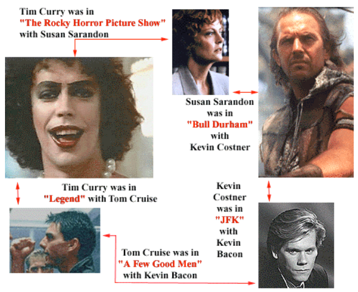
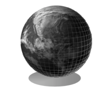
4.7 Small World Phenomenon



Introduction to Computer Science · Robert Sedgewick and Kevin Wayne · Copyright © 2005 · http://www.cs.Princeton.EDU/IntroCS

Small World Phenomenon



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. e.g., occupation and age
- 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

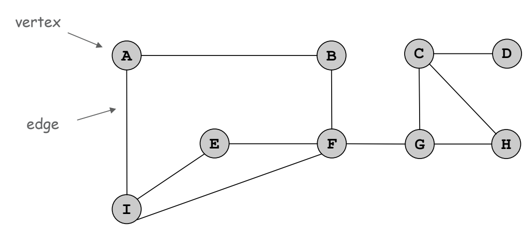
Sociology applications.
- Looking for a job.
- Marketing products or ideas.
- Formation and spread of fame and fads.
- Train of thought followed in a conversation.
- Defining representative-ness of political bodies.
- Kevin Bacon game (movies, rock groups, facebook, etc.).

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.

Reference. Duncan J. Watts, *Small Worlds: The Dynamics of Networks between Order and Randomness*, Princeton University Press, 1999.

Graph Data Type

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



Applications of Graphs

Graph	Vertices	Edges
communication	telephones, computers	fiber optic cables
circuits	gates, registers, processors	wires
mechanical	joints	rods, beams, springs
hydraulic	reservoirs, pumping stations	pipelines
financial	stocks, currency	transactions
transportation	street intersections, airports	highways, airway routes
scheduling	tasks	precedence constraints
software systems	functions	function calls
internet	web pages	hyperlinks
games	board positions	legal moves
social relationship	people, actors	friendships, movie casts
neural networks	neurons	synapses
protein networks	proteins	protein-protein interactions
chemical compounds	molecules	bonds

Internet Movie Database

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, Neve/Dillon, Matt/Murray, Bill/Richards, Denise
JFK (1991)/Asner, Edward/Bacon, Kevin/Costner, Kevin/Jones, Tommy Lee/Grubbs, Gary
Braveheart (1995)/Gibson, Mel//Marceau, Sophie/McGoohan, Patrick/Hanly, Peter
...
```

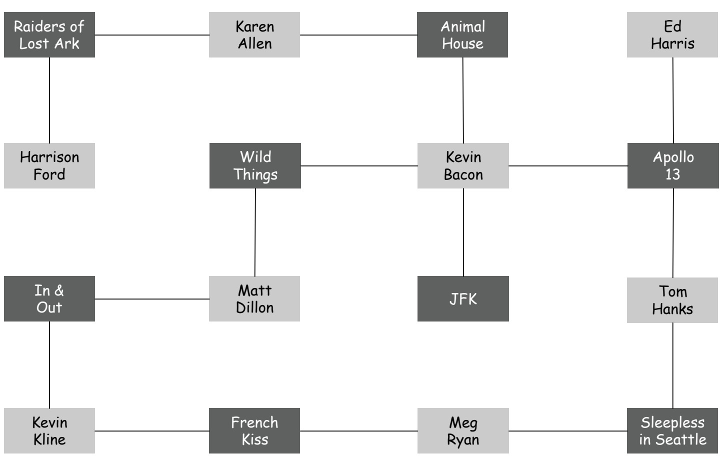
Reference: http://www.imdb.com/interfaces

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.

Actor-Movie Graph (Partial)



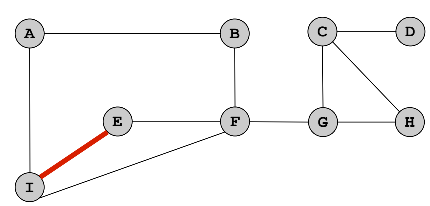
Graph Representation

Graph representation: use a symbol table.

- Key = name of vertex (e.g., movie or actor).
- Value = adjacency list of neighbors.

Graph API.

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



Symbol Table

Key	Value
A	B I
B	A F
C	D G H
D	C
E	I F
F	E B G I
G	C F H
H	C G
I	A E F
String	AdjList

Adjacency List Implementation

Adjacency list implementation. No surprises.

```
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

```
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.

