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.

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.

Application demands a new data type.

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

<table>
<thead>
<tr>
<th>graph</th>
<th>vertices</th>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>telephones, computers</td>
<td>fiber optic cables</td>
</tr>
<tr>
<td>circuits</td>
<td>gates, registers, processors</td>
<td>wires</td>
</tr>
<tr>
<td>mechanical</td>
<td>joints</td>
<td>rods, beams, springs</td>
</tr>
<tr>
<td>hydraulic</td>
<td>reservoirs, pumping stations</td>
<td>pipelines</td>
</tr>
<tr>
<td>financial</td>
<td>stocks, currency</td>
<td>transactions</td>
</tr>
<tr>
<td>transportation</td>
<td>street intersections, airports</td>
<td>highways, airway routes</td>
</tr>
<tr>
<td>scheduling</td>
<td>tasks</td>
<td>precedence constraints</td>
</tr>
<tr>
<td>software systems</td>
<td>functions</td>
<td>function calls</td>
</tr>
<tr>
<td>internet</td>
<td>web pages</td>
<td>hyperlinks</td>
</tr>
<tr>
<td>games</td>
<td>board positions</td>
<td>legal moves</td>
</tr>
<tr>
<td>social relationship</td>
<td>people, actors</td>
<td>friendships, movie casts</td>
</tr>
<tr>
<td>neural networks</td>
<td>neurons</td>
<td>synapses</td>
</tr>
<tr>
<td>protein networks</td>
<td>proteins</td>
<td>protein-protein interactions</td>
</tr>
<tr>
<td>chemical compounds</td>
<td>molecules</td>
<td>bonds</td>
</tr>
</tbody>
</table>
Reference: Cosmopolitan, Nov. 2000
High School Dating

Bearman, Moody, and Stovel, 2004
Image by Mark Newman
Corporate Email Communications

Adamic and Adar, 2005
Power Transmission Grid of Western US

Reference: Duncan Watts
Protein Interaction Network

Reference: Jeong et al, Nature Review | Genetics
ARPANET LOGICAL MAP, MARCH 1977

[ ARPANET diagram with various nodes and connections ]

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
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()
Graph(In in)
void addEdge(String v, String w)
Iterable<String> adjacentTo(String v)

create an empty graph
read graph from input stream
add edge v-w
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
G/C/F/H
H/C/G
I/A/E/F
Graph Representation

Graph representation: use a symbol table.
- Key = name of vertex.
- Value = set of neighbors.

<table>
<thead>
<tr>
<th>String</th>
<th>SET&lt;String&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>value</td>
</tr>
<tr>
<td>A</td>
<td>B I</td>
</tr>
<tr>
<td>B</td>
<td>A F</td>
</tr>
<tr>
<td>C</td>
<td>D G H</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>I F</td>
</tr>
<tr>
<td>F</td>
<td>E B G I</td>
</tr>
<tr>
<td>G</td>
<td>C F H</td>
</tr>
<tr>
<td>H</td>
<td>C G</td>
</tr>
<tr>
<td>I</td>
<td>A E F</td>
</tr>
</tbody>
</table>

symbol table
### Set Data Type

**Set data type.** Unordered collection of distinct keys.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET()</td>
<td>create a set</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>is the set empty?</td>
</tr>
<tr>
<td>void add(Key key)</td>
<td>add key to the set</td>
</tr>
<tr>
<td>boolean contains(Key key)</td>
<td>is key in the set?</td>
</tr>
</tbody>
</table>

*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

```java
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.

```java
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
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.

```java
public class MovieFinder {
    public static void main(String[] args) {
        In in = new In(args[0]);
        Graph G = new Graph(in);

        while (!StdIn.isEmpty()) {
            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)
Tremors (1990)

Roberts, Julia
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)
Kevin Bacon Numbers

Tim Curry was in "The Rocky Horror Picture Show" with Susan Sarandon

Susan Sarandon was in "Bull Durham" with Kevin Costner

Tim Curry was in "Legend" with Tom Cruise

Kevin Costner was in "JFK" with Kevin Bacon

Tom Cruise was in "A Few Good Men" with Kevin Bacon
**Kevin Bacon Game**

*Game.* Find (shortest) chain of movies connecting a performer to Kevin Bacon.

<table>
<thead>
<tr>
<th>performer</th>
<th>was in</th>
<th>with</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kevin Kline</td>
<td>French Kiss</td>
<td>Meg Ryan</td>
</tr>
<tr>
<td>Meg Ryan</td>
<td>Sleepless in Seattle</td>
<td>Tom Hanks</td>
</tr>
<tr>
<td>Tom Hanks</td>
<td>Apollo 13</td>
<td>Kevin Bacon</td>
</tr>
<tr>
<td>Kevin Bacon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Kevin Kline was in "French Kiss" with Meg Ryan
- Meg Ryan was in "Sleepless in Seattle" with Tom Hanks
- Tom Hanks was in "Apollo 13" with Kevin Bacon
- Kevin Bacon was in "Kevin Bacon"
Computing Bacon Numbers

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

Path finder API.

```java
public class PathFinder
{
    PathFinder(Graph G, String s)
    int distanceTo(String v)
    Iterable<String> pathTo(String v)
}
```

Design principles.
- Decouple graph algorithm from graph data type.
- Avoid feature creep.
Computing Bacon Numbers: Java Implementation

```java
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);
        }
    }
}
```

% java Bacon top-grossing.txt
Stallone, Sylvester
Rocky III (1982)
Tamburro, Charles A.
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.
- ...
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 Searcher: Preprocessing

```java
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);
                }
            }
        }
    }
}
```
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

```
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;
}
```

<table>
<thead>
<tr>
<th>key</th>
<th>prev</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>I</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>G</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>A</td>
<td>1</td>
</tr>
</tbody>
</table>

symbol tables
Running Time Analysis

**Analysis.** BFS scales to solve huge problems.

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

60MB

data as of April 9, 2007
**Data Analysis**

**Exercise.** Compute histogram of Kevin Bacon numbers.

**Input.** 285,462 movies, 933,864 actors.

<table>
<thead>
<tr>
<th>Bacon #</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2,249</td>
</tr>
<tr>
<td>2</td>
<td>218,088</td>
</tr>
<tr>
<td>3</td>
<td>561,161</td>
</tr>
<tr>
<td>4</td>
<td>111,149</td>
</tr>
<tr>
<td>5</td>
<td>7,905</td>
</tr>
<tr>
<td>6</td>
<td>903</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>∞</td>
<td>32,294</td>
</tr>
</tbody>
</table>

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

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

*data as of April 9, 2007*
Applications of Breadth First Search

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

Extensions. Google maps.
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.