Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής

Εισαγωγή στην Επιστήμη των Υπολογιστών

2024-25

Αλγόριθμοι και Δομές Δεδομένων (ΙΙ)

(γράφοι και δένδρα)

https://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

Στοίβα (Stack)

- Μια βοηθητική αφηρημένη δομή δεδομένων
  - Είναι μια ακολουθία δεδομένων
  - Με τη γνωστή λειτουργία LIFO (Last-In-First-Out): θα πάρουμε πρώτο ό,τι βάλαμε στη στοίβα τελευταίο
- Λειτουργίες
  - Εφαρμόζονται πάντα στη μία άκρη της στοίβας (κορυφή)
    - Top of Stack (ToS)
  - Ώθηση (push)
    - Εισαγωγή στοιχείου στην κορυφή
  - Απώθηση (pop)
    - Εξαγωγή στοιχείου από την κορυφή

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)

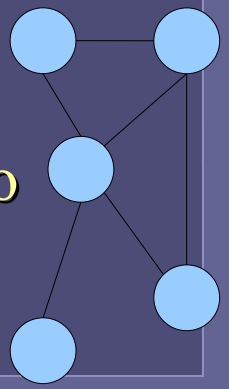
- Μια άλλη βοηθητική αφηρημένη δομή δεδομένων
  - Πρόκειται επίσης για ακολουθία δεδομένων
  - Με λειτουργία FIFO (First-In-First-Out): θα πάρουμε πρώτο ό,τι βάλαμε στην ουρά πρώτο
- Λειτουργίες
  - Γίνονται και στις δύο άκρες της ουράς
  - Εισαγωγή (enqueue)
    - Εισαγωγή στοιχείου στη μία άκρη
  -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

Υλοποίηση ουράς

- Η "φυσική" υλοποίηση: με διπλά διασυνδεδεμένη λίστα
  - Ξέροντας και τις δύο άκρες 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)

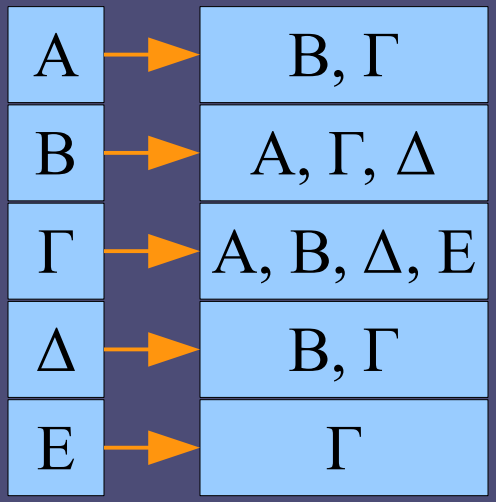


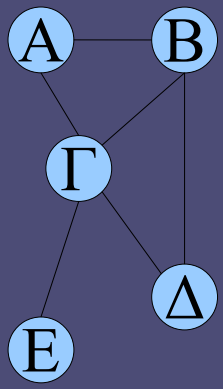
Γράφοι (Graphs)

- 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)

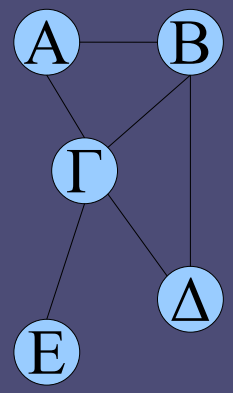
- Λίστα γειτνίασης (adjacency list)
  - Για κάθε κορυφή του γράφου
  - Διατηρούμε μια λίστα με όλες τις γειτονικές κορυφές
  - Ενδεχομένως και το βάρος της ακμής (αν υπάρχει)





Υλοποίηση γράφων (2)

- Πίνακας γειτνίασης (adjacency matrix)
  - NxN πίνακας, πληροφορία για κάθε ζεύγος κορυφών
  - Αν ο γράφος είναι μη προσανατολισμένος, ο πίνακας είναι συμμετρικός
  - Σε κανονικούς γράφους (οι κόμβοι δεν συνδέονται με τον εαυτό τους) η διαγώνιος του πίνακα είναι μηδενική



