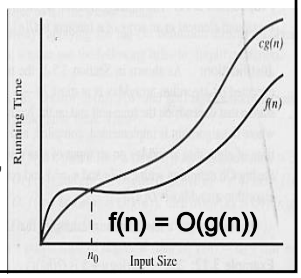
Μετρήστε με ακρίβεια όλες τις στοιχειώδεις πράξεις (αναφορές στη μνήμη, αναθέσεις, συγκρίσεις, ...

35

Ασυμπτωτικός Συμβολισμός O

- Μια συνάρτηση f(n) είναι τάξης **O** με μια άλλη συνάρτηση g(n), όταν υπάρχουν ακέραιος n_0 και σταθερά c για τα οποία ισχύει για κάθε $n>n_0$, ότι $f(n) < c.g(n)$

Πρακτικά:
Για μέγεθος εισόδου πάνω από κάποιο όριο, η συνάρτηση g «φράσσει» την f.



Μέγεθος Τάξης O

- Όροι χαμηλότερης τάξεως μπορούν να παραληφθούν.
- Παραδείγματα:
  - $f(n)=3n^2+2n+1$, τότε $f(n) = O(n^2)$
  - $f(n)=n.\lg n+8n+3\lg n$, τότε $f(n) = O(n.\lg n)$
- lgn συμβολίζει $\log_2 n$
  - logn υπονοεί $\log_{10} n$
- Έστω $f(n) = 3n^2+4n+2$. Τότε $f(n) = O(n^2)$
- Αν f(n) είναι πολυώνυμο βαθμού d, τότε
  - Αγνοούμε τους όρους χαμηλότερων βαθμών.
  - Αγνοούμε το συντελεστή του x^d.
- Σύμφωνα με τον ορισμό ισχύει επίσης $f(n)=O(n^3)$, αλλά ας μην είμαστε σπάταλοι…

37

Γραμμική Αναζήτηση

```
function γραμμικήΑναζήτηση
  Είσοδος: πίνακας A[1..n], μη ταξινομημένος
  Είσοδος: αριθμός x
  Εξοδος:  η θέση του x στον πίνακα, αλλιώς 0

  for i = 1 to n
     if A[i] = x then 'βρέθηκε
          return i
     end if
  end for
  return 0 'δεν βρέθηκε
end function
```

Με πολλή τύχη: 1 επανάληψη
Αν γκαντέμης: n επαναλήψεις
Κατά μέσο όρο;

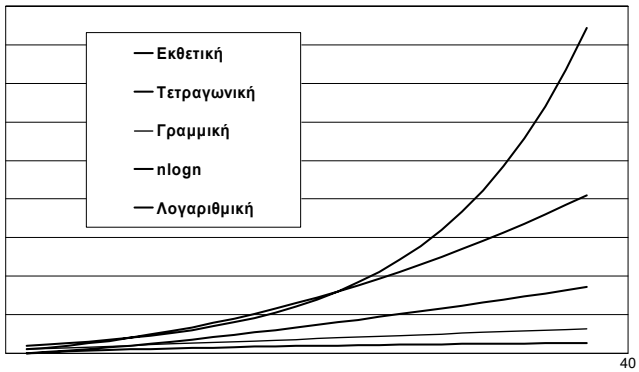
38

Ρυθμοί Ανάπτυξης

Σταθερή	O(c)	Ο αλγόριθμος τερματίζει σε σταθερό χρόνο ανεξαρτήτως του μεγέθους της εισόδου
Λογαριθμική	O(lgn)	Δυαδική αναζήτηση
Γραμμική	O(n)	Σειριακή αναζήτηση
-	O(n.lgn)	
Πολυωνυμική	$O(n^k)$	Για k=2, τετραγωνική. Για k=3, κυβική.
Εκθετική	$O(2^n)$	Αργοί αλγόριθμοι

39

Εξέλιξη Πολυπλοκότητας



40

Τιμές Κατηγοριών

n	$\log n$	$\sqrt{n}$	n	$n\log n$	n^2	n^3	2^n
2	1	1.4	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2.8	8	24	64	512	256
16	4	4	16	64	256	4,096	65,536
32	5	5.7	32	160	1,024	32,768	4,294,967,296
64	6	8	64	384	4,096	262,144	1.84×10^{19}
128	7	11	128	896	16,384	2,097,152	3.40×10^{38}
256	8	16	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	23	512	4,608	262,144	134,217,728	1.34×10^{154}
1,024	10	32	1,024	10,240	1,048,576	1,073,741,824	1.79×10^{308}

41

Ταξινόμηση Συναρτήσεων Επιδόσεων

- Ποιες συναρτήσεις (που συναντούμε συχνά αναλύοντας αλγορίθμους) «απογειώνονται» πιο γρήγορα;

$$\lg\lg n \prec \lg n \prec \lg^2 n \prec n^{1/10} \prec \sqrt{n} \prec n$$

$$n \prec n\lg n \prec n^2 \prec 2^n \prec n! \prec 2^{2^n} \prec 2^{n!}$$

42

Τυπολόγιο Αθροισμάτων

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

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

43

Πολυπλοκότητα Προβλήματος

- Κανονικά ένας αλγόριθμος χαρακτηρίζεται από την πολυπλοκότητά του.
- Μπορεί να αποδειχθεί ότι δεν είναι δυνατόν να υπάρξει κάποιος αλγόριθμος που να λύνει το πρόβλημα κάτω από μια συγκεκριμένη κατηγορία πολυπλοκότητας.
  - Αυτό είναι εγγενές χαρακτηριστικό του προβλήματος, και δεν εξαρτάται από κάποια λύση του.

44

Μέσοι Όροι Προθεμάτων εκδ. 1

```
subroutine μέσοςΌροςΠροθεμάτων1
  Είσοδος: πίνακας X[1..n]
  Εξοδος: πίνακας A[1..n]      Επαναλήψεις
  for i = 1 to n                          n
    s = 0
    for j = 1 to i                        i
      s = s + X[j]
    end for j
    A[i] = s / i
  end for i
end sub
```

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{n.(n+1)}{2} \approx 0.5n^2 = O(n^2)$$

45

Μέσοι Όροι Προθεμάτων έκδ. 2

```
subroutine μέσοςΌροςΠροθεμάτων2
     Είσοδος: πίνακας X[1..n]
     Εξοδος: πίνακας A[1..n]
     s = 0                    Επαναλήψεις
     for i = 1 to n                    n
          s = s + X[i]
          A[i] = s / i
     end for i
end sub
```

$$\sum_{i=1}^{n} 1 = n = O(n)$$

46

Μέγιστη Υποακολουθία

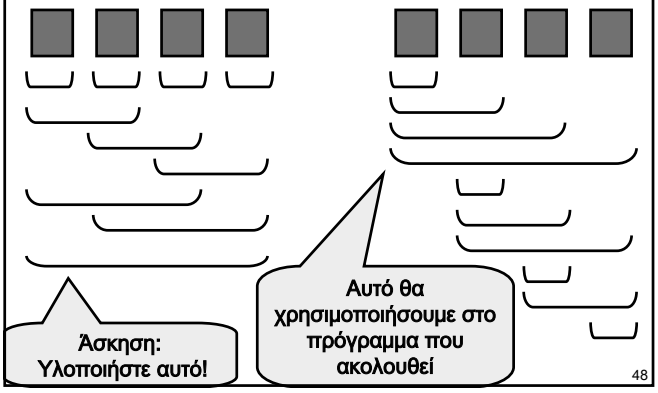
- Δίδεται πίνακας ακεραίων μεγέθους n και ζητείται να βρεθεί η μέγιστη τιμή ανάμεσα στα αθροίσματα των συνεχόμενων στοιχείων.*
- Παράδειγμα

Είσοδος:	4	-5	7	10
Υποακολουθίες:	4	-1	6	16
		-5	2	12
			7	**17**
				10

* Κατά σύμβαση το μέγιστο πρόθεμα είναι τουλάχιστον 0.

47

Απαρίθμηση Υποακολουθιών



48

Λύση 1 – O(n³)

Αρχή υποακολουθίας

Τέλος υποακολουθίας

```
public static int maxSubSum1(int [] a) {
  int maxSum = 0;
  for (int i = 0; i < a.length; i++)
     for (int j = i; j < a.length; j++) {
          thisSum = sumFromTo(i, j)
          if (thisSum > maxSum)
               maxSum = thisSum;
     } return maxSum;
}
```

Υπολογισμός μερικού αθροίσματος

Ανάλυση Λύσης 1

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j}1=\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)=\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}j-\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}i+\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}1=$$

$$=\sum_{i=0}^{n-1}\frac{n^2-n-i^2+i}{2}-\sum_{i=0}^{n-1}(ni-i^2)+\sum_{i=0}^{n-1}(n-i)$$

$$=\frac{1}{2}\sum_{i=0}^{n-1}(n^2+n+i^2-i-2ni)$$

$$=\frac{1}{2}\left(n^2\sum_{i=0}^{n-1}1+n\sum_{i=0}^{n-1}1+\sum_{i=0}^{n-1}i^2-\sum_{i=0}^{n-1}i-2n\sum_{i=0}^{n-i}i\right)$$

$$=\frac{1}{2}\left(n^3+n^2+\frac{2n^3-3n^2+n}{6}-\frac{n^2-n}{2}-2n^2\right)=\frac{4n^3-6n^2+2n}{6}$$

Λύση 2 - O(n²)

Αρχή υποακολουθίας

Τέλος υποακολουθίας

```
public static int maxSubSum2(int [] a) {
  int maxSum = 0;
  for (int i = 0; i < a.length; i++) {
     int thisSum = 0;
     for (int j = i; j < a.length; j++) {
          thisSum += a[j];
          if (thisSum > maxSum)
               maxSum = thisSum;
     } //end for j
  } return maxSum;
} //end maxPro2
```

Συσσώρευση του μερικού αθροίσματος

Λύση 3 - O(n)

Ένα πέρασμα

Τέλος υποακολουθίας

```
public static int maxSubSum3(int [] a) {
  int maxSum = 0, thisSum = 0;
  for (int j = 0; j < a.length; j++) {
     thisSum += a[j];
     if (thisSum > maxSum)
        maxSum = thisSum;
     else if (thisSum < 0) thisSum = 0;
  }
  return maxSum;
}
```


Εκτέλεση Λύσης 3

Εκτελέστε τον αλγόριθμο για:
3 -4 5 -2 1 -1 2 -7 6

4	-5	7	10	j	thisSum	maxSum
					0	0
⌣				0	4	4
⌣	⌣			1	-1	4
					0	
		⌣		2	7	7
		⌣	⌣	3	17	17

Δύσκολα Προβλήματα

- Εννοιολογικά δύσκολο: δεν έχουμε αλγόριθμο, επειδή δεν καταλαβαίνουμε καλά το πρόβλημα.
- Αναλυτικά δύσκολο: έχουμε αλγόριθμο που λύνει το πρόβλημα, αλλά δεν μπορούμε να υπολογίσουμε τις επιδόσεις του.
- Υπολογιστικά δύσκολο: έχουμε αλγόριθμο, αλλά είναι απελπιστικά αργός.
- Υπολογιστικά άλυτο: δεν υπάρχει αλγόριθμος που να λύνει το πρόβλημα.

Ιωάννης Γαβιώτης
gaviotis@aegean.gr
http://www.syros.aegean.gr/users/gaviotis/add

Πώς Αντιμετωπίζουμε τα Δύσκολα

- Όταν απαιτείται εκθετικός χρόνος και πάνω, ο χρόνος απογειώνεται ακόμη και για μικρές εισόδους.
- Προσεγγιστικοί αλγόριθμοι: κοντινές λύσεις
- Πιθανοτικοί αλγόριθμοι: βρίσκουν τη λύση τις περισσότερες φορές – όχι πάντα

55

Επίλογος

- Η ανάλυση του αλγορίθμου είναι χρήσιμη γιατί μας δείχνει πόσο καλός είναι.
  - Αν είναι κακός, τουλάχιστον το ξέρουμε.
- Αν το πρόβλημα δεν είναι εγγενώς δύσκολο, θα προσπαθήσουμε να βρούμε καλύτερο.
  - Προσπαθώντας με μικρές βελτιώσεις στον κώδικα, είναι σα να κάνουμε μαραγκοδουλειά χρησιμοποιώντας γυαλόχαρτο (αντί για πριόνι).

56

3
Αναδρομικοί & Παράλληλοι Αλγόριθμοι

Είδη Αλγορίθμων

- Αναδρομικοί (recursive): καλούν τον εαυτό τους
  - Για να λύσουν ένα πρόβλημα συγκεκριμένου μεγέθους, λύνουν το ίδιο πρόβλημα για μικρότερο μέγεθος
- Παράλληλοι: εκτελούνται σε υπολογιστές με πολλές μονάδες επεξεργασίας
  - Τα μερικά αποτελέσματα από κάθε μονάδα συνδυάζονται για τον υπολογισμό της λύσης
  - Δεν αλλάζει τάξη μεγέθους των επιδόσεων

58

Αναδρομή

- Ένα πρόβλημα μπορεί να επιλυθεί αναδρομικά όταν:
  - είναι γνωστό το τελικό βήμα, και
  - το πρώτο βήμα της γενικής λύσης του προβλήματος οδηγεί στο αρχικό πρόβλημα, αλλά λίγο πιο κοντά στο τελικό βήμα.
- Υλοποίηση σε γλώσσες προγραμματισμού
  - Υπορουτίνα που στο σώμα της καλεί τον εαυτό της.

59

Παραγοντικό

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n = \prod_{i=1}^{n} i$$

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

Έκδοση με επανάληψη

```
function efac (n)
  gin = 1
  for i = 1 to n
    gin = gin * i
  end for i
  return gin
end function
```

Έκδοση με αναδρομή

```
function rfac (n)
  if n = 1 then
    return 1
  else
    return n*rfac(n-1)
end function
```

60

Παράλληλο Παραγοντικό

```
function pgin (i, j) // i*(i+1)* … *j
  if i = j then return i
  else parbegin
     k = (i + j) / 2
     return pgin (i, k) * pgin (k+1, j)
  parend
```

5! = pgin(1,5)
= pgin(1,3)*pgin(4,5)
= (pgin(1,2)*pgin(3,3))*(pgin(4,4)*pgin(5,5))

61

Χρυσή Τομή

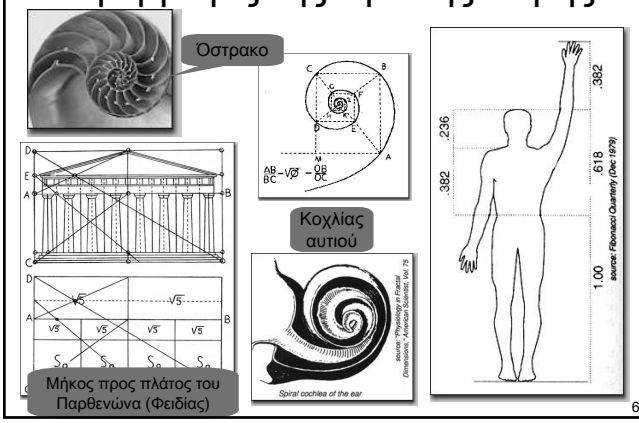
- Χωρίστε γραμμή σε δυο τμήματα, έτσι ώστε ο λόγος του όλου προς το μεγάλο τμήμα να ισούται με το μεγάλο προς το μικρό τμήμα.

φ = 1,6180339...

$$1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cdots}}}}$$

62

Εφαρμογές της Χρυσής Τομής



63

Λαγοί

- Έχεις δύο λαγούς. Από δύο λαγούς γεννιούνται κάθε μήνα άλλοι δύο από το δεύτερο μήνα της ζωής τους.
- Πώς αυξάνει ο πληθυσμός τους κάθε μήνα;
- Κάθε μήνα θα υπάρχουν όσοι λαγοί και τον προηγούμενο συν τα γεννητούρια των λαγών που ζούσαν και τον προ-προηγούμενο μήνα.

$$f(n)=\begin{cases}1 & n=0,1\\ f(n-1)+f(n-2) & n\geq 2\end{cases}$$

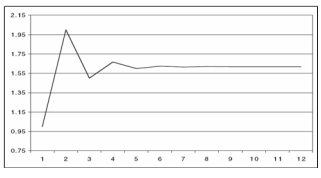
64

Αριθμοί Φιμπονάτσι

Ιταλός μαθηματικός γεννημένος το 1175

n	0	1	2	3	4	5	6	7	8
f(n)	1	1	2	3	5	8	13	21	34

Έκπληξη!
Ο ρυθμός αύξησης της ακολουθίας είναι **φ**.



```
function rfibo(n)
  if n<=1 then return 1
  else return rfibo(n-1) + rfibo(n-2)
```

65

Φιμπονάτσι με Επανάληψη

```
function efibo(n)
  past = 1;
  previous = 1;
  present = 1
  for i = 2 to n
    past = previous;
    previous = present
    present = previous + past
  end for i
  return present
end function
```

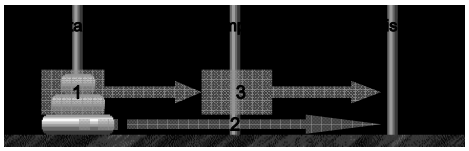
Αν και κομψός, ο αναδρομικός αλγόριθμος έχει εκθετικό κόστος $O(\varphi^n)$.

