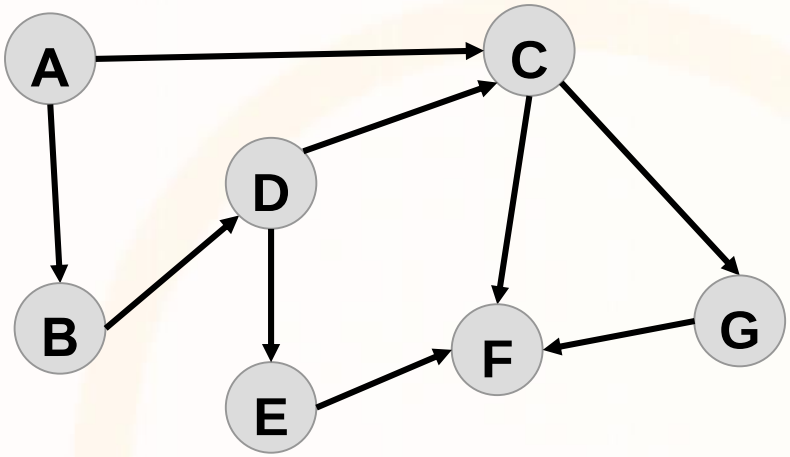
# Διάλεξη 20: Γράφοι Ι - Εισαγωγή

**Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:**

- Γράφοι - ορισμοί και υλοποίηση
- Διάσχιση Γράφων

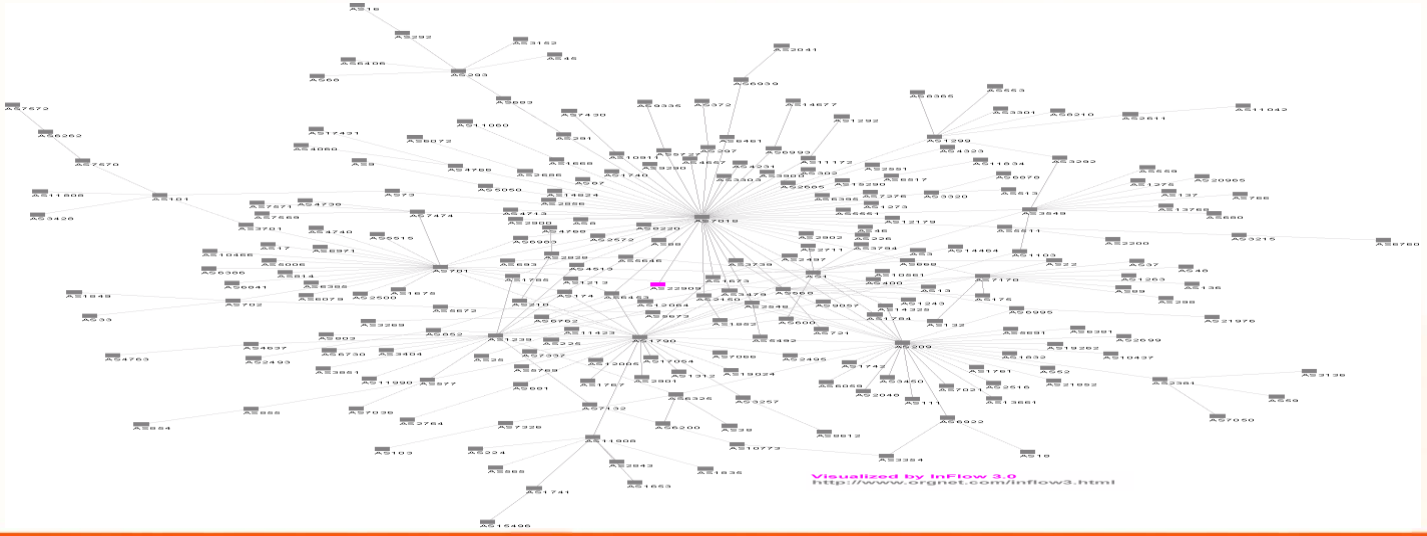
# Εισαγωγή στους Γράφους

- **Η πιο γενική μορφή δομής δεδομένων, με την έννοια ότι όλες οι προηγούμενες δομές μπορούν να θεωρηθούν ως περιπτώσεις γράφων.**
- Ένα γράφος **G(V,E)** αποτελείται από
  - ένα σύνολο **V κορυφών (vertices),** ή σημείων, ή κόμβων , και
  - ένα σύνολο **E ακμών (edges),** ή τόξων, ή γραμμών. Μια ακμή είναι ένα ζεύγος (u,v) από κορυφές.
- Παράδειγμα γράφου:



# Γράφοι

- Οι γράφοι προσφέρουν μια χρήσιμη μέθοδο για τη διατύπωση και λύση πολλών προβλημάτων, σε
  - δίκτυα και συστήματα τηλεπικοινωνιών (π.χ. το Internet),
  - χάρτες - επιλογή δρομολογίων ,
  - προγραμματισμό εργασιών (scheduling),
  - ανάλυση προγραμμάτων (flow charts).
- Η θεωρία των γράφων θεωρείται ότι ξεκίνησε από τον Euler στις αρχές του 18ου αιώνα (1736).



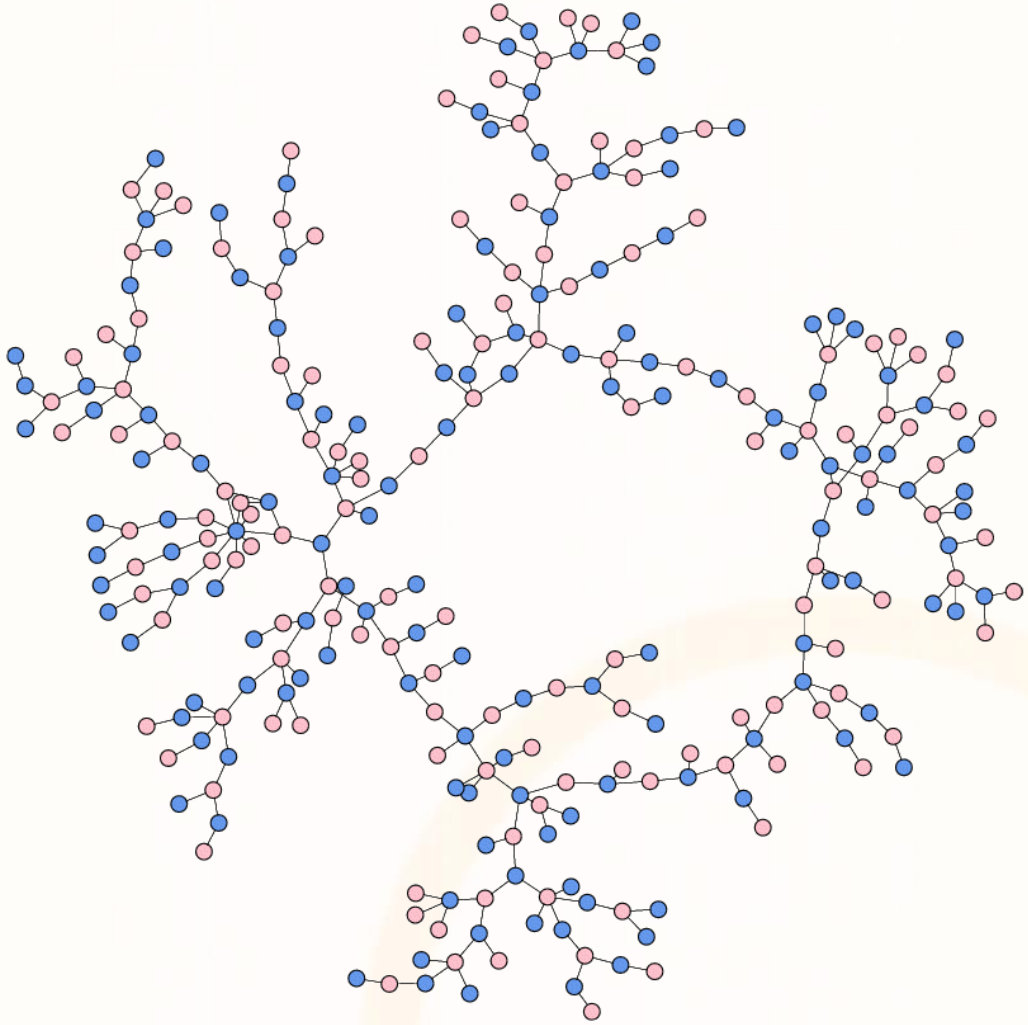
# ΕΦΑΡΓΟΓΕΣ ΓΡΑΦΩΝ

# Εφαρμογές Γράφων

| graph | vertices | edges |
|---|---|---|
| communication | telephones, computers | fiber optic cables |
| circuits | gates, registers, processors | wires |
| mechanical | joints | rods, beams, springs |
| hydraulic | reservoirs, pumping stations | pipelines |
| financial | stocks, currency | transactions |
| transportation | street intersections, airports | highways, airway routes |
| scheduling | tasks | precedence constraints |
| software systems | functions | function calls |
| internet | web pages | hyperlinks |
| games | board positions | legal moves |
| social relationship | people, actors | friendships, movie casts |
| neural networks | neurons | synapses |
| protein networks | proteins | protein-protein interactions |
| chemical compounds | molecules | bonds |

