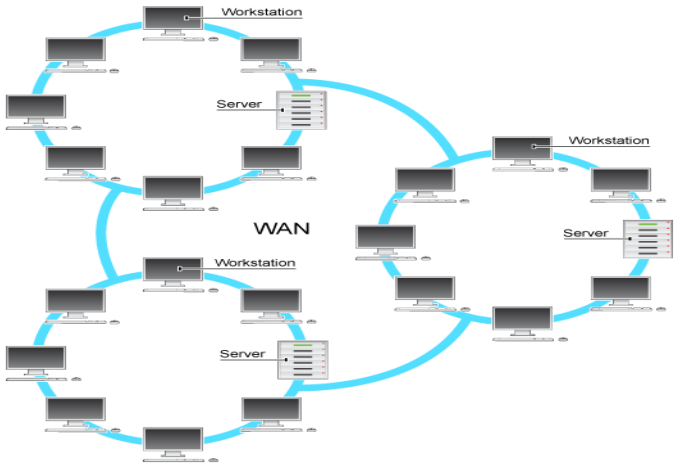
ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ
6° Εξάμηνο

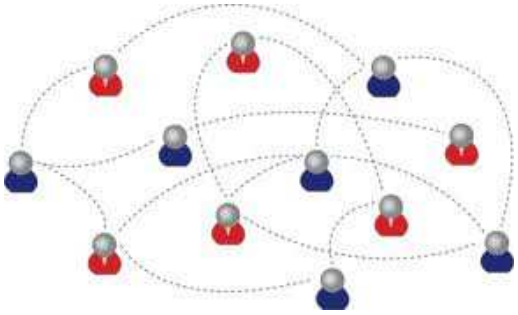
Προβλήματα που βασίζονται σε γραφήματα (graphs)

Δημοσθένης Σταμάτης
http:/www.iee.ihu.gr/~demos
Τμήμα Μηχανικών Πληροφορικής & Ηλεκτρονικών Συστημάτων

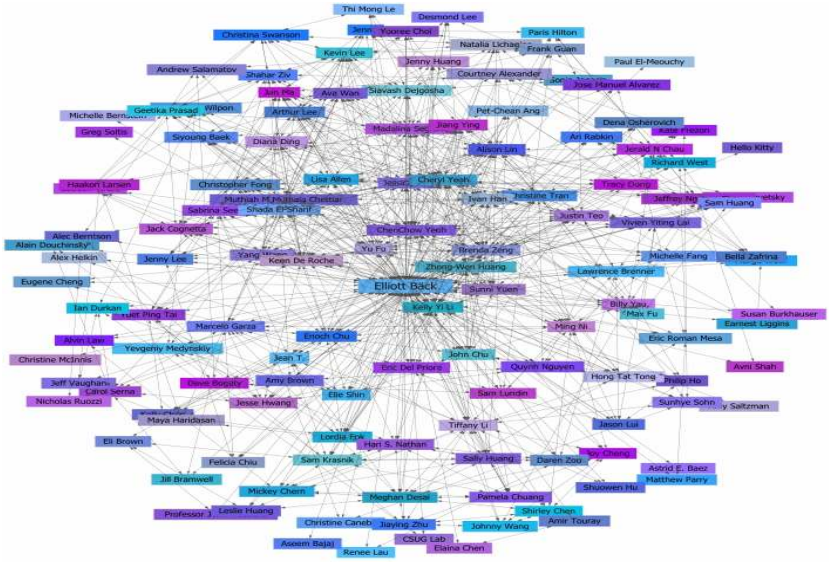
Παραδείγματα Γραφημάτων:
Δίκτυα Υπολογιστών



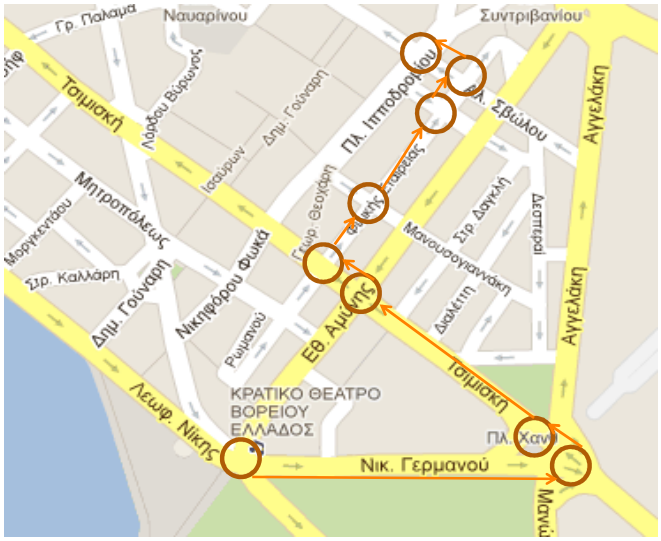
Παραδείγματα Γραφημάτων:
Κοινωνικά δίκτυα



