

Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής  
Εισαγωγή στην Επιστήμη των Υπολογιστών  
2023-24

# Αλγόριθμοι και Δομές Δεδομένων (II)

(γράφοι και δένδρα)

<http://mixstef.github.io/courses/csintro/>

Μ.Στεφανιδάκης



# Αφηρημένες Δομές Δεδομένων

- **Abstract Data Types (ADTs)**
  - Αφηρημένα μοντέλα δομών δεδομένων
  - Χωρίς τις λεπτομέρειες υλοποίησης
  - Προσδιορίζονται μόνο **από τις λειτουργίες που εφαρμόζονται σε αυτά**
- **Στη συνέχεια**
  - μια γλώσσα προγραμματισμού χρησιμοποιώντας συγκεκριμένες δομές
    - **όπως οι πίνακες ή οι διασυνδεδεμένες λίστες**
  - προσφέρει **υλοποιήσεις** των αφηρημένων δομών δεδομένων

# Στοιίβα (Stack)

- Μια βοηθητική αφηρημένη δομή δεδομένων
  - Ακολουθία δεδομένων
  - Με τη γνωστή λειτουργία **LIFO (Last-In-First-Out)**: θα πάρουμε πρώτο ό,τι βάλουμε στη στοιίβα τελευταίο
- **Λειτουργίες**
  - Ώθηση (push)
    - εισαγωγή στοιχείου στην κορυφή
  - Απώθηση (pop)
    - εξαγωγή στοιχείου από την κορυφή
  - push και pop από την ίδια άκρη!

# Υλοποίηση στοίβας

- Με τη χρήση πίνακα (array)
- Αποδοτικό σχήμα όταν
  - Η ώθηση και η απώθηση γίνεται στο τέλος του πίνακα
  - Η πολυπλοκότητα είναι  $O(1)$  και στις δύο περιπτώσεις!

# Ουρά (Queue)

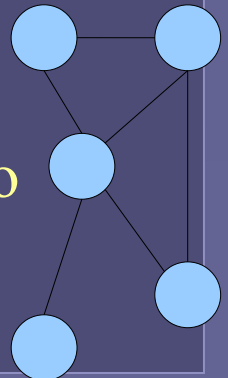
- Μια άλλη βοηθητική αφηρημένη δομή δεδομένων
  - Πρόκειται επίσης για ακολουθία δεδομένων
  - Με λειτουργία **FIFO (First-In-First-Out)**: θα πάρουμε πρώτο ό,τι βάλαμε στην ουρά πρώτο
- **Λειτουργίες**
  - Εισαγωγή (enqueue)
    - εισαγωγή στοιχείου στη μία άκρη
  - Εξαγωγή (dequeue)
    - εξαγωγή στοιχείου από την άλλη άκρη
  - **enqueue** και **dequeue** από διαφορετικές άκρες!

# Υλοποίηση ουράς

- Η “φυσική” υλοποίηση: με διπλά διασυνδεδεμένη λίστα
  - Ξέρουμε και τις δύο άκρες και μπορούμε να διασχίσουμε τη λίστα και προς τις δύο κατευθύνσεις
  - Εισαγωγή και εξαγωγή με  $O(1)$
- Πρακτικά όμως
  - Σε πολλά συστήματα η ουρά υλοποιείται ως “κυκλικός” πίνακας
    - πεπερασμένο μέγεθος, η άκρη εξαγωγής “κυνηγά” την άκρη εισαγωγής
    - και οι δύο άκρες, στο τέλος του πίνακα επιστρέφουν στην αρχή (wrap-around)

# Γράφοι (Graphs)

- Ένα από τα βασικότερα “αλγοριθμικά εργαλεία”
  - Πάρα πολλά προβλήματα ανάγονται σε γράφους και στη συνέχεια επιλύονται με αλγορίθμους γράφων!
- Γράφος
  - Ένα σύνολο **κορυφών** (κόμβων - nodes) που διασυνδέονται μέσω **ακμών** (edges).
  - Οι ακμές μπορούν να έχουν κατεύθυνση ή όχι
    - Προσανατολισμένοι και μη γράφοι (directed & undirected graphs)
  - Οι ακμές μπορούν να έχουν βάρη ή όχι
    - Αναπαράσταση κόστους (ανάλογα με το επιλυόμενο πρόβλημα)



