ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΥΠΟΛΟΓΙΣΤΩΝ

Φροντιστηριακή Διάλεξη | Άννα Κεφάλα

ΑΦΑΙΡΕΤΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Lists, Queues, Stacks, Graphs

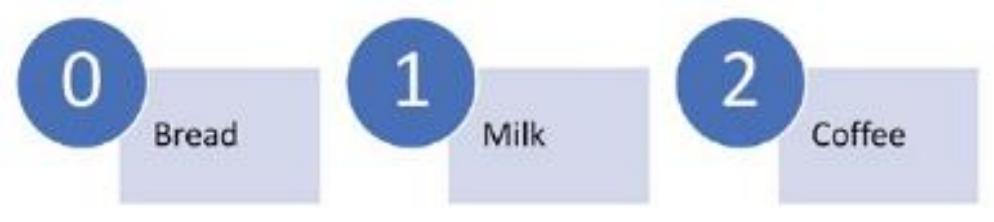
ΑΦΑΙΡΕΤΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

- **Αφαιρετικές Δομές δεδομένων** (abstract data structures): αφαιρετικοί τύποι (containers) για την οργάνωση των δεδομένων, οι ιδιότητες (properties) τους –δεδομένα και λειτουργίες – προσδιορίζονται ανεξαρτήτως του τρόπου υλοποίησης.
- **Δομές δεδομένων**: η υλοποίηση σύνθετων δεδομένων σε κάποιο αφαιρετικό τύπο δεδομένων.
- **Βασικές δομές δεδομένων**
  - **arrays** (πίνακες): ομογενείς (δεδομένα ίδιου τύπου) ή ετερογενείς
  - **συνδεδεμένες lists** (λίστες): stacks (στοίβες) και queues (ουρές)
  - **trees** (δέντρα)

LIST | ΜΟΝΟΔΙΑΣΤΑΤΗ

- **List** (μονο-διάστατη): συλλογή δεδομένων σε σειριακή διάταξη, τα στοιχεία της έχουν προηγούμενο και επόμενο στοιχείο (εκτός από το πρώτο –head και το τελευταίο – tail)
- Κάθε στοιχείο ταυτοποιείται με βάση το δείκτη (index) θέσης του στη λίστα

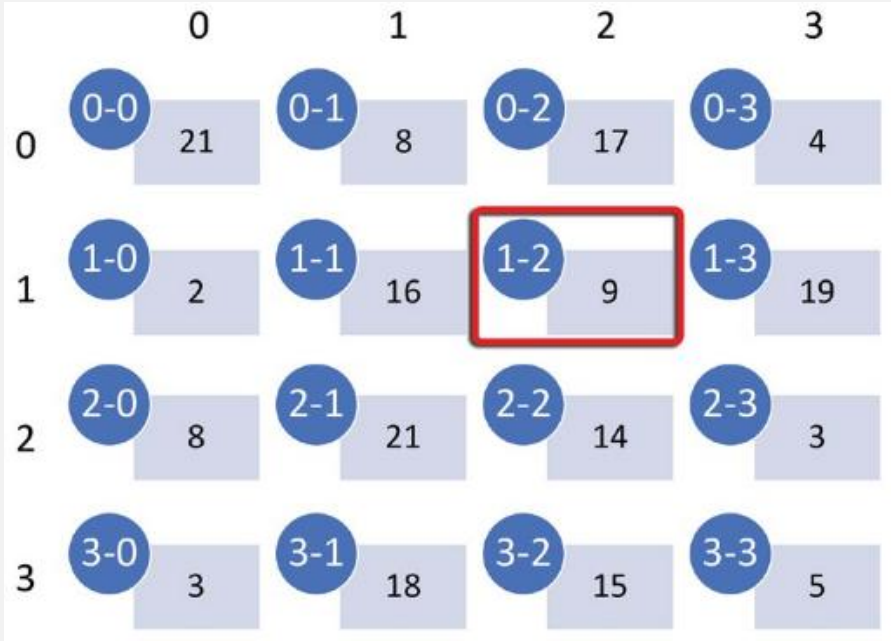
```
shoppingList = ['bread', 'milk', 'coffee’]
print (shoppingList[1]) → milk
```



LISTS | 2–DIMENSION

- **2D List** (δι-διάστατη): συλλογή δεδομένων σε μορφή **grid** (σαν πίνακας με γραμμές και στήλες), κάθε γραμμή και κάθε στήλη ξεκινάει με index 0
- Κάθε στοιχείο ταυτοποιείται με βάση το κελί στο οποίο βρίσκεται στον πίνακα δείκτης γραμμής, δείκτης στήλης

```
scoreSheet = [
    [ 21, 8, 17, 4 ],
    [ 2, 16, 9, 19 ],
    [ 8, 21, 14, 3 ],
    [ 3, 18, 15, 5 ]
]
```



print (scoreSheet[1][2])

