http://algs4.cs.princeton.edu

4.1 UNDIRECTED GRAPHS

- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

4.1 Undirected Graphs

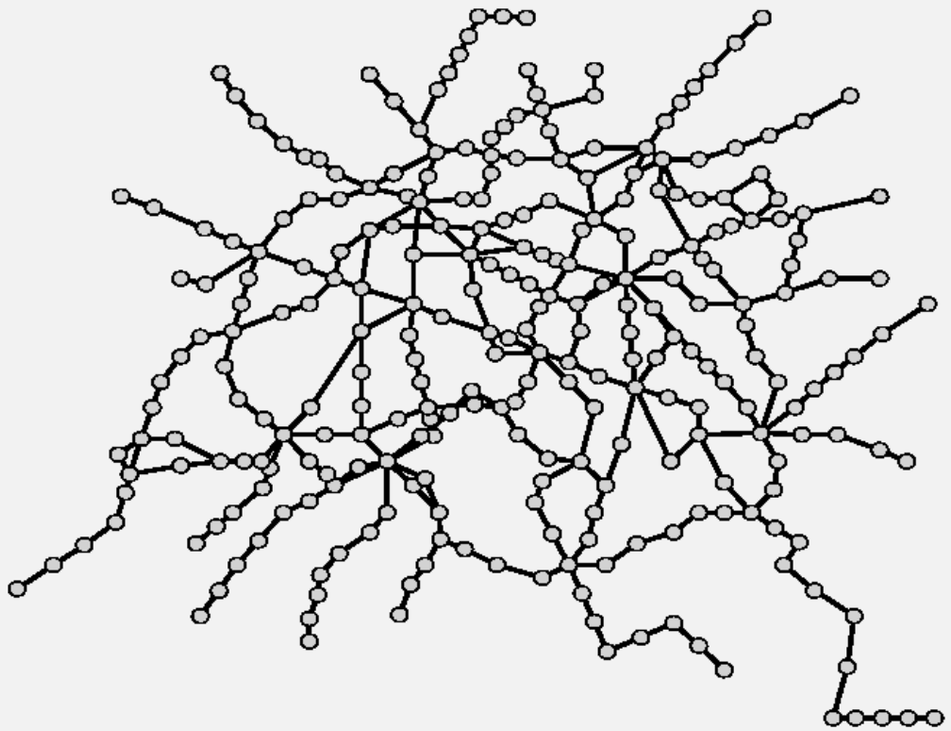
- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

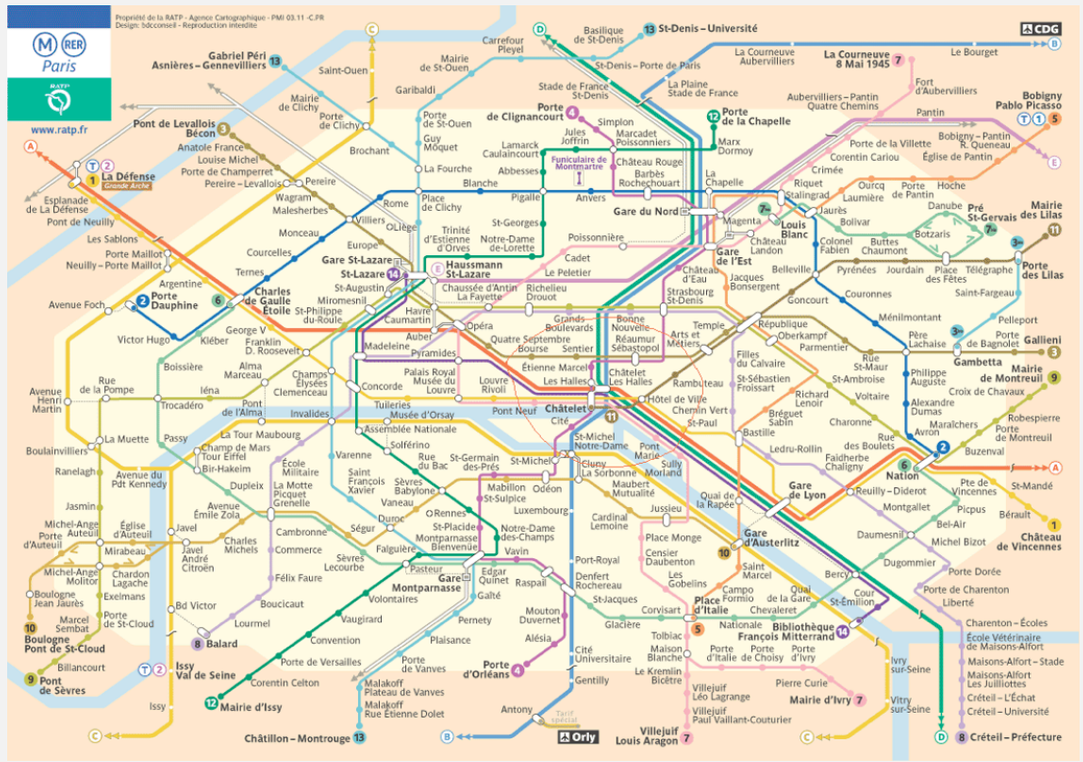
Undirected graphs

Graph. Set of vertices connected pairwise by edges.

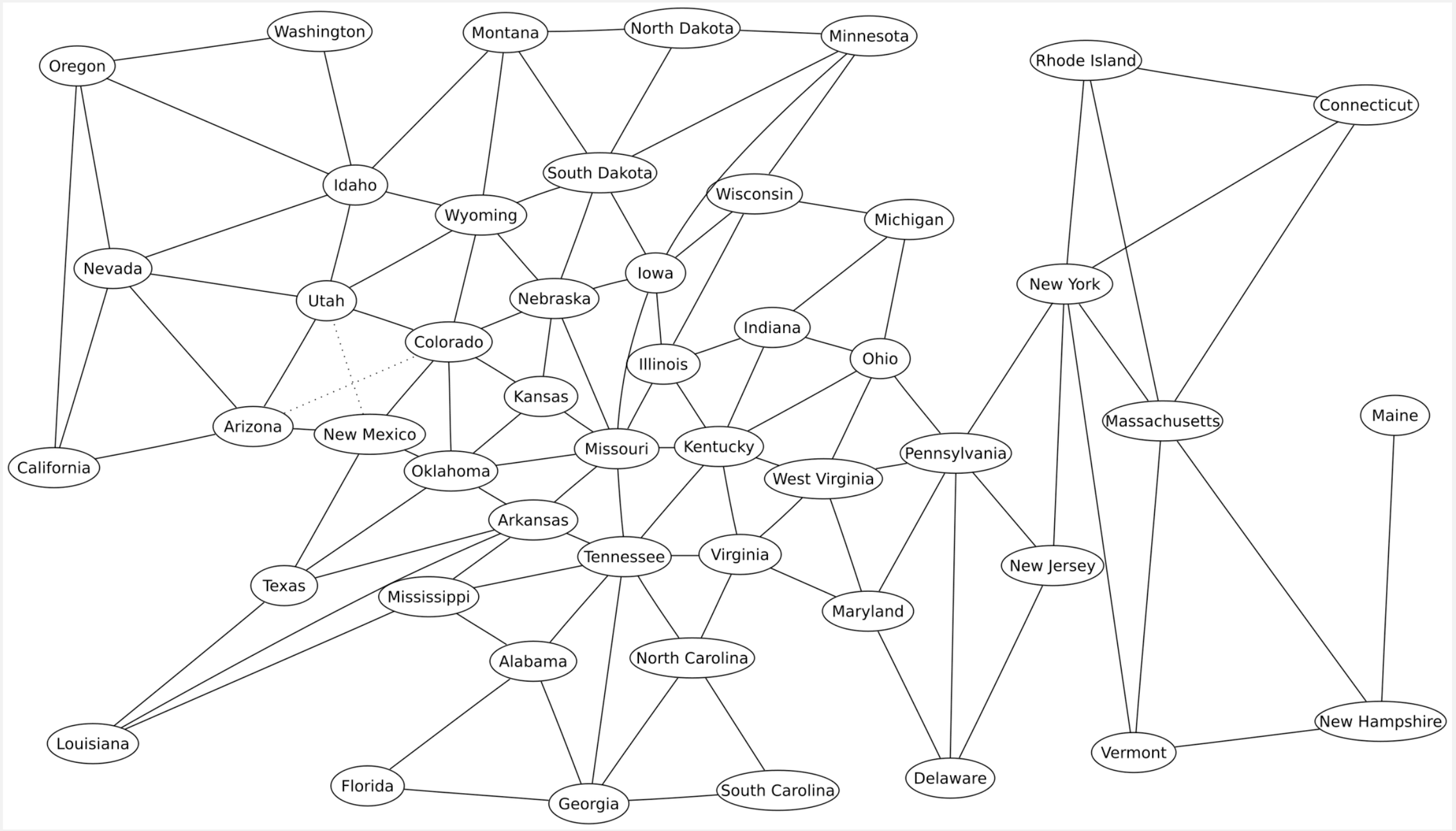
Why study graph algorithms?

- Thousands of practical applications.
- Hundreds of graph algorithms known.
- Interesting and broadly useful abstraction.
- Challenging branch of computer science and discrete math.

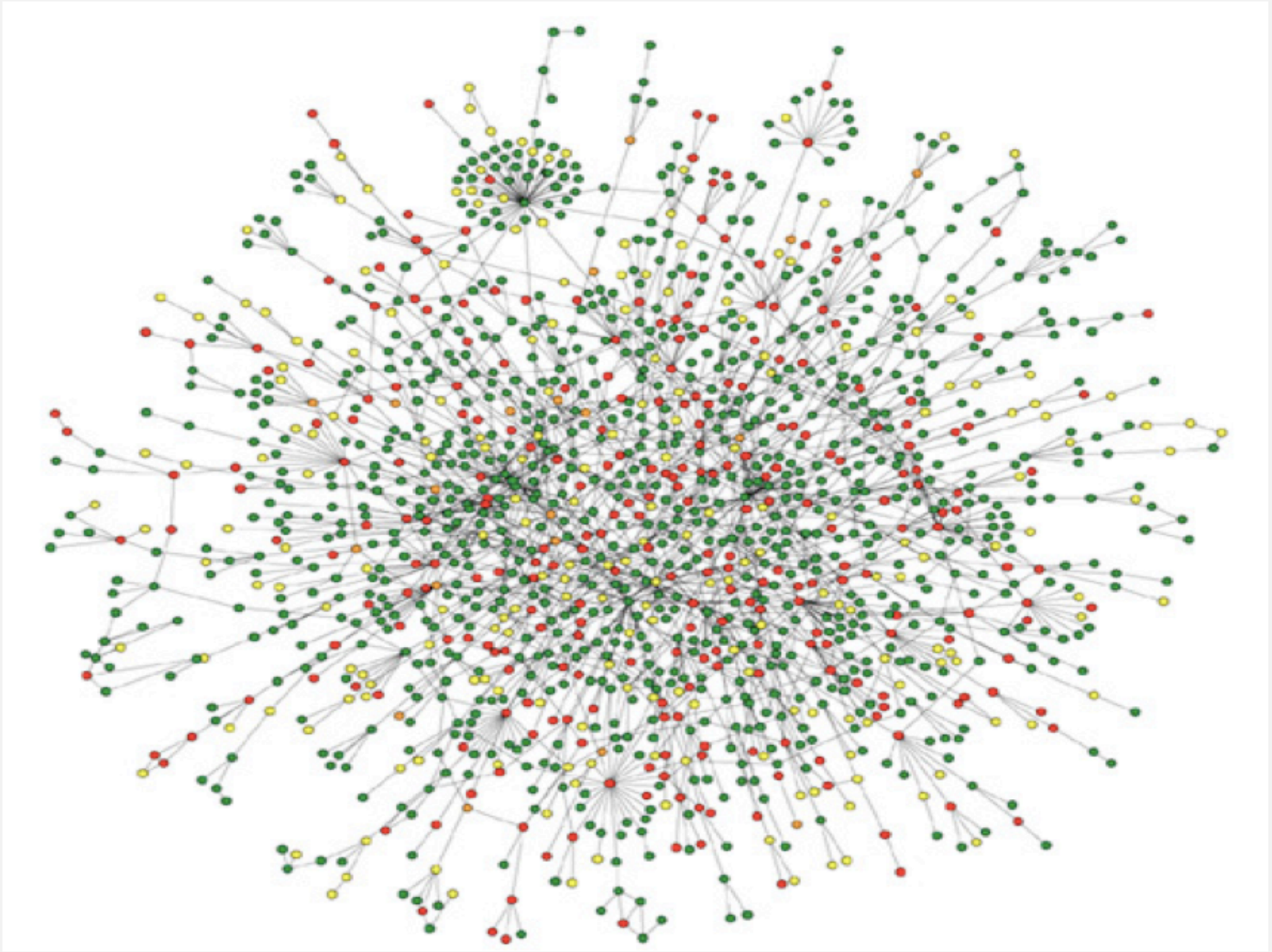



Paris
www.ratp.fr
La Défense
Pont de Levallois Bécon
Gabriel Péri Asnières – Gennevilliers
St-Denis – Université
La Courneuve 8 Mai 1945
Bobigny Pablo Picasso
Porte de Clignancourt
Porte de la Chapelle
Gare du Nord
Gare de l'Est
Gare St-Lazare
Haussmann St-Lazare
Charles de Gaulle Étoile
Porte Dauphine
Châtelet
Gare de Lyon
Gare d'Austerlitz
Nation
Bastille
Opéra
Pyramides
Concorde
Tuileries
Musée d'Orsay
Assemblée Nationale
Invalides
Champs Élysées Clemenceau
Trocadéro
Pont de l'Alma
Tour Eiffel
Bir-Hakeim
Passy
La Muette
Boulainvilliers
Ranelagh
Jasmin
Michel-Ange Auteuil
Porte d'Auteuil
Église d'Auteuil
Mirabeau
Michel-Ange Molitor
Chardon Lagache
Exelmans
Boulogne Jean Jaurès
Marcel Sembat
Porte de St-Cloud
Boulogne Pont de St-Cloud
Billancourt
Pont de Sèvres
Issy Val de Seine
Mairie d'Issy
Balard
Bd Victor
Lourmel
Boucicaut
Félix Faure
Commerce
Cambronne
Sèvres Lecourbe
Dupleix
Avenue Émile Zola
Javel André Citroën
Avenue du Pdt Kennedy
Avenue Foch
Porte Maillot
Neuilly – Porte Maillot
Les Sablons
Pont de Neuilly
Esplanade de La Défense
Courbevoie
Argentine
Victor Hugo
George V
Franklin D. Roosevelt
Alma Marceau
Iéna
Rue de la Pompe
Avenue Henri Martin
Varenne
École Militaire
La Tour Maubourg
Champ de Mars
La Motte Picquet Grenelle
Saint François Xavier
Duroc
Ségur
Falguière
Pasteur
Vaneau
Sèvres Babylone
Rue du Bac
Rennes
Saint-Placide
Montparnasse Bienvenüe
Gare Montparnasse
Volontaires
Vavin
Notre-Dame des-Champs
Raspail
Edgar Quinet
Galté
Pernety
Plaisance
Convention
Porte de Versailles
Corentin Celton
Porte de Vanves
Malakoff Plateau de Vanves
Malakoff Rue Étienne Dolet
Porte d'Orléans
Antony
Châtillon – Montrouge
Orly
Villejuif Louis Aragon
Cité Universitaire
Gentilly
Le Kremlin Bicêtre
Villejuif Léo Lagrange
Villejuif Paul Vaillant-Couturier
Mairie d'Ivry
Pierre Curie
Ivry sur-Seine
Vitry sur-Seine
Porte d'Italie
Porte de Choisy
Porte d'Ivry
Place d'Italie
Campo Formio
Chevaleret
Bibliothèque François Mitterrand
Nationale
Glacière
Tolbiac
Maison Blanche
Corvisart
Saint Marcel
Gare d'Austerlitz
Quai de la Rapée
Cardinal Lemoine
Jussieu
Censier Daubenton
Les Gobelins
Place Monge
Port-Royal
Denfert Rochereau
St-Jacques
Cluny La Sorbonne
Maubert Mutualité
Saint-Michel Notre-Dame
Pont Marie
Sully Morland
Châtelet Les Halles
Cité
Pont Neuf
Étienne Marcel
Les Halles
Rambuteau
Hôtel de Ville
Chemin Vert
St-Paul
Filles du Calvaire
St-Sébastien Froissart
Richard Lenoir
Bréguet Sabin
Ledru-Rollin
Faidherbe Chaligny
Rue des Boulets
St-Ambroise
Voltaire
Charonne
Alexandre Dumas
Maraîchers
Avron
Reuilly – Diderot
Montgallet
Daumesnil
Bel-Air
Michel Bizot
Dugommier
Porte Dorée
Porte de Charenton
Liberté
Charenton – Écoles
École Vétérinaire de Maisons-Alfort
Maisons-Alfort – Stade
Maisons-Alfort Les Juilliottes
Créteil – L'Échat
Créteil – Université
Créteil – Préfecture
Buzenval
Robespierre
Porte de Montreuil
Philippe Auguste
Croix de Chavaux
Mairie de Montreuil
Gambetta
Rue St-Maur
Père Lachaise
Porte de Bagnolet
Gallieni
Pelleport
Saint-Fargeau
Porte des Lilas
Mairie des Lilas
Pré St-Gervais
Télégraphe
Place des Fêtes
Jourdain
Pyrénées
Belleville
Louis Blanc
Château Landon
Colonel Fabien
Buttes Chaumont
Botzaris
Jaurès
Laumière
Riquet
Stalingrad
Ourcq
Porte de Pantin
Hoche
Bobigny – Pantin R. Queneau
Église de Pantin
Porte de la Villette
Corentin Cariou
Crimée
Danube
Aubervilliers – Pantin Quatre Chemins
Fort d'Aubervilliers
Pantin
Le Bourget
La Courneuve Aubervilliers
La Plaine Stade de France
Stade de France St-Denis
Basilique de St-Denis
Carrefour Pleyel
Mairie de St-Ouen
Saint-Ouen
Garibaldi
Mairie de Clichy
Porte de Clichy
Porte de St-Ouen
Guy Môquet
Brochant
Lamarck Caulaincourt
Jules Joffrin
Marcadet Poissonniers
Simplon
Funiculaire de Montmartre
Château Rouge
Barbès Rochechouart
Abbesses
La Fourche
Blanche
Pigalle
Anvers
Place de Clichy
Rome
Wagram
Malesherbes
Villiers
Liège
Monceau
Europe
Courcelles
Ternes
Pereire
Pereire – Levallois
Louise Michel
Porte de Champerret
Anatole France
Trinité d'Estienne d'Orves
St-Georges
Notre-Dame de-Lorette
Poissonnière
Cadet
Le Peletier
Chaussée d'Antin La Fayette
Richelieu Drouot
Grands Boulevards
Bonne Nouvelle
Réaumur Sébastopol
Arts et Métiers
Temple
Strasbourg St-Denis
Château d'Eau
Jacques Bonsergent
Goncourt
Couronnes
Ménilmontant
Oberkampf
Parmentier
République
Magenta
Havre Caumartin
Auber
Madeleine
Miromesnil
St-Augustin
St-Philippe du-Roule
Quatre Septembre
Bourse
Sentier
Palais Royal Musée du Louvre
Louvre Rivoli
Les Halles
Châtelet
Cité
St-Michel Notre-Dame
Cluny La Sorbonne
Maubert Mutualité
Pont Marie
Sully Morland
Saint-Michel
Odéon
Mabillon
St-Sulpice
Luxembourg
Notre-Dame des-Champs
Vavin
Raspail
Edgar Quinet
Galté
Mouton Duvernet
Alésia
Plaisance
Pernety
Vaugirard
Convention