Η επαναληπτική εκδοχή του έχει γραμμικό κόστος O(n).

66

Πύργοι του Ανόι

```
function hanoi (n, start, finish, temp)
' Μετακινεί n δίσκους από το στύλο start στο
' finish, χρησιμοποιώντας τον temp ως ενδιάμεσο
if n > 0 then
1 hanoi (n-1, start, temp,   finish)
2 print ("από " +start + " βάλε στο " + finish)
3 hanoi (n-1, temp,  finish, start)  '3
endif
```



Δοκιμή

67

Διερεύνηση Ανόι

- Πόσες μετακινήσεις απαιτεί ο αλγόριθμος;

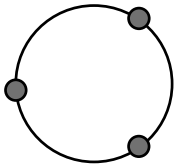
$$T(n) = 2T(n-1)+1 =$$
$$= 2[2T(n-2)+1]+1 =$$
$$= 4T(n-2)+1+2 =$$
$$= \cdots =$$
$$= 2^i T(n-i) +$$
$$+ (2^0 + 2^1 + \cdots + 2^{i-1}) =$$
$$= 2^i T(n-i) + \sum_{j=0}^{i-1} 2^j =$$
$$\overset{i=n}{=} 2^n T(0) + \sum_{j=0}^{n-1} 2^j =$$
$$= \sum_{j=0}^{n-1} 2^j = \frac{2^n - 1}{2-1} =$$
$$= 2^n - 1 = O(2^n).$$

68

Επαναληπτικός Αλγόριθμος για Πύργους του Ανόι

- Βάλε τους τρεις πασσάλους σε έναν κύκλο.
- Μετάφερε τον μικρότερο δίσκο στον επόμενο πάσσαλο ως προς τη φορά των δεικτών του ρολογιού, εκτός κι αν η προηγούμενη κίνηση ήταν η μεταφορά του μικρότερου δίσκου.
- Διαφορετικά, κάνε τη μοναδική άλλη νόμιμη κίνηση

Άσκηση: υλοποιήστε τον



69

4
Αφαιρετικοί Τύποι Δεδομένων

Ορισμός – Υλοποίηση

Τύποι Δεδομένων

Data types

- Ένας τύπος δεδομένων είναι ένα σύνολο τιμών και μια συλλογή λειτουργιών επί των τιμών αυτών.
  - Πχ, οι ακέραιοι αριθμοί (**`int`**) και οι πράξεις επ' αυτών (+, -, *, /, modulo, κλπ).
- Μπορούμε να ορίσουμε και νέους τύπους δεδομένων ή να ορίσουμε νέες λειτουργίες για υπάρχοντες τύπους δεδομένων.
  - Πχ, οι ημερομηνίες και οι λειτουργίες: σύγκριση, αφαίρεση (χρονικό διάστημα)

71

Αφαιρετικοί Τύποι Δεδομένων

Abstract data types

- Προδιαγραφή μιας δομής δεδομένων
  - Ορίζει ιδιότητες & λειτουργίες
    - Πχ, σε ημερομηνία, ιδιότητες είναι έτος, μήνας, ημέρα
    - Στις λειτουργίες καθορίζεται τι παίρνουν ως είσοδο (λίστα παραμέτρων) και τι επιστρέφουν.
      - Επίσης, τι σφάλματα παράγουν και υπό ποιες συνθήκες.
    - Περιγράφεται η σημασιολογία (**τι** κάνει – όχι **πώς** το κάνει).
  - Δεν καθορίζεται τρόπος υλοποίησης. Διότι:
    - Υπάρχουν διάφοροι τρόποι υλοποίησης ενός ΑΤΔ.
    - Εξαρτάται από γλώσσα προγραμματισμού, από προσδοκώμενη χρήση, κλπ.

72

Υλοποίηση ΑΤΔ

- Αποκρύπτεται ο τρόπος που αναπαριστάνεται η πληροφορία που περικλείει.
  - Μπορεί να μην αποκαλύπτονται ούτε καν οι αλγόριθμοι που χρησιμοποιεί.
  - Ο κατασκευαστής της υλοποίησης μπορεί να εξελίσσει το «εσωτερικό», χωρίς ο χρήστης του ΑΤΔ να χρειαστεί να αλλάξει το πρόγραμμά του.
- Παράδειγμα: αναπαράσταση κλάσματος
  - δύο ακέραιοι, πχ ar = 3, par = 4
  - μια συμβολοσειρά, πχ val = “3/4”

73

Ευρετήριο Ονομάτων

- Πρέπει να υποστηρίζει λειτουργίες:
  - εντοπισμός (υπάρχει το τάδε όνομα;)
  - εισαγωγή
  - διαγραφή
- Δεν έχουν έννοια οι επιδόσεις, γιατί δεν υπάρχει υλοποίηση.
  - Μπορεί να υπάρχουν απαιτήσεις επιδόσεων.
  - Μπορεί να υπάρχει αναμενόμενη χρήση, πχ ποια είναι η πιο κοινή λειτουργία;

74

ΑΤΔ στη Java

- Φτιάχνουμε μια διεπαφή (**`interface`**) που ορίζει την προδιαγραφή των λειτουργιών.
- Η λειτουργικότητα του ορίζεται από μια κλάση που υλοποιεί (**`implements`**) το **`interface`**.
- Τα στιγμιότυπα του ΑΤΔ χρησιμοποιούνται από μια κλάση που εισάγει (**`imports`**) μια κλάση που υλοποιεί τον ΑΤΔ.

75

Παράδειγμα ΑΤΔ

- Μοντέλο για χρηματιστηριακές πράξεις (αγοραπωλησίες μετοχών)
  - Τα δεδομένα που αποθηκεύονται είναι οι εντολές αγοράς / πώλησης.
  - Οι λειτουργίες που υποστηρίζονται είναι:
    - **`order buy(stock, shares, price)`**
    - **`order sell(stock, shares, price)`**
    - **`void cancel(order)`**
  - Εξαιρέσεις:
    - Αγορά / Πώληση ανύπαρκτης μετοχής
    - Ακύρωση ανύπαρκτης εντολής

76

Πώς «Βλέπουμε» έναν ΑΤΔ

- Ως χρήστες
  - Μας ενδιαφέρει ο ΑΤΔ να υποστηρίζει τις λειτουργίες που μας είναι απαραίτητες
  - Μας ενδιαφέρει οι λειτουργίες να εκτελούνται γρήγορα.
- Ως κατασκευαστές
  - Μας ενδιαφέρει η εσωτερική δομή των δεδομένων.
  - Μας ενδιαφέρει η υλοποίηση αποδοτικών αλγορίθμων που να εκτελούν τις λειτουργίες.

77

Υλοποίηση ΑΤΔ

```
public class BankAccount {
  private double balance;
  public BankAccount() {
  }
  public boolean deposit(double amount) {
    if (amount > 0.0d) {
      balance += amount; //καταθεση
      return true; }
    else return false;
  }

  public boolean withdraw(double amount) {
    if ((amount>0) && (amount <= balance)) {
      balance -= amount; //αναληψη
      return true; }
    else return false;
  }

  public double getBalance() { return balance; }
}
```

```
/**
 * Καταθέτει το ποσόν στον λογαριασμό
 *
 * @param θετικός αριθμός-αντιστοιχεί στο ποσό της κατάθεσης
 * @return Αληθές αν έγινε η κατάθεση, ψευδές αν δεν έγινε.
 */
```

78

Χρήση ΑΤΔ

```
public class BankAccountDemo {
  public static void main(String[] args) {
    BankAccount myAccount = new BankAccount();
    boolean ok;
    ok = myAccount.deposit(100.0);
    ok = myAccount.withdraw(50.0);
    ok = myAccount.withdraw(70.0);
    ok = myAccount.deposit(-200.0);
    System.out.println(myAccount.getBalance());
  }
}
```

79

Ο-Ο για τη υλοποίηση ΑΤΔ

- Ο αντικειμενοστρεφής προγραμματισμός είναι κατάλληλο εργαλείο για την υλοποίηση ΑΤΔ:
  - Ενθυλακώνει την αναπαράσταση των δεδομένων (**`private`** / **`protected`**).
  - Παρέχει πρόσβαση στις λειτουργίες μέσω της δημόσιας διεπαφής (**`public`**).
  - Απαγορεύοντας την άμεση πρόσβαση στα δεδομένα του ΑΤΔ, επιβάλει την ορθότητα της κατάστασης.
  - Διαχωρίζει προδιαγραφή από υλοποίηση (**`interface`** – **`implements`**).

80

Γιατί Java;

- Ανεξάρτητη υπολογιστικής πλατφόρμας
- Υποστηρίζει δείκτες (pointers)
  - Όχι αριθμητική δεικτών που είναι επικίνδυνη
- Επίπεδα ορατότητας δεδομένων
  - Public, protected, private
- Ενσωματωμένη διαχείριση μνήμης
  - Συλλογή απορριμμάτων
- Διαχείριση σφαλμάτων και εξαιρέσεων

81

5
Πίνακες

Λειτουργίες – Χρήση

Πίνακες

- Η πιο βασική δομή αποθήκευσης δεδομένων
  - Υποστηρίζεται από όλες τις γλώσσες προγραμματισμού.
- Συνεχόμενες θέσεις στην κύρια μνήμη
  - Προσπέλαση μέσω δείκτη
  - Στατική δομή – το μέγεθος ορίζεται εξαρχής και δεν αλλάζει.

Διεύθυνση βάσης

Χώρος για κάθε στοιχείο

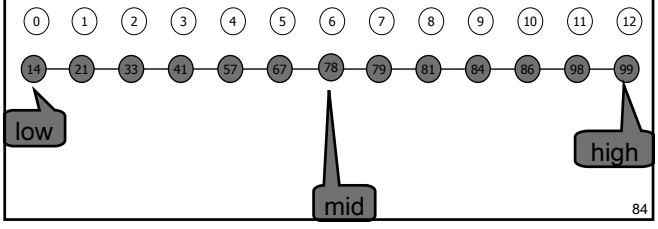
Δείκτης	Τιμή
0	43
1	7
2	13
3	8
4	31
5	13
6	43

83

Δυαδική Αναζήτηση

Binary Search

- Σε ποια θέση βρίσκεται η τιμή 45;
- Διατηρούνται τρεις δείκτες low, high, και mid → μεσαίος



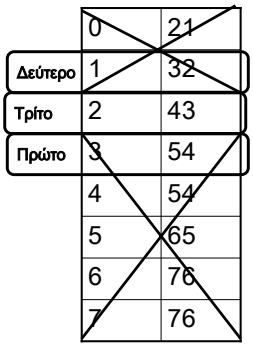
84

Κώδικας Δυαδικής Αναζήτησης

```
'αναζητεί το x στις θέσεις a[l] έως a[h]
'και επιστρέφει τη θέση που βρέθηκε
'αν δεν βρεθεί επιστρέφει NOT_FOUND
function binsrch (l, h, x)
  m = (h + l) / 2 'ακέραια διαίρεση
  if x = a[m] then return (m) 'βρέθηκε
  else if l = h then return (NOT_FOUND)
  else
    if x > a[m] then binsrch (m+1, h, x)
                else binsrch (l, m-1, x)
```

85

Ανάλυση Δυαδικής Αναζήτησης



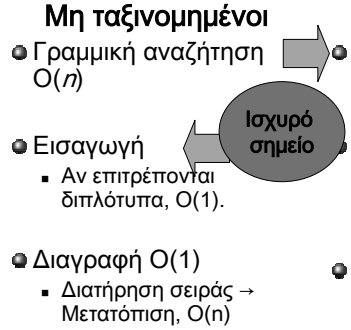
- Αναζητούμε την τιμή 39.
- Με κάθε προσπάθεια, υποδιπλασιάζουμε την περιοχή αναζήτησης.

$$T(n) = 1 + T(\frac{n}{2}) = 2 + T(\frac{n}{4}) \ldots$$

$$= i + T(\frac{n}{2^i}) \underset{i=\lg n}{\overset{n=2^i}{=}} \lg n + T(1)$$

86

Λειτουργίες επί Πινάκων



87

Κόσκινο του Ερατοσθένη

```
'βρίσκει τους πρώτους αριθμούς μέχρι το n
declare a as array[ n ] of boolean
for i = 2 to n
  a[i] = PRWTOS 'αρχικά όλοι θεωρούνται πρώτοι
endfor

for i = 2 to n
  if a[i]=PRWTOS then       'αν είναι πρώτος,
    for j = i+i to n step i 'όλα τα πολλαπλάσιά του
      a[j] = OXI_PRWTOS     'ΔΕΝ είναι πρώτοι
    endfor
  endif
endfor
```

Βελτιστοποίηση2: είναι ανάγκη μέχρι το *n*;

Βελτιστοποίηση1: αγνόησε τους άρτιους

88

Μειονεκτήματα Πινάκων

- Για να αλλάξει το μέγεθός τους απαιτείται να ξαναφτιαχτούν.
  - Όσο είναι άδεια, έχουμε σπατάλη.
  - Όταν γεμίσουν, πρέπει να δεσμευτεί άλλος χώρος, να γίνει αντιγραφή και να αποδεσμευτεί ο αρχικός χώρος.
- Όχι καλές επιδόσεις για την αναζήτηση.

89

Διεπαφή ΑΤΔ 'Πίνακας'

```
public interface ArrayInterface {

/**
 * @param element το στοιχείο που ψάχνω να βρω
 * @return η θέση του στον πίνακα, διαφορετικά (-1)
 */
 public int isIn (Object element);

/**
 * @param element στοιχείο που θέλω να προσθέσω
 * @throws Exception αν ο πίνακας είναι γεμάτος
 */
 public void add (Object element) throws Exception ;

/**
 * @param element στοιχείο που θέλω να αφαιρέσω
 */
 public void remove (Object element);
}
```

Φτιάξτε κλάση που να υλοποιεί τη διεπαφή

90

Πακέτο `java.util.Vector`

- Διαχειρίζεται πίνακες μεταβαλλόμενου μεγέθους και ετερογενών περιεχομένων.
- `Vector v = new Vector (100,20)`

Αρχικό μέγεθος

Αύξηση

Δείκτης	Αναφορά
0	
1	
2	
3	
4	null

Νίκος

-73.6

Άρης

321

- Όταν γεμίσει, μεγαλώνει.

91

Μέθοδοι για Vector

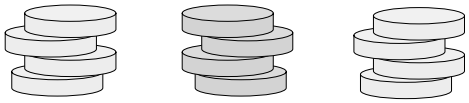
- **`addElement("X")`** στο τέλος
- **`insertElementAt("Y", 3)`** μετακινώντας τα επόμενα μια θέση προς τα κάτω
- **`removeElement("X")`** διαγράφει την πρώτη εμφάνιση του X που θα βρει
- **`removeElementAt(3)`**
- Συρρίκνωση με **`trimToSize()`**
- Αναζήτηση με **`contains("X")`**

Δείτε τεκμηρίωση για `java.util.Vector`

92

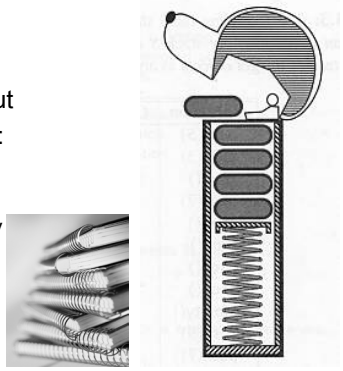
6
Στοίβες

Stacks



Πώς Λειτουργεί μια Στοίβα

- Βγαίνει πρώτος, αυτός που μπήκε τελευταίος.
- LIFO = Last In First Out
- Παραδείγματα χρήσης:
  - Πιάτα στο ντουλάπι
  - Οπισθοδρόμηση σε επισκέψεις ιστοσελίδων
  - Αναίρεση / Επανάληψη επεξεργασίας



94

Στοίβα ως ΑΤΔ

- Αποθηκεύει κάθε είδους αντικείμενο
- Κύριες λειτουργίες:
  - void **push**(object): εισάγει στοιχείο στη στοίβα
  - Object **pop**(): αφαιρεί το στοιχείο που μπήκε τελευταίο
  - boolean **isEmpty**(): είναι η στοίβα άδεια;
- Βοηθητικές λειτουργίες:
  - Object **top**(): επιστρέφει το τελευταίο στοιχείο χωρίς να το αφαιρεί από τη στοίβα
  - int **size**(): επιστρέφει το πλήθος των αποθηκευμένων στοιχείων
- Εξαίρεση εγείρεται όταν προσπαθήσουμε να αφαιρέσουμε από άδεια στοίβα.

