Graphs

Outline

- Undirected graphs
  - Depth-first search (DFS) - path finding
  - Breadth-first search (BFS) - shortest path
- Directed graphs - DFS, BFS
- Directed acyclic graphs
  - Topological sort
- Shortest path finding
  - Dijkstra's algorithm
- Minimum spanning trees
  - Prim's and Kruskal's algorithms - greedy algorithms
- Shortest paths
  - Single source shortest path
    - Topological sort - acyclic graphs
    - Dijkstra - non negative weights
    - Bellman-Ford - non-negative cycles
  - Single-pair shortest path
    - A* search algorithm
  - All paris shortest path
  - Floyd-Warshall
- Dynamic programming examples
  - Bellman-Ford and Floyd-Washall
- Network flow
  - Positive edge (capacity) directed graph with a source and sink
  - Maxflow problem - flow of maximum capacity
  - Ford-Fulkerson

Background applications

- Graph = a set of vertices connected pairwise by edges
- Thousands of practical applications
- Hundreds of graph algorithms known.
- Interesting and broadly useful abstraction.
- Challenging branch of computer science and discrete math

How connected are we?

- Small-world experiment by Milgram
- 6 degrees of separation
  - 3.57 on facebook
- 6 degrees of Kevin Bacon
- Erdos number

Graph terminology

- Path - sequence of vertices connected by edges.
- Cycle - Path whose first and last vertices are the same.
- Two vertices are connected if there is a path between them.

```
// read graph from input stream.
In in = new In(args[0]);
Graph G = new Graph(in);

//print out each edge (twice)
for(int v = 0; v < G.V(); v++)
    for(int w : G.adj(v))
        stdOut.println(v + "-" + W);
```

```
//Generate/Calculate the degree of a given vertex
public static int degree(Graph G, int v){
    int degree = 0;
    for(int w : G.adj(v))
        degree++;
    return degree;
}
```

Adjacency-matrix graph representations

Two possible choices for data based representation

1. Maintain a 2-D V by V boolean array;
   for each edge v-w in graph adj[v][w] = adj[w][v] = true
   - Long iteration time O(v * w)
2. Maintain a vertex-indexed array of lists ("Bags", of objects)

In practice. Use adjacency lists.

- Algorithms based on iterating over vertices adjacent to v.
- Real-world graphs tend to be *sparse*. (Lots of vertices, small average vertex degree)

```
public class Graph {
    private final int V;
    //Adjacently lists (using Bag data type)
    private Bag<Integer>[] adj;

    public Graph(int V){
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        //Create empty graph with V vertices
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }

    public void addEdge(int v, int w){
        //add edge v-w (parallel edges and self-loops allowed)
        adj[v].add(w);
        adj[w].add(v);
    }

    public Iterable<Integer> adj(int v){
        return adj[v];
    }
}
```

Depth first Search (DFS)

Maze graph

- Vertex = intersection.
- Edge = passage.

Algorithm:

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options

DFS

- Undirected or directed graphs
    - Undirected for now
- Find all vertices connected to a given soruce vertex
- Find a path between 2 vertices
- Mark vertex v as visited
- Recursively visit all unmarked vertices adjavent to v (or pointing to v, in directed graphs)
- boolean marked[] - list visited vertices
- Integer edgeTo[] - edgeTo[w] = v, means v-w taken to visit W (for the first time)