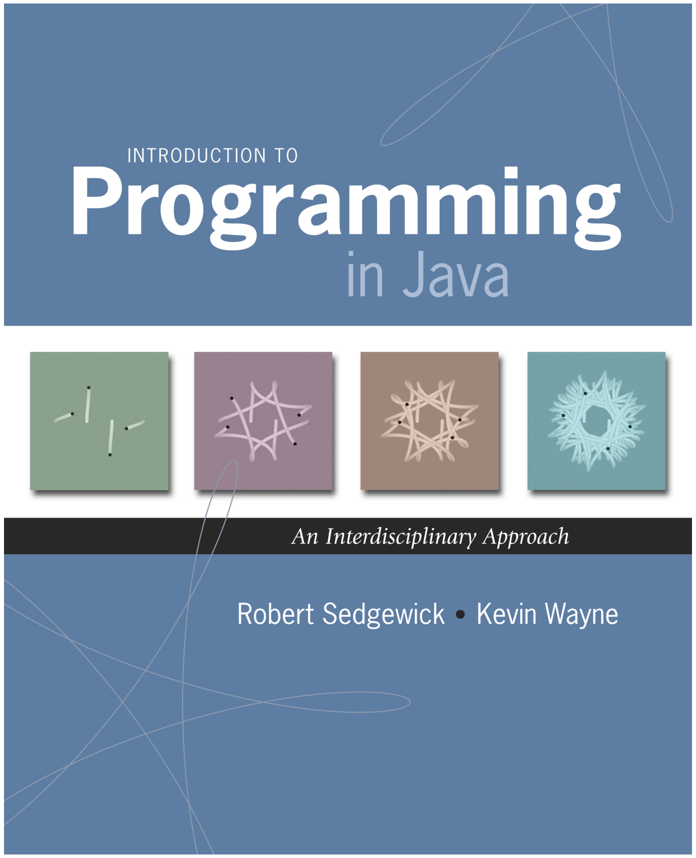
4.5 Small World Phenomenon



Small World Phenomenon

Small world phenomenon. Six handshakes away from anyone.

An experiment to quantify effect. [Stanley Milgram, 1960s]

- You are given personal info of another person.
- Goal: deliver message.
- Restriction: can only forward to someone you know by first name.
- Outcome: message delivered with average of 5 intermediaries.

e.g., occupation and age

Stanley Milgram

Kevin Bacon

Applications of Small World Phenomenon

Sociology applications.

- Looking for a job.
- Marketing products or ideas.
- Formation and spread of fame and fads.
- Train of thought followed in a conversation.
- Defining representative-ness of political bodies.
- Kevin Bacon game (movies, rock groups, facebook, etc.).

Other applications.

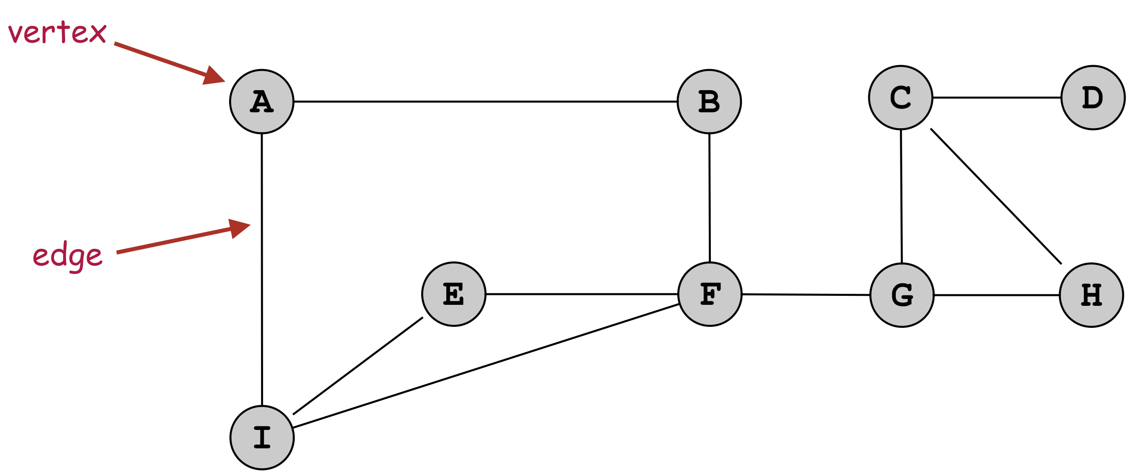
- Electronic circuits.
- Synchronization of neurons.
- Analysis of World Wide Web.
- Design of electrical power grids.
- Modeling of protein interaction networks.
- Phase transitions in coupled Kuramoto oscillators.
- Spread of infectious diseases and computer viruses.
- Evolution of cooperation in multi-player iterated Prisoner's Dilemma.

Reference. Duncan J. Watts, Small Worlds: The Dynamics of Networks between Order and Randomness, Princeton University Press, 1999.

Graph Data Type

Application demands a new data type.

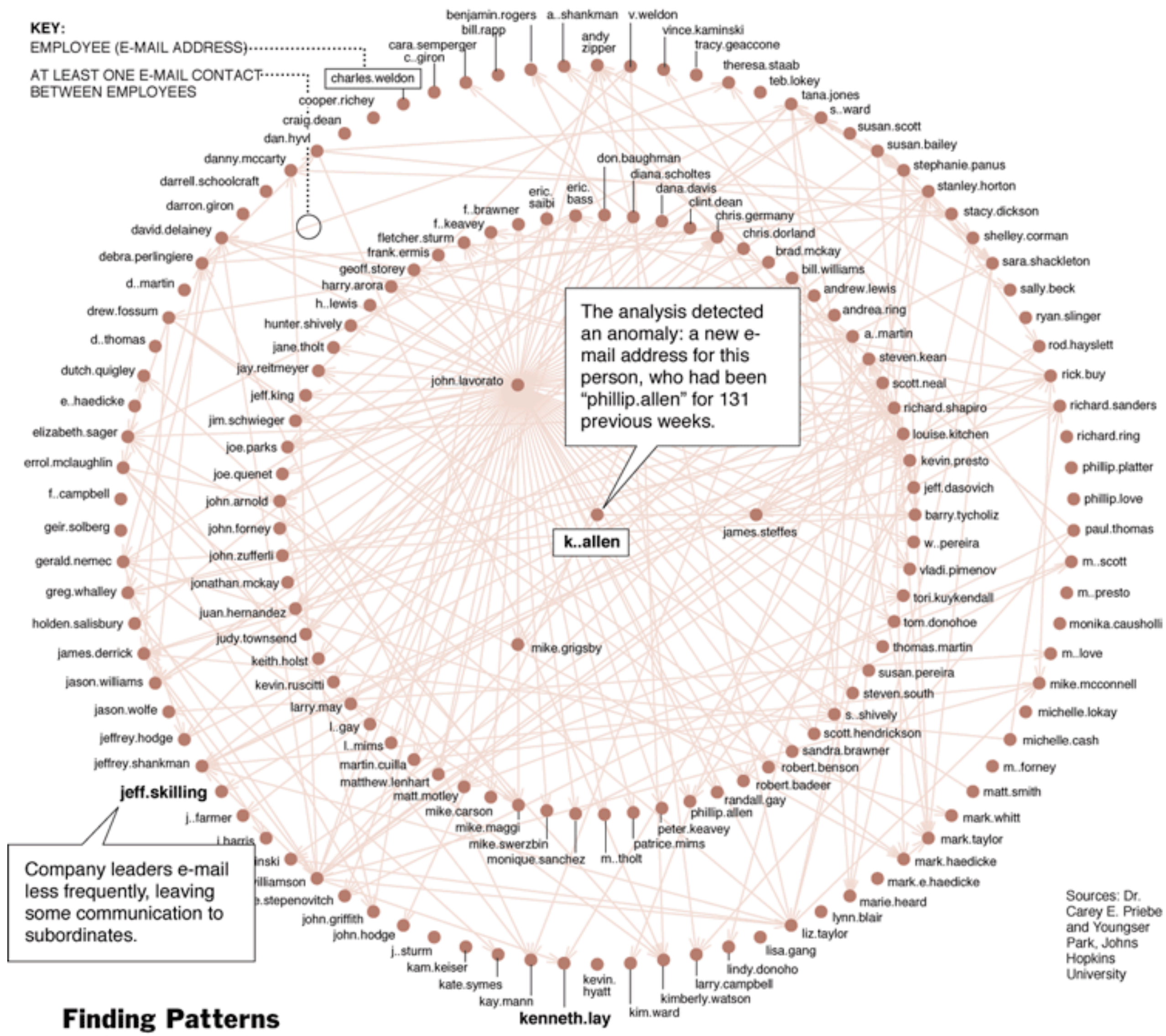
- Graph = data type that represents pairwise connections.
- Vertex = element.
- Edge = connection between two vertices.