95

Διεπαφή Στοίβας

```
public interface StackInterface {

/**
 * @param το στοιχείο που εισάγεται στην κορυφή της στοίβας
 */
 public void push(Object element);

/**
 * @return το στοιχείο που αφαιρέθηκε από την στοίβα
 */
 public Object pop();

/**
 * @return αν είναι άδεια η στοίβα
 */
 public boolean isEmpty();
}
```

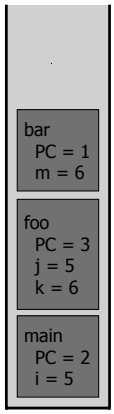
Υλοποιήστε `top` και `size` χρησιμοποιώντας τις βασικές

96

Χρήση: Κλήσεις Μεθόδων

- Το περιβάλλον εκτέλεσης διατηρεί την αλυσίδα των ενεργών κλήσεων σε μια στοίβα.
- Όταν καλείται μια μέθοδος, εισάγεται στη στοίβα μια εγγραφή με
  - Τοπικές μεταβλητές και αποτέλεσμα
  - Μετρητής προγράμματος (PC): ποια είναι η εντολή που εκτελείται
- Όταν μια μέθοδος επιστρέφει, η εγγραφή της αφαιρείται από τη στοίβα και η εκτέλεση συνεχίζεται στη μέθοδος που είναι στην κορυφή της στοίβας.

```
main() {
  int i = 5;
  foo(i);
  }

foo(int j) {
  int k;
  k = j+1;
  bar(k);
  }

bar(int m) {
  ...
  }
```



97

Υλοποίηση Στοίβας

- Ένας **πίνακας** S και μια μεταβλητή t που δείχνει στο στοιχείο του πίνακα που αποθηκεύει την κορυφή της στοίβας. Αρχικά t = -1.
- Ο πίνακας ξεκινά να γεμίζει από το στοιχείο με δείκτη 0.
- Στην κατασκευή της στοίβας, δηλώνεται το μέγεθος N του πίνακα.

Algorithm size():
  return $t + 1$

Algorithm isEmpty():
  return $(t < 0)$

Algorithm top():
  if isEmpty() **then**
    throw a StackEmptyException
  return $S[t]$

Algorithm push(o):
  if size() $= N$ **then**
    throw a StackFullException
  $t \leftarrow t + 1$
  $S[t] \leftarrow o$

Algorithm pop():
  if isEmpty() **then**
    throw a StackEmptyException
  $e \leftarrow S[t]$.
  $S[t] \leftarrow$ **null**
  $t \leftarrow t - 1$
  return e

Επιδόσεις & Περιορισμοί

- Επιδόσεις
  - Έστω n το πλήθος των στοιχείων στη στοίβα
  - Απαιτούμενος χώρος $O(n)$
  - Κάθε λειτουργία τρέχει σε χρόνο $O(1)$
- Περιορισμοί
  - Το μέγιστο μέγεθος της στοίβας πρέπει να δηλωθεί εκ των προτέρων (κατά την κατασκευή της) και δεν μπορεί να αλλάξει.
  - Αν προσπαθήσουμε να εισάγουμε στοιχείο σε μια πλήρη στοίβα, θα προκληθεί εξαίρεση.

99

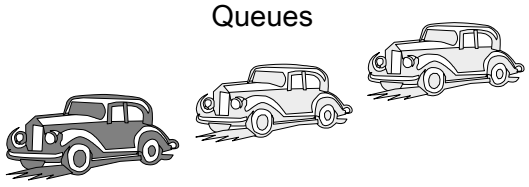
Αναπτυσσόμενη Στοίβα

- Όταν ζητείται προσθήκη στοιχείου και η στοίβα είναι γεμάτη, αντί να εγερθεί εξαίρεση αντικαθιστούμε τον πίνακα με έναν μεγαλύτερου μεγέθους.
  - Ο παλιός πίνακας πετιέται στα σκουπίδια (garbage collection)
- Πόσο μεγαλύτερο:
  - Αύξησε το μέγεθος κατά c θέσεις (σταθερά)
  - Διπλασίασε το μέγεθος
- Για συμμετρία:
  - Αν η πληρότητα πέσει κάτω από πχ 50%, να μειωθεί το μέγεθος.

Algorithm *push*(o)
  if $t = S.length - 1$ **then**
    $A \leftarrow$ new array of size …
    for $i \leftarrow 0$ **to** t **do**
      $A[i] \leftarrow S[i]$
    $S \leftarrow A$
  $t \leftarrow t + 1$
  $S[t] \leftarrow o$

7
Ουρές

Queues



Πώς Λειτουργεί μια Ουρά

- Βγαίνει πρώτος, αυτός που μπήκε πρώτος.
- FIFO = First In First Out
- Παραδείγματα χρήσης:
  - Σειρά αναμονής για εξυπηρέτηση
  - Αυτοκίνητα σε πάρκινγκ με μια είσοδο/έξοδο

102

Ουρά ως ΑΤΔ

- Εισαγωγές και διαγραφές ακολουθούν FIFO.
- Μέγεθος ουράς απεριόριστο
- Τύπος στοιχείων οποιοσδήποτε
- Βασικές λειτουργίες:
  - **enqueue** (object): εισάγει στοιχείο στο πίσω μέρος της ουράς
  - Object **dequeue**(): αφαιρεί και επιστρέφει το στοιχείο στην κορυφή της ουράς

103

Λειτουργίες σε Ουρές

- Βοηθητικές λειτουργίες
  - Object **front**(): επιστρέφει το στοιχείο στην κορυφή, χωρίς να το αφαιρεί
  - int **size**(): πλήθος στοιχείων
  - boolean **isEmpty**(): είναι άδεια η ουρά;
- Εξαιρέσεις
  - Προσπάθεια εκτέλεσης της dequeue ή της front σε μια άδεια ουρά προκαλεί έγερση της EmptyQueueException

104

Διεπαφή Ουράς

```
public interface Queue {
  public int size();
  public boolean isEmpty();
  public Object front()
                  throws EmptyQueueException;
  public void enqueue(Object o);
  public Object dequeue()
                   throws EmptyQueueException;
  }
```

105

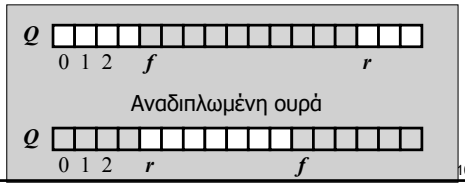
Εφαρμογές για Ουρές

- Λίστες αναμονής
- Προσπέλαση σε διαμοιραζόμενους πόρους
  - Παράδειγμα: δικτυακός εκτυπωτής που εξυπηρετεί πολλούς χρήστες.

106

Ουρά με Πίνακα

- Πίνακας μεγέθους ***N*** με κυκλικό τρόπο
- Μεταβλητές διατηρούν κορυφή και οπίσθια
  - ***f*** δείχνει στο πρώτο στοιχείο
  - ***r*** δείχνει αμέσως μετά το τελευταίο στοιχείο
- Η θέση ***r*** του πίνακα παραμένει κενή.
- Αρχικά ***f = r*** άδεια ουρά



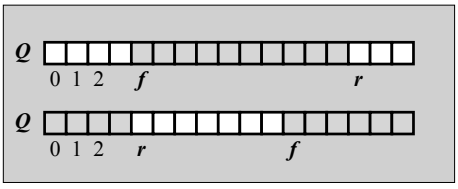
107

Τρόπος Λειτουργίας

- Χρησιμοποιούμε τον τελεστή modulo (υπόλοιπο ακέραιας διαίρεσης)

Algorithm ***size***()
  return ($N - (f - r)$) mod N

Algorithm ***isEmpty***()
  return ($f = r$)



108

Εισαγωγή / Διαγραφή σε Ουρά

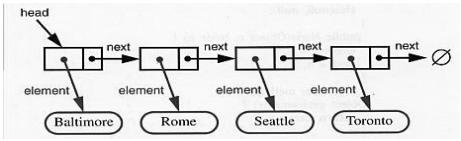
- Παρατηρήστε ότι μια θέση του πίνακα παραμένει πάντα κενή.

Algorithm *enqueue*(*o*)	**Algorithm** *dequeue*()
if *size*() = *N* − 1 **then**	**if** *isEmpty*() **then**
throw *FullQueueException*	**throw** *EmptyQueueException*
else	**else**
Q[*r*] ← *o*	*o* ← *Q*[*f*]
r ← (*r* + 1) mod *N*	*f* ← (*f* + 1) mod *N*
	return *o*

109

8
Συνδεδεμένες Λίστες

Απλά & διπλά συνδεδεμένες



Απλά Συνδεδεμένη Λίστα (ΑΣΛ)

Linked list

- Αλυσίδα κόμβων που ο καθένας παραπέμπει στον επόμενό του
- Ο κάθε κόμβος έχει δύο τμήματα:
  - **`data`**: περιέχει είτε τα δεδομένα, είτε μια αναφορά προς τα δεδομένα.
  - **`next`**: περιέχει μια αναφορά προς τον επόμενο κόμβο της λίστας.
    - Ο τελευταίος κόμβος το έχει **`null`**.
- Χρειάζεται να διατηρούμε μια αναφορά προς τον πρώτο κόμβο της λίστας.

111

Χαρακτηριστικά Λιστών

- Καλύτερη χρησιμοποίηση (utilization) χώρου
  - Εκχωρείται όσος χρησιμοποιείται, αλλά
  - χρειάζεται χώρος για αποθήκευση συνδέσμου.
- Καθυστέρηση εξαιτίας δυναμικής δέσμευσης και αποδέσμευσης χώρου
  - Στους πίνακες η δέσμευση γίνεται μια φορά στην αρχή και η αποδέσμευση στο τέλος.
    - Στους αναπτυσσόμενους / συρρικνούμενους υπάρχει κόστος διαχείρισης, όταν το μέγεθος μεταβάλλεται σημαντικά

112

Πότε Χρησιμοποιούμε Λίστες;

- Όταν ενδιαφερόμαστε κυρίως να διαπερνούμε γραμμικά μια συλλογή στοιχείων
  - Αντιπαράθεση: τυχαία προσπέλαση στο i-οστό στοιχείο καλύτερα πίνακας
- Όταν θέλουμε να αλλάζουμε συχνά τη σειρά των στοιχείων και η μετακίνησή τους κοστίζει.
- Όταν θέλουμε να διατηρούμε πολλαπλές ταξινομήσεις των ίδιων στοιχείων
  - Πχ ταξινόμηση υπαλλήλου με ΑΜ, επώνυμο & όνομα, τμήμα, κλπ

113

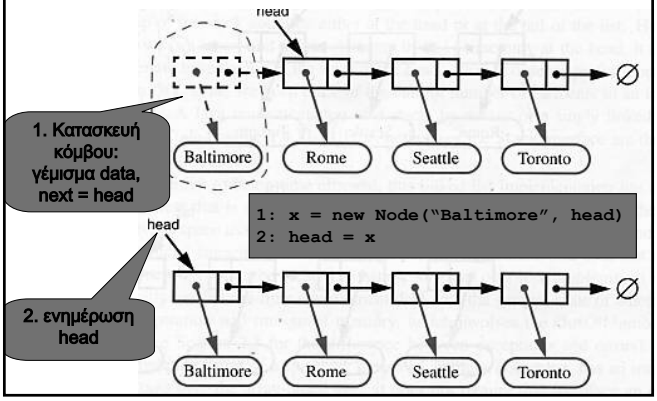
Υλοποίηση Κόμβου

Κυκλική αναφορά: ένας κόμβος περιέχει δεδομένα (data) και μια αναφορά προς έναν άλλο κόμβο.

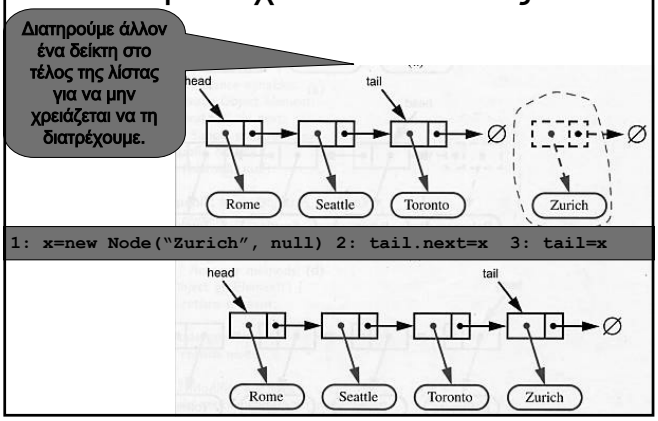
```
public class Node {
  private Object data;
  private Node next;

  public Node(Object d, Node n) {
    data = d; next = n; }

  public Node() {
    this(null, null); }

  //μέθοδοι get και set
}
```

114

Ένθεση Στοιχείου στην Αρχή ΑΣΛ



Ένθεση Στοιχείου στο Τέλος ΑΣΛ



Αναζήτηση σε ΑΣΛ

```
function efind(list l, element x) {
  while (l <> null) //επανάληψη
    if (l.data = x) return true
    else l = l.next //επόμενος κόμβος
  return false }
```

```
function rfind(list l, element x)
  if (l = null) return false
  elseif (l.data = x) return true
  else rfind(l.next, x) //αναδρομή
```


Συνένωση ΑΣΛ

Concatenation: Θέτουμε ως επόμενο του τελευταίου κόμβου της μιας τον αρχικό κόμβο της άλλης.

```
function concat (list x, list y) {
  if (x = null) then x = y // κενή λίστα
  else {
    t = x
    while (t.next <> null) // διαπέραση
      t = t.next
    t.next = y // τελευταίος κόμβος
  }
}
```


Ταξινομημένες ΑΣΛ

- Σειριακή αναζήτηση
  - Δεν συμφέρει να χρησιμοποιήσουμε δυαδική
- Ένθεση κόμβου
  - Πρέπει να διατηρούμε και τον προηγούμενο
- Συγχώνευση (merging) δύο ταξινομημένων λιστών
  - Αρχικά το αποτέλεσμα είναι μια κενή λίστα.
  - Συγκρίνουμε τους πρώτους κόμβους της καθεμιάς. Τον μικρότερο τον αφαιρούμε από την λίστα που ανήκει και τον εισάγουμε στο τέλος του αποτελέσματος.
  - Όταν εξαντληθεί η μια λίστα, κολλάμε τα υπόλοιπα στοιχεία της άλλης λίστας στο αποτέλεσμα.

