ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

http://eclass.aueb.gr/courses/INF511/

Αφαίρεση Δεδομένων (ΚΕΦΑΛΑΙΟ 8)

Αλκμήνη Σγουρίτσα

Κοδριγκτώνος 12, 2ος όροφος

E-mail: alkmini@aueb.gr

ΚΕΦΑΛΑΙΟ 8: Αφαίρεση δεδομένων

- Βασικές Δομές Δεδομένων
  - Λίστες, στοίβες, ουρές, δέντρα
- Αποθήκευση Δομών Δεδομένων στη μνήμη
  - Πίνακες, ετερογενείς πίνακες, λίστες, στοίβες, ουρές και δέντρα
- Επίδραση της Δομής Δεδομένων στο σχεδιασμό αποδοτικών αλγορίθμων
  - Παράδειγμα: Δυαδική αναζήτηση

Ανάγκη για Δομές δεδομένων (data structures)

Αδόμητα δεδομένα

οδός Ζέας

14, οδός Πατησίων

27

Φρύνη Βάρναλη

56 47, Ιωάννης Βρανάς

Ιωσήφ Κούλης 23

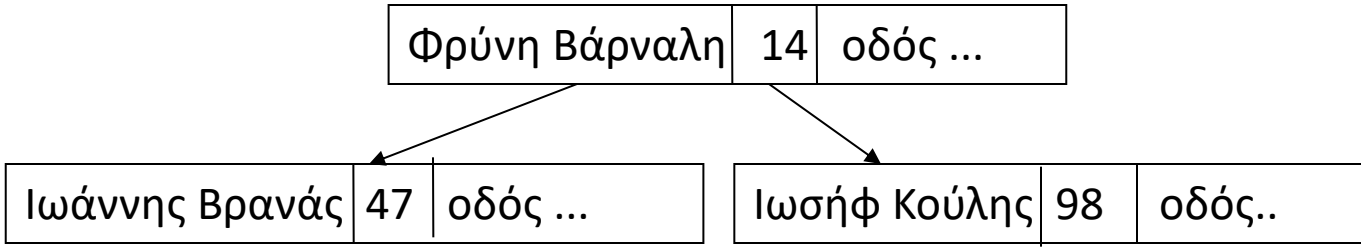
οδός Σκουφά

98,

Δομημένα δεδομένα

Ιωάννης Βρανάς	47	οδός Ζέας 27
Φρύνη Βάρναλη	14	οδός Σκουφά 56
Ιωσήφ Κούλης	98	Πατησίων 23

Λογική σχέση



Φύση των Δομών Δεδομένων

- Δομές Δεδομένων (data structures) (πίνακες, λίστες, στοίβες, ουρές, δέντρα): αφαιρετικά (abstract) εργαλεία για καλύτερη οργάνωση των δεδομένων
  - στο μυαλό του προγραμματιστή
  - για ευκολότερο σχεδιασμό αλγορίθμων
- Στατική (static): η δομή δεδομένων της οποίας το μέγεθος δεν αλλάζει
  - Ποιο εύκολη στη διαχείριση
- Δυναμική (dynamic): η δομή της οποίας το μέγεθος μπορεί να αλλάξει
  - Εισαγωγή και διαγραφή δεδομένων
  - Δυναμική ανάθεση μέρους της μνήμης (dynamic memory allocation) → αυξο-μείωση μεγέθους της δομής

Βασικές Δομές Δεδομένων

- **Ομογενής πίνακας** (homogeneous array)
  - Περιλαμβάνει ιδίου τύπου δεδομένα
  - Π.χ. Δι-διάστατος πίνακας m x n: m γραμμές (rows) n στήλες (columns)
  - Η θέση κάθε στοιχείου προσδιορίζεται με ένα ζευγάρι δεικτών (i,j), i=1,…m, j=1,…n
- **Ετερογενής πίνακας** (heterogeneous array)
  - Περιλαμβάνει δεδομένα διαφορετικών τύπων
  - Π.χ. Για υπάλληλο: όνομα, ηλικία, κλπ.
- **Συνδεδεμένη Λίστα** (list)
  - **Στοίβα** (stack)
  - **Ουρά** (queue)
- **Δέντρο** (tree)

Πως τα υλοποιούμε στη μνήμη;

Πίνακες

Μονοδιάστατος πίνακας:

α1	α2	α3	α4	α5	α6

Διδιάστατος πίνακας:

α11	α12	α13	α14
α21	α22	α23	α24
α31	α32	α33	α34

γραμμή 1
γραμμή 2
γραμμή 3

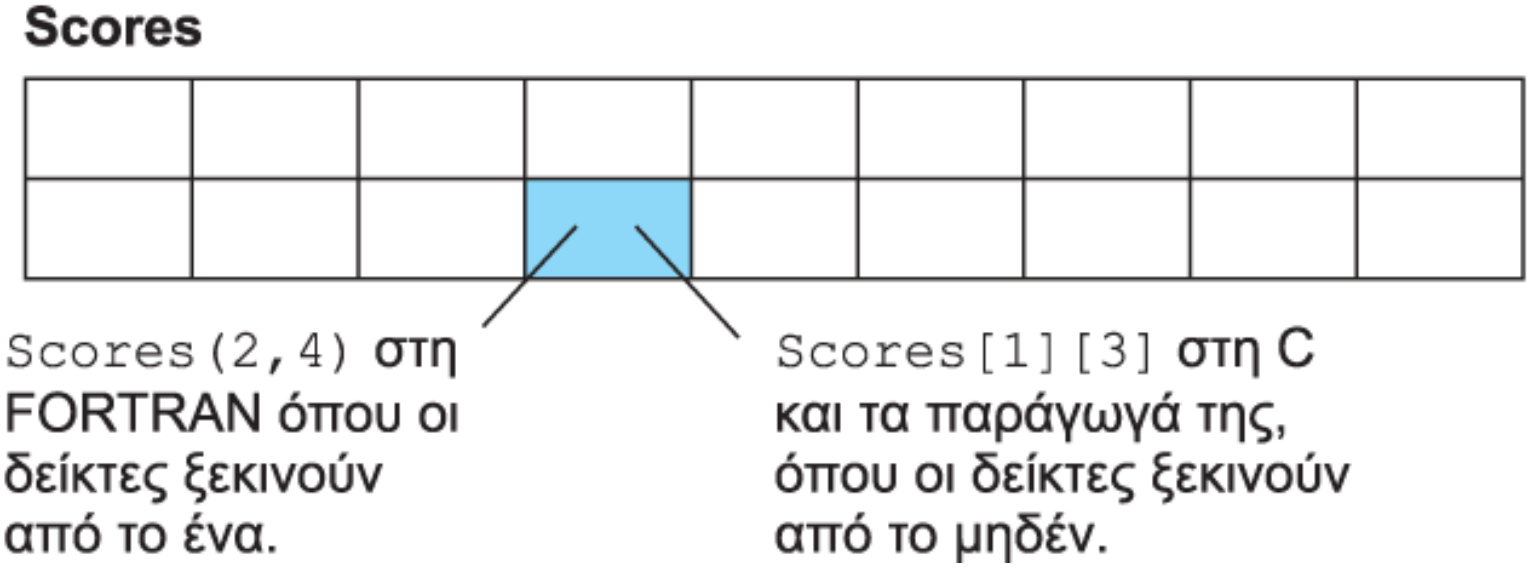
στήλη 1	στήλη 2	στήλη 3	στήλη 4

Παράδειγμα: Δι-διάστατος πίνακας 2 γραμμών και 9 στηλών

Ορισμός πίνακα:

int Scores[2][9] στην C

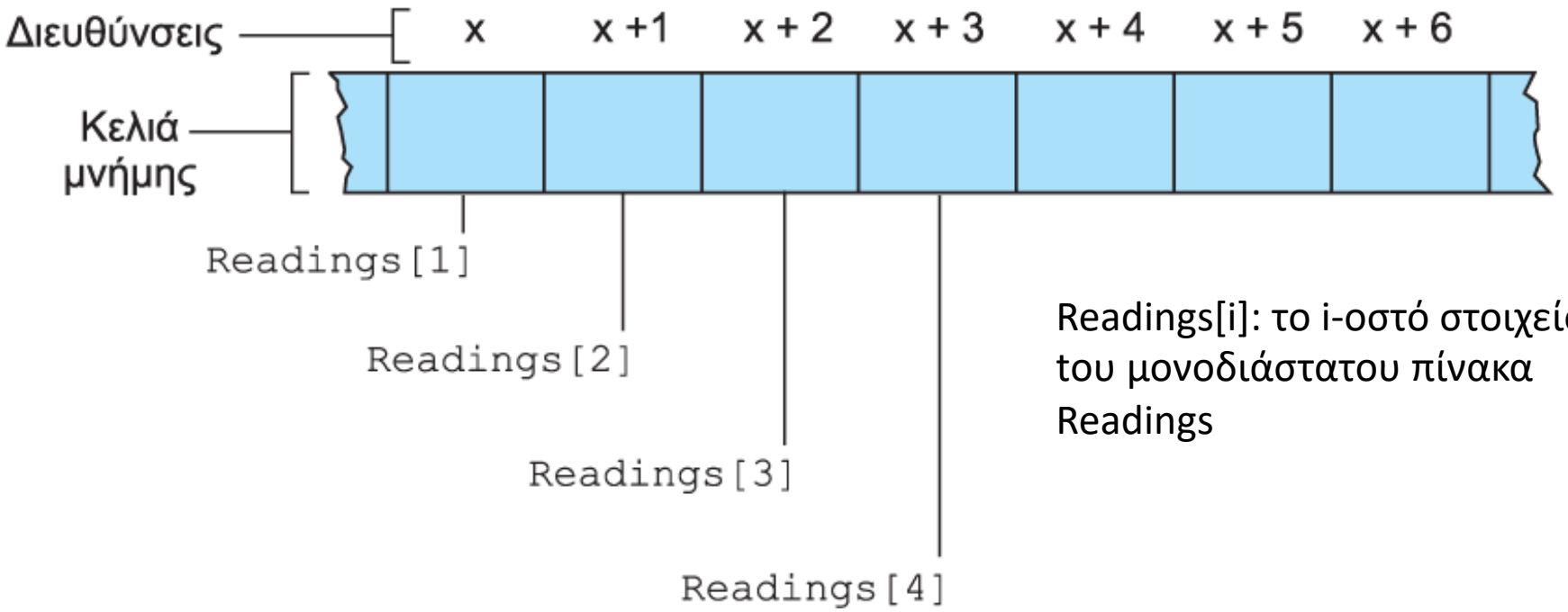
INTEGER Scores(2,9) στην Fortran



- Πάνω αριστερά στοιχείο του πίνακα: Scores[0][0] στην C, Scores(1,1) στην FORTRAN
- Στην Python ξεκινάμε από τη θέση (0,0)

Παράδειγμα: Αποθήκευση μονοδιάστατων ομογενών πινάκων (arrays)

Αποθήκευση μονοδιάστατου πίνακα σε διαδοχικά κελιά μνήμης



Readings[i]: το i-οστό στοιχείο του μονοδιάστατου πίνακα Readings

int Readings[24];
Δήλωση ενός μονοδιάστατου πίνακα 24 στοιχείων

{Διεύθυνση του Readings[i]} = {Διεύθυνση του Readings[0]} + i, αν η αρίθμηση στοιχείων του πίνακα ξεκινάει από το Readings[0]

