Introduction to Graphs

- Graphs consist of two things
  - A set of items
    - Example: \{Baltimore, Chicago, Dallas, Las Vegas, New York, San Francisco\}
  - A set of connections

Problems Solved using Graphs
- Google’s PageRank Algorithm
  - Analyzes links between web pages to decide which pages are better authorities on a topic
- Networks
  - Determine how vulnerable the Internet is to attack. Is there any small subset of connections that could bring down the entire Internet?
- Kevin Bacon Game
  - How many degrees of separation are there between Audrey Hepburn and Kevin Bacon?

Graph Algorithms
- Study the most efficient algorithms that solve a particular problem
- Cost of an algorithm depends on
  - properties of the items
  - properties of the connections
- Accurate models of the types of graphs that we might face are difficult to develop
- Assessing performance
  - May consider the worst case
  - May depend on the size of the graph
  - May depend on particular observed properties
Graph Nomenclature

- A graph is
  - Vertices are indexed 0 to V-1
  - By convention
    - V is the number of vertices
    - E is the number of edges

Simple Graphs

- Graphs that disallow two properties
  - Parallel edges
  - Self-loops

- Property
  - The number of edges in a simple graph of V vertices is at most V(V-1)/2

More Nomenclature

- Another word for vertex
- Other words for edge

- Adjacent –
- Degree of a vertex –

- Notation v-w represents

- Subgraph –

- Graphs are unordered sets of vertices and edges so there are many graphical representations of the same graph

Consider \( V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \) and \( E = \{0-5, 4-3, 0-1, 9-12, 6-4, 5-4, 0-2, 11-12, 9-10, 0-6, 7-8, 9-11, 5-3\} \)
Graph Types
- Euclidean graphs –

- Non-Euclidean graphs represent relationships, schedules, or connectivity
  o Much more common
- Isomorphic graphs –
  o This is a hard problem!

Graph Attributes
- A path is a sequence of vertices in which each successive vertex (after the first) is adjacent to its predecessor in the path.
  o Simple path – vertices and edges are distinct
  o Cycle – a path that is simple except that the first and final vertices are the same
- Cyclic path – a path whose first and final vertices are the same, but it is not necessarily simple
- Tour – a cyclic path that includes every vertex
- Two paths are disjoint if they have no vertices in common, other than, possibly, their endpoints.
- Connected graph – graphs where there is a path from every vertex to every other vertex in the graph
- Maximal connected subgraph – there is no path from a subgraph vertex to any vertex in the graph that is not in the subgraph
  o Known as connected components

Acyclic Graph
- Tree – a graph without any cycles
  o A graph $G$ with $V$ vertices is a tree if and only if it satisfies any of the following conditions
    ▪ $G$ has $V-1$ edges and no cycles
    ▪ $G$ has $V-1$ edges and is connected
    ▪ Exactly one simple path connects each pair of vertices in $G$
    ▪ $G$ is connected, but removing any edge disconnects it
- Forest – a set of trees
- Spanning Tree – a subgraph that contains all the vertices and is a single tree
- Spanning Forest – a subgraph that contains all the vertices and is a forest

More Graph Types
- Complete graph – a graph where all possible edges are present
- Complement of graph $G$ – begins with complete graph and removes all the edges of $G$
- Union – graph induced by the union of the set of edges
- Clique – a complete subgraph
Graph Density
- Average vertex degree \((2E/V)\)
  - Dense graph – average vertex degree is proportional to \(V\) (\(E\) is proportional to \(V^2\))
  - Sparse graph – a graph whose complement is dense
- Key factor in selecting an efficient algorithm to process the graph
  - Consider an two algorithms that solve the same problem with running times
    - \(O(V^2)\)
    - \(O(E \log E)\)

Directed Graphs
- Edges are one-way (directed edge)
  - Edges defined as an ordered pair
    - first vertex – source
    - second vertex – destination
- Vertices have in-degrees and out-degrees

Graph ADT
- Basic Graph operations
  - create (number of vertices, isDirected)
  - destroy
  - get number of edges
  - get number of vertices
  - insert an edge
  - remove an edge
  - tell whether an edge exists between two vertices
A Possible C Implementation

// File: graph.h

typedef struct {
    int from, to;
} Edge;

Edge createEdge(int, int);

typedef struct graph *Graph;

Graph graphInit(int);
void graphInsertEdge(Graph, Edge);
void graphRemoveEdge(Graph, Edge);
int graphEdges(Edge [], Graph);  // Creates an array of
    // edges

Graph graphCopy(Graph);
void graphDestroy(Graph);
bool graphIsEdge(Graph, int, int);
void graphShow(Graph);
Graph graphRand(int, int);

Using the ADT

// Program takes the number of vertices and edges as
// arguments, generates a random graph, and displays it
#include <stdlib.h>
#include <stdio.h>
#include "graph.h"

int main(int argc, char **argv)
{
    if (argc < 3)
    {
        fprintf(stderr, "Usage: randGraph <# vertices
        " > <# edges>n");
        exit(1);
    }

    int vertices = atoi(argv[1]);
    int edges = atoi(argv[2]);
    Graph myGraph = graphRand(vertices, edges);

    if (vertices < 20)
        graphShow(myGraph);
    else
        printf("%d vertices, %d edges\n", vertices, edges);
}