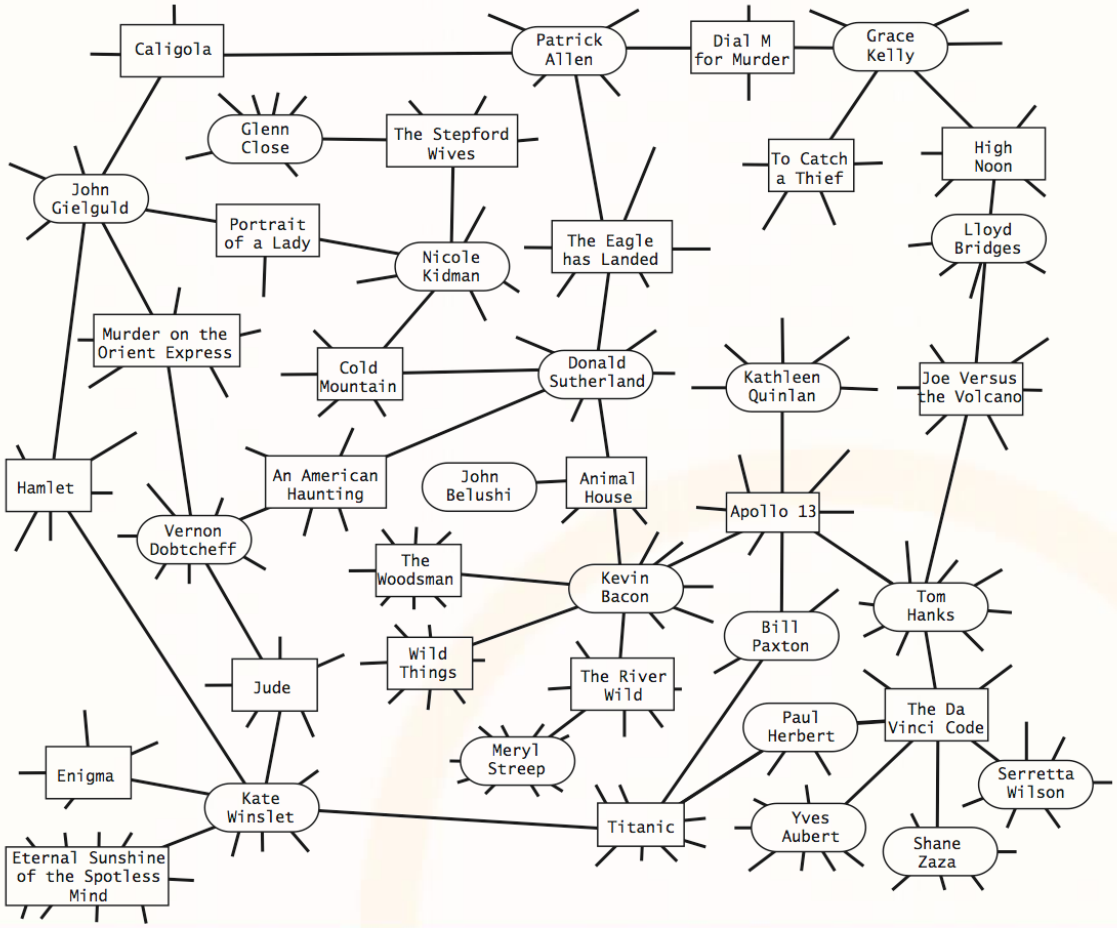
# High School Dating

- Bearman, Moody, and Stovel, 2004 Image by Mark Newman



# Internet Movie Database

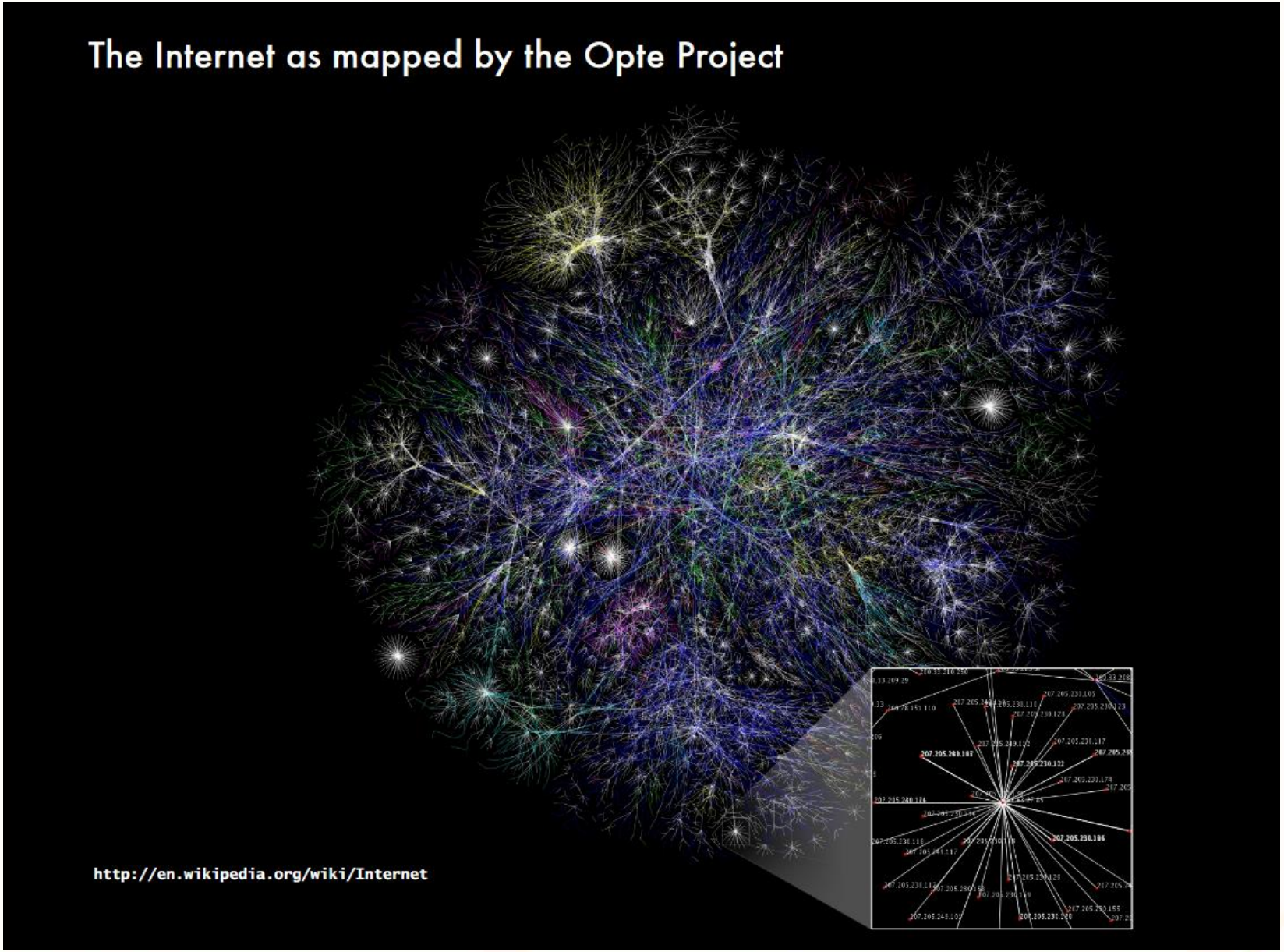
- Πώς να αναπαραστήσουμε τις σχέσεις μεταξύ των ηθοποιών και των ταινιών;
- Να χρησιμοποιήσουμε τον τύπο δεδομένων γράφο.
  - Κόμβος: ηθοποιός ή ταινία.
  - Ακμή: σύνδεση του ηθοποιού με την ταινία.



