4.5 Small World Phenomenon

Stanley Milgram

Kevin Bacon

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

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 representativeness of political bodies.
- Kevin Bacon game (movies, rock groups, facebook, etc.).

Other applications.

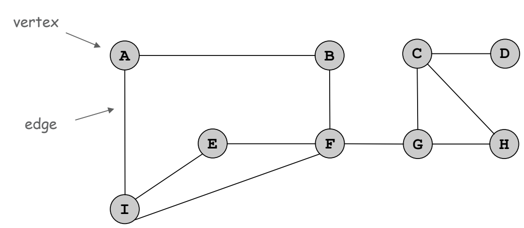
- Electronic circuits.
- Synchronization of neurons.
- Analysis of World Wide Web.
- Design of electrical power grids.
- Modeling of protein interaction networks.
- Phase transitions in coupled Kuramoto oscillators.
- Spread of infectious diseases and computer viruses.
- Evolution of cooperation in multi-player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

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



Applications of Graphs

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

Internet Movie Database

Actor and movie queries.
- Given an actor, find all movies in which they appeared.
- Given a movie, find all actors.

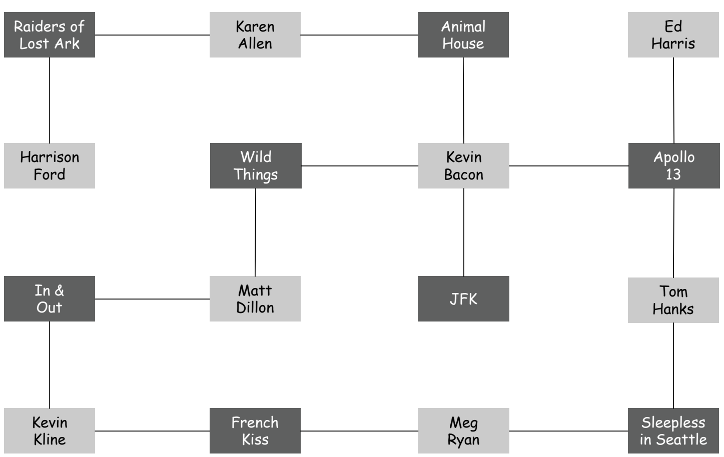
Input format. Movie followed by list of actors, separated by slashes.

```
Wild Things (1998)/Bacon, Kevin/Campbell, Neve/Dillon, Matt/Murray, Bill/Richards, Denise
JFK (1991)/Asner, Edward/Bacon, Kevin/Costner, Kevin/Jones, Tommy Lee/Grubbs, Gary
Braveheart (1995)/Gibson, Mel//Marceau, Sophie/McGoohan, Patrick/Hanly, Peter
...
```

Reference: http://www.imdb.com/interfaces

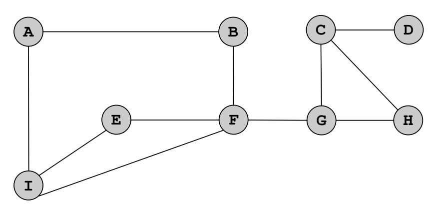
Q. How to represent the actor-movie relationships?
A. Use a graph.
- Vertices: actors, movies.
- Edges: connect actor with each movie in which they appear.

Actor-Movie Graph (Partial)



Graph API

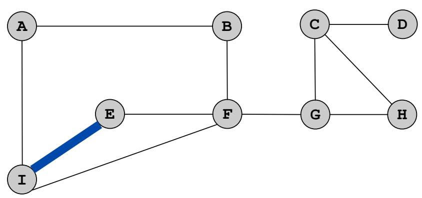
	`public class Graph<Vertex>`	(graph data type)
	`Graph()`	create an empty graph
`void`	`addEdge(Vertex v, Vertex w)`	add edge v-w
`String`	`addVertex(Vertex v)`	add vertex v
`Iterable<Vertex>`	`adj(Vertex v)`	return an iterator over the neighbors of v



Graph Representation

Graph representation: use a symbol table.

- Key = name of vertex (e.g., movie or actor).
- Value = set of neighbors.