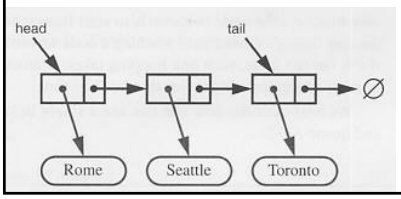
Επιδόσεις ΑΣΛ

- Εισαγωγή στην αρχή O(1)
- Διαγραφή στην αρχή O(1)
- Εισαγωγή στο τέλος O(1)
  - Εάν διατηρούμε για τη λίστα τον αρχικό και τον τελικό κόμβο της
- Διαγραφή στο τέλος O(1)
  - Φτάνει να μην χρειάζεται να διατηρούμε τον τελικό κόμβο της λίστας - αλλιώς απαιτείται διαπέραση O(n)
- Αναζήτηση στοιχείου O(n)

Ιωάννης Γαβιώτης
gaviotis@aegean.gr
http://www.syros.aegean.gr/users/gaviotis/add

Στοίβες & Ουρές με ΑΣΛ

- Στοίβα: Στο head της λίστας είναι η κορυφή της στοίβας - όλες οι λειτουργίες σε O(1)
- Ουρά: Χρησιμοποιούμε την παραλλαγή της ΑΣΛ με δείκτες στον πρώτο και στον τελευταίο κόμβο της. Στο head της λίστας γίνονται οι διαγραφές και στο tail οι ενθέσεις - όλες οι λειτουργίες σε O(1)



121

Επίδοση Δυαδικής Αναζήτησης σε ΑΣΛ (χάριν παραδείγματος)

- Για να ψάξουμε λίστα μήκους n, πρέπει να φτάσουμε στο μεσαίο της στοιχείο, που απαιτεί n/2 πράξεις, και έπειτα το μέγεθος της λίστας προς αναζήτηση είναι n/2.
- Το κόστος της δυαδικής αναζήτησης σε ΑΣΛ ισούται με της γραμμικής !

$$T(n) = T(\frac{n}{2}) + \frac{n}{2} = T(\frac{n}{4}) + \frac{n}{4} + \frac{n}{2} = \cdots =$$
$$= T(\frac{n}{2^i}) + \sum_{j=1}^{i} \frac{n}{2^j} \underset{i=\lg n}{\overset{n=2^i}{=}} T(1) + n\sum_{j=1}^{i} (\frac{1}{2})^j \approx$$
$$\approx n(\frac{1-(\frac{1}{2})^{i+1}}{1-\frac{1}{2}} - 1) = n - 1$$

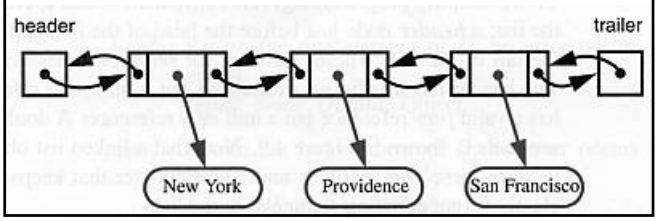
Αναστροφή ΑΣΛ

```
static Node reverse (Node x) {
  Node t, y = x, r = null;
  while (y != null) {
     t = y.next;
     y.next = r;
     r = y;
     y = t;
  }
  return r;
}
```

Απαιτείται να διατηρούμε συνδέσμους προς τρεις συνεχόμενους κόμβους:
***r** δείχνει προς την ήδη ανεστραμμένη λίστα*
***y** το τμήμα της λίστας που έπεται*
***t** ο δεύτερος κόμβος της μη ανεστραμμένης λίστας*

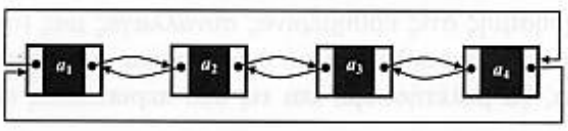
Διπλά Συνδεδεμένες Λίστες (ΔΣΛ)

- Χρήση για διπλοουρές (dequeues): Ένθεση & διαγραφή και από τα δύο άκρα
- Διαπέραση και προς τις δύο κατευθύνσεις.
- Ειδικοί «άδειοι» κόμβοι για αρχή / τέλος



Κυκλικές Λίστες

- Παραλλαγή διπλά διασυνδεδεμένης λίστας
- Δεν χρησιμοποιούμε ειδικούς κόμβους αρχής / τέλους.



125

9
Ακολουθίες

Διανύσματα & Λίστες

Ακολουθία ως ΑΤΔ

- Μια συλλογή στοιχείων στα οποία έχει επιβληθεί κάποια σειρά.
  - Μέσω θέσης (rank): 1ος 2ος, …, n-οστός
  - Μέσω σχέσης: προηγούμενος / επόμενος
- Διάνυσμα (μονοδιάστατος πίνακας): συλλογή που τα στοιχεία της προσπελαύνονται μέσω θέσης
- Λίστα: συλλογή που τα στοιχεία της προσπελαύνονται μέσω σχέσης

127

Διεπαφή Διανύσματος

- `Object elementAtRank(rank)`
- `void replaceAtRank(rank, element)`
- `void insertAtRank(rank, element)`
- `Object removeAtRank(rank)`

- Όλες εγείρουν εξαίρεση InvalidRank αν η θέση δεν υπάρχει.

128

Διεπαφή Λίστας

- Position first()
- Position last()
- boolean isFirst(p)
- boolean isLast(p)
- Position before(p)
- Position after(p)
- replaceElement(p,e)
- insertFirst(e)
- insertLast(e)
- insertBefore(p,e)
- insertAfter(p,e)
- remove(p)

Position είναι ένας προσδιορισμός του χώρου που ενθυλακώνει το στοιχείο, πχ ο κόμβος.

129

Επιδόσεις Υλοποιήσεων Ακολουθίας

Λειτουργίες	Πίνακας	Λίστα
size, isEmpty	O(1)	O(1)
atRank, rankOf, elementAtRank	O(1)	O(n)
first, last, before, after	O(1)	O(1)
replaceElement	O(1)	O(1)
replaceAtRank	O(1)	O(n)
insertAtRank, removeAtRank	O(n)	O(n)
insertFirst, insertLast	O(1)	O(1)
insertAfter, insertBefore	O(n)	O(1)
remove	O(n)	O(1)

130

Άσκηση

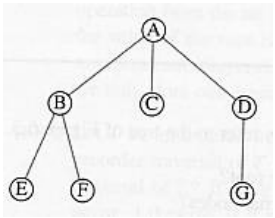
- *n* παιδιά κάθονται κυκλικά και περνούν το καθένα στο διπλανό του (κατά τη φορά του ρολογιού) μια ‘πατάτα’. Όταν η πατάτα μεταβιβαστεί *m* φορές, το παιδί που κρατάει την πατάτα, αποχωρεί και ο κύκλος κλείνει. Αυτός που μένει τελευταίος, κερδίζει.

- Πίνακας ή λίστα είναι η καλύτερη δομή;

131

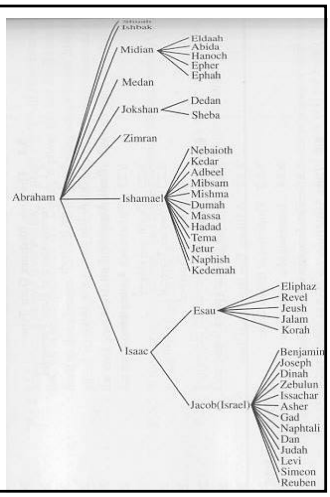
10
Δέντρα

Δυαδικά Δέντρα Αναζήτησης



Ορισμοί

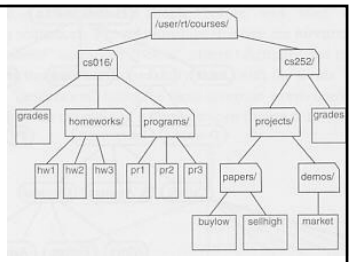
- Ιεραρχικός
- Μη γραμμικός τύπος δεδομένων
- Γονέας – Παιδιά
- Αδέλφια: κόμβοι με κοινό πατέρα
- Πρόγονοι – Απόγονοι

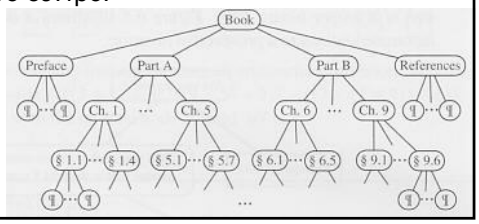


Μερικοί από τους απογόνους του Αβραάμ, Γένεση κεφ.25-26

Ορολογία

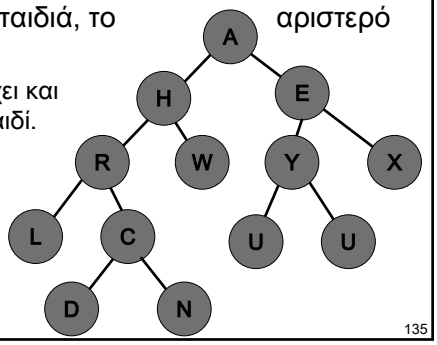
- Ρίζα: κόμβος που δεν έχει γονέα
- Φύλλα: κόμβοι χωρίς παιδιά
- Ταξινομημένο δέντρο: υπάρχει σειρά στα παιδιά





Δυαδικό Δέντρο

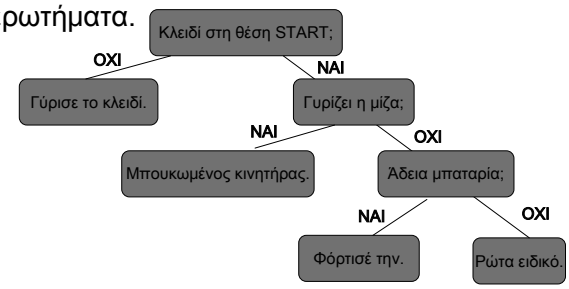
- Ταξινομημένο δέντρο, όπου κάθε κόμβος έχει *το πολύ* δύο παιδιά, το αριστερό και το δεξί.
  - Μπορεί να έχει και ένα μόνον παιδί.
- Σε ένα γνήσιο δυαδικό δέντρο κάθε κόμβος έχει 0 ή 2 παιδιά.



Δέντρα Αποφάσεων

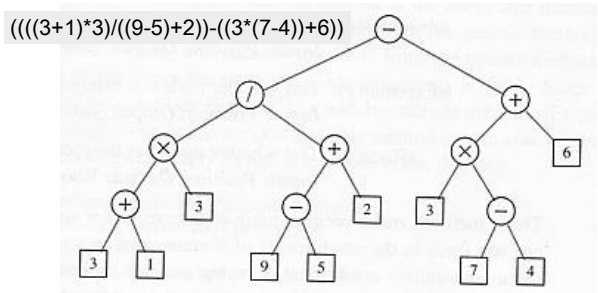
Decision trees

- Γνήσια δυαδικά δέντρα που προκύπτουν από τις απαντήσεις ΝΑΙ – ΟΧΙ σε διαδοχικά ερωτήματα.



Αριθμητικές Παραστάσεις

- Εσωτερικοί κόμβοι είναι πράξεις.
- Εξωτερικοί κόμβοι (φύλλα) είναι τιμές.



Μέθοδοι Δέντρων

- data(): επιστρέφει τα δεδομένα ενός κόμβου
- root(): επιστρέφει τη ρίζα ενός δέντρου
- parent(): επιστρέφει τον πατέρα του κόμβου
- children(): επιστρέφει τα παιδιά του κόμβου
  - Για δυαδικά δέντρα left(), right()
- isInternal(), isExternal(), isRoot(): βοηθητικές
- size(), elements(): βοηθητικές

Ιωάννης Γαβιώτης
gaviotis@aegean.gr
http://www.syros.aegean.gr/users/gaviotis/add

Απαριθμητής

Iterator

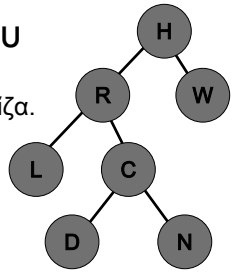
- Η μέθοδος children() δεν επιστρέφει ένα αποτέλεσμα, αλλά ένα 'σύνολο' τιμών.
- Ορίζεται να είναι απαριθμητής, δηλαδή όταν κληθεί, επιστρέφει μια ακολουθία παιδιών.
- Διατρέχουμε με τη hasNext()

```
PositionIterator ch = t.children(n);
  while ch.hasNext() {
      //διαπερνά τα παιδιά του n
  }
```

139

Βάθος Κόμβου

- Η απόστασή του από τη ρίζα.
- Ισούται με το πλήθος των προγόνων του κόμβου.

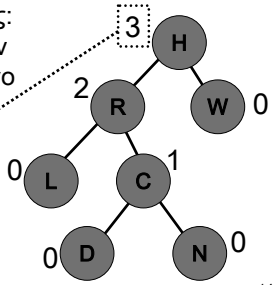



```
function depth (t, n)
//επιστρέφει το βάθος του κόμβου n στο δέντρο t
  if isRoot(t, n) return 0 //κόμβος είναι ρίζα του δέντρου
  else //το βάθος του πατέρα προσαυξημένο κατά 1
    return 1+depth(t, t.parent(n)) //ανεβαίνει ένα επίπεδο
```

140

Ύψος Κόμβου & Δέντρου

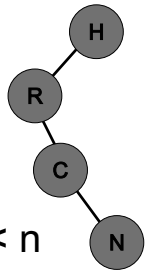
- Ύψος ενός κόμβου είναι η απόστασή του από το μακρινότερο παιδί του.
  - Αναδρομικός υπολογισμός: το μέγιστο από τα ύψη των παιδιών του προσαυξημένο κατά 1.
- Ύψος ενός δέντρου είναι το ύψος της ρίζας του.



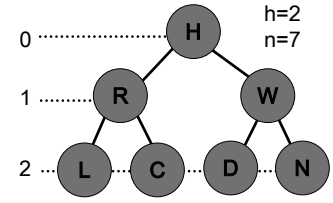
141

Κόμβοι n και Ύψος h

h=3
n=4



$h < n$



Στο επίπεδο i, το πολύ 2^i κόμβοι

$$\sum_{i=0}^{h} 2^i = \frac{2^{h+1}-1}{2-1}$$

$n \le 2^{h+1} - 1$

142

Διαπέραση Δέντρου

Tree traversal

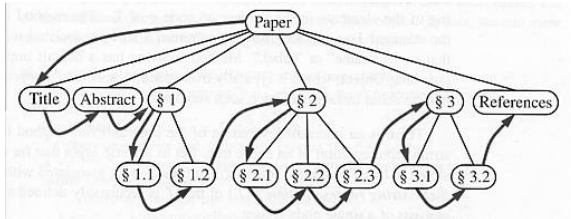
- Ένας συστηματικός τρόπος επίσκεψης όλων των κόμβων του δέντρου
- Συνήθως ορίζεται αναδρομικά.
- Θα δούμε 4 τρόπους διαπέρασης:
  - Εσωδιάταξη
  - Προδιάταξη
  - Μεταδιάταξη
  - Διάταξη κατά επίπεδα

143

Προδιάταξη Δέντρου

Preorder traversal

- Επισκεπτόμαστε τον πατέρα και μετά τα παιδιά του.
- Εφαρμόζουμε αυτόν τον κανόνα αναδρομικά στα υποδέντρα.

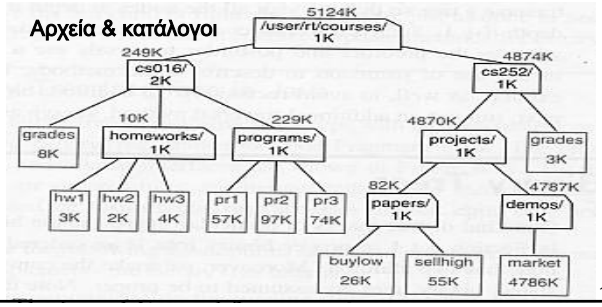


144

Μεταδιάταξη Δέντρου

Postorder traversal

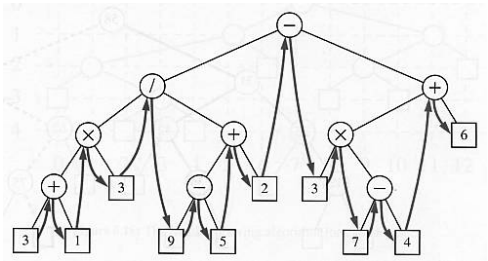
- Επισκεπτόμαστε όλα τα παιδιά πριν επισκεφθούμε τον πατέρα τους.



Εσωδιάταξη Δέντρου

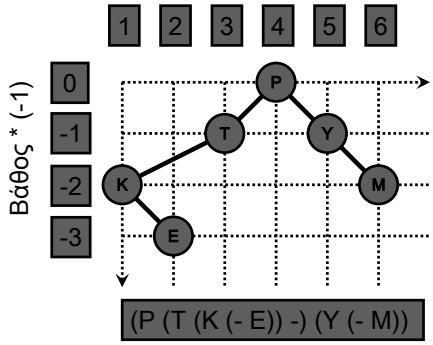
Inorder traversal

- Ορίζεται μόνον για δυαδικά δέντρα.
- Επισκεπτόμαστε πρώτα το αριστερό παιδί, μετά τον πατέρα και τέλος το δεξί παιδί.



Σχεδίαση Δέντρου

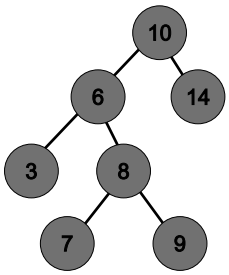
Σειρά επίσκεψης κατά εσωδιάταξη



Δυαδικά Δέντρα Αναζήτησης (ΔΔΑ)

Binary search trees

- Κάθε κόμβος είναι μεγαλύτερος των κόμβων του αριστερού του κλάδου και μικρότερος των κόμβων του δεξιού του κλάδου.
- Όταν διαπεραστούν με εσωδιάταξη (πατέρας, αριστερό, δεξί), εμφανίζονται τα στοιχεία ταξινομημένα.