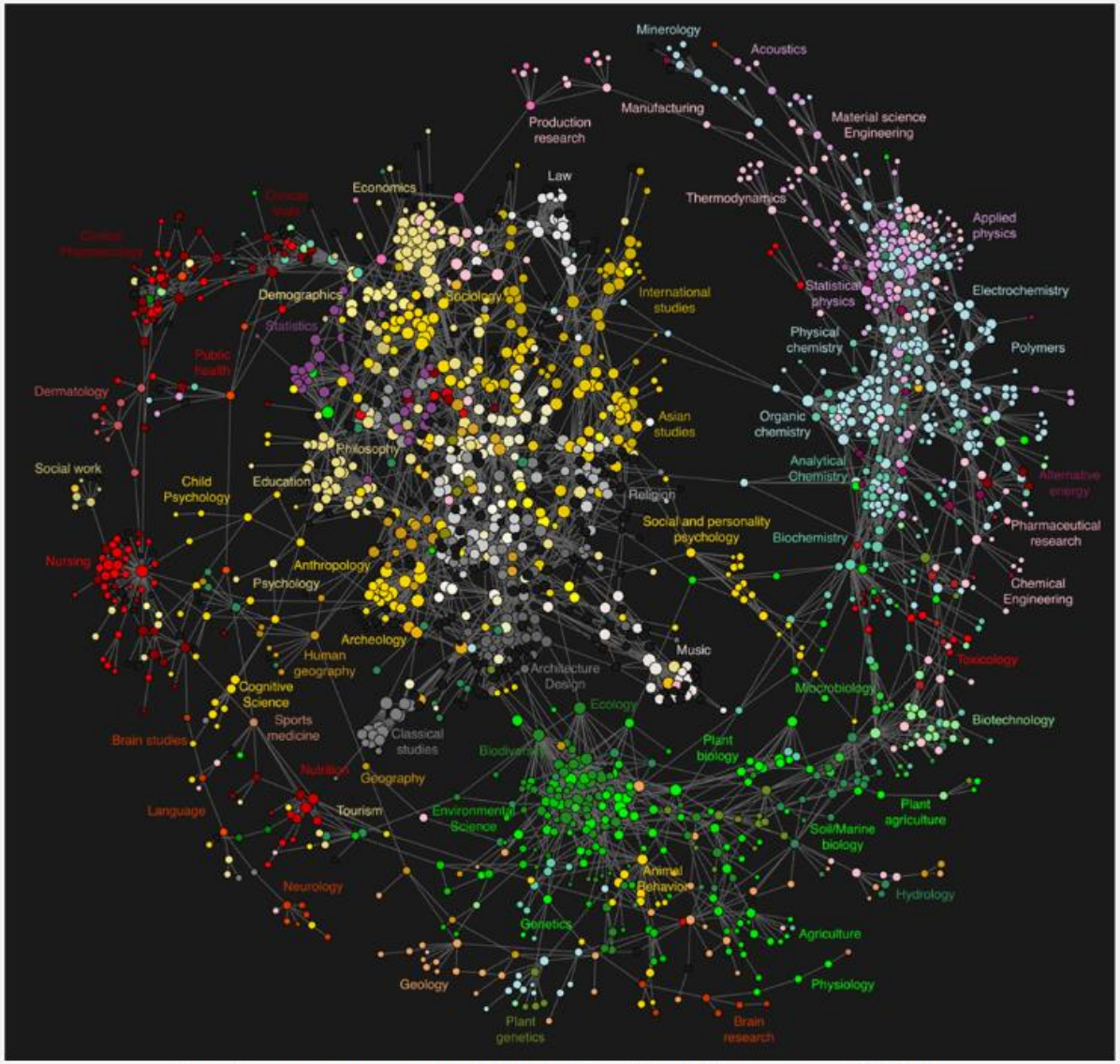
# Protein Interaction Network

- Jeong et al, Nature Review | Genetics

# The Internet as mapped by the Opte Project



# Map of science clickstreams



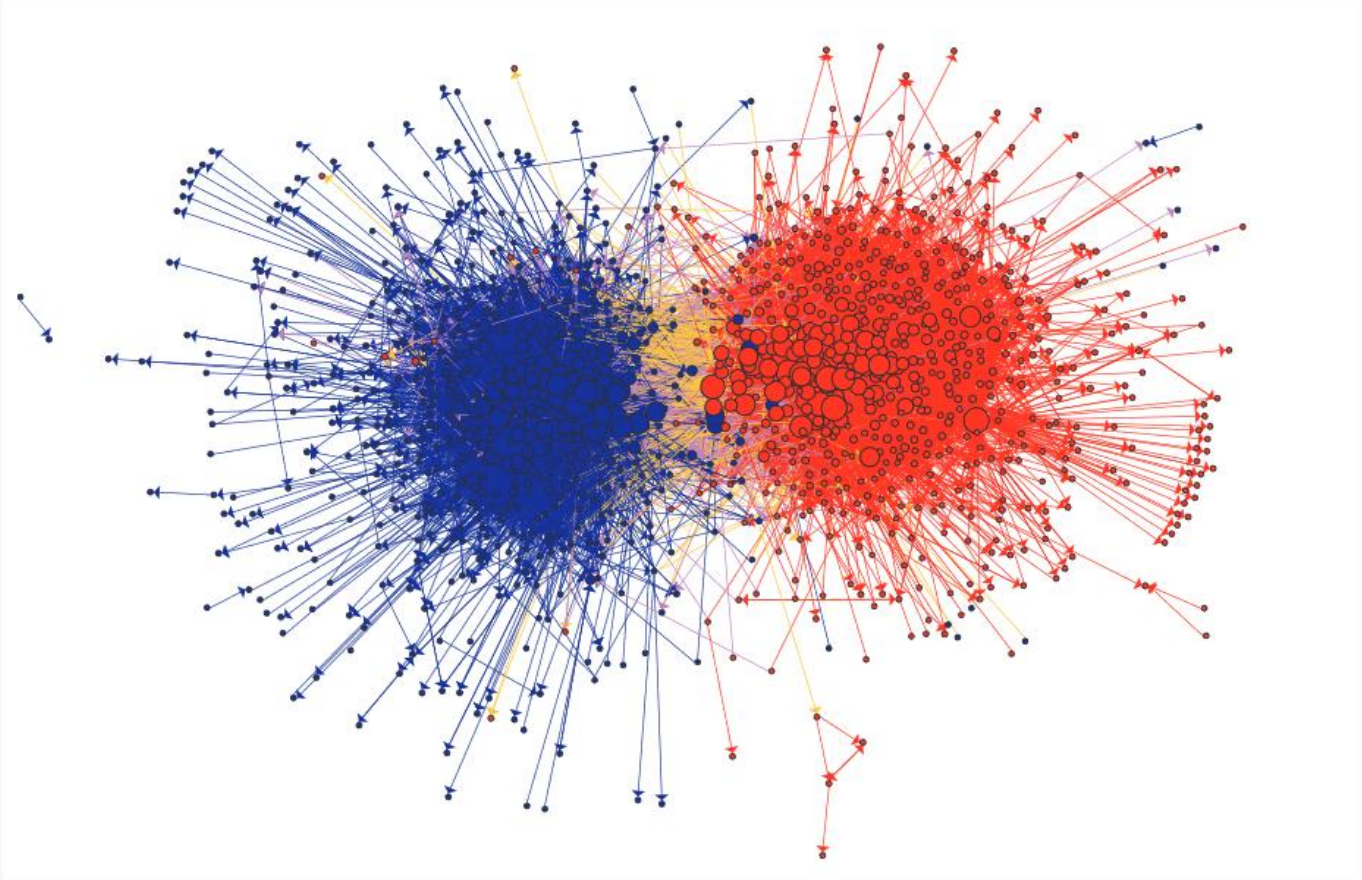

http://www.plosone.org/article/info:doi/10.1371/journal.pone.0004803

# 10 million Facebook friends


"Visualizing Friendships" by Paul Butler

# Political blogosphere graph

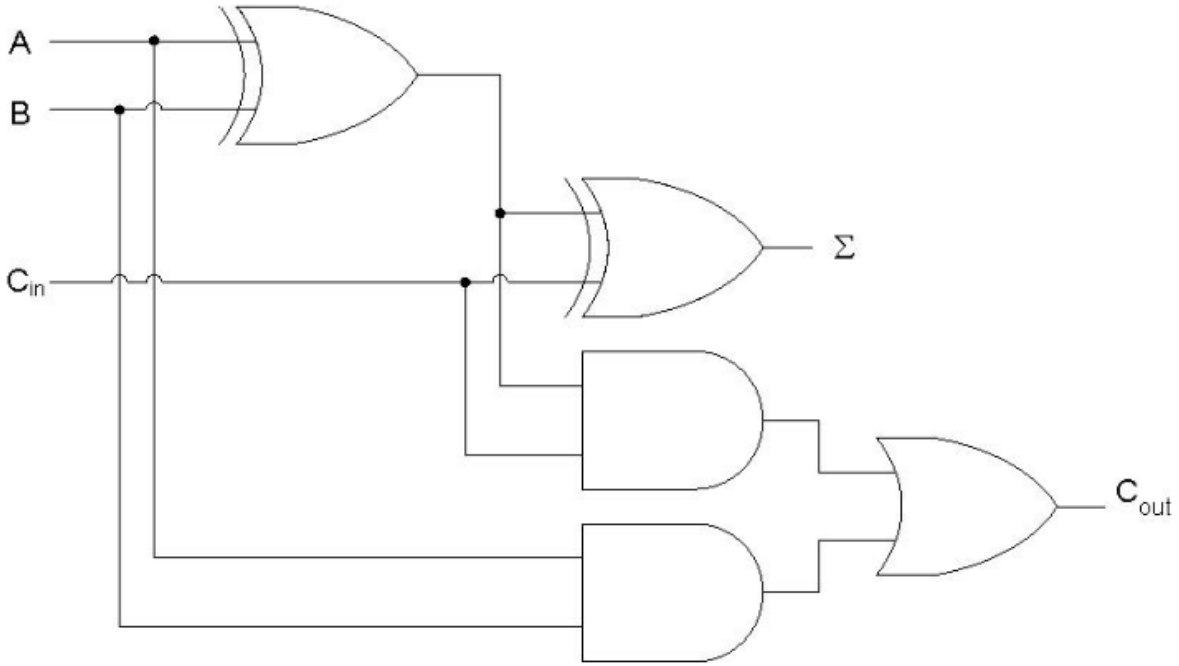
Vertex = political blog; edge = link.



The Political Blogosphere and the 2004 U.S. Election: Divided They Blog, Adamic and Glance, 2005

# Combinational circuit

Vertex = logical gate; edge = wire.



# Digraph applications

| digraph | vertex | directed edge |
|---|---|---|
| transportation | street intersection | one-way street |
| web | web page | hyperlink |
| food web | species | predator-prey relationship |
| WordNet | synset | hypernym |
| scheduling | task | precedence constraint |
| financial | bank | transaction |
| cell phone | person | placed call |
| infectious disease | person | infection |
| game | board position | legal move |
| citation | journal article | citation |
| object graph | object | pointer |
| inheritance hierarchy | class | inherits from |
| control flow | code block | jump |

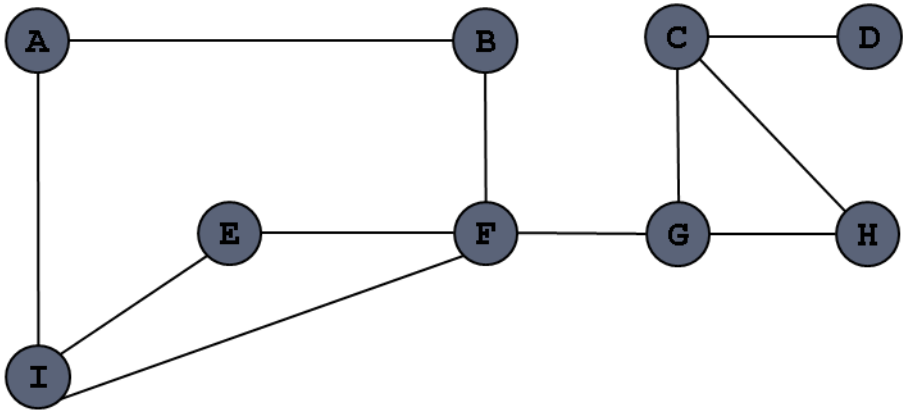
# Graph Sample API

- Graph data type.

public class Graph (graph with String vertices)

| | | |
|---|---|---|
| | Graph() | *create an empty graph* |
| | Graph(In in) | *read graph from input stream* |
| void | addEdge(String v, String w) | *add edge v-w* |
| Iterable<String> | adjacentTo(String v) | *neighbors of v* |

για να υποστηρίζει τη χρήση του foreach




```
% more tiny.txt
A/B/I
B/A/F
C/D/G/H
D/C
E/F/I
F/B/E/G
G/C/F/H
H/C/G
I/A/E/F
```

# Graph Sample API

| public class Graph | |
|---|---|
| `Graph(int V)` | *create an empty graph with V vertices* |
| `Graph(In in)` | *create a graph from input stream* |
| `void addEdge(int v, int w)` | *add an edge v-w* |
| `Iterable<Integer> adj(int v)` | *vertices adjacent to v* |
| `int V()` | *number of vertices* |
| `int E()` | *number of edges* |
| `String toString()` | *string representation* |

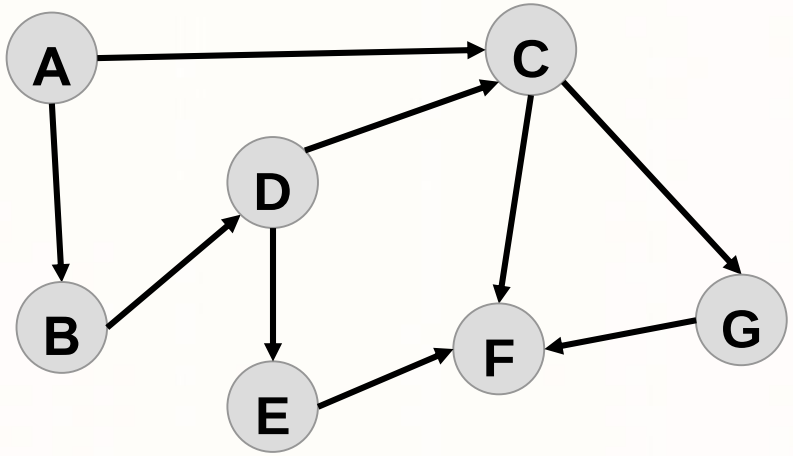
```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
   for (int w : G.adj(w))
      StdOut.println(v + "-" + w);
```

read graph from input stream

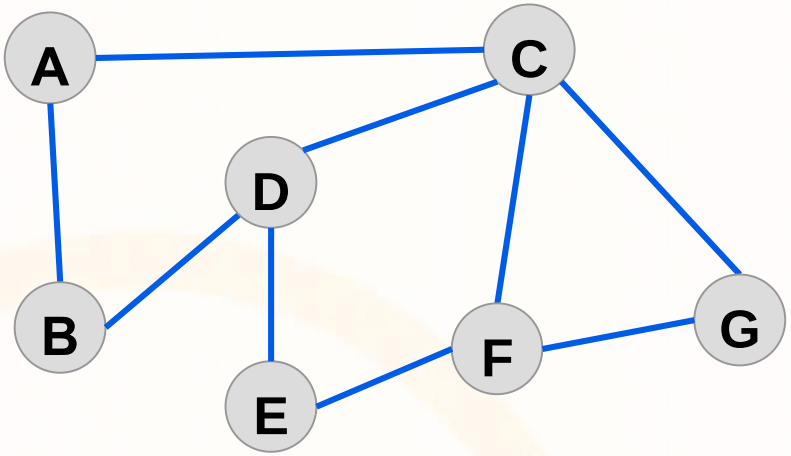
print out each edge (twice)

# Ορισμοί

- Ένας γράφος ονομάζεται **κατευθυνόμενος (directed graph, digraph)** αν κάθε μια από τις ακμές του είναι προσανατολισμένη προς μία κατεύθυνση. (πχ: ένα οδικό δίκτυο με λωρίδες μονής κατεύθυνσης)



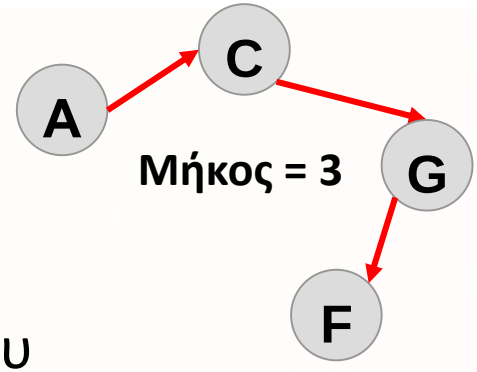

- Ένας γράφος ονομάζεται **μη-κατευθυνόμενος (undirected)** αν οι ακμές του δεν είναι προσανατολισμένες.(πχ: ένα οδικό δίκτυο με λωρίδες διπλής κατεύθυνσης)




- Αν (u,v) είναι ακμή τότε λέμε ότι οι κορυφές u και v είναι **γειτονικές (adjacent)** ή ότι *γειτνιάζουν*.