Symbol Table

Key	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
String	SET

Graph Implementation

```
public class Graph<Vertex> {
   private ST<Vertex, SET<Vertex>> st =
       new ST<Vertex, SET<Vertex>>();

   public void addEdge(Vertex v, Vertex 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
   }

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

   public Iterable<Vertex> adj(Vertex v) {
      return st.get(v);
   }
}
```


Graph Client Warmup: Movie Finder

Movie finder. Given actor, find all movies in which they appeared.

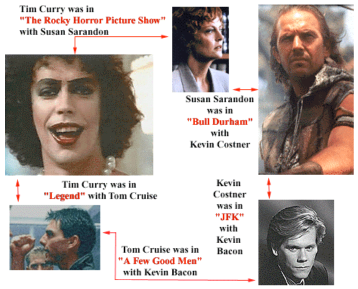
```
public class MovieFinder {
   public static void main(String[] args) {
                                                    build graph
      Graph<String> G = new Graph<String>();
      In in = new In(args[0]);    ← file input
      while (!in.isEmpty()) {
         String line = in.readLine();
         String[] names = line.split("/");   ← tokenize input line
         String movie = names[0];
         for (int i = 1; i < names.length; i++)
            G.addEdge(movie, names[i]);  ← movie-actor edge
      }
                                                    print all of actor's movies
      while (!StdIn.isEmpty()) {
         String actor = StdIn.readLine();
         for (String v : G.adj(actor))
            StdOut.println(v);
      }
   }
}
```


Graph Client Warmup: Movie Finder

```
% java MovieFinder top-grossing.txt
Bacon, Kevin
Animal House (1978)
Apollo 13 (1995)
Few Good Men, A (1992)

Roberts, Julia
Hook (1991)
Notting Hill (1999)
Pelican Brief, The (1993)
Pretty Woman (1990)
Runaway Bride (1999)

Tilghman, Shirley
```

```
% java MovieFinder mpaa.txt
Bacon, Kevin
Air Up There, The (1994)
Animal House (1978)
Apollo 13 (1995)
Few Good Men, A (1992)
Flatliners (1990)
Footloose (1984)
Hero at Large (1980)
Hollow Man (2000)
JFK (1991)
My Dog Skip (2000)
Novocaine (2001)
Only When I Laugh (1981)
Picture Perfect (1997)
Planes, Trains & Automobiles (1987)
Sleepers (1996)
Tremors (1990)
White Water Summer (1987)
Wild Things (1998)
...
```

Kevin Bacon Numbers



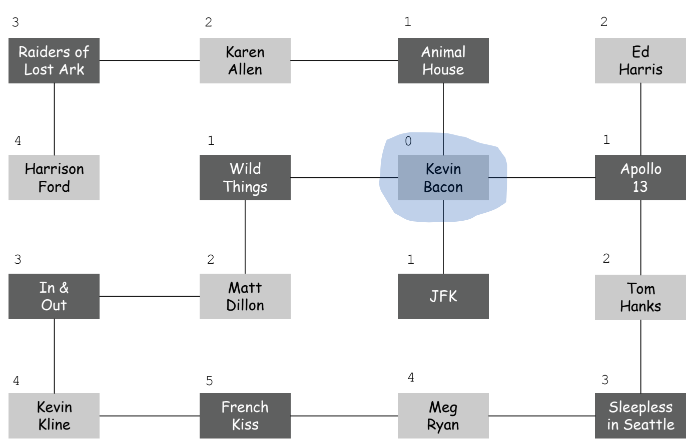
Kevin Bacon Game

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

Actor	Was in	With
Kevin Kline	French Kiss	Meg Ryan
Meg Ryan	Sleepless in Seattle	Tom Hanks
Tom Hanks	Apollo 13	Kevin Bacon
Kevin Bacon		

Bacon Numbers

How to compute. Find shortest path in graph (and divide length by 2).