Graph Applications

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

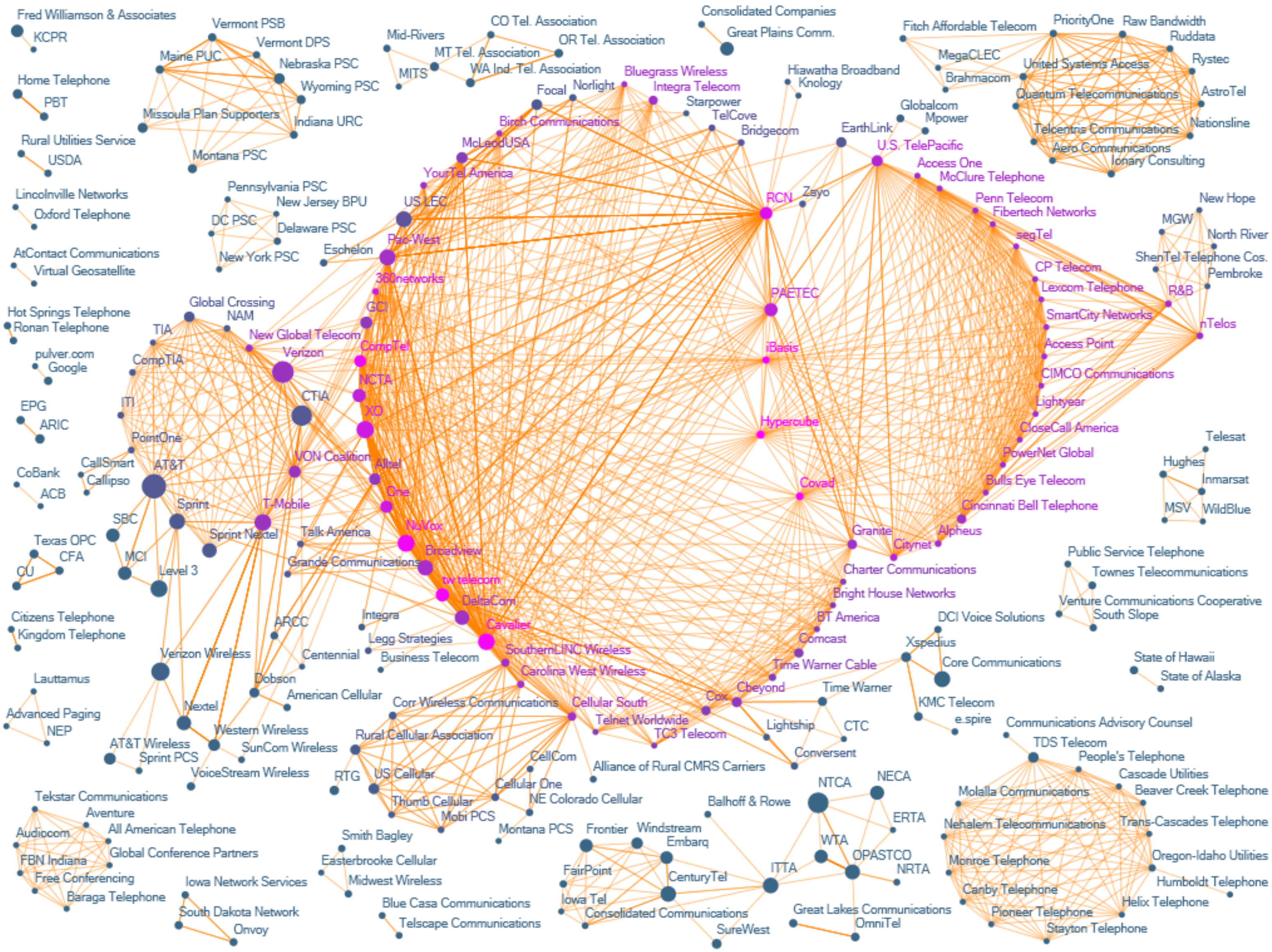
One Week of Enron Emails



Finding Patterns In Corporate Chatter

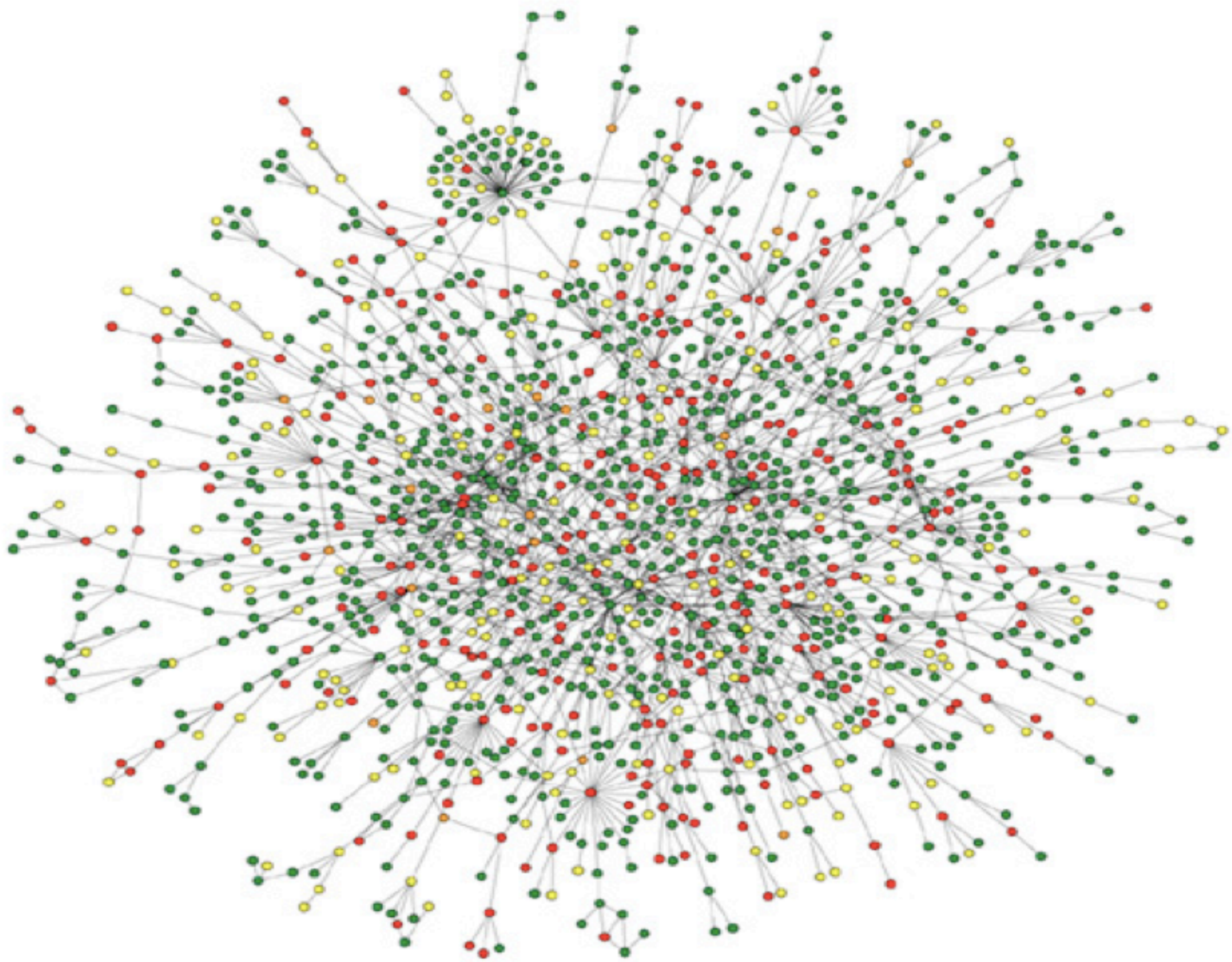
Computer scientists are analyzing about a half million Enron e-mails. Here is a map of a week's e-mail patterns in May 2001, when a new name suddenly appeared. Scientists found that this week's pattern differed greatly from others, suggesting different conversations were taking place that might interest investigators. Next step: word analysis of these messages.

FCC Lobbying Graph



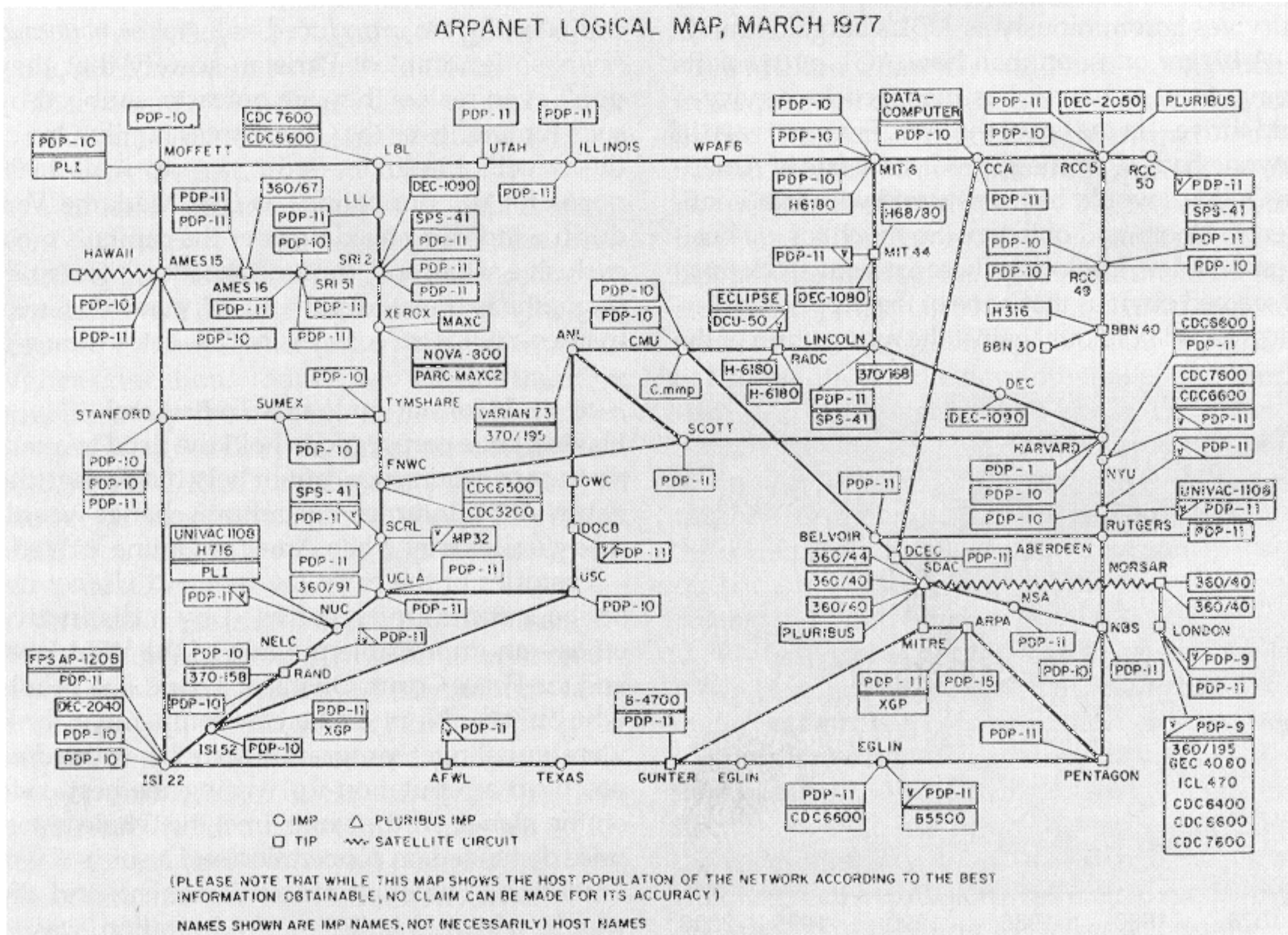
"The Evolution of FCC Lobbying Coalitions" by Pierre de Vries in JoSS Visualization Symposium 2010

