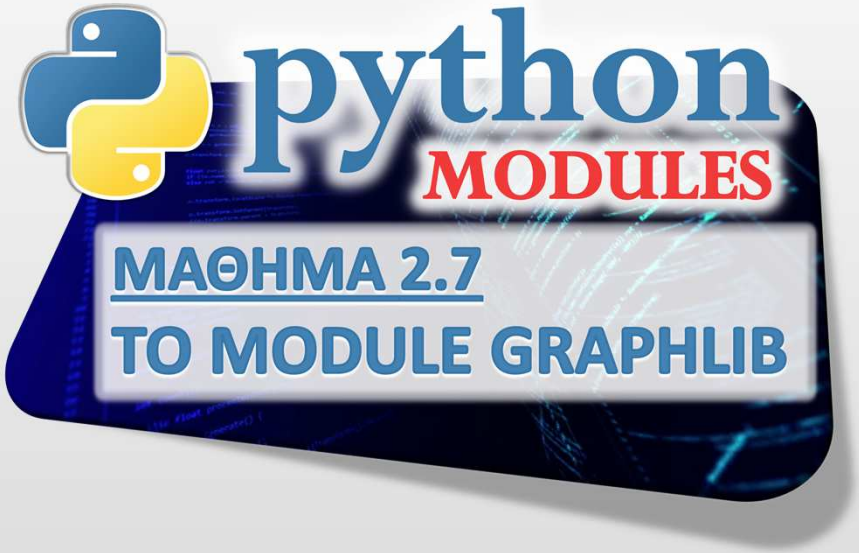


ΠΕΡΙΕΧΟΜΕΝΑ:

1. Ορισμός Γράφου
2. Υπολογισμός Διάταξης

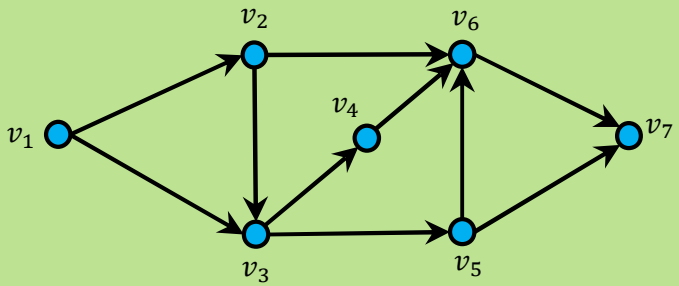
ΜΑΘΗΜΑ 2.7: Το module graphlib

1. Ορισμός Γράφου

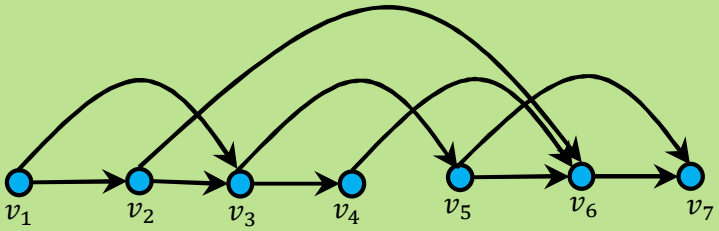
modules psounis

<u>Το module graphlib</u> περιέχει:

- **<u>Υλοποίηση του αλγόριθμου τοπολογικής ταξινόμησης</u>** σε ένα γράφο (που μπορεί να γίνει και παράλληλα με πολλαπλά νήματα)
- Ένας κατευθυνόμενος γράφος είναι μια δομή δεδομένων που οι κόμβοι συνδέονται με κατευθυνόμενες ακμές.



- Μία τοπολογική ταξινόμηση των κόμβων, είναι μία διάταξή τους τέτοια ώστε να μην υπάρχει ακμή από κάποιον κόμβο σε προηγούμενό του.
- Έτσι π.χ. μία τοπολογική ταξινόμηση των κόμβων του παραπάνω γραφήματος είναι η:



- Η τοπολογική ταξινόμηση έχει εφαρμογές στον προγραμματισμό διεργασιών, τη σειριοποίηση αντικειμένων κ.α.

<u>Σημαντικό:</u>

- Το γράφημα **<u>πρέπει να είναι άκυκλο</u>**.

- Ο γράφος που απαιτεί το module, πρέπει να αποθηκεύεται στη μορφή λεξικού:
  - Τα κλειδιά είναι οι κόμβοι
  - Η τιμή πρέπει να είναι ένα σύνολο με τους προηγούμενους του κόμβου.
- Αρχικοποιούμε τη δομή του αλγορίθμου με τον κατασκευαστή:

TopologicalSorter(graph=None)	Αρχικοποίηση με το γράφο graph

- προσθέτουμε (επιπλέον) κόμβους με τη μέθοδο του αντικ/νου:

add(node, *pred)	Προσθέτει τον κόμβο, ως επόμενο των pred

- και προετοιμάζουμε το αντικείμενο προς εκτέλεση με την:

prepare()	Οριστικοποιεί το αντικ/νο. Προκαλεί CycleException αν υπάρχει κύκλος.

<u>Παράδειγμα 1: prepare_for_ts</u>

```
graph = {
   "v1": {},
   "v2": {"v1"},
   "v3": {"v1", "v2"},
   "v4": {"v3"},
   "v5": {"v3"},
   "v6": {"v2", "v4", "v5"},
   "v7": {"v5", "v6"}
}
```

```
sorter = TopologicalSorter(graph)
# sorter.prepare()

sorter.add("v1", "v5")
sorter.prepare()
```

- Ο υπολογισμός της τοπολογικής ταξινόμησης γίνεται με τη μέθοδο του αντικειμένου TopologicalOrder:

Μέθοδος	Επεξήγηση
static_order()	Επιστρέφει iterator με τις κορυφές, ταξινομημένες σύμφωνα με την τοπολογική ταξινομηση.

Παράδειγμα 2: calculate_ts.py

```
sorter = TopologicalSorter(graph)
order = sorter.static_order()
print(list(order))
```

Σημείωση:

- Η static_order() καλεί την prepare() και δεν πρέπει να την καλέσουμε εμείς.

- **Β' τρόπος χρήσης:** Να παίρνουμε τους κόμβους «κατά επίπεδα» της ταξινόμησης (δύο κόμβοι μπορούν να είναι στο ίδιο επίπεδο, αν ο ένας δεν είναι προηγούμενος του άλλου και σέβονται τους περιορισμούς της ταξινόμησης)

Μέθοδος	Επεξήγηση
get_ready()	Επιστρέφει τους κόμβους που είναι έτοιμοι προς «υπολογισμό» (δηλ. κόμβοι που δεν έχουν προηγούμενους)
done(*nodes)	Ορίζει ότι οι κόμβοι nodes, «έχουν υπολογιστεί», άρα οι επόμενοι τους δεν έχουν πλέον εξάρτηση από αυτούς.
is_active()	True, αν υπάρχουν ακόμη δεδομένα προς «υπολογισμό»

Παράδειγμα 3: searching.py

```
graph = {
   "v1": {},
   "v2": {"v1"},
   "v3": {"v1", "v2"},
   "v4": {"v3"},
   "v5": {"v3"},
   "v6": {"v2", "v4", "v5"},
   "v7": {"v5", "v6"}
}

sorter = TopologicalSorter(graph)
sorter.prepare()

while sorter.is_active():
   ready = sorter.get_ready()
   # do something with these nodes
   print(ready)
   sorter.done(*ready)
```