	Α	Β	Γ	Δ	Ε
Α	0	1	1	0	0
Β	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

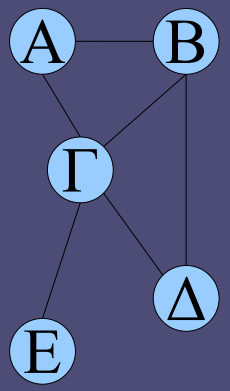
- Depth-First Search (DFS)
  - Η ιδέα της αναδρομικής επίσκεψης των γειτόνων:

```
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 επόμενής μας επίσκεψης, έως ότου η στοίβα αδειάσει
  - Στη στοίβα αρχικά μπαίνει ο πρώτος κόμβος της επίσκεψης

Παράδειγμα διάσχισης DFS

- Για τον γράφο του σχήματος
  - Και την εικονιζόμενη λίστα γειτνίασης
- Η σειρά επίσκεψης ξεκινώντας από τον Α είναι
  Α Γ Ε Δ Β



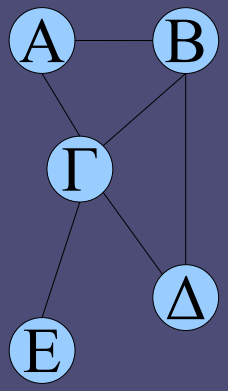
Α	Β, Γ
Β	Α, Γ, Δ
Γ	Α, Β, Δ, Ε
Δ	Β, Γ
Ε	Γ

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

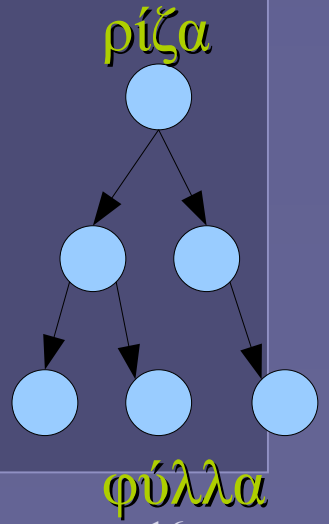
- Για τον γράφο του σχήματος
  - Και την εικονιζόμενη λίστα γειτνίασης
- Η σειρά επίσκεψης ξεκινώντας από τον Α είναι
  Α Β Γ Δ Ε



Α	Β, Γ
Β	Α, Γ, Δ
Γ	Α, Β, Δ, Ε
Δ	Β, Γ
Ε	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

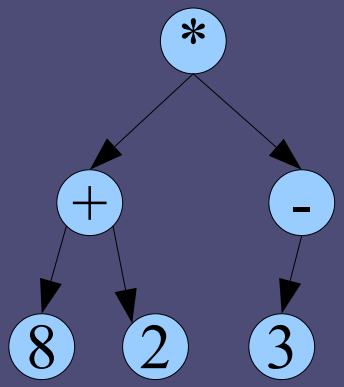
- Δεν υπάρχει ένας και μοναδικός τρόπος
  - Ανάλογα με το είδος του δένδρου
  - Υπάρχουν πολλοί τύποι δένδρων
  - Ανάλογα με το πρόβλημα που καλούνται να λύσουν
- Σε γενικές γραμμές
  - Σύνολο διασυνδεδεμένων πινάκων (arrays)
    - Αποθηκεύουν τα δεδομένα κάθε κόμβου, τις διασυνδέσεις με τα παιδιά του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

- Έστω ότι η διάσχιση DFS επισκέπτεται τους κόμβους με τη σειρά * + 8 2 - 3



- Με ποια σειρά θα επεξεργαστούμε τον κάθε κόμβο;

```
Preorder:    *  +  8  2  -  3

Postorder:   8  2  +  3  -  *

Inorder:     8  +  2  *  3  -
```

- Ποια σειρά επεξεργασίας αποδίδει το αριθμητικό νόημα του δένδρου;
- Θεωρώντας ότι κάθε κόμβος επιστρέφει «προς τα πάνω» μια αριθμητική τιμή

Η δυαδική αναζήτηση (ξανά)

- Στόχοι
  - Εύρεση μέσα σε ένα σύνολο τιμών (membership test)
  - Εύρεση τιμής που αντιστοιχεί σε κλειδί (mapping)
- 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)
  - 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 αναζήτησης 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
  - 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