→ 9

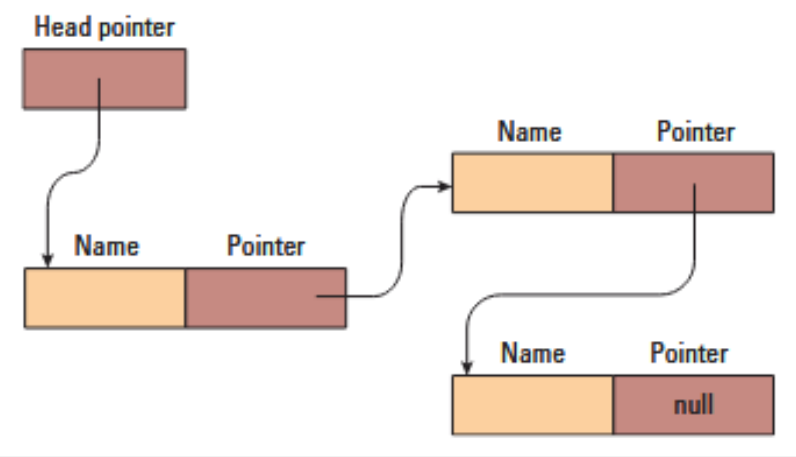
scoresheet [0][3] = 6

EXAMPLE CODE

- **2dList.py**
  - Δημιουργία δι-διάστατης λίστας
  - Εμφάνιση περιεχομένων δι-διάστατης λίστας σε μορφή πίνακα
  - Παράδειγμα χρήσης

