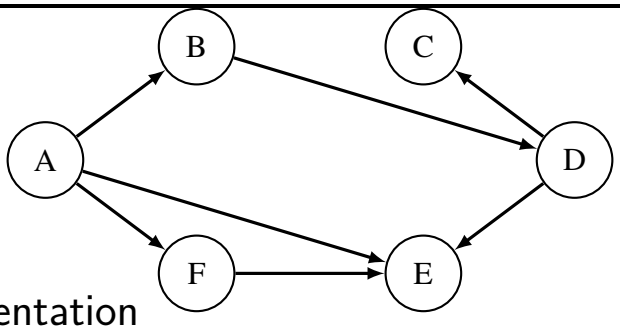


1 Graph Representation

Represent the graph above with an adjacency list and an adjacency matrix representation.

NOTE: Edge lists and adjacency lists are not the same! 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 but be aware of what is meant by an edge list. Here, we stick with a convention that nodes don't have an edge to themselves; conventions vary.

Adjacency List

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

FROM

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

2 Searches and Traversals

Run depth first search (DFS) and breadth first search (BFS) on the graph above, starting from node A. List the order in which each node is first visited. Whenever there is a choice of which node to visit next, visit nodes in alphabetical order.

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

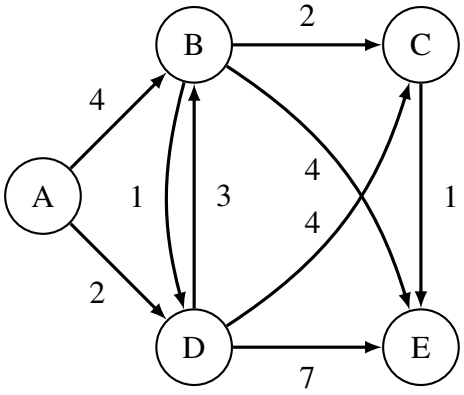
3 Topological Sorting

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

One valid ordering: A, B, D, C, F, E
The ordering is not unique. One way to approach this problem is to take any node with no edges leading to it and return it as the next node. After returning a node, we delete it and any edges leaving from it and look for a node with no incoming edges in the updated graph. We can repeat this until we have no nodes left. If at any point in this process we have a multiple choices for which node to return then the topological ordering is not unique.

4 Dijkstra's Algorithm

Given the following graph, run Dijkstra's algorithm starting at node A. At each step, write down the entire state of the algorithm. This includes the value `dist(v)` for all vertices v for that iteration as well as what node was popped off of the fringe for that iteration.

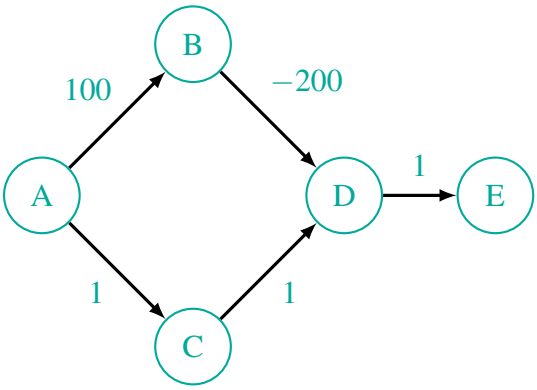


	`dist(v)`					
v	Init	Pop A	Pop D	Pop B	Pop C	Pop E
A	**0**	0	0	0	0	0
B	∞	4	4	**4**	4	4
C	∞	∞	6	6	**6**	6
D	∞	2	**2**	2	2	2
E	∞	∞	9	8	7	**7**

5 Dijkstra's Correctness

What must be true about our graph in order to guarantee Dijkstra's will return the shortest path's tree to every vertex? Draw an example of a graph that demonstrates why Dijkstra's might fail if we do not satisfy this condition.

In order to guarantee Dijkstra's will return the shortest path to every vertex, we must have a graph that has no negative edge weights. Take the following graph as an example of why negative edge weights might cause an error:



For this graph, if we ran Dijkstra's starting from A, then we would get the incorrect shortest path to E since we would choose the bottom path through C instead of the top path through B. Note that having negative edge weights does not guarantee Dijkstra's will fail, but if we have all non-negative edge weights then we are guaranteed to get the shortest path. This is great if we want to use Dijkstra's on graphs that represent distances in real life since those are all positive!