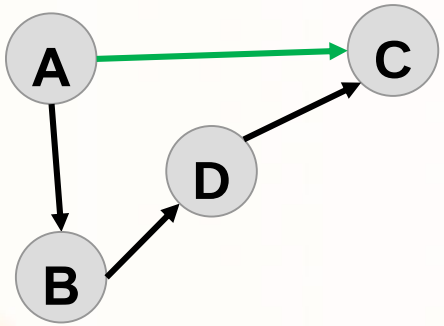
# Ορισμοί

- **Μονοπάτι ή διαδρομή (path)** ενός γράφου μήκους n, είναι μια ακολουθία κόμβων $v_0, v_1, \ldots, v_n$, όπου για κάθε i , $0 \leq i < n$, $(v_i , v_{i+1})$ είναι ακμή του γράφου.

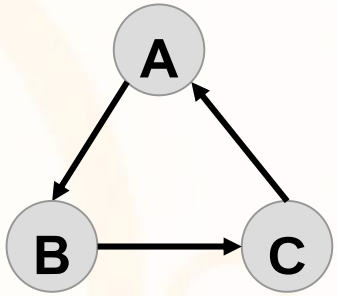



- **Μήκος** ενός μονοπατιού είναι ο αριθμός ακμών που περιέχει (εκτός και εάν ορίζεται διαφορετικά)

- **Βραχύτερο Μονοπάτι μεταξύ δυο κορυφών (Shortest Path):** Το μονοπάτι μεταξύ δυο κορυφών που έχει το ελάχιστο μήκος.



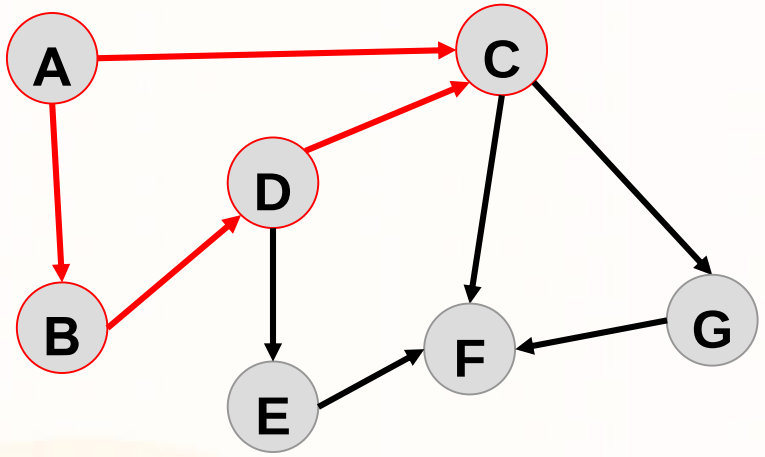

- Μια *διαδρομή* ενός γράφου ονομάζεται **απλή (simple path)** αν όλες οι κορυφές της είναι διαφορετικές μεταξύ τους, εκτός από την πρώτη και την τελευταία οι οποίες μπορούν να είναι οι ίδιες.
- **Κυκλική Διαδρομή (cycle)** ονομάζεται μια *διαδρομή* με μήκος >1 οπου $v_0 = v_n$.



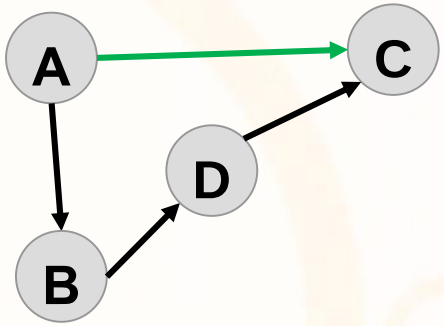
# Ορισμοί

- Ένας γράφος που δεν περιέχει κύκλους ονομάζεται **άκυκλος (acyclic graph)**

- Έστω **G=(V,E)** και **G΄= (V΄, E΄)** γράφοι, όπου V΄⊆V και E΄⊆ E.
  Τότε ο γράφος G΄ είναι **υπογράφος (subgraph)** του γράφου G.

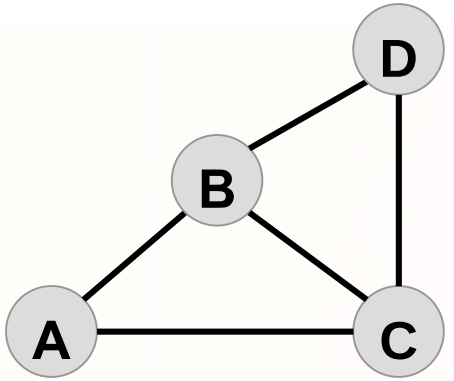



- **Απόσταση Κορυφών (Vertex Distance):** Το μήκος της συντομότερης διαδρομής που οδηγεί από τη μια κορυφή στην άλλη.

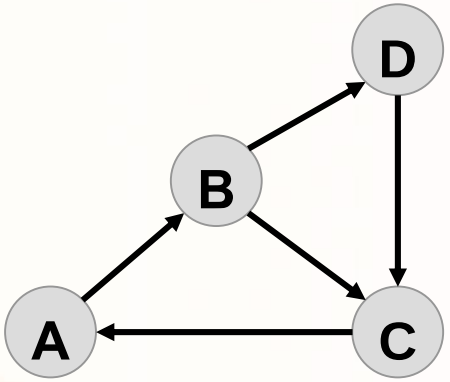


# Ορισμοί

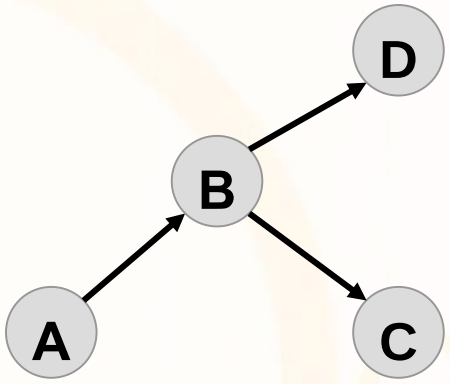
- **Συνεκτικός Γράφος (Connected Graph):** Ένας *μη-κατευθυνόμενος* γράφος στον οποίο υπάρχει τουλάχιστο μια διαδρομή μεταξύ όλων των κορυφών.




- **Ισχυρά Συνεκτικός Γράφος (Strongly Connected Graph):** Ένας *κατευθυνόμενος* γράφος στον οποίο υπάρχει μια διαδρομή από οποιαδήποτε κορυφή σε οποιανδήποτε άλλη.




- **Ελαφρά Συνεκτικός Γράφος (Weakly Connected Graph):** Ένας *κατευθυνόμενος* (συνεκτικός) γράφος στον οποίο **δεν** υπάρχει μια διαδρομή από οποιαδήποτε κορυφή σε οποιανδήποτε άλλη.



# Ορισμοί

- Πόσες **ακμές** μπορεί να έχει ένας γράφος με **n** κορυφές;

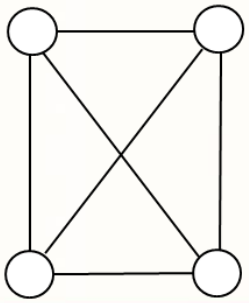
  Α) Κατευθυνόμενος: **nx(n-1)**  Β) Μη-κατευθυνόμενος: **nx(n-1)/2**

- **<u>Αραιός Γράφος (Sparse Graph):</u>** Αν ο αριθμός των ακμών του είναι της τάξης **O(n),** όπου n είναι ο αριθμός κορυφών του, διαφορετικά λέγεται **<u>Πυκνός Γράφος (Dense Graph)</u>**

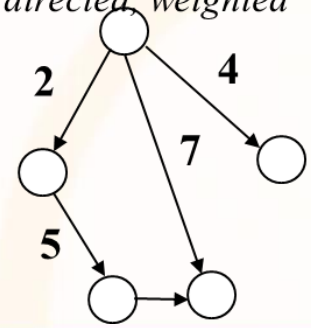
- **<u>Γράφος με Βάρη (Weighted Graph):</u>** Συχνά συσχετίζουμε κάθε ακμή ενός γράφου με κάποιο βάρος (weight).

- Ποιες ιδιότητες ικανοποιούν οι πιο κάτω γράφοι;

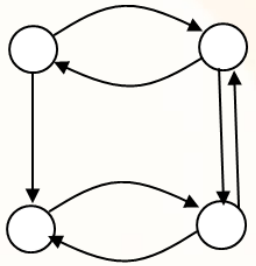
*Connected, undirected, unweighted*



*Weakly connected, directed, weighted*




*strongly connected directed, unweighted,*

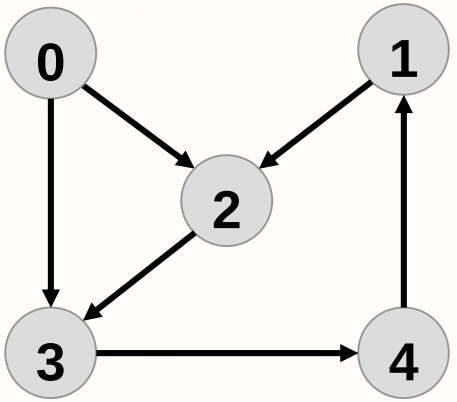


# Αναπαράσταση γράφων στην Μνήμη

## Αναπαράσταση γράφων με Πίνακες Γειτνίασης (Adjacency Matrix)

- Ένας γράφος G=(V,E) με n κορυφές μπορεί να αναπαρασταθεί ως ένας **n×n** πίνακας που περιέχει τις τιμές 0 και 1, και όπου
  - αν η (i,j) είναι ακμή τότε A[(i,j)]=1, διαφορετικά A[(i,j)]=0.
- Αν ο γράφος είναι γράφος με βάρη, και το βάρος κάθε ακμής είναι τύπου t, τότε για την αναπαράσταση του γράφου μπορεί να χρησιμοποιηθεί πίνακας τύπου t με
  - A[(i,j)] = βάρος(i,j), αν υπάρχει ακμή (i,j)
  - A[(i,j)] = ∞, αν δεν υπάρχει ακμή (i,j)
- Αυτή η αναπαράσταση απαιτεί χώρο $\Theta(n^2)$, όπου n=|V|.
- Αν ο γράφος είναι αραιός (δηλαδή τάξης O(n)) η μέθοδος οδηγεί σε σπάταλη χώρου.

**Γράφος**




**Πίνακας Γειτνίασης (Adjacency Matrix)**

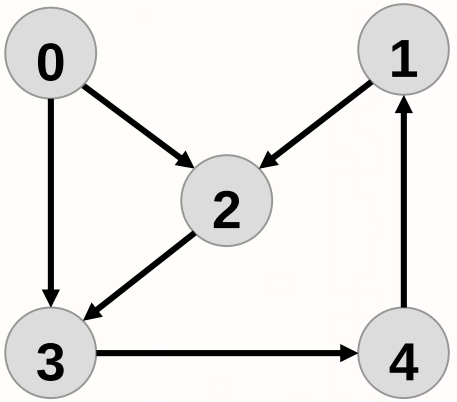
| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | 1 | 1 | |
| 1 | | | 1 | | |
| 2 | | | | 1 | |
| 3 | | | | | 1 |
| 4 | | 1 | | | |

# Αναπαράσταση γράφων στην Μνήμη

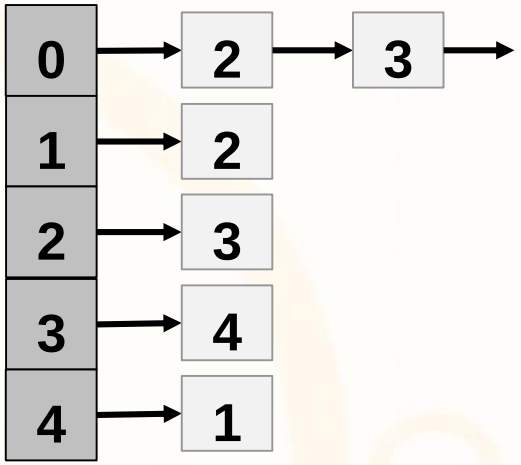
## Αναπαράσταση γράφων με Λίστες Γειτνίασης (Adjacency Lists)

- Ένας γράφος G=(V,E) αναπαρίσταται ως ένας μονοδιάστατος πίνακας Α.
- Για κάθε κορυφή **v**, **A[v]** είναι ένας δείκτης σε μια συνδεδεμένη λίστα στην οποία αποθηκεύονται οι κορυφές που γειτνιάζουν με την **v**.
- Η μέθοδος απαιτεί χώρο Θ(|V|+ |E|).
- **Επιτυγχάνεται εξοικονόμηση χώρου για αραιούς γράφους.**
- Στην περίπτωση γράφων με βάρη στη λίστα γειτνίασης αποθηκεύουμε επίσης το βάρος κάθε ακμής.