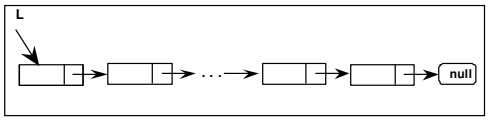
Κοινωνικά δίκτυα

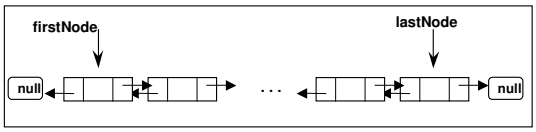


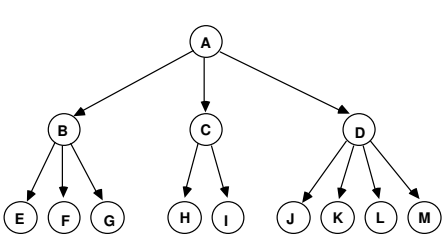
Αναπαράσταση Προβλημάτων με Γραφήματα

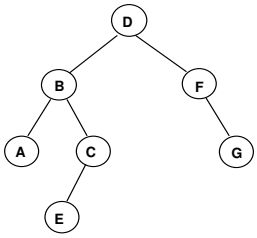


Παραδείγματα Γραφημάτων: Συνδεδεμένες δομές δεδομένων



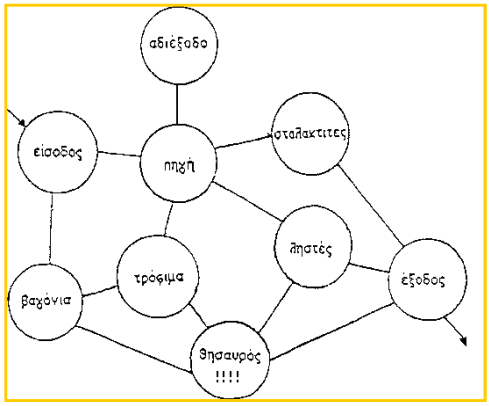






Το Πρόβλημα της Αναζήτησης του Θησαυρού

Το παρακάτω γράφημα παριστάνει μία σπηλιά με επιμέρους χώρους. Να βρεθεί διαδρομή από την είσοδο στην έξοδο που να περνάει από το θησαυρό αλλά να μην περνάει από τους ληστές.



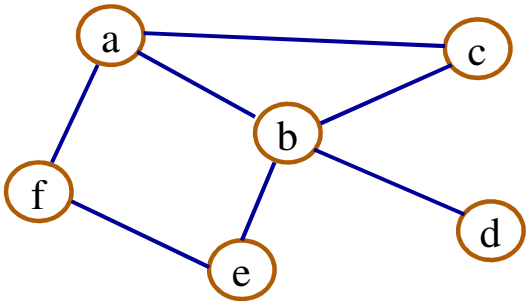
- **Γραφήματα ή Γράφοι**
  (Graphs)

Ορισμοί-Αναπαράσταση-Επίλυση

Γράφημα (Graph)

Ορισμός 1:

Έστω το μη κενό και πεπερασμένο σύνολο **V** *με* **n** *διακεκριμένα στοιχεία* $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$, *και* **E** *ένα σύνολο με* $\mathbf{m} \geq \mathbf{0}$ *με μη-διατεταγμένα ζεύγη* $e_{ij} = \{v_i, v_j\}$, $i \neq j$, *στοιχείων του* **V**.
Τότε το διατεταγμένο ζεύγος **G = (V,E)** *ονομάζεται* ***μη κατευθυνόμενο γράφημα*** *(undirected graph) ή απλώς* ***γράφημα****.*



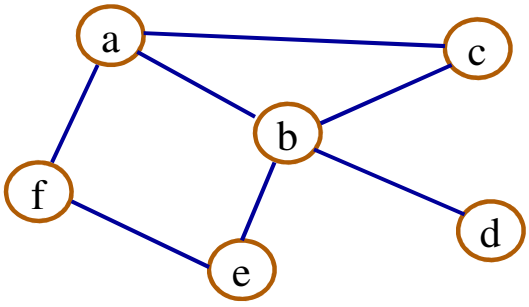
Μη κατευθυνόμενο γράφημα

Στο παράδειγμα:

G = (V, E) *με*:

V = {a, b, c, f, e, d}, *και*

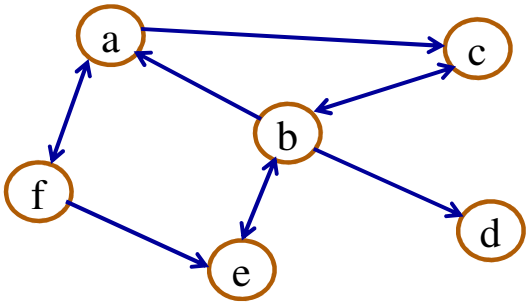
E = { {a, b}, {a, c}, {a, f}, {b, c}, {b, d}, {b, e}, {f, e} }



Κατευθυνόμενο Γράφημα (Directed Graph)

Ορισμός 2:

Έστω το μη κενό και πεπερασμένο σύνολο **V** *με* **n** *διακεκριμένα στοιχεία* $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$, *και* **E** *ένα σύνολο με* $\mathbf{m} \geq \mathbf{0}$ με *διατεταγμένα ζεύγη* $e_{ij} = \{v_i, v_j\}$, $i \neq j$, *στοιχείων του* **V**.
Τότε το διατεταγμένο ζεύγος **DG = (V,E)** *ονομάζεται* ***κατευθυνόμενο γράφημα*** *(directed graph).*

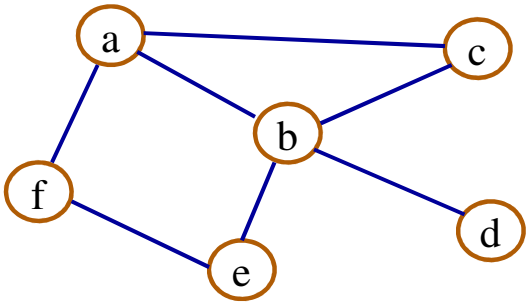


Τάξη γραφήματος

Ορισμός 3:

Τάξη (order) ενός γραφήματος ονομάζεται το πλήθος των κορυφών του , ενώ ***μέγεθος (size)*** ονομάζεται το πλήθος των ακμών του.

Στο παράδειγμα η ***order = 6*** και ***size = 7***

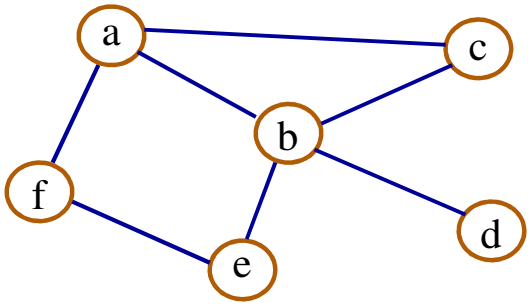


Διπλανές κορυφές γραφήματος
(ή διπλανοί κόμβοι)

Ορισμός 4:

Δύο κορυφές v_i, v_j ονομάζονται ***διπλανές*** *(adjacent)* όταν υπάρχει ακμή $e_{ij} = \{v_i, v_j\}$, $i =/= j$ που να τις έχει άκρα. Δύο κορυφές που δεν είναι διπλανές ονομάζονται ***ανεξάρτητες*** *(independent).*

Στο παράδειγμα οι ***a*** και ***b είναι*** *διπλανές* ενώ οι ***a*** και ***d*** *ανεξάρτητες*

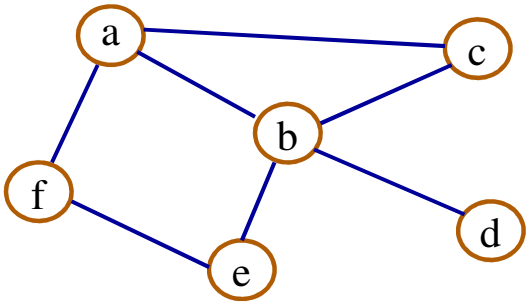


Βαθμός κορυφής ενός γραφήματος

Ορισμός 5:

Βαθμός (degree) μιας κορυφής ονομάζεται το πλήθος των διπλανών κορυφών της, ή αλλιώς το πλήθος των ακμών που πρόσκεινται στην κορυφή. Μια κορυφή ονομάζεται *άρτια* ή *περιττή* ανάλογα με το αν ο βαθμός της είναι άρτιος ή περιττός