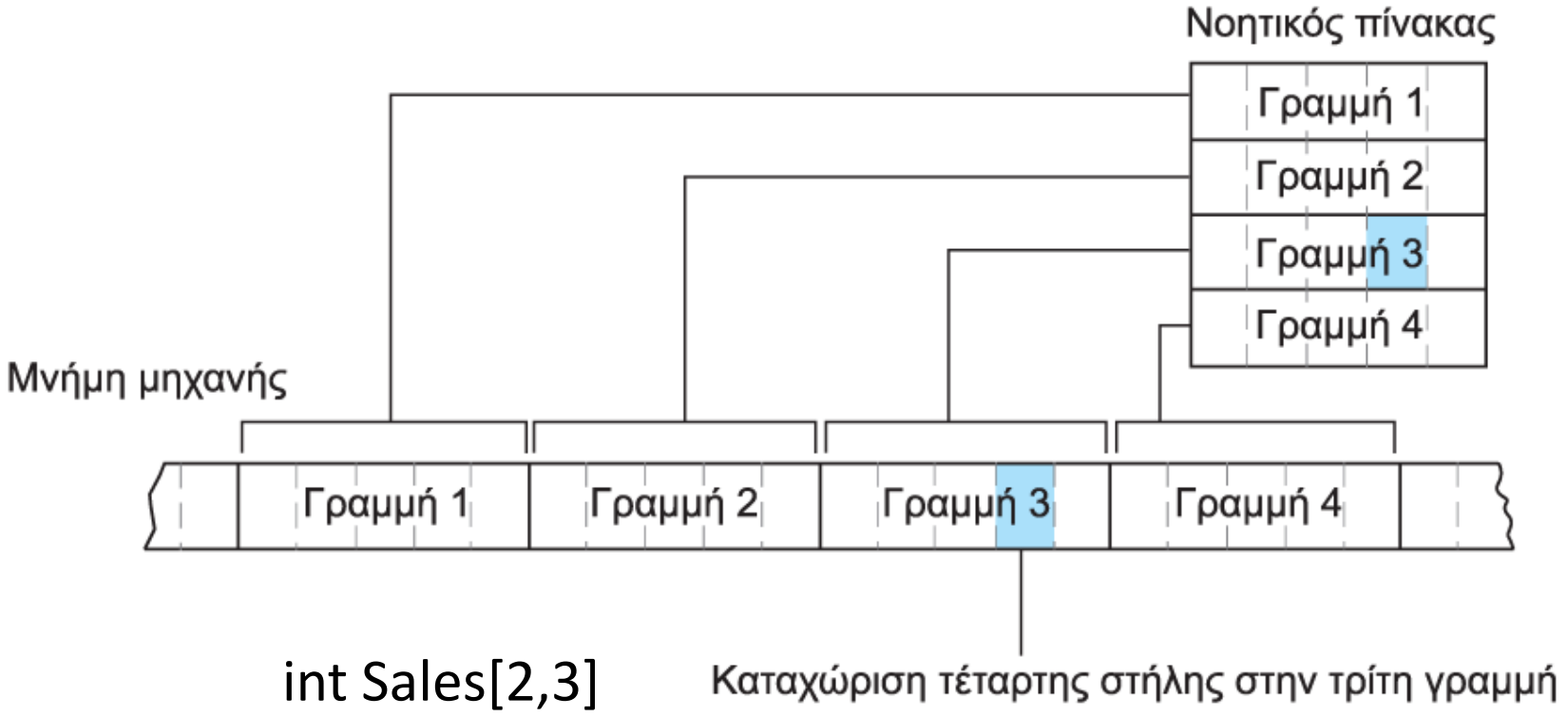
Αποθήκευση δι-διάστατων πινάκων

Πίνακας (array):
2 γραμμές, 3 στήλες

a11	a12	a13
a21	a22	a23

- Απαιτείται επαρκές τμήμα από **συνεχόμενα** κελιά μνήμης για να αποθηκευτούν όλα τα δεδομένα
- Διδιάστατοι πίνακες
  - **Διάταξη κατά γραμμές (row-major order):** κάθε γραμμή του πίνακα αποθηκεύεται μαζί (αποθήκευση γραμμή προς γραμμή)
  - **Διάταξη κατά στήλες (column-major order):** ο πίνακας αποθηκεύεται στήλη προς στήλη

Παράδειγμα: Αποθήκευση δι-διάστατου πίνακα κατά γραμμές



Εντοπισμός / πρόσβαση / αλλαγή τιμής σε συγκεκριμένο στοιχείο

Το στοιχείο A[r][c] βρίσκεται στη διεύθυνση: r x (# στηλών) + (c+1) (αν η αποθήκευση γίνεται κατά γραμμές και ξεκινάει από τη διεύθυνση 1, και η απαρίθμηση στοιχείων του πίνακα ξεκινάει από το στοιχείο [0][0])

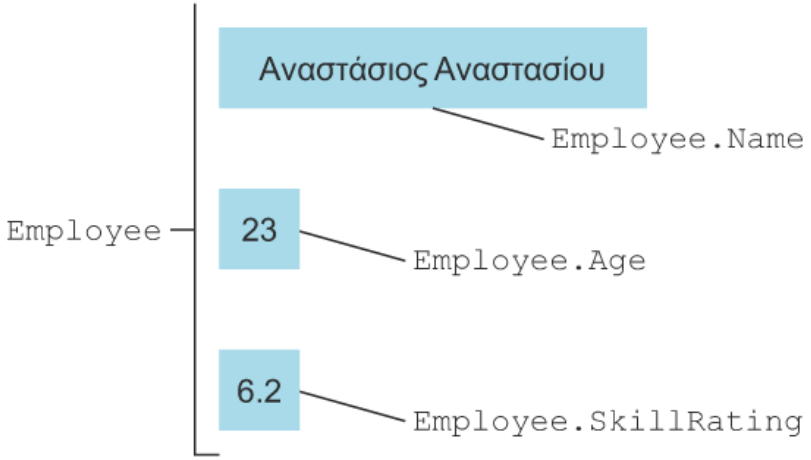
Ετερογενής πίνακας (structure)

Ετερογενής πίνακας ή **δομή** (structure) ή record

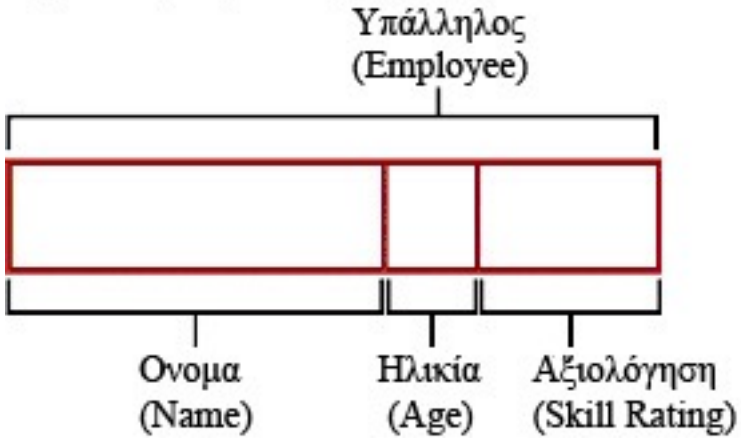
Μπλοκ δεδομένων με στοιχεία (πεδία), ενδεχομένως διαφορετικού τύπου

α. Η δήλωση του πίνακα

```
struct
{ char Name [8];
  int Age;
  float SkillRating;
} Employee;
```



β. Η νοητική διάταξη του πίνακα



Employee.Name
Employee.Age
Employee.SkillRating

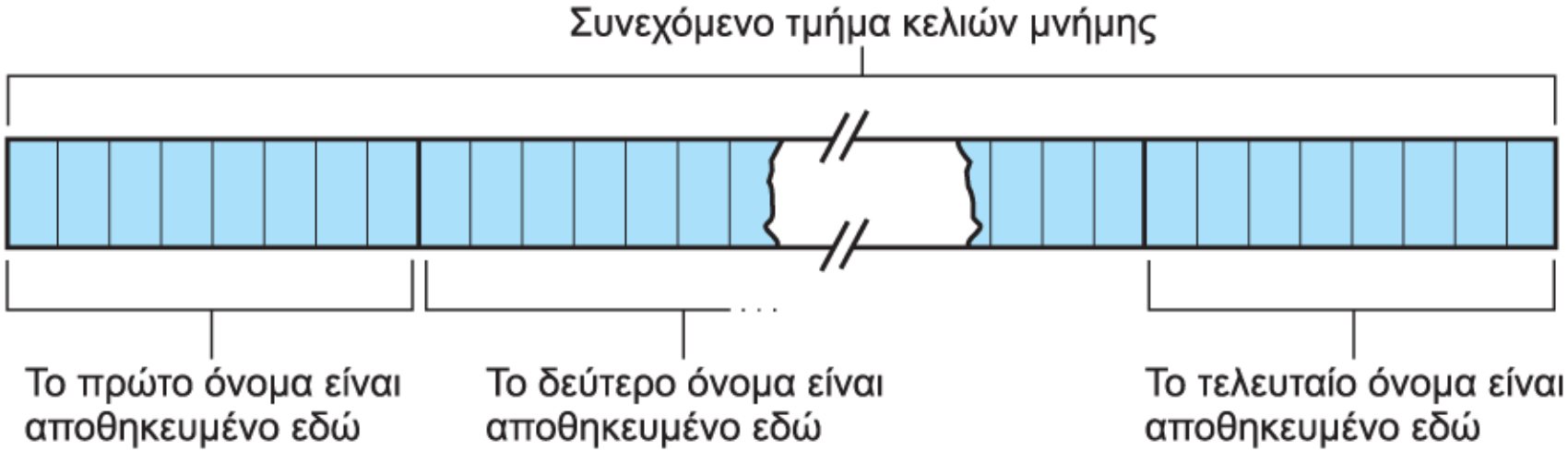
Λίστες

- **Λίστα:** συλλογή δεδομένων με καταχωρήσεις σε σειριακή διάταξη
- Το ένα άκρο της λίστας ονομάζεται **αρχή** (head), ενώ το άλλο άκρο **τέλος** (tail)
- Παραδείγματα
  - ένας διδιάστατος πίνακας μπορεί να θεωρηθεί ως λίστα γραμμών
  - ένα κείμενο θεωρείται ως λίστα από χαρακτήρες

α. Λίστα ονομάτων

γραμμή 1
γραμμή 2
γραμμή 3

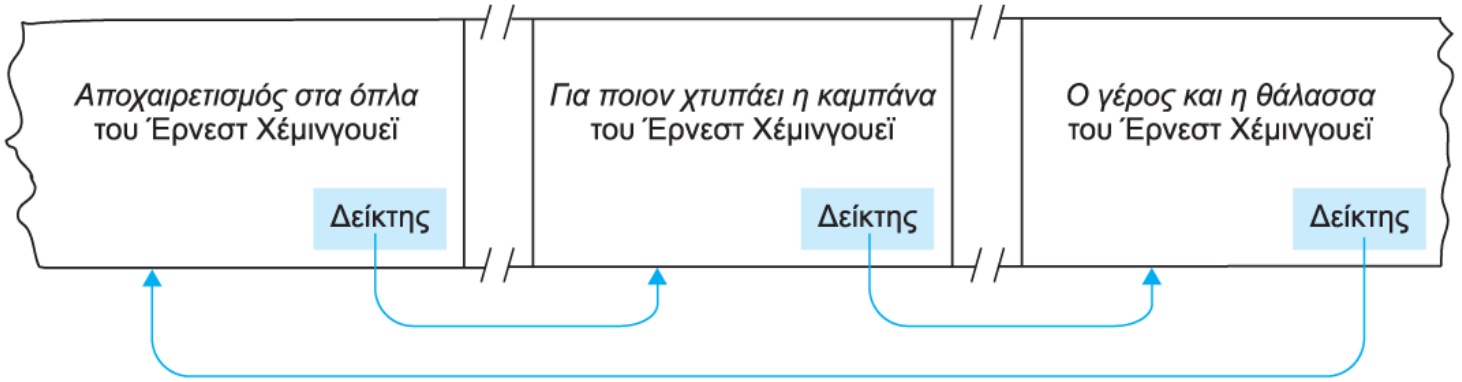
Αποθήκευση λιστών: Συνεχόμενη λίστα



- **Συνεχόμενη λίστα (contiguous list):** λίστα που αποθηκεύεται ως ομογενής πίνακας
  - Τα στοιχεία της αποθηκεύονται διαδοχικά το ένα μετά το άλλο
  - Συνολικά καταλαμβάνουν ένα μεγάλο μπλοκ μνήμης
  - Χρήσιμη για στατική λίστα, **όχι χρήσιμη για δυναμική λίστα** (όπου συχνά συμβαίνουν προσθήκες και αφαιρέσεις στοιχείων)

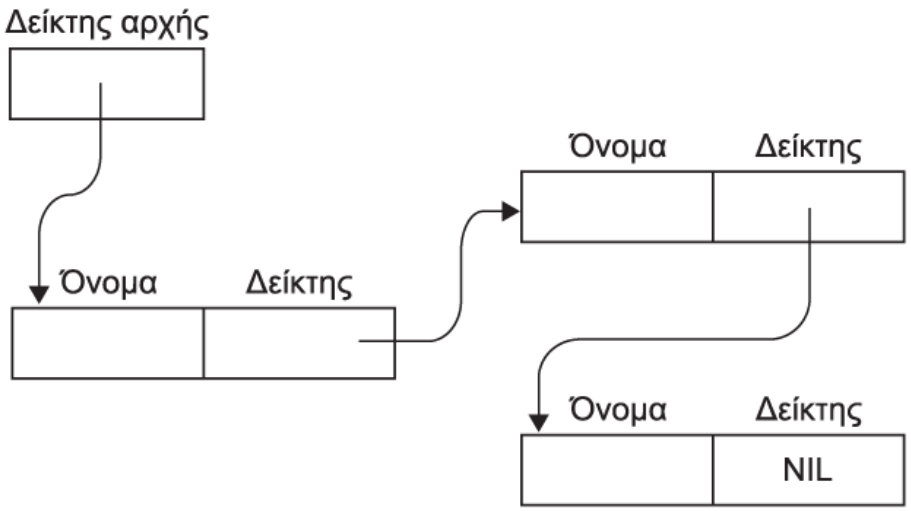
Δείκτες

- Τα κελιά μνήμης αναγνωρίζονται από τις διευθύνσεις τους
- Ως αριθμοί, οι διευθύνσεις αποθηκεύονται σε άλλα κελιά μνήμης
- **Δείκτης** (Pointer): αποθηκευτικός χώρος σε μια θέση μνήμης (στην κυρίως μνήμη ή προσωρινή μνήμη/καταχωρητή) που περιλαμβάνει τον αριθμό μιας άλλης θέσης μνήμης
  - π.χ. Ο Program counter (μετρητής προγράμματος) είναι ένας δείκτης
  - Περιέχει τη διεύθυνση της επόμενης εντολής προς εκτέλεση