Protein Interaction Network

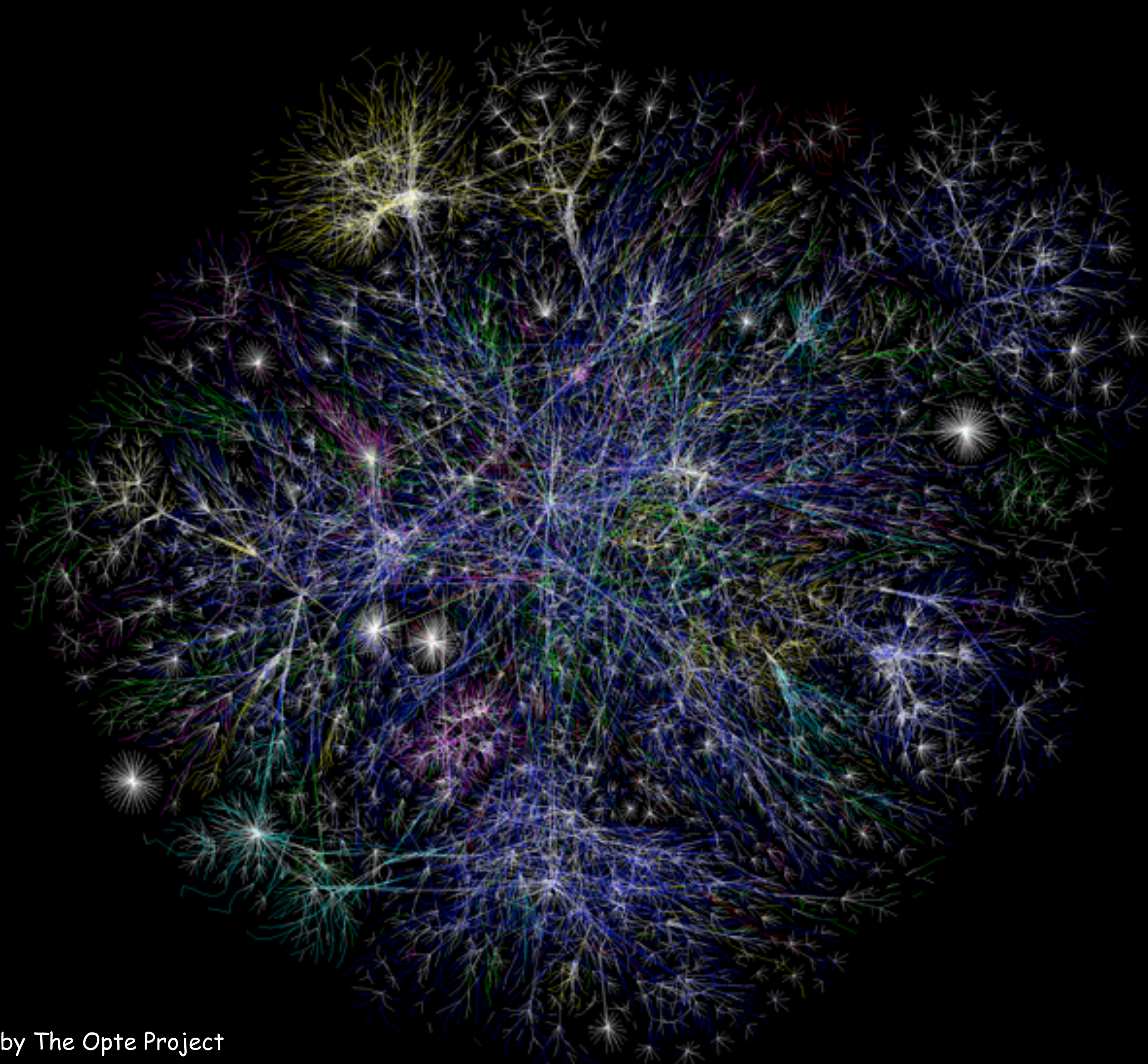


Reference: Jeong et al, Nature Review | Genetics

ARPANET


ARPANET LOGICAL MAP, MARCH 1977
MOFFETT
LBL
UTAH
ILLINOIS
WPAFB
MIT 6
CCA
RCC5
RCC 50
HAWAII
AMES 15
AMES 16
LLL
SRI 2
SRI 51
XEROX
MIT 44
RCC 49
ANL
CMU
RADC
LINCOLN
BBN 40
BBN 30
DEC
STANFORD
SUMEX
TYMSHARE
SCOTT
HARVARD
FNWC
GWC
NYU
SCRL
DOCB
BELVOIR
DCEC
ABERDEEN
RUTGERS
SDAC
NORSAR
UCLA
USC
NSA
NUC
ARPA
NBS
LONDON
NELC
RAND
MITRE
ISI 52
EGLIN
ISI 22
AFWL
TEXAS
GUNTER
EGLIN
PENTAGON
IMP
PLURIBUS IMP
TIP
SATELLITE CIRCUIT
(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)
NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

The Internet



The Internet as mapped by The Opte Project
http://www.opte.org

Internet Movie Database

Input format. Movie followed by list of performers, separated by slashes.

```
% more movies.txt
...
Tin Men (1987)/DeBoy, David/Blumenfeld, Alan/... /Geppi, Cindy/Hershey, Barbara
Tirez sur le pianiste (1960)/Heymann, Claude/.../Berger, Nicole (I)
Titanic (1997)Paxton, Bill/DiCaprio, Leonardo/.../Winslet, Kate
Titus (1999)/Weisskopf, Hermann/Rhys, Matthew/.../McEwan, Geraldine
To All a Good Night (1980)/George, Michael (II)/.../Gentile, Linda
To Be or Not to Be (1942)/Verebes, Ernö (I)/.../Lombard, Carole (I)
To Be or Not to Be (1983)/Brooks, Mel (I)/.../Bancroft, Anne
To Catch a Thief (1955)/París, Manuel/Grant, Cary/.../Kelly, Grace
To Die For (1989)/Bond, Steve (I)/Jones, Duane (I)/.../Maddalena, Julie
To Die For (1995)/Smith, Kurtwood/Kidman, Nicole/.../Tucci, Maria
To Die Standing (1990)/Sacha, Orlando/Anthony, Gerald/.../Rose, Jamie
To End All Wars (2001)/Kimura, Sakae/Ellis, Greg (II)/.../Sutherland, Kiefer
To Kill a Clown (1972)/Alda, Alan/Clavering, Eric/Lamberts, Heath/Danner, Blythe
To Live and Die in L.A. (1985)/McGroarty, Pat/Williams, Donnie/.../Dafoe, Willem
...
```