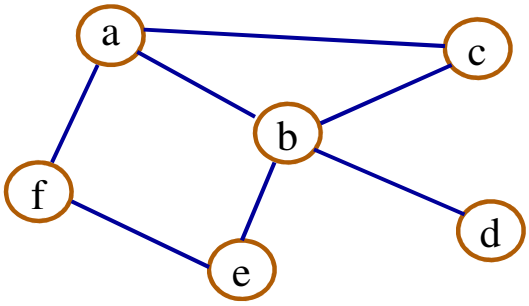
Στο παράδειγμα η ***b*** είναι ***άρτια*** και ο ***βαθμός*** της είναι ***4***



Τερματική κορυφή γραφήματος

Ορισμός 6:
Μία κορυφή v_i ονομάζεται ***τερματική*** *κορυφή (end vertex)* αν degree(v_i) = 1.

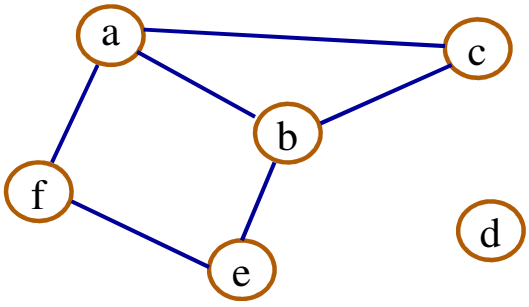
Στο παράδειγμα η κορυφή ***d*** είναι ***τερματική***



Απομονωμένη κορυφή γραφήματος

Ορισμός 7:
Μία κορυφή v_i ονομάζεται ***απομονωμένη*** *(isolated)*, αν ο βαθμός της degree(v_i) = 0

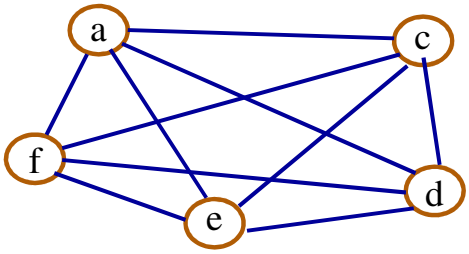
Στο παράδειγμα η κορυφή ***d*** είναι ***απομονωμένη***



Πλήρες Γράφημα (Graphs)

Ορισμός 8:
Ένα γράφημα ονομάζεται ***πλήρες*** *(complete graph)* όταν περιλαμβάνει όλους τους δυνατούς συνδυασμούς ακμών ανάμεσα στις κορυφές του.

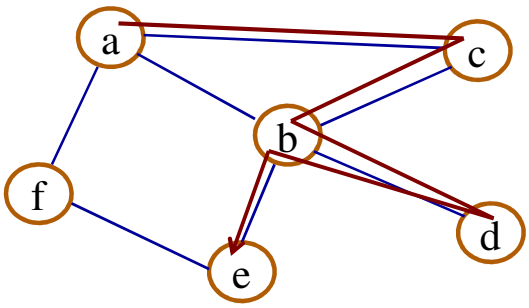
Παρατήρηση: Δεν αποτελεί συνηθισμένη περίπτωση!



Περίπατος γραφήματος

Ορισμός 9:
Ονομάζουμε ***περίπατο (walk)*** σε ένα γράφημα G μία ακολουθία κορυφών του γραφήματος, της μορφής $W = \langle v_0, v_1, v_2, \ldots, v_n \rangle$ και λέμε ότι έχει μήκος n.

Στο παράδειγμα: W = <a, c, b, d, b, e> με μήκος 5

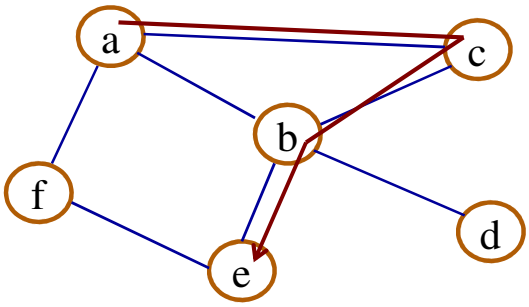


Μονοπάτι γραφήματος

Ορισμός 10:

Ονομάζουμε ***μονοπάτι*** *(path)* σε ένα γράφημα G μία ακολουθία κορυφών του γραφήματος, της μορφής $P = <v_0, v_1, v_2, \ldots, v_n>$ που είναι περίπατος στον οποίο καμία κορυφή δεν επαναλαμβάνεται

