# Graphs and Algorithms

- Graphs are a mathematical concept readily adapted into computer programming.
- Graphs are not just data structures, that is, they are not solutions to simple data storage problems.
- Graphs are usually used because of the algorithms associated with them.

# Introduction to Graphs

- Graphs are rather like trees.
- From a math perspective, a tree is a graph, but there are many graphs that are not trees.
- From a programming perspective, graphs are used differently from trees.
- Usually used to represent data that has a physical existence.

# Definitions

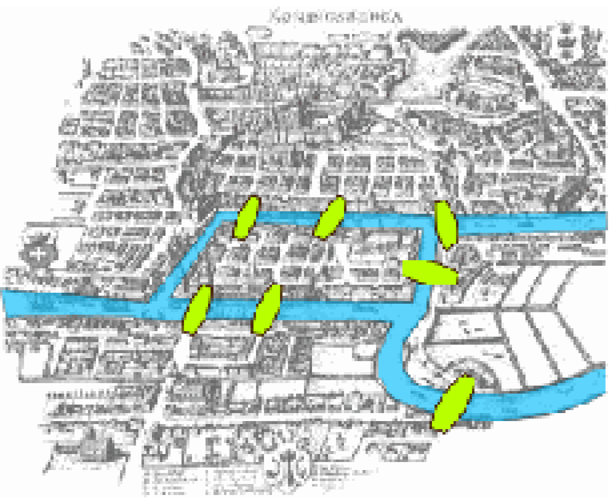
- A graph consists of vertices and edges.
- Two nodes are said to be adjacent if they are connected by a single edge.
- A path is a sequence of edges.
- A graph is said to be connected if there is at least on path from every vertex to every other.

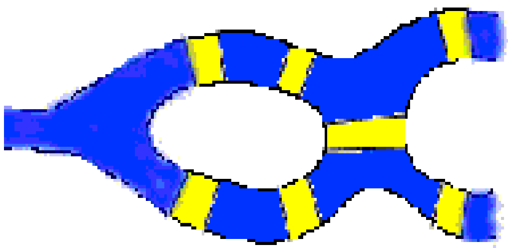
# Directed and Weighted Graphs

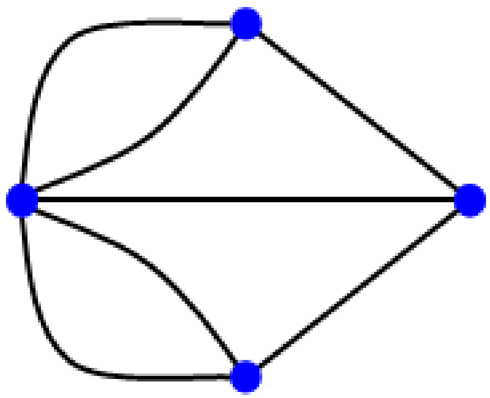
- These are graphs produced from basic graphs by introducing edge variations.
- Directed graphs add directions to the edges, that is, one can only travel one way along the edges.
- Weighted graphs add weights to the edges, often to represent physical distances between the vertices.

# The Seven Bridges of Koenigsberg

- The is the first well known application of the graphs.
- Leonhard Euler, in the early 1700s.







# Representing Graphs in a Program

- Unlike trees, whose data is only stored at the nodes (vertices), a graph can store data both at the vertices and at the edges.
- Also unlike binary trees, a graph may have any number of adjacent vertices.

# Vertices

- The class usually stores an object representing all pertinent data of the physical object.
- Also often stores a flag/counter used by searching/traversing algorithms.
- All vertices are usually placed in arrays, and referred to using their index number.

# `Vertex` class

```
class Vertex {
  public City c;
  public char label;
  public boolean visited;

  public Vertex (City new, char l) {
    c = new;
    label = l;
    visited = false;
  }
}
```

# Edges

- The solution to the freeform connections in graphs is to use either an adjacent matrix or an adjacent list.
- An adjacent matrix is a two dimensional array in which the elements indicate whether an edge is present between tow vertices.

# The Adjacent Matrix

- If a graph has n vertices, the adjacent matrix will be of size nxn.
- The existence of an edge is indicated by a 1, and 0 otherwise.

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 1 | 0 | 0 | 1 |
| C | 1 | 0 | 0 | 0 |
| D | 1 | 1 | 0 | 0 |

# The Adjacency List

- Use a linked list to indicate all the vertices adjacent to a particular vertex.
- Need one linked list per vertex.

| vertex | Adjacency List |
|---|---|
| A | BCD |
| B | AD |
| C | A |
| D | AB |

# Adding Vertices and Edges to a Graph

- For each vertex:
  - make a new vertex object and insert it into the vertex array **`vertexList`**.

    ```
    vertexList[n++] = new Vertex(NewYork, 'F');
    ```

  - Add all edges connecting to the new vertex into the adjacency matrix or list.

