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.

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

<table>
<thead>
<tr>
<th>graph applications</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>

One Week of Enron Emails

"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

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)/Neymann, Claude/.../Hermitte, Nicole (I)  
Titanic (1997)/Paxton, Bill/Dicaprio, Leonardo/.../Nichols, Kate  
Titus (1999)/Neiszaphof, Hermann/Byys, Matthew/.../McEwan, Geraldine  
To All a Good Night (1980)/George, Michael (II)/.../Gentile, Linda  
To Be or Not To Be (1942)/Verhees, Erné (I)/.../Lombard, Carole (I)  
To Be or Not To Be (1983)/Brooks, Mel (I)/.../Bancroft, Anne  
To Catch a Thief (1955)/Paris, Manuel/Grant, Cary/.../Kelly, Grace  
To Die For (1989)/Hanold, Steve (I)/Jones, Duane (I)/.../Maddalena, Julie  
To Die For (1995)/Smith, Kortwood/Kidman, Nicole/.../Tucci, Maria  
To Die Standing (1999)/Sacha, Orlando/Anthony, Gerald/.../Rose, Jamie  
To End All Wars (2001)/Almura, Sake/Bills, Greg (II)/.../Sutherland, Kiefer  
To Kill a Clown (1972)/Alda, Alan/Cleaver, Eric/Lambers, Heath/Danner, Blythe  
To Live and Die in L.A. (1985)/McGarvey, Pat/Williams, Bonnie/.../Safoe, Milhem  
...%```

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.

```java
public class Graph (graph with String vertices) {
    public Graph()
    public Graph(In in)
    void addEdge(String v, String w)
    Iterable<String> adjacentTo(String v)
}
```

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

```java
String key
SET<String> value
```

Set Data Type

Set data type. Unordered collection of distinct keys.

```java
public class Set<Key extends Comparable<Key>> {
    public Set()
    boolean isEmpty()
    void add(Key key)
    boolean contains(Key key)
}
```

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.containsKey(v)) addVertex(v);
        if (!st.containsKey(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
    }

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

Second constructor. To read graph from input stream.

private void addEdge(String v, String w) {
    st.put(v, st.get(v).add(w));
    st.put(w, st.get(w).add(v));
}

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);
    }
}

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

Graph Client: Movie Finder

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
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)
Wild Things (1998)

Kevin Bacon Numbers
Oracle of Kevin Bacon

Kevin Bacon Game

Game. Given an actor or actress, find 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 Grey</td>
</tr>
<tr>
<td>Jennifer Grey</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

Design principles.
• Decouple graph algorithm from graph data type.
• Avoid feature creep: don’t encrust Graph with search features; instead make a new datatype.
public class Bacon {
    public static void main(String[] args) {
        In in = new In[args[0]];              // read in the graph from a file
        Graph G = new Graph(in);             // create object to return shortest paths
        String s = "Bacon, Kevin";            // process queries
        PathFinder finder = new PathFinder(G, s);
        while (!StdIn.isEmpty()) {            // read in the graph from a file
            String performer = StdIn.readLine(); // process queries
            for (String v : finder.pathTo(s)) {  // read in the graph from a file
                StdOut.println(v);               // process queries
            }
        }
    }
}

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.
- ...

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);
                }
            }
        }
    }
}
```

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

```java
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>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>300K</td>
<td>0.31 sec</td>
<td>0.72 sec</td>
<td>0.22 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>

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>...</td>
</tr>
<tr>
<td>32,294</td>
<td></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.

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 …

Paul Erdős (1913-1996)
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.