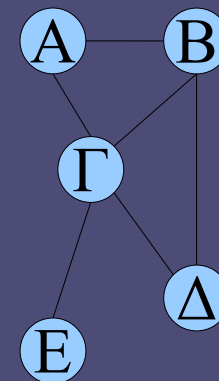
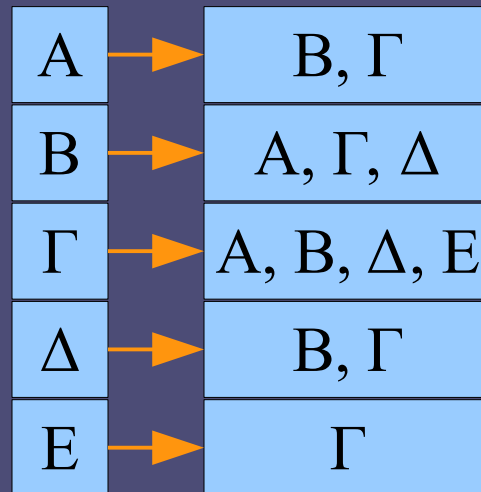
# Γράφοι (Graphs)

- **Έννοιες γράφων**
  - Οι κορυφές που ενώνει μια ακμή ονομάζονται **γειτονικές** (adjacent)
  - **Διαδρομή** (path) είναι μία ακολουθία κορυφών, η μία γειτονική με την επόμενη
    - Χωρίς να επισκεφτούμε ξανά κάποια από τις κορυφές αυτές
  - Αν η διαδρομή τελειώνει στην αρχική κορυφή, πρόκειται για **κύκλο** (cycle)
    - Τυπικά, τουλάχιστον τρεις κορυφές
  - Ένας γράφος είναι **συνδεδεμένος** αν μπορούμε από κάθε κορυφή να μεταβούμε σε κάθε άλλη



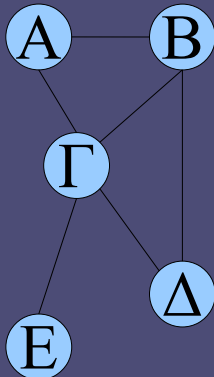
# Υλοποίηση γράφων (1)

- **Λίστα γειτνίασης (adjacency list)**
  - Για κάθε κορυφή του γράφου
  - Διατηρούμε μια λίστα με όλες τις γειτονικές κορυφές
  - Ενδεχομένως και το βάρος της ακμής (αν υπάρχει)



# Υλοποίηση γράφων (2)

- Πίνακας γειτνίασης (adjacency matrix)
  - $N \times N$  πίνακας, πληροφορία για κάθε ζεύγος κορυφών
  - Αν ο γράφος είναι μη προσανατολισμένος, ο πίνακας είναι συμμετρικός



	A	B	Γ	Δ	Ε
A	0	1	1	0	0
B	1	0	1	1	0
Γ	1	1	0	1	1
Δ	0	1	1	0	0
Ε	0	0	1	0	0

# Διάσχιση Γράφου (Graph Traversal)

- Πολύ συχνά η επίλυση ενός προβλήματος απαιτεί την εύρεση μιας «σωστής» διαδρομής μεταξύ δύο κορυφών
  - «Σωστή»: με τα κριτήρια του εκάστοτε επιλυόμενου προβλήματος
  - Αναζητώντας τη διαδρομή αυτή πρέπει αλγοριθμικά να διασχίσουμε τον γράφο
  - Ξεκινώντας από μία κορυφή
  - Επισκεπτόμενοι διάφορες άλλες κορυφές (ενδεχομένως όλες)
- Αν ο γράφος έχει κύκλους
  - Πρέπει να εξασφαλιστεί ότι δεν θα επισκεφθούμε ξανά τον ίδιο κόμβο

# Διάσχιση με προτεραιότητα βάθους

- **Depth-First Search (DFS)**

- Η ιδέα της αναδρομικής επίσκεψης των γειτόνων:

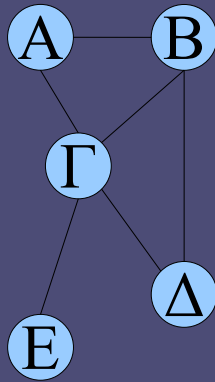
```
visit(node) {  
    for each neighbor of node {  
        if not_visited(neighbor)  
            visit(neighbor)  
    }  
}
```

- **Μπορεί να υλοποιηθεί επαναληπτικά με τη βοήθεια μιας στοίβας**

- Όταν επισκεφθούμε μια κορυφή, αν δεν την έχουμε ήδη επισκεφτεί, ωθούμε στη στοίβα όλους τους γείτονές της
- Από τη στοίβα παίρνουμε τον στόχο της επόμενης μας επίσκεψης, έως ότου η στοίβα αδειάσει