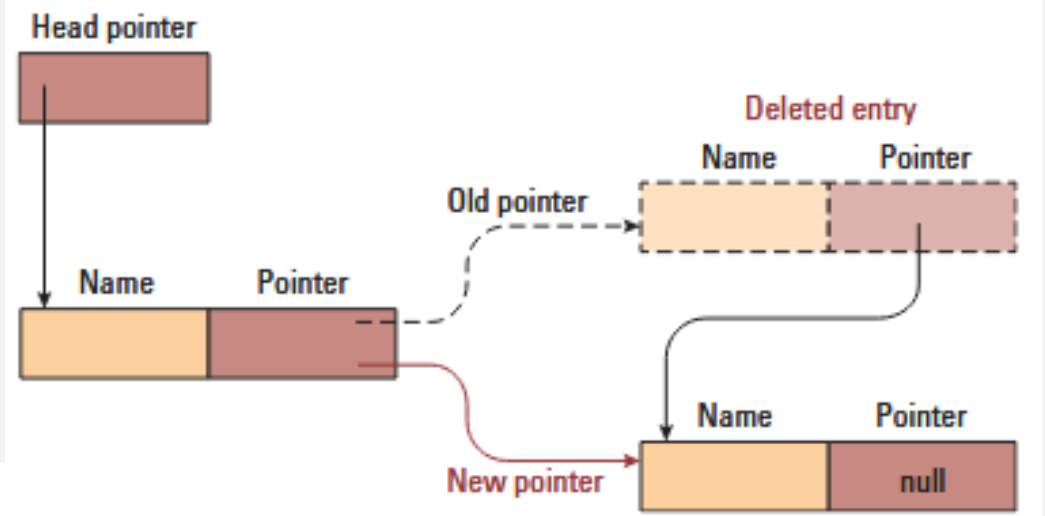
ΣΥΝΔΕΔΕΜΕΝΕΣ ΛΙΣΤΕΣ

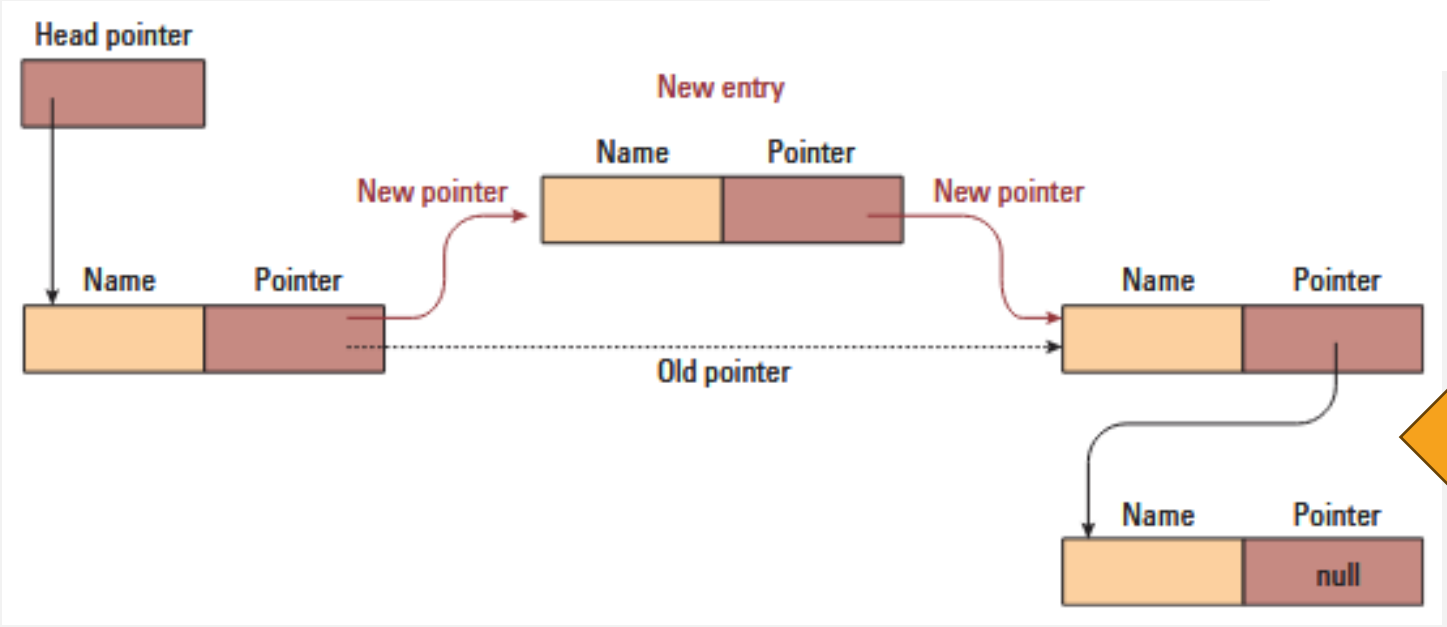
- **Linked List**: λίστα στην οποία κάθε στοιχείο δείχνει στο επόμενό του με δείκτη (*pointer*)
  - Δείκτης αρχής (*head*): δείχνει στην αρχή της λίστας
  - Δείκτης τέλους (*none*): ειδικό σχήμα bit (συνήθως το 0) στη θέση του δείκτη του τελευταίου στοιχείου της λίστας που σημαίνει ότι δεν υπάρχουν άλλες καταχωρίσεις στη λίστα
- Κινούμαι στη λίστα από ένα στοιχείο σε άλλο, ακολουθώντας τους δείκτες
- Χρήσιμες όταν έχω δυναμικές λίστες που αυξομειώνονται



ΣΥΝΔΕΔΕΜΕΝΕΣ ΛΙΣΤΕΣ | ΔΙΑΓΡΑΦΗ & ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΟΥ

- **Διαγραφή** στοιχείου: αλλαγή δείκτη προηγούμενου στοιχείου ώστε να δείχνει στο επόμενο αυτού που θα διαγραφεί





- **Εισαγωγή** στοιχείου: δείκτης προηγούμενου στοιχείου → νέο στοιχείο, δείκτης νέου στοιχείου → επόμενο στοιχείο

ΣΤΟΙΒΑ | STACK

- **Stack** (στοίβα): συλλογή δεδομένων σε σειριακή διάταξη, ακολουθεί την αρχή **LIFO** (Last-In-First-Out), μία στοίβα στοιχείων, το πρώτο που μπορείς να αφαιρέσεις είναι το τελευταίο που προστέθηκε
- Βασικές **λειτουργίες**:
  - **Push**: προσθέτει ένα νέο στοιχείο στη στοίβα
  - **Pop**: αφαιρεί το πιο πάνω στοιχείο της στοίβας (και το επιστρέφει)
  - **Peek**: βρίσκει και επιστρέφει το πιο πάνω (τελευταίο) στοιχείο της στοίβας
  - **isEmpty**: ελέγχει αν η στοίβα είναι κενή
  - **Size**: βρίσκει τον αριθμό στοιχείων της στοίβας
- Υλοποίηση (*python*): με array, list ή linked-list