Εισαγωγή σε Δέντρα Αναζήτησης

- Διατρέχει το δέντρο από τη ρίζα προς τα κάτω μέχρι να φτάσει σε κάποιο φύλλο και εκεί εισάγει νέο φύλλο.
- Άρα επίδοση O(*h*), όπου *h* το ύψος του δέντρου.
- Χειρότερη περίπτωση: *h* = O(*n*) μακρόστενο δέντρο

```
//εισάγει κόμβο με πληροφορία x στο δέντρο t
function insert(x, t) //εισάγει και διπλότυπα
        if t = null then
                t = new Node(x, null, null) //κατασκευή
        else //ψάξε παρακάτω (στα παιδιά)
                if x < t.data then insert(x, t.left)
                        else  insert(x, t.right)
```

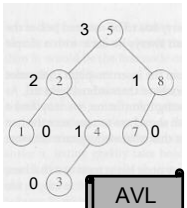

Διαγραφή σε Δέντρα Αναζήτησης

```
function remove (x, t) //διαγραφή κόμβου με x στο δέντρο t
  if x = null then return(t) //δεν βρέθηκε – μην κάνεις τίποτα
  elseif x<t.data then t.left = remove(x, t.left) //ψάξε
  elseif x>t.data then t.right = remove(x, t.right) //ψάξε
  else //βρήκαμε τον κόμβο που θα διαγράψουμε
    if t.left = null then t = t.right //μόνο δεξί παιδί
    elseif t.right=null then t = t.left //μόνο αριστερό παιδί
    else //δύο παιδιά
      //βρες το μικρότερο κόμβο του δεξιού υποδέντρου
      t.data = findMin(t.right).data
      t.right = remove(t.data, t.right) //σβήσε το μικρό
```

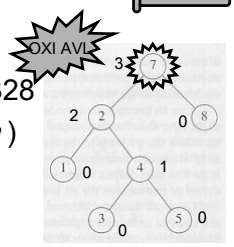

Ιωάννης Γαβιώτης
gaviotis@aegean.gr
http://www.syros.aegean.gr/users/gaviotis/add

Δέντρα AVL

Από Adelson-Velskii Landis 1962



- ΔΔΑ όπου για κάθε κόμβο το ύψος του αριστερού και το δεξιού του παιδιού διαφέρουν το πολύ κατά 1.
- Παρότι δεν είναι πλήρη, έχουν ύψος 1,44lg(n + 2)-0,328
- Όλες οι λειτουργίες σε O(lg$n$)

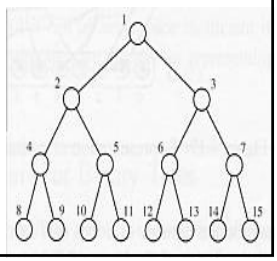


Ανισοσκελή Δέντρα Αναζήτησης

- Αν ένα δυαδικό δέντρο n κόμβων είναι ισοσκελισμένο, τότε έχει ύψος : $\lfloor \lg n \rfloor$
- Εισαγωγές / διαγραφές μπορεί να παράγουν ανισοσκελή δέντρα με ύψος n (το χειρότερο).
  - Πότε συμβαίνει αυτό;
- Οι μέθοδοι των ΔΔΑ είναι O(h), όπου h είναι το ύψος και $h \in \left[\lfloor \lg n \rfloor, n \right)$
- Διατήρηση ισορροπίας
  - Τροποποιούμε το δυαδικό δέντρο, έτσι ώστε να είναι 'γεμάτο' κόμβους

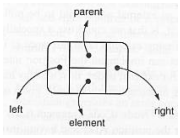
Διαπέραση κατά Επίπεδα

```
function diaperasiKataEpipeda (node)
  enqueue(node) //τοποθέτησε στην ουρά
  ενόσω η ουρά δεν είναι άδεια {
     w = dequeue()
     //κάνε ό,τι θέλεις με το w
     για κάθε παιδί c του w
        enqueue(c)
  }
```

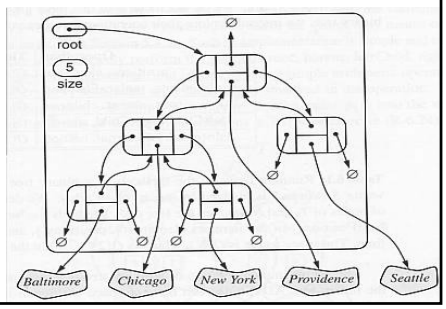


Διασυνδεδεμένα Δυαδικά Δέντρα

Κάθε κόμβος του δέντρου είναι ένα αντικείμενο με ιδιότητες: data, left, right, parent.

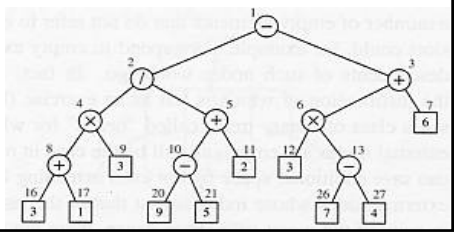


Ο parent ίσως δεν χρειάζεται.



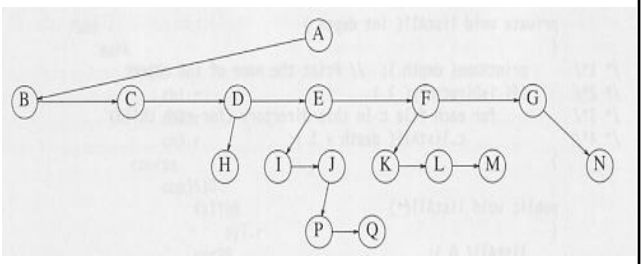
Αναπαράσταση Δυαδικού Δέντρου με Διάνυσμα

- Πίνακας 1…2^{h+1}-1 θέσεων, όπου h το ύψος του δέντρου. Η ρίζα του δέντρου στη θέση 1.
- Το αριστερό παιδί του κόμβου της θέσης χ στη θέση 2χ και το δεξί στη θέση 2χ+1.
- Γρήγορο
- Σπατάλη χώρου
  - Ειδικά για αραιά δέντρα



Αναπαράσταση Δέντρου με Αδέλφια

- Οι κόμβοι δεν έχουν σταθερό αριθμό παιδιών.
- Σύνδεσμοι προς παιδί και πρώτο αδέλφι



Ιωάννης Γαβιώτης
gaviotis@aegean.gr
http://www.syros.aegean.gr/users/gaviotis/add

Υπολογιζόμενες Ιδιότητες Δέντρων

- Ο συντελεστής ισορροπίας ενός κόμβου είναι η διαφορά του ύψους του δεξιού και του αριστερού του υποδέντρου.
- Κοντινότερος κοινός πρόγονος δύο κόμβων
- Η απόσταση δύο κόμβων είναι το πλήθος των ακμών αναμεταξύ τους.
- Η μέγιστη απόσταση ανάμεσα σε οποιουσδήποτε κόμβους καλείται διάμετρος του δέντρου.

157

11
Ουρές Προτεραιοτήτων

Δέντρα Σωρού

Ορισμός Ουράς Προτεραιότητας

Priority queue

- Συλλογή στοιχείων που ενσωματώνουν προτεραιότητες
  - Λίστα αναμονής επιβατών αεροπλάνου
  - Διεργασίες προς εξυπηρέτηση στην ουρά του επεξεργαστή / εκτυπωτή
- Υποστηρίζει (α) εισαγωγή και (β) εξαγωγή κατά την προτεραιότητα - όχι κατά FIFO.
- Μικρή τιμή ↔ Υψηλή προτεραιότητα

159

Μέθοδοι Ουράς Προτεραιότητας

- insert(d, p) : εισάγει ένα νέο στοιχείο με τιμή d και προτεραιότητα p
- removeMin() : αφαιρεί από την ουρά και επιστρέφει το στοιχείο με την υψηλότερη προτεραιότητα (μικρότερη τιμή)
- minPriority() : επιστρέφει την υψηλότερη προτεραιότητα
- size(), isEmpty() : βοηθητικές

160

Υλοποίηση με Μη Ταξινομημένη Ακολουθία

- insert
  - Εισαγωγή στο τέλος της ακολουθίας
  - O(1) είτε για διάνυσμα, είτε για διασυνδεδεμένη λίστα
- removeMin
  - Αναζήτηση της μικρότερης προτεραιότητας
  - O(n) για γραμμική αναζήτηση και κάλυψη κενού
    - Όχι μόνον στη χειρότερη, αλλά και στην καλύτερη περίπτωση

161

Υλοποίηση με Ταξινομημένη Ακολουθία

- insert
  - Για να τοποθετηθεί στη σωστή θέση πρέπει να αναζητηθεί: O(n) για λίστα και O($\lg n$) για διάνυσμα + να εισαχθεί O(1) για λίστα O(n) για διάνυσμα
  - Συνολικά O(n), όποιο κι αν διαλέξεις
- removeMin
  - O(1), αφού η μικρότερη προτεραιότητα είναι τοποθετημένη στην πρώτη θέση

162

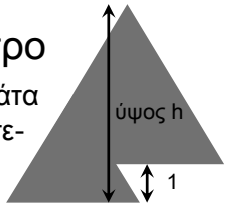
Υλοποίηση με Δέντρο Δυαδικής Αναζήτησης

- insert
  - Για να τοποθετηθεί στη σωστή θέση O(*h*)
  - Στην καλύτερη περίπτωση *h* = O(lg*n*) και στη χειρότερη περίπτωση *h* = O(*n*).
- removeMin
  - Όπως και η insert

Μπορούμε καλύτερα;

163

Πλήρες Δυαδικό Δέντρο



- Όλα τα επίπεδά του είναι γεμάτα κόμβους, εκτός ίσως από το τελευταίο, όπου οι κενές θέσεις έπονται των κατειλημμένων.
- Σωρός *n* κόμβων έχει ύψος $\lfloor \lg n \rfloor$
- Με ύψος *h* έχει μεταξύ 2^h και 2^{h+1}-1 κόμβους.
- Συνήθως αναπαριστάνεται με διάνυσμα
  - Αν ένας κόμβος είναι στη θέση *j*, το αριστερό του παιδί στη θέση 2*j*, το δεξί στη 2*j*+1, πατέρας $\lfloor j/2 \rfloor$
  - Σπατάλη < 2^{h-1} κενοί κόμβοι στο τελευταίο επίπεδο

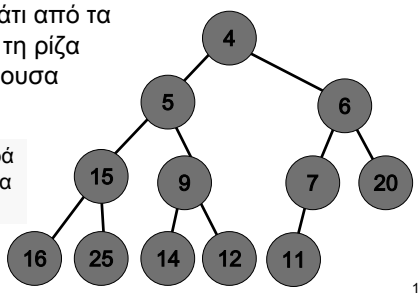
164

Δέντρο Σωρού

Heap tree

- Πλήρες δυαδικό δέντρο όπου ο πατέρας κάθε κόμβου είναι μικρότερος ή ίσος με το κόμβο.
  - Κάθε μονοπάτι από τα φύλλα προς τη ρίζα είναι σε φθίνουσα ταξινόμηση.

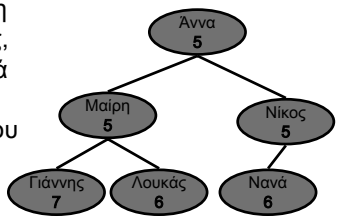
Σημειώστε διαφορά με Δυαδικά Δέντρα Αναζήτησης



165

Εφαρμογή Δέντρων Σωρού

- Για την αναπαράσταση ουράς προτεραιότητας, τοποθετούμε τα κλειδιά των στοιχείων της ουράς σαν στοιχεία που ταξινομούμε στους κόμβους.
- Εισαγωγή διαδοχικά: (Γιάννης, 7), (Άννα, 5), (Νίκος, 5), (Λουκάς, 6), (Μαίρη, 5), (Νανά, 6)



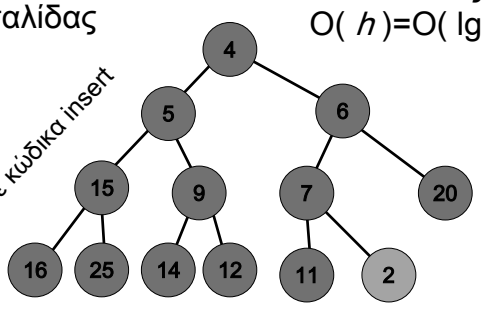
166

Εισαγωγή στο Δέντρο Σωρού

Κίνηση φυσαλίδας

Επιδόσεις: O(*h*)=O(lg*n*)

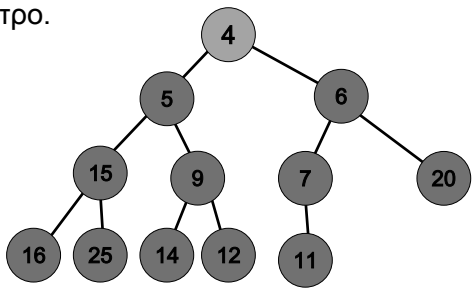
Φτιάξτε κώδικα insert



167

Διαγραφή από το Δέντρο Σωρού - 1

Η ρίζα (που θα διαγραφεί) αντιμετατίθεται με τον τελευταίο κόμβο για να διατηρηθεί πλήρες δέντρο.



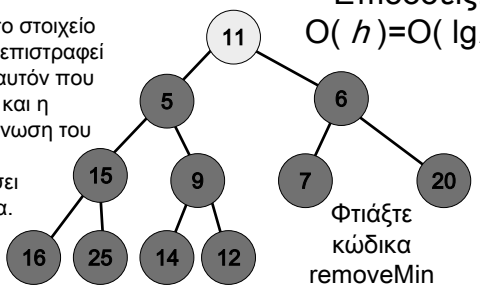
168

Διαγραφή από το Δέντρο Σωρού - 2

Κίνηση σαν βαρίδι:
ανταλλαγή με μικρότερο παιδί

Επιδόσεις:
$O(h)=O(\lg n)$

Το ελάχιστο στοιχείο μπορεί να επιστραφεί άμεσα σε αυτόν που το ζήτησε, και η αναδιοργάνωση του σωρού να προχωρήσει ασύγχρονα.



Φτιάξτε κώδικα removeMin

169

Σύγκριση Επιδόσεων

... στη χειρότερη περίπτωση

Αναπαραστάσεις ουράς προτεραιοτήτων

Μέθοδος \ Δομή	Μη ταξινομημένη λίστα	Ταξινομημένη λίστα	Δυαδικό Δέντρο Αναζήτησης	Σωρός
insert	O(1)	O(n)	O(lg$n$)..O(n)*	O(lg$n$)
removeMin	O(n)	O(1)	O(lg$n$)..O(n)*	O(lg$n$)

* Όταν το ΔΔΑ γίνεται ανισοσκελές.

170

Κατασκευή Δέντρου Σωρού

- Για να κατασκευάσουμε ένα σωρό από n στοιχεία, μπορούμε να εκτελέσουμε n διαδοχικά insert με κόστος $\sum_{i=1}^{n} \lg i = O(n \lg n)$
- Αλλά μπορούμε και καλύτερα:
  Φτιάχνουμε ένα πλήρες δέντρο με τα n στοιχεία (δεν είναι σωρός) και μετά εκτελούμε για όλους τους εσωτερικούς κόμβους (από τον τελευταίο προς τη ρίζα) 'κίνηση σαν βαρίδι' για να γίνει σωρός.
  Κόστος buildHeap = O(n)

171

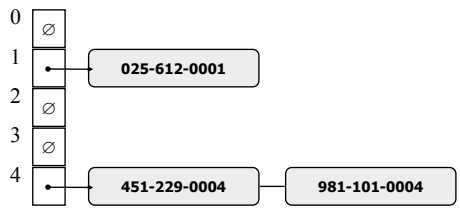
Το πρόβλημα της Επιλογής

- Σε μια ακολουθία n στοιχείων, βρες το k-οστό (Selection Problem)
- Ειδικές περιπτώσεις k
  - Για k = 1, βρίσκεις τον πρώτο σε n
  - Για k = 2, βρίσκεις τον πρώτο σε n και τον δεύτερο σε n-1. Υπάρχει βέλτιστος αλγόριθμος ≈ n+lg$n$
  - Για k = n/2, βρίσκεις τον μεσαίο
- Προσεγγίσεις:
  - Ταξινομείς n σε O(n.lg$n$) και παίρνεις το k-οστό σε O(1)
  - Κατασκευάζεις σωρό n στοιχείων σε O(n) και εκτελείς k φορές removeMin σε O(k.lg$n$)

172

12
Λεξικά

Πίνακες Κατακερματισμού



Ορισμός Λεξικού

Dictionary

- Συλλογή στοιχείων (κλειδί, τιμή) στην οποία ***αναζητούμε*** την τιμή με βάση το κλειδί (dictionary).
  - Δεν αναζητούμε με βάση την τιμή του στοιχείου, απλώς την ανασύρουμε.
  - Δεν ενδιαφέρει η ταξινομημένη διαπέραση, ή η εύρεση ελάχιστου / μέγιστου.
  - Δευτερεύουσας σημασίας: εισαγωγή / διαγραφή
- Παραδείγματα: τηλεφωνικός κατάλογος, εγκυκλοπαίδεια, χρονολόγιο

174

Ιδιότητες Λεξικού

- Το κλειδί και το στοιχείο μπορεί να είναι οποιουδήποτε τύπου.
- Μη ταξινομημένο λεξικό: αν δεν ορίζεται διάταξη των κλειδιών
  - Ή δεν έχει νόημα, πχ τραπεζικοί λογαριασμοί
- Επιτρέπονται διπλότυπες τιμές κλειδιών.
  - Όταν απαγορεύονται, έχουμε μια συνειρμική αποθήκη (associative store).
- Συμβολίζουμε με *n* το πλήθος των στοιχείων

175

Λεξικό ως ΑΤΔ

- findElement(k) : αναζήτηση με το κλειδί, επιστρέφει την τιμή
- insertItem(k, e) : εισάγει νέο στοιχείο
- removeElement(k) : διαγράφει στοιχείο
- findAllElements(k) : απαριθμητής
- removeAllElements(k) : απαριθμητής
- size(), isEmpty()
- keys(), elements() : απαριθμητές

176

Αρχείο Καταγραφής

Log File

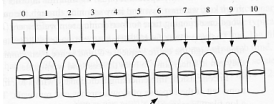
- Μια απλοϊκή υλοποίηση λεξικού ως μη ταξινομημένη γραμμική ακολουθία.
  - Αρχείο δοσοληψιών βάσεων δεδομένων: καταγράφει τις αλλαγές στα δεδομένα με τη σειρά που συμβαίνουν
- Γρήγορες εισαγωγές O(1), συμβαίνουν συχνά
- Αργές αναζητήσεις O(n), συμβαίνουν αραιά

177

Κάδοι

Buckets

- Για να αποθηκεύσουμε *n* στοιχεία στο λεξικό που τα κλειδιά τους είναι ακέραιοι 0..*N*-1, χρησιμοποιούμε πίνακα μεγέθους *N*
  - Στην *i*-οστή θέση αποθηκεύεται η τιμή του στοιχείου με κλειδί *i* – αν υπάρχει στο λεξικό.
- Χώρος O(N), χρόνος O(1), χρησιμοποίηση *n*/*N*
- Δεν επιτρέπονται διπλότυπα κλειδιά, αλλιώς προκαλούνται συγκρούσεις (collisions).