Στο παράδειγμα: P = <a, c, b, e> με μήκος 3

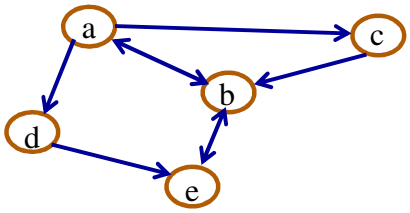


Αναπαράσταση γραφήματος

Πίνακας διπλανών κορυφών

Έστω ένα κατευθυνόμενο γράφημα με **n** κορυφές, οι οποίες μπορούν να αριθμηθούν με διακριτές συνεχόμενες τιμές. Η αναπαράστασή του μπορεί να γίνει με τη βοήθεια ενός δισδιάστατου πίνακα **D** με διαστάσεις n x n, στον οποίο:

- ✓ η θέση του πίνακα (i,j) δηλώνει την ύπαρξη ή όχι ακμής
- ✓ **D**(i,j)=1, αν υπάρχει η ακμή, **D**(i,j)=0, αν δεν υπάρχει



$$\mathbf{D} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

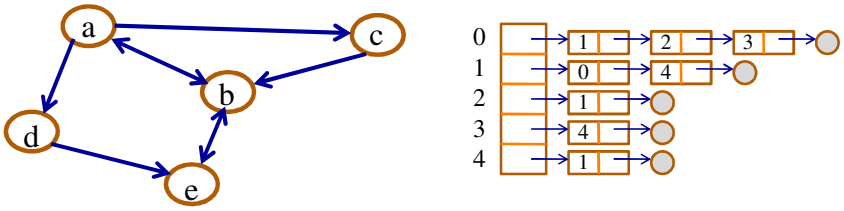
- ✓ Αρίθμηση κορυφών: a -> 0, b -> 1, c -> 2, d -> 3, e -> 4

Αναπαράσταση γραφήματος

Λίστες διπλανών κορυφών

Έστω ένα κατευθυνόμενο γράφημα με **n** κορυφές, οι οποίες μπορούν να αριθμηθούν με διακριτές συνεχόμενες τιμές. Η αναπαράστασή του μπορεί να γίνει με τη βοήθεια ενός μονοδιάστατου πίνακα συνδεδεμένων λιστών:

- ✓ η θέση i του πίνακα αντιστοιχεί στον κόμβο i
- ✓ το περιεχόμενο του πίνακα στη θέση i είναι η λίστα των κόμβων που συνδέονται με τον i



- ✓ Αρίθμηση κορυφών: a -> 0, b -> 1, c -> 2, d -> 3, e -> 4

Επίλυση προβλήματος γραφήματος

- **Αναζητούμε ένα μονοπάτι το οποίο να ξεκινάει από έναν κόμβο X (κόμβος αρχή) και να καταλήγει σε έναν κόμβο Y (κόμβος στόχος).**

- **Αντίστοιχα μπορεί να αναζητούμε έναν περίπατο ο οποίος να ξεκινάει από έναν κόμβο X (κόμβος αρχή) και να καταλήγει σε έναν κόμβο Y (κόμβος στόχος).**

- **Το γενικότερο πρόβλημα μπορεί να περιλαμβάνει:**
  1. **θετικούς περιορισμούς (πρέπει οπωσδήποτε να περάσουμε από κάποιους κόμβους).**
  2. **Αρνητικούς περιορισμούς (πρέπει οπωσδήποτε να αποφύγουμε κάποιους κόμβους).**
  3. **Πρέπει να λάβουμε υπόψη μας τα βάρη σύνδεσης των κόμβων (προβλήματα ελαχιστοποίησης ή μεγιστοποίησης**

Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

- **Στόχος μας είναι να επισκεφτούμε όλους τους κόμβους του γραφήματος**

- **Παράδειγμα**: Αλγόριθμος «πρώτα σε βάθος» (depth first search)

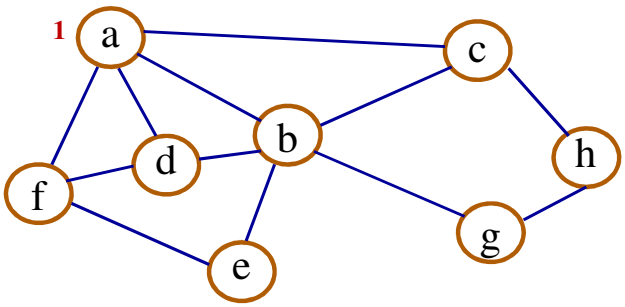
Ξεκινώντας από έναν κόμβο i:

✓ επισκεπτόμαστε τον κόμβο i
✓ επισκεπτόμαστε αναδρομικά κάθε κόμβο που συνδέεται με τον κόμβο i με την προϋπόθεση ότι δεν τον έχουμε ήδη επισκεφθεί.

Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

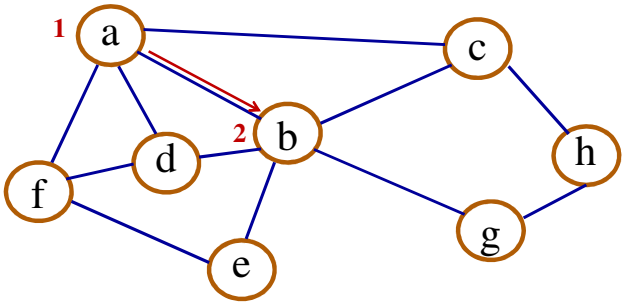
Σειρά επίσκεψης των κόμβων:



Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

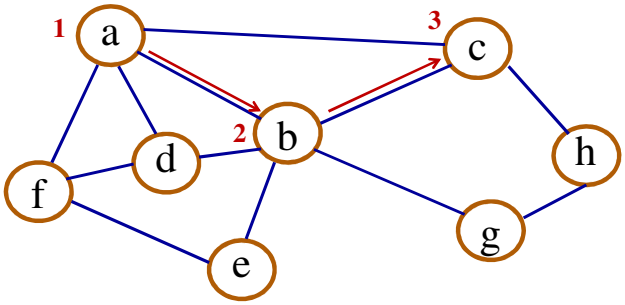
Σειρά επίσκεψης των κόμβων:



Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

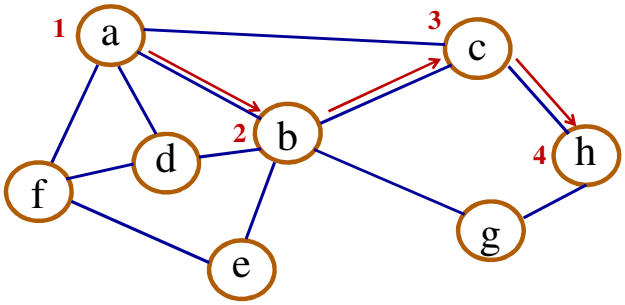
Σειρά επίσκεψης των κόμβων:



Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

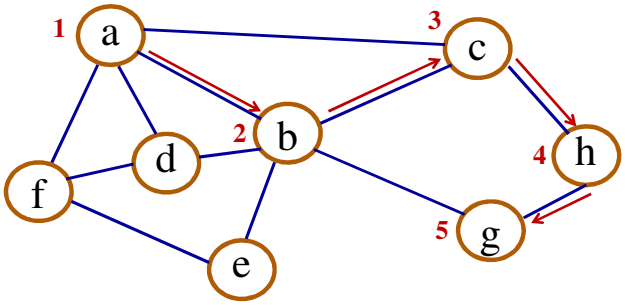
Σειρά επίσκεψης των κόμβων:



Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

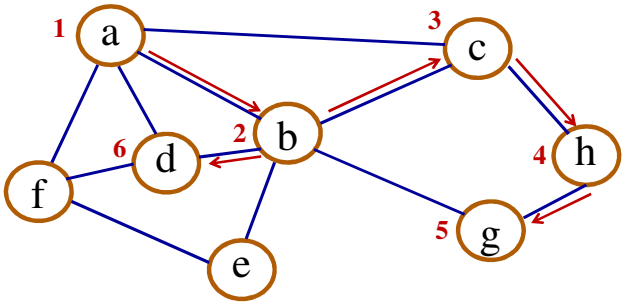
Σειρά επίσκεψης των κόμβων:



Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

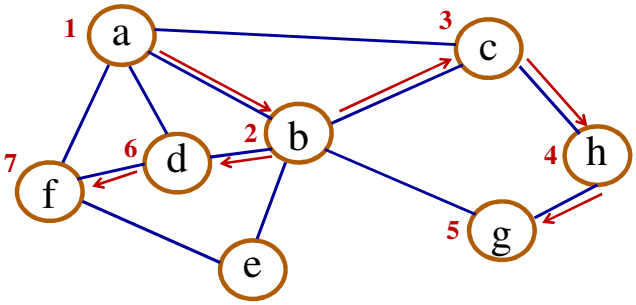
Σειρά επίσκεψης των κόμβων:



Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

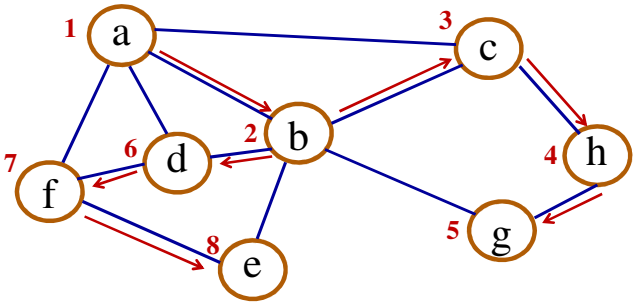
Σειρά επίσκεψης των κόμβων:



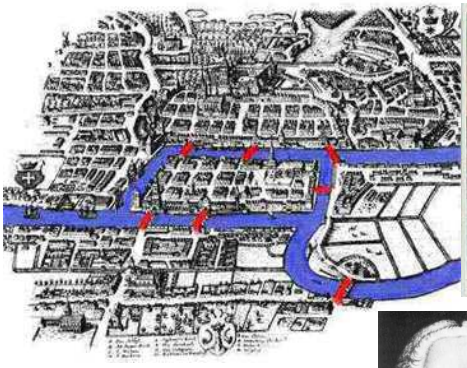
Αλγόριθμοι Επίσκεψης κόμβων γραφήματος

Αλγόριθμος «πρώτα σε βάθος» (depth first search)

Σειρά επίσκεψης των κόμβων:



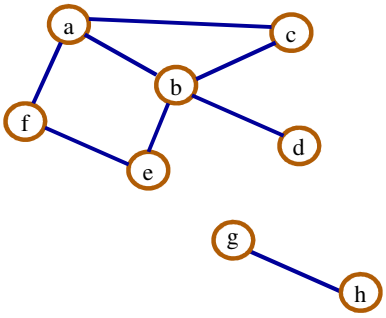
Λίγη ιστορία: οι γέφυρες του Konigsberg



Leonhard Euler
(1707-1783)

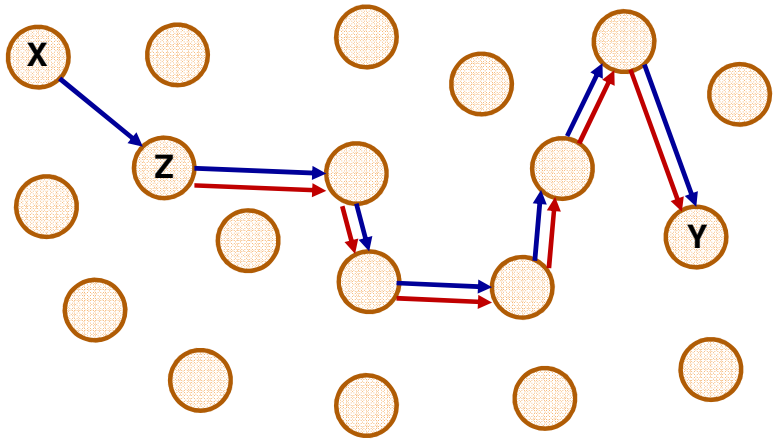
Εύρεση διαδρομής ανάμεσα σε 2 τυχαίους κόμβους
(αν υπάρχει ή όχι)

```
next_to(a, b).
next_to(a, c).
next_to(a, f).
next_to(b, c).
next_to(b, d).
next_to(b, e).
next_to(e, f).
next_to(g, h).

joint_with(X,Y):- next_to(X,Y).
joint_with(X,Y):- next_to(Y,X).

exists_path(Node, FinalNode):-
        joint_with(Node, FinalNode).

exists_path(Node, FinalNode):-
        joint_with(Node, NextNode),
        exists_path(NextNode, FinalNode).
```



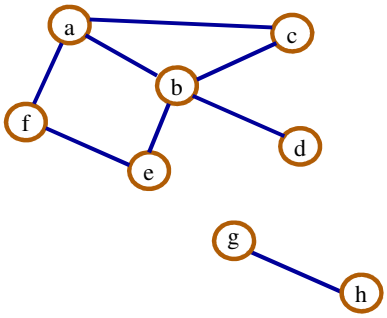
Εύρεση διαδρομής ανάμεσα σε 2 τυχαίους κόμβους **X** και **Y**