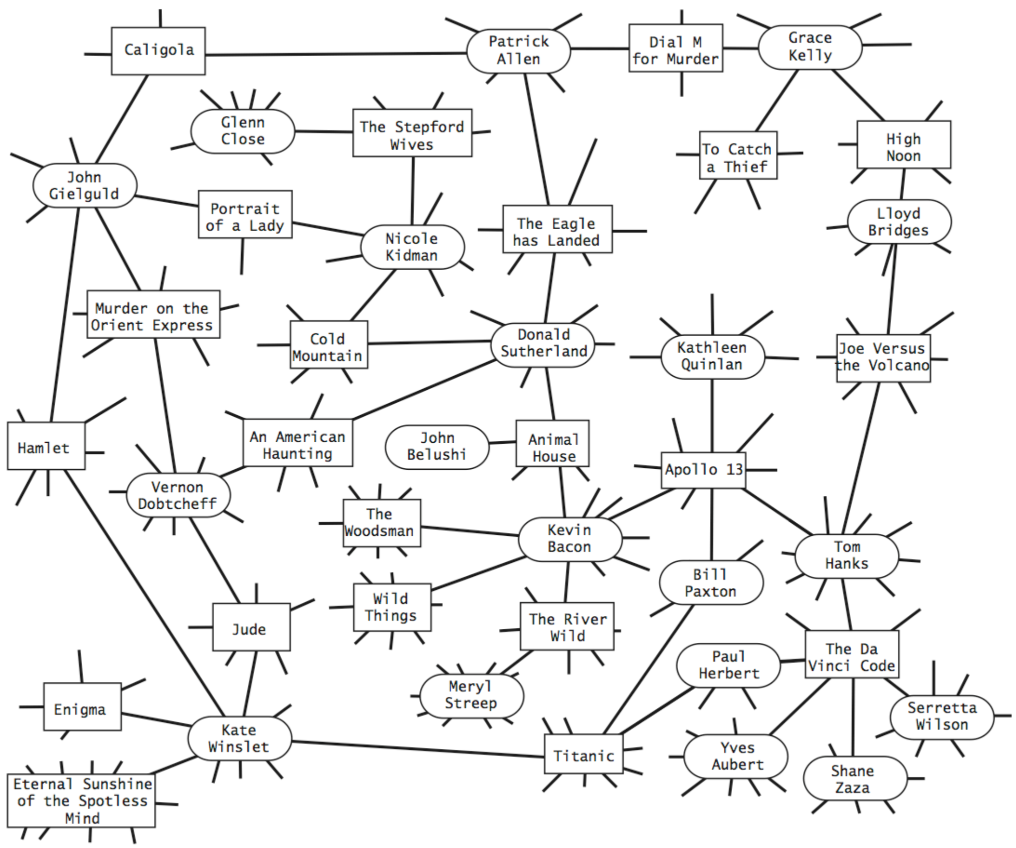
http://www.imdb.com/interfaces

Internet Movie Database

Q. How to represent the movie-performer relationships?

A. Use a graph.

- Vertex: performer or movie.
- Edge: connect performer to movie.

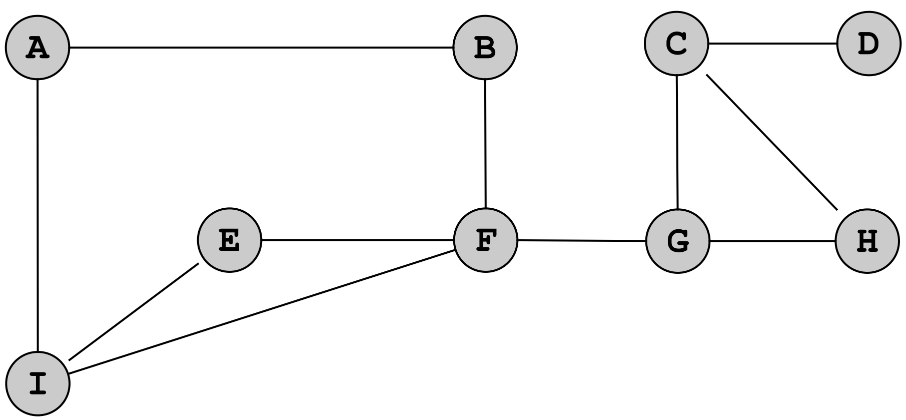


Graph API

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*

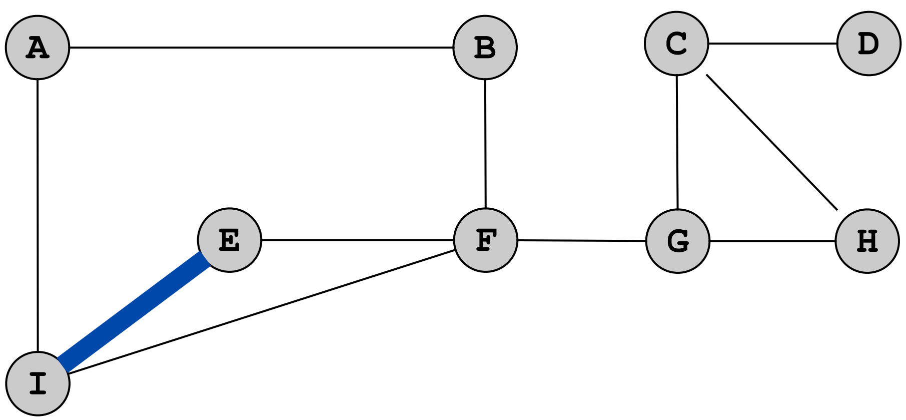
to support use with foreach




```
% more tiny.txt

A/B/I
B/A/F
C/D/G/H
D/C
E/F/I
F/B/E/G/I
G/C/F/H
H/C/G
I/A/E/F
```

Graph representation: use a symbol table.

- Key = name of vertex.
- Value = set of neighbors.



String key	SET<String> value
A	B I
B	A F
C	D G H
D	C
E	I F
F	E B G I
G	C F H
H	C G
I	A E F

symbol table

Set Data Type

Set data type. Unordered collection of distinct keys.

`public class SET<Key extends Comparable<Key>>`

	`SET()`	*create a set*
`boolean`	`isEmpty()`	*is the set empty?*
`void`	`add(Key key)`	*add* `key` *to the set*
`boolean`	`contains(Key key)`	*is* `key` *in the set?*

Note: Implementations should also implement the `Iterable<Key>` *interface to enable clients to access keys in sorted order with foreach loops*

Q. How to implement?

A. Identical to symbol table, but ignore values.

Graph Implementation

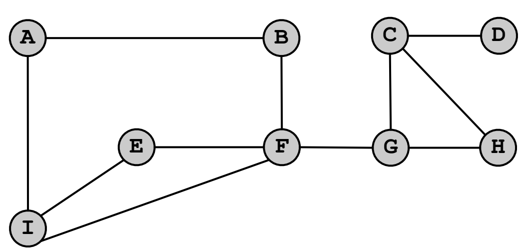
```
public class Graph {
   private ST<String, SET<String>> st;

   public Graph() {
      st = new ST<String, SET<String>>();
   }

   public void addEdge(String v, String w) {
      if (!st.contains(v)) addVertex(v);
      if (!st.contains(w)) addVertex(w);
      st.get(v).add(w);    ← add w to v's set of neighbors
      st.get(w).add(v);    ← add v to w's set of neighbors
   }

   private void addVertex(String v) {
      st.put(v, new SET<String>());   ← add new vertex v
   }                                    with no neighbors

   public Iterable<String> adjacentTo(String v) {
      return st.get(v);
   }
}
```

Graph Implementation (continued)

Second constructor. To read graph from input stream.

```
public Graph(In in) {
   st = new ST<String, SET<String>>();
   while (!in.isEmpty()) {
      String line = in.readLine();
      String[] names = line.split("/");
      for (int i = 1; i < names.length; i++)
         addEdge(names[0], names[i]);
   }
}
```

```
In in = new In("tiny.txt");
Graph G = new Graph(G, in);
```




```
% more tiny.txt

A/B/I
B/A/F
C/D/G/H
D/C
E/F/I
F/B/E/G/I
G/C/F/H
H/C/G
I/A/E/F
```

Graph Client: Movie Finder

Performer and movie queries.

- Given a performer, find all movies in which they appeared.
- Given a movie, find all performers.

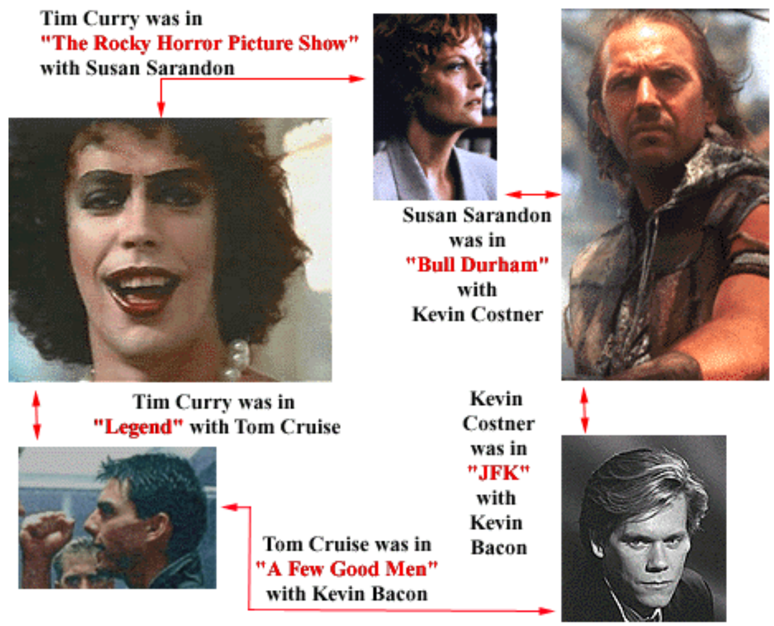
```
public class MovieFinder {
   public static void main(String[] args) {

      In in    = new In(args[0]);        ← read in graph from a file
      Graph G = new Graph(in);

      while (!StdIn.isEmpty()) {          ← process queries
         String v = StdIn.readLine();
         for (String w : G.adjacentTo(v))
             StdOut.println(w);
      }

   }
}
```

