"Hopefully, you've played around a bit with The Oracle of Bacon at Virginia and discovered how few steps are necessary to link just about anybody who has ever been in a movie to Kevin Bacon, but could there be some actor or actress who is even closer to the center of the Hollywood universe? By processing all of the almost half of a million people in the Internet Movie Database I discovered that there are currently 1160 people who are better centers than Kevin Bacon! ... By computing the average of these numbers we see that the average (Sean) Connery Number is about 2.682 making Connery a better center than Bacon"

- Who is the Center of the Hollywood Universe?, University of Virginia
That was in 2001.
In 2013 Harvey Keitel has become the center of the Hollywood Universe. Connery is 136th. Bacon has moved up to 370th.

An Early Problem in Graph Theory
- Leonhard Euler (1707 - 1783)
  – One of the first mathematicians to study graphs
- The Seven Bridges of Konigsberg Problem
  – Konigsberg now called Kaliningrad
- A puzzle for the residents of the city
- The river Pregel flows through the city
- 7 bridges cross the river
- Can you cross all bridges while crossing each bridge only once?

Clicker Question 1
- How many solutions does the Seven Bridges of Konigsberg Problem have?
  A. 0
  B. 1
  C. 2
  D. 3
  E. >= 4
How to Solve

- Brute Force?
- Euler's Solution
  - Redraw the map as a graph (really a multigraph)

Graph Definitions

- A graph is comprised of a set of vertices (nodes) and a set of edges (links, arcs) connecting the vertices
  - An edge connects 2 vertices
- in a directed graph edges are one-way
  - movement allowed from first node to second, but not second to first
  - directed graphs also called digraphs
- in an undirected graph edges are two-way
  - movement allowed in either direction

Euler's Proposal

- A connected graph has an Euler tour (cross every edge exactly one time) if and only if every vertex has an even number of edges
  - Eulerian Circuit
- What if we reduce the problem to only crossing each edge (bridge) exactly once?
  - Doesn't matter if we end up where we started
  - Eulerian Trail

Definitions

- In a weighted graph the edge has cost or weight that measures the cost of traveling along the edge
- A path is a sequence of vertices connected by edges
  - The path length is the number of edges
  - The weighted path length is the sum of the cost of the edges in a path
- A cycle is a path of length 1 or more that starts and ends at the same vertex
  - a directed acyclic graph is a directed graph with no cycles
Computer Scientists use graphs to model all kinds of things.

Arpanet 1969, 1971
Enron emails 2001

US Airport Network

"Jefferson" High School, Ohio and Sexual Networks, 2005,
Representing Graphs

- How to store a graph as a data structure?

### Adjacency Matrix Representation

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>Br</th>
<th>Bl</th>
<th>Ch</th>
<th>Co</th>
<th>E</th>
<th>FG</th>
<th>G</th>
<th>Pa</th>
<th>Pe</th>
<th>S</th>
<th>U</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Br</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bl</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ch</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Co</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FG</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Pa</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pe</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>A</td>
</tr>
<tr>
<td>Brazil</td>
<td>Br</td>
</tr>
<tr>
<td>Bolivia</td>
<td>Bl</td>
</tr>
<tr>
<td>Chile</td>
<td>Ch</td>
</tr>
<tr>
<td>Columbia</td>
<td>Co</td>
</tr>
<tr>
<td>Ecuador</td>
<td>E</td>
</tr>
<tr>
<td>French Guiana</td>
<td>FG</td>
</tr>
<tr>
<td>Guyana</td>
<td>G</td>
</tr>
<tr>
<td>Paraguay</td>
<td>Pa</td>
</tr>
<tr>
<td>Peru</td>
<td>Pe</td>
</tr>
<tr>
<td>Suriname</td>
<td>S</td>
</tr>
<tr>
<td>Uruguay</td>
<td>U</td>
</tr>
<tr>
<td>Venezuela</td>
<td>V</td>
</tr>
</tbody>
</table>

### Undirected Graph?

- Use a ragged 2d array to save space

#### The Map Coloring Problem

- How many colors do you need to color a map, so that no 2 countries that have a common border (not a point) are colored the same?
- How to solve using Brute Force?
Example

A four-coloring of a map of the states of the United States (ignoring lakes).