- Συνδεδεμένη λίστα: Λίστα που χρησιμοποιεί δείκτες
  - Οι δείκτες διασυνδέουν ομοειδή δεδομένα (π.χ. μυθιστορήματα)
  - Χρειάζεται ένα επιπλέον κελί μνήμης για κάθε δεδομένο (μυθιστόρημα), για αποθήκευση της **διεύθυνσης** του επόμενου μυθιστορήματος στην λίστα

Αποθήκευση λιστών: Συνδεδεμένη λίστα

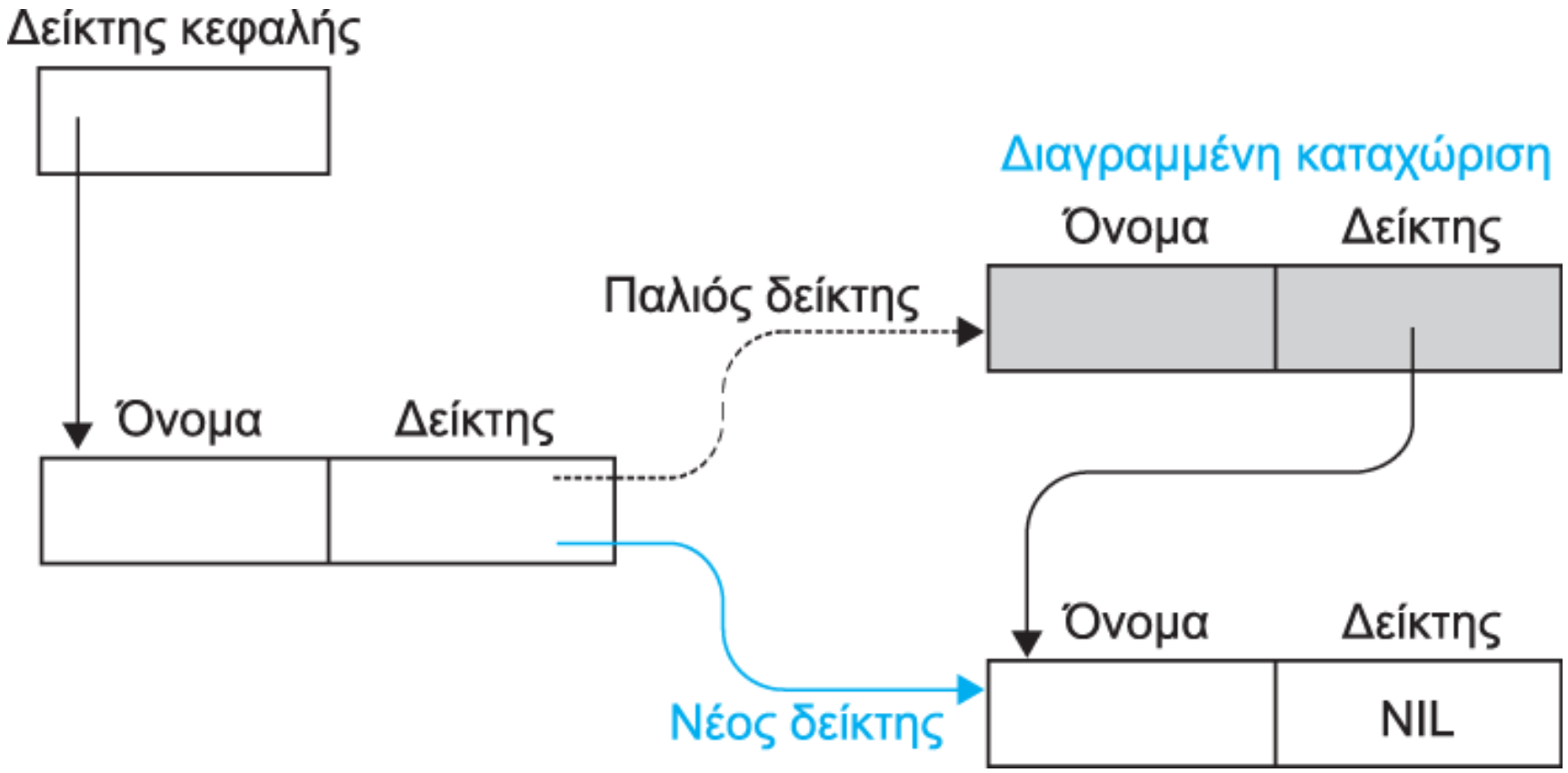


- **Συνδεδεμένη λίστα (linked list):** λίστα στην οποία κάθε κόμβος δείχνει στον επόμενο του, με δείκτες (pointers)
  - **Δείκτης αρχής:** ο δείκτης που δείχνει στη αρχή (κεφαλή) της λίστας.
  - **Δείκτης NIL**: ειδικό σχήμα bit που τοποθετείται στο κελί-δείκτη της τελευταίας καταχώρισης για να σημάνει ότι δεν υπάρχουν άλλες καταχωρίσεις στη λίστα

(π.χ.) 8 κυψέλες μνήμης για το όνομα + 1 κυψέλη για τον Pointer (διεύθυνση του επόμενου στοιχείου στη λίστα)

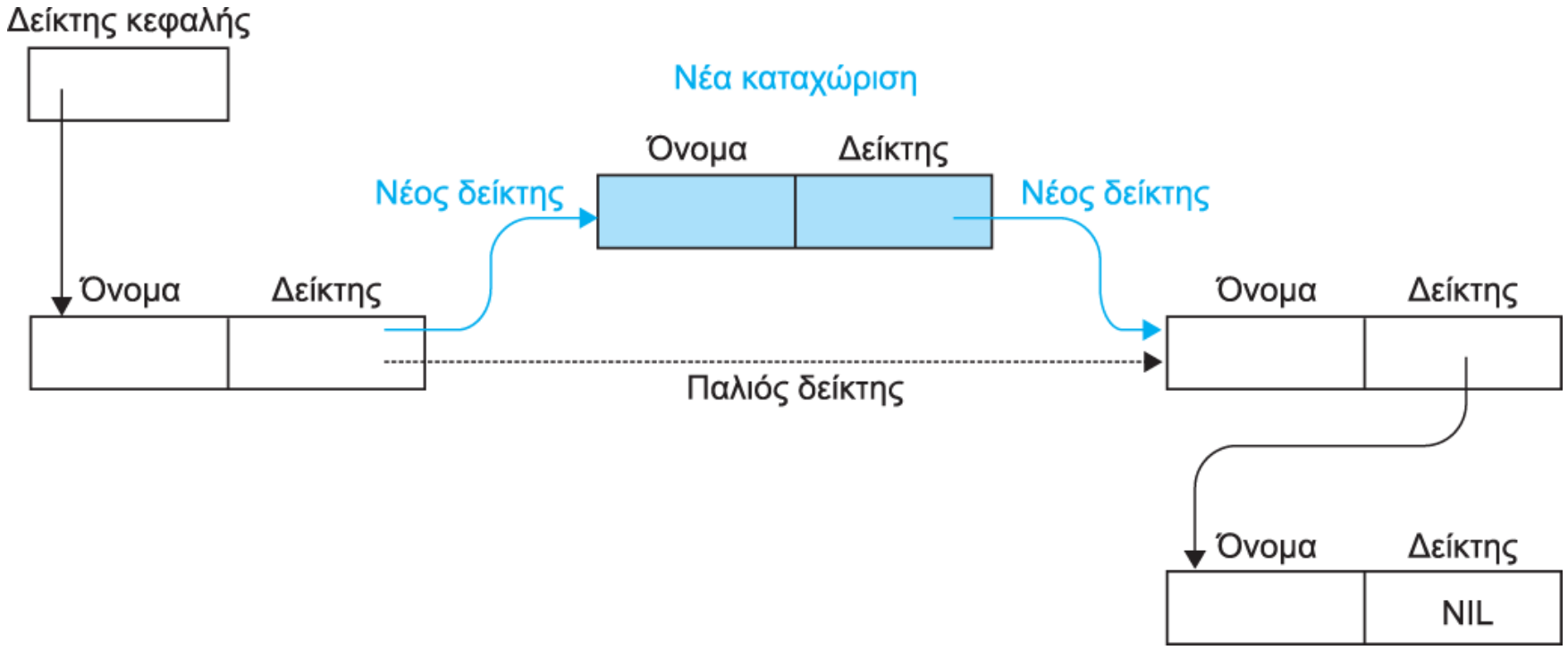
- Κινούμαι από στοιχείο σε στοιχείο της λίστας **ακολουθώντας κάθε φορά τον δείκτη**
- Τα στοιχεία είναι αποθηκευμένα διάσπαρτα μέσα στη μνήμη
- Χρήσιμο για δυναμικές λίστες που αυξομειώνονται
- Δείκτες μπορεί να χρησιμοποιηθούν και στους πίνακες.

Διαγραφή καταχώρισης από συνδεδεμένη λίστα



Αλλαγή **pointer του προηγούμενου στοιχείου** της λίστας από αυτό που θέλω να διαγραφεί, ώστε να δείχνει στο επόμενο στοιχείο από αυτό που θέλω να διαγραφεί

Προσθήκη καταχώρισης σε συνδεδεμένη λίστα



Ο **pointer της νέας καταχώρησης** να δείχνει στην καταχώρηση που θα βρίσκεται μετά τη νέα καταχώρηση.

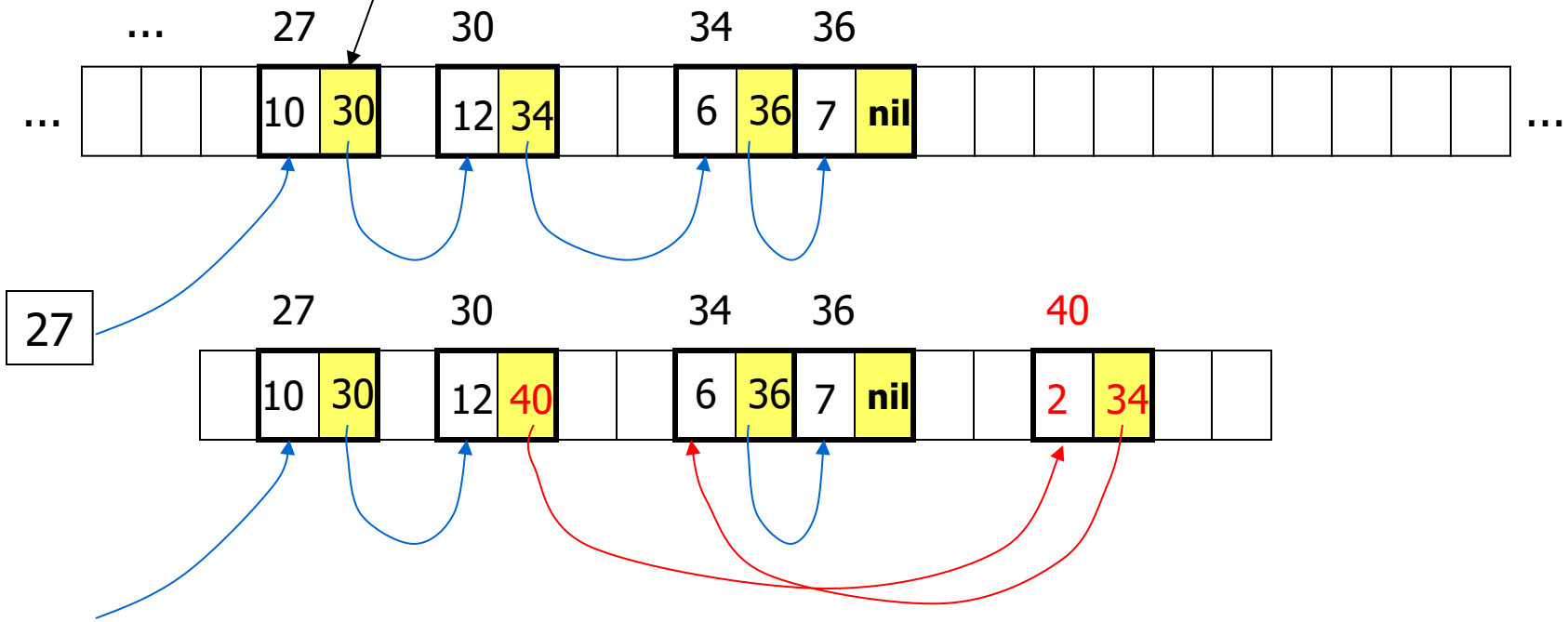
Ο **pointer της καταχώρησης** που θα βρίσκεται **πριν από την νέα**, να δείχνει στη νέα καταχώρηση.