Border graph of 48 contiguous United States

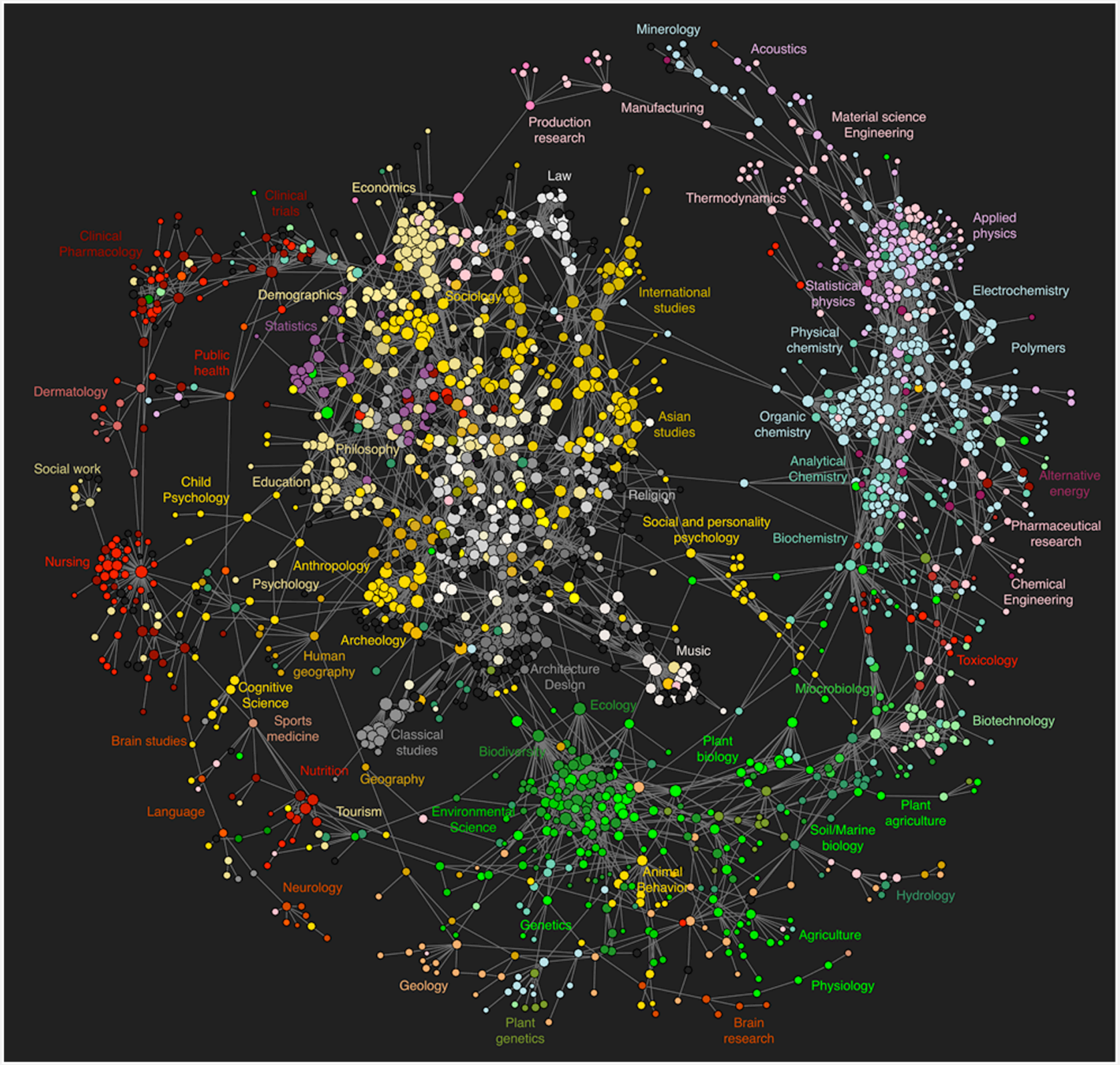


Protein-protein interaction network



Reference: Jeong et al, Nature Review | Genetics

Map of science clickstreams

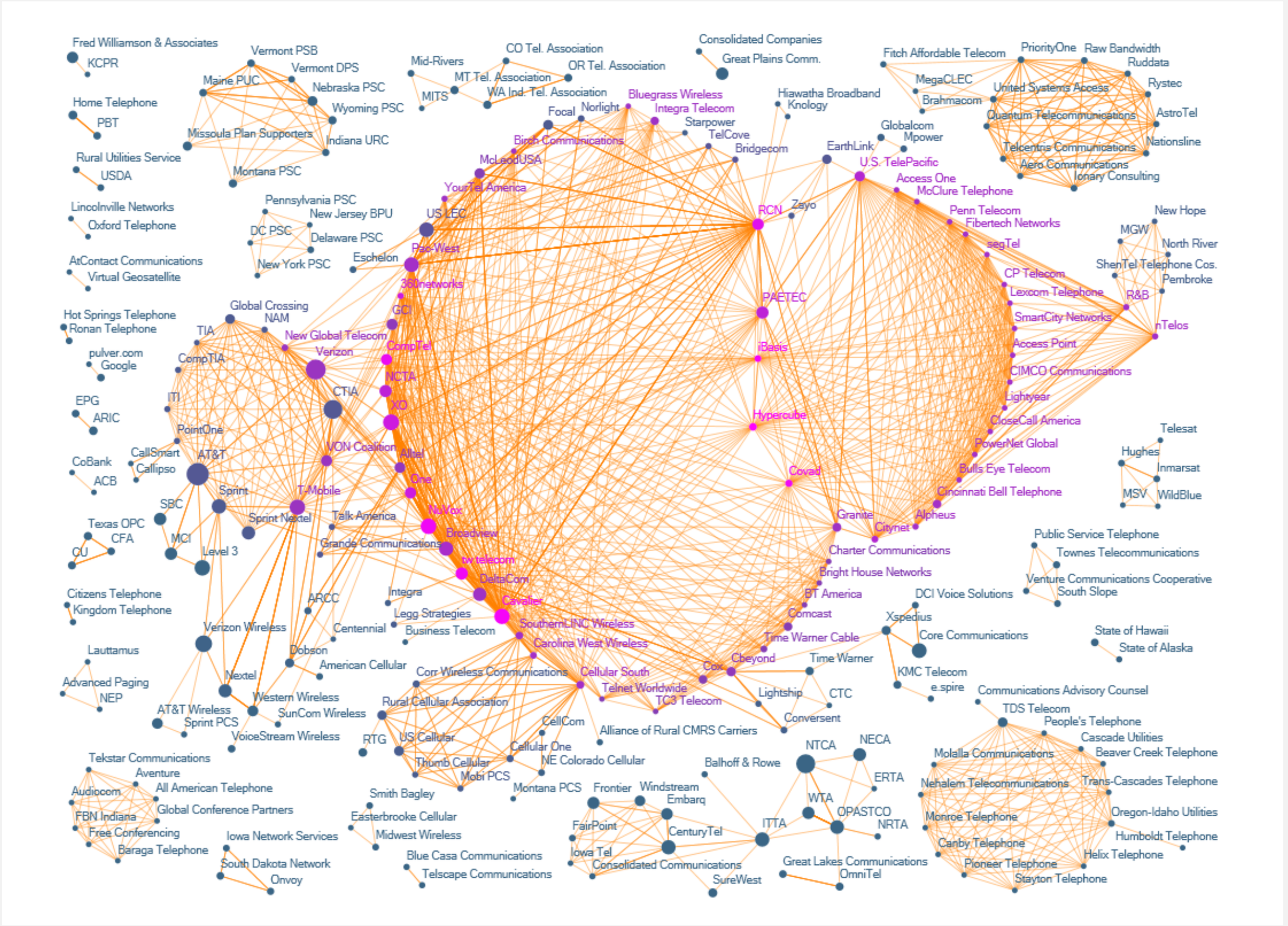


http://www.plosone.org/article/info:doi/10.1371/journal.pone.0004803

10 million Facebook friends

"Visualizing Friendships" by Paul Butler

The evolution of FCC lobbying coalitions



“The Evolution of FCC Lobbying Coalitions” by Pierre de Vries in JoSS Visualization Symposium 2010

Framingham heart study

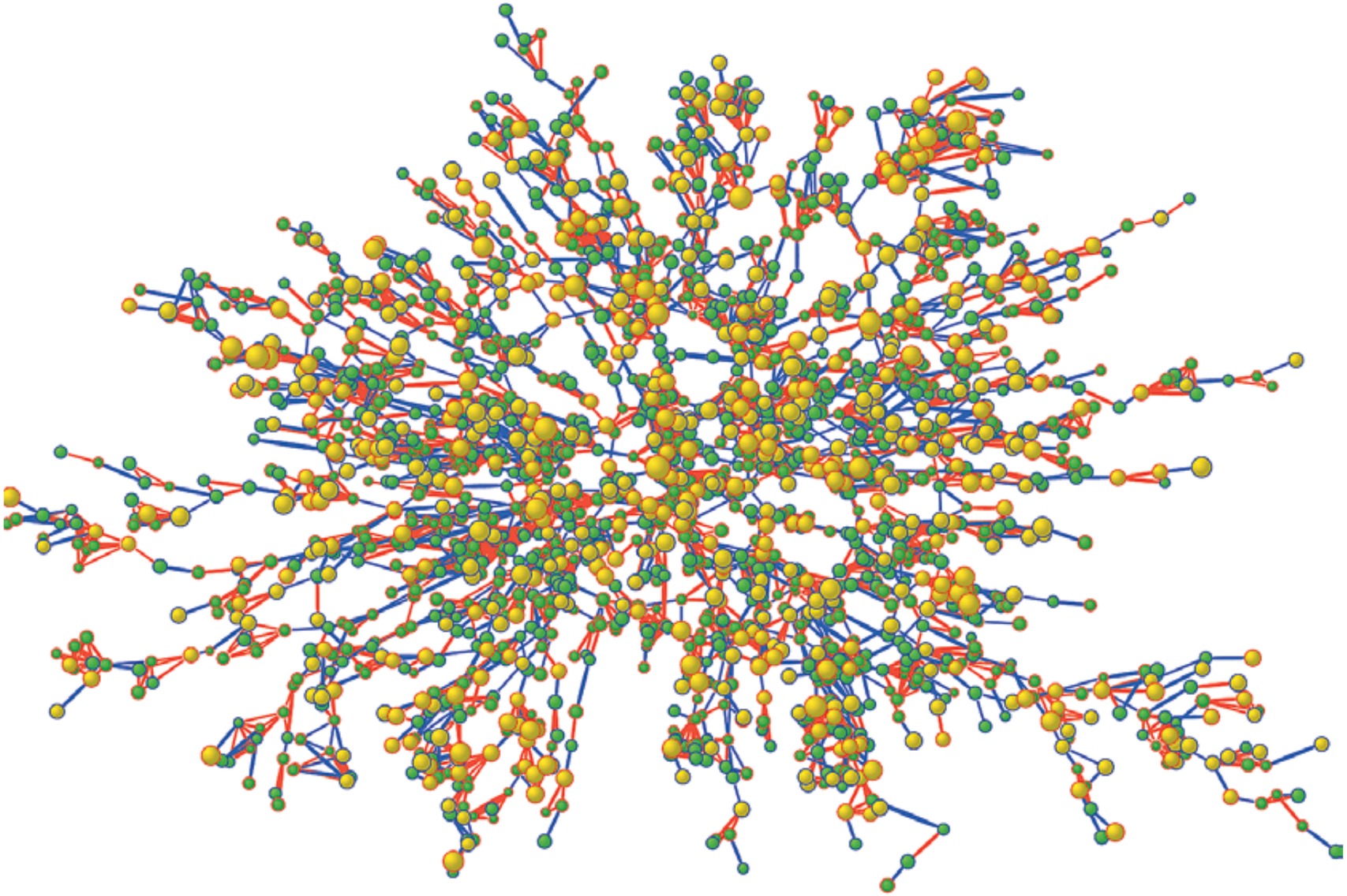
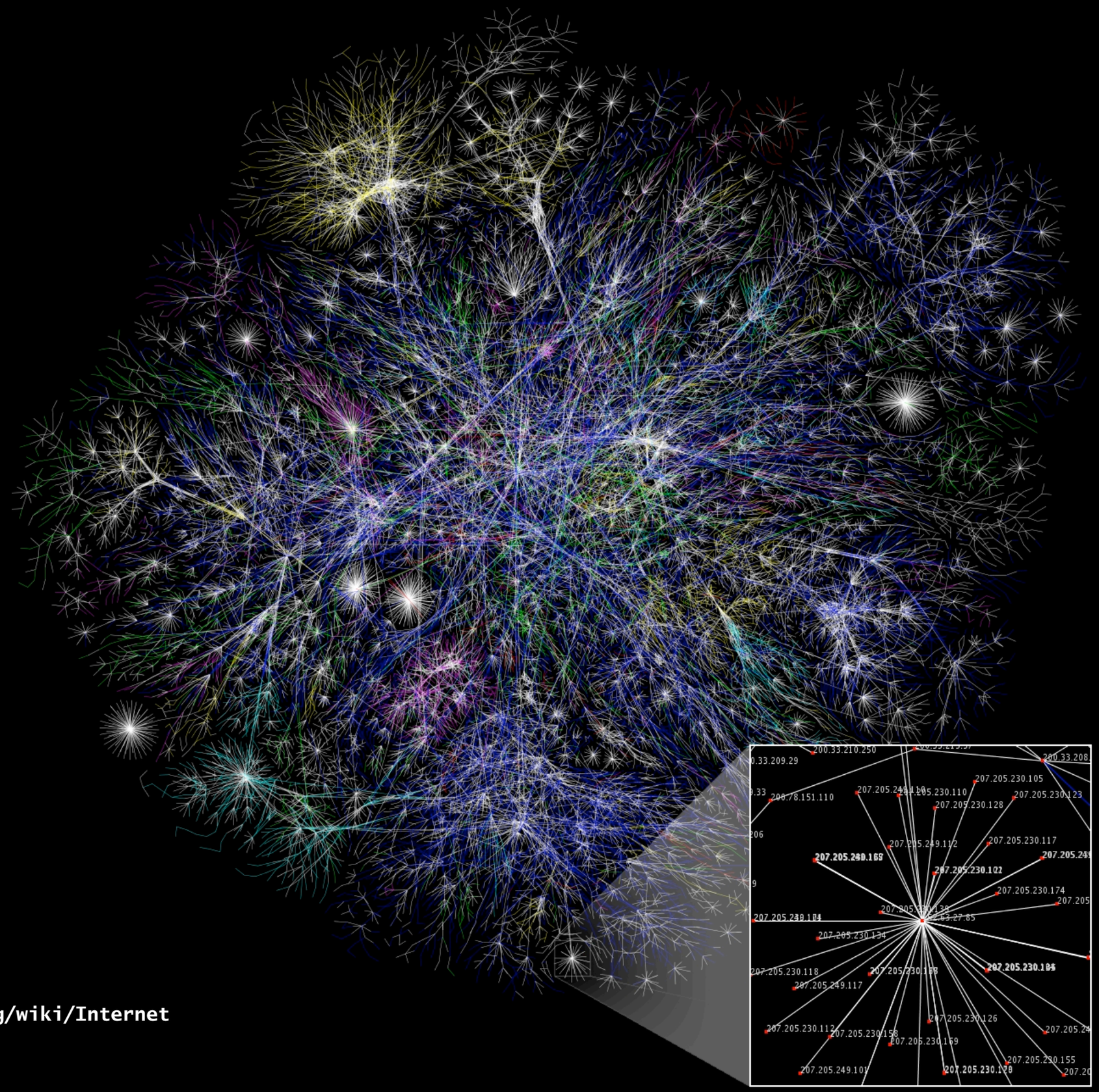


Figure 1. Largest Connected Subcomponent of the Social Network in the Framingham Heart Study in the Year 2000.

Each circle (node) represents one person in the data set. There are 2200 persons in this subcomponent of the social network. Circles with red borders denote women, and circles with blue borders denote men. The size of each circle is proportional to the person's body-mass index. The interior color of the circles indicates the person's obesity status: yellow denotes an obese person (body-mass index, ≥30) and green denotes a nonobese person. The colors of the ties between the nodes indicate the relationship between them: purple denotes a friendship or marital tie and orange denotes a familial tie.

"The Spread of Obesity in a Large Social Network over 32 Years" by Christakis and Fowler in New England Journal of Medicine, 2007

The Internet as mapped by the Opte Project



http://en.wikipedia.org/wiki/Internet

Graph applications

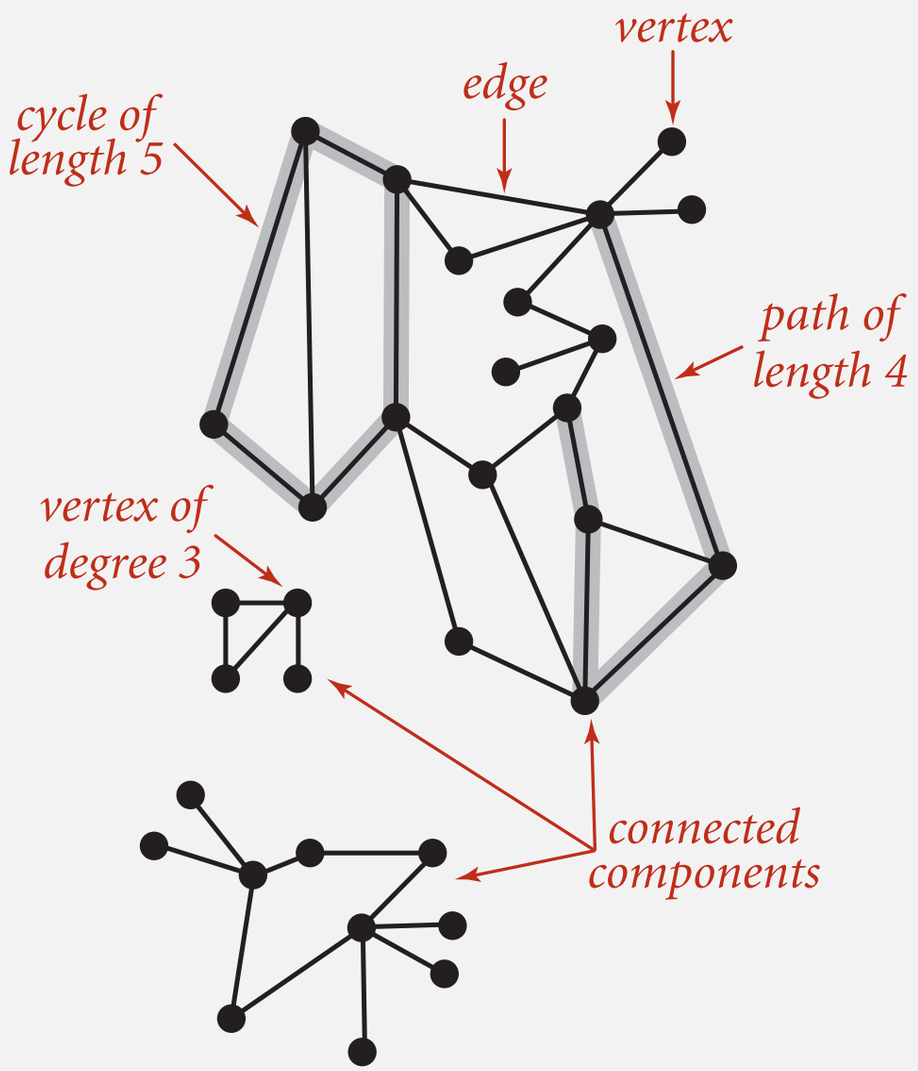
graph	vertex	edge
communication	telephone, computer	fiber optic cable
circuit	gate, register, processor	wire
mechanical	joint	rod, beam, spring
financial	stock, currency	transactions
transportation	intersection	street
internet	class C network	connection
game	board position	legal move
social relationship	person	friendship
neural network	neuron	synapse
protein network	protein	protein-protein interaction
molecule	atom	bond

Graph terminology

Path. Sequence of vertices connected by edges.

Cycle. Path whose first and last vertices are the same.

Two vertices are connected if there is a path between them.



Some graph-processing problems

problem	description
s-t path	*Is there a path between s and t ?*
shortest s-t path	*What is the shortest path between s and t ?*
cycle	*Is there a cycle in the graph ?*
Euler cycle	*Is there a cycle that uses each edge exactly once ?*
Hamilton cycle	*Is there a cycle that uses each vertex exactly once ?*
connectivity	*Is there a way to connect all of the vertices ?*
biconnectivity	*Is there a vertex whose removal disconnects the graph ?*
planarity	*Can the graph be drawn in the plane with no crossing edges ?*
graph isomorphism	*Do two adjacency lists represent the same graph ?*

Challenge. Which graph problems are easy? difficult? intractable?

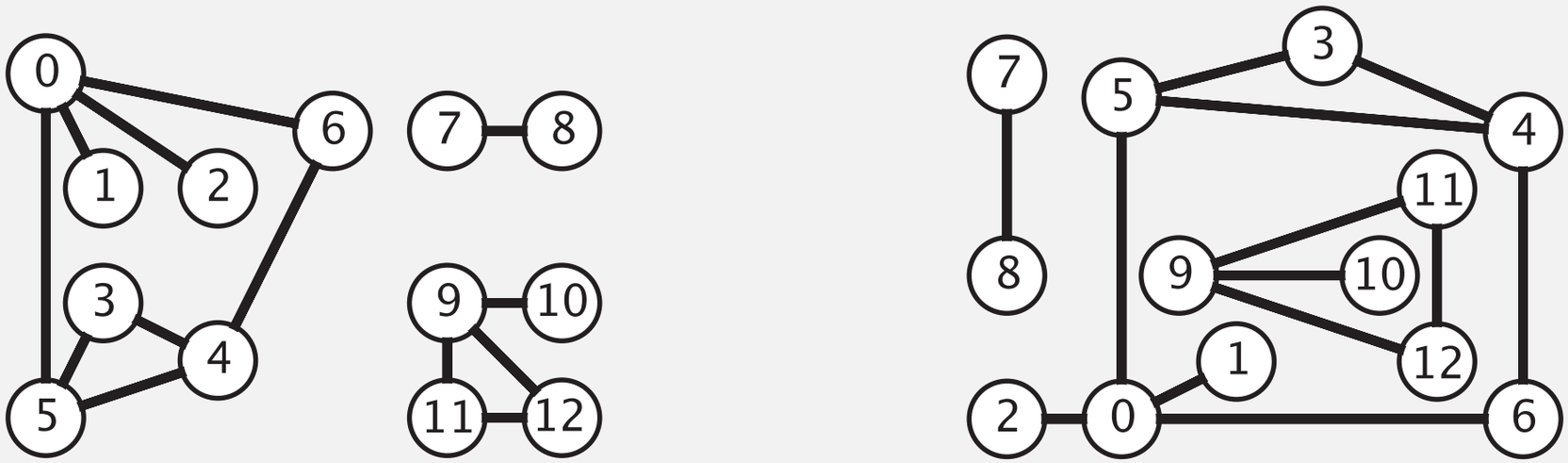
http://algs4.cs.princeton.edu

4.1 Undirected Graphs

- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

Graph representation

Graph drawing. Provides intuition about the structure of the graph.