EXAMPLE CODE

- **stack.py**
  - Δημιουργία στοίβας
  - Βασικές λειτουργίες

ΟΥΡΑ | QUEUE

- **Queue** (ουρά): συλλογή δεδομένων σε σειριακή διάταξη, ακολουθεί την αρχή **FIFO** (First-In-First-Out), μία ουρά στοιχείων, το πρώτο που μπορείς να αφαιρέσεις είναι το πρώτο που προστέθηκε.
- Βασικές **λειτουργίες**:
  - **Enqueue**: προσθέτει ένα νέο στοιχείο στην ουρά
  - **Dequeue**: αφαιρεί το πρώτο στοιχείο της ουράς (και το επιστρέφει)
  - **Peek**: βρίσκει και επιστρέφει πρώτο στοιχείο της ουράς
  - **isEmpty**: ελέγχει αν η ουρά είναι κενή
  - **Size**: βρίσκει τον αριθμό στοιχείων της ουράς
- Υλοποίηση (*python*): με array, list ή linked-list

EXAMPLE CODE

- **queue.py**
  - Δημιουργία ουράς
  - Βασικές λειτουργίες

COLLECTIONS: **DEQUE**

Collection: container τύπων δεδομένων, εξειδικευμένων – εναλλακτικών των γενικών built-in containers της python

deque: container τύπου list (deque: double-ended queue)

- γενίκευση στοίβας και ουράς, προσφέρει γρήγορη προσθήκη ή αφαίρεση στοιχείου σε οποιοδήποτε άκρο της λίστας (ουράς)
- πιο ευέλικτο container, χρήσιμο για πραγματικές εφαρμογές όπως χρονοπρογραμματισμός εργασιών, επεξεργασία δεδομένων σε πραγματικό χρόνο, μετάδοση δεδομένων στο δίκτυο

DEQUE | ΚΥΡΙΕΣ ΥΠΟΣΤΗΡΙΖΟΜΕΝΕΣ ΜΕΘΟΔΟΙ

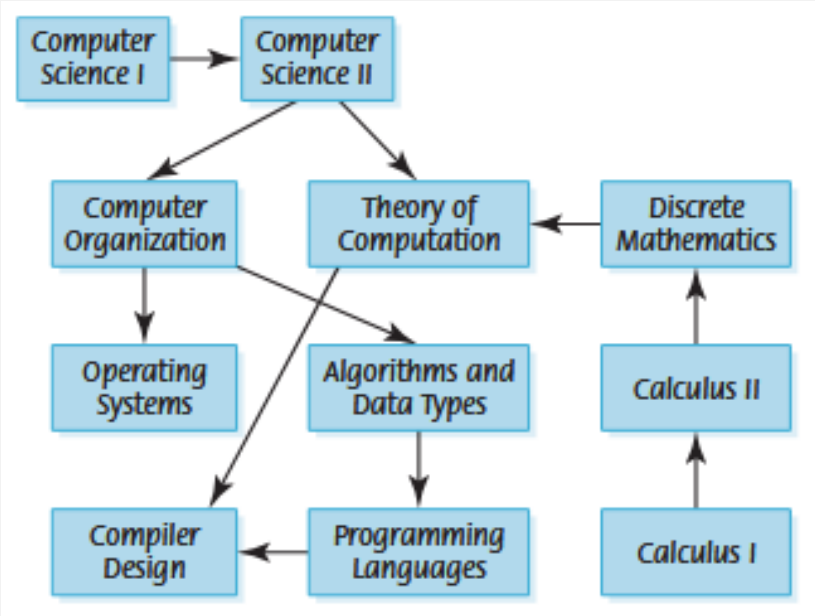
- **append(x)**: προσθέτει το x στο δεξί άκρο του deque, **appendleft(x)**: προσθέτει το x στο αριστερό άκρο του deque
- **pop()**: αφαιρεί (και το επιστρέφει) ένα στοιχείο από το δεξί άκρο του deque, **popleft()**: αφαιρεί (και το επιστρέφει) ένα στοιχείο από το αριστερό άκρο του deque
- **extend(iterable)**: προσθέτει όλα τα στοιχεία του iterable στο δεξί άκρο, **extendleft(iterable)**: προσθέτει όλα τα στοιχεία του iterable στο αριστερό άκρο
- **len()**: επιστρέφει τον αριθμό στοιχείων (μέγεθος του deque)
- **rotate(n)**: ολίσθηση των περιεχομένων κατά n προς τα δεξιά, αν n αρνητικός, ολίσθηση προς τα αριστερά
- **reverse()**: αντιστρέφει τα στοιχεία του deque
- **remove(x)**: αφαιρεί το x στην πρώτη θέση που θα το βρει, **count(x)**: μετράει πόσες φορές υπάρχει το x στο deque, **index(x)**: επιστρέφει το δείκτη θέσης του x την $1^{η}$ φορά που το βρίσκει στο deque
- **clear()**: αδειάζει το deque