**Γράφος**




**Πίνακας Γειτνίασης (Adjacency Matrix)**



# Διάσχιση Γράφων

- Αν θέλουμε να επισκεφτούμε όλους τους κόμβους ενός γράφου μπορούμε να χρησιμοποιήσουμε έναν από πολλούς τρόπους, οι οποίοι διαφέρουν στη σειρά με την οποία εξετάζουν τους κόμβους.
- Διαδικασίες διάσχισης χρησιμοποιούνται για τη διακρίβωση ύπαρξης μονοπατιού μεταξύ δύο κόμβων κ.α.

**Άλλα Παραδείγματα**

1. **Οι μηχανές αναζήτησης** (search engines π.χ. Google), χρησιμοποιούν προγράμματα τα οποία ονομάζονται crawlers, για να διασχίσουν όλες τις ιστοσελίδες του WWW. Αυτές εκτελούνε αναδρομικά την εξής διαδικασία: a) Κατέβασε την σελίδα X, b) Βρες όλους του επόμενους συνδέσμους (links) από την X, c) Για κάθε σύνδεσμο εκτέλεσε τα βήματα a-c.
2. **Προγράμματα ανταλλαγής αρχείων** (P2P File-sharing Systems, π.χ. Gnutella), χρησιμοποιούνε αλγόριθμους διάσχισης για να μπορούνε οι χρήστες να αποστέλλουν την αναζήτηση τους (query) στους κόμβους του δικτύου. Αν κάποιος κόμβος έχει κάποιο αρχείο το οποίο ψάχνουμε τότε στέλνει κάποια απάντηση στο query.

# Διάσχιση Γράφων

- Οι πιο διαδεδομένοι τύποι διάσχισης είναι:
  1. Κατά Βάθος Διάσχιση(Depth-First Search (DFS))
  2. Κατά Πλάτος Διάσχιση (Breadth-First Search (BFS))

## Depth-First Search

- Γενίκευση της **προθεματικής διάσχισης δένδρων**: Ξεκινώντας από ένα κόμβο v, επισκεπτόμαστε πρώτα τον v και ύστερα καλούμε αναδρομικά τη διαδικασία στο καθένα από τα παιδιά του. Δηλαδή:

```
Print_Preorder(treenode u)
    Output data at u;
    for each child v of u
        Print_Preorder(v)
```

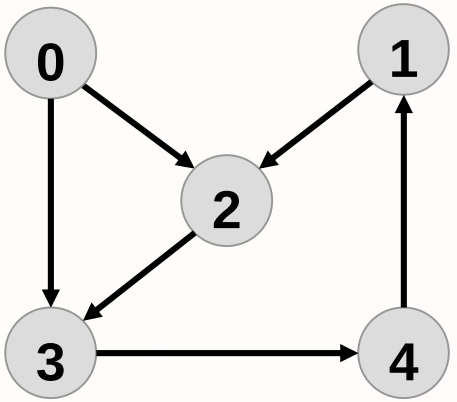
- Σε ένα δένδρο δεν έχουμε κυκλικές διαδρομές. Σε ένα γράφο όμως έχουμε κυκλικές διαδρομές. **Άρα πως θα διασχίσουμε τον γράφο;**

- Θα διατηρήσουμε ένα πίνακα `Visited` ο οποίος θα κρατά πληροφορίες ως προς το ποιους κόμβους έχουμε επισκεφθεί ανά πάσα στιγμή. Δεν θα επισκεπτόμαστε ξανά τον ίδιο κόμβο

# Διαδικασία Προθεματικής Διάσχισης

```
DepthFirstSearch(graph G,vertex v){
   for each vertex w in G
      visited[w] = False;
   DFS(v);
}

DFS(vertex v){
   visited[v] = True;
   print "Visited node v"
   for each w adjacent to v
      if (visited[w]==False)
         DFS(w)
}
```

**Γράφος**




**Χρόνος Εκτέλεσης:**
**Θ(|V| + |E|)**

Θα επισκεφτούμε ακριβώς μια φορά όλες τις κορυφές και ακμές

Υπόθεση: Ο γράφος είναι συνεκτικός (connected).

# Προθεματική Διάσχιση

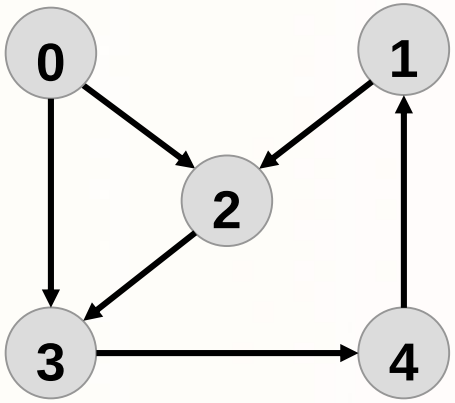
- Αν ο γράφος **δεν είναι συνεκτικός** (disconnected) αυτή η στρατηγική πιθανό να αγνοήσει μερικούς κόμβους. Αν ο στόχος μας είναι να επισκεφθούμε όλους τους κόμβους τότε μετά το τέλος της εκτέλεσης του DFS(v) θα πρέπει να ελέγξουμε τον πίνακα Visited να βρούμε τους κόμβους που δεν έχουμε επισκεφθεί και να καλέσουμε σε αυτούς τη διαδικασία DFS:

```
DepthFirstSearch(graph G,vertex v){
  for each vertex w in G
      Visited[w] = False;
  DFS(v);
  for each vertex w in G
      if (Visited[w]== False)
         DFS(w);
}
```

# Παράδειγμα Depth-First-Search

- σε **Κατευθυνόμενο Γράφο**

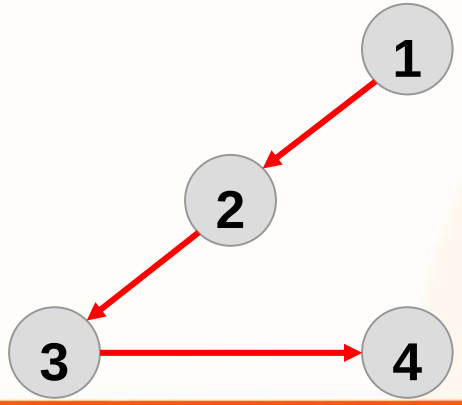
**Γράφος**




**Πίνακας Γειτνίασης (Adjacency Matrix)**

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | 1 | 1 | |
| 1 | | | 1 | | |
| 2 | | | | 1 | |
| 3 | | | | | 1 |
| 4 | | 1 | | | |

- DFS(G, 1) = 1,2,3,4





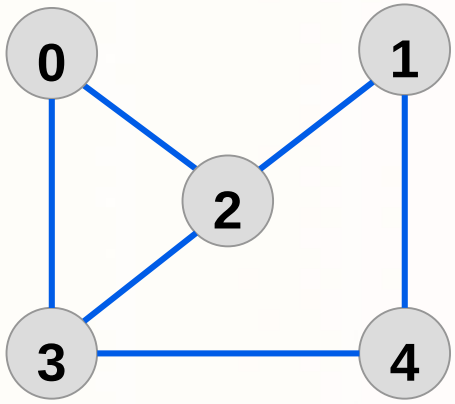
**Στον αρχικό γράφο δεν υπάρχει μια διαδρομή μεταξύ όλων των ζευγών**

**Π.χ. 1->0**

# Παράδειγμα Depth-First-Search

- σε **Μη-Κατευθυνόμενο Γράφο**

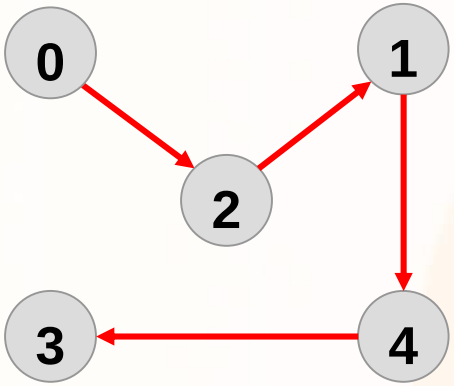
**Γράφος**




**Πίνακας Γειτνίασης (Adjacency Matrix)**

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | 1 | 1 | |
| 1 | | | 1 | | 1 |
| 2 | 1 | 1 | | 1 | |
| 3 | 1 | | 1 | | 1 |
| 4 | | 1 | | 1 | |

- DFS(G, 0) = 0,2,1,4,3



# Μερικά Σχόλια

- Η διαδικασία καλείται σε κάθε κόμβο το πολύ μια φορά.
- Χρόνος Εκτέλεσης: Θ(|V| + |E|), δηλαδή γραμμικός ως προς τον αριθμό των ακμών και κορυφών.
- ***Αρίθμηση DFS*** των κορυφών ενός γράφου ονομάζεται η σειρά με την οποία επισκέπτεται η διαδικασία DepthFirstSearch τις κορυφές του γράφου.
- Η διαδικασία μπορεί να κληθεί και για μη-κατευθυνόμενους και για κατευθυνόμενους γράφους.
- Αντί με αναδρομή η διαδικασία μπορεί να υλοποιηθεί, ως συνήθως, με τη χρήση στοιβών.