178

Κατακερματισμός

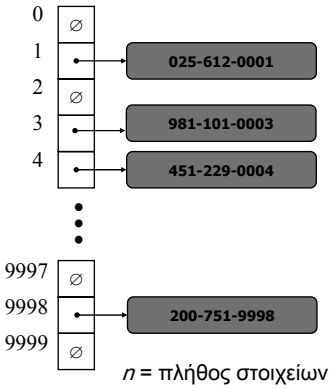
Hashing

- Για να αντιμετωπίσουμε τα μειονεκτήματα των κάδων, αποθηκεύουμε το στοιχείο με κλειδί *k*, στη θέση *h(k)* του πίνακα.
- Συνάρτηση κατακερματισμού *h* :
  - Όταν *k* δεν είναι ακέραιος, το *h(k)* είναι.
  - Έχει τιμές από 0 ως *N*-1.
- Συγκρούσεις, όταν $k_1 \neq k_2$ και $h(k_1) = h(k_2)$
  - Διαφορετικά κλειδιά πέφτουν στον ίδιο κάδο (θέση στον πίνακα κατακερματισμού).

179

Πίνακας Κατακερματισμού

Hash Table



Συμβολίζουμε με *N* το μέγεθος του πίνακα, εδώ N = 10000

Χρησιμοποιείται η συνάρτηση κατακερματισμού: «πάρε τα 4 τελευταία ψηφία», δηλ. *h(k)* = *k* mod N

180

Αντιμετώπιση Συγκρούσεων

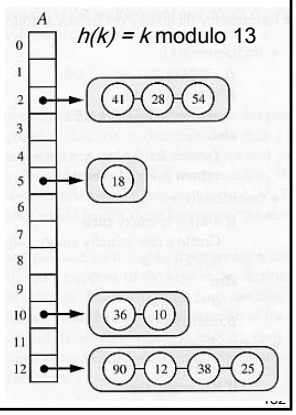
- Όσο δεν έχουμε συγκρούσεις, ο κατακερματισμός αποδίδει σε χρόνο O(1)
- Ορισμός: Παράγων Φόρτου $\lambda = n / N$
  - n = πλήθος στοιχείων που έχουν εισαχθεί
  - N = μέγεθος πίνακα
- Στην περίπτωση συγκρούσεων τα στοιχεία μπορεί να φυλάσσονται:
  - σε άλλη δομή δεδομένων έξω από τον πίνακα
  - σε εναλλακτικές θέσεις του πίνακα κατακερματισμού

181

Εξωτερική Αλυσίδωση

External Chaining

- Αντί ο κάδος να περιέχει τα στοιχεία του λεξικού, παραπέμπει σε (μη ταξινομημένες) ακολουθίες.
- Καλύτερη περίπτωση: n στοιχεία ισοκατανέμονται σε N κάδους, άρα κάθε αλυσίδα έχει μήκος n / N.
- Μειώνει χρόνο αναζήτησης
- Αυξάνει απαιτήσεις χώρου



182

Ανοικτή Διευθυνσιοδότηση με Δοκιμή

Open Addressing with Probing

Αδύνατο σημείο εξωτερικής αλυσίδωσης είναι οι ακολουθίες λόγω συγκρούσεων.

- Όταν η θέση είναι κατειλημμένη (σύγκρουση), παρέχεται εναλλακτική θέση μέσα στον πίνακα.
  - Όλα τα στοιχεία μέσα στον πίνακα
- Πρέπει οι κάδοι να είναι περισσότεροι των στοιχείων του λεξικού ($n < N$)
  - Πρακτικά η μέθοδος λειτουργεί καλά, όταν $\lambda < 0,5$

183

Γραμμική Δοκιμή

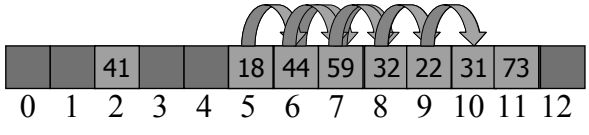
Linear Probing

- Όταν προκύπτει σύγκρουση, το στοιχείο τοποθετείται στον επόμενο μη κατειλημμένο κάδο.
- Διαγραφή: για όλα τα επόμενα στοιχεία της ομάδας, χρειάζεται επανεισαγωγή.
- Προκαλεί ομαδοποίηση (clustering): πάλι σχηματίζονται «αλυσίδες» μόνο που είναι εσωτερικές στον πίνακα
  - Αντιμετωπίζεται με τετραγωνική δοκιμή (quadratic probing): 0, 1, 4, 9, 16

184

Παράδειγμα Γραμμικής Δοκιμής

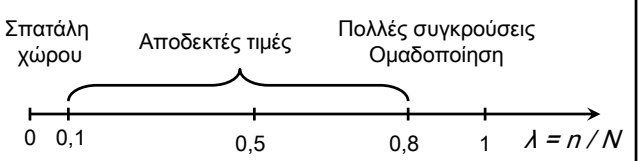
- Συνάρτηση κατακερματισμού
$$h(x) = x \bmod 13$$
- Εισάγονται τα κλειδιά: 18, 41, 22, 44, 59, 32, 31, 73, με αυτή τη σειρά



185

Ανακατακερματισμός

Rehashing



- Όταν ο παράγων φόρτου βγαίνει εκτός ορίων, αλλάζουμε το πλήθος N των κάδων και εφαρμόζουμε σε όλα τα στοιχεία την (τροποποιημένη) συνάρτηση κατακερματισμού.

186

Επιδόσεις Κατακερματισμού

- Στην χειρότερη περίπτωση, όλες οι λειτουργίες απαιτούν χρόνο O(n).
- Ο αναμενόμενος χρόνος είναι O(1), φτάνει ο συντελεστής πληρότητας να μην φτάνει κοντά στο 100%.
- Κρίσιμοι παράγοντες:
  - επιλογή συνάρτησης με βάση τις ιδιότητες και την κατανομή των κλειδιών, και
  - τήρηση του λ,
  - μέθοδος αντιμετώπισης των συγκρούσεων

187

13
Ταξινόμηση

Sorting

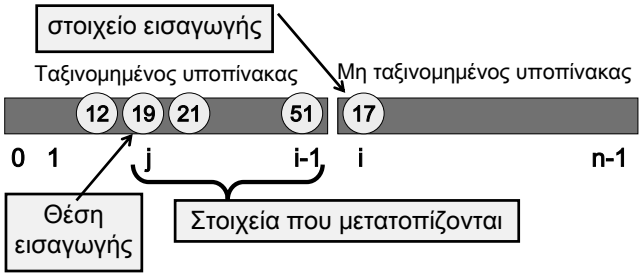
Ταξινόμηση Εισαγωγής

Insertion Sort

- Το αριστερό τμήμα του πίνακα είναι ταξινομημένο – το δεξί όχι.
- Το πρώτο στοιχείο του δεξιού τμήματος τοποθετείται στη σωστή θέση του στο αριστερό τμήμα, σπρώχνοντας τα υπόλοιπα μια θέση προς τα δεξιά.
- Βήματα:
  1. Εύρεση σωστής θέσης (σειριακή / δυαδική αναζ)
  2. Μετατόπιση των στοιχείων από τη σωστή και μετά

189

Επιδόσεις Ταξινόμησης Εισαγωγής



Στο βήμα i (για $1 \le i \le n-1$): κάνουμε (j+1) συγκρίσεις συν (i-j) μετατοπίσεις, δηλαδή συνολικά i+1 πράξεις $\sum_{i=1}^{n-1}(i+1) = \frac{n^2+n-1}{2} = O(n^2)$

Υποθέτουμε σειριακή αναζήτηση

190

Ταξινόμηση Επιλογής

Selection Sort

- Το αριστερό τμήμα του πίνακα είναι ταξινομημένο – το δεξί όχι.
- Βρίσκει το μικρότερο στοιχείο του δεξιού τμήματος και το ενθέτει ως τελευταίο του αριστερού τμήματος
  - Χρειάζεται να αντιμεταθέσει (swap) δύο στοιχεία: το πρώτο του μη ταξινομημένου τμήματος με το μικρότερο του μη ταξινομημένου τμήματος.
- Χρειάζεται (n -1)+(n -2)+...+ 2+1 συγκρίσεις συν 3n αναθέσεις (για τις αντιμεταθέσεις), δηλαδή (n -1).n / 2+3n = O(n^2)

191

Ταξινόμηση Φυσαλίδας

Bubble Sort

- Σαρώνουμε τον πίνακα από τα αριστερά προς τα δεξιά, αντιμεταθέτοντας όλα τα εκτός σειράς στοιχεία.
- Στο τέλος του περάσματος i, οι i τελευταίες θέσεις είναι ταξινομημένες.
  - Γι αυτό το επόμενο πέρασμα ξεκινά καλύπτει τα στοιχεία 0 έως (n – i).
- Αν σε κάποιο πέρασμα δεν γίνει καμιά αντιμετάθεση, τερματίζουμε πρόωρα.

192

Ταχεία Ταξινόμηση

Quick Sort

- Επιλέγουμε ένα στοιχείο του πίνακα (στοιχείο διαχωρισμού, pivot).
- Αναδιατάσσουμε τα στοιχεία του πίνακα, έτσι ώστε όλα τα μικρότερά του να είναι στον αριστερό υποπίνακα και όλα τα μεγαλύτερά του στον δεξί. Το στοιχείο διαχωρισμού έχει τοποθετηθεί στη σωστή του θέση.
- Η διαδικασία εφαρμόζεται αναδρομικά για τον αριστερό και το δεξί υποπίνακα, μέχρι να εκφυλιστούν σε μέγεθος 1.

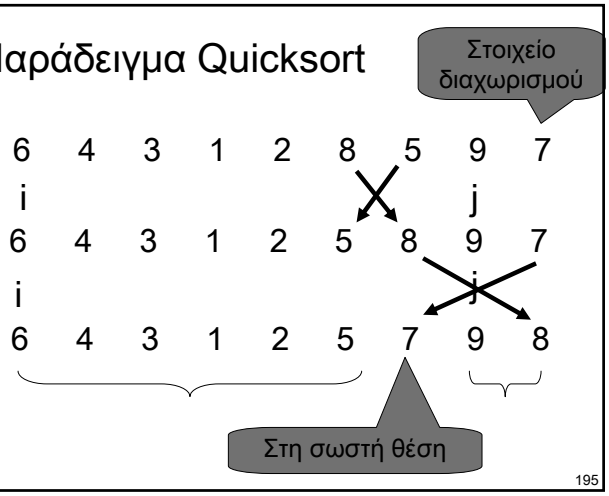
193

Υλοποίηση Quicksort

- Ως στοιχείο διαχωρισμού s επιλέγεται το τελευταίο.
- i ξεκινάει από αριστερό άκρο και ανεβαίνει μέχρι να βρει στοιχείο >= του s
- j ξεκινάει από δεξί άκρο και κατεβαίνει μέχρι να βρει στοιχείο <= s
  - Και τα δύο τερματίζουν αν φτάσουν στο άλλο άκρο ή να ξεπεράσουν το ένα το άλλο.
- Τα στοιχεία στα i, j αντιμετατίθενται.

194

Παράδειγμα Quicksort



195

Ψευδοκώδικας* Quicksort

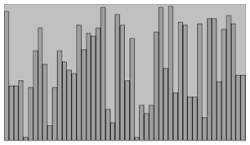
```
function quicksort(q)
  var list less, pivotList, greater
  if length(q) ≤ 1 then
    return q
  else
    select a pivot value pivot from q
    for each x in q except the pivot element
      if x < pivot then add x to less
      if x ≥ pivot then add x to greater
    add pivot to pivotList
    return concatenate(quicksort(less), pivotList, quicksort(greater))
```

Αναδρομή

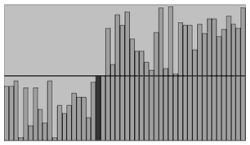
* Πηγή: http://en.wikipedia.org/wiki/Quicksort

196

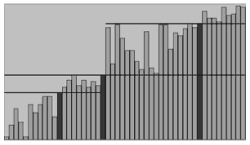
Στιγμιότυπα του Πίνακα κατά το QuickSort



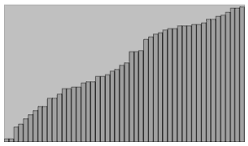
1. Αρχικά μη ταξινομημένος



2. Με κόκκινο το στοιχείο διαχωρισμού



3. Αναδρομή αριστερός και δεξιός υποπίνακας



4. Τελικό αποτέλεσμα

197

Διαίρει & Βασίλευε

Divide & Conquer

- Το αρχικό πρόβλημα διασπάται σε περισσότερα προβλήματα μικρότερου μεγέθους.
- Το κάθε υποπρόβλημα επιλύεται με αναδρομικό τρόπο
  - Είτε επαναδιασπάται, είτε φτάνει σε οριακό σημείο απλότητας, οπότε επιλύεται.
- Οι επιμέρους λύσεις συνδυάζονται για να λύσουν το μεγαλύτερο πρόβλημα.

198

Επιδόσεις Quicksort

- Η χειρότερη περίπτωση συμβαίνει όταν ο πίνακας είναι ήδη ταξινομημένος.
  - Ο αλγόριθμος αποδίδει καλύτερα, όσο πιο τυχαία είναι τοποθετημένα τα στοιχεία του πίνακα.
- Μέση περίπτωση 1,39.n.lg$n$
- Βελτιώσεις:
  - Στοίβα αντί αναδρομής
  - Ένα πέρασμα ελέγχει για διάταξη για να αποφύγουμε την χειρότερη περίπτωση.

Ιστορικό: Εφευρέθηκε από τον CAR Hoare το 1961 και είναι ο ταχύτερος αλγόριθμος ταξινόμησης γενικής χρήσης.

199

Ταξινόμηση Σωρού

Heap Sort

- Εισάγουμε τα στοιχεία που θέλουμε να ταξινομήσουμε σε έναν σωρό – O(n)
  - Σωρός: δυαδικό, ισοσκελισμένο δέντρο που κάθε κόμβος είναι μικρότερος από τα παιδιά του.
- Διαδοχικά απομακρύνουμε από το σωρό το μικρότερο στοιχείο του (ρίζα). Κάθε φορά ο σωρός αναδιαρθρώνεται – O(n.lg$n$)

200

Ταξινόμηση Συγχώνευσης

Merge Sort

1. Διαιρεί τον πίνακα σε δύο (ίσα) κομμάτια μεγέθους ~n/2.
2. Ταξινομεί τους δύο υποπίνακες αναδρομικά, μέχρι το μέγεθος να γίνει 2.
3. Συγχωνεύει τους δύο υποπίνακες σε ένα διατεταγμένο πίνακα.