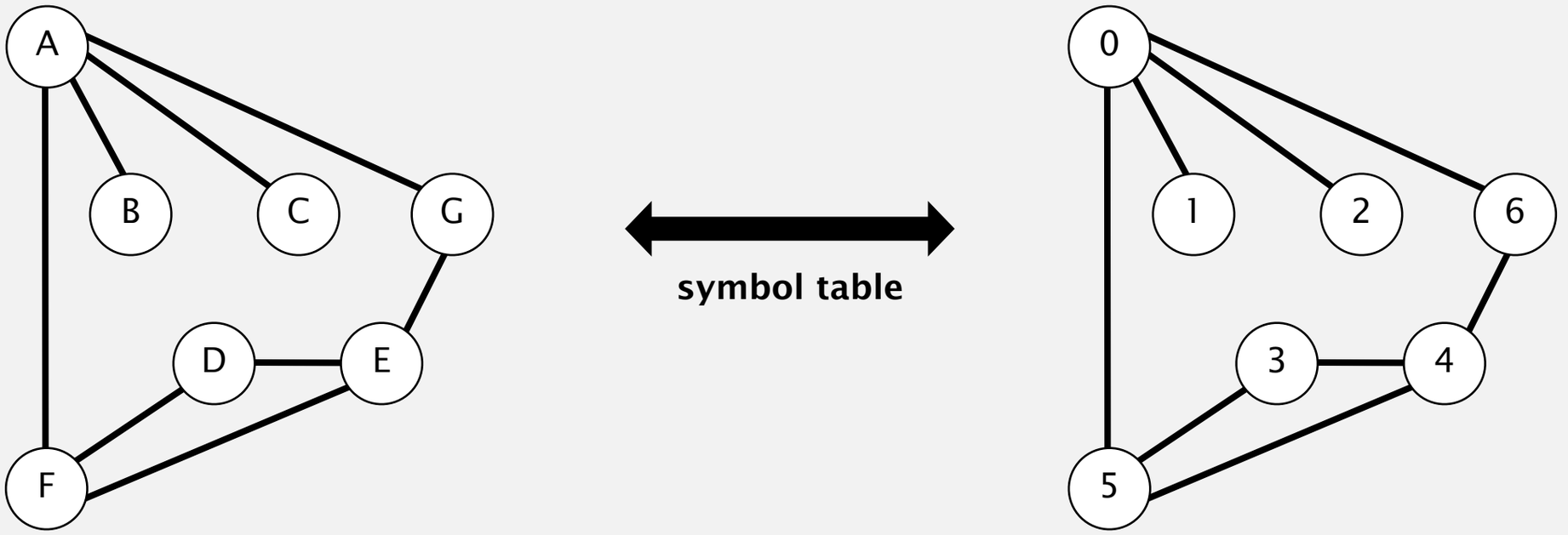
two drawings of the same graph

Caveat. Intuition can be misleading.

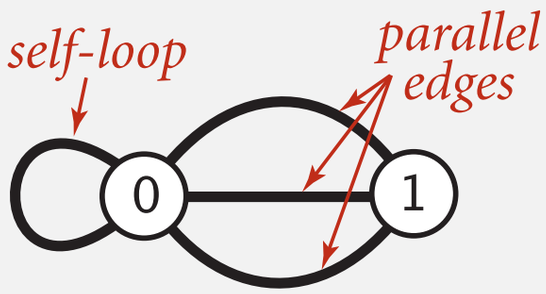
Graph representation

Vertex representation.

- This lecture: use integers between 0 and $V - 1$.
- Applications: convert between names and integers with symbol table.



Anomalies.



Graph 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*

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

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

read graph from input stream

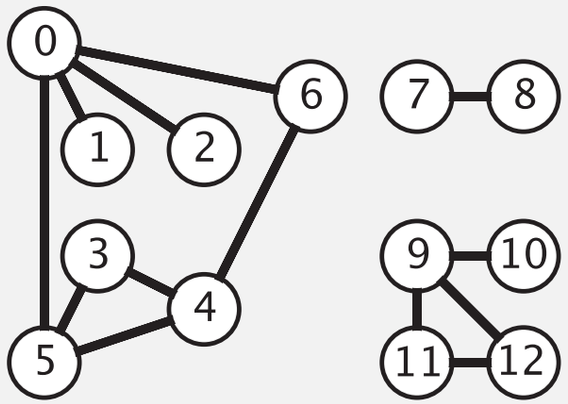
print out each edge (twice)

Graph API: sample client

Graph input format.

tinyG.txt

```
V → 13
    13 ← E
    0 5
    4 3
    0 1
    9 12
    6 4
    5 4
    0 2
    11 12
    9 10
    0 6
    7 8
    9 11
    5 3
```




```
% java Test tinyG.txt
0-6
0-2
0-1
0-5
1-0
2-0
3-5
3-4
⋮
12-11
12-9
```

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

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

read graph from input stream

print out each edge (twice)

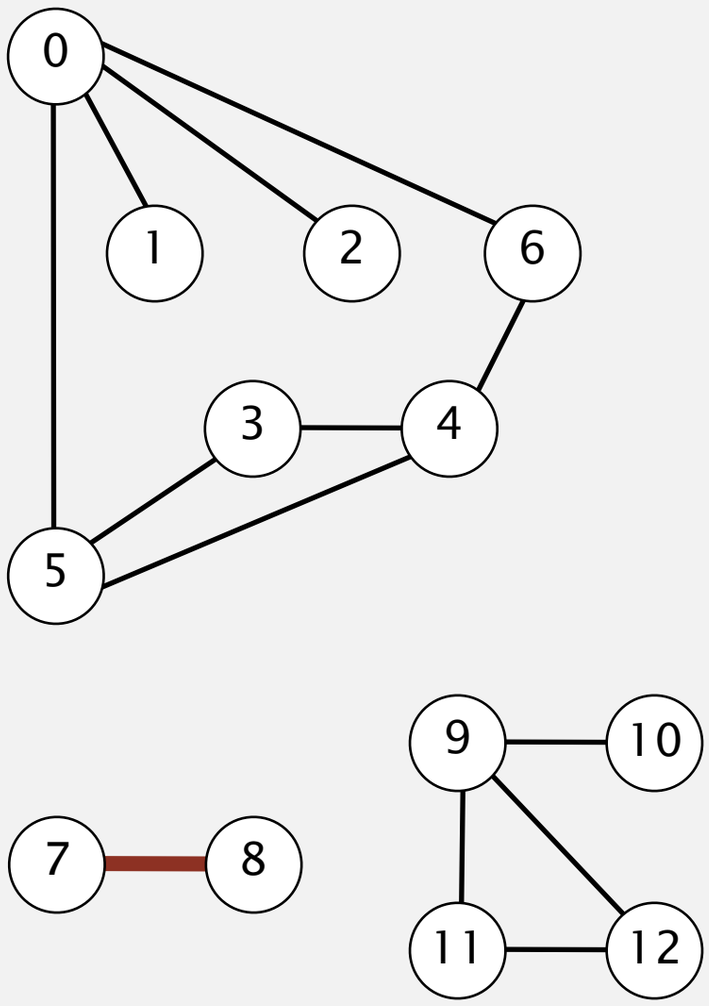
Typical graph-processing code

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*

```
// degree of vertex v in graph G
public static int degree(Graph G, int v)
{
    int degree = 0;
    for (int w : G.adj(v))
       degree++;
    return degree;
}
```

Set-of-edges graph representation

Maintain a list of the edges (linked list or array).



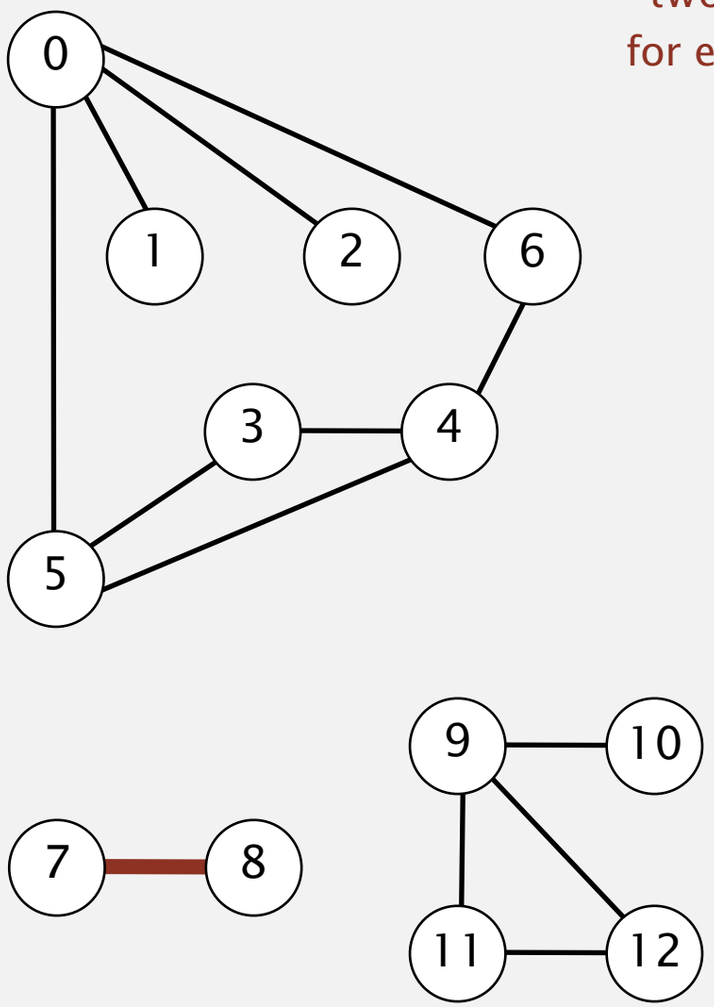
0	1
0	2
0	5
0	6
3	4
3	5
4	5
4	6
7	8
9	10
9	11
9	12
11	12

Q. How long to iterate over vertices adjacent to v ?

Adjacency-matrix graph representation

Maintain a two-dimensional V-by-V boolean array;

for each edge v–w in graph: `adj[v][w] = adj[w][v] = true`.



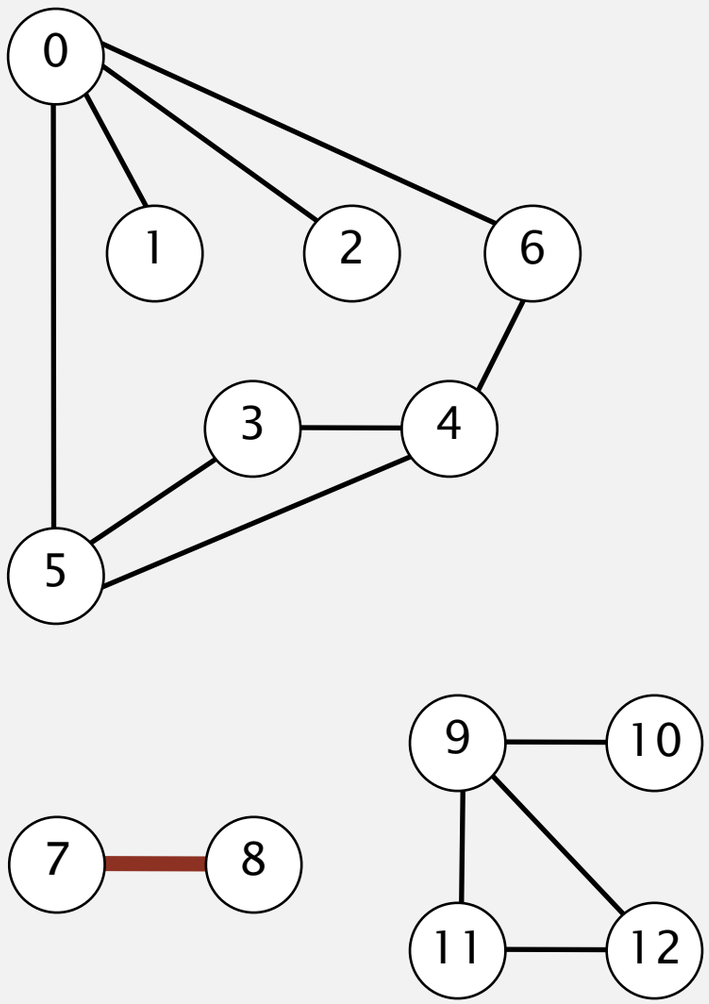
two entries for each edge

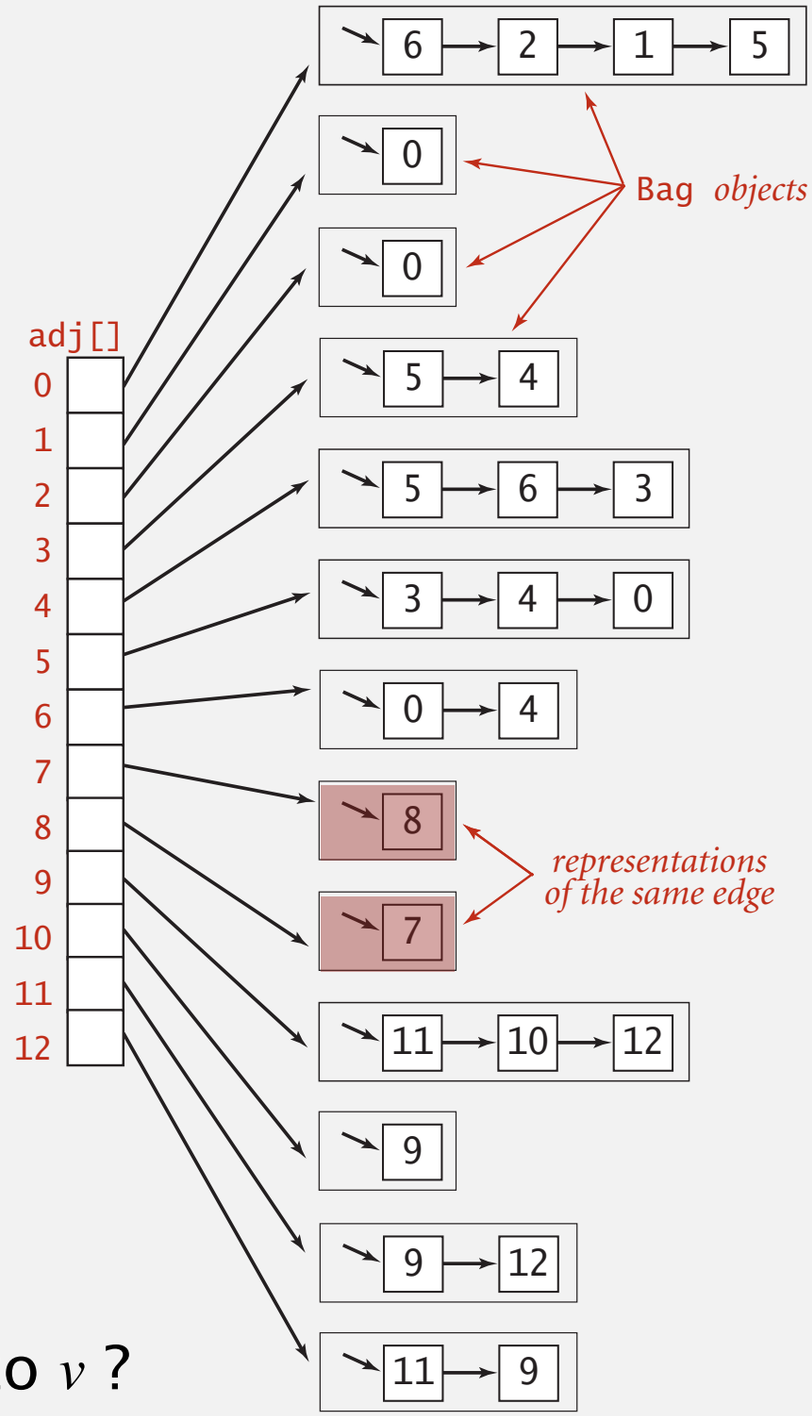
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Q. How long to iterate over vertices adjacent to v ?

Adjacency-list graph representation

Maintain vertex-indexed array of lists.





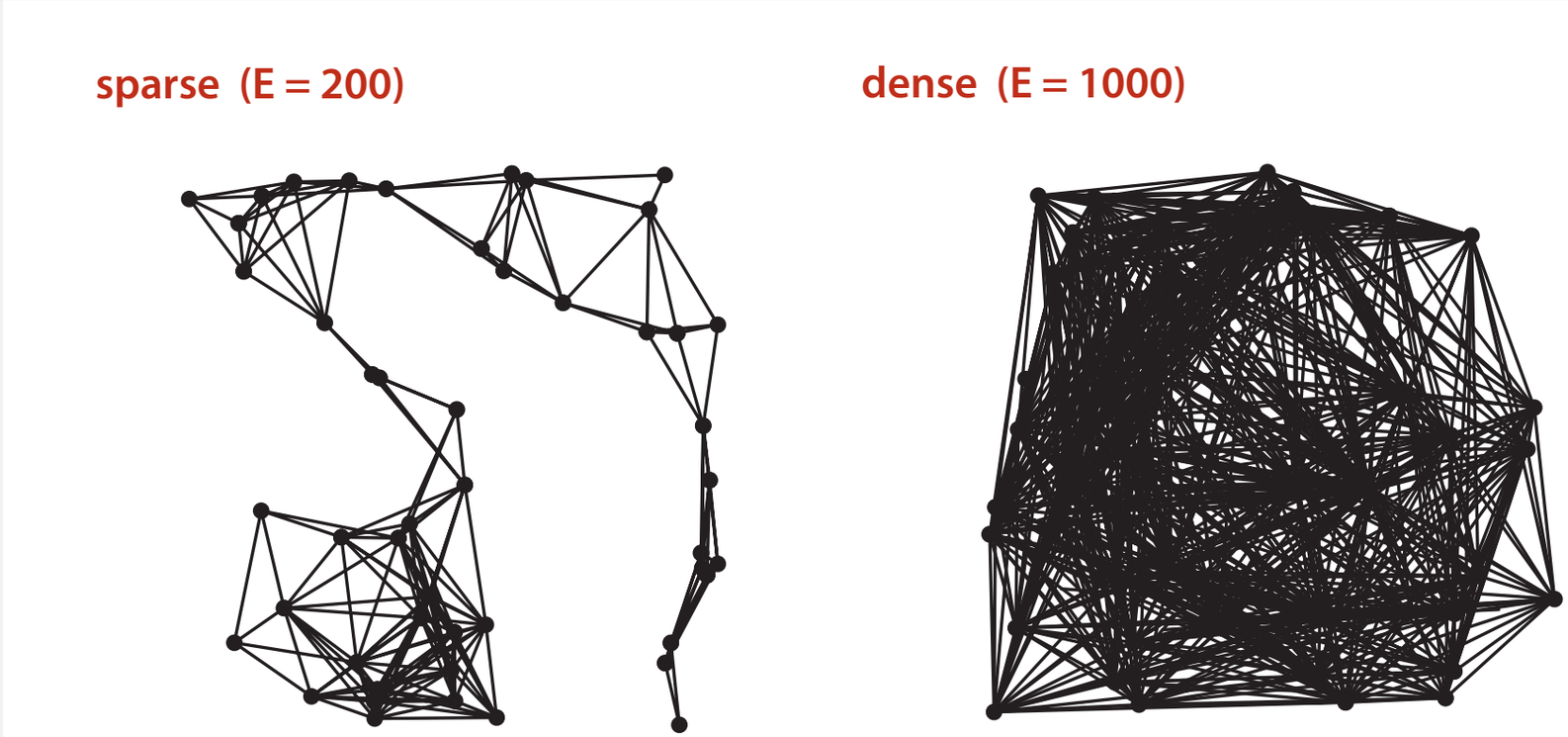
Q. How long to iterate over vertices adjacent to v ?

Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

huge number of vertices,
small average vertex degree



Two graphs (V = 50)

Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be sparse.

huge number of vertices,
small average vertex degree

representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	$E + V$	1	$degree(v)$	$degree(v)$

* disallows parallel edges

Adjacency-list graph representation: Java implementation

```
public class Graph
{
   private final int V;
   private Bag<Integer>[] adj;

   public Graph(int V)
   {
      this.V = V;
      adj = (Bag<Integer>[]) new Bag[V];
      for (int v = 0; v < V; v++)
         adj[v] = new Bag<Integer>();
   }

   public void addEdge(int v, int w)
   {
      adj[v].add(w);
      adj[w].add(v);
   }

   public Iterable<Integer> adj(int v)
   {  return adj[v];  }
}
```

adjacency lists
(using Bag data type)

create empty graph
with V vertices

add edge v-w
(parallel edges and
self-loops allowed)

iterator for vertices adjacent to v

http://algs4.cs.princeton.edu

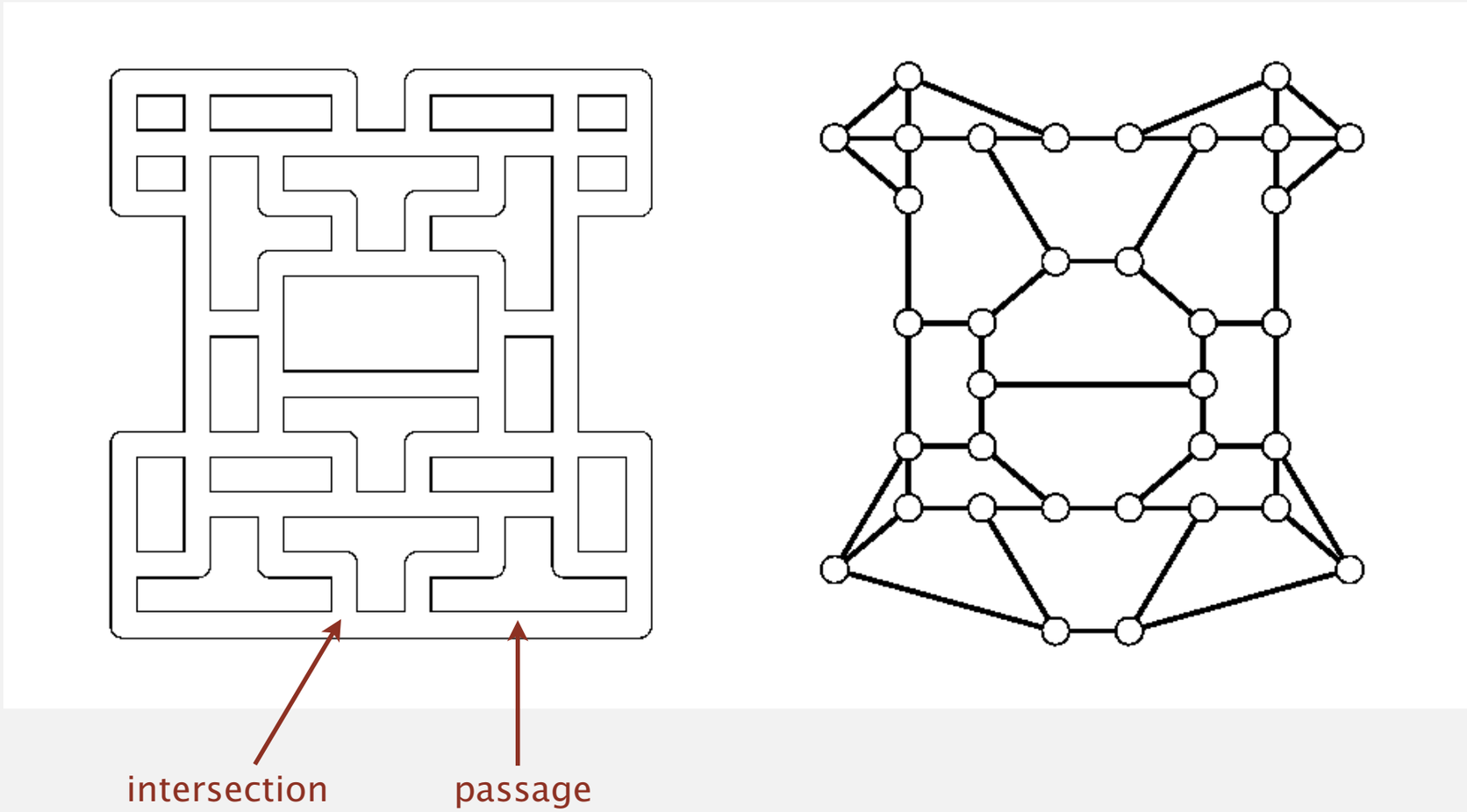
4.1 Undirected Graphs

- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

Maze exploration

Maze graph.

- Vertex = intersection.
- Edge = passage.

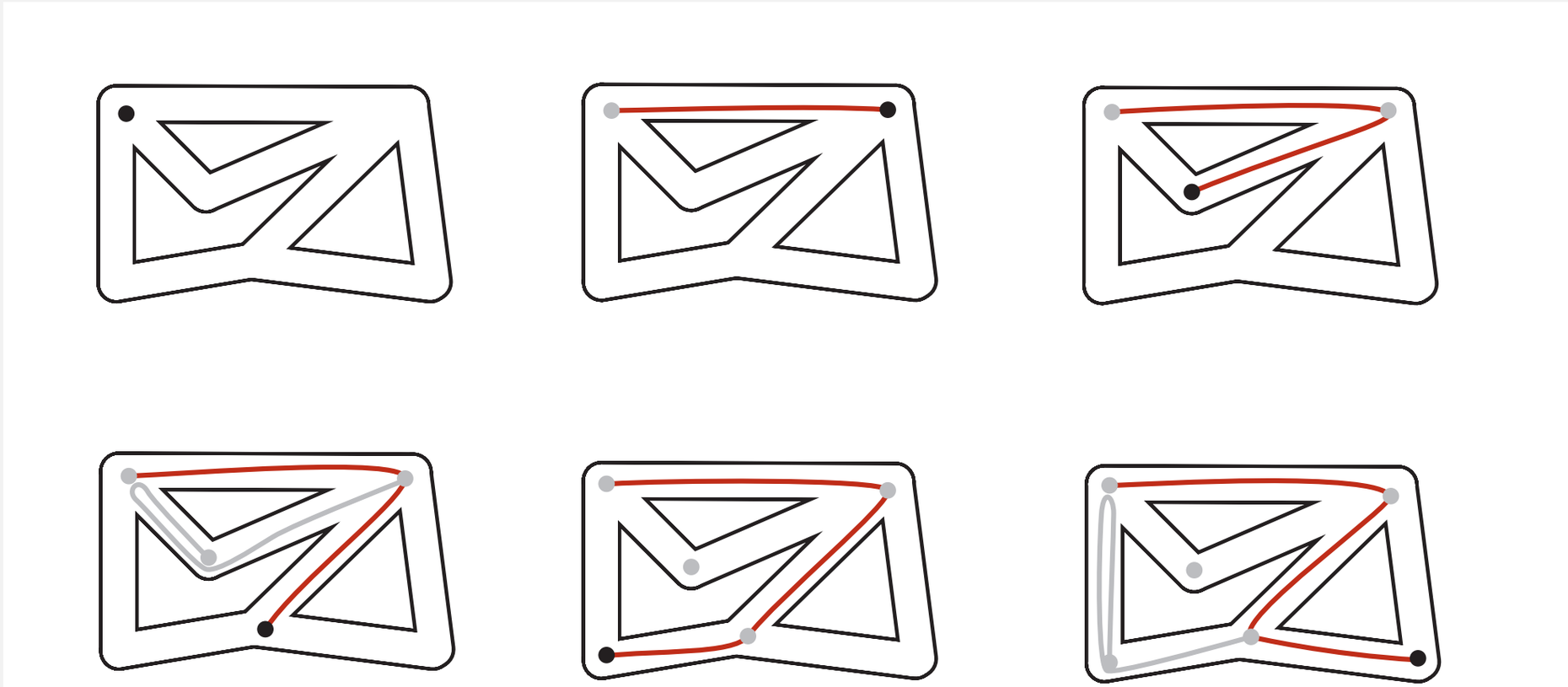


Goal. Explore every intersection in the maze.

Trémaux maze exploration

Algorithm.

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.



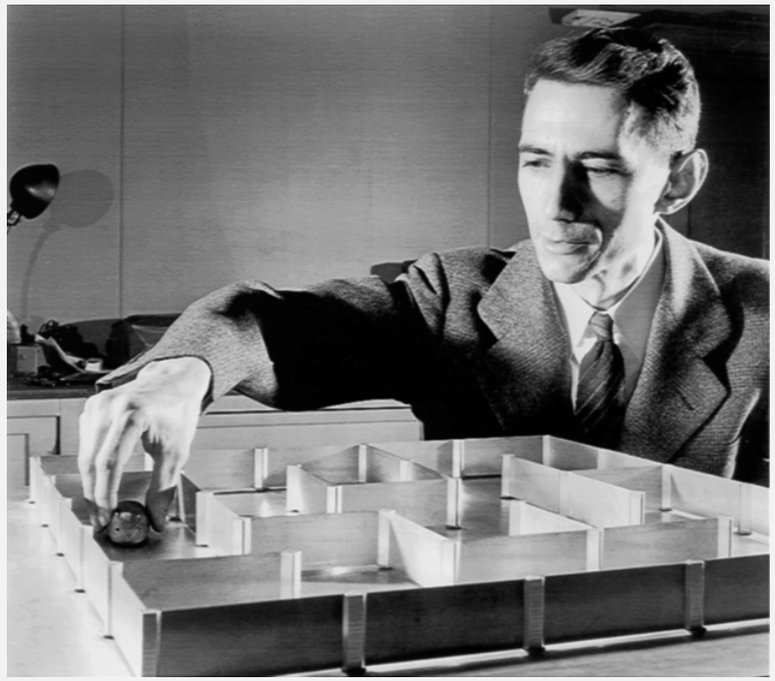
Trémaux maze exploration

Algorithm.

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.

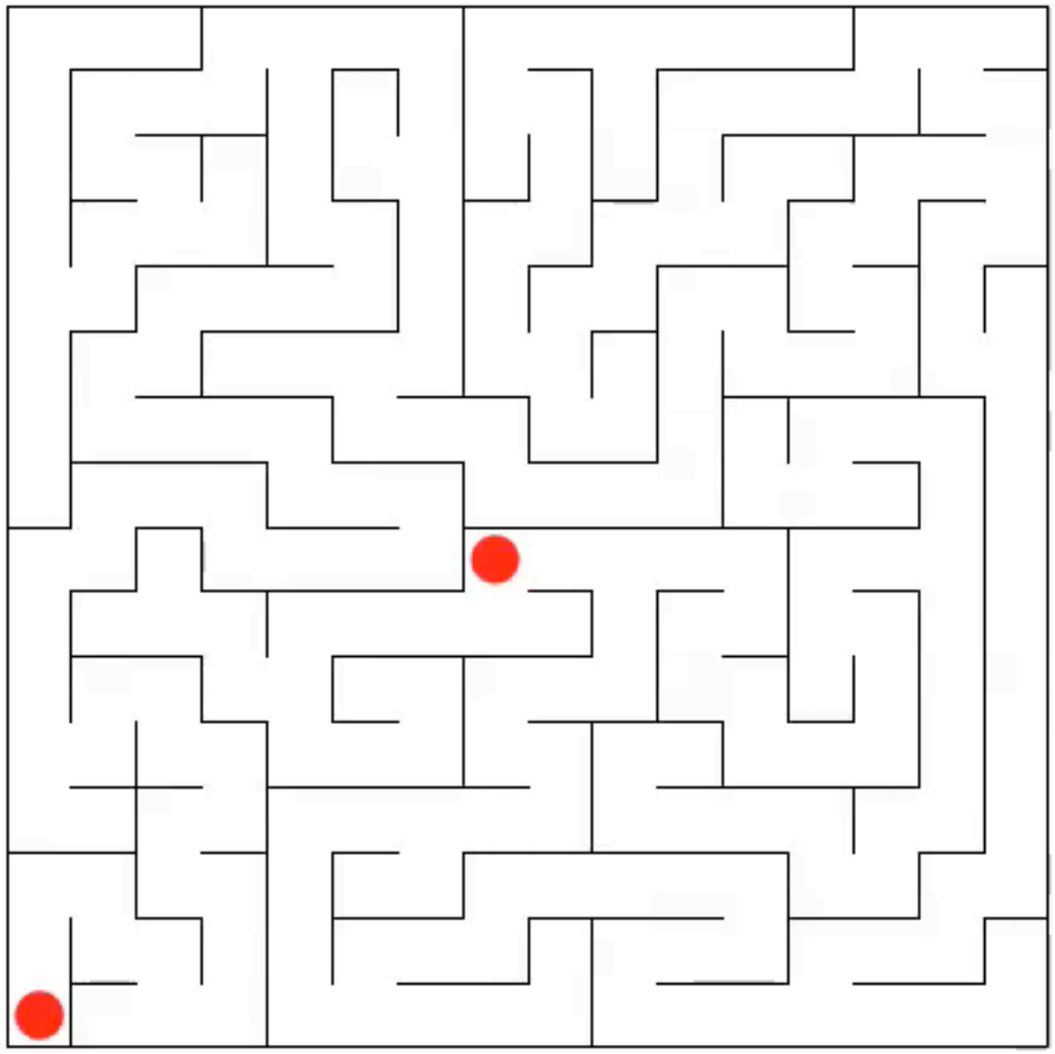
First use? Theseus entered Labyrinth to kill the monstrous Minotaur;
Ariadne instructed Theseus to use a ball of string to find his way back out.

The Labyrinth (with Minotaur)

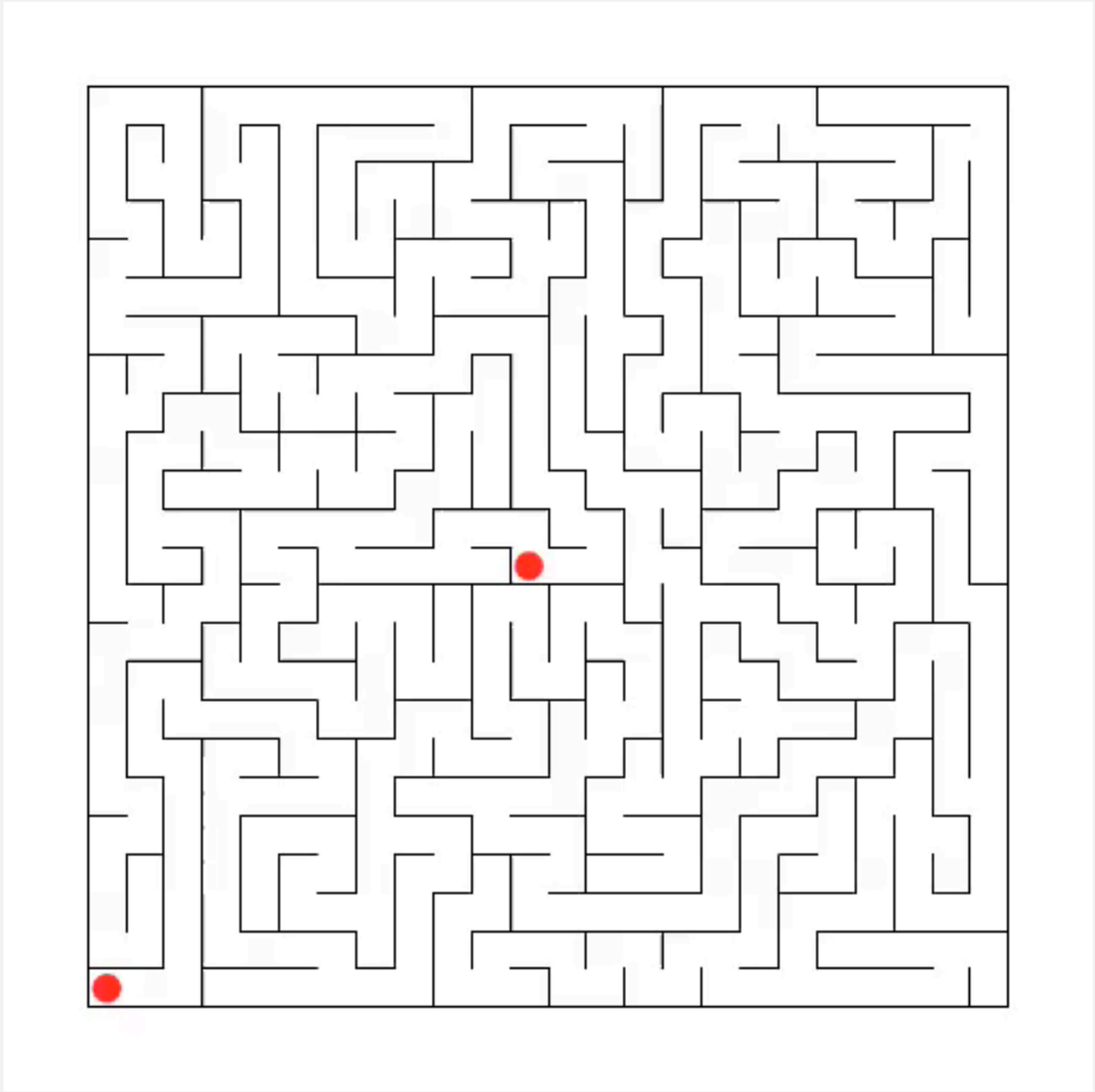


Claude Shannon (with Theseus mouse)

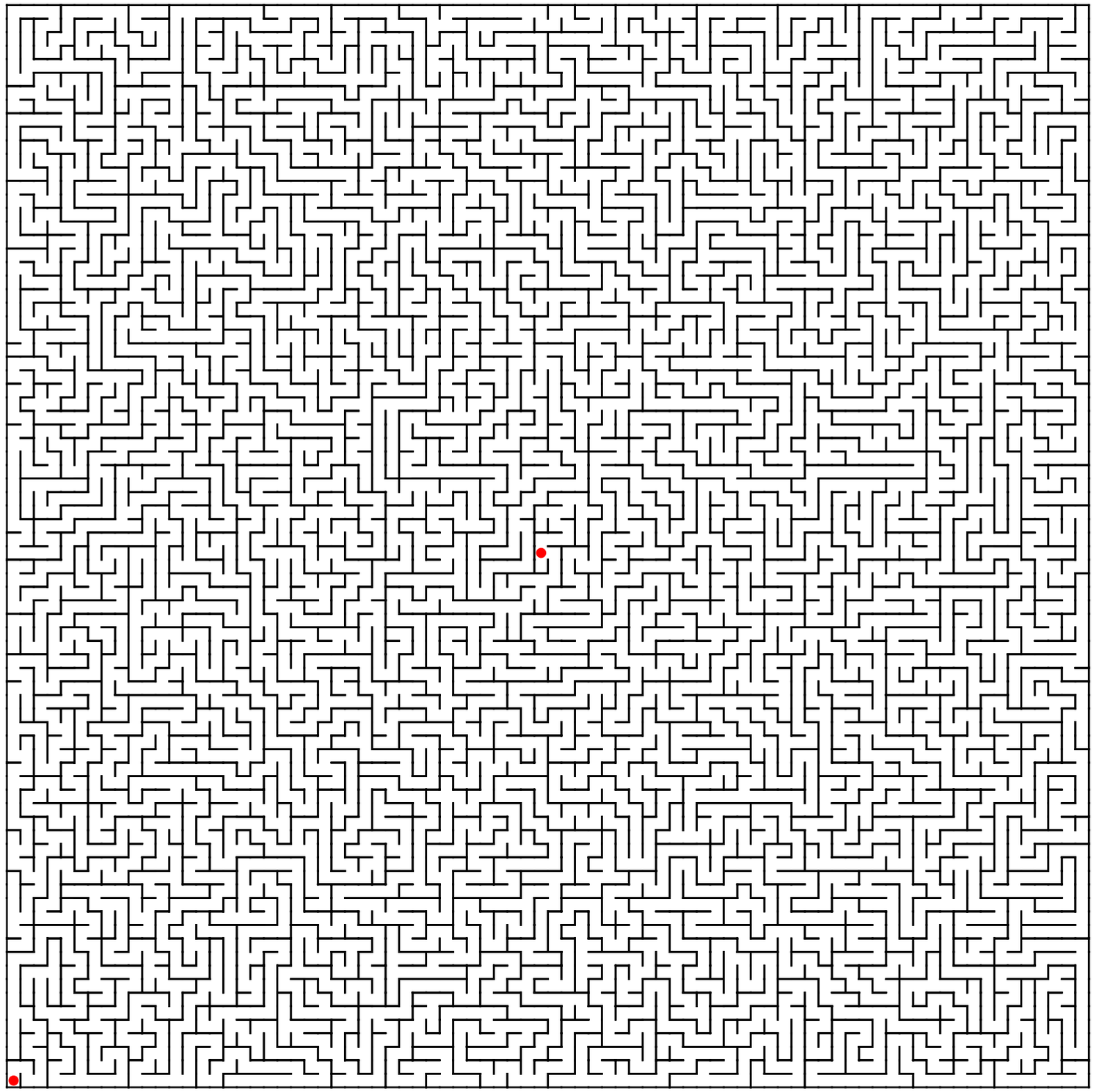
Maze exploration: easy



Maze exploration: medium



Maze exploration: challenge for the bored



Depth-first search

Goal. Systematically traverse a graph.

Idea. Mimic maze exploration. ← function-call stack acts as ball of string

DFS (to visit a vertex v)

Mark v as visited.

Recursively visit all unmarked vertices w adjacent to v.

Typical applications.