    ```
    adjMat[1][n-1] = 1; adjMat[n-1][1] = 1;
    lists[n-1].append(1); lists[1].append(n-1);
    ```

# Searches

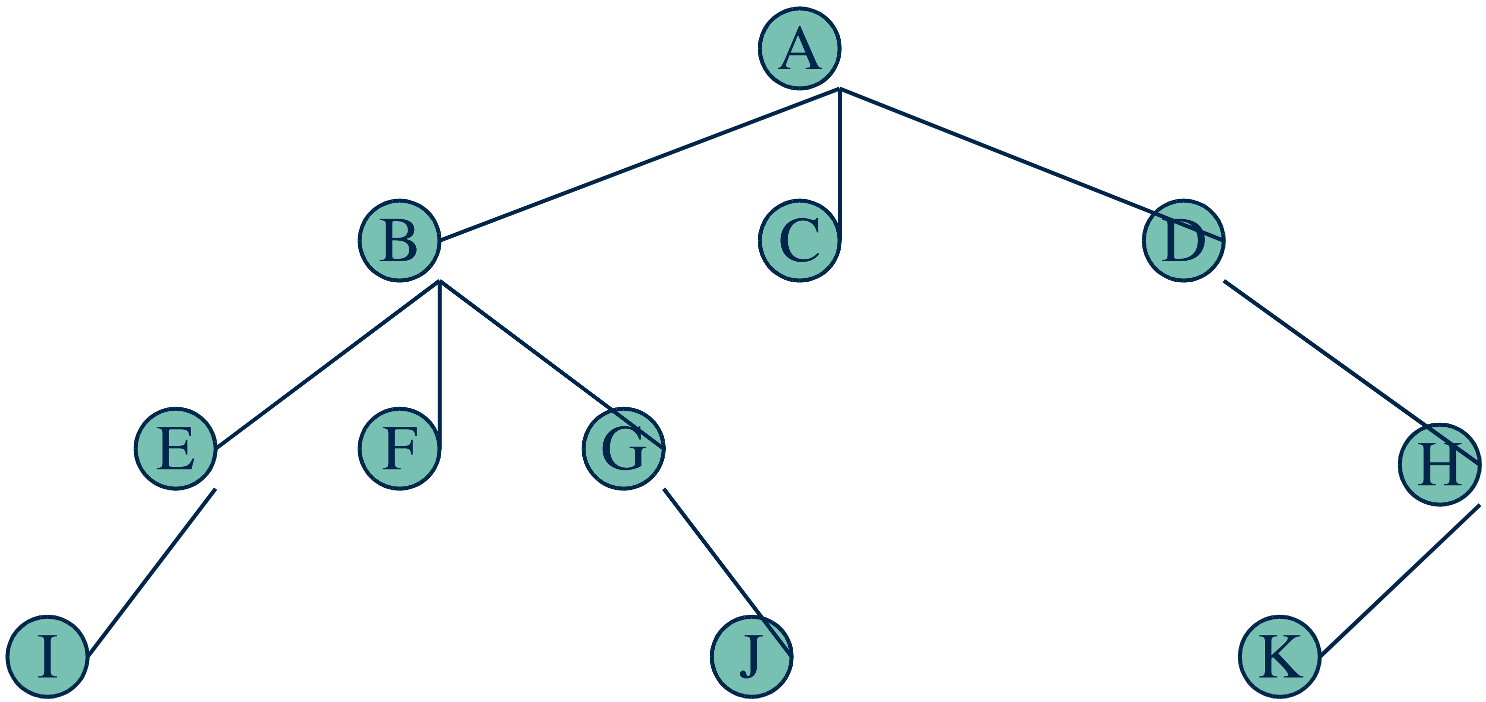
- Finding which vertices can be reached from a specified vertex.
- One of the most fundatmental operations on a graph.
- An algorithm which starts at any specified vertex, systematically moves along edges to other vertices such that when it's done, you are guaranteed to have visited every connected vertex.

# Depth-First Search

- As the name suggests, the algorithm will try to go down (to an adjacent vertex) for as far as possible, and then back track.
- This is especially obvious when the graph is indeed a tree (which then has a clear up-down relationship between the vertices).
- It is implemented using a stack to remember where it was when a dead-end is reached.

# Example

What is the depth first traversal order



# The DFS Algorithm

- If possible, visit an adjacent unvisited vertex, mark it, then push it onto the stack.
- If you can’t follow rule 1, then if possible, pop a vertex off the stack.
- If you can’t follow rule 1 or rule 2, you are done.

# Breadth-First Search

- Instead of trying to get as far away from the start point, we can also try to stay as close as possible.
- In tree terminology, we want to visit all nodes level by level.
- BFS for the previous example??
- BFS is implemented with a queue.

# The BFS Algorithm

Make start vertex current vertex, visit it.

- If possible, visit the next unvisited vertex that is adjacent to the current one, mark it and insert it into the queue.
- If you can't carry out rule 1, remove a vertex from the queue and make it the current vertex.
- If you can't carry out rule 1 or 2, you are done.

# Classic Graph Problems

- The 7 bridges of Koenigsberg Problem
  - Is there a path by which I can cross every bridge exactly once?
  - More generally, given a graph is there a path on on which I can traverse every EDGE once
  - Also, is there a path on which I can visit every vertex once
- Traveling Salesman problem
  - Given a graph with weighted edges, what is the path with minimum weight that visits every

# Brute Force vs Analytic Solutions

- Every solvable graph theory problem has a "brute force" solution
  - Problem, brute force could take a long time
    - TSP with 100 fully connected cities srequires considering about $2^{100}$ paths
    - http://www.tsp.gatech.edu/cpapp/index.html
- Sometimes there are anlaytic soltions
  - 7 bridges of Koenigsberg Problem
    - Theorem: If a network has more than two odd vertices, it does not have an Euler path. Also, Theorem: If a network has two or less odd vertices, it has at least one Euler path.A vertex is odd if