4.7 Small World Phenomenon

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.

Small World Phenomenon

- Six handshakes away from anyone else in the world.
- Long a matter of folklore.
- “It’s a small world after all.”

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.

Graph Data Type

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

Applications of Graphs

<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>

Actor-Movie Graph (Partial)

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

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

Wild Things (1998) /Bacon, Kevin/Campbell, Kevin/Dillon, Matt/Murray, Bill/Richards, Denise
JFK (1991) /Anson, Edward/Bacon, Kevin/Costner, Kevin/Jones, Tommy Lee/Grubbs, Gary

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

Internet Movie Database

Graph API.
- addVertex(v) add a vertex v
- addEdge(v, w) add connection v-w
- neighbors(v) return neighbors of v as array

Graph representation: use a symbol table.
- Key = name of vertex (e.g., movie or actor).
- Value = adjacency list of neighbors.

Symbol Table

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<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>

String Adjlist
**Adjacency List Implementation**

Adjacency list implementation. No surprises.

```java
public class AdjList {
    private Node first;

    private static class Node {
        private String name;
        private Node next;
        public Node(String name, Node next) {
            this.name = name;
            this.next = next;
        }
    }

    public void insert(String s) {
        first = new Node(s, first);
    }

    public String[] toArray() { ... }
}
```

**Graph Implementation**

```java
public class Graph {
    private SymbolTable st = new SymbolTable();

    public void addEdge(String v, String w) {
        if (st.get(v) == null) addVertex(v);
        if (st.get(w) == null) addVertex(w);
        AdjList vlist = (AdjList) st.get(v);
        AdjList wlist = (AdjList) st.get(w);
        vlist.insert(w);  // add w to v's list
        wlist.insert(v);  // add v to w's list
    }

    public void addVertex(String v) {
        st.put(v, new AdjList());  // add new vertex v with no neighbors
    }

    public String[] neighbors(String v) {
        AdjList adjlist = (AdjList) st.get(v);
        return adjlist.toArray();
    }
}
```

---

**Graph Client Warmup: Movie Finder**

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

```java
public class MovieFinder {
    public static void main(String[] args) {
        Graph G = new Graph(); build graph
        In data = new In(args[0]);  // input
        while (!data.isEmpty()) {
            String line = data.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
        }

        while (!StdIn.isEmpty()) {
            String actor = StdIn.readLine();
            String[] neighbors = G.neighbors(actor);
            for (int i = 0; i < neighbors.length; i++)
                System.out.println(neighbors[i]);
        }
    }
}
```

---

**Graph Client Warmup: Movie Finder**

```java
// moviesFinder 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

// moviesFinder mpaa.txt
Bacon, Kevin
Air Up There, The (1994)
Animal House (1978)
Apollo 13 (1995)
Few Good Men, A (1992)
Flaminos (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 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>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 Bacon Problem: Java Implementation

```java
public class Bacon {
    public static void main(String[] args) {
        // build graph (identical to warmup)
        process queries
    }
}
```

Bacon Numbers

Bacon number. Length of shortest such chain to Kevin Bacon.

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

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.
Berkeley, Xander
Apollo 13 (1995)
Bacon, Kevin
Tilghman, Shirley
```
Breadth First Searcher ADT

**Goal:** given one 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 visited in increasing order of distance from \( s \) because we use FIFO queue.

---

Breadth First Searcher: Preprocessing

**Goal:** given one vertex \( s \) find shortest path to every other vertex \( v \).

```java
public class BFSReader {
    private SymbolTable visited = new SymbolTable();
    
    public BFSReader(Graph G, String s) {
        Queue q = new Queue();
        q.enqueue(s);
        visited.put(s, "");
        while (!q.isEmpty()) {
            String v = (String) q.dequeue();
            String[] neighbors = G.neighbors(v);
            for (int i = 0; i < neighbors.length; i++) {
                String w = neighbors[i];
                if (visited.get(w) == null) {
                    q.enqueue(w);
                    visited.put(w, v);
                }
            }
        }
    }
}
```

---

Breadth First Searcher ADT Design

**Isolate BFS algorithm from graph data type.**
- Avoid feature creep.
- Keep modules independent.
- Enable client to run BFS from more than one source vertex.

```java
public class BFSReader {
    private SymbolTable visited;
    
    public BFSReader(Graph G, String s) { ... }

    public void showPath(String v) { ... }
    public int distance(String v) { ... }
    public String[] path(String v) { ... }
}
```
Applications of Breadth First Search

More BFS applications.
- Word ladder: green → great → great → groan → grown → brown
- Shortest number of hops for Internet packet.
- Particle tracking.
- Image processing.
- Crawling the Web.
- ...

Extensions. Google maps.

Running Time Analysis

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

<table>
<thead>
<tr>
<th>Data File</th>
<th>Movies</th>
<th>Actors</th>
<th>Edges</th>
<th>Read input</th>
<th>Build graph</th>
<th>BFS</th>
<th>Show</th>
</tr>
</thead>
<tbody>
<tr>
<td>top.txt</td>
<td>187</td>
<td>8,265</td>
<td>10K</td>
<td>0.10 sec</td>
<td>0.10 sec</td>
<td>0.10 sec</td>
<td>0 sec</td>
</tr>
<tr>
<td>mpaa-y.txt</td>
<td>967</td>
<td>13,850</td>
<td>18K</td>
<td>0.16 sec</td>
<td>0.24 sec</td>
<td>0.13 sec</td>
<td>0 sec</td>
</tr>
<tr>
<td>ykl.txt</td>
<td>4,754</td>
<td>43,940</td>
<td>57K</td>
<td>0.29 sec</td>
<td>0.56 sec</td>
<td>0.30 sec</td>
<td>0 sec</td>
</tr>
<tr>
<td>mpaa.txt</td>
<td>14,192</td>
<td>170,539</td>
<td>383K</td>
<td>0.87 sec</td>
<td>3.4 sec</td>
<td>1.4 sec</td>
<td>0 sec</td>
</tr>
<tr>
<td>all.txt</td>
<td>122,812</td>
<td>418,468</td>
<td>1.5M</td>
<td>2.8 sec</td>
<td>14.9 sec</td>
<td>9.4 sec</td>
<td>0 sec</td>
</tr>
</tbody>
</table>

26MB

**Perspective.** Google indexes 8 billion web pages (50TB), and executes 250 million searches per day!

Data Analysis

Exercise. Compute histogram of Kevin Bacon numbers.
Input. 122,812 movies, 418,468 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>1,494</td>
</tr>
<tr>
<td>2</td>
<td>127,778</td>
</tr>
<tr>
<td>3</td>
<td>239,608</td>
</tr>
<tr>
<td>4</td>
<td>36,455</td>
</tr>
<tr>
<td>5</td>
<td>2,963</td>
</tr>
<tr>
<td>6</td>
<td>275</td>
</tr>
<tr>
<td>7</td>
<td>39</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
</tr>
<tr>
<td>9</td>
<td>99</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>9,692</td>
</tr>
</tbody>
</table>

Conclusions

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

**Layers of abstraction.**
- Adjacency list: linked list.
- Queue: linked list.
- Symbol table: array of linked lists.
- Graph: symbol table of adjacency lists.
- Breadth first searcher: graph + queue + symbol table.

**Importance of ADTs.**
- Enables us to build and debug large programs.
- Enables us to solve large problems efficiently.