```
public class MovieFinder {
   public static void main(String[] args) {
                                                        build graph
      Graph G = new Graph();
      In data = new In(args[0]);  ← file 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
      }
                                                  print all of actor's movies
      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 MovieFinder 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
```

```
% java MovieFinder mpaa.txt
Bacon, Kevin
Air Up There, The (1994)
Animal House (1978)
Apollo 13 (1995)
Few Good Men, A (1992)
Flatliners (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)
...
```

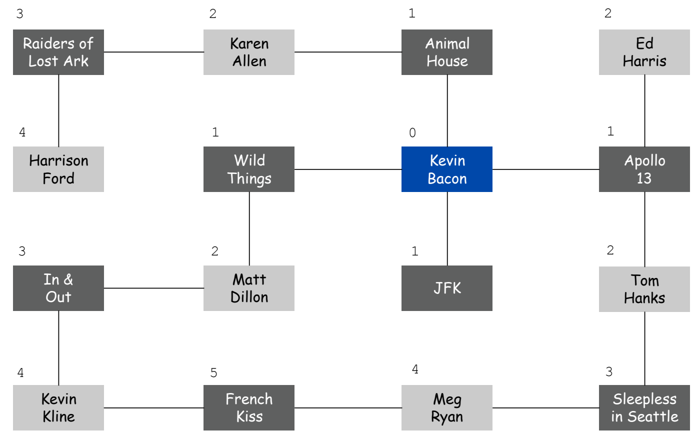
Kevin Bacon Game

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

Actor	Was in	With
Kevin Kline	French Kiss	Meg Ryan
Meg Ryan	Sleepless in Seattle	Tom Hanks
Tom Hanks	Apollo 13	Kevin Bacon
Kevin Bacon		

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 Problem: Java Implementation

```
public class Bacon {
   public static void main(String[] args) {

      Graph G = new Graph();                       build graph (identical to warmup)
      In data = new In(args[0]);
      while (!data.isEmpty()) {
         String line = data.readLine();
         String[] names = line.split("/");
         String movie = names[0];
         for (int i = 1; i < names.length; i++)
            G.addEdge(movie, names[i]);
      }

      BFSearcher bfs = new BFSearcher(G, "Bacon, Kevin");

      while (!StdIn.isEmpty()) {                   process queries
         String actor = StdIn.readLine();
         bfs.showPath(actor);
      }
   }
}
```


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.
Terminator 2: Judgment Day (1991)
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`.

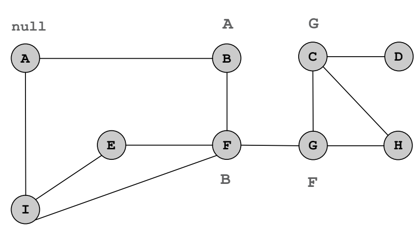
```
public class BFSearcher {
   private SymbolTable visited = new SymbolTable();

   public BFSearcher(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: Printing the Path

Print the shortest path from `v` to `s`.

- Follow visited path from `v` back to `s`.
- Print `v`, `visited[v]`, `visited[visited[v]]`, …, s. source
- Ex: shortest path from `C` to `A`: `C - G - F - B - A`

```
public void showPath(String v) {
   while (visited.get(v) != null) {
      System.out.println(v);
      v = (String) visited.get(v);
   }
}
```



Symbol Table

key	visited
A	-
B	A
C	G
D	C
E	I
F	B
G	F
H	G
I	A

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.

```
public class BFSearcher {
   private SymbolTable visited;

   public BFSearcher(Graph G, String s) { … }

   public void showPath(String v) { … }
   public int  distance(String v) { … }
   public String[] path(String v) { … }

}
```

Running Time Analysis

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

Data File	Movies	Actors	Edges	Read input	Build graph	BFS	Show
top.txt	187	8,265	10K	0.10 sec	0.10 sec	0.10 sec	0 sec
mpaa-g.txt	967	13,850	18K	0.16 sec	0.24 sec	0.13 sec	0 sec
y2k.txt	4,754	43,940	57K	0.29 sec	0.56 sec	0.30 sec	0 sec
mpaa.txt	14,192	170,539	383K	0.87 sec	3.4 sec	1.4 sec	0 sec
all.txt	122,812	418,468	1.5M	2.8 sec	14.9 sec	9.4 sec	0 sec

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.

Bacon #	Frequency
0	1
1	1,494
2	127,778
3	239,608
4	36,455
5	2,963
6	275
7	39
8	47
9	99
10	15
11	2
∞	9,692

Akbar Abdi, star of Iranaian film *Honarpisheh*

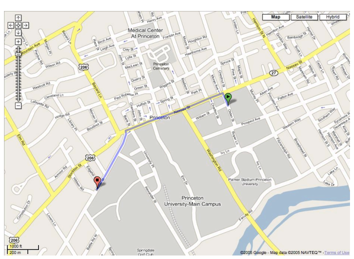
Fred Ott, solo actor in *Fred Ott Holding a Bird* (1894)

Applications of Breadth First Search

More BFS applications.

- Word ladder: `green` - `greet` - `great` - `groat` - `groan` - `grown` - `brown`
- Shortest number of hops for Internet packet.
- Particle tracking.
- Image processing.
- Crawling the Web.
- ...

Extensions. Google maps.



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.