- Τακτική «διαίρει & βασίλευε»
- Κόστος χειρότερης περίπτωσης: T(n)=T(n/2)+T(n/2)+n = n.lg$n$

201

Επιδόσεις Αλγορίθμων Ταξινόμησης

Αλγόριθμος Ταξινόμησης	Χείριστος χρόνος	Μέσος χρόνος	Βέλτιστος χρόνος
Εισαγωγής	O(n^2)	O(n^2)	n
Επιλογής	O(n^2)	O(n^2)	n
Φυσαλίδων	O(n^2)	O(n^2)	n
Ταχεία ταξινόμηση	O(n^2)	O(n.lg$n$)	2n
Συγχώνευσης	O(n^2)	O(n.lg$n$)	n+lg$n$
Σωρού	O(n.lg$n$)	O(n.lg$n$)	n

202

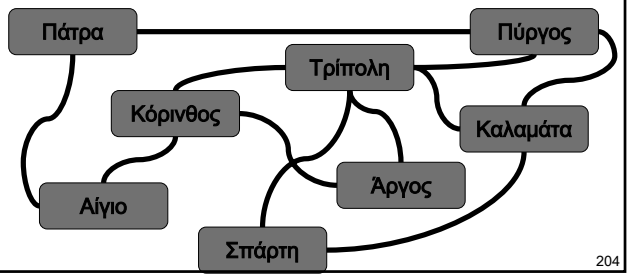
14
Γραφήματα

Graphs

Ορισμός

Graph, vertices, edges

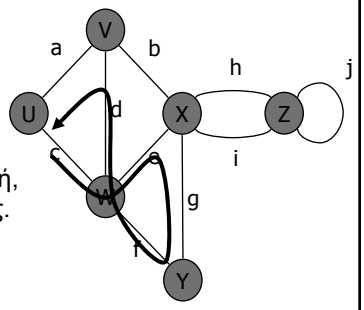
- Ένα γράφημα G αποτελείται από ένα σύνολο κορυφών V και ένα σύνολο ακμών E. Οι ακμές είναι ζεύγη κορυφών.



204

Παραλλαγές Γραφημάτων

- Ακμές με βάρη
- Πολυγραφήματα: μεταξύ δύο κορυφών, πολλές ακμές.
- Όταν δύο κορυφές συνδέονται με μια ακμή, ονομάζονται γειτονικές.
- Ένα σύνολο συνεχόμενων ακμών λέγεται μονοπάτι.



205

Κι Άλλοι Ορισμοί

- Όταν οι ακμές έχουν αρχή και τέλος, το γράφημα λέγεται κατευθυνόμενο (directed).
  - Βαθμός εισόδου (in-degree) μιας κορυφής: οι ακμές που καταλήγουν σε αυτήν.
  - Βαθμός εξόδου (out-degree) μιας κορυφής: οι ακμές που ξεκινούν από αυτήν.
  - Βαθμός (degree) μιας κορυφής: βαθμός εισόδου + βαθμός εξόδου

206

Μέθοδοι του ΑΤΔ Γράφημα

- Για μη κατευθυνόμενα γραφήματα
  - vertices() : απαριθμητής κορυφών γραφήματος
  - edges() : απαριθμητής ακμών γραφήματος
  - incidentEdges(v) : απαριθμητής ακμών που άπτονται της κορυφής v
  - endVertices(e) : ζευγάρι κορυφών που συνδέει η ακμή e
    - Επιστρέφει πίνακα δύο στοιχείων

207

Δευτερεύουσες Μέθοδοι

- Χρησιμοποιώντας τις προηγούμενες, υλοποιήστε: numVertices(), numEdges() : πλήθος, degree(v), opposite(v,e), adjacentVertices(v), areAdjacent(v,w)

```
function Vertex opposite(v, e) {
        int[] a = endVertices(e); //πίνακας 2 στοιχείων
        return (t[0] == v) ? t[1] : t[0]; }
function boolean areAdjacent(v, w) {
        Iterator x = incidentEdges(v);
        while (x.hasNext)
                if (opposite(v,x) == w) return true;
        return false; }
```

208

Μέθοδοι (συνέχεια ...)

- insertVertex(o)
- insertEdge(v, w, o)
- removeVertex(v)
- removeEdge(e)

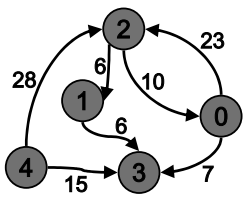
- Για κατευθυνόμε-να γραφήματα:
  - inDegree(v)
  - outDegree(v)
  - inIncidentEdges(v)
  - outIncidentEdges(v)
  - inAdjacentVertices(v)
  - outAdjacent-Vertices(v)

209

Πίνακας Γειτνιάσεως

Adjacency Matrix

- Πίνακας δύο διαστάσεων που οι γραμμές είναι οι κορυφές εκκίνησης και οι στήλες οι κορυφές τερματισμού. Κάθε θέση του πίνακα περιέχει τα βάρη των ακμών.



	0	1	2	3	4
0			23	7	
1				6	
2	10	6			
3					
4			28	15	

210

Αξιολόγηση Πίνακα Γειτνιάσεως

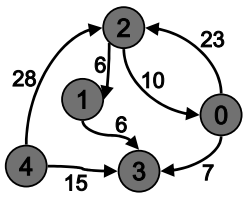
- Χώρος $O(n^2)$, όπου n = πλήθος κορυφών
- Αν το γράφημα είναι μη κατευθυνόμενο, ο πίνακας είναι συμμετρικός ως προς την κύρια διαγώνιο.
- Ιδανική αναπαράσταση για «πυκνά» γραφήματα
- Προσθαφαίρεση κορυφής απαιτεί αναδιαμόρφωση του πίνακα.
- Οι ακμές κορυφής απαιτούν $O(n)$ χρόνο.
- Γειτονικές κορυφές σε $O(1)$ χρόνο

211

Λίστα Γειτνιάσεως

Adjacency list

- Μια πρωτεύουσα συλλογή αναπαριστάνει κορυφές. Κάθε κορυφή οδηγεί σε μια δευτερεύουσα συλλογή που περιέχει τις γειτονικές κορυφές.



0	→	2 | 23	→	3 | 7
1	→	3 | 6		
2	→	0 | 10	→	1 | 6
3				
4	→	2 | 28	→	3 | 15

212

Αξιολόγηση Λίστας Γειτνιάσεως

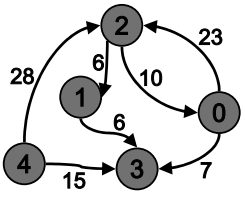
- Χώρος $O(n+m)$, όπου n = πλήθος κορυφών και m = πλήθος ακμών
  - Λιγότερος από τον πίνακα γειτνιάσεως
- Ιδανική αναπαράσταση για «αραιά» γραφήματα
- Οι ακμές κορυφής απαιτούν χρόνο της τάξης του βαθμού της κορυφής.
  - Ενώ απαιτούσαν σταθερό χρόνο με τον πίνακα.

213

Πίνακας Ακμών

Edge Matrix

- Χρησιμοποιεί έναν πίνακα κορυφές και έναν για τις ακμές, όπου εκτός από το βάρος τους αποθηκεύονται και δύο δείκτες προς τις κορυφές.

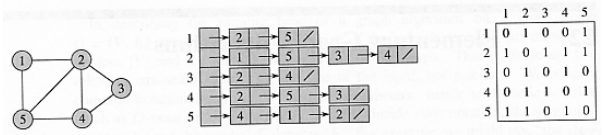


0
1
2
3
4

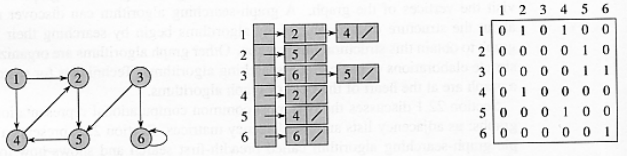
0	2	23
0	3	7
1	3	6
2	1	6
2	0	10
4	2	28
4	3	15

214

Παραδείγματα Αναπαραστάσεων



Μη Κατευθυνόμενο γράφημα



Κατευθυνόμενο γράφημα

215

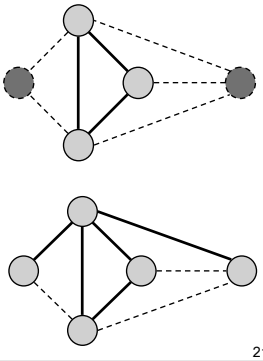
Επιδόσεις Αναπαραστάσεων

• n κορυφές • m ακμές • όχι παράλληλες ακμές • όχι ακμές (v, v)	Πίνακας Γειτνιάσεως	Λίστα Γειτνιάσεως	Πίνακας Ακμών
Χώρος	n^2	$n + m$	$n + m$
incidentEdges(v)	n	βαθμός(v)	m
areAdjacent (v, w)	1	ελάχιστο(βαθμός(v), βαθμός(w))	m
insertVertex(o)	n^2	1	1
insertEdge(v, w, o)	1	1	1
removeVertex(v)	n^2	βαθμός(v)	m
removeEdge(e)	1	1	1

216

Υπογραφήματα

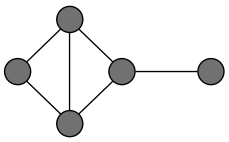
- Υπογράφημα (subgraph) S ενός γραφήματος G είναι ένα γράφημα που
  - οι ακμές του S είναι υποσύνολο ων ακμών του G
  - οι κορυφές του S είναι υποσύνολο των κορυφών του G
- Επικαλύπτον (spanning) υπογράφημα του G είναι ένα υπογράφημα που περιέχει όλες τις κορυφές του G.



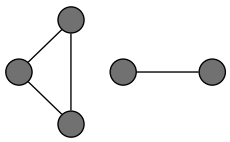
217

Συνεκτικότητα Γραφημάτων

- Ένα γράφημα με n κορυφές, έχει $O(n^2)$ ακμές.
- Ένα γράφημα είναι συνεκτικό αν υπάρχει μονοπάτι ανάμεσα σε όλες του τις κορυφές.
  - Έχει τουλάχιστον *n*-1 ακμές.



Συνεκτικό γράφημα

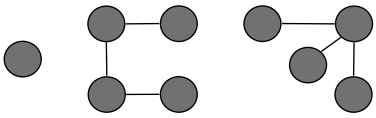


Μη συνεκτικό γράφημα με δύο συνεκτικές συνιστώσες

218

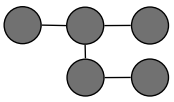
Δέντρα & Δάση

- Δάσος είναι ένα γράφημα χωρίς κύκλους.



- Δέντρο είναι ένα συνεκτικό γράφημα δίχως κύκλους.
  - Κάθε δέντρο n κορυφών έχει *ακριβώς* n-1 ακμές.

Μη 'γεωμετρικός' ορισμός



219

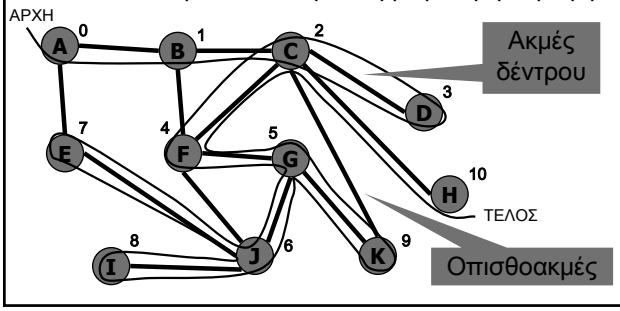
Αναζήτηση σε Βάθος (ΑσΒ)

- Αρχικά όλες οι κορυφές μαρκάρονται ως ανεξερεύνητες.
- Η διαπέραση ξεκινά από μια τυχαία κορυφή v η οποία σημειώνεται ως εξερευνημένη.
- Επιλέγεται τυχαία μια ακμή της v.
  - Αν αυτή οδηγεί σε μια ανεξερεύνητη κορυφή w, ο αλγόριθμος εφαρμόζεται αναδρομικά στη w.
  - [Όταν δεν βρεθεί καμιά ανεξερεύνητη κορυφή, η αναζήτηση συνεχίζεται από την προηγούμενη επισκεφθείσα κορυφή.]

220

Παράδειγμα ΑσΒ

- Αρχική κορυφή A
  - Υποθέτουμε ότι επιλέγεται η μικρότερη κορυφή



221

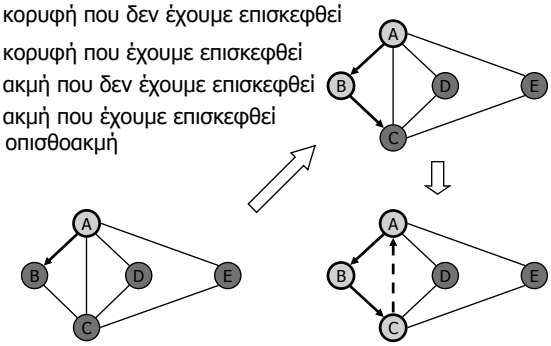
Ψευδοκώδικας ΑσΒ

```
διαδικασία ΑσΒ(κορυφή v) {
  εάν δεν έχουμε επισκεφθεί την κορυφή v {
      σημείωσε ότι τώρα επισκεπτόμαστε τη v
      για όλες τις ακμές (v, w)
          // ξεκινούν από τη v και οδηγούν σε μια w
          ΑσΒ(w) //αναδρομή
  }
}
```

222

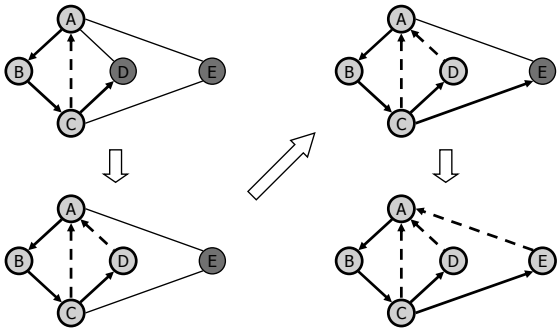
Παράδειγμα ΑσΒ

A κορυφή που δεν έχουμε επισκεφθεί
A κορυφή που έχουμε επισκεφθεί
ακμή που δεν έχουμε επισκεφθεί
ακμή που έχουμε επισκεφθεί
οπισθοακμή



223

Παράδειγμα (συνέχεια)



224

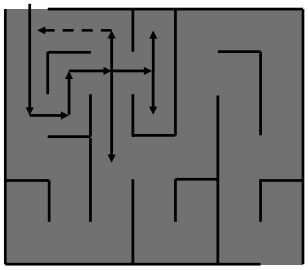
Παρατηρήσεις για ΑσΒ

- Επισκέπτεται όλες τις κορυφές που ανήκουν στην ίδια συνεκτική συνιστώσα που ανήκει η κορυφή από την οποία ξεκίνησε ο ΑσΒ.
- Οι ακμές που ακολουθεί συνιστούν ένα επικαλύπτον δέντρο του γραφήματος.
- Φτιάχνει «μακρόστενα» δέντρα: μεγάλο ύψος, μικρό πλάτος
- Είναι γενίκευση της προδιάταξης που είχαμε δει στα δέντρα.

225

Λαβύρινθος

- Ο αλγόριθμος της ΑσΒ είναι ανάλογος με τη στρατηγική που ακολουθούμε στο παιχνίδι όπου εξερευνούμε ένα λαβύρινθο.



226

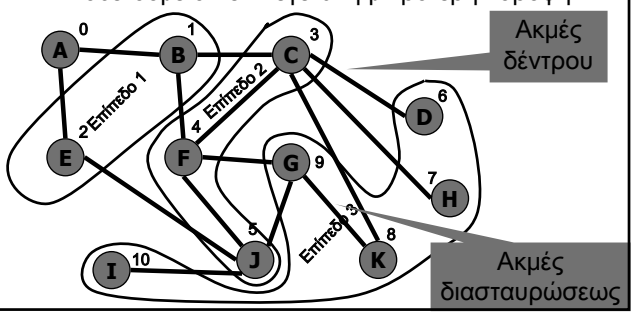
Αναζήτηση σε Πλάτος (ΑσΠ)

- Διατρέχει το γράφημα κατά «επίπεδα».
- Για κάθε κορυφή επισκέπτεται όλες τις γειτονικές της (που δεν έχει ήδη επισκεφθεί).
- Έπειτα ξεκινά τις επισκέψεις στους γείτονες των κορυφών του προηγούμενου επιπέδου.
- Ο ΑσΠ ξεκινά από μια (τυχαία) κορυφή και επισκέπτεται
  - όλες τις κορυφές που έχουν απόσταση 1,
  - μετά όλες εκείνες που έχουν απόσταση 2, κοκ.