Graph Client: Movie Finder

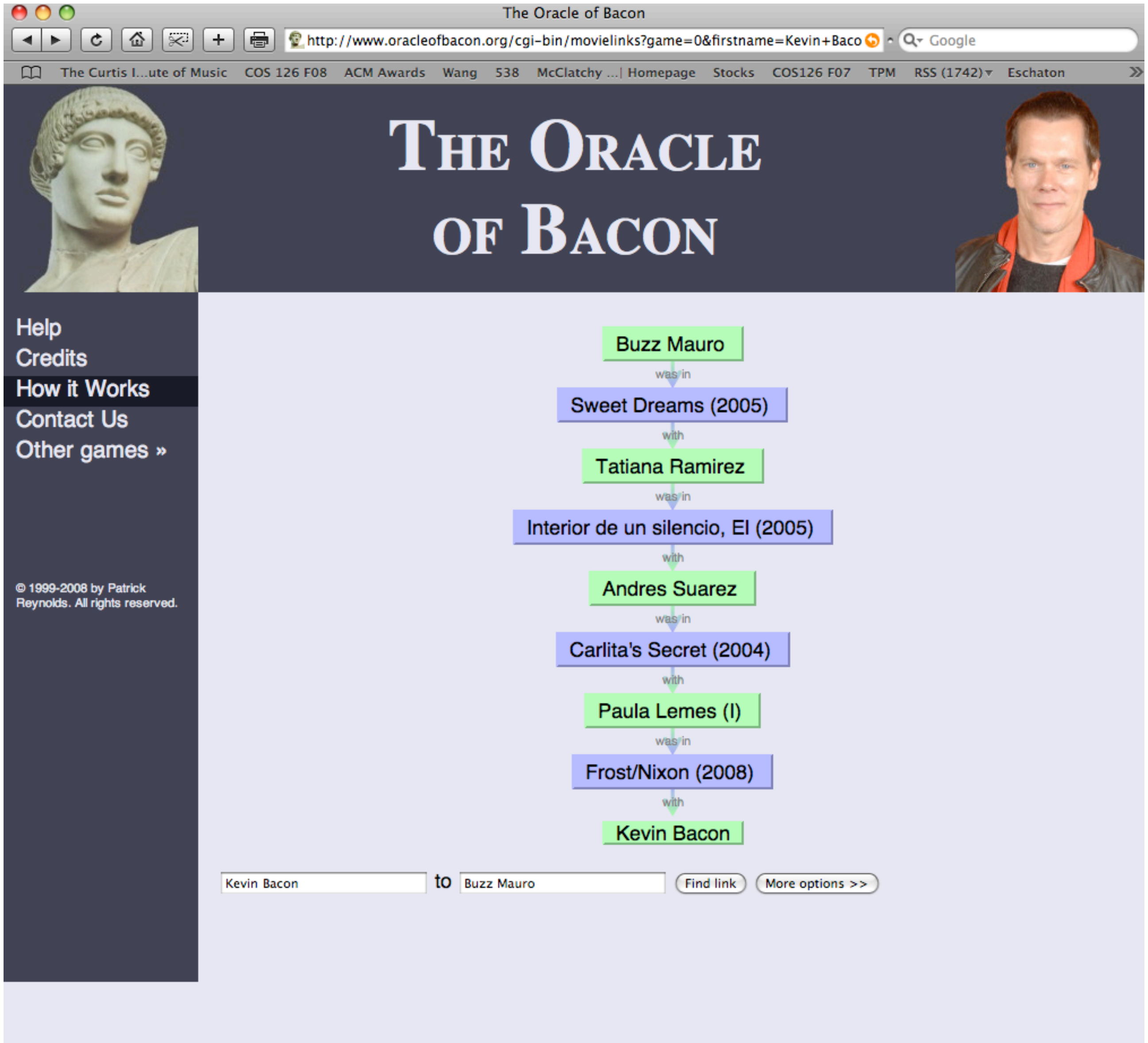
```
% java MovieFinder action.txt
Bacon, Kevin
Death Sentence (2007)
River Wild, The (1994)
Tremors (1990)

Roberts, Julia
Blood Red (1989)
I Love Trouble (1994)
Mexican, The (2001)
Ocean's Eleven (2001)

Tilghman, Shirley
```

```
% java MovieFinder mpaa.txt
Bacon, Kevin
Air I Breathe, The (2007)
Air Up There, The (1994)
Animal House (1978)
Apollo 13 (1995)
Balto (1995)
Beauty Shop (2005)
Big Picture, The (1989)
…
Sleepers (1996)
Starting Over (1979)
Stir of Echoes (1999)
Telling Lies in America (1997)
Trapped (2002)
Tremors (1990)
We Married Margo (2000)
Where the Truth Lies (2005)
White Water Summer (1987)
Wild Things (1998)
Woodsman, The (2004)
```

Kevin Bacon Numbers



Oracle of Kevin Bacon



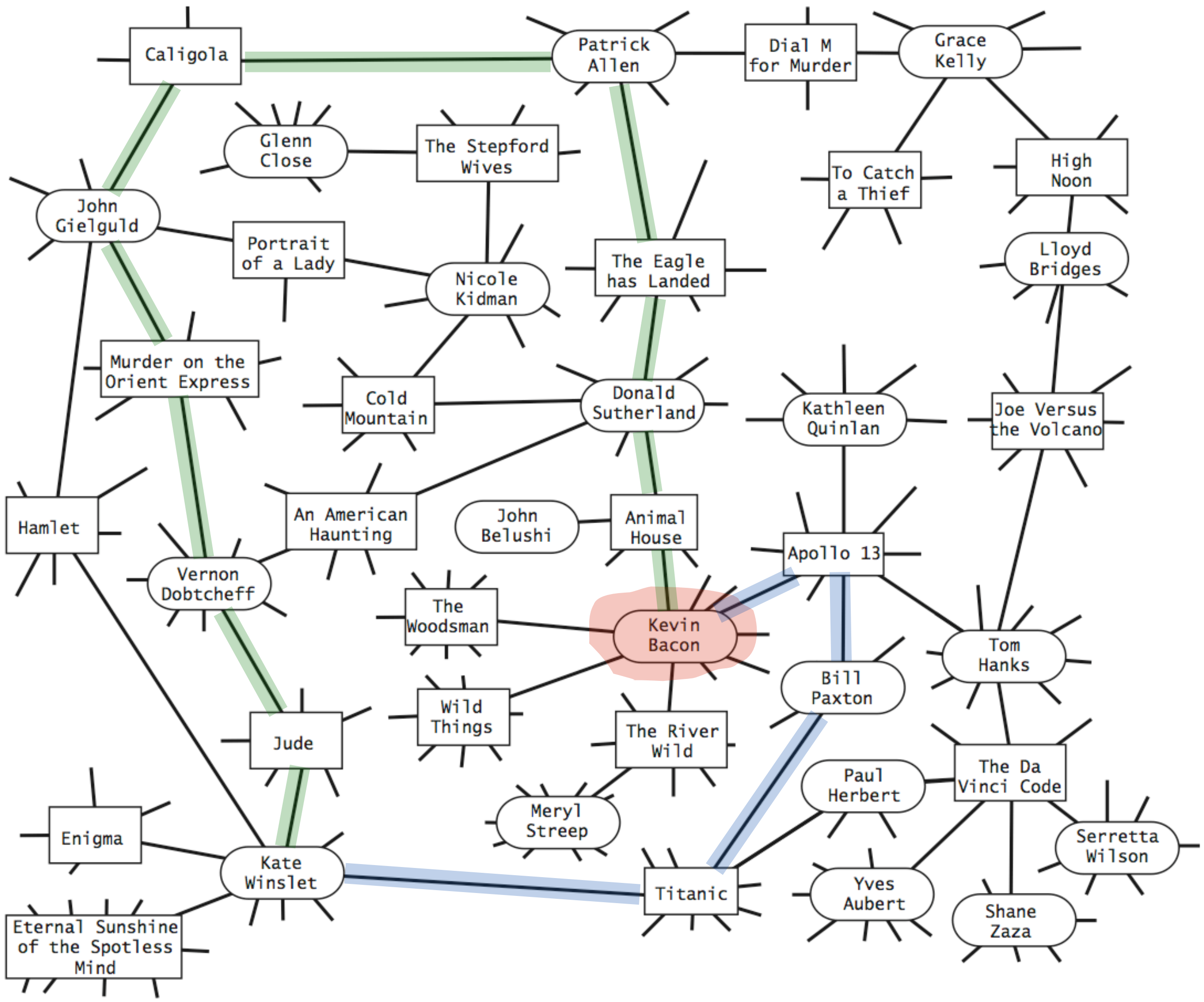
Kevin Bacon Game

Game. Given an actor or actress, find shortest chain of movies connecting them to Kevin Bacon.

Actor	Was in	With
Whoopi Goldberg	Ghost	Patrick Swayze
Patrick Swayze	Dirty Dancing	Jennifer Gray
Jennifer Gray	Ferris Beuller's Day Off	Matthew Broderick
Matthew Broderick	The Road to Wellville	John Cusack
John Cusack	Bullets Over Broadway	Dianne West
Dianne West	Footloose	Kevin Bacon
Kevin Bacon		

Computing Bacon Numbers

How to compute. Find shortest path in performer-movie graph.



PathFinder API

PathFinder API.

public class PathFinder		
	`PathFinder(Graph G, String s)`	*constructor*
`int`	`distanceTo(String v)`	*length of shortest path from* s *to* v *in* G
`Iterable<String>`	`pathTo(String v)`	*shortest path from* s *to* v *in* G

Design principles.

- Decouple graph algorithm from graph data type.
- Avoid feature creep: don't encrust Graph with search features; instead make a new datatype.

