# 2nd tutorial



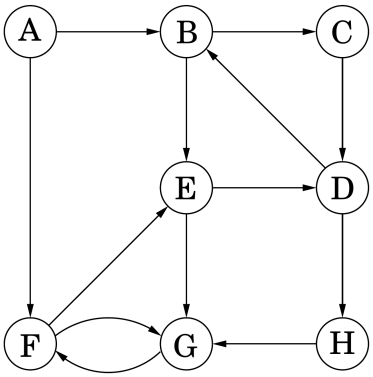
**1. Representation.** We know three basic graph representations – list of edges, list of neighbors, and the adjacency matrix. Write out the complexity of the following standard operations:

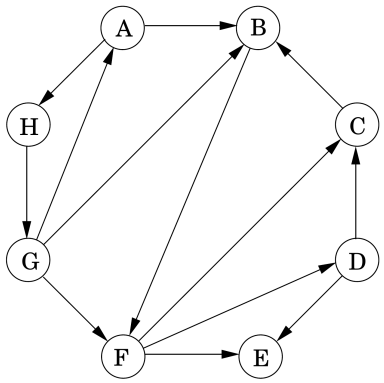
- `add-edge(u,v)`
- `delete-edge(u,v)`
- `neighbors(u,v)`
- `degree(v)`
- `neighbor-list(u,v)`

*bonus:* What if in the "list of neighbors" representation you use a dictionary (implemented as a hash table) instead of a linked list? If you don't know what a dictionary is, skip the question.

**2. Boss.** An adjacency matrix of a graph $G$ is preloaded into memory. Find a *boss* in $G$ or say that it doesn't contain any boss. A boss is a vertex who has an edge to all other vertices, and no edge is coming in to it.

**3. DFS Runs.** Perform depth-first search on each of the following graphs; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, forward edge, back edge, or cross edge, and give the `pre` and `post` number of each vertex.




**4. Cycle with given edge or vertex.** You're given a graph $G$ and an edge $e$ in it. How to detect whether $G$ has a cycle containing $e$?
If given a vertex $v$ instead of an edge $e$, how to detect whether $G$ has a cycle containing $v$?

**5. Peeling a graph.** You're given a connected undirected graph $G$. What is the fastest way to determine an order of the vertices $v_1, \ldots, v_n$ such that for each $i$, after we have removed $v_1, \ldots, v_i$, the remaining graph is still connected?

**6. Pouring Water.** We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

(1) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.
(2) What algorithm should be applied to solve the problem?
(3) Find the answer by applying the algorithm.

**7. Is it bipartite?** Design an algorithm which decides whether a given graph $G$ is bipartite, that is, whether its vertices can be partitioned into sets $A, B$ such that every edge only goes between those sets (i.e., there is no edge between two vertices of $A$ or two vertices of $B$).