EXAMPLE CODE

- **dequeue.py**
  - Δημιουργία ουράς
  - Βασικές λειτουργίες

ΓΡΑΦΟΣ | GRAPH

- **graph**: δομή δεδομένων που αποτελείται από ένα σύνολο κόμβων και ένα σύνολο ακμών που συνδέουν τους κόμβους μεταξύ τους
- **directed/undirected graph**: οι ακμές έχουν /δεν έχουν κατεύθυνση
- **vertex**: ένας κόμβος του γράφου
- **edge**: μία ακμή, σύνδεση μεταξύ δύο κόμβων του γράφου
- **adjacent** (γειτονικοί): κόμβοι που συνδέονται με μία ακμή



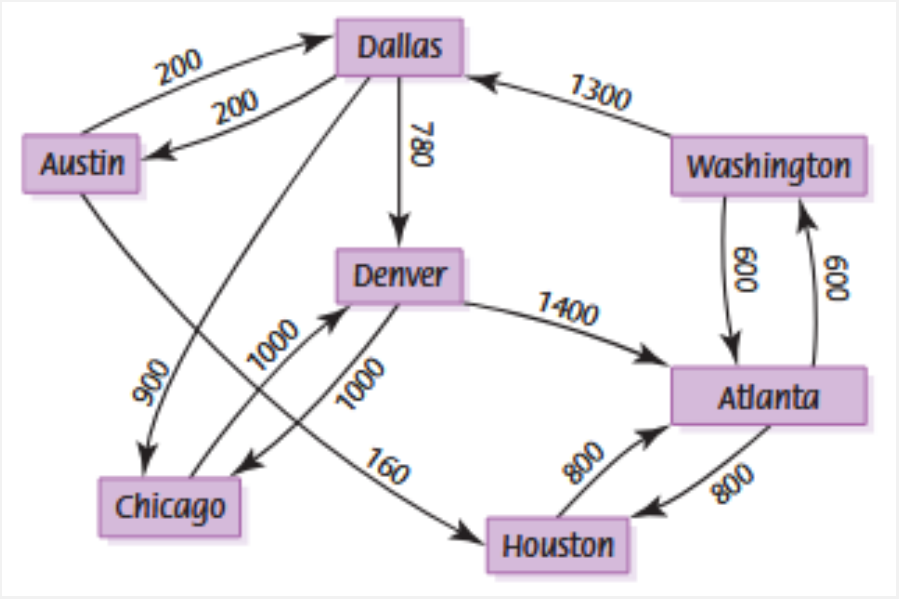
Παράδειγμα γράφου

Vertices (κόμβοι): μαθήματα

Edges (ακμές): προ-απαιτούμενα

ΓΡΑΦΟΣ | ΠΑΡΑΔΕΙΓΜΑ (1)

- Έστω ένας γράφος που περιγράφει τις αεροπορικές συνδέσεις μεταξύ πόλεων (πόλεις → κόμβοι, απευθείας αεροπορική σύνδεση → ακμή)



Ας σκεφτούμε τον γράφο σαν **πίνακα**:

- οι πόλεις (κόμβοι) του γράφου → γραμμές και στήλες
- απευθείας σύνδεση (ακμή) → μίλια απόστασης, αν δεν υπάρχει απευθείας σύνδεση → 0

	Atlanta	Austin	Chicago	Dallas	Denver	Houston	Washington
Atlanta	0	0	0	0	0	800	600
Austin	0	0	0	200	0	160	0
Chicago	0	0	0	0	1000	0	0
Dallas	0	200	900	0	780	0	0
Denver	1400	0	1000	0	0	0	0
Houston	800	0	0	0	0	0	0
Washington	600	0	0	1300	0	0	0

ΓΡΑΦΟΣ | ΠΑΡΑΔΕΙΓΜΑ (2)

- Υλοποίηση σε python του πίνακα: με 2D-list (λίστα με λίστες)

```
cities = {
  0 : "Atlanta",
  1 : "Austin",
  2 : "Chicago",
  3 : "Dallas",
  4 : "Denver",
  5 : "Houston",
  6 : "Washington"
}
```

Dictionary αντιστοίχισης των πόλεων με Α/Α γραμμής – στήλης

Αποστάσεις πόλεων με απευθείας πτήση ή 0 (χωρίς σύνδεση)

```
flights= [
    [0, 0, 0, 0, 0, 800, 600],
    [0, 0, 0, 200, 0, 160, 0],
    [0, 0, 0, 0, 1000, 0, 0],
    [0, 200, 900, 0, 780, 0, 0],
    [1400, 0 , 1000, 0, 0, 0, 0],
    [800, 0, 0, 0, 0, 0, 0],
    [600, 0, 0, 1300, 0, 0, 0]
]
```

