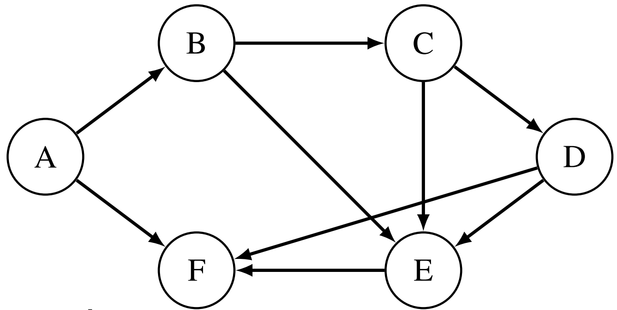
Discussion 12: Graphs



1 Graph Representation

Represent the graph with edge list (a.k.a. adjacency list) and adjacency matrix representation.

NOTE: Edge lists and adjacency lists are not the same! That was a mistake. An edge list is like a linked list (see lecture), and and an adjacency list is more of a table that lists the adjacent vertices for each vertex in the graph. Graphs are commonly represented using adjacency lists and matrices.

Adjacency List

A → [B, F]
B → [C, E]
C → [D, E]
D → [E, F]
E → [F]
F → []

FROM \ TO	A	B	C	D	E	F
A	F	T	F	F	F	T
B	F	F	T	F	T	T
C	F	F	F	T	T	F
D	F	F	F	F	T	T
E	F	F	F	F	F	F
F	F	F	F	F	F	F

2 Searches and Traversals

Run depth first search (DFS) and breadth first search (BFS) on the graph, starting from node A. List the order in which each node is traversed. Whenever there is a choice of which node to visit next, break ties alphabetically (choosing earlier values).

DFS preorder: A, B, C, D, E, F
DFS postorder: F, E, D, C, B, A
BFS: A, B, F, C, E, D

As an exercise, if we replace $E \to F$ with $B \to F$, we get:
DFS preorder: A, B, C, D, E, F
DFS postorder: E, F, D, C, B, A
BFS: A, B, F, C, E, D

3 Topological Sorting

Give a valid topological ordering of the graph. Is the topological ordering of the graph unique?

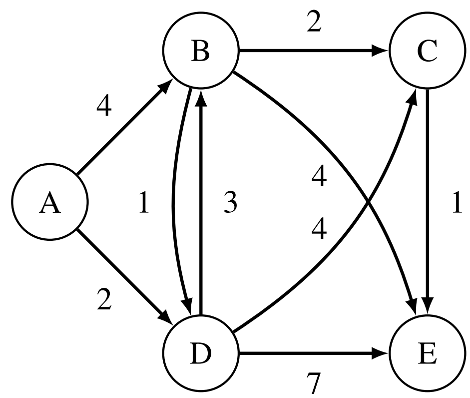
One valid ordering: A, B, C, D, E, F
The ordering is unique.

As an exercise, if we replace $E \to F$ with $B \to F$, we get the following as valid topological orderings:
A, B, C, D, E, F
A, B, C, D, F, E

4 Dijkstra's Algorithm

Given the following graph, write down the value `dist(v)` for all vertices v during each iteration of Dijkstra's algorithm, starting at node A.

		dist(v)				
v	Init	Pop A	Pop D	Pop B	Pop C	Pop E
A	∞	**0**	0	0	0	0
B	∞	4	4	**4**	4	4
C	∞	∞	6	6	**6**	6
D	∞	2	**2**	2	2	2
E	∞	∞	9	8	7	**7**



5 Exercise: Bipartite Graphs

An undirected graph is a bipartite graph if its vertices can be separated into two disjoint sets such that each edge in the graph spans both sets (is connected to a vertex in each set). Given a connected graph G, fill in the method below so that it returns `True` iff G is a bipartite graph.

```
public static boolean isBipartite(Graph G) {
    Node start = getRandomNode(G);
    // This may have been misleading; VISITED tells us the set for each node
    HashMap<Node, Boolean> visited = new HashMap<Node, Boolean>();
    ArrayList<Node> fringe = new ArrayList<Node>();
    visited.put(start, true);
    fringe.add(start);
    while (!fringe.isEmpty()) {
        Node n = fringe.pop();
        boolean curr = visited.get(n);
        for (Node neighbor: n.neighbors()) {
            if (visited.contains(neighbor)
                && visited.get(neighbor) == curr)
                return false;
            else {
                visited.put(neighbor, !curr);
                fringe.add(neighbor);
            }
        }
    }
    return true;
}
```