# Παράδειγμα διάσχισης DFS

- Για τον γράφο του σχήματος
  - Και την εικονιζόμενη λίστα γειτνίασης
- Η σειρά επίσκεψης ξεκινώντας από τον A είναι  
**A Γ E Δ B**



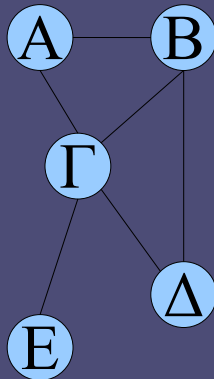
A	→	B, Γ
B	→	A, Γ, Δ
Γ	→	A, B, Δ, E
Δ	→	B, Γ
E	→	Γ

# Διάσχιση με προτεραιότητα εύρους

- Τι θα συμβεί αν αντικαταστήσουμε τη στοίβα του προηγούμενου αλγορίθμου με μια ουρά;
  - Breadth-First Search (BFS)
- Πρακτικά:
  - Επισκεπτόμαστε πρώτα τις κορυφές που βρίσκονται κοντύτερα στην αρχή
    - σε ζώνες (επίπεδα) απόστασης από εκεί που ξεκινήσαμε
  - Η αναζήτηση BFS θα βρει λύσεις με **συντομότερες** διαδρομές (shortest paths)

# Παράδειγμα διάσχισης BFS

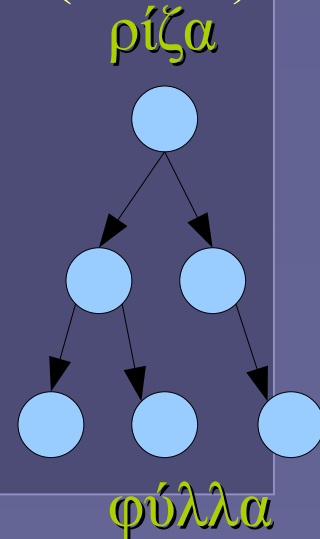
- Για τον γράφο του σχήματος
  - Και την εικονιζόμενη λίστα γειτνίασης
- Η σειρά επίσκεψης ξεκινώντας από τον A είναι  
**A B Γ Δ E**



A	→	B, Γ
B	→	A, Γ, Δ
Γ	→	A, B, Δ, E
Δ	→	B, Γ
E	→	Γ

# Δένδρα (Trees)

- **Υποκατηγορία γράφων**
  - Συνδεδεμένοι και χωρίς κύκλους γράφοι
  - Με πολλές αλγοριθμικές εφαρμογές, ιδίως στην αναζήτηση
- **Έννοιες δένδρων:**
  - Διασυνδεδεμένοι **κόμβοι** (nodes), με προγόνους και απογόνους
    - στην κορυφή η **ρίζα** (root) και στο τέλος τα **φύλλα** (leaves)
    - **siblings**: κόμβοι με τον ίδιο πατέρα
  - **Επίπεδο** κόμβου: η απόστασή του από τη ρίζα
  - **Ύψος** δένδρου: το μήκος (σε κόμβους) της μέγιστης διαδρομής ρίζα-φύλλο





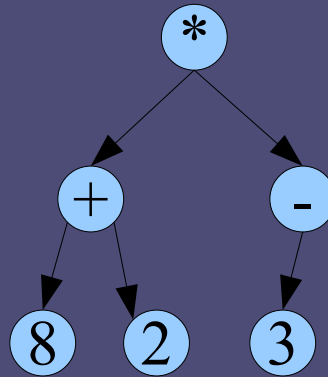
# Υλοποίηση δένδρων

- **Δεν υπάρχει ένας και μοναδικός τρόπος**
  - Ανάλογα με το είδος του δένδρου
  - Υπάρχουν πολλοί τύποι δένδρων
  - Ανάλογα με το πρόβλημα που καλούνται να λύσουν
- **Σε γενικές γραμμές**
  - Σύνολο διασυνδεδεμένων πινάκων (arrays)
    - αποθηκεύουν τα δεδομένα κάθε κόμβου, τις διασυνδέσεις με τα παιδιά του κόμβου, και όποιες άλλες πληροφορίες διαχείρισης του δένδρου