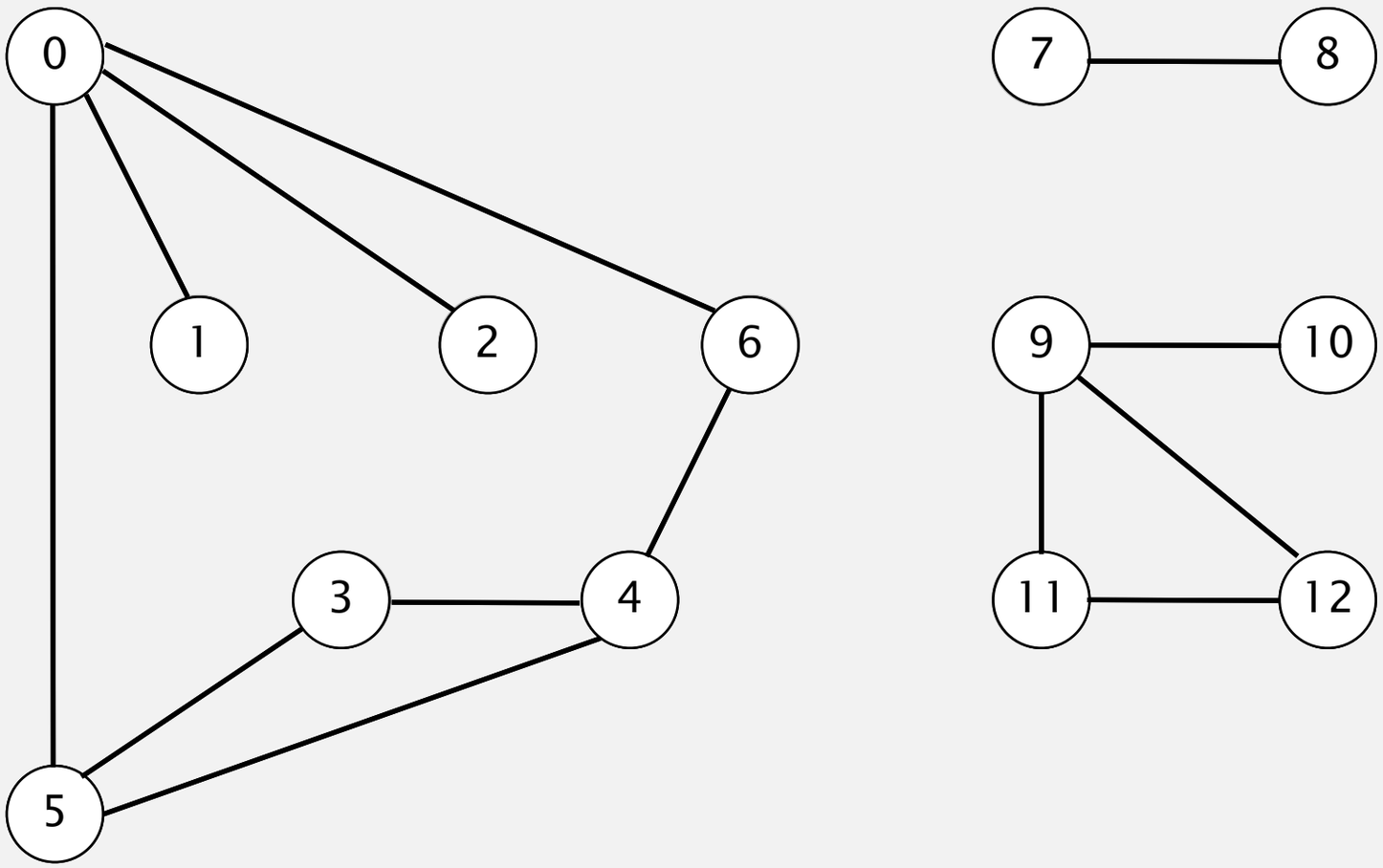
- Find all vertices connected to a given source vertex.
- Find a path between two vertices.

Design challenge. How to implement?

Depth-first search demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.



graph G

tinyG.txt

V → 13

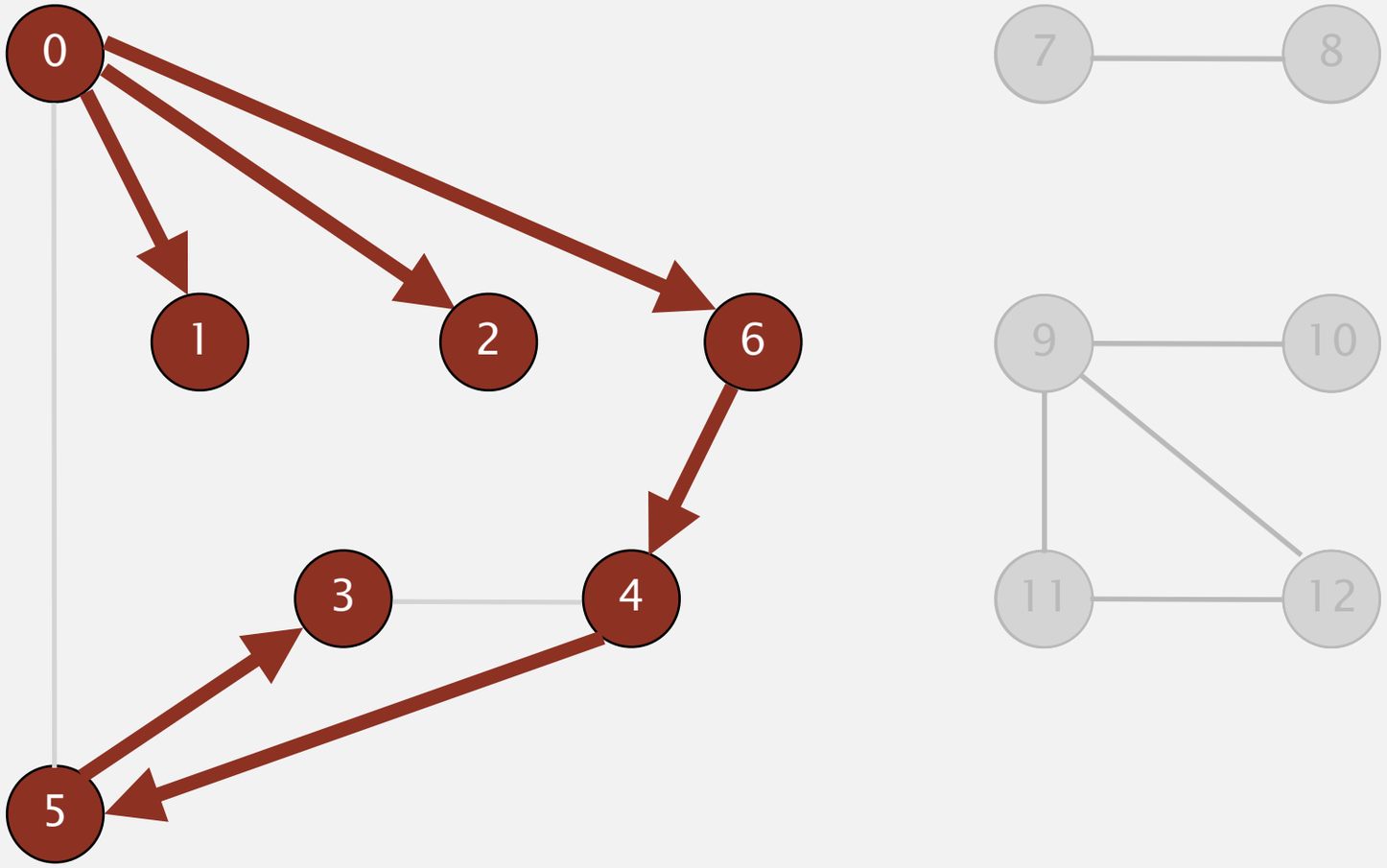
E → 13

```
13
13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3
```

Depth-first search demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.



v	marked[]	edgeTo[]
0	T	–
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

vertices reachable from 0

Design pattern for graph processing

Design pattern. Decouple graph data type from graph processing.

- Create a `Graph` object.
- Pass the `Graph` to a graph-processing routine.
- Query the graph-processing routine for information.

`public class`	`Paths`	
	`Paths(Graph G, int s)`	*find paths in G from source s*
`boolean`	`hasPathTo(int v)`	*is there a path from s to v?*
`Iterable<Integer>`	`pathTo(int v)`	*path from s to v; null if no such path*

```
Paths paths = new Paths(G, s);
for (int v = 0; v < G.V(); v++)
   if (paths.hasPathTo(v))
      StdOut.println(v);
```

print all vertices connected to s

Depth-first search: data structures

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.

Data structures.

- Boolean array `marked[]` to mark visited vertices.
- Integer array `edgeTo[]` to keep track of paths.

  `(edgeTo[w] == v)` means that edge `v-w` taken to visit `w` for first time
- Function-call stack for recursion.

Depth-first search: Java implementation

```
public class DepthFirstPaths
{
   private boolean[] marked;
   private int[] edgeTo;
   private int s;

   public DepthFirstPaths(Graph G, int s)
   {
      ...
      dfs(G, s);
   }

   private void dfs(Graph G, int v)
   {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w])
         {
            dfs(G, w);
            edgeTo[w] = v;
         }
   }
}
```

marked[v] = true if v connected to s

edgeTo[v] = previous vertex on path from s to v

initialize data structures

find vertices connected to s

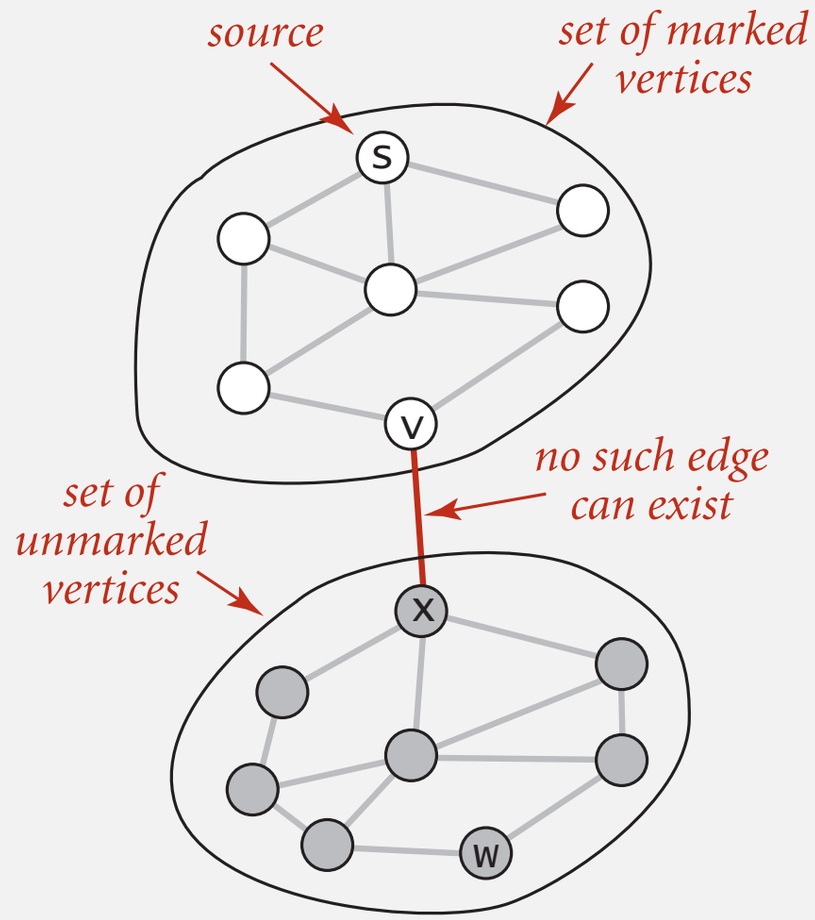
recursive DFS does the work

Depth-first search: properties

Proposition. DFS marks all vertices connected to s in time proportional to the sum of their degrees (plus time to initialize the `marked[]` array).

Pf. [correctness]

- If w marked, then w connected to s (why?)
- If w connected to s, then w marked.
  (if w unmarked, then consider last edge on a path from s to w that goes from a marked vertex to an unmarked one).



Pf. [running time]

Each vertex connected to s is visited once.

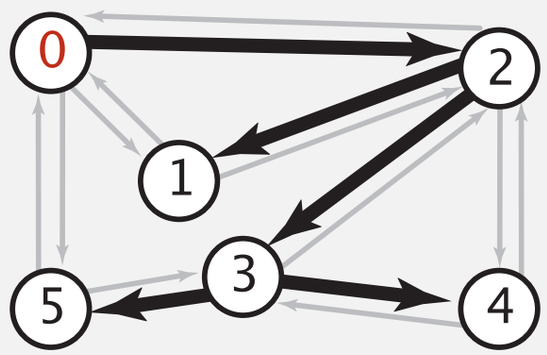
Depth-first search: properties

Proposition. After DFS, can check if vertex v is connected to s in constant time and can find v–s path (if one exists) in time proportional to its length.

Pf. `edgeTo[]` is parent-link representation of a tree rooted at vertex `s`.

```
public boolean hasPathTo(int v)
{  return marked[v];  }

public Iterable<Integer> pathTo(int v)
{
   if (!hasPathTo(v)) return null;
   Stack<Integer> path = new Stack<Integer>();
   for (int x = v; x != s; x = edgeTo[x])
      path.push(x);
   path.push(s);
   return path;
}
```



edgeTo[]

0	
1	2
2	0
3	2
4	3
5	3

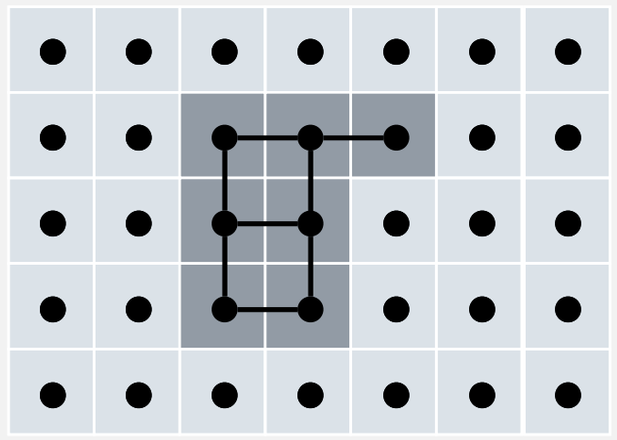
Depth-first search application: flood fill

Challenge. Flood fill (Photoshop magic wand).

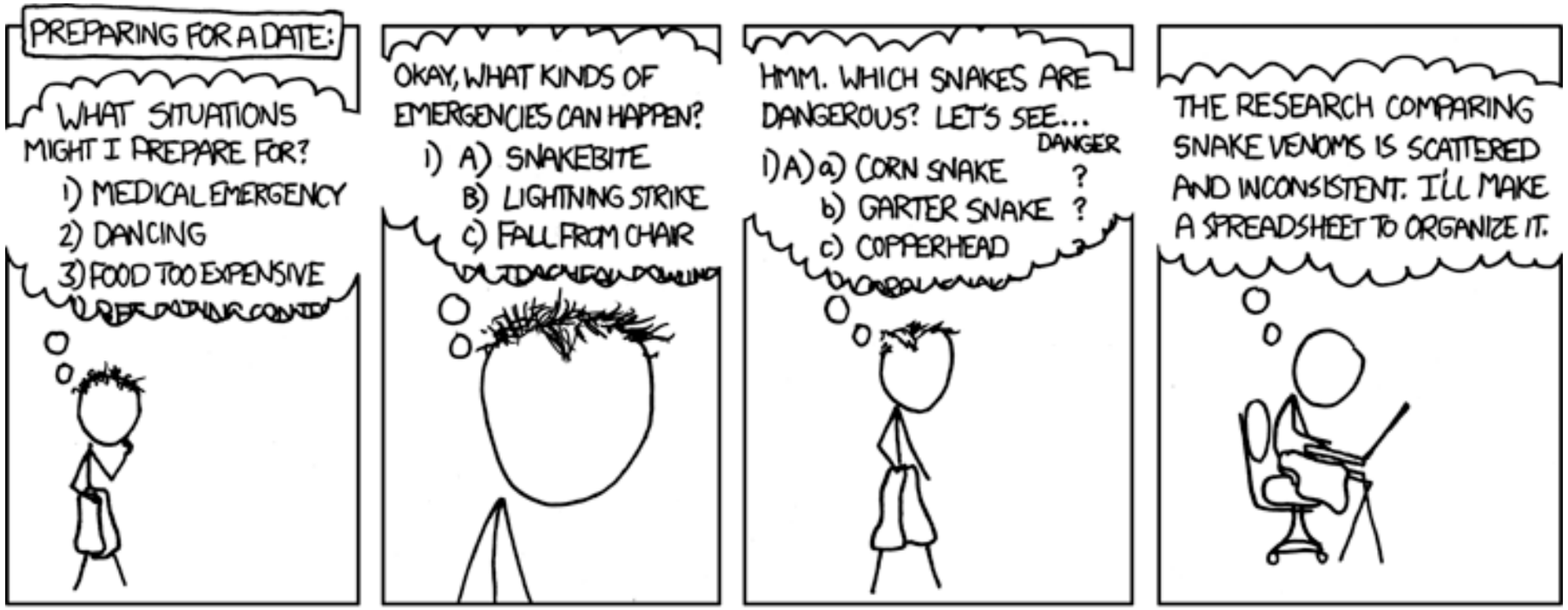
Assumptions. Picture has millions to billions of pixels.

Solution. Build a grid graph (implicitly).

- Vertex: pixel.
- Edge: between two adjacent gray pixels.
- Blob: all pixels connected to given pixel.



Depth-first search application: preparing for a date



I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

xkcd

http://xkcd.com/761/

http://algs4.cs.princeton.edu

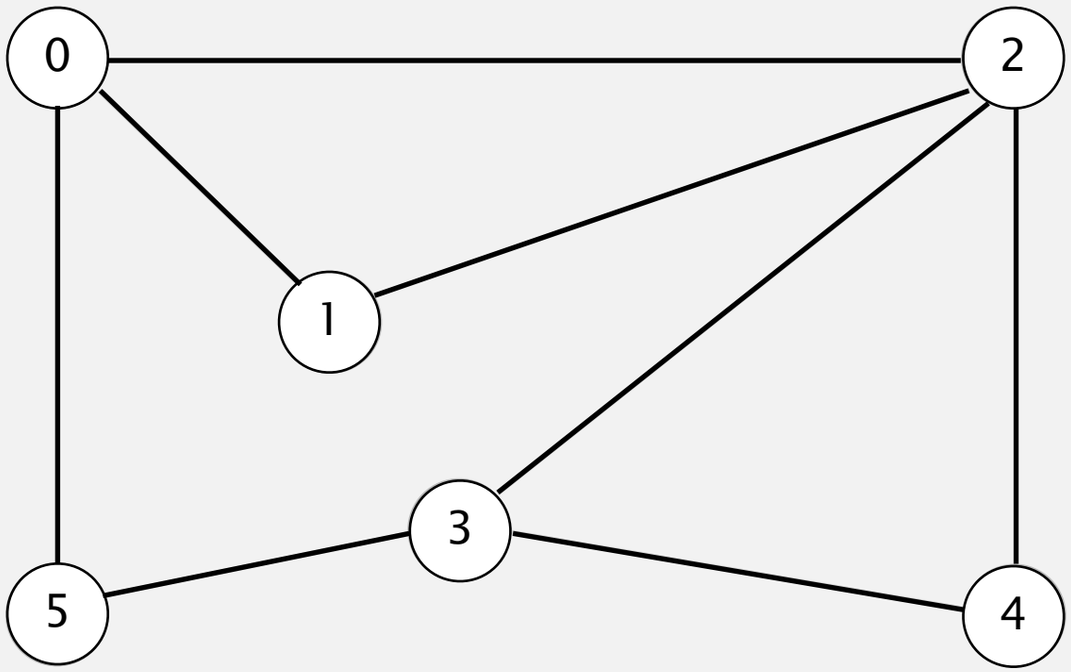
4.1 Undirected Graphs

- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

Breadth-first search demo

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



graph G

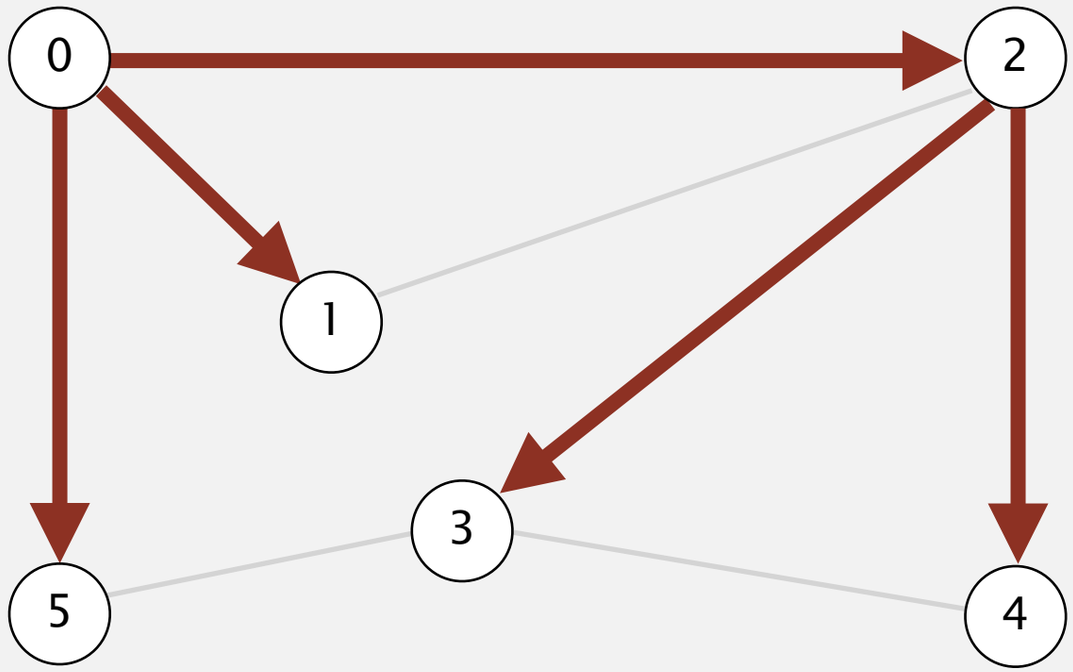
tinyCG.txt

```
V
6
8   E
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
```

Breadth-first search demo

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



v	edgeTo[]	distTo[]
0	–	0
1	0	1
2	0	1
3	2	2
4	2	2
5	0	1

done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.

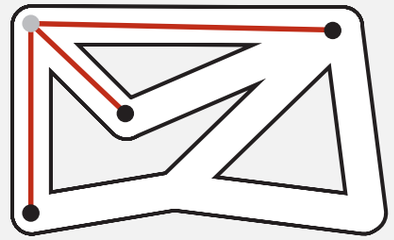
BFS (from source vertex s)

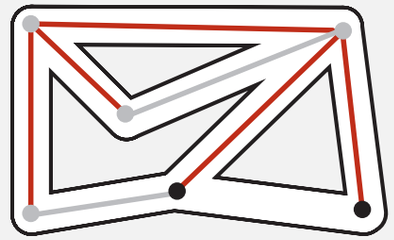
Put s onto a FIFO queue, and mark s as visited.