# Διαδικασία Depth-First Search με στοίβα

Διάσχιση Γράφου G ξεκινώντας από το v

**Ψευδοκώδικας**

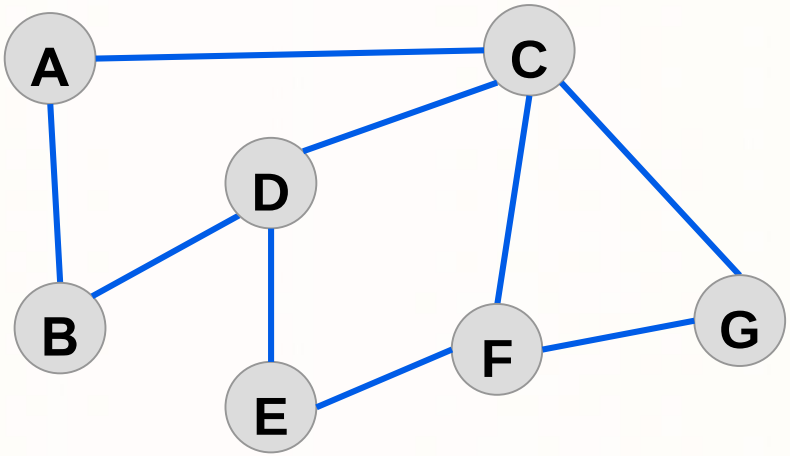
```
DFSearch(graph G, vertex v){
   S = new Stack();
   for each w in G
       Visited[w]=False;

   Visited[v]= True;
   S.push(v);

   while (!S.isEmpty()){
       w = S.pop();
       Print(w);
       for each u adjacent to w
             if (Visited[u]==False){
                   Visited[u]=True;
                   S.push(u);
             }
    }
}
```

**Υλοποίηση DFS με χρήση στοίβας**

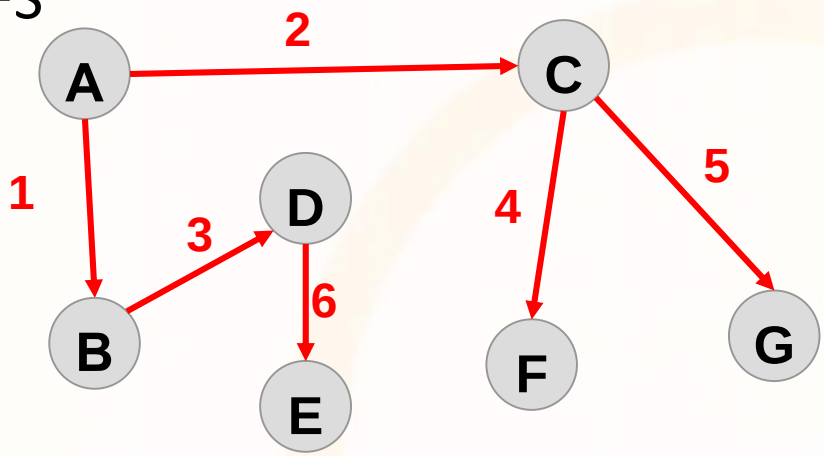
**Χρόνος Εκτέλεσης: O(|V| + |E|)**

# Breadth-First Search (BFS)

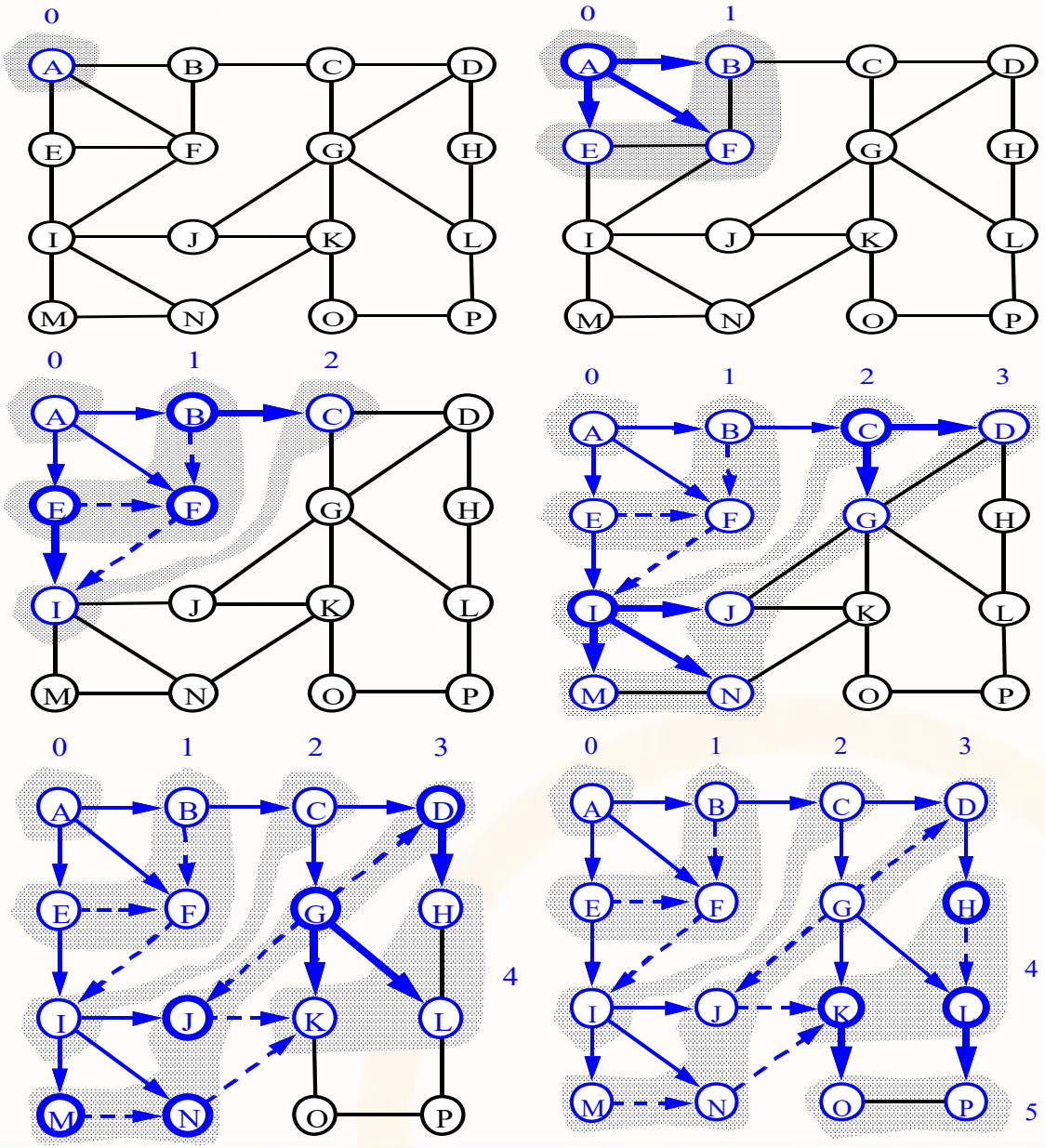
- Ξεκινώντας από ένα κόμβο v, επισκεπτόμαστε πρώτα το v, ύστερα τους κόμβους που γειτνιάζουν με τον v, ύστερα τους κόμβους που βρίσκονται σε απόσταση 2 από τον v, και ούτω καθεξής.




- Διάσχιση με BFS



# BFS: Παράδειγμα



# Διαδικασία Breadth-First Search

Διάσχιση Γράφου G ξεκινώντας από το v

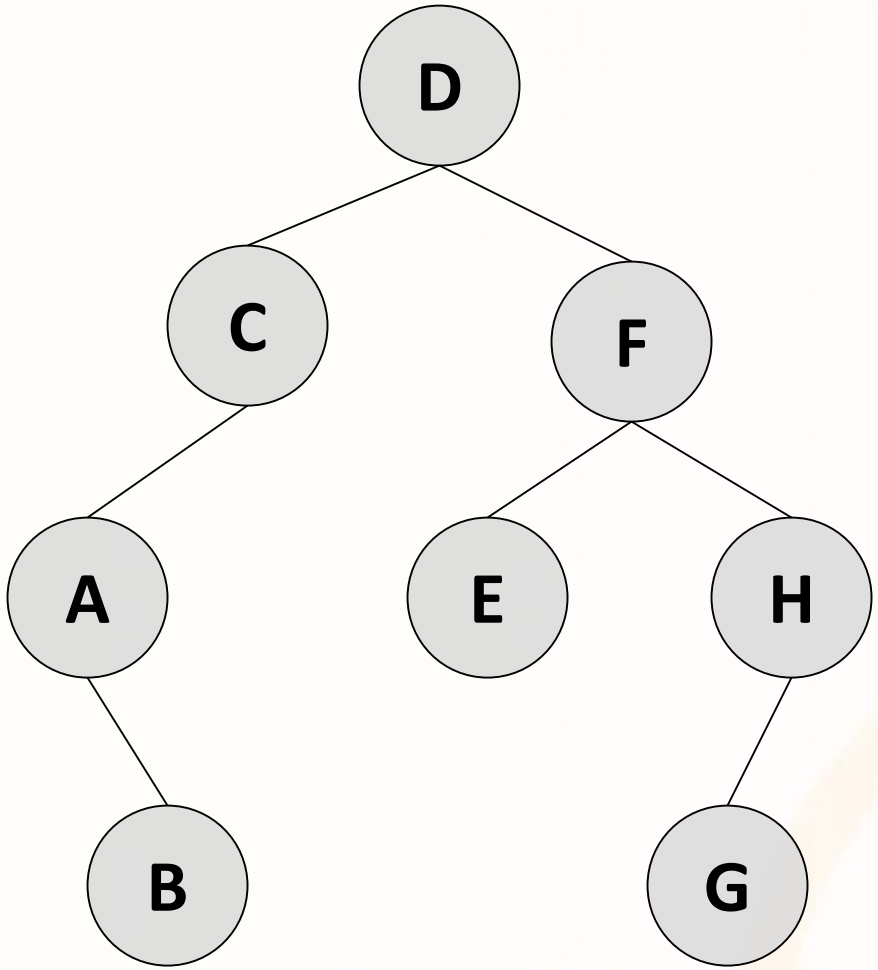
**Ψευδοκώδικας**

```
BFSearch(graph G, vertex v){
   Q = new Queue();
   for each w in G
       Visited[w]=False;

   Visited[v]= True;
   Q.enqueue(v);

   while (!Q.isEmpty()){
       w = Q.dequeue();
       Print(w);
       for each u adjacent to w
             if (Visited[u]==False){
                   Visited[u]=True;
                   Q.enqueue(u);
             }
    }
}
```

**Υλοποίηση BFS με χρήση ουράς**

**Χρόνος Εκτέλεσης: O(|V| + |E|)**

# Παράδειγμα Εκτέλεσης BFS




| Ουρά Q | Έξοδος Διαδικασίας |
|---|---|
| D | D |
| C,F | C |
| F, A | F |
| A, E, H | A |
| E, H, B | E |
| H, B | H |
| B, G | B |
| G | G |
| { } | |