Computing Bacon Numbers: Java Implementation

```
public class Bacon {
   public static void main(String[] args) {

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

      String s = "Bacon, Kevin";
      PathFinder finder = new PathFinder(G, s);

      while (!StdIn.isEmpty()) {
         String performer = StdIn.readLine();
         for (String v : finder.pathTo(s))
            StdOut.println(v);
      }
   }
}
```

← read in the graph from a file

← create object to return shortest paths

← process queries

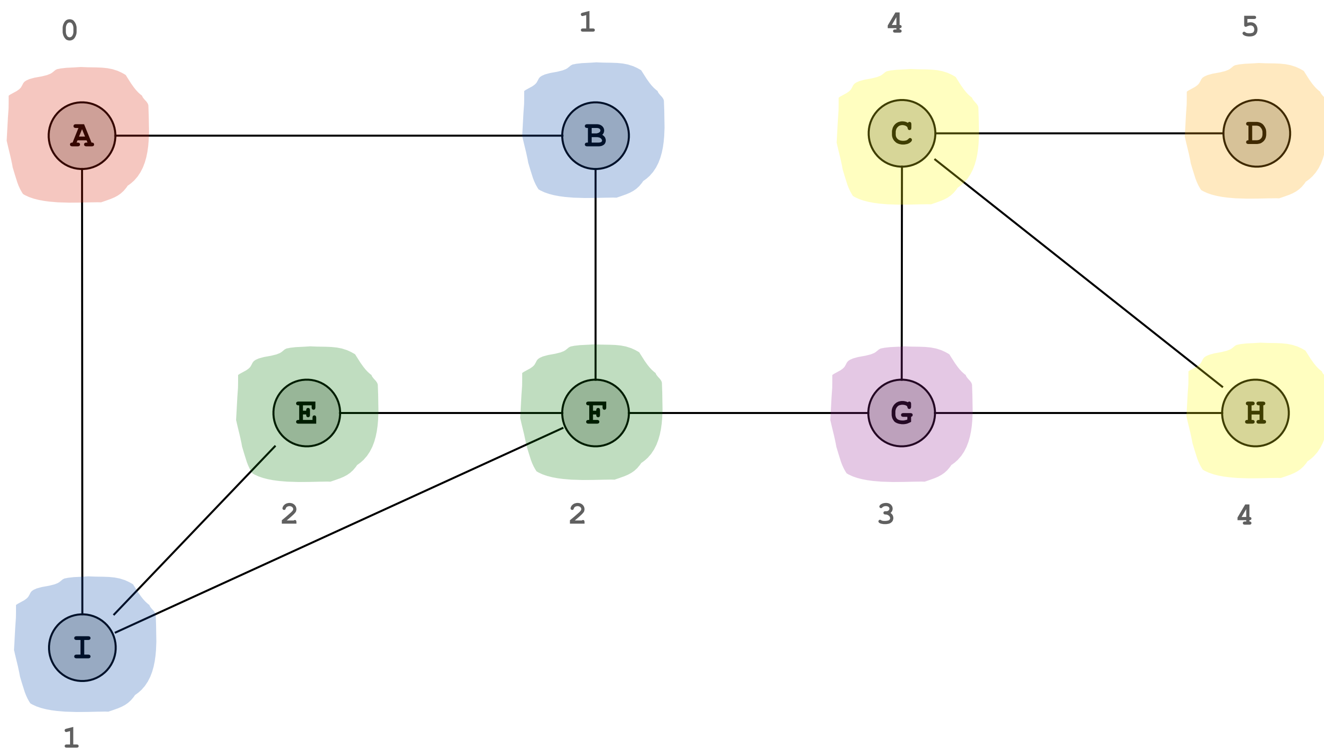
```
% java Bacon top-grossing.txt
Stallone, Sylvester
Rocky III (1982)
Tamburro, Charles A.
Terminator 2: Judgment Day (1991)
Berkeley, Xander
Apollo 13 (1995)
Bacon, Kevin
```

```
% java Bacon top-grossing.txt
Goldberg, Whoopi
Sister Act (1992)
Grodénchik, Max
Apollo 13 (1995)
Bacon, Kevin

Tilghman, Shirley
```

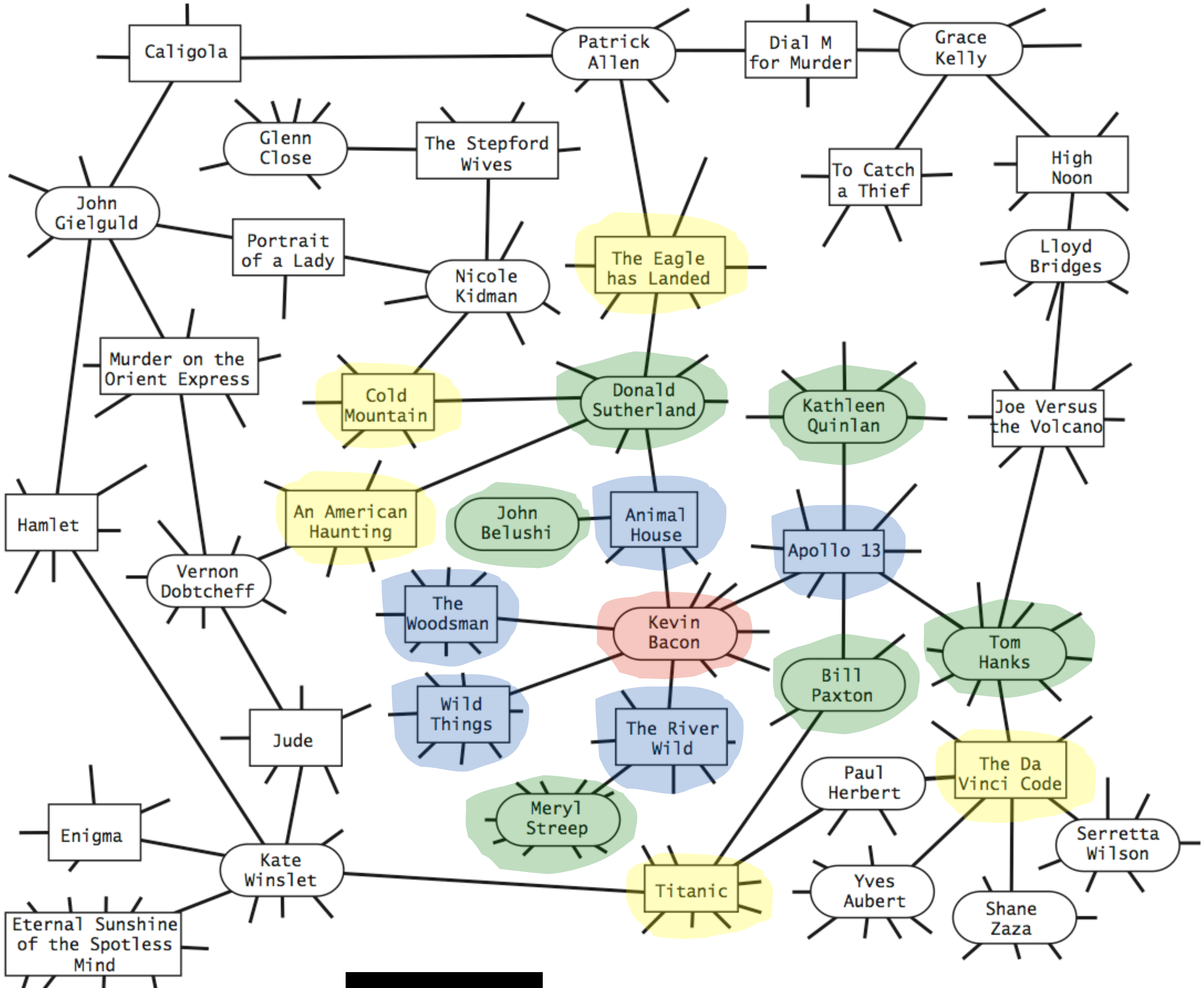
Computing Shortest Paths

To compute shortest paths:

- Source vertex is at distance 0.
- Its neighbors are at distance 1.
- Their remaining neighbors are at distance 2.
- Their remaining neighbors are at distance 3.
- ...



Computing Shortest Paths



distance = 3

Breadth First Search

Goal. Given a vertex **s**, find shortest path to every other vertex **v**.

BFS from source vertex **s**

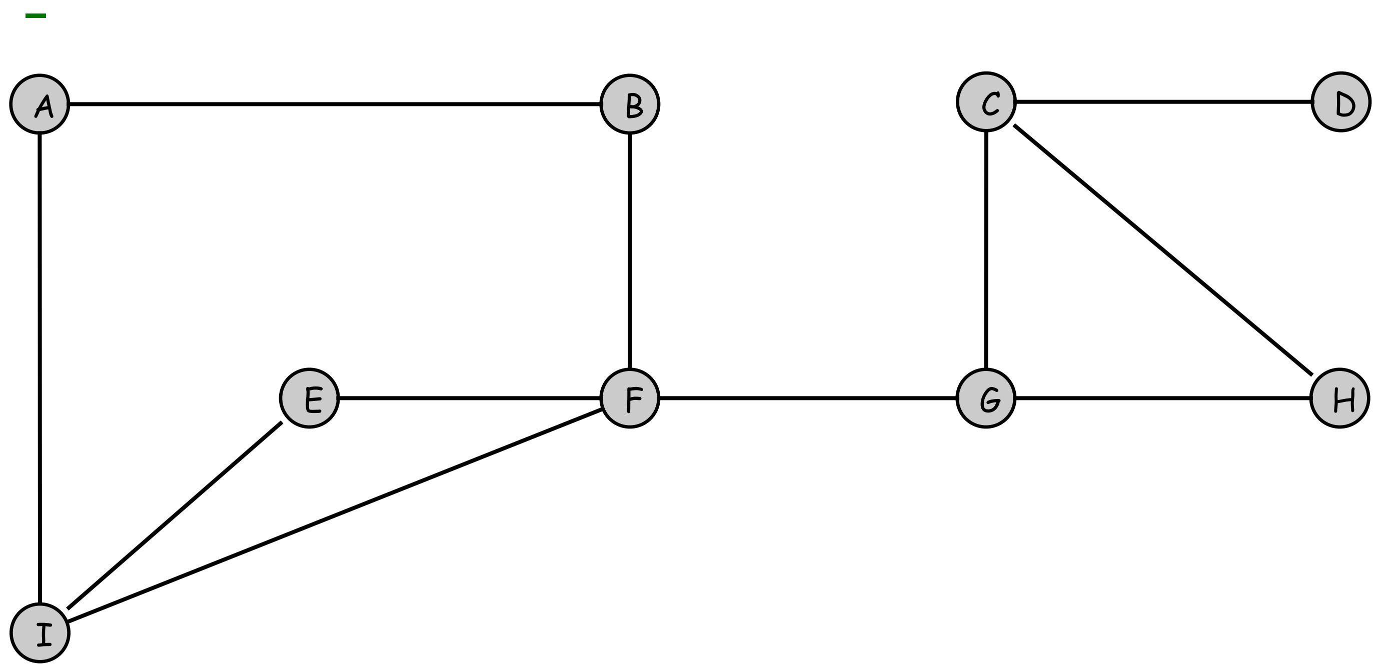
Put **s** onto a FIFO queue.

