CS106X
Winter 2017

Chris Gregg

Section Handout #8

This week has practice with graph strctures, including graph properties and algorithms that act on graphs. Assume the following structures have been declared:

Vertex

```
struct Vertex {
  string name;
  Set<Edge *> edges;
  double cost;
  bool visited;
  Vertex *previous;
};
```

Edge

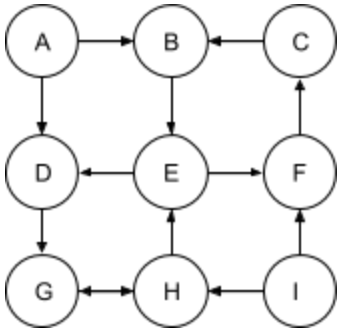
```
struct Edge {
  Vertex *start;
  Vertex *finish;
  double cost;
  bool visited;
}
```

1. Graph Properties

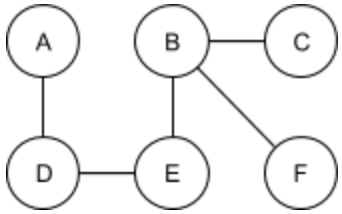
For each of the graphs shown below, answer the following questions.

a) Is the graph directed or undirected?
b) Is the graph weighted or unweighted?
c) Which graphs are connected, and which are not? Is any graph **strongly** connected?
d) Which graphs are cyclic, and which are acyclic?
e) What is the degree of each vertex? (If directed, what is the in-degree and the out-degree?)

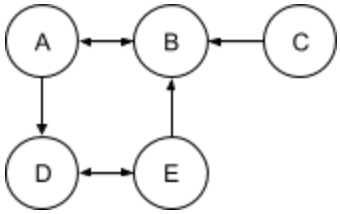
Graph 1



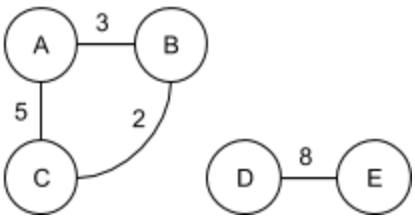
Graph 2



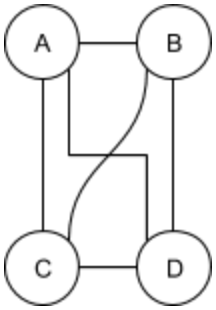
Graph 3



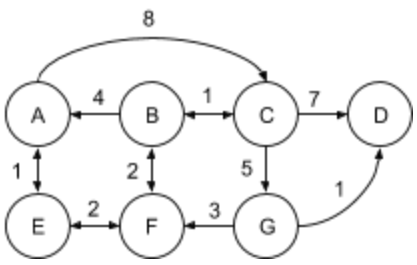
Graph 4



Graph 5



Graph 6



Thanks to Marty Stepp, Victoria Kirst, Jerry Cain, and other past CS106 instructors for contributing content on this handout.

2. Depth-First Search (DFS)
Write the paths that a depth-first search would find from vertex A to all other vertices in the following graphcs. If a given vertex is not reachable from vertex A, write "no path" or "unreachable."

- in Graph 1
- in Graph 6

3. Breadth-First Search (BFS)
Write the paths that a depth-first search would find from vertex A to all other vertices in the following graphcs. If a given vertex is not reachable from vertex A, write "no path" or "unreachable."

- in Graph 1
- in Graph 6

4. Minimum Weight Paths
Which paths found by DFS and BFS on Graph 6 in the previous problems are not minimal weight? What are the minimal weight paths from vertex A to all other nodes?

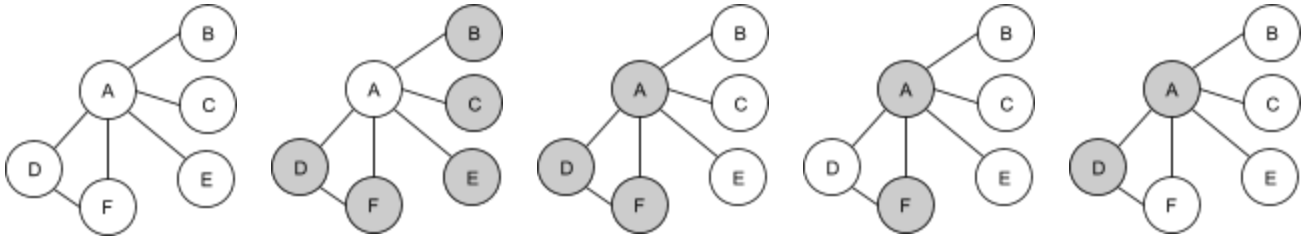
5. *k*th Level Friends
Imagine a graph of Facebook friends, where users are vertices and friendship are edges. Write a function that takes in a social network graph, a Vertex in that graph, and a value k and returns the set of people who are exactly k hops away from the Vertex (and not fewer). For example, if $k = 1$, those are v's direct friends; if $k = 2$, they are your friends-of-friends. If $k = 0$, return a set containing only the user. Assume input arguments are valid.

6. Has a Cycle
Write a function that returns true if a graph contains any cycles, or false if not.

7. Is It Connected?
Write a function that takes in a graph and returns true if a path can be made from every vertex to any other vertex, or false if there is any vertex that cannot be reached by a path from some other vertex. An empty graph is defined as being connected.

8. Minimum Vertex Cover
A *vertex cover* is a subset of an undirected graph's vertices such that each and every edge in the graph is incident to at least one vertex in the subset. A *minimum vertex cover* is a vertex cover of the smallest possible size (where the size is determined by the number of nodes in the cover). Suppose you have the following graph on the left.

Graph **Vertex Covers**



Each of the four illustrations on the right shows some vertex cover where shaded nodes are included in the vertex cover and hollow ones are excluded. Each one is a vertex cover because each edge touches at least one vertex in the cover. The two vertex covers on the right are minimum vertex covers because you can't have a vertex cover with fewer than two nodes.

Write a function that, given a graph, returns a minimum vertex cover for the graph. If there are multiple minimum vertex covers, you can return an arbitrary one.

9. Game of Thrones

Recall that a *complete* graph is an undirected graph where every single graph node is connected to every other node. A *tournament graph* is a directed graph that comes from a complete graph where you impose a direction on each and every arc. Informally, a tournament graph is a summary of who prevailed over whom in an exhaustive competition of one-on-one matches, where every single person eventually competes – exactly once – against everyone else. Below, on the left, is a complete graph on five nodes, and on the right is one possible tournament graph that can be derived from the complete graph.

The tournament graph on the above right states that player 1 beat players 2, 3, and 4 (but not 5), that player 2 lost to everybody, and so on.

A *tournament king* is a node in a tournament representing someone who, for every other player, either directly prevailed over that player, or prevailed over someone who prevailed over that player. In other words, a node is a king if one can travel from it to every other node via a path at most 2 arcs. In the above example, players 1, 3, and 5 are all kings, but players 2 and 4 are not. Take some time to figure out why that is. Then, write a function that, given a tournament graph, returns the tournament kings for that graph.