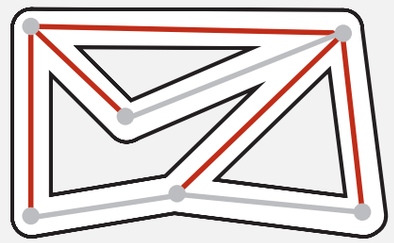
Repeat until the queue is empty:

- remove the least recently added vertex v

- add each of v's unvisited neighbors to the queue, and mark them as visited.







Breadth-first search: Java implementation

```
public class BreadthFirstPaths
{
   private boolean[] marked;
   private int[] edgeTo;
   private int[] distTo;

   …

   private void bfs(Graph G, int s) {
      Queue<Integer> q = new Queue<Integer>();
      q.enqueue(s);
      marked[s] = true;
      distTo[s] = 0;

      while (!q.isEmpty()) {
         int v = q.dequeue();
         for (int w : G.adj(v)) {
            if (!marked[w]) {
               q.enqueue(w);
               marked[w] = true;
               edgeTo[w] = v;
               distTo[w] = distTo[v] + 1;
            }
         }
      }
   }
}
```

initialize FIFO queue of vertices to explore

found new vertex w via edge v-w

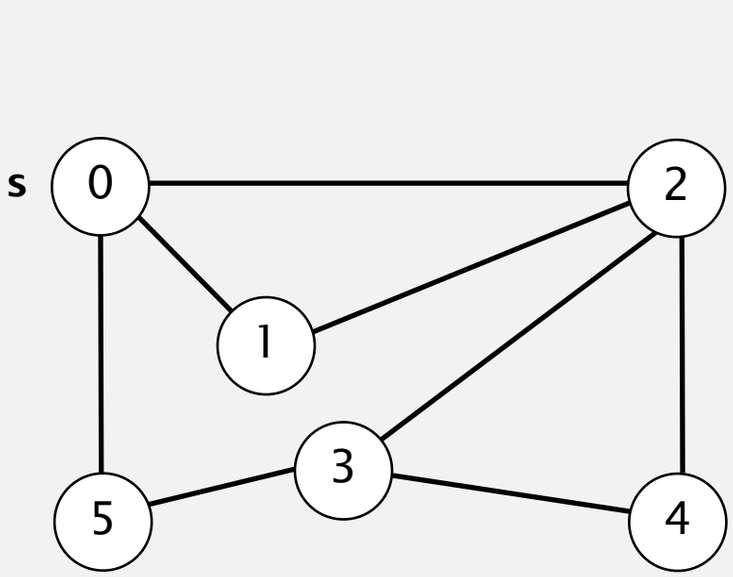
Breadth-first search properties

Q. In which order does BFS examine vertices?

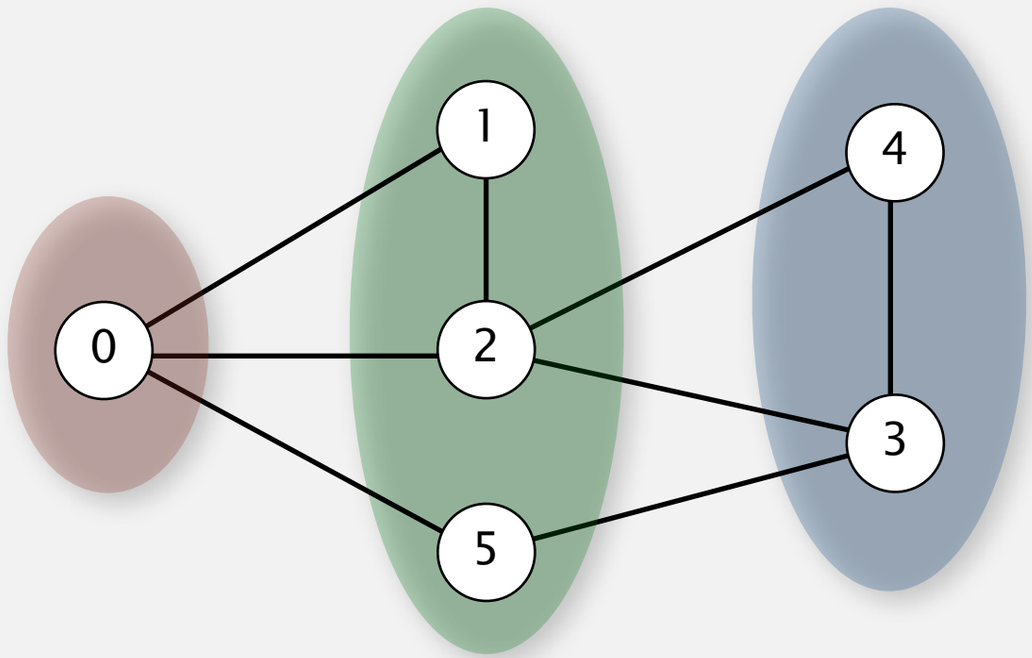
A. Increasing distance (number of edges) from s.

queue always consists of ≥ 0 vertices of distance k from s,
followed by ≥ 0 vertices of distance k+1

Proposition. In any connected graph G, BFS computes shortest paths from s to all other vertices in time proportional to $E + V$.



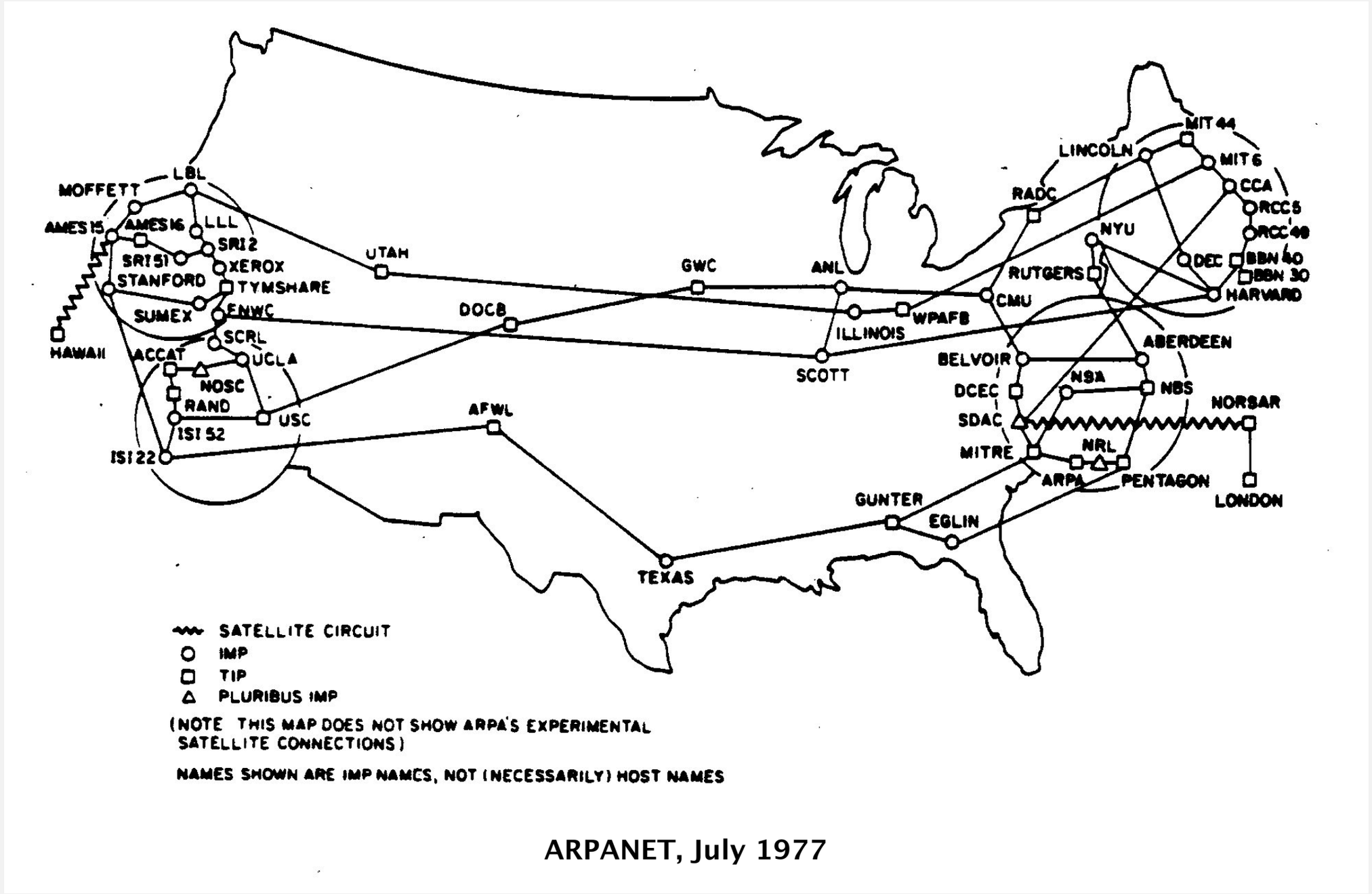
graph G



dist = 0 dist = 1 dist = 2

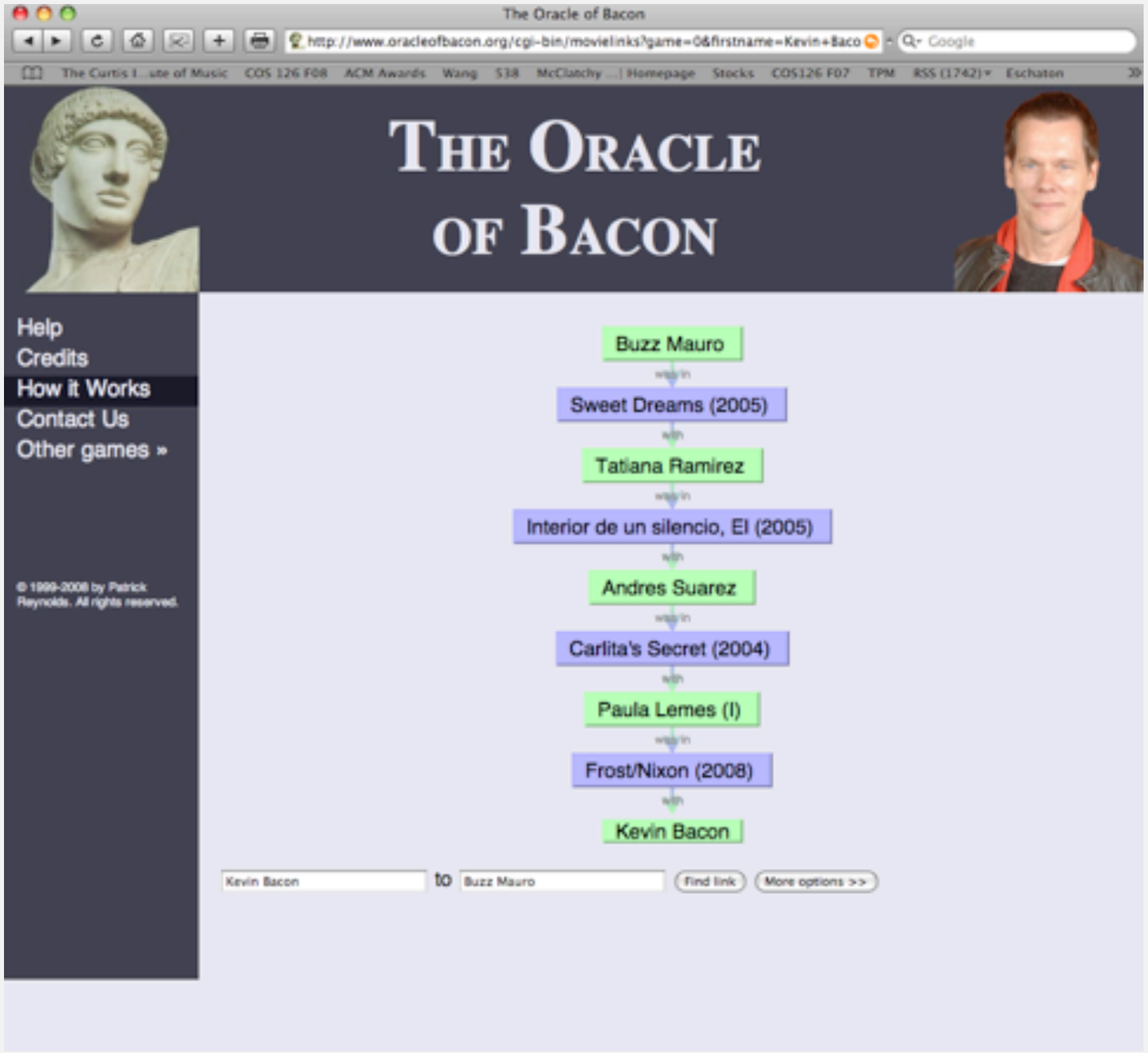
Breadth-first search application: routing

Fewest number of hops in a communication network.



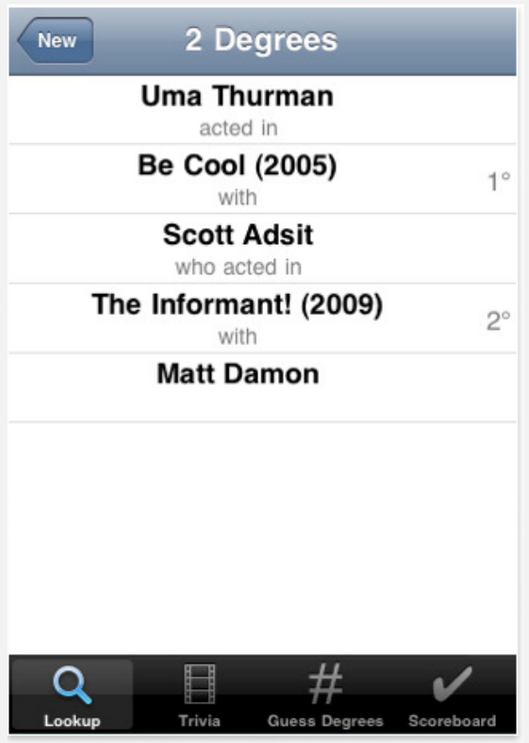
ARPANET, July 1977

Breadth-first search application: Kevin Bacon numbers



http://oracleofbacon.org

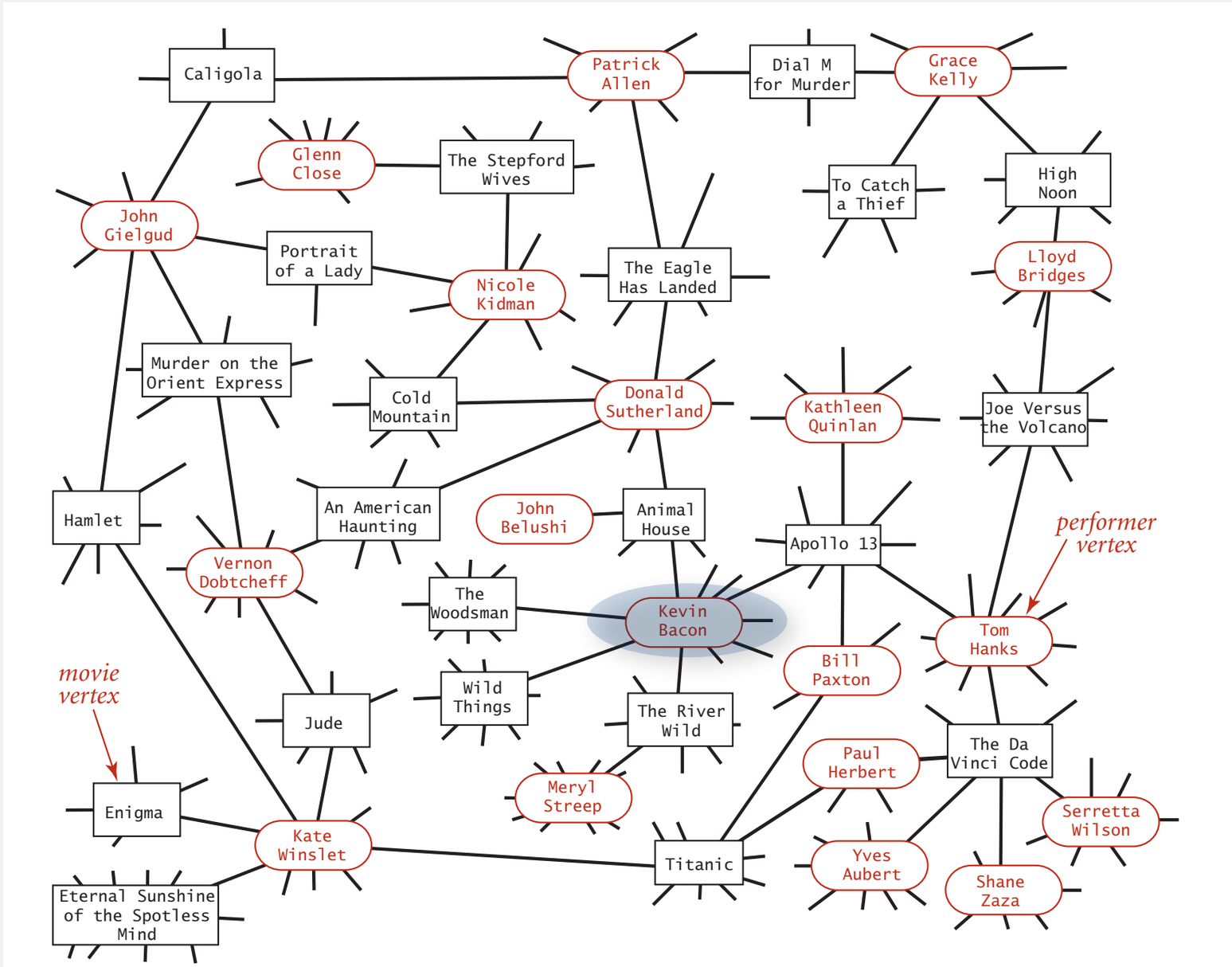
Endless Games board game



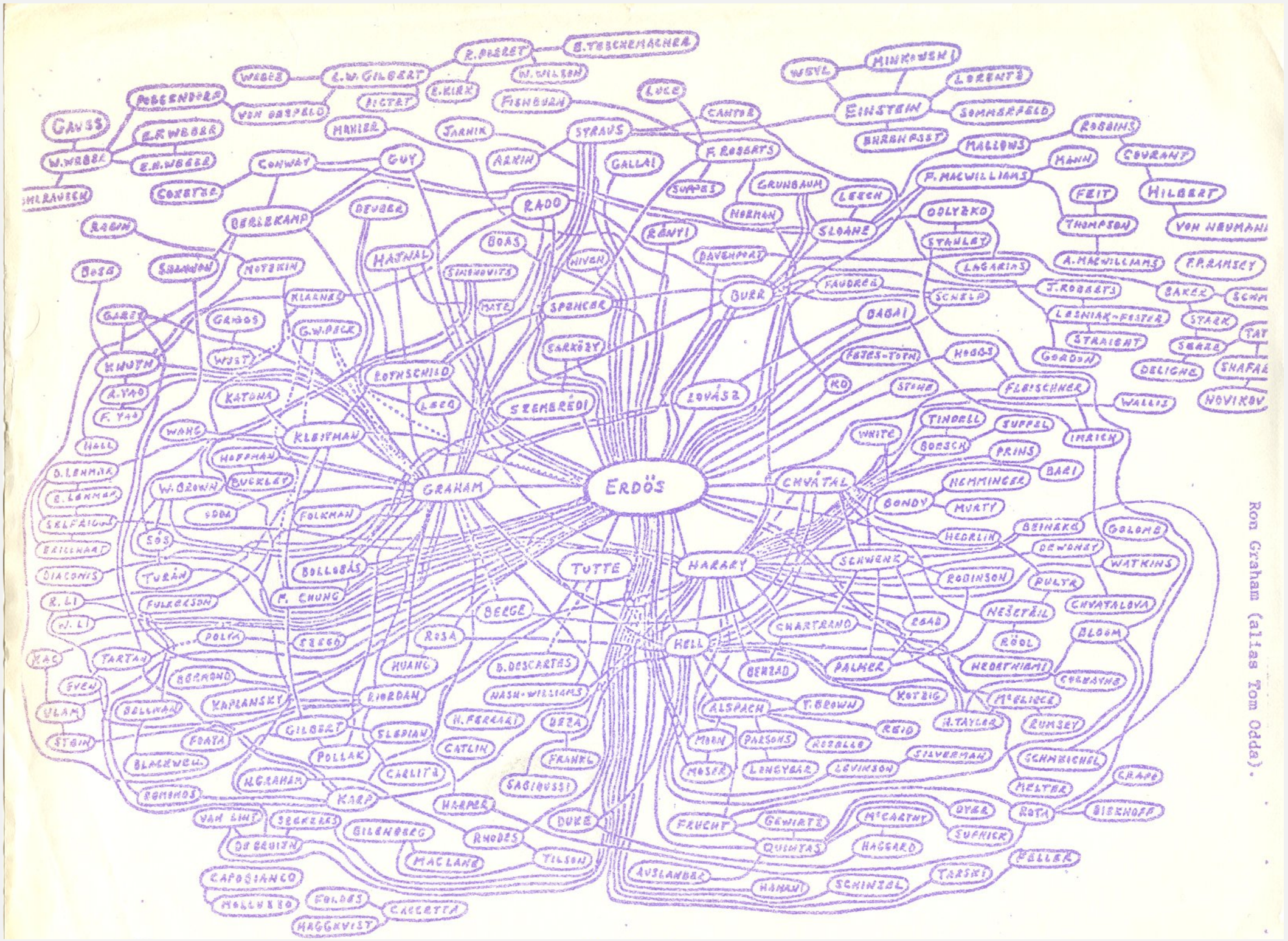
SixDegrees iPhone App

Kevin Bacon graph

- Include one vertex for each performer and one for each movie.
- Connect a movie to all performers that appear in that movie.
- Compute shortest path from s = Kevin Bacon.