# Διάσχιση δένδρων

- Τα δένδρα είναι υποκατηγορία γράφων
  - Συνεπώς μπορούμε να εφαρμόσουμε οποιαδήποτε τεχνική διάσχισης (π.χ. DFS ή BFS)
  - Σε κάθε κόμβο μπορούμε να εφαρμόσουμε κάποια μορφή επεξεργασίας
- Επεξεργασία κατά τη διάσχιση DFS
  - Preorder: πριν προχωρήσουμε στα παιδιά του
  - Postorder: αφού επιστρέψουμε από την επεξεργασία των παιδιών
  - Ειδικά για δυαδικά δένδρα (όχι πάνω από δύο παιδιά): inorder επεξεργασία, αριστερό παιδί - κόμβος - δεξί παιδί

# Παράδειγμα επεξεργασίας σε διάσχιση DFS



Preorder: \* + 8 2 - 3

Postorder: 8 2 + 3 - \*

Inorder: 8 + 2 \* 3 -

- Ποια σειρά επεξεργασίας αποδίδει το αριθμητικό νόημα του δένδρου;

# Η δυαδική αναζήτηση (ξανά)

- Έχουμε ήδη δει την ισχύ της δυαδικής αναζήτησης
  - Η ισχύς του  $\log_2 n$  για γρήγορη εύρεση σε μεγάλο αριθμό δεδομένων
- Όμως
  - Χρειαζόμαστε ταξινομημένους πίνακες
    - Πόσο εύκολο αν τα δεδομένα αλλάζουν συνεχώς;
  - Και οι διασυνδεδεμένες λίστες δεν αποτελούν λύση
    - Απώλεια του  $O(1)$  για την εύρεση (ή μη) ενός στοιχείου
  - Τι άλλο μπορούμε να κάνουμε;

# Πρώτη λύση: αποφυγή αναζήτησης!

- **Μέθοδος κατακερματισμού (hashing)**
  - Για εύρεση κλειδιού (membership test) ή αντιστοίχιση κλειδιού-τιμής (mapping) χωρίς αναζήτηση
- **Πώς γίνεται;**
  - Κάθε κλειδί μετατρέπεται σε έναν αριθμό μέσω συνάρτησης κατακερματισμού (hash function)
    - ο αριθμός αυτός (ή κάποια bits αυτού) χρησιμοποιούνται ως δείκτης  $i$  σε έναν πίνακα
  - Διαλέγουμε συναρτήσεις που κατανέμουν ομοιόμορφα τα κλειδιά στις θέσεις του πίνακα
    - Εναλλακτικές θέσεις σε περίπτωση σύγκρουσης (collision)
  - Πολυπλοκότητα (σχεδόν)  $O(1)$ !

# Δεύτερη λύση: δένδρα δυαδικής αναζήτησης

- **Δυαδικά δένδρα με τοποθέτηση των κλειδιών σε κάθε κόμβο**
  - Με τη **σημαντική** ιδιότητα: όλα τα κλειδιά στο αριστερό υποδένδρο είναι **μικρότερα** (ή ίσα) από το κλειδί του κόμβου
  - Και όλα τα κλειδιά στο δεξί υποδένδρο είναι **μεγαλύτερα** από το κλειδί του κόμβου
- **Αναζήτηση**
  - Σε κάθε βήμα, συγκρίνουμε την επιθυμητή τιμή με το κλειδί του τρέχοντος κόμβου και στη συνέχεια προχωράμε ανάλογα στο αριστερό ή στο δεξί υποδένδρο
    - από τη ρίζα προς τα φύλλα σε λογαριθμικό χρόνο

# Δεύτερη λύση: δένδρα δυαδικής αναζήτησης

- **Εισαγωγή νέου στοιχείου**
  - Αναζητούμε το σημείο όπου θα έπρεπε να είναι το νέο στοιχείο
  - Και το εισάγουμε στην κατάλληλη θέση
  - **Προσοχή**: η μορφή του δένδρου εξαρτάται από τη σειρά εισαγωγής των στοιχείων
- **Ισορροπία (balance) δένδρου**
  - Το απλό δυαδικό δένδρο μετά από εισαγωγή νέων στοιχείων μπορεί να πάψει να έχει ισορροπία (να είναι ομοιόμορφα επεκταμένο)
    - **Η απόδοση της αναζήτησης μειώνεται**
  - Υπάρχουν εξελιγμένες μορφές δένδρων που φροντίζουν για τη διατήρηση της ισορροπίας τους κατά την εισαγωγή ή διαγραφή στοιχείων