Προσθήκη καταχώρισης σε συνδεδεμένη λίστα - Παράδειγμα

Λίστα: ακολουθία από records (int ή πιο σύνθετα records, βλ. παρακάτω)

Λίστα: 10, 12, 6, 7 Βασική λειτουργία: Προσθήκη νέους μέλους: 10, 12, 2, 6, 7

pointer (δείκτης): Γράφει την διεύθυνση στη μνήμη όπου βρίσκεται το επόμενο στοιχείο της λίστας



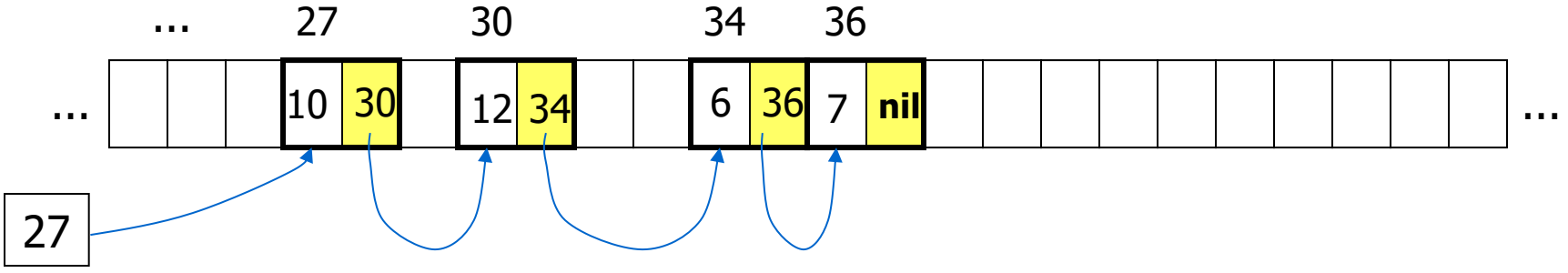
Διαδικασία για εκτύπωση μιας συνδεδεμένης λίστας

διαδικασία PrintList(Λίστα)

ΤρέχωνΔείκτης ← δείκτης κεφαλής της Λίστας

όσο (ΤρέχωνΔείκτης διάφορος του NIL) **κάνε**

- Τύπωσε το όνομα της καταχώρισης στην οποία δείχνει ο ΤρέχωνΔείκτης
- Βρες την τιμή στο κελί-δείκτη της καταχώρισης της Λίστας όπου δείχνει ο ΤρέχωνΔείκτης και ανάθεσε τον ΤρέχονταΔείκτη την τιμή αυτή.

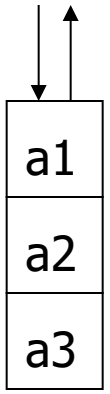


ΤρέχωνΔείκτης = ~~27~~ ~~30~~ ~~34~~ ~~36~~ nil

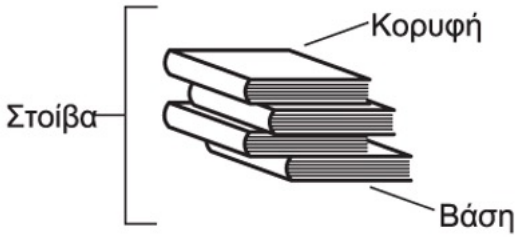
Εκτυπώνεται: 10 12 6 7

Στοίβες

- **Στοίβα:** λίστα της οποίας οι καταχωρήσεις αφαιρούνται και εισάγονται μόνο από την αρχή
  - Δομή **LIFO** (Last In, First Out)
- Η αρχή της στοίβας ονομάζεται **κορυφή** (top) της στοίβας, ενώ το τέλος ονομάζεται **βάση** (base)
- **Απώθηση** (pop): αφαίρεση μίας καταχώρησης από την κορυφή
- **Ώθηση** (push): εισαγωγή νέας καταχώρησης στην κορυφή
- Χρήσιμη για περιπτώσεις όπου αντικείμενα ανακτώνται με σειρά αντίστροφη από αυτήν με την οποία εισάγονται
  - π.χ. Depth first search (DFS) algorithm



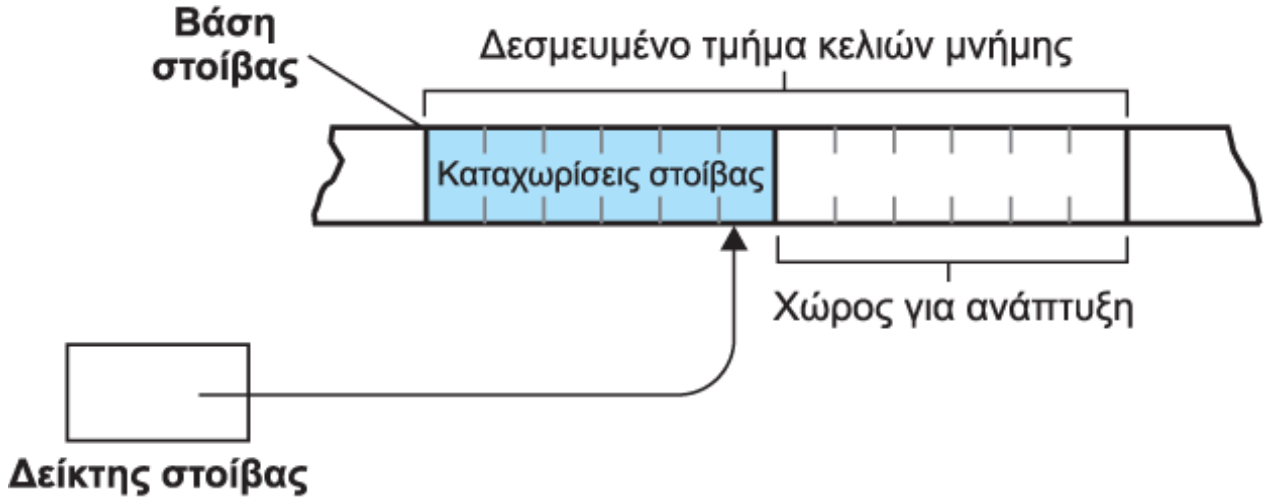
Στοίβα



β. Στοίβα βιβλίων

Αποθήκευση στοιβών

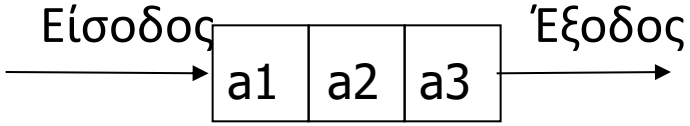
- Δέσμευση ενός μέρους της μνήμης για την στοίβα
  - Η απόφαση για το πόση μνήμη θα δεσμευτεί είναι σημαντική
  - Μικρός χώρος: μπορεί να εξαντληθεί γρήγορα η μνήμη
  - Μεγάλος χώρος: σπατάλη μνήμης, αν δεν υπάρχουν αρκετά δεδομένα προς αποθήκευση
- Δείκτης στοίβας (stack pointer): διεύθυνση της κορυφής της στοίβας
  - Push: ο pointer μετακινείται στο κενό δεξιά από την τωρινή θέση του και το νέο στοιχείο εισάγεται
  - Pop: ο pointer μετακινείται στο στοιχείο αριστερά από την τωρινή θέση του, και το στοιχείο βγαίνει από τη στοίβα

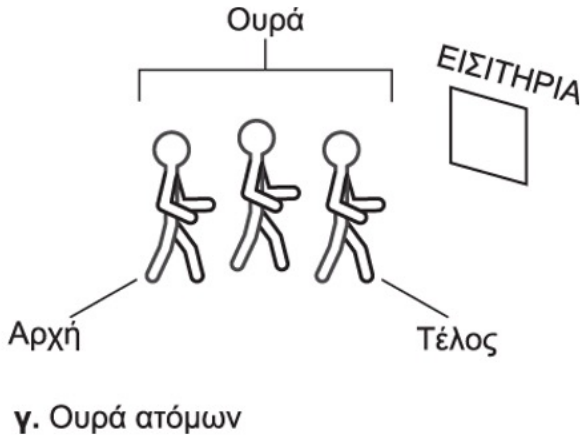


Ουρές

- **Ουρά:** λίστα στην οποία οι πιο παλιές καταχωρήσεις αφαιρούνται από την αρχή της
  - Οι νέες καταχωρήσεις προστίθενται στο τέλος
  - **FIFO** (First In, First Out)
- Χρήσιμη για περιπτώσεις όπου αντικείμενα ανακτώνται με τη σειρά που εισάγονται
  - π.χ. Breadth first search (BFS) algorithm
  - π.χ. Χρήσιμη ως δομή για προσωρινή μνήμη (buffer)

Ουρά:

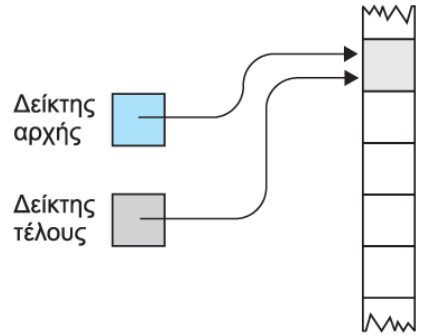




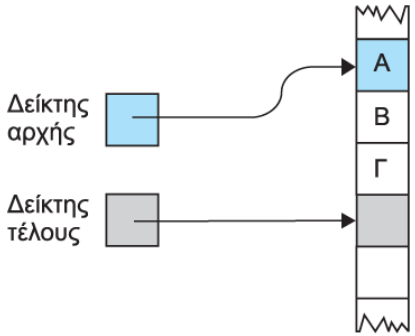
γ. Ουρά ατόμων

Αποθήκευση Ουρών

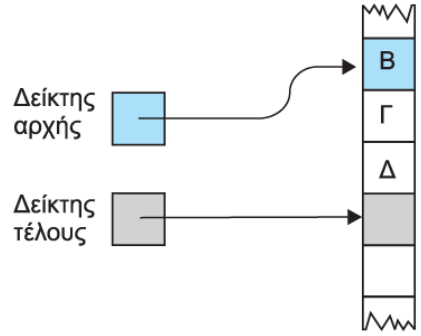
Συνεχές μπλοκ μνήμης
Δείκτης αρχής (head pointer), δείκτης τέλους (tail pointer)