	Atlanta	Austin	Chicago	Dallas	Denver	Houston	Washington
Atlanta	0	0	0	0	0	800	600
Austin	0	0	0	200	0	160	0
Chicago	0	0	0	0	1000	0	0
Dallas	0	200	900	0	780	0	0
Denver	1400	0	1000	0	0	0	0
Houston	800	0	0	0	0	0	0
Washington	600	0	0	1300	0	0	0

ΓΡΑΦΟΣ | ΠΑΡΑΔΕΙΓΜΑ (3)

- Υλοποίηση σε python του πίνακα: με 2D-list (λίστα με λίστες)

```
cities = {
  0 : "Atlanta",
  1 : "Austin",
  2 : "Chicago",
  3 : "Dallas",
  4 : "Denver",
  5 : "Houston",
  6 : "Washington"
}
```

```
flights= [
        [0, 0, 0, 0, 0, 800, 600],
        [0, 0, 0, 200, 0, 160, 0],
        [0, 0, 0, 0, 1000, 0, 0],
        [0, 200, 900, 0, 780, 0, 0],
        [1400, 0 , 1000, 0, 0, 0, 0],
        [800, 0, 0, 0, 0, 0, 0],
        [600, 0, 0, 1300, 0, 0, 0]
]
```

```
if flights[4][0] != '0':
   print (f"Οι πόλεις {cities[4]} και
   {cities[0]}  συνδέονται μεταξύ
   τους με απευθείας πτήση και η
   απόσταση είναι {flights[4][0]}
   μίλια")
```

Τί θα εμφανίσει ο κώδικας;

Οι πόλεις Denver και Atlanta συνδέονται μεταξύ τους με απευθείας πτήση και η απόσταση είναι 1400 μίλια

EXAMPLE CODE

- **graph.py**
  - Υλοποίηση γράφου με λίστα και λεξικό
  - Εμφάνιση στοιχείων του γράφου