Source: https://en.wikipedia.org/wiki/Four_color_theorem

What About the Ocean?

<table>
<thead>
<tr>
<th>A</th>
<th>Br</th>
<th>Bl</th>
<th>Ch</th>
<th>Co</th>
<th>E</th>
<th>FG</th>
<th>G</th>
<th>Pa</th>
<th>Pe</th>
<th>S</th>
<th>U</th>
<th>V</th>
<th>Oc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Br</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Co</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Pa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Oc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

A Solution

Make the Ocean Blue
More Definitions

- A **dense** graph is one with a large number of edges
  - maximum number of edges?
- A **sparse** graph is one in which the number of edges is much less than the maximum possible number of edges
  - No standard cutoff for dense and sparse graphs

Graph Representation

- For dense graphs the adjacency matrix is a reasonable choice
  - For weighted graphs change booleans to double or int
  - Can the adjacency matrix handle directed graphs?
- Most graphs are sparse, not dense
- For sparse graphs an **adjacency list** is an alternative that uses less space
- Each vertex keeps a list of edges indicating the vertices it is connected to.

Graph Implementation

```java
public class Graph {
    private static final double INFINITY = Double.MAX_VALUE;
    private Map<String, Vertex> vertices;

    public Graph() // create empty Graph
    public void addEdge(String source, String dest, double cost)

    // find all paths from given vertex
    public void findUnweightedShortestPath (String startName)

    // called after findUnweightedShortestPath
    public void printPath(String destName)
}
```

Graph Class

- This Graph class stores vertices
- Each vertex has an adjacency list
  - what vertices does it connect to?
- shortest path method finds all paths from start vertex to every other vertex in graph
- after shortest path method called queries can be made for path length from start node to destination node
**Vertex Class (nested in Graph)**

```java
private static class Vertex
    private String name;
    private List<Edge> adjacent;

    public Vertex(String n)
    
    // for shortest path algorithms
    private double distance;
    private Vertex prev;
    private int scratch;

    // call before finding new paths
    public void reset()
```

**Unweighted Shortest Path**

- Given a vertex, S (for start) find the shortest path from S to all other vertices in the graph
- Graph is unweighted (set all edge costs to 1)

![Graph Diagram](image)

**Edge Class (nested in Graph)**

```java
private static class Edge
    private Vertex dest;
    private double cost;

    private Edge(Vertex d, double c)
```

**6 Degrees of Wikipedia**

- [https://www.sixdegreesofwikipedia.com/](https://www.sixdegreesofwikipedia.com/)
Word Ladders

- Agree upon dictionary
- Start word and end word of same length
- Change one letter at a time to form step
- Step must also be a word
- Example: Start = silly, end = funny

Graph Representation

- What are the vertices and when does an edge exist between two vertices?

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Letters</td>
<td>Words</td>
</tr>
<tr>
<td>B. Words</td>
<td>Words that share one or more letters</td>
</tr>
<tr>
<td>C. Letters</td>
<td>Words that share one or more letters</td>
</tr>
<tr>
<td>D. Words</td>
<td>Words that differ by one letter</td>
</tr>
<tr>
<td>E. Words</td>
<td>Letters</td>
</tr>
</tbody>
</table>

Size of Graph

- Number of vertices and edges depends on dictionary
- Modified Scrabble dictionary, 5 letter words
- Words are vertices
  - 8660 words
- Edge exists between word if they are one letter different
  - 24,942 edges

Is this graph sparse or dense?

A. Sparse
B. Dense

Max number of edges = 37,493,470
**Unweighted Shortest Path Algorithm**

- Problem: Find the shortest word ladder between two words if one exists
- What kind of search should we use?