227

Παράδειγμα ΑσΠ

- Αρχική κορυφή Α
  - Υποθέτουμε ότι επιλέγεται η μικρότερη κορυφή



Ψευδοκώδικας ΑσΠ

```
διαδικασία ΑσΠ(γράφημα G) {
   θέσε όλες τις κορυφές του G ως ανεξερεύνητες
   φτιάξε μια ουρά q
   τοποθέτησε στην ουρά το αρχικό στοιχείο της διαπέρασης
   ενόσω η ουρά δεν είναι άδεια
        v = q.dequeue()
        εάν δεν την έχει επισκεφθεί ακόμη {
              σημείωσε ότι την επισκέφθηκες
              για όλες τις ακμές (v, w) που ξεκινούν από την v
                    q.enqueue(w)
        } //τέλος εάν
   } //τέλος ενόσω
} //τέλος ΑσΠ
```

229

Παρατηρήσεις για ΑσΠ

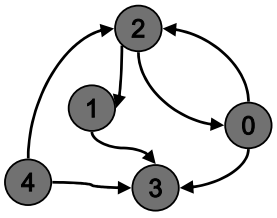
- Η υλοποίηση χρησιμοποιεί μια ουρά.
- Φτιάχνει «πλατιά» δέντρα: μικρό ύψος, πολλά κλαδιά
- Χρήσιμη αν θέλουμε να βρούμε τις κορυφές που απέχουν συγκεκριμένη (πχ μικρότερη) απόσταση από μια κορυφή.
- Είναι γενίκευση της διαπέρασης κατά επίπεδα που είχαμε δει στα δέντρα

230

Προσπελασιμότητα

Reachability

- Πρόβλημα: Βρείτε αν υπάρχει μονοπάτι από μια κορυφή v προς μια κορυφή w.
- Διαπερνώντας το γράφημα βρίσκουμε όλες τις κορυφές που είναι προσπελάσιμες από τη v.
  - Αν χρησιμοποιήσουμε ΑσΠ το μονοπάτι είναι το ελάχιστο (σε πλήθος ακμών).
- Πχ v=4, w=3



231

Ισχυρά Συνεκτικότητα

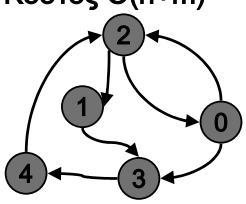
Strong Connectivity

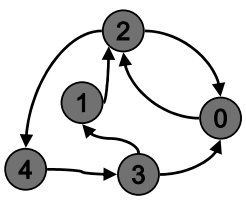
- Πρόβλημα: Προσδιορίστε αν ένα κατευθυνόμενο γράφημα είναι ισχυρά συνεκτικό, δηλ. από κάθε κορυφή v μπορείς να προσπελάσεις κάθε κορυφή w.
  - Αν όχι, υπολογίστε το πλήθος των ισχυρά συνεκτικών συνιστωσών του.
- **Λύση 1**: εφαρμόζω ΑσΒ για όλες τις κορυφές.
  - Αν από κάθε κορυφή είναι προσπελάσιμες όλες οι υπόλοιπες κορυφές, τότε το γράφημα είναι ισχυρά συνεκτικό. **Κόστος O(n.(n+m))**

232

Ισχυρά Συνεκτικότητα (2)

- **Λύση 2**: εφαρμόζω ΑσΒ για μια κορυφή v και αν οι υπόλοιπες n-1 προσπελαύνονται, αναστρέφω τις ακμές και εφαρμόζω πάλι ΑσΒ. **Κόστος O(n+m)**





Ανάστροφο

233

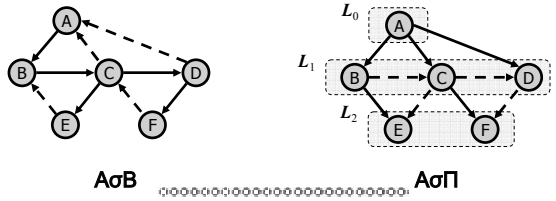
Κύκλοι σε Γράφημα

```
διαδικασία ΑσΒ-Κύκλοι(κορυφή v) { // εάν το γράφημα
   //είναι ισχυρά συνεκτικό, η εκκίνηση δεν έχει σημασία
   εάν δεν έχουμε επισκεφθεί την κορυφή v {
        σημείωσε ότι τώρα επισκεπτόμαστε τη v
        για όλες τις ακμές (v, w)
              // ξεκινούν από τη v και οδηγούν σε μια w
              ΑσΒ-Κύκλοι(w) //αναδρομή
   }
   αλλιώς τύπωσε «Υπάρχει κύκλος» & επέστρεψε
}
```

234

ΑσΒ έναντι ΑσΠ

Τομείς εφαρμογής	ΑσΒ	ΑσΠ
Συνεκτικές συνιστώσες, μονοπάτια, κύκλοι	√	√
Κοντινότερο μονοπάτι		√
Συνεκτικότητα δι-γράφων	√	



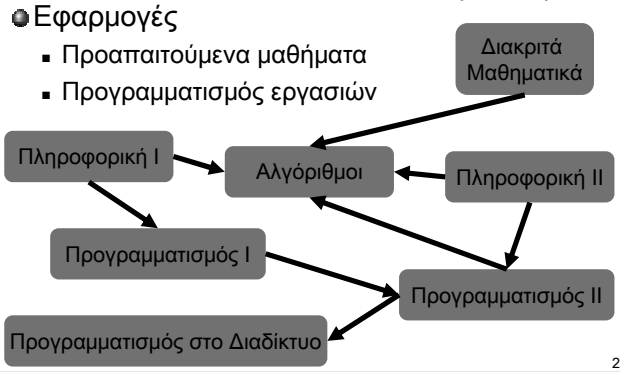
ΑσΒ ΑσΠ

235

Κατευθυνόμενα Άκυκλα Γραφήματα

Directed Acyclic Graph - DAG

- Εφαρμογές
  - Προαπαιτούμενα μαθήματα
  - Προγραμματισμός εργασιών



236

Τοπολογική Διάταξη

Topological Sort

- Πρόβλημα: οι κορυφές να μπουν σε μια σειρά ώστε να μην παραβιάζεται καμιά προαπαίτηση.
- Ορισμοί:
  - Πηγή: μια κορυφή με βαθμό εισόδου = 0
  - Καταβόθρα: μια κορυφή με βαθμό εξόδου = 0
- Λύση με επανάληψη: επισκέψου πρώτα τις πηγές και αφαίρεσε τις ακμές που φεύγουν από αυτές.

237

Αλγόριθμος Τοπολογικής Διάταξης

διαδικασία τοπολΔιάτ (G)
  υπολόγισε τον βαθμό εισόδου των κορυφών
  εάν υπάρχουν κορυφές που δεν έχεις επισκεφθεί
    αν υπάρχει μία πηγή τότε
      αφαίρεσε τις ακμές του γραφήματος που
        ... εξέρχονται της πηγής ενημερώνοντας τους
        ... αντίστοιχους βαθμούς εισόδου
      επέστρεψε πηγή & τοπολΔιάτ(G) //αναδρομή
    αλλιώς τύπωσε ΑΠΟΤΥΧΙΑ

238

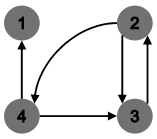
Μεταβατική Κλειστότητα

Transitive Closure

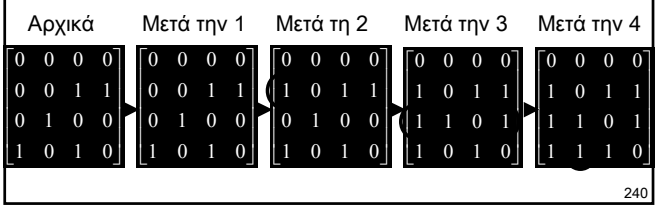
- Πρόβλημα: Για ένα γράφημα G=(V, E), κατασκευάστε ένα $G^*=(V, E^*)$ που έχει ακμές (u, w), αν στο G, η w είναι προσπελάσιμη από τη u (δηλ. υπάρχει μονοπάτι u → … → w).
  - V = κορυφές (Vertices) E = ακμές (Edges)
  - Μήκος μονοπατιού 1 .. ∞
- Εφαρμογή: δοθέντος του οργανογράμματος μιας εταιρίας, βρες όλους τους ανθρώπους που προϊστανται ενός υπαλλήλου.

239

Αλγόριθμος Μεταβατικής Κλειστότητας



Για όλες τις κορυφές w
  πρόσθεσε στο γράφημα του αποτελέσματος τις ακμές από μια αναζήτηση με κορυφή εκκίνησης τη w



240

Ελάχιστο Επικαλύπτον Δέντρο

Minimum Spanning Tree

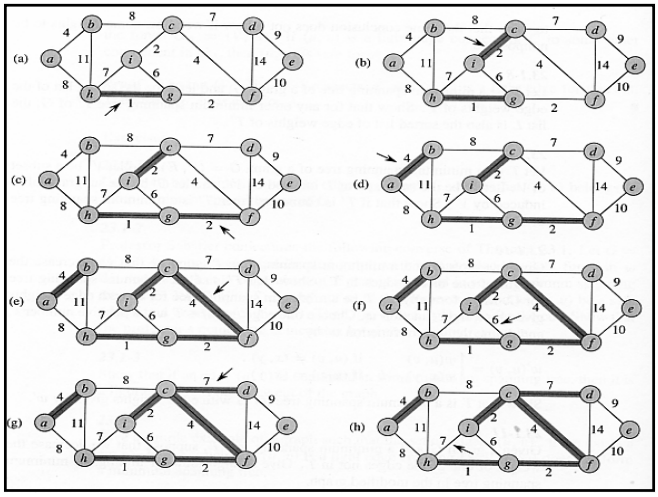
- Για συνεκτικό μη κατευθυνόμενο γράφημα με βάρη
- Πρόβλημα: να βρεθεί συνεκτικό επικαλύπτον γράφημα με το ελάχιστο συνολικό βάρος.
- Εφαρμογή: Δίκτυο ηλεκτροδότησης
  - Θέλουμε να κατασκευάσουμε ένα δίκτυο από γραμμές που να συνδέουν τις πόλεις σε ένα χάρτη. Για κάθε διαδρομή που συνδέει δύο πόλεις γνωρίζουμε το κόστος (πχ απόσταση).
- Ελάχιστο μήκος καλωδίων ηλεκτρονικού κυκλώματος

241

Αλγόριθμος Kruskal

διαδικασία Kruskal(G)
  A = {} //το αποτέλεσμα
  για κάθε κορυφή v: φτιάξε-σύνολο(v)
  ταξινόμησε τις ακμές κατ΄ αύξουσα σειρά
  για κάθε ακμή (u, w) από τη «φθηνότερη»
    εάν σύνολο(u) ≠ σύνολο(w) τότε
      A = A U {(u, w)}
      ένωσε τα σύνολα u και w

242



Συντομότερο Μονοπάτι Μοναδικής Πηγής

- Πρόβλημα: Για συγκεκριμένη κορυφή εκκίνησης v να βρεθούν τα μονοπάτια με το μικρότερο βάρος* για όλες τις υπόλοιπες κορυφές του γραφήματος.
- Εφαρμογή: κατευθυνόμενο γράφημα με (θετικά) βάρη

* όχι με το μικρότερο μήκος – βλέπε ΑσΠ

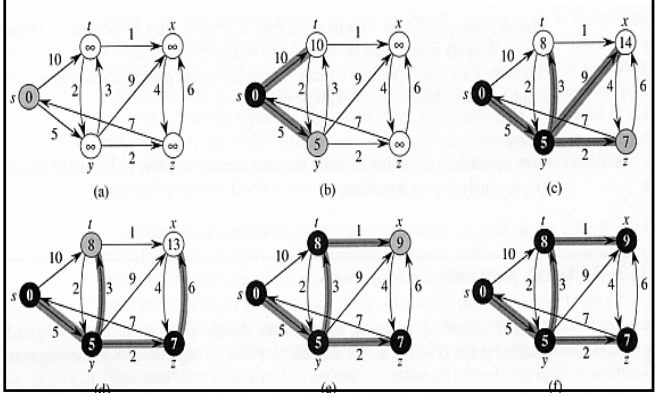
244

Αλγόριθμος Dijkstra

διαδικασία Dijkstra (κορυφές V, ακμές E, κορυφή εκκίνησης s)
  για όλες τις κορυφές w του V θέσε distance(w) = ∞
  distance(s) = 0 // απέχει 0 από τον εαυτό της
  S = {s} // κορυφές που έχουμε επισκεφθεί
  Q = V - {s} // κορυφές που δεν έχουμε επισκεφθεί
  ενόσω Q ≠ {} //μέχρι να εξαντληθούν οι κορυφές
    για κάθε γείτονα w της u // υπάρχει (u,w) στο E
      // ενημέρωσε την απόσταση της w από τη u
      distance(w) = min {distance(w), distance(u) + E(u,w) }
    θέσε u = κορυφή στο Q με την ελάχιστη distance
    S = S U {u} Q = Q - {u}

245

Εκτέλεση Dijkstra (εκκίνηση 0)



15
Ταίριασμα Προτύπων

Pattern Matching

Ακριβές Ταίριασμα Προτύπου

- Πρόβλημα: Δίδεται μια συμβολοσειρά T μήκους *n* και ένα πρότυπο P μήκους *m*. Ζητείται ο εντοπισμός των εμφανίσεων του P μέσα στο T.
- Εφαρμογές:
  - Αναζήτηση κειμένου
  - Αναζήτηση υποδομής στο DNA

248

Απλοϊκή Λύση

- Αλγόριθμος τύπου brute force
- Ξεκινούμε από την πρώτη θέση του T και εξετάζουμε αν οι χαρακτήρες του ταυτίζονται με τους χαρακτήρες του προτύπου. Μόλις βρούμε διαφορά, εγκαταλείπουμε.
- Συνεχίζουμε την ίδια διαδικασία με την επόμενη θέση του T, κοκ μέχρι τη θέση *n-m*+1. (Γιατί;)

T = **abrababracadabra**
P = **abracadabra**
n = 16, m = 11

```
abrababracadabra
abracadabra
 abracadabra
  abracadabra
   abracadabra
    abracadabra
     abracadabra
      abracadabra
```

249

Ψευδοκώδικας Ταιριάσματος

```
Αλγόριθμος ΑκριβέςΤαίριασμα(T, P)
  n = length(T); m = length(P);
  for i = 1 to n-m+1 //το ξεκίνημα του παραθύρου
      for j = 1 to m //οι χαρακτήρες του προτύπου
            if T(i+j-1) != P(j) then next i //αποτυχία
      next j
      return i //βρέθηκε ταίριασμα
  next i
```

Κόστος χειρότερης περίπτωσης: O(n.m)

250

Αλγόριθμος Knuth-Morris-Pratt (1977)

- Γίνεται προεπεξεργασία του προτύπου P:
  - Για κάθε θέση του προτύπου βρίσκουμε πόσο μεγάλο πρόθεμα του προτύπου υπάρχει που να συμπίπτει με το επίθεμα που τελειώνει σε αυτή τη θέση.

	a	b	r	a	c	a	d	a	b	r	a
j	1	2	3	4	5	6	7	8	9	10	11
f(j)	0	0	0	1	0	1	0	1	2	3	4

Πρόθεμα μήκους 2 του προτύπου συμπίπτει με επίθεμα που τελειώνει εδώ

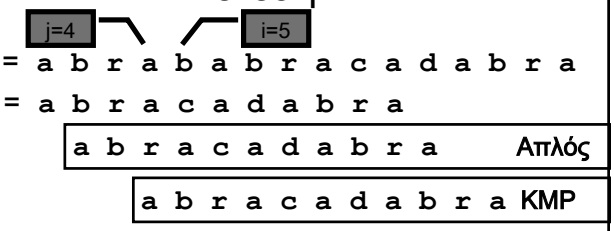
251

Ταίριασμα κατά KMP

- Κατά την αναζήτηση του ταιριάσματος όσο οι πρώτοι χαρακτήρες διαφέρουν προχωρούμε γραμμικά στο κείμενο T.
- Αν οι j πρώτοι χαρακτήρες του προτύπου και κειμένου ταυτίζονται και j=m, τότε βρήκαμε ταίριασμα.
  - Αν όμως j<m και στον επόμενο χαρακτήρα διαφέρουν, τότε το κείμενο μπορεί να συνεχίσει να συγκρίνεται με ασφάλεια με τον f(j) χαρακτήρα.
  - Συνεπώς μετακινήσαμε το παράθυρο κατά j-f(j).

252

Εκτέλεση KMP



Η σύγκριση ξεκίνησε από την αρχή με επιτυχία ως τον πέμπτο χαρακτήρα του T, όπου απέτυχε.

Ο απλός αλγόριθμος θα συνέχιζε συγκρίνοντας το P με το δεύτερο χαρακτήρα του T.

Ο KMP συγκρίνει τον T_i με τον $P_{f(j)+1} = P_2$.

253

Επιδόσεις

- Χάρη στα «άλματα», ο αλγόριθμος KMP είναι ταχύτερος. Απαιτεί χρόνο O(*n*+*m*)
  - O(m) για την προεπεξεργασία του προτύπου
  - O(n) για τη φάση του ταιριάσματος στη χειρότερη περίπτωση

254