ΑΛΓΟΡΙΘΜΟΙ ΑΝΑΖΗΤΗΣΗΣ ΓΙΑ ΤΗ ΔΙΑΣΧΙΣΗ ΕΝΟΣ ΓΡΑΦΟΥ

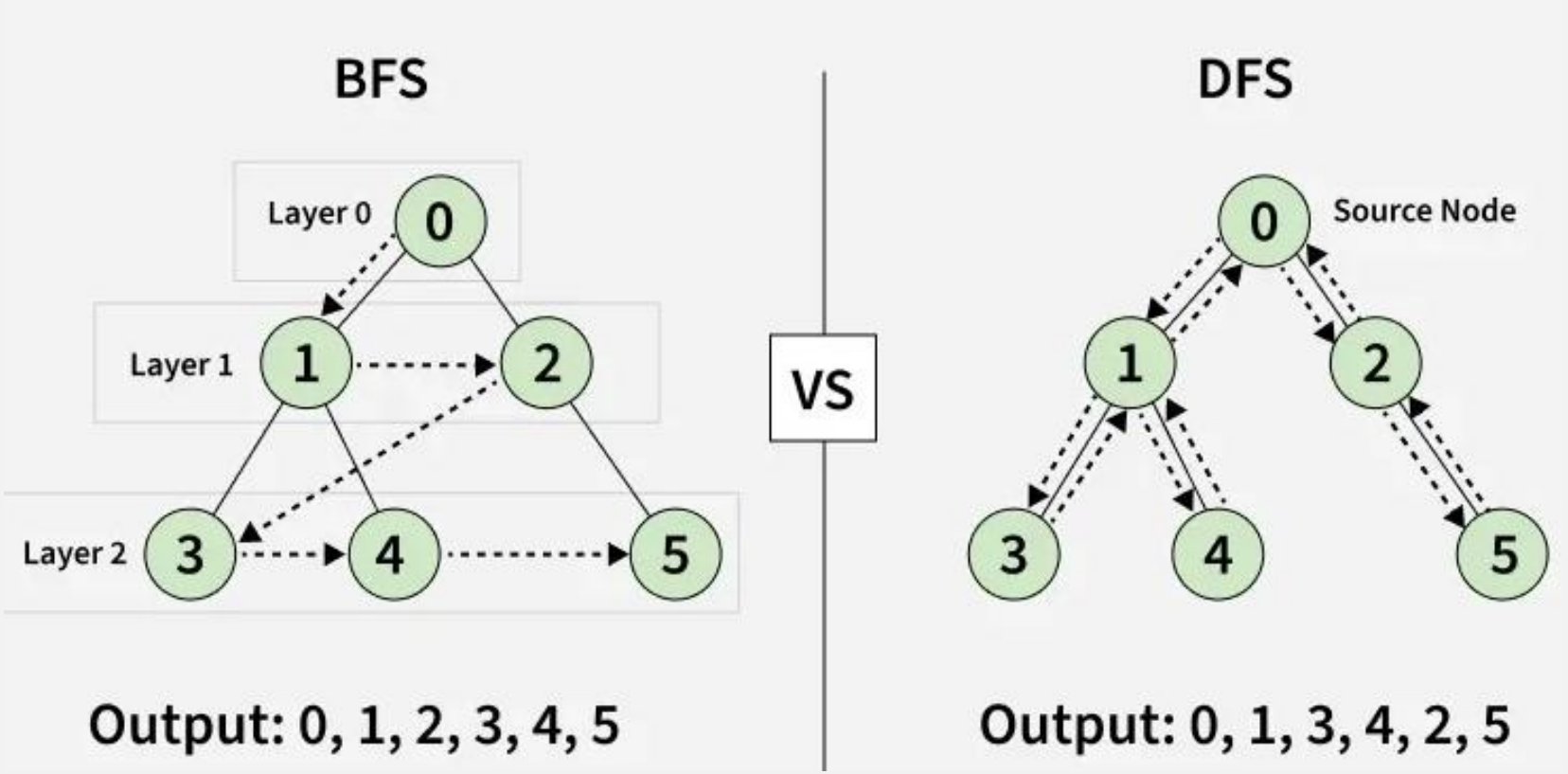
ΑΛΓΟΡΙΘΜΟΙ ΔΙΑΣΧΙΣΗΣ ΓΡΑΦΟΥ | DFS

- **Depth-First Search (DFS)**: εξερεύνηση/διάσχιση σε βάθος ενός γράφου, προτίμηση για έναν γείτονα μέχρι τέλος, πριν κάνει οπισθοδρόμηση για να βρει τον επόμενο γείτονα/κόμβο που δεν έχει επισκεφθεί
  - λογική **LIFO**, υλοποίηση με χρήση στοίβας (**stack**)
  - χρήσιμος για πλήρη εξερεύνηση, δεν εξασφαλίζει ελάχιστη διαδρομή σε grid
  - **αλγόριθμος**:
    1. Start with source node, push it on stack, mark it as visited
    2. Pop a node, visit it, push all unvisited neighbors on stack
    3. Repeat until stack is empty

ΑΛΓΟΡΙΘΜΟΙ ΔΙΑΣΧΙΣΗΣ ΓΡΑΦΟΥ | **BFS**

- **Breadth-First Search (BFS)**: εξερεύνηση/διάσχιση ενός γράφου κατά επίπεδα, ξεκινώντας από έναν αρχικό κόμβο επισκέπτεται όλους τους γειτονικούς του κόμβους, συνεχίζει με όποιον από τους γείτονες δεν έχει επισκεφθεί
  - λογική **FIFO**, υλοποίηση με χρήση ουράς (**queue**)
  - αποτελεσματικός για αναζήτηση μικρότερης διαδρομής σε γράφους χωρίς βάρη στις ακμές
  - αλγόριθμος:
    1. Start with source node, enqueue it, mark it as visited
    2. Dequeue a node, visit it, enqueue all unvisited neighbors and mark them as visited
    3. Repeat until queue is empty

BFS VS DFS

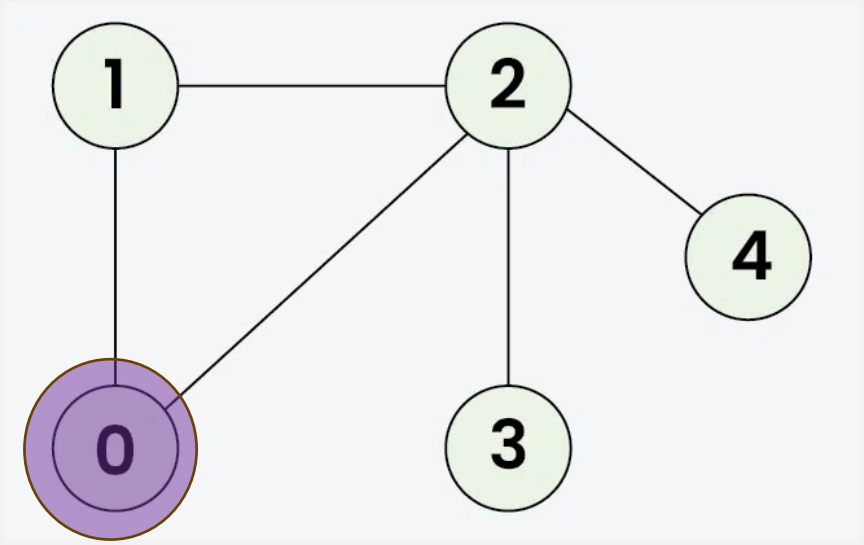

BFS
Layer 0
0
Layer 1
1
2
Layer 2
3
4
5
Output: 0, 1, 2, 3, 4, 5
VS
DFS
0
Source Node
1
2
3
4
5
Output: 0, 1, 3, 4, 2, 5