Repeat until the queue is empty:

- dequeue the least recently added vertex **v**
- add each of **v**'s unvisited neighbors to the queue, and mark them as visited.

Key observation. Vertices are visited in increasing order of distance from s because we use a FIFO queue.

Breadth First Search

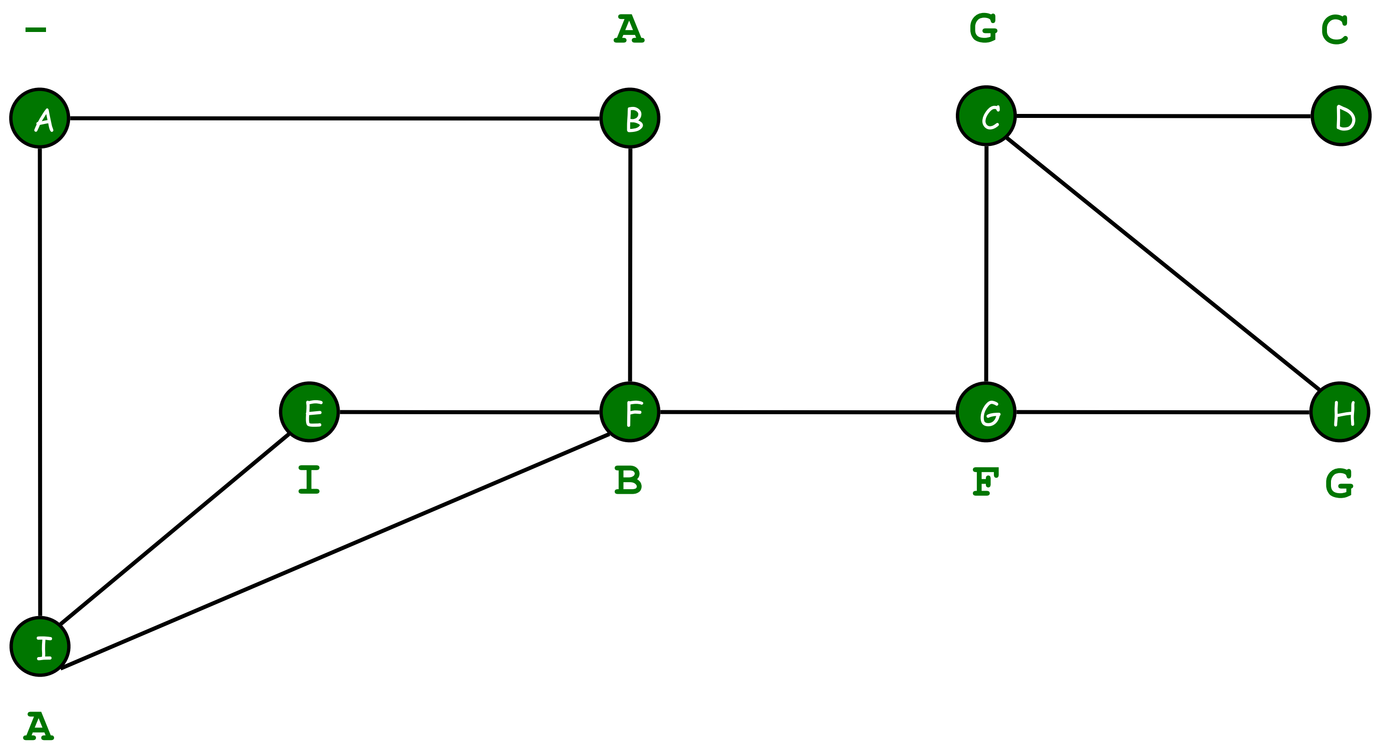


front

FIFO Queue

33

Breadth First Search



STOP

front

FIFO Queue

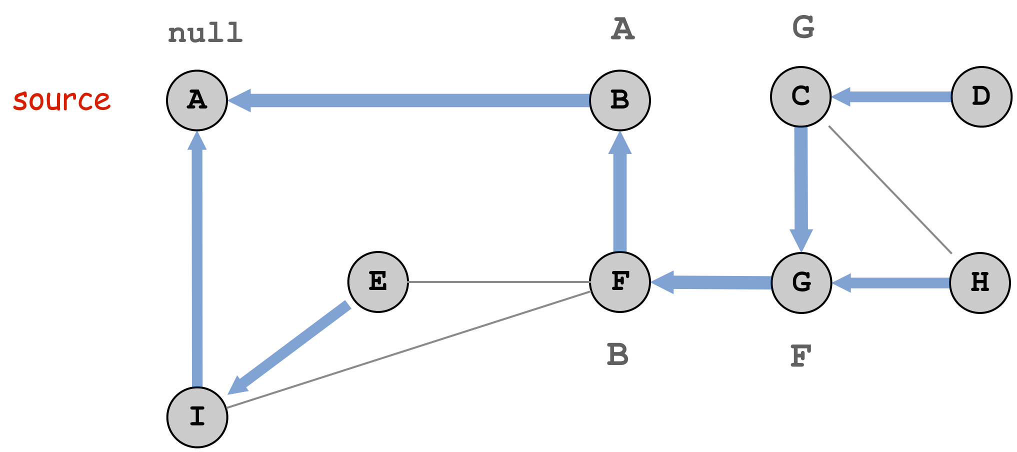
Breadth First Searcher: Preprocessing

```
public class PathFinder {
   private ST<String, String>  prev = new ST<String, String>();
   private ST<String, Integer> dist = new ST<String, Integer>();

   public PathFinder(Graph G, String s) {
      Queue<String> q = new Queue<String>();
      q.enqueue(s);
      dist.put(s, 0);
      while (!q.isEmpty()) {
         String v = q.dequeue();
         for (String w : G.adjacentTo(v)) {
            if (!dist.contains(w)) {
               q.enqueue(w);
               dist.put(w, 1 + dist.get(v));
               prev.put(w, v);
            }
         }
      }
   // other PathFinder methods go here
   }
```

Breadth First Searcher: Finding the Path

To find shortest path: follow `prev[]` from vertex `v` back to source `s`.

- Consider vertices: `v`, `prev[v]`, `prev[prev[v]]`, ..., `s`.
- Ex: shortest path from `C` to `A`: `C - G - F - B - A`



key	prev	dist
A	-	0
B	A	1
C	G	4
D	C	5
E	I	2
F	B	2
G	F	3
H	G	4
I	A	1

symbol tables

```
public Iterable<String> pathTo(String v) {
   Stack<String> path = new Stack<String>();
   while (dist.contains(v)) {
      path.push(v);
      v = prev.get(v);
   }
   return path;
}
```

Running Time Analysis

Analysis. BFS scales to solve huge problems.

data File	movies	performers	edges	read input	build graph	BFS	pathTo
G.txt	1,288	21,177	28K	0.26 sec	0.52 sec	0.32 sec	0 sec
PG13.txt	2,538	70,325	100K	0.31 sec	0.99 sec	0.72 sec	0 sec
action.txt	14,938	139,861	270K	0.72 sec	2.8 sec	2.0 sec	0 sec
mpaa.txt	21,861	280,624	610K	2.1 sec	7.5 sec	5.5 sec	0 sec
all.txt	285,462	933,864	3.3M	15 sec	56 sec	39 sec	0 sec

60MB

data as of April 9, 2007

Data Analysis

Exercise. Compute histogram of Kevin Bacon numbers.
Input. 285,462 movies, 933,864 actors.