Εστω ότι υπάρχει η διαδρομή. Το πρόβλημα λύνεται αναδρομικά καθώς η διαδρομή από το **Z** στο **Y** είναι μικρότερη της αρχικής (από το **X** στο **Y**)



Εύρεση διαδρομής ανάμεσα σε 2 τυχαίους κόμβους
(η διαδρομή επιστρέφεται με τη μορφή λίστας – 3η παραμετρος)

```
next_to(a, b).
next_to(a, c).
next_to(a, f).
next_to(b, c).
next_to(b, d).
next_to(b, e).
next_to(e, f).
next_to(g, h).

joint_with(X,Y):- next_to(X,Y).
joint_with(X,Y):- next_to(Y,X).

path(Node, FinalNode, [Node, FinalNode]):-
        joint_with(Node, FinalNode).

path(Node, FinalNode, [Node | RestRoute]):-
        joint_with(Node, SomeNextNode),
        path(SomeNextNode, FinalNode, RestRoute).
```

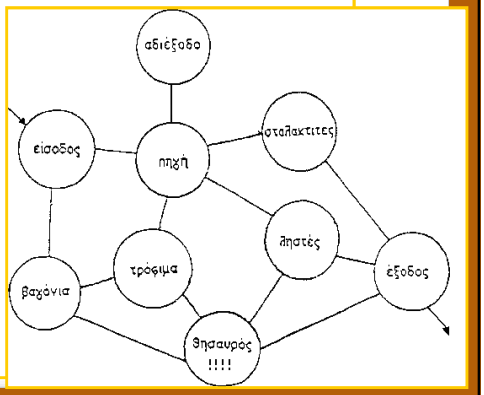


Εύρεση διαδρομής ανάμεσα σε 2 τυχαίους κόμβους
(χωρίς κύκλους: δεν επιτρέπεται η επίσκεψη κόμβου 2 φορές)

```
path_loopcheck(InitialNode, FinalNode, Route):-
        path_loopcheck (InitialNode, FinalNode, [InitialNode], Route).

path_loopcheck(Node, FinalNode, _, [Node, FinalNode]) :-
        joint_with(Node, FinalNode).

path_loopcheck(Node, FinalNode, VisitedNodes, [Node | RestRoute]) :-
        joint_with(Node, SomeNextNode),
        not(member(SomeNextNode, VisitedNodes)),
        path_loopcheck(SomeNextNode, FinalNode,
                [SomeNextNode|VisitedNodes], RestRoute).
```

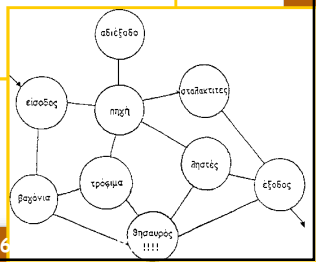
Το Πρόβλημα της Αναζήτησης του Θησαυρού

```
next_to(εισοδος,βαγονια).
next_to(εισοδος,πηγη).
next_to(πηγη,αδιεξοδο).
next_to(πηγη,ληστες).
next_to(πηγη,τροφιμα).
next_to(πηγη,σταλακτιτες).
next_to(ληστες,εξοδος).
next_to(τροφιμα,θησαυρος).
next_to(βαγονια,τροφιμα).
next_to(βαγονια,θησαυρος).
next_to(σταλακτιτες,εξοδος).
next_to(Θησαυρος,εξοδος).
next_to(ληστες,Θησαυρος).
```



Το Πρόβλημα της Αναζήτησης του Θησαυρού

```
joint_with(Place1,Place2) :-
        next_to(Place1,Place2).

joint_with(Place1,Place2) :-
        next_to(Place2,Place1).

avoid([αδιεξοδο,ληστες]).

traverse(Place1,Place2) :-
        avoid(Dangers),
        path (Place1,Place2,Dangers,[Place1]).
```



Το Πρόβλημα της Αναζήτησης του Θησαυρού

```
path(Place1,Place2,Dangers,Memory) :-
        member(Place2,Memory),
        member(θησαυρος,Memory),
        reverse_write(Memory).

path (Place1,Place2,Dangers,Memory):-
        joint_wíth(Place1,NewPlace),
        not member(NewPlace,Dangers),
        not member(NewPlace,Memory),
        path(NewPlace,Place2,Dangers,[NewPlace|Memory]).

reverse_write([ ]).
reverse_write([Head|Tail]) :-
        reverse_write(Tail), nl,
        write(Head).
```