BFS | ΠΑΡΑΔΕΙΓΜΑ (Ι)

- **Graph**: 2D list με τους adjacent γειτονικούς) nodes για τους nodes 0..4

0 1 2 3 4

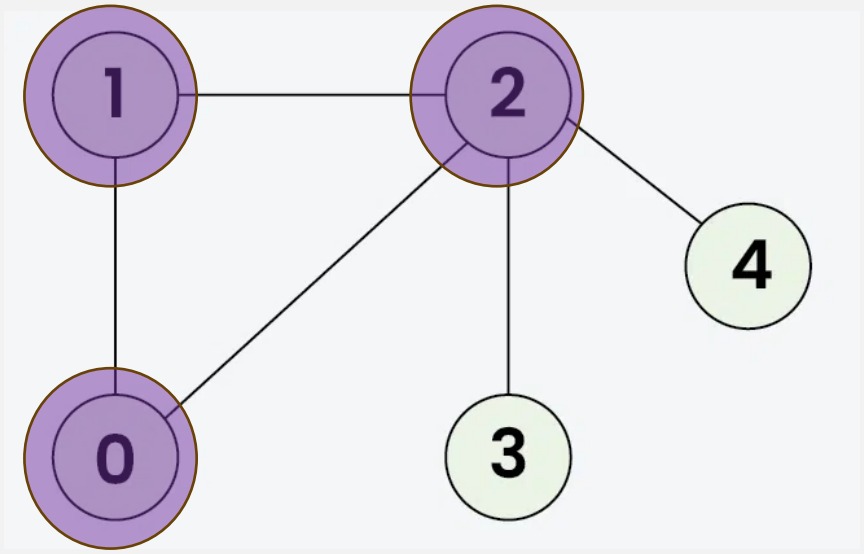
- adj = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]
- Αλγόριθμος **BFS**
  - ξεκινάω από node 0, enqueue 0, mark as visited



Visited:	0				
Queue:	0				

BFS | ΠΑΡΑΔΕΙΓΜΑ (2)

- Αλγόριθμος **BFS**
  - dequeue node 0, enqueue unvisited neighbors (node 1, node 2), mark them as visited

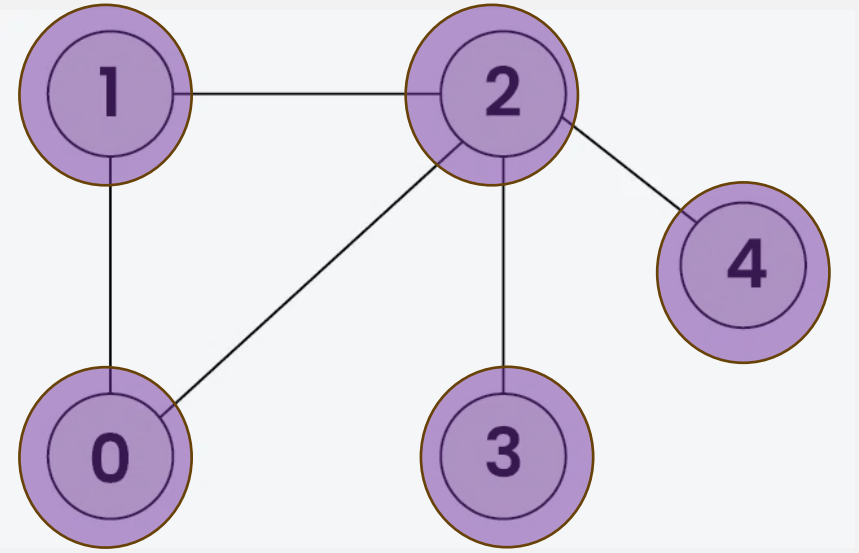


Visited:	0	1	2		
Queue:	0	1	2		

αρχή ουράς

BFS | ΠΑΡΑΔΕΙΓΜΑ (3)

- Αλγόριθμος **BFS**
  - dequeue node 1, no unvisited neighbors
  - dequeue node 2, enqueue unvisited neighbors (node 3, node 4), mark them as visited
  - dequeue node 3, node 4, no unvisited neighbors



Visited:	0	1	2	3	4
Queue:	0	1	2	3	4

αρχή ουράς