Bacon #	Frequency
0	1
1	2,249
2	218,088
3	561,161
4	111,149
5	7,905
6	903
7	100
8	14
∞	32,294

data as of April 9, 2007

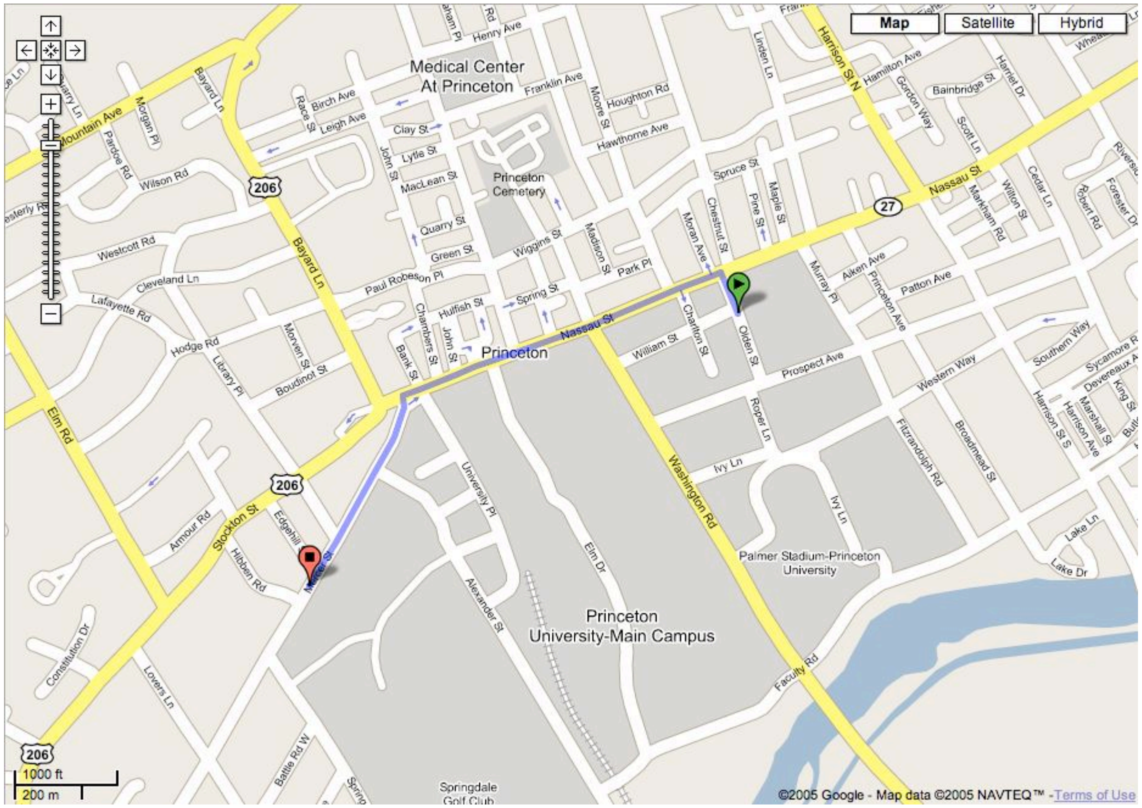
Buzz Mauro, Jessica Drizd, Pablo Capussi
Argentine short film *Sweet Dreams* (2005) → 8

Fred Ott, solo actor in
Fred Ott Holding a Bird (1894) → ∞

More BFS applications.

- Particle tracking.
- Image processing.
- Crawling the Web.
- Routing Internet packets.
- ...

Extensions. Google maps.



Erdös Numbers

Erdös Numbers

Paul Erdös. Legendary, brilliant, prolific mathematician who wrote over 1500 papers!

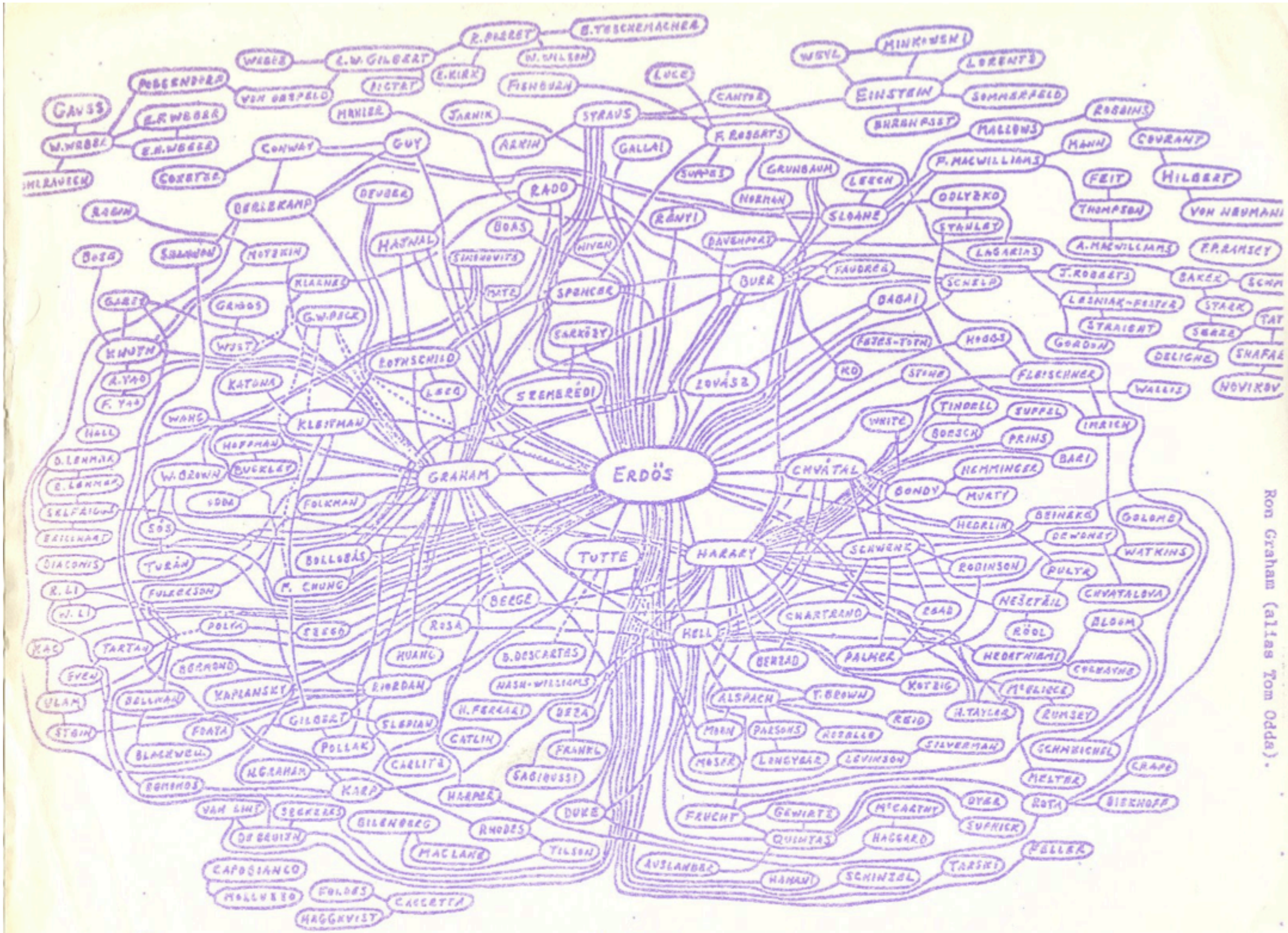
What's your Erdös number?

- Co-authors of a paper with Erdös: 1.
- Co-authors of those co-authors: 2.
- And so on …

Paul Erdös (1913-1996)

Erdös #	Frequency
0	1
1	502
2	5,713
3	26,422
4	62,136
5	66,157
6	32,280
7	10,431
8	3,214
9	953
10	262
11	94
12	23
13	4
14	7
15	1
∞	4 billion +

Erdös Graph



ERDÖS
GRAHAM
TUTTE
HARARY
HELL
CHVÁTAL
LOVÁSZ
SZEMERÉDI
RADO
HAJNAL
BERLEKAMP
CONWAY
GUY
COXETER
GAUSS
W. WEBER
E.R. WEBER
E.H. WEBER
WEYL
MINKOWSKI
LORENTZ
EINSTEIN
SOMMERFELD
EHRENFEST
LUCE
CANTOR
FISHBURN
STRAUS
F. ROBERTS
GALLAI
ARKIN
JARNIK
MAHLER
ROBBINS
COURANT
HILBERT
VON NEUMANN
MALLOWS
MANN
F. MACWILLIAMS
FEIT
THOMPSON
ODLYZKO
STANLEY
SLOANE
LEECH
GRUNBAUM
NORMAN
SUPPES
RÉNYI
BOAS
DAVENPORT
BURR
FAUDREE
SCHELP
J. ROBERTS
A. MACWILLIAMS
F.P. RAMSEY
LAGARIAS
NIVEN
SIMONOVITS
SPENCER
SÁRKÖZY
G.W. PECK
GRIGGS
WEST
ROTHSCHILD
KATONA
KLEITMAN
KNUTH
A. YAO
F. YAO
HALL
D. LEHMER
E. LEHMER
SELFRIDGE
DIACONIS
R. LI
W. LI
KAC
ULAM
STEIN
SÓS
TURÁN
W. BROWN
HOFFMAN
FOLKMAN
BOLLOBÁS
F. CHUNG
BERGE
ROSA
HUANG
B. DESCARTES
NASH-WILLIAMS
RIORDAN
KAPLANSKY
GILBERT
SLEPIAN
POLLAK
CATLIN
DEZA
FRANKL
CARLITZ
SABIDUSSI
KARP
HARPER
RHODES
DUKE
TILSON
MACLANE
VAN LINT
DE BRUIJN
SZEKERES
CAPOBIANCO
HAGGKVIST
KO
STONE
FLEISCHER
GORDON
DELIGNE
WALLIS
NOVIKOV
WHITE
IMRICH
PRINS
HEMMINGER
BONDY
MURTY
SCHWENK
WATKINS
ROBINSON
NEŠETŘIL
CHARTRAND
BLOOM
PALMER
ALSPACH
T. BROWN
REID
H. TAYLOR
RUMSEY
MOON
PARSONS
MOSER
LONGYEAR
SILVERMAN
MELTER
FRUCHT
GEWIRTZ
QUINTAS
McCARTHY
HAGGARD
ROTA
AUSLANDER
HANANI
SCHINZEL
TARSKI
FELLER
Ron Graham (alias Tom Odda).

Conclusions

Linked list. Ordering of elements.
Binary tree. Hierarchical structure of elements.
Graph. Pairwise connections between elements.

Data structures.

- Queue: linked list.
- Set: binary tree.
- Symbol table: binary tree.
- Graph: symbol table of sets.
- Breadth first searcher: graph + queue + symbol table.

Importance of data structures.

- Enables us to build and debug large programs.
- Enables us to solve large problems efficiently.