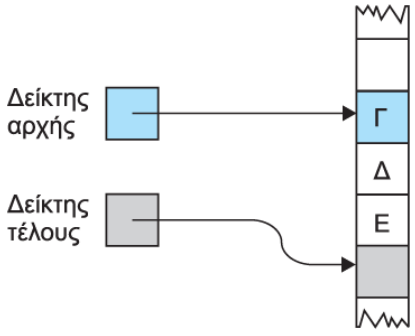
α. Άδεια ουρά



β. Μετά την εισαγωγή των καταχωρίσεων Α, Β, και Γ



γ. Μετά την αφαίρεση του Α και την εισαγωγή του Δ

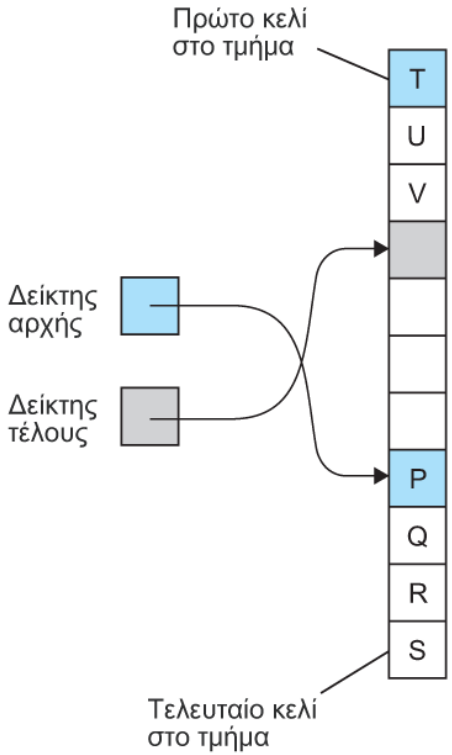


δ. Μετά την αφαίρεση του Β και την εισαγωγή του Ε

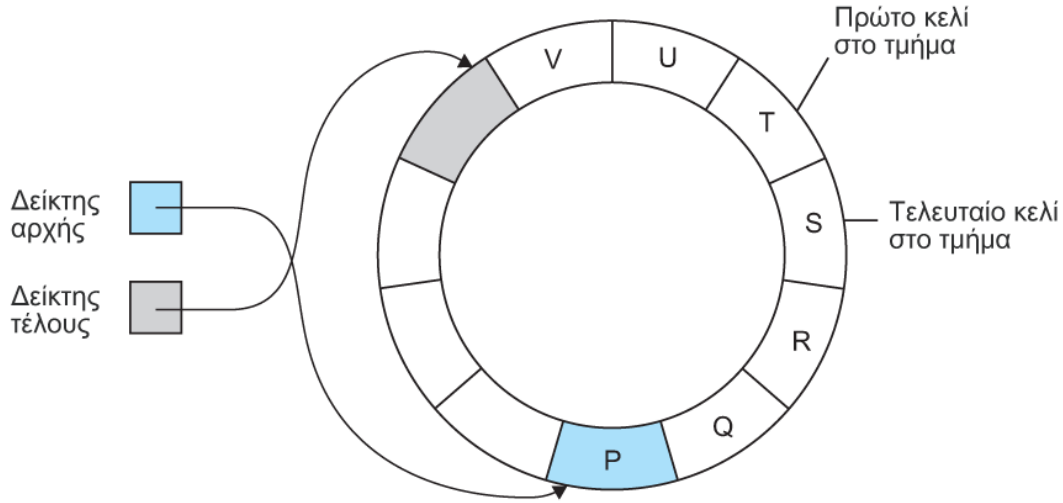
- **Εισαγωγή νέου στοιχείου** στη θέση που δείχνει ο δείκτης τέλους (ο οποίος μετά την εισαγωγή μετακινείται στην επόμενη θέση)
- **Εξαγωγή στοιχείου** από τη θέση που δείχνει ο δείκτης αρχής (ο οποίος μετά την εξαγωγή μετακινείται στην επόμενη θέση)
- Παρατηρήστε πώς «γλιστράει» στη μνήμη η ουρά καθώς προστίθενται και αφαιρούνται καταχωρήσεις

Κυκλική ουρά

Κυκλική ουρά : Προστατεύει την ουρά από το να ξεφύγει εκτός του τμήματος μνήμης που έχει δεσμευτεί για αυτήν

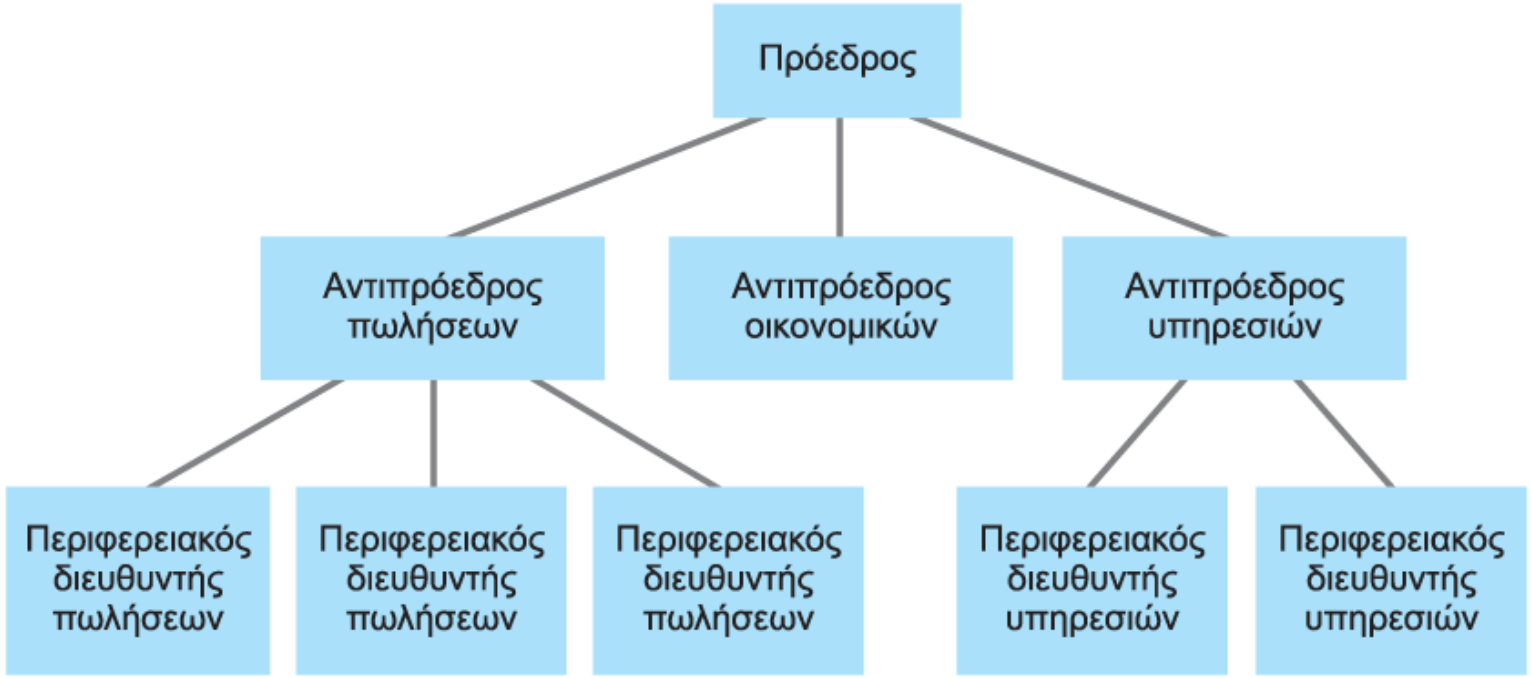


α. Ουρά όπως είναι πραγματικά αποθηκευμένη



β. Σχηματική αποθήκευση με το τελευταίο κελί να είναι "γειτονικό" του πρώτου κελιού

Δέντρα

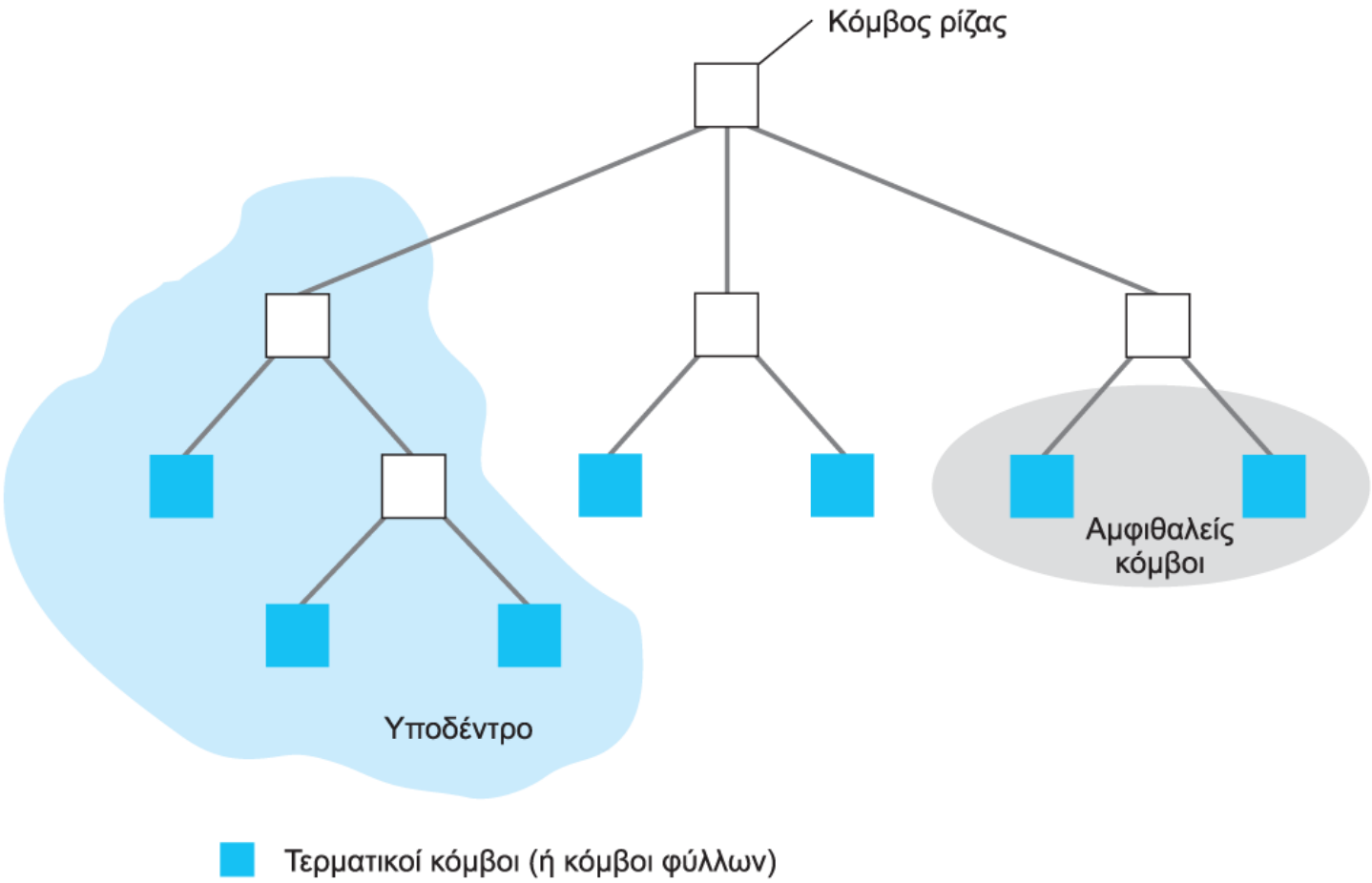


- **Δέντρο:** ομάδα δεδομένων όπου οι καταχωρίσεις έχουν ιεραρχική οργάνωση
- **Κόμβος (node):** η κάθε θέση σε ένα δέντρο
- **Κόμβος-ρίζα (root node):** ο κόμβος στην κορυφή
- **Τερματικός κόμβος (terminal node)** ή **φύλλο (leaf):** ο κόμβος που βρίσκεται στο κάτω κάτω μέρος
- **Βάθος (depth):** η μεγαλύτερη διαδρομή από τη ρίζα μέχρι ένα φύλλο.

Ορολογία δέντρων (2)

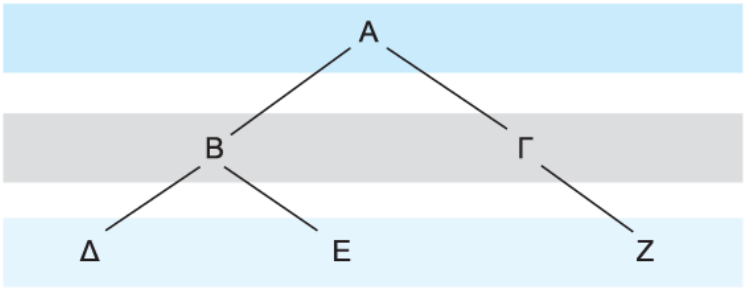
- **Γονικός κόμβος** (γονέας) **(parent):** ο αμέσως προηγούμενος (από πάνω) κόμβος από έναν συγκεκριμένο κόμβο
- **Θυγατρικός κόμβος** (παιδί) **(child):** ο αμέσως επόμενος (από κάτω) κόμβος από έναν συγκεκριμένο κόμβο
- **Πρόγονος (ancestor)** ενός κόμβου**:** ο γονέας, ο γονέας του γονέα κλπ.
- **Απόγονος (descendant)** ενός κόμβου**:** το παιδί, το παιδί του παιδιού κτλ.
- **Αμφιθαλείς** (ή κόμβοι-αδέρφια) **(siblings):** δύο κόμβοι με τον ίδιο γονέα.
- **Δυαδικό δένδρο (binary tree):** δένδρο στο οποίο κάθε κόμβος έχει μέχρι δύο θυγατρικούς κόμβους.
  - Έχουμε επίσης τριαδικά δέντρα (ternary trees) κλπ. αλλά πολύ σπανιότερα

Ορολογία δέντρων (3)