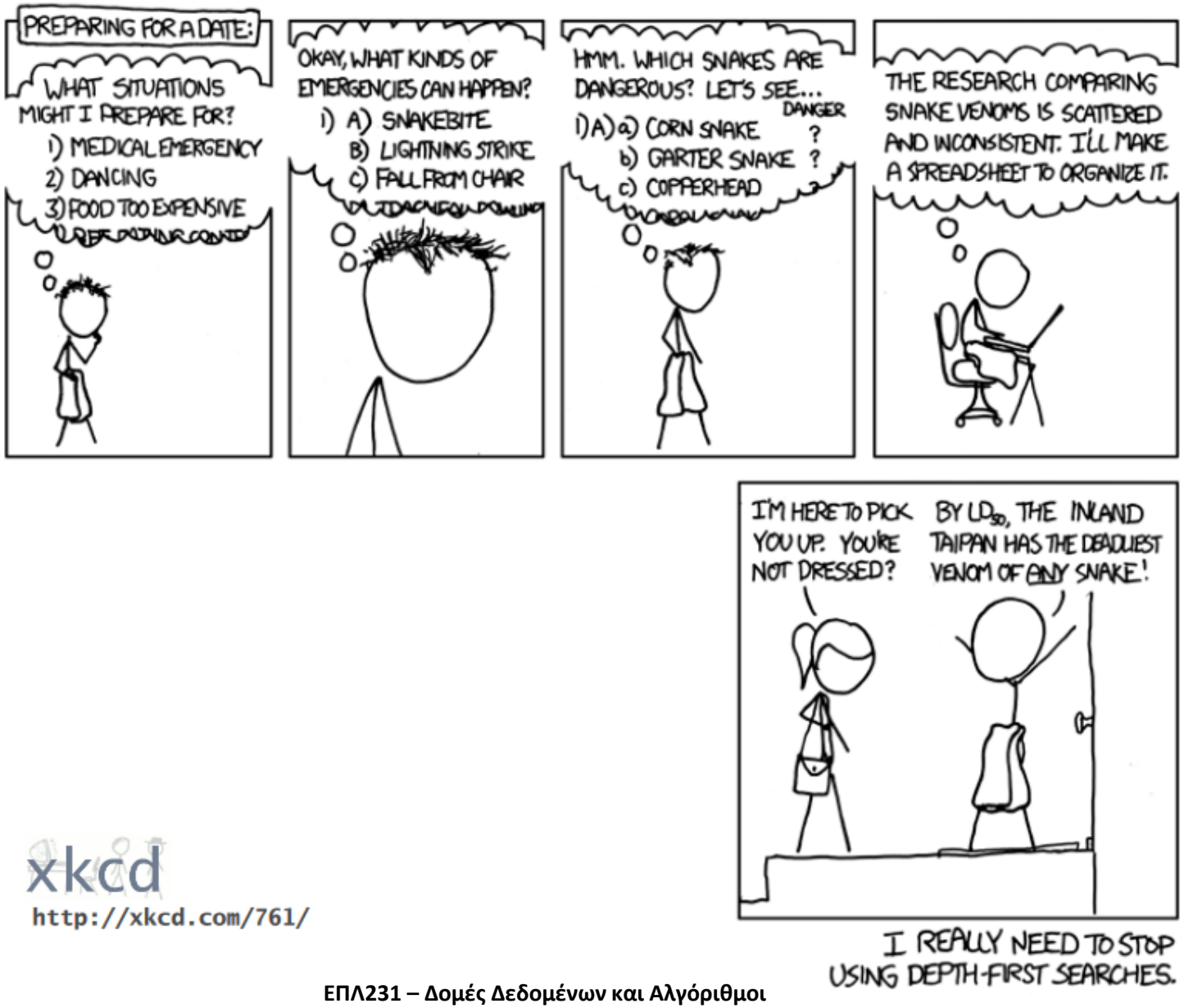
# DFS vs BFS implementation

```
DFSearch(graph G, vertex v){
  S = new Stack();
  for each w in G
     visited[w]=False;
  visited[v]= True;
  S.push(v);

  while (!S.isEmpty()){
     w = S.pop();
     print(w);
     for each u adjacent to w
       if (visited[u]==False){
         visited[u]=True;
         S.push(u);
       }
   }
}
```

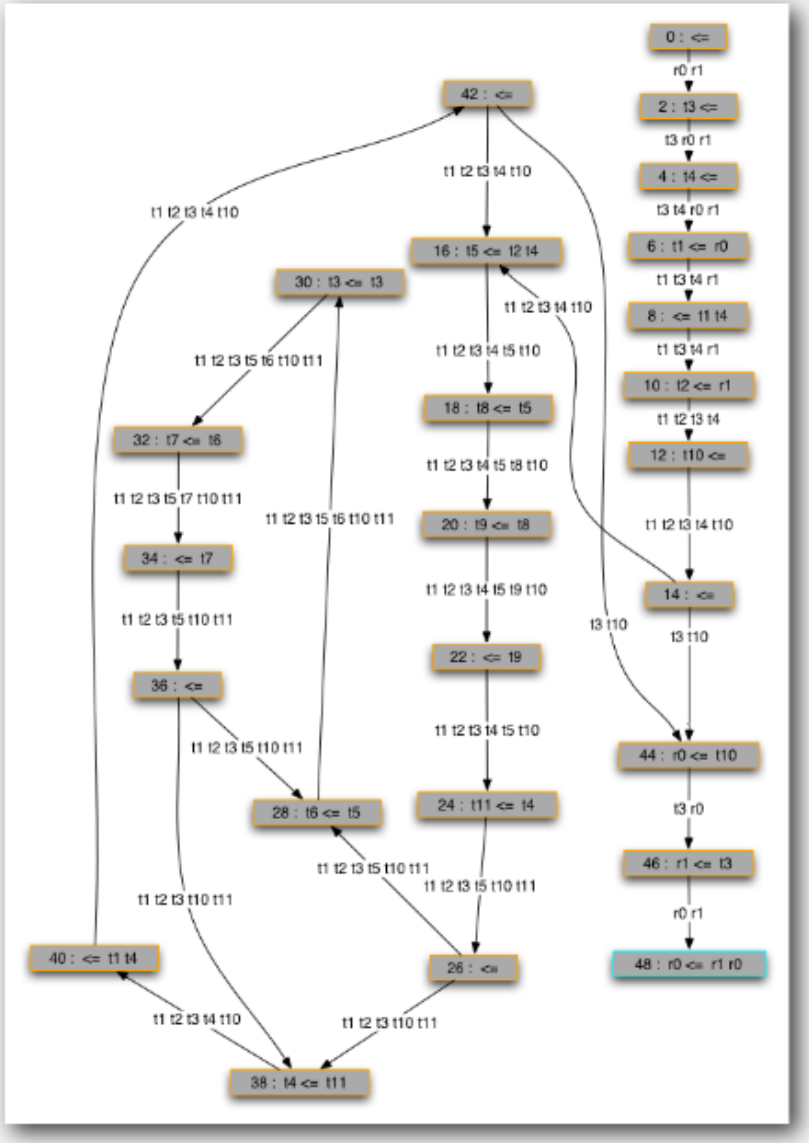
```
BFSearch(graph G, vertex v){
  Q = new Queue();
  for each w in G
     visited[w]=False;
  visited[v]= True;
  Q.enqueue(v);

  while (!Q.isEmpty()){
     w = Q.dequeue();
     print(w);
     for each u adjacent to w
       if (visited[u]==False){
         visited[u]=True;
         Q.enqueue(u);
       }
   }
}
```

# ΕΦΑΡΜΟΓΕΣ DFS + BFS

# Depth-first search application: preparing for a date



# Control Flow Analysis of Programs



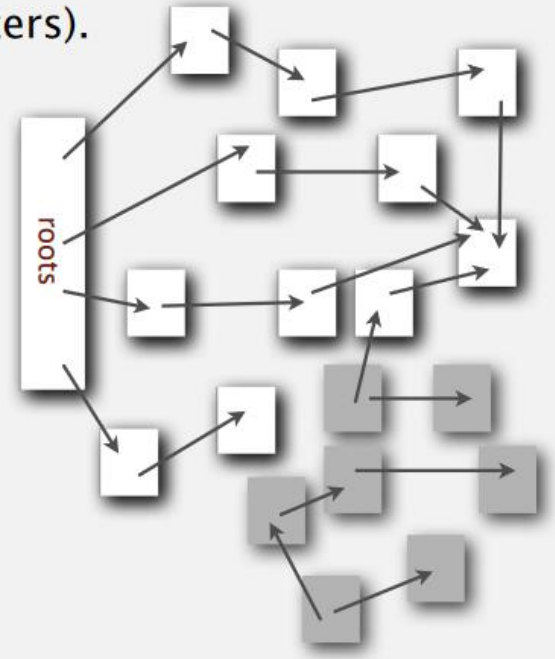
# Reachability application: mark-sweep garbage collector

Every data structure is a digraph.

- Vertex = object.
- Edge = reference.

Roots. Objects known to be directly accessible by program (e.g., stack).

Reachable objects. Objects indirectly accessible by program (starting at a root and following a chain of pointers).



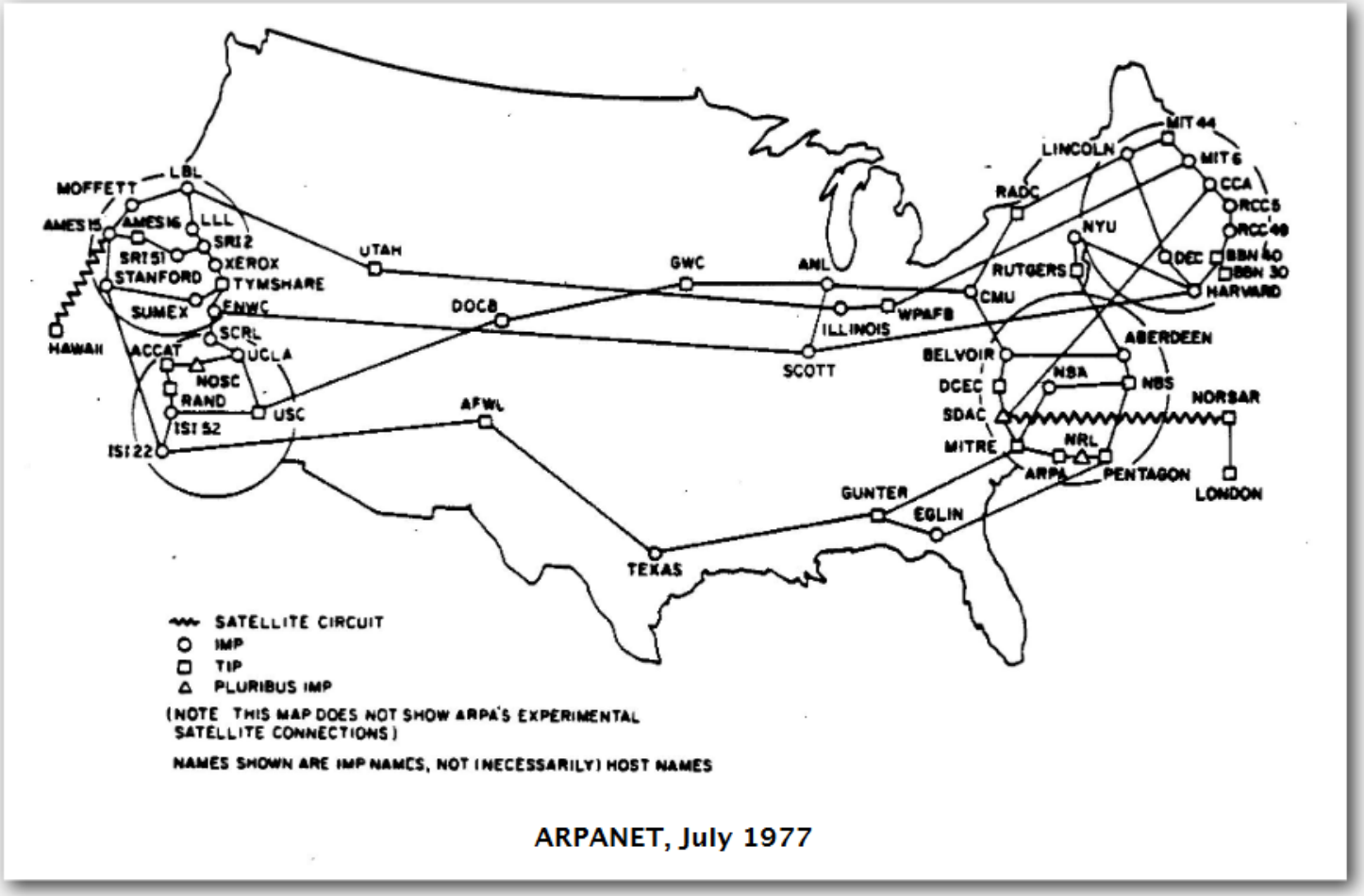

Mark-sweep algorithm. [McCarthy, 1960]

- Mark: mark all reachable objects.
- Sweep: if object is unmarked, it is garbage (so add to free list).

Memory cost. Uses 1 extra mark bit per object (plus DFS stack).

# Breadth-first search application: routing

Fewest number of hops in a communication network.