Kevin Bacon Problem: Java Implementation

```
public class Bacon {
   public static void main(String[] args) {
      Graph<String> G = new Graph<String>();
      In in = new In(args[0]);
      while (!in.isEmpty()) {
         String line = in.readLine();
         String[] names = line.split("/");
         String movie = names[0];
         for (int i = 1; i < names.length; i++)
            G.addEdge(movie, names[i]);
      }
      String s = "Bacon, Kevin";
      BFSearcher<String> bfs = new BFSearcher<String>(G, s);

      while (!StdIn.isEmpty()) {
         String actor = StdIn.readLine();
         bfs.showPath(actor);
      }
   }
}
```

build graph (identical to warmup)

process queries

Kevin Bacon: Sample Output

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

Stallone, Sylvester
Rocky III (1982)
Tamburro, Charles A.
Terminator 2: Judgment Day (1991)
Berkeley, Xander
Apollo 13 (1995)
Bacon, Kevin

Tilghman, Shirley
```


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:

- remove 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 Searcher API

`public class BFSearcher<Vertex>`		(breadth first searcher data type)
	`BFSearcher(Graph<Vertex>, String s)`	run BFS on graph G from source s
`int`	`distance(Vertex v)`	return distance from s to v
`void`	`showPath(Vertex v)`	print shortest path from s to v

Decouple BFS algorithm from graph data type.

- Avoid feature creep.
- Enable client to run BFS from more than one source vertex.

Breadth First Searcher: Preprocessing

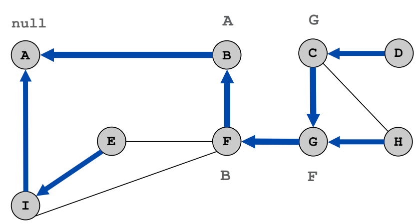
```
public class BFSearcher<Vertex> {
   private ST<Vertex, Vertex>  prev = new ST<Vertex, Vertex>();
   private ST<Vertex, Integer> dist = new ST<Vertex, Integer>();

   public BFSearcher(Graph<Vertex> G, Vertex s) {
      Queue<Vertex> q = new Queue<Vertex>();
      q.enqueue(s);
      dist.put(s, 0);
      while (!q.isEmpty()) {
         Vertex v = q.dequeue();
         for (Vertex w : G.adj(v)) {
            if (!dist.contains(w)) {
               q.enqueue(w);
               dist.put(w, 1 + dist.get(v));
               prev.put(w, v);
            }
         }
      }
   }
```

Breadth First Searcher: Printing the Path

To print shortest path from `v` to `s`:

- Follow `prev[]` from `v` back to `s`.
- Print `v, prev[v], prev[prev[v]]`, …, `s`.
- Ex: shortest path from `C` to `A`: `C - G - F - B - A` ← source

```
public void showPath(Vertex v) {
   while (prev.contains(v)) {
      StdOut.println(v);
      v = prev.get(v);
   }
}
```



Symbol Table

key	prev
A	-
B	A
C	G
D	C
E	I
F	B
G	F
H	G
I	A

Running Time Analysis

Analysis. BFS runs in linear time and scales to solve huge problems.

Data File	Movies	Actors	Edges	Read input	Build graph	BFS	Show
`G.txt`	1,261	20,734	28K	0.26 sec	0.44 sec	0.19 sec	0 sec
`PG13.txt`	2,455	67,901	100K	0.28 sec	0.94 sec	0.80 sec	0 sec
`action.txt`	14,388	133,423	250K	0.59 sec	2.1 sec	1.7 sec	0 sec
`mpaa.txt`	21,372	272,948	610K	1.8 sec	7.6 sec	5.6 sec	0 sec
`all.txt`	276,266	893,903	3.2M	14 sec	54 sec	41 sec	0 sec

58MB (all.txt)

data as of November 2, 2006

Data Analysis

Exercise. Compute histogram of Kevin Bacon numbers.
Input. 276,266 movies, 893,903 actors.

Bacon #	Frequency
0	1
1	2,221
2	212,126
3	536,111
4	104,115
5	7,475
6	739
7	88
8	10
∞	31,017

November 2, 2006

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

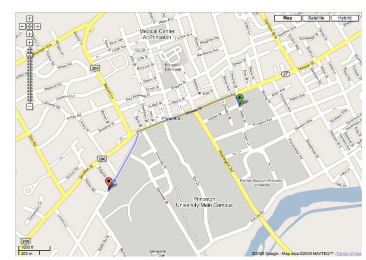
Fred Ott, solo actor in *Fred Ott Holding a Bird* (1894) → ∞

Applications of Breadth First Search

More BFS applications.

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

Extensions. Google maps.



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 +

Conclusions

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

Modules.

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

Importance of modularity.

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