A. Breadth First Search
B. Depth First Search
C. Either one

---

**Unweighted Shortest Path Algorithm**

- Set distance of start to itself to 0
- Create a queue and add the start vertex
- while the queue is not empty
  - remove front
  - loop through all edges of current vertex
    - get node edge connects to
    - if this node has not been visited
      - sets its distance to current distance + 1
      - sets its previous node to current node
      - add new node to queue

---

Start at "smart" and enqueue it [smart]
Dequeue (smart), loop through edges
[swart]

Dequeue (smart), loop through edges
[swart, start, scart]

Dequeue (smart), loop through edges
[swart, start]
Deque (smart), loop through edges
[start, swart, start, scart, smalt, smarm]

Done with smart, dequeue (swart)
[start, scart, smalt, smarm]

loop through edges of swart (start already present)
[start, scart, smalt, smarm]

loop through edges of swart (scart already present)
[start, scart, smalt, smarm]
Unweighted Shortest Path

- Implement method
- demo
- how is path printed?
- The diameter of a graph is the longest shortest past in the graph
- How to find?
- How to find center of graph?
  - vertex connected to the largest number of other vertices with the shortest average path length

Positive Weighted Shortest Path

- Edges in graph are weighted and all weights are positive
- Similar solution to unweighted shortest path
- Dijkstra's algorithm
- Edsger W. Dijkstra (1930–2002)
- UT Professor 1984 - 2000
- Algorithm developed in 1956 and published in 1959.
Dijkstra on Creating the Algorithm

- What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame.


Dijkstra's Algorithm

- Pick the start vertex
- Set the cost of the start vertex to 0 and all other vertices to INFINITY
- While there are unvisited vertices:
  - Let the current vertex be the lowest cost vertex that has not yet been visited
  - mark current vertex as visited
  - for each edge from the current vertex
    - if the sum of the cost of the current vertex and the cost of the edge is less than the cost of the destination vertex
      - update the cost of the destination vertex
      - set the previous of the destination vertex to the current vertex

Vertex Class (nested in Graph)

```java
private static class Vertex
{
    private String name;
    private List<Edge> adjacent;

    public Vertex(String n)
    {
        // for shortest path algorithms
        private double distance;
        private Vertex prev;
        private int scratch;

        // call before finding new paths
        public void reset()
    }
```

Dijkstra's Algorithm

- Example of a Greedy Algorithm
  - A Greedy Algorithm does what appears to be the best thing at each stage of solving a problem
- Gives best solution in Dijkstra's Algorithm
- Does NOT always lead to best answer
- Fair teams:
  - (10, 10, 8, 8, 8), 2 teams
- Making change with fewest coins
  - (1, 5, 10) 15 cents
  - (1, 5, 12) 15 cents
What is the cost of the lowest cost path from A to E?

A. 3
B. 17
C. 20
D. 28
E. 37

A is start vertex
Set cost of A to 0, all others to INFINITY
Place A in a priority queue

[(A,0)] pq
dequeue (A,0)
Mark A as visited

[ ] pq
current vertex A:
loop through A's edges
if sum of cost from A to edge is less than current cost
update cost and prev
[ ] pq
A -> B, 0 + 1 < INFINITY

[(B,1)] pq

[(B,1), (C, 7)] pq
A -> G, 0 + 17 < INFINITY

[(B,1), (C, 7), (G, 17)] pq

[(B,1)] pq
A -> C, 0 + 7 < INFINITY

[(B,1), (C, 7)] pq

[(B,1), (C, 7), (G, 17)] pq

current vertex B:
loop through B's edges
if sum of cost from B to edge is less than current cost
update cost and prev
[(C, 7), (G, 17)] pq
B -> C, 1 + 3 < 7
update C's cost and previous
[(C, 4), (C, 7), (G, 17)] pq

[(C, 4), (C, 7), (G, 17), (D, 22)] pq
current vertex is C, cost 4
loop through C's edges