Breadth-first search application: Erdös numbers



hand-drawing of part of the Erdös graph by Ron Graham

http://algs4.cs.princeton.edu

4.1 Undirected Graphs

- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

Connectivity queries

Def. Vertices v and w are connected if there is a path between them.

Goal. Preprocess graph to answer queries of the form *is v connected to w?* in constant time.

public class CC	
CC(Graph G)	*find connected components in G*
boolean connected(int v, int w)	*are v and w connected?*
int count()	*number of connected components*
int id(int v)	*component identifier for v* (*between* 0 *and* count() - 1)

Union-Find? Not quite.

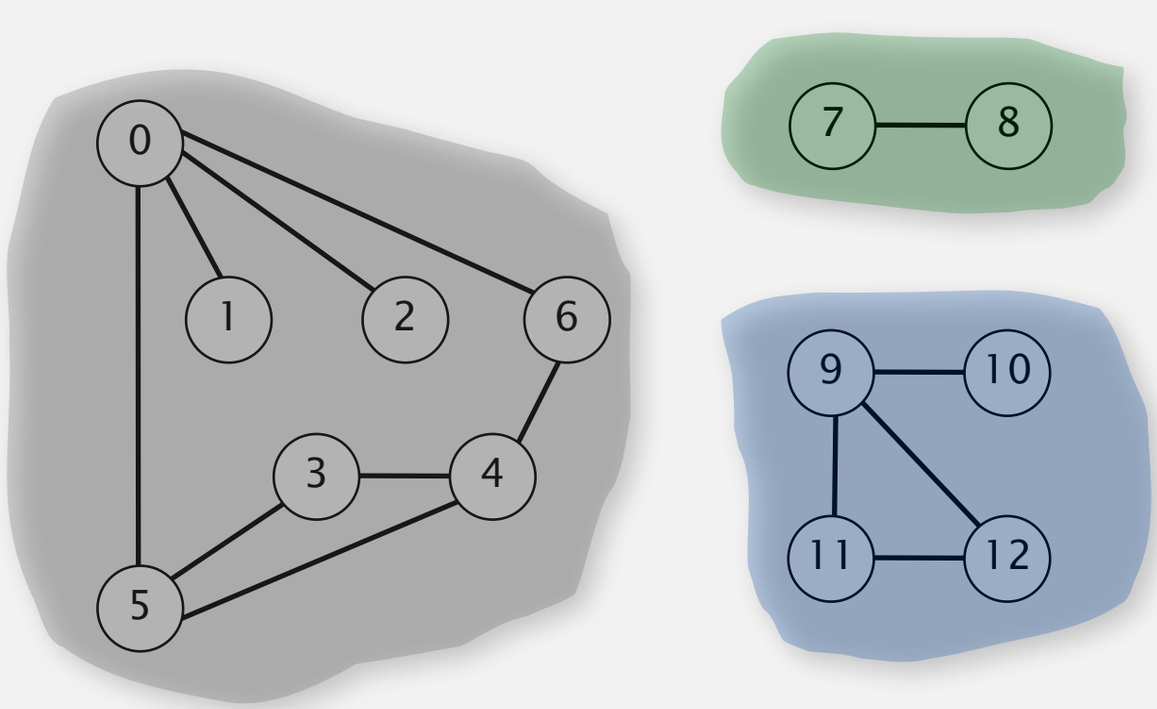
Depth-first search. Yes. [next few slides]

Connected components

The relation "is connected to" is an equivalence relation:

- Reflexive: v is connected to v.
- Symmetric: if v is connected to w, then w is connected to v.
- Transitive: if v connected to w and w connected to x, then v connected to x.

Def. A connected component is a maximal set of connected vertices.



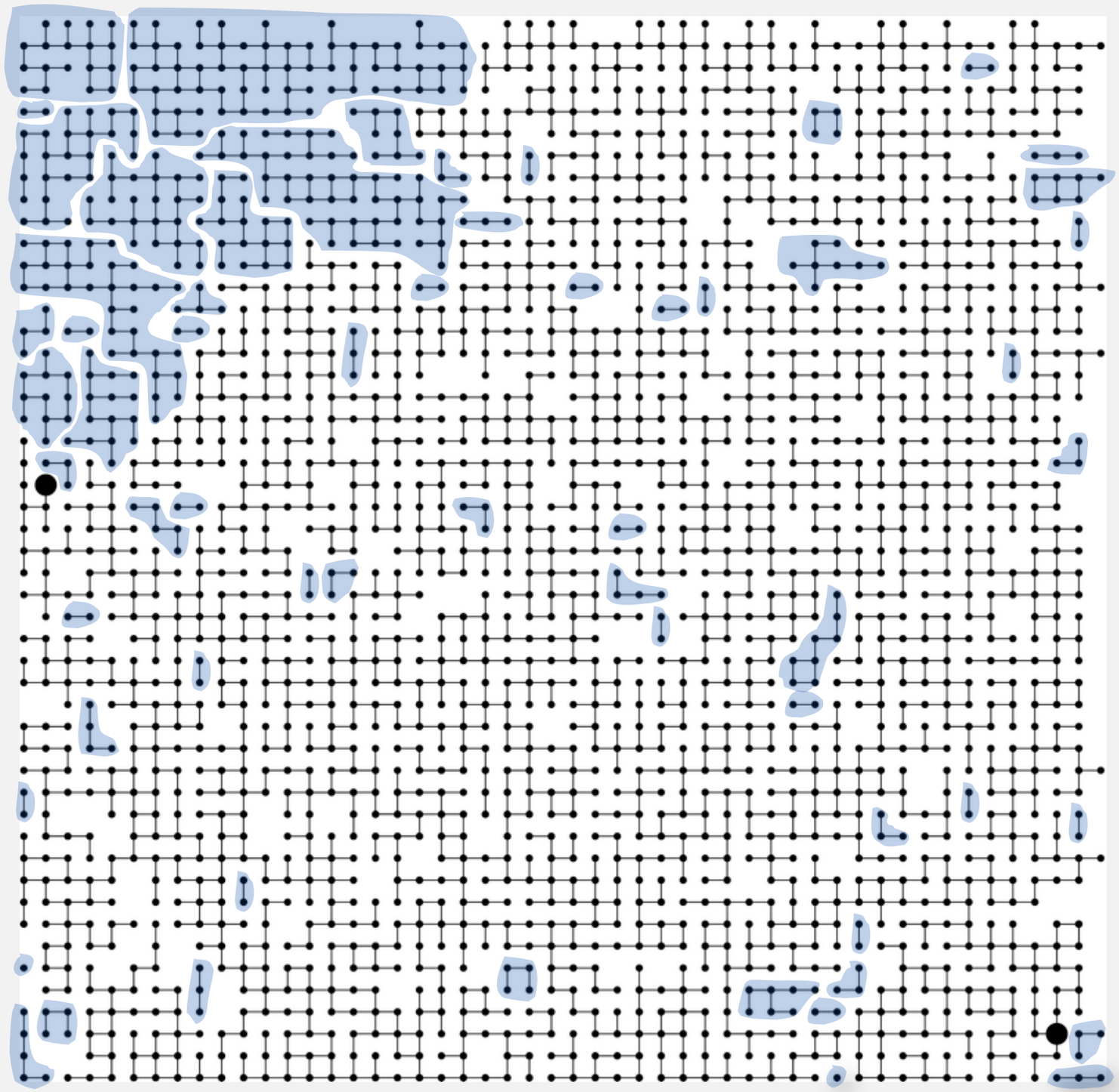
3 connected components

v	id[]
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	2
10	2
11	2
12	2

Remark. Given connected components, can answer queries in constant time.

Connected components

Def. A connected component is a maximal set of connected vertices.



63 connected components

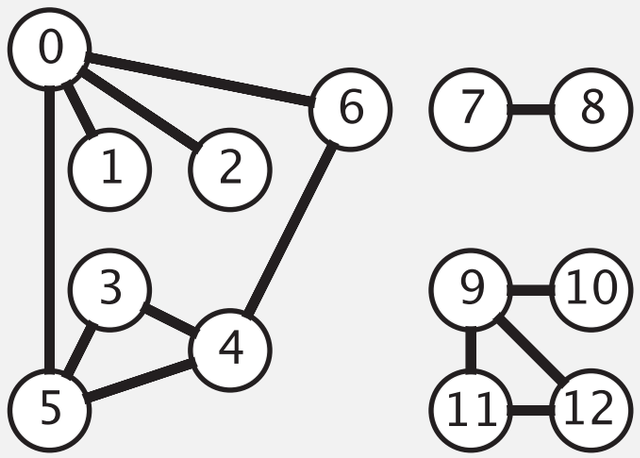
Connected components

Goal. Partition vertices into connected components.

Connected components

Initialize all vertices v as unmarked.

For each unmarked vertex v, run DFS to identify all vertices discovered as part of the same component.



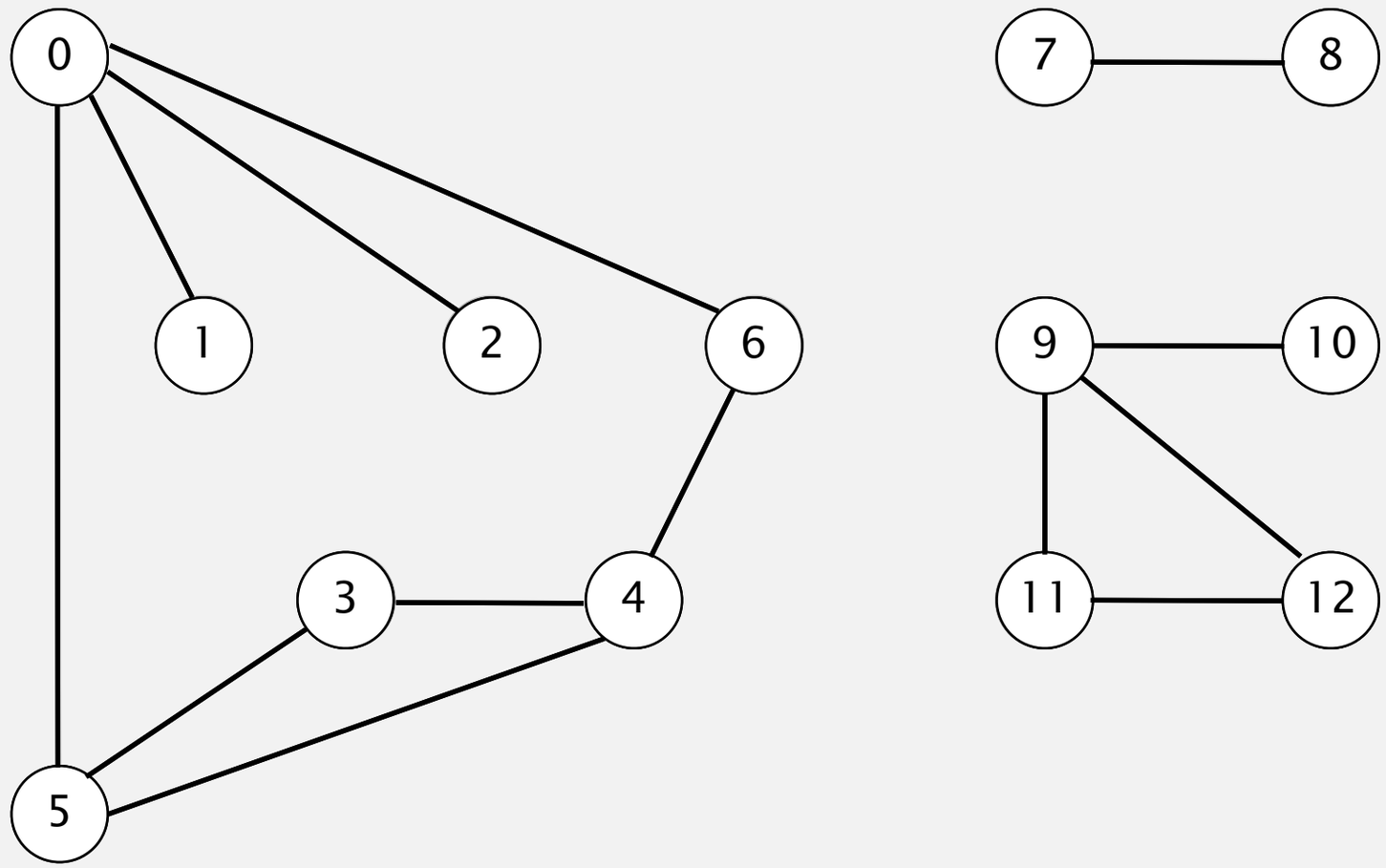
tinyG.txt

```
V → 13
13 ← E
0  5
4  3
0  1
9  12
6  4
5  4
0  2
11  12
9  10
0  6
7  8
9  11
5  3
```

Connected components demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.



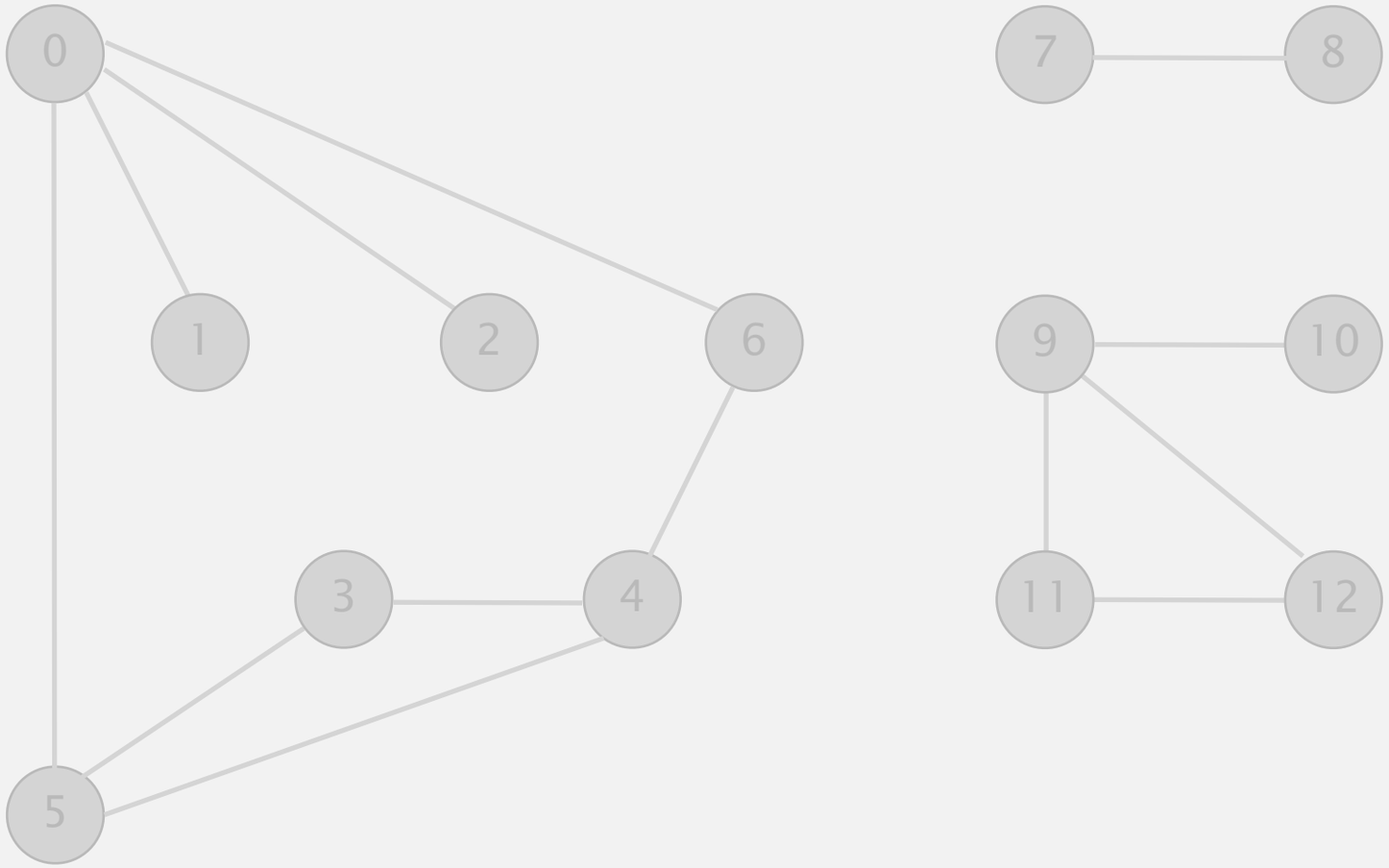
graph G

v	marked[]	id[]
0	F	–
1	F	–
2	F	–
3	F	–
4	F	–
5	F	–
6	F	–
7	F	–
8	F	–
9	F	–
10	F	–
11	F	–
12	F	–

Connected components demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.



v	marked[]	id[]
0	T	0
1	T	0
2	T	0
3	T	0
4	T	0
5	T	0
6	T	0
7	T	1
8	T	1
9	T	2
10	T	2
11	T	2
12	T	2

done

Finding connected components with DFS

```
public class CC
{
   private boolean[] marked;
   private int[] id;
   private int count;

   public CC(Graph G)
   {
      marked = new boolean[G.V()];
      id = new int[G.V()];
      for (int v = 0; v < G.V(); v++)
      {
         if (!marked[v])
         {
            dfs(G, v);
            count++;
         }
      }
   }

   public int count()
   public int id(int v)
   public boolean connected(int v, int w)
   private void dfs(Graph G, int v)
}
```

id[v] = id of component containing v

number of components

run DFS from one vertex in each component

see next slide

Finding connected components with DFS (continued)

```
public int count()
{  return count;  }
```

number of components

```
public int id(int v)
{  return id[v];  }
```

id of component containing v

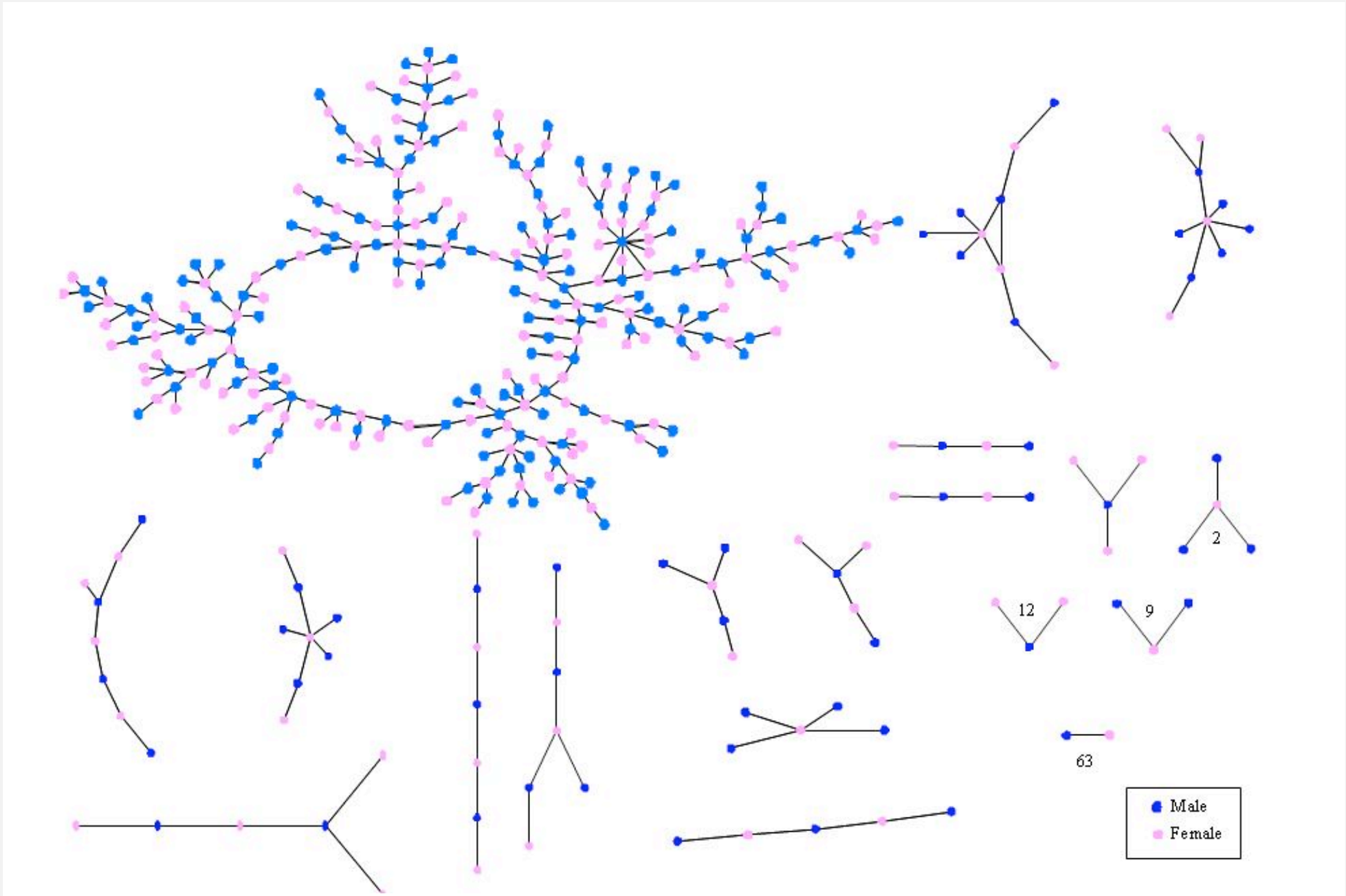
```
public boolean connected(int v, int w)
{ return id[v] == id[w];  }
```

v and w connected iff same id

```
private void dfs(Graph G, int v)
{
   marked[v] = true;
   id[v] = count;
   for (int w : G.adj(v))
      if (!marked[w])
         dfs(G, w);
}
```

all vertices discovered in same call of dfs have same id

Connected components application: study spread of STDs



Relationship graph at "Jefferson High"