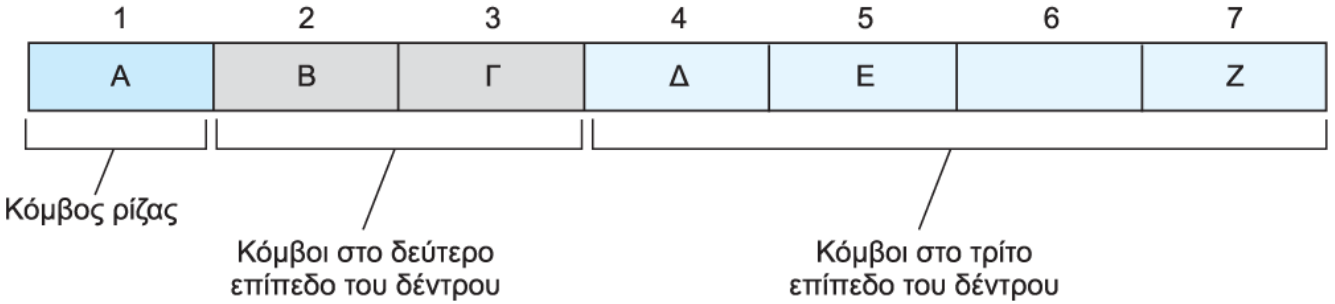


Αποθήκευση Δυαδικών Δέντρων σε συνεχές τμήμα μνήμης

Σχηματική αναπαράσταση του δέντρου



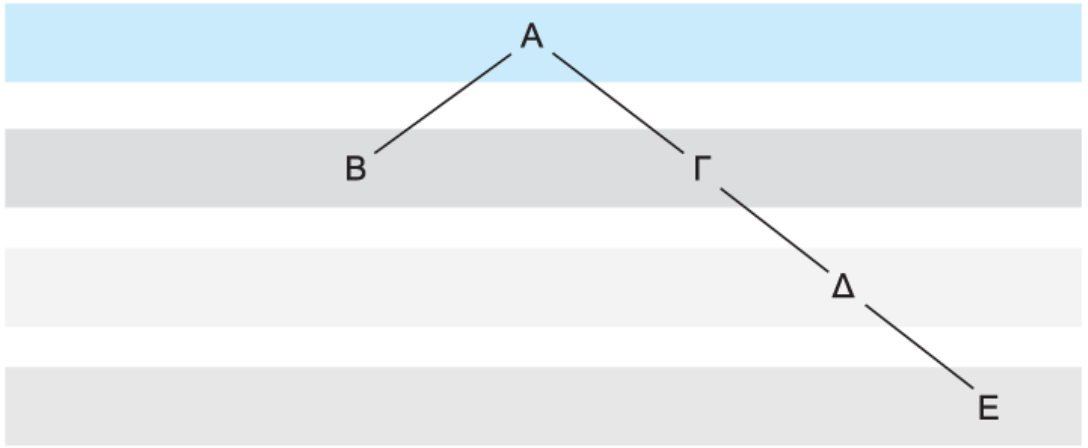
Πραγματική οργάνωση της αποθήκευσης



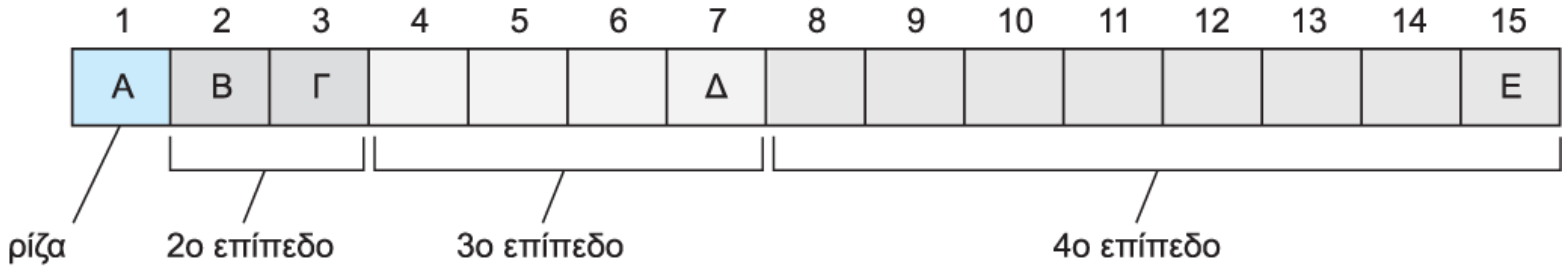
- Το περιεχόμενο των κόμβων του δέντρου με κατεύθυνση από πάνω προς τα κάτω (**χωρίς τους δείκτες**) αποθηκεύεται σειριακά στην μνήμη, επίπεδο προς επίπεδο, από αριστερά προς τα δεξιά
- Οι θυγατρικοί κόμβοι του κόμβου στη θέση μνήμης n αποθηκεύονται στις θέσεις 2n, 2n+1
- Ο γονέας ενός κόμβου στη θέση μνήμης n είναι στην θέση n div 2
- Ο αδερφός κόμβος ενός κόμβου στη θέση μνήμης n είναι στην θέση n+1 αν n άρτιος ή στη θέση n-1 αν n περιττός

Μη αποδοτική αποθήκευση δέντρου σε συνεχές τμήμα μνήμης

Σχηματική αναπαράσταση του δέντρου



Πραγματική οργάνωση της αποθήκευσης



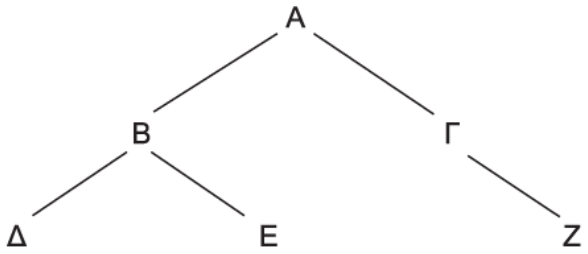
Αραιό (sparse) και μη ισορροπημένο δέντρο → μη αποδοτικός τρόπος αποθήκευσης του δέντρου σε συνεχές τμήμα μνήμης

Αποθήκευση Δυαδικών Δέντρων με δείκτες

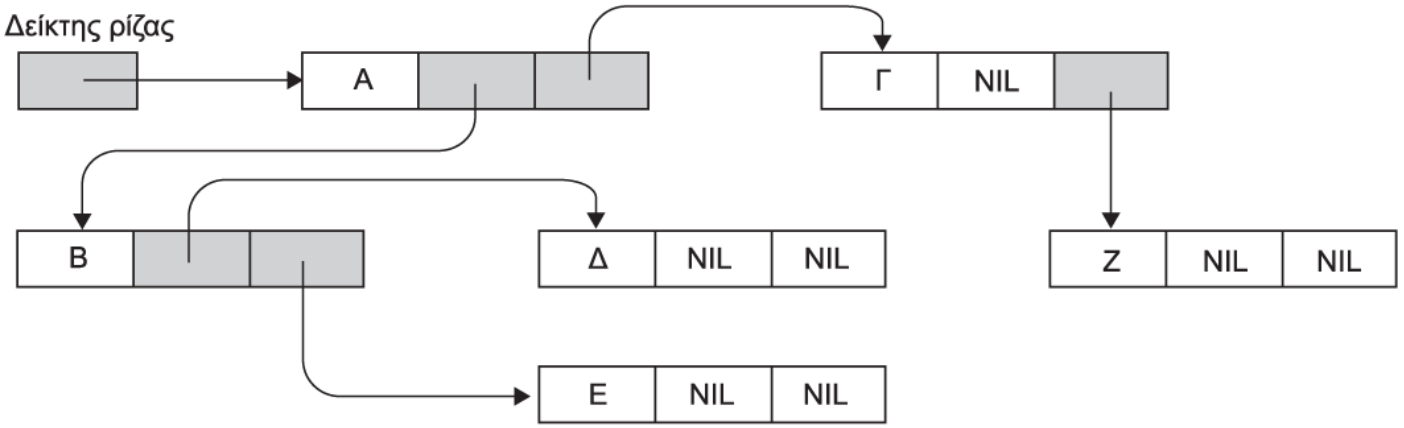
Κόμβος

Θέσεις (κελιά) μνήμης για:
δεδομένα + Δείκτη στον αριστερό θυγατρικό κόμβο + Δείκτη στον δεξιό θυγατρικό κόμβο