[(C, 4), (C, 7), (G, 17)] pq
B -> D, 1 + 21 < INFINITY
[(C, 4), (C, 7), (G, 17), (D, 22)] pq

[(C, 7), (G, 17), (D, 22)] pq
C -> A, A already visited so skip
[\[(C, 7), (G, 17), (D, 22)\] pq
C \rightarrow B, B already visited so skip] 69

[\[(C, 7), (G, 17), (D, 22)\] pq
current vertex is C
Already visited so skip] 71

[\[(C, 7), (G, 17), (D, 22)\] pq
C \rightarrow F, 4 + 3 < INFINITY] 70

[\[(C, 7), (F, 7), (G, 17), (D, 22)\] pq
current vertex is F
loop through F's edges] 72
[(G, 17), (D, 22)] pq
F -> C, already visited so skip

[(D, 11), (G, 17), (D, 22)] pq
current vertex is D
loop through D's edges

[(G, 17), (D, 22)] pq
F -> D, 7 + 4 < 22
update D's cost and previous

[(D, 11), (G, 17), (D, 22)] pq
D -> B, already visited so skip
((G, 17), (E, 17), (D, 22)) pq
D -> E, 11 + 6 < INFINITY
update E's cost and previous

((G, 17), (E, 17), (D, 22)) pq
D -> G, 11 + 5 < 17
update G's cost and previous

((G, 17), (E, 17), (D, 22)) pq
D -> F, already visited so skip

((G, 17), (E, 17), (D, 22)) pq
current vertex is G
loop though edges, already visited all neighbors
Implementing Dijkstra's

- Create a Path class to allow for multiple distances to a given vertex

```java
private static class Path implements Comparable<Path> {
    private Vertex dest;
    private double cost;
}
```

- Use a priority queue of Paths to store the vertices and distances

Alternatives to Dijkstra's Algorithm

- A*, pronounced "A Star"
- A heuristic, goal of finding shortest weighted path from single start vertex to goal vertex
- Uses actual distance like Dijkstra's but also estimates remaining cost or distance
  - `distance` is set to current distance from start PLUS the estimated distance to the goal
- For example when finding a path between towns, estimate the remaining distance as the straight-line (as the crow flies) distance between current location and goal.
Spanning Tree

- **Spanning Tree**: A tree of edges that connects all the vertices in a graph.

Minimum Spanning Tree

- **Minimum Spanning Tree**: A spanning tree in a weighted graph with the lowest total cost.
  - used in network design, taxonomy, Image registration, and more!

Prim's Algorithm

- Pick a vertex arbitrarily from graph.
  - In other words, it doesn't matter which one.
- Add lowest cost edge between the tree and a vertex that is not part of the tree UNTIL every vertex is part of the tree.
- Greedy Algorithm, very similar to Dijkstra's.

Cost of spanning tree shown?
- A. 6
- B. 7
- C. 29
- D. 61
- E. None of These

Pick D as root.
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
2 from D to A (or C)

Lowest cost edge from tree to vertex not in Tree?
2 from A to B

Lowest cost edge from tree to vertex not in Tree?
2 from D to C (OR from A o B)

Lowest cost edge from tree to vertex not in Tree?
5 from D to G
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
1 from G to F

Pick D as root

Lowest cost edge from tree to vertex not in Tree?
6 from G to E

Lowest cost edge from tree to vertex not in Tree?
4 from D to F
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
3 from F to C

Lowest cost edge from tree to vertex not in Tree?
1 from B to A

Lowest cost edge from tree to vertex not in Tree?
3 from C to B

Lowest cost edge from tree to vertex not in Tree?
5 from D to G
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
6 from D to E

Cost of Spanning Tree?

Other Graph Algorithms

- Lots!
- [http://en.wikipedia.org/wiki/Category:Graph_algorithms](http://en.wikipedia.org/wiki/Category:Graph_algorithms)