4.5 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.

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.

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
  - telephones, computers
  - gates, registers, processors
  - reservoirs, pumping stations
  - streets, intersections, airports
- edges
  - fiber optic cables
  - wires
  - rods, beams, springs
  - pipelines
  - highways, airway routes
  - transactional constraints
- mechanical
  - joints
  - rods, beams, springs
  - valves, pipes
- hydraulic
  - reservoirs, pumping stations
  - pipelines
- electrical
  - circuits
  -ards, registers, processors
- computer science
  - software systems
  - web pages
  - board positions
  - functions
  - web pages
- social network
  - people, actors
  - friendships, movie casts
- biological
  - neurons
  - protein-protein interactions
  - proteins
  - molecular interactions
- chemical
  - molecules
  - bonds
**ARPANET**

[ARPANET Logical Map, March 1977]

---

**Internet Movie Database**

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

```
% more movies.txt
Til Wee (1987)/DeBoy, David/Blumenfeld, Alan/.../Geppi, Cindy/Hershey, Barbara
Tires on le pianiste (1960)/Heymann, Claude/.../Berger, Nicole (I)
Titanic (1997)/Paxton, Bill/DiCaprio, Leonardo/.../Winslet, Kate
To All a Good Night (1983)/George, Michael (II)/.../Gentile, Linda
To Be or Not to Be (1942)/Verebes, Erno (I)/.../Lombard, Carole (I)
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 (1999)/Sacha, Orlando/Anthony, Gerald/.../Rose, Jamie
To End All Wars (2001)/Kimura, Saka/Ellis, Greg (II)/.../Sutherland, Kiefer
To Kill a Clown (1972)/Rida, Alan/Cleaver, Eric/Lamberts, Brent/Gannas, Blythe
To Live and Die in L.A. (1985)/McGuarty, Pat/Williams, Donnie/.../Safoe, Niles
...```

*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 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

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?

Q. How to implement?
A. Identical to symbol table, but ignore values.

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

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.

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

Performers 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);
        }
    }
}
```

```
% 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
Oracle of Kevin Bacon

Kevin Bacon Game

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

<table>
<thead>
<tr>
<th>Actor</th>
<th>Was in</th>
<th>With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whoopi Goldberg</td>
<td>Ghost</td>
<td>Patrick Swayze</td>
</tr>
<tr>
<td>Patrick Swayze</td>
<td>Dirty Dancing</td>
<td>Jennifer Gray</td>
</tr>
<tr>
<td>Jennifer Gray</td>
<td>Ferris Bueller’s Day Off</td>
<td>Matthew Broderick</td>
</tr>
<tr>
<td>Matthew Broderick</td>
<td>The Road to Wellville</td>
<td>John Cusack</td>
</tr>
<tr>
<td>John Cusack</td>
<td>Bullets Over Broadway</td>
<td>Dianne West</td>
</tr>
<tr>
<td>Dianne West</td>
<td>Footloose</td>
<td>Kevin Bacon</td>
</tr>
<tr>
<td>Kevin Bacon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computing Bacon Numbers

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

PathFinder API

PathFinder 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: don’t encrust Graph with search features; instead make a new datatype.
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);
        }
    }
}
```

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:
- Distance = 0
- Distance = 1
- Distance = 2
- Distance = 3

Goal. Given a vertex \( s \), find shortest path to every other vertex \( v \).

Breadth First Search

- 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

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: Preprocessing

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

Breadth First Searcher: Finding the Path

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.

<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>pathTo</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>MP3.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>14,928</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.

### Bacon # Frequency

<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)

Erdős Numbers

Applications of Breadth First Search

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

Extensions. Google maps.
Erdős Numbers

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

What's your Erdős number?
• Co-authors of a paper with Erdős: 1.
• Co-authors of those co-authors: 2.
• And so on ...

<table>
<thead>
<tr>
<th>Erdős #</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>502</td>
</tr>
<tr>
<td>2</td>
<td>5,713</td>
</tr>
<tr>
<td>3</td>
<td>26,422</td>
</tr>
<tr>
<td>4</td>
<td>62,136</td>
</tr>
<tr>
<td>5</td>
<td>66,157</td>
</tr>
<tr>
<td>6</td>
<td>32,280</td>
</tr>
<tr>
<td>7</td>
<td>10,431</td>
</tr>
<tr>
<td>8</td>
<td>3,214</td>
</tr>
<tr>
<td>9</td>
<td>953</td>
</tr>
<tr>
<td>10</td>
<td>262</td>
</tr>
<tr>
<td>11</td>
<td>94</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

= 4 billion +

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.