Σχηματική αναπαράσταση του δέντρου

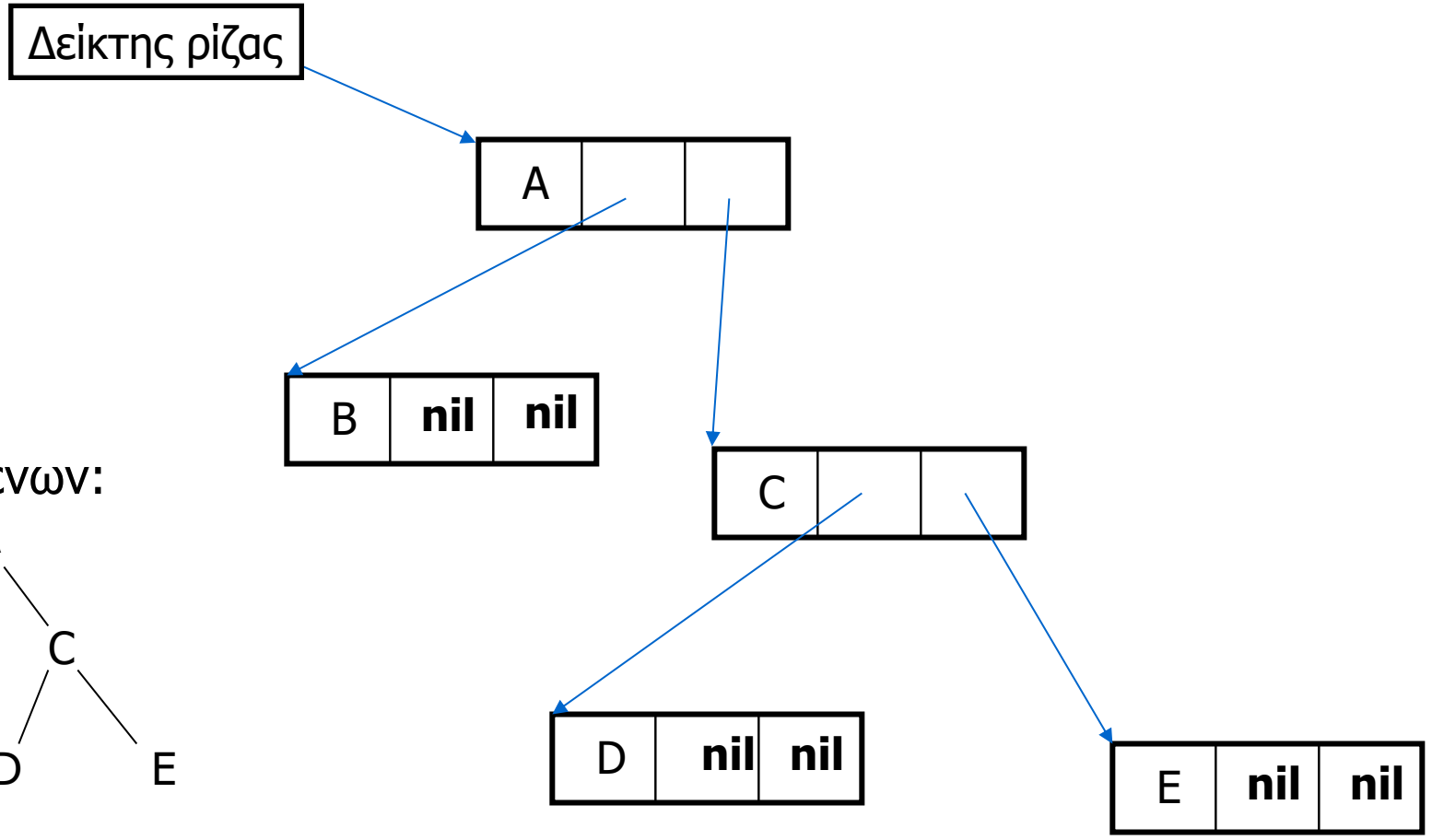


Πραγματική οργάνωση της αποθήκευσης



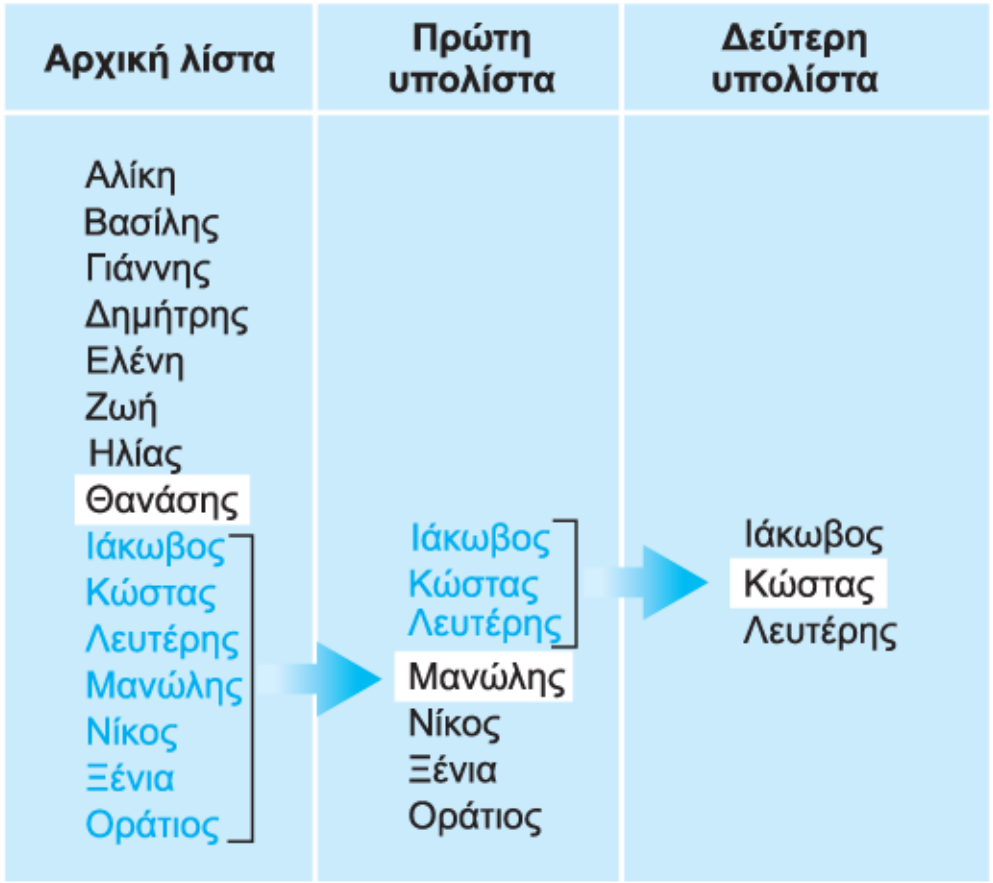
Παράδειγμα: Δένδρα

Υλοποίηση στη μνήμη:



Δομή δεδομένων:

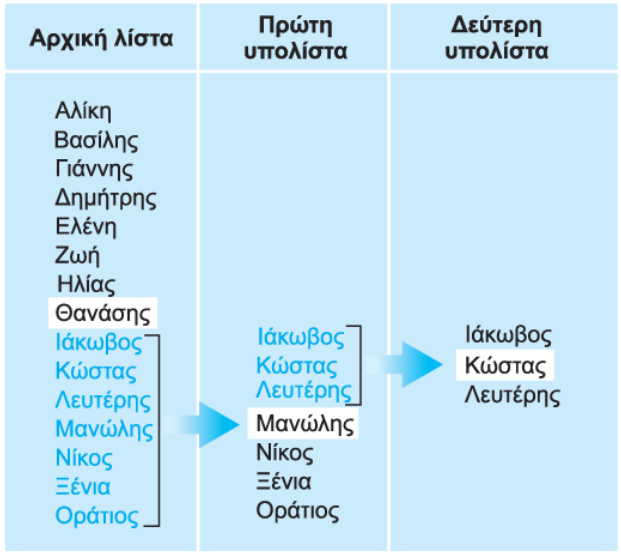
Επίδραση Δομών Δεδομένων στο σχεδιασμό αλγορίθμων: Δυαδική Αναζήτηση

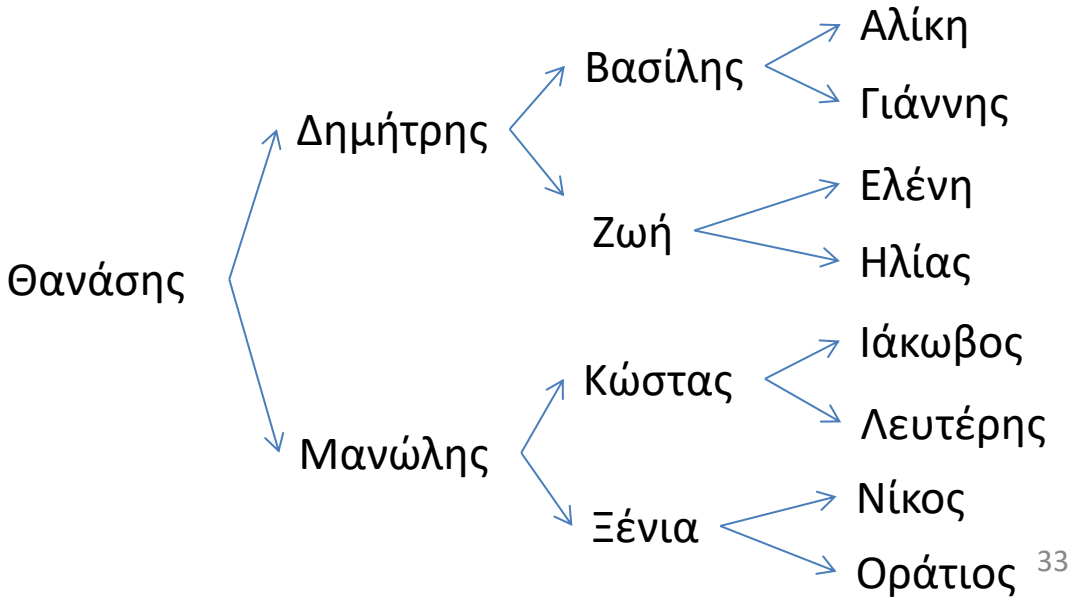


Είσοδος: Λίστα που είναι ταξινομημένη αλφαβητικά

Αποθηκευση ταξινομημένης λίστας ως δυαδικό δέντρο (1)

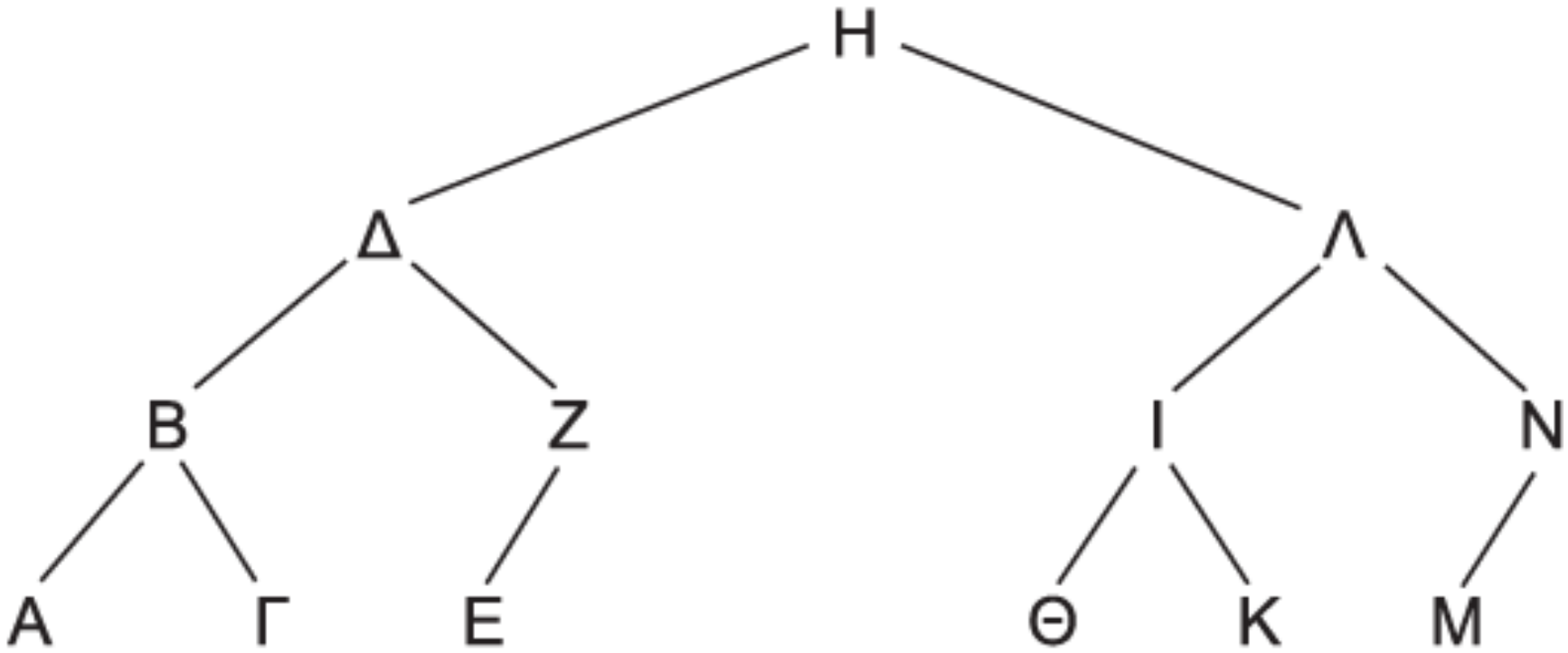
- Υπενθύμιση: για να αναζητήσω ένα στοιχείο σε μια λίστα διατρέχω διαδοχικά τα μέσα υπο-λιστών
  - Της πάνω ή της κάτω υπο-λίστας σε σχέση με ένα στοιχείο
- Ιδέα: Αποθηκεύω τα **μέσα** των υπο-λιστών αυτών ως παιδιά του στοιχείου που είναι στο μέσον της αρχικής λίστας
- Π.χ. Η παραπάνω λίστα θα αποθηκευτεί ως δέντρο εξής:





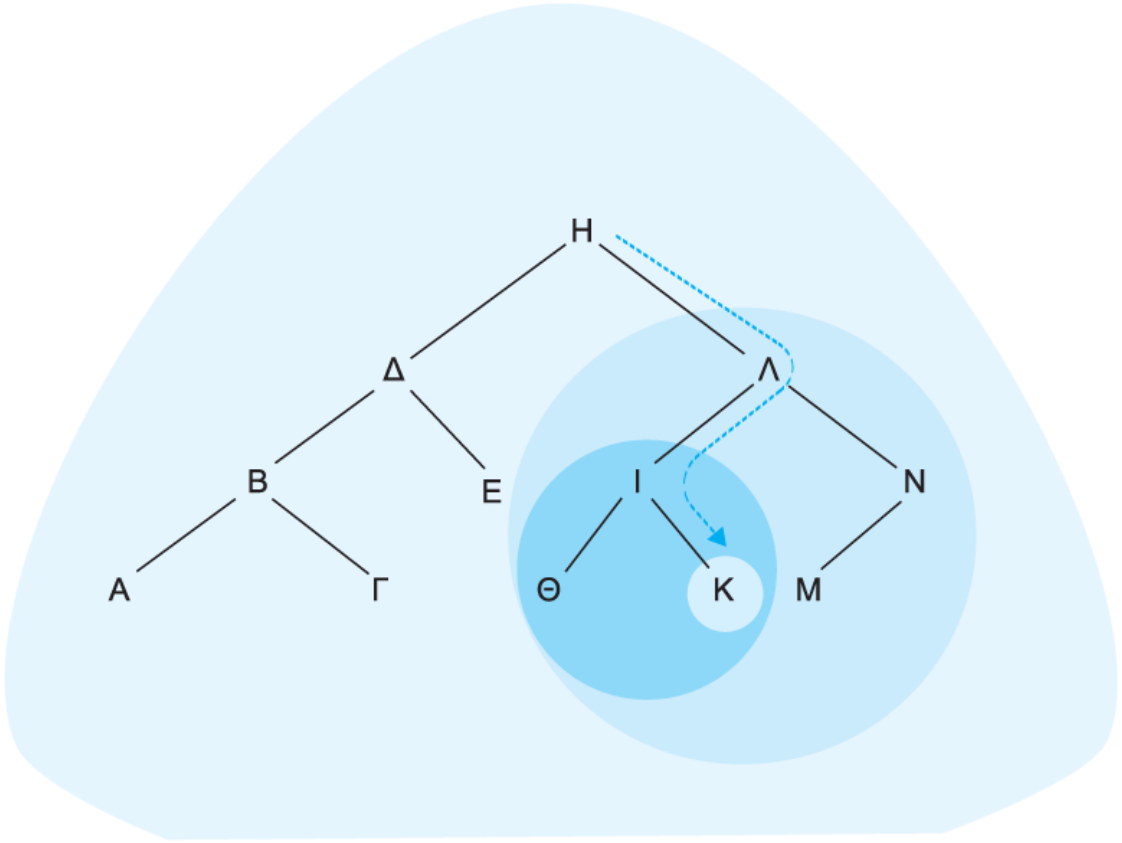
Αποθήκευση ταξινομημένης λίστας ως δυαδικό δέντρο (2)