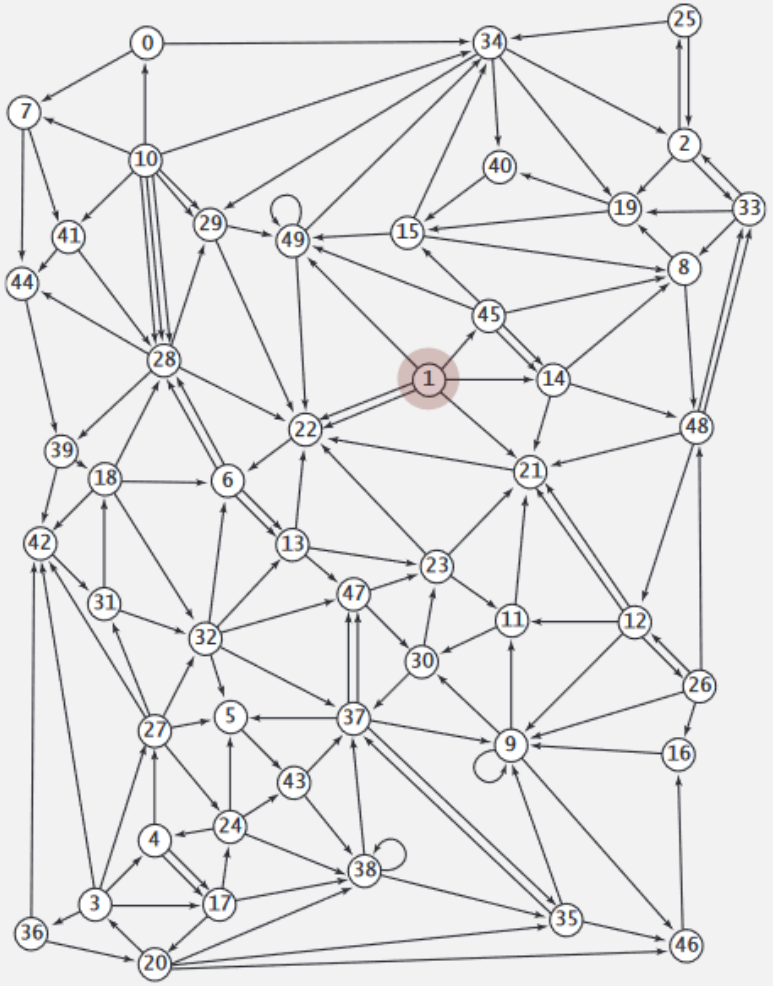

ARPANET, July 1977

# Breadth-first search in digraphs application: web crawler

Goal. Crawl web, starting from some root web page, say `www.princeton.edu`.

Solution. [BFS with implicit digraph]

- Choose root web page as source $s$.
- Maintain a `Queue` of websites to explore.
- Maintain a `SET` of discovered websites.
- Dequeue the next website and enqueue websites to which it links (provided you haven't done so before).




Q. Why not use DFS?

# Bare-bones web crawler: Java implementation

```
Queue<String> queue = new Queue<String>();
SET<String> marked = new SET<String>();

String root = "http://www.princeton.edu";
queue.enqueue(root);
marked.add(root);

while (!queue.isEmpty())
{
   String v = queue.dequeue();
   StdOut.println(v);
   In in = new In(v);
   String input = in.readAll();

   String regexp = "http://(\\w+\\.)*(\\w+)";
   Pattern pattern = Pattern.compile(regexp);
   Matcher matcher = pattern.matcher(input);
   while (matcher.find())
   {
      String w = matcher.group();
      if (!marked.contains(w))
      {
         marked.add(w);
         queue.enqueue(w);
      }
   }
}
```

queue of websites to crawl

set of marked websites

start crawling from root website

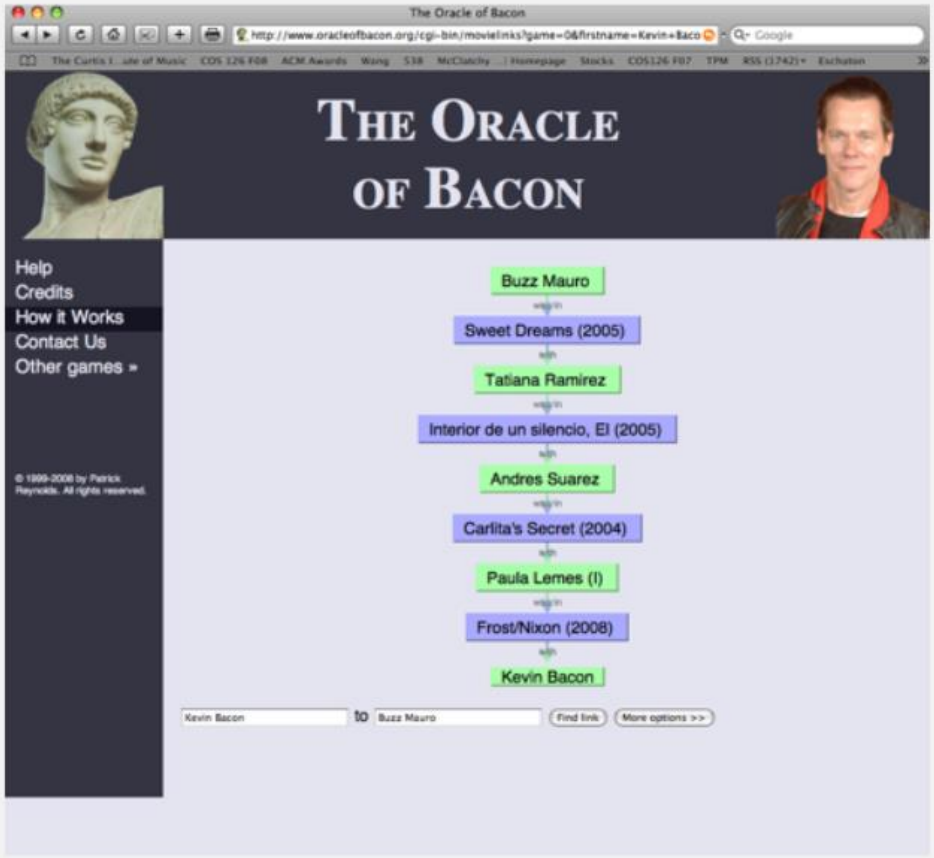
read in raw html from next website in queue

use regular expression to find all URLs in website of form `http://xxx.yyy.zzz` [crude pattern misses relative URLs]

if unmarked, mark it and put on the queue

# Breadth-first search application: Kevin Bacon numbers

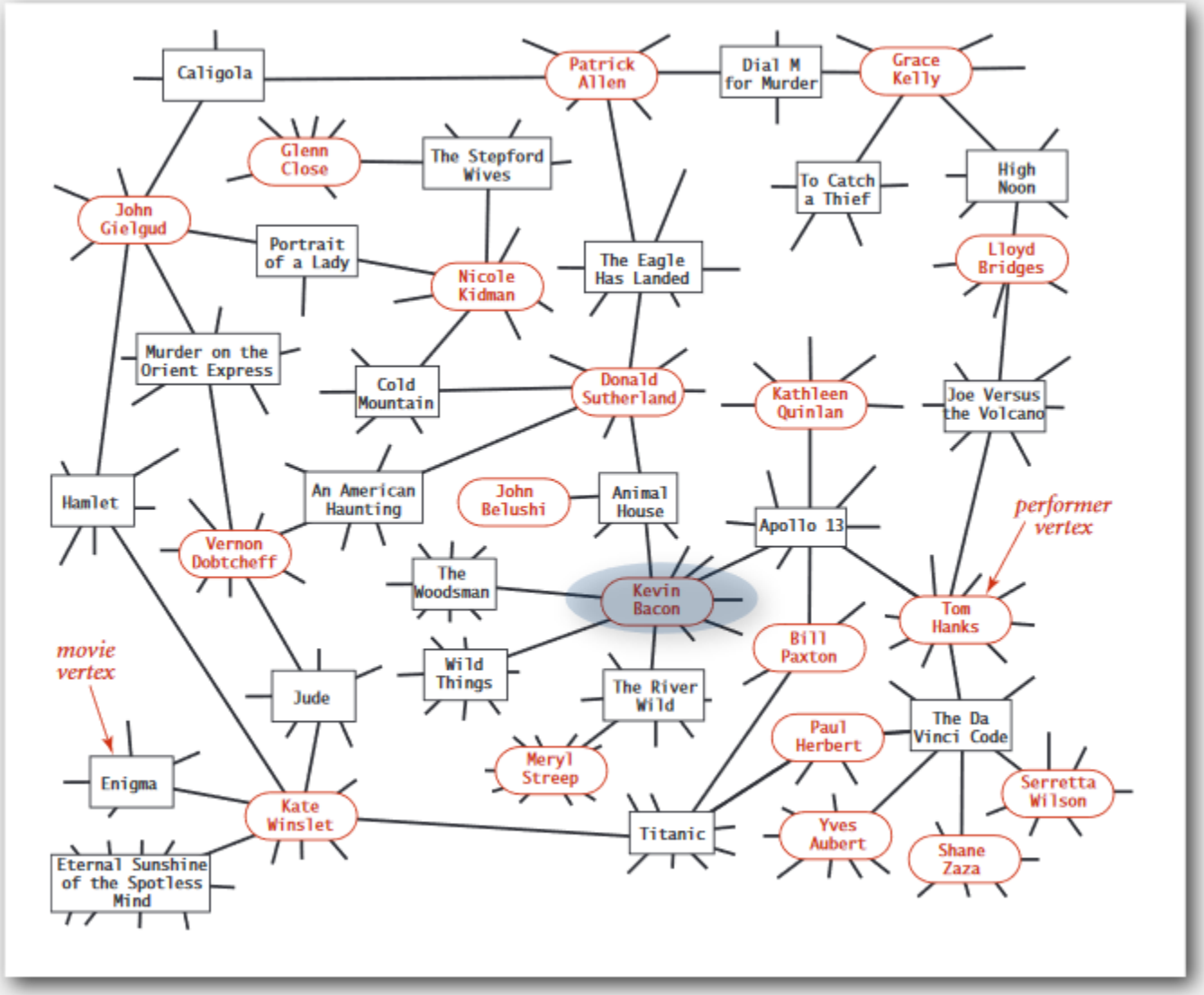
Kevin Bacon numbers.




http://oracleofbacon.org


Endless Games board game


SixDegrees iPhone App


Caligola
Patrick Allen
Dial M for Murder
Grace Kelly
Glenn Close
The Stepford Wives
To Catch a Thief
High Noon
John Gielgud
Portrait of a Lady
Nicole Kidman
The Eagle Has Landed
Lloyd Bridges
Murder on the Orient Express
Cold Mountain
Donald Sutherland
Kathleen Quinlan
Joe Versus the Volcano
Hamlet
An American Haunting
John Belushi
Animal House
Apollo 13
performer vertex
Vernon Dobtcheff
The Woodsman
Kevin Bacon
Tom Hanks
movie vertex
Bill Paxton
Wild Things
The River Wild
Jude
Paul Herbert
The Da Vinci Code
Meryl Streep
Enigma
Serretta Wilson
Kate Winslet
Titanic
Yves Aubert
Shane Zaza
Eternal Sunshine of the Spotless Mind

# Actually, this guy is the center of the Hollywood Universe

The average Jackson Number is about 2.978!!!

# Γράφοι: Ασκήσεις

- Υλοποιήστε τις συναρτήσεις DFS και BFS στην γλώσσα προγραμματισμού JAVA