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

<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

---

### Kissing Network

[Kissing Network Image](image)

**Reference:** Cosmopolitan, Nov. 2000

---

### High School Dating

[High School Dating Image](image)

**Reference:** Bearman, Moody, and Stovel, 2004

**Image by Mark Newman**

---

### Corporate Email Communications

[Corporate Email Communications Image](image)

**Reference:** Adamic and Adar, 2005
Power Transmission Grid of Western US

Reference: Duncan Watts

Protein Interaction Network

Reference: Jeong et al, Nature Review | Genetics

ARPANET

The Internet

The Internet as mapped by The Opte Project
http://www.opte.org
**Input format.** Movie followed by list of performers, separated by slashes.

<table>
<thead>
<tr>
<th>Movie</th>
<th>Performers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tin Man (1987)</td>
<td>DeBoy, David/Blumenfeld, Alan/Geppi, Cindy/Hershey, Barbara</td>
</tr>
<tr>
<td>Titanic (1997)</td>
<td>Paxton, Bill/DiCaprio, Leonardo/Winslet, Kate</td>
</tr>
<tr>
<td>To Die For (1989)</td>
<td>Bond, Steve/Jones, Duane/Maddelena, Julie</td>
</tr>
<tr>
<td>To Die Standing (1990)</td>
<td>Sachs, Orlando/Anthony, Geraldine/Rose, Jamie</td>
</tr>
<tr>
<td>To Kill a Clown (1972)</td>
<td>Alda, Alan/Clavering, Eric/Lambers, Heath/Danner, Blythe</td>
</tr>
<tr>
<td>To Be or Not to Be (1942)</td>
<td>Verebes, Ernő/Lombard, Carole</td>
</tr>
<tr>
<td>To Be or Not to Be (1983)</td>
<td>Brooks, Mel/Bancroft, Anne</td>
</tr>
<tr>
<td>To Catch a Thief (1955)</td>
<td>Paris, Manuel/Grant, Cary/Kelly/Grace</td>
</tr>
<tr>
<td>To End All Wars (2001)</td>
<td>Kimura, Sakae/Ellis, Greg/Sutherland, Kiefer</td>
</tr>
<tr>
<td>To Kill a Clown (1972)</td>
<td>Alda, Alan/Clavering, Eric/Lambers, Heath/Danner, Blythe</td>
</tr>
</tbody>
</table>

**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 {
    private Map<String, Set<String>> adjacencies;

    public Graph() {
        adjacencies = new HashMap<>();
    }

    public void addEdge(String v, String w) {
        adjacencies.computeIfAbsent(v, x -> new HashSet<>()).add(w);
    }

    // other methods...
}
```

**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>

**Graph representation:** use a symbol table.

- **Key:** name of vertex.
- **Value:** set of neighbors.
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?

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 (continued)

Second constructor. To read graph from input stream.

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]);
    }
}

Graph Client: Movie Finder

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

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);
        }
    }
}
Kevin Bacon Numbers

Kevin Bacon Game

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

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

Path Finder API

Path finder API.

Design principles.
- Decouple graph algorithm from graph data type.
- Avoid feature creep.

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.
- ...
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 $\text{prev}[]$ from vertex $v$ back to source $s$.

- Consider vertices: $v$, $\text{prev}[v]$, $\text{prev}[	ext{prev}[v]]$, ..., $s$.
- 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.containsKey(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>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>npaa.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>∞</td>
<td>32,294</td>
</tr>
</tbody>
</table>

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.