- Ο τρόπος που θα αποθηκεύσω τη λίστα (ως δέντρο) διευκολύνει την αναζήτηση
- Δυαδική αναζήτηση στη λίστα Α,Β,Γ,Δ,Ε,Ζ,Η,Θ,Ι,Κ,Λ,Μ,Ν



- Το μέσον της λίστας αποθηκεύεται στην ρίζα του δέντρου (Η)
- Το μέσον της πάνω υπο-λίστας του Η (Δ) αποθηκεύεται ως το αριστερό παιδί του Η
- Το μέσον της κάτω υπο-λίστας του Η (Λ) αποθηκεύεται ως το δεξιό παιδί του Η
- κοκ

Παράδειγμα 1: Δυαδική αναζήτηση σε Λίστα που είναι αποθηκευμένη ως Δυαδικό Δέντρο



Αναζήτηση σε διαδοχικά μικρότερα υπο-δέντρα. Πολυπλοκότητα;

- **Χειρότερη** περίπτωση: O(n) (non-balanced δεντρο)
- **Μέση** περίπτωση: O(log n) (balanced δεντρο)

Ψευδοκώδικας για Παράδειγμα 1

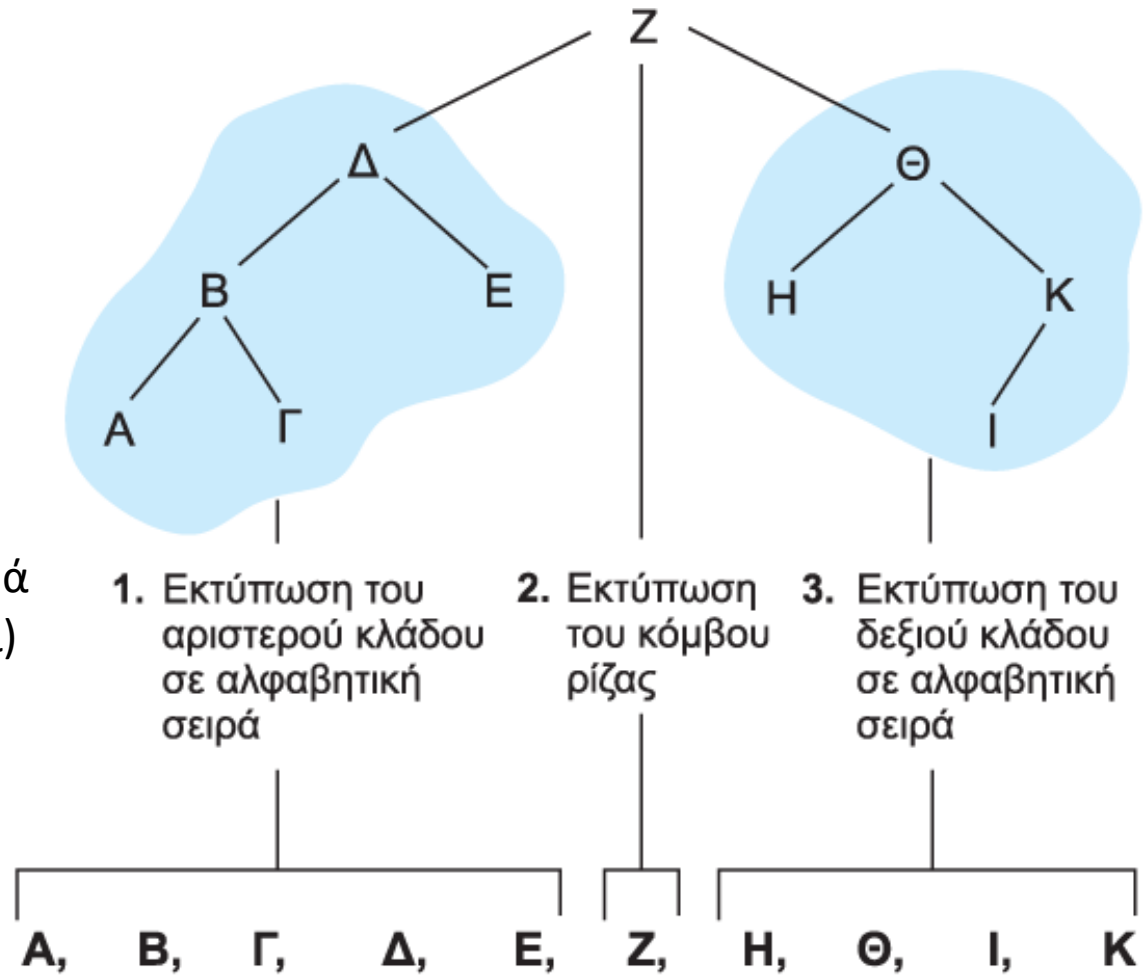
διαδικασία Αναζήτηση (Δέντρο, ΤιμήΣτόχος)
αν (δείκτης ρίζας Δέντρου = NIL)
τότε
(δήλωσε την αναζήτηση ως ανεπιτυχή)
αλλιώς
περίπτωση 1: ΤιμήΣτόχος = τιμή κόμβου ρίζας
(Ανάφερε ότι η αναζήτηση πέτυχε)
περίπτωση 2: ΤιμήΣτόχος < τιμή κόμβου ρίζας
(Εφάρμοσε τη διαδικασία Αναζήτησης για να δεις αν
η ΤιμήΣτόχος βρίσκεται στο υποδέντρο που
προσδιορίζεται από τον αριστερό θυγατρικό
κόμβο της ρίζας, και ανάφερε το αποτέλεσμα
αυτής της αναζήτησης)
περίπτωση 3: ΤιμηΣτόχος > τιμή κόμβου ρίζας
(Εφάρμοσε τη διαδικασία Αναζήτησης για να δεις αν
η ΤιμήΣτόχος βρίσκεται στο υποδέντρο που
προσδιορίζεται από τον δεξιό θυγατρικό
κόμβο της ρίζας, και ανάφερε το αποτέλεσμα
αυτής της αναζήτησης)
) **τέλος αν**

Παράδειγμα 2: Εκτύπωση δέντρου αναζήτησης σε αλφαβητική σειρά

Αναδρομική διαδικασία για εκτύπωση σε αλφαβητική σειρά

OrderedPrint(Tree)
if Tree not empty
then OrderedPrint(left subtree)
print the root node
OrderedPrint(right subtree)

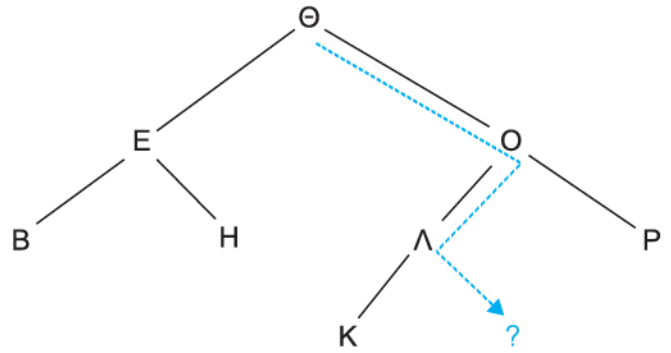
Πολυπλοκότητα;
O(n)
(η OrderedPrint καλείται μία φορά για κάθε κόμβο που εκτυπώνεται)



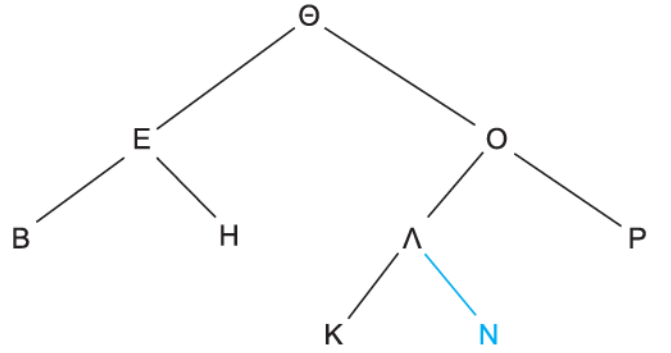
Παράδειγμα 3: Εισαγωγή νέας καταχώρισης Ν σε ταξινομημένη λίστα Β,Ε,Η,Θ,Κ,Λ,Ο,Ρ που έχει αποθηκευτεί ως Δυαδικό Δέντρο

α. Αναζήτηση για τη νέα καταχώριση μέχρι να διαπιστωθεί η απουσία της

+ N →



β. Αυτή είναι η θέση στην οποία πρέπει να προστεθεί η νέα καταχώριση



Η καταχώριση εισάγεται στη θέση που θα κατείχε αν υπήρχε στην λίστα (οπότε και θα είχε ανακαλυφθεί από τον αλγόριθμο δυαδικής αναζήτησης)
Πολυπλοκότητα στη χειρότερη περίπτωση: O(n)

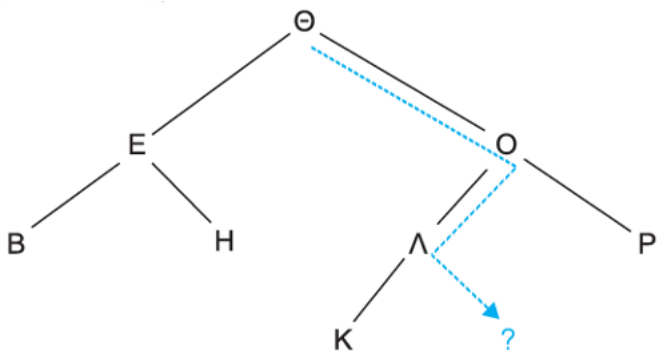
Ερώτηση: Πώς μπορείτε να χρησιμοποιήσετε τα παραπάνω για να κατασκευάσετε έναν αλγόριθμο ταξινόμησης; Ποια η πολυπλοκότητά του;

- Δημιουργία δυαδικού δέντρου με n καταχωρήσεις → $O(n^2)$
- Εκτύπωση με αλφαβητική σειρά → O(n)

Συνολικά: $O(n^2)+O(n) = O(n^2)$

Ψευδοκώδικας για Παράδειγμα 3

διαδικασία Εισαγωγή (Δέντρο, ΝέαΤιμή)
αν (δείκτης ρίζα Δέντρου = NIL)
(όρισε το δείκτη ρίζας **ώστε να δείχνει σε ένα νέο φύλλο που περιέχει τη ΝέαΤιμή**)
αλλιώς (εκτέλεσε το τμήμα των εντολών παρακάτω που σχετίζεται με την κατάλληλη περίπτωση)



περίπτωση 1: ΝέαΤιμή = τιμή ριζικού κόμβου
(Μην κάνεις τίποτα)

περίπτωση 2: ΝέαΤιμή < τιμή ριζικού κόμβου
(**αν** (αριστερός θυγατρικός δείκτης του ριζικού κόμβου = NIL)
τότε (όρισε το δείκτη αυτόν **ώστε να δείχνει σε ένα νέο κόμβο φύλλο που περιέχει τη ΝέαΤιμή**)
αλλιώς (εφάρμοσε τη διαδικασία Εισαγωγή για την εισαγωγή της ΝέαςΤιμής στο υποδέντρο που προσδιορίζεται από τον αριστερό θυγατρικό δείκτη)

περίπτωση 3:ΝέαΤιμή > τιμή ριζικού κόμβου
(**αν** (δεξιός θυγατρικός δείκτης του ριζικού κόμβου = NIL)
τότε (όρισε το δείκτη αυτόν **ώστε να δείχνει σε ένα νέο κόμβο φύλλο που περιέχει τη ΝέαΤιμή**)
αλλιώς (εφάρμοσε τη διαδικασία Εισαγωγή για την εισαγωγή της ΝέαςΤιμής στο υποδέντρο που προσδιορίζεται από τον δεξιό θυγατρικό δείκτη)
)
) **τέλος αν**

Για όποιον/α ενδιαφέρεται να το ψάξει περισσότερο

- Τι είναι τα B-δέντρα;
- Πώς υλοποιείται ο heapsort και τι πολυπλοκότητα έχει;
- Απαντήστε (όσο μπορείτε) τις ερωτήσεις που υπάρχουν στο PDF στο Link 2 παρακάτω Εξηγείστε σύντομα. Μπορείτε να αναζητήσετε στο Google ό,τι δεν γνωρίζετε.

Link 1: https://en.wikipedia.org/wiki/List_of_data_structures
Link 2: https://www.cs.cornell.edu/courses/cs2110/2014sp/L09-Lists/data_structures.pdf

Τέλος Κεφαλαίου 8

- Μιλήσαμε για δομές δεδομένων και πώς χρησιμοποιούνται στο σχεδιασμό αποδοτικών αλγορίθμων. προγραμματισμού.

- Στο επόμενο κεφάλαιο θα ασχοληθούμε με βάσεις δεδομένων, το σχεσιακό μοντέλο και τη γλώσσα SQL.