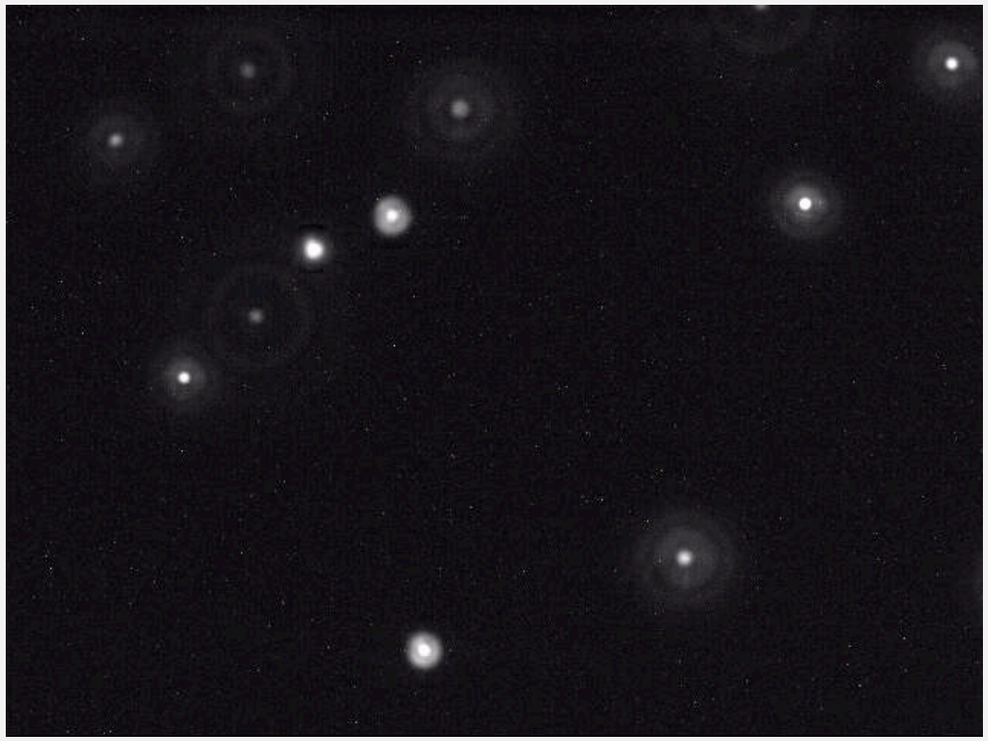
Peter Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. American Journal of Sociology, 110(1): 44–99, 2004.

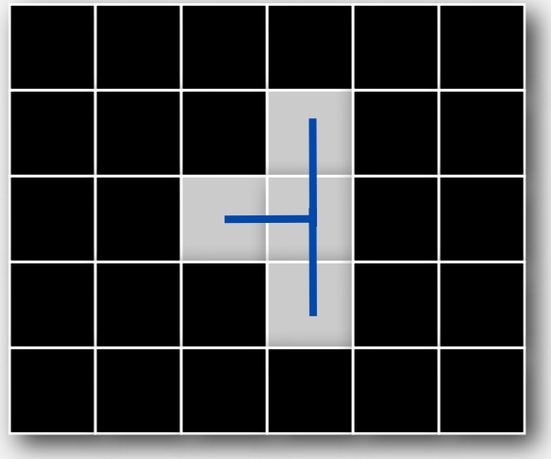
Connected components application: particle detection

Particle detection. Given grayscale image of particles, identify "blobs."

- Vertex: pixel.
- Edge: between two adjacent pixels with grayscale value ≥ 70.
- Blob: connected component of 20-30 pixels.

black = 0
white = 255





Particle tracking. Track moving particles over time.

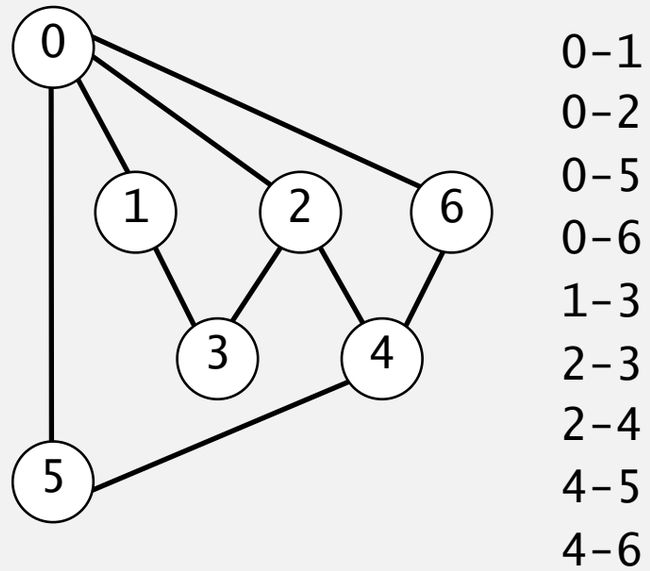
http://algs4.cs.princeton.edu

4.1 Undirected Graphs

- *introduction*
- *graph API*
- *depth-first search*
- *breadth-first search*
- *connected components*
- *challenges*

Graph-processing challenge 1

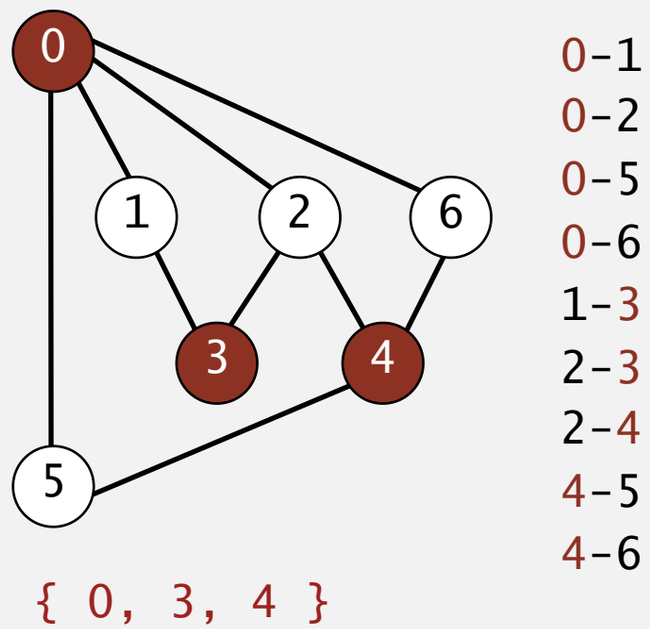
Problem. Is a graph bipartite?



How difficult?

- Any programmer could do it.
- ✓ Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

simple DFS-based solution
(see textbook)



{ 0, 3, 4 }

Bipartiteness application: is dating graph bipartite?

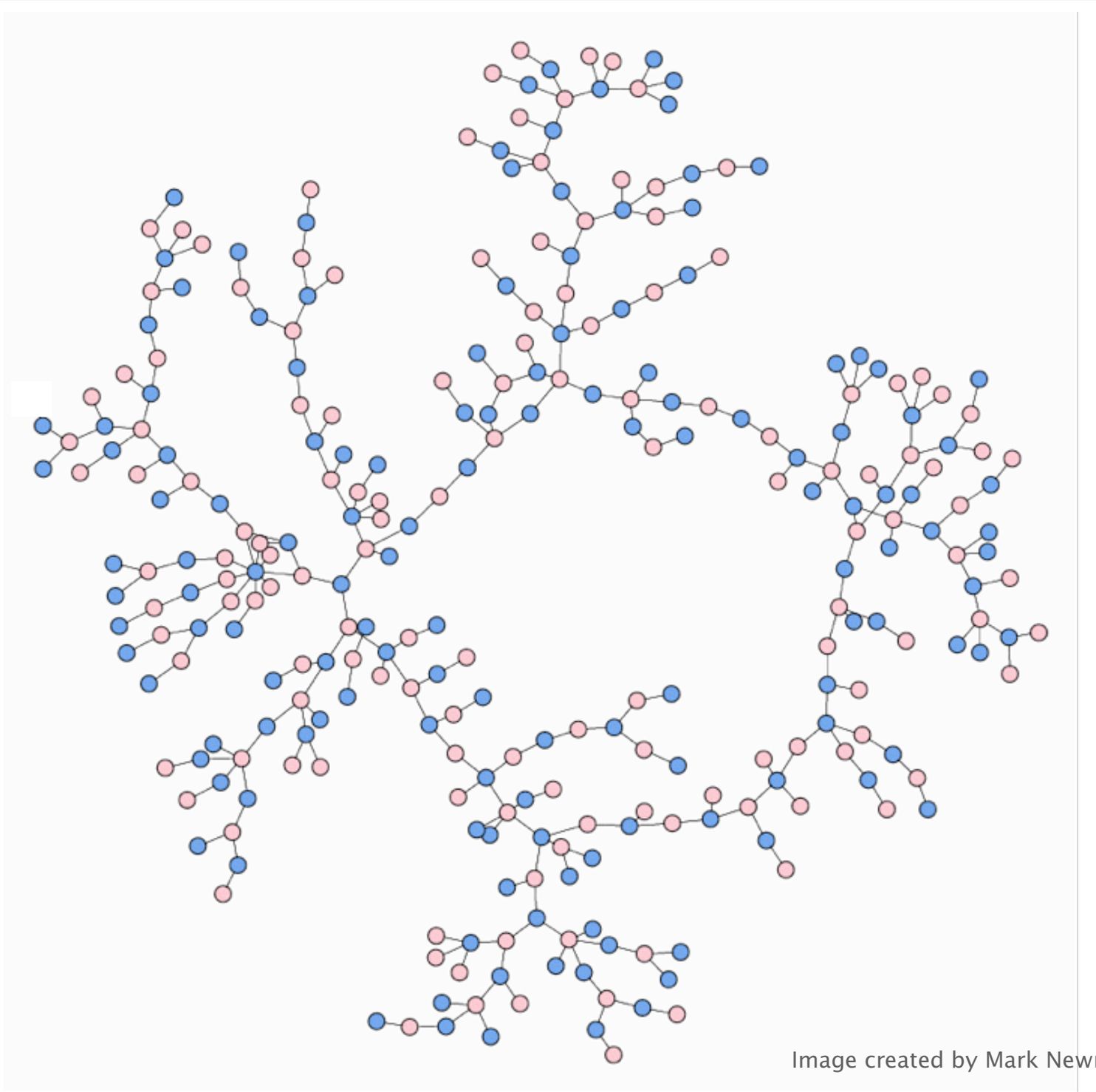
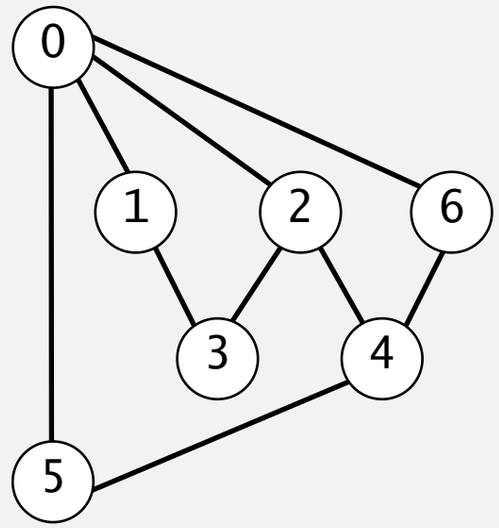


Image created by Mark Newman.

Graph-processing challenge 2

Problem. Find a cycle.



0-1
0-2
0-5
0-6
1-3
2-3
2-4
4-5
4-6

0-5-4-6-0

How difficult?

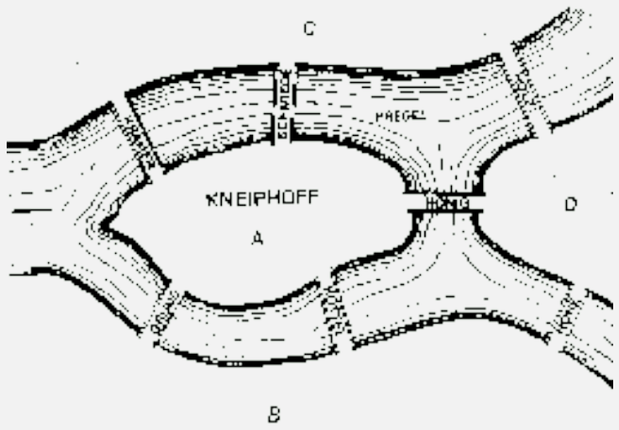
- Any programmer could do it.
- ✓ Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

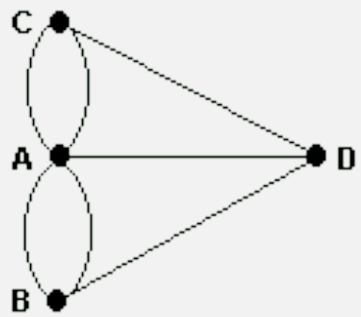
simple DFS-based solution
(see textbook)

Bridges of Königsberg

The Seven Bridges of Königsberg. [Leonhard Euler 1736]

> *" … in Königsberg in Prussia, there is an island A, called the Kneiphof; the river which surrounds it is divided into two branches ... and these branches are crossed by seven bridges. Concerning these bridges, it was asked whether anyone could arrange a route in such a way that he could cross each bridge once and only once. "*



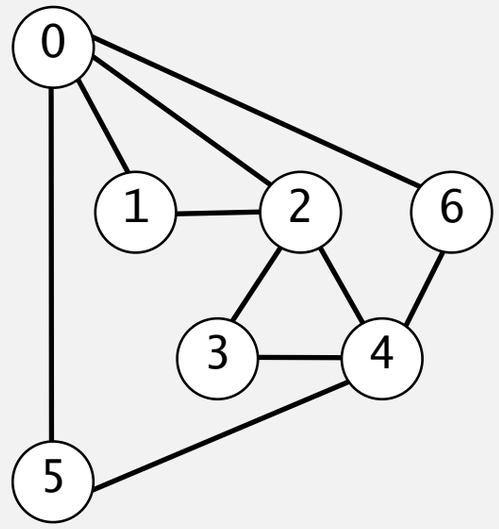


Euler cycle. Is there a (general) cycle that uses each edge exactly once?

Answer. A connected graph is Eulerian iff all vertices have even degree.

Graph-processing challenge 3

Problem. Find a (general) cycle that uses every edge exactly once.



0-1
0-2
0-5
0-6
1-2
2-3
2-4
3-4
4-5
4-6

0-1-2-3-4-2-0-6-4-5-0

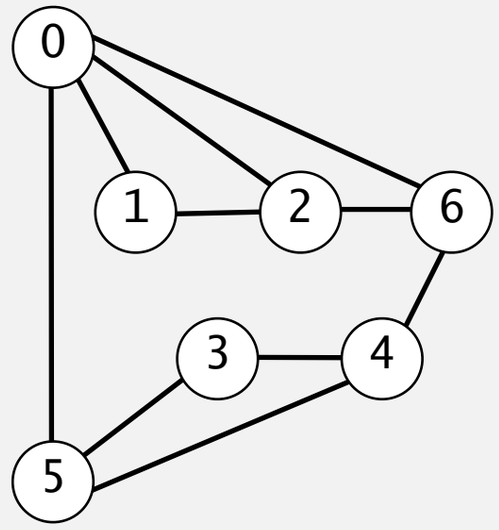
How difficult?

- Any programmer could do it.
- ✓ Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Euler cycle
(classic graph-processing problem)

Graph-processing challenge 4

Problem. Find a cycle that visits every vertex exactly once.



0-1
0-2
0-5
0-6
1-2
2-6
3-4
3-5
4-5
4-6

0-5-3-4-6-2-1-0

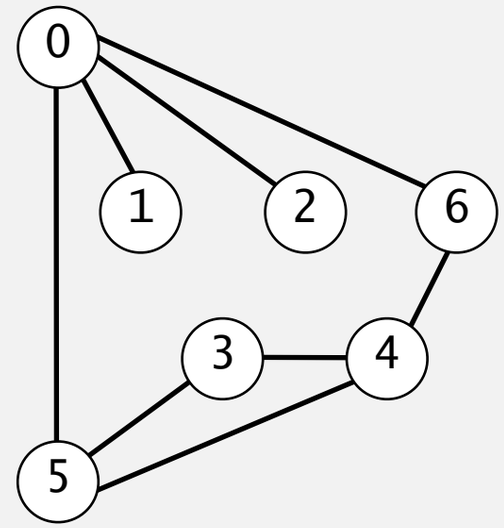
How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- ✓ Intractable.
- No one knows.
- Impossible.

Hamilton cycle
(classical NP-complete problem)

Graph-processing challenge 5

Problem. Are two graphs identical except for vertex names?

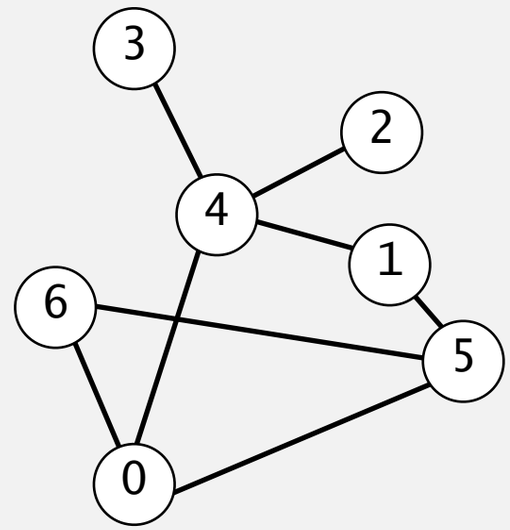


0-1
0-2
0-5
0-6
3-4
3-5
4-5
4-6

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- ✓ No one knows.
- Impossible.

graph isomorphism is longstanding open problem

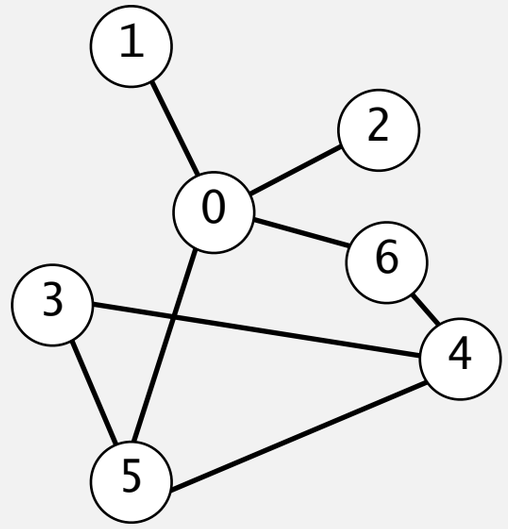


0-4
0-5
0-6
1-4
1-5
2-4
3-4
5-6

0↔4, 1↔3, 2↔2, 3↔6, 4↔5, 5↔0, 6↔1

Graph-processing challenge 6

Problem. Lay out a graph in the plane without crossing edges?



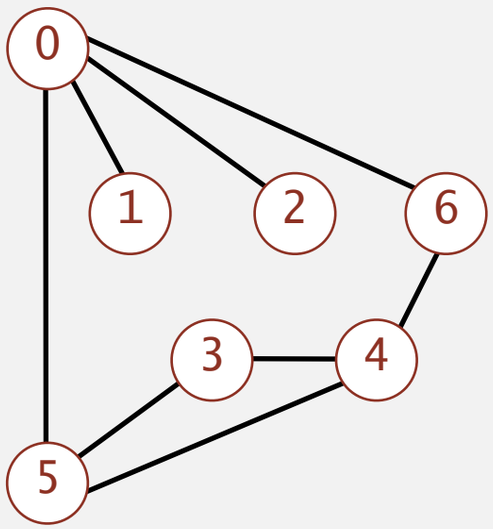
0-1
0-2
0-5
0-6
3-4
3-5
4-5
4-6

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.

✓
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

linear-time DFS-based planarity algorithm
discovered by Tarjan in 1970s
(too complicated for most practitioners)



Graph traversal summary

BFS and DFS enables efficient solution of many (but not all) graph problems.

problem	BFS	DFS	time
path between s and t	✔	✔	$E + V$
shortest path between s and t	✔		$E + V$
connected components	✔	✔	$E + V$
biconnected components		✔	$E + V$
cycle	✔	✔	$E + V$
Euler cycle		✔	$E + V$
Hamilton cycle			$2^{1.657\,V}$
bipartiteness	✔	✔	$E + V$
planarity		✔	$E + V$
graph isomorphism			$2^{c\sqrt{V \log V}}$ quase-poli Babai '15+ 2^(log V)^O(1)