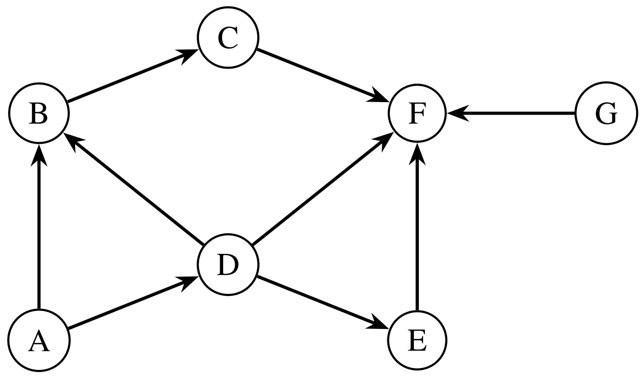
CS 61B — Discussion 11 — Spring 2016



1 Graph Representations

Write the graph above as an adjacency matrix, then as an adjacency list.

```
Matrix:

  A B C D E F G <- end node
A 0 1 0 1 0 0 0
B 0 0 1 0 0 0 0
C 0 0 0 0 0 1 0
D 0 1 0 0 1 1 0
E 0 0 0 0 0 1 0
F 0 0 0 0 0 0 0
G 0 0 0 0 0 1 0
^ start node

List:

A: {B, D}
B: {C}
C: {F}
D: {B, E, F}
E: {F}
F: {}
G: {F}
```

2 DFS and BFS

Give the DFS preorder, DFS postorder, and BFS order of the graph starting from vertex A. Break ties alphabetically.

```
DFS preorder: ABCFDE
DFS postorder: FCBEDA
BFS: ABDCEF
```

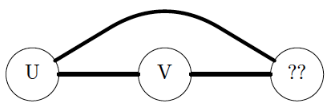
3 Topological Sorting

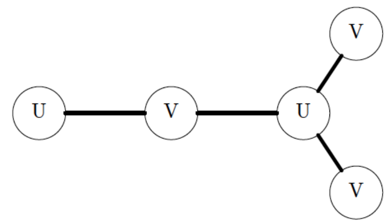
Give a valid topological sort of the graph above. (Hint: Use the reverse postorder.)

One valid topological sort is GADEBCF. There are many others. In particular, G can go anywhere except after F, since it has no incoming edges and only one outgoing edge (to F).

4 Graph Algorithm Design: Bipartite Graphs

An undirected graph is said to be bipartite if all of its vertices can be divided into two disjoint sets U and V such that every edge connects an item in U to an item in V. For example, the graph on the left is bipartite, whereas on the graph on the left is not. Provide an algorithm which determines whether or not a graph is bipartite. What is the runtime of your algorithm?





To solve **this** problem, we simply run a special version of DFS or BFS from any vertex. This special version marks the start vertex with a U, then each of its children with a V, and each of their children with a U, and so forth. If any vertex already has a U and the visited vertex has a V (or vice-versa), then the graph is not bipartite.

If the graph is not connected, we repeat **this** process **for** each connected component.

If the algorithm completes, marking every vertex in the graph, then it is bipartite.

5 Extra for Experts: Shortest Directed Cycles

Provide an algorithm that finds the shortest directed cycle in a graph in $O(EV)$ time and $O(E)$ space, assuming $E > V$.

The key realization here is that the shortest directed cycle involving a particular source vertex is just some shortest path plus one edge back to s. Using **this** knowledge, we can create a shortestCycleFromSource(s) subroutine. This subroutine first runs BFS on s, then checks every edge in the graph to see **if** it points at s. For each such edge originating at vertex v, it computes the cycle length by adding one to distTo(x) (which was computed by BFS).

This subroutine takes $O(E+V)$ time because it is BFS. To find the shortest cycle in the entire graph, we simply call shortestCycleFromSource() **for** each vertex, resulting in an $V * O(E+V) = O(EV+V^2)$ runtime. Since $E > V$, **this** is just $O(EV)$.

6 Extra for Experts: DFS Gone Wrong

Consider the following implementation of DFS, which contains a crucial error:

```
create the fringe, which is an empty Stack
    push the start vertex onto the fringe and mark it
    while the fringe is not empty:
        pop a vertex off the fringe and visit it
        for each neighbor of the vertex:
            if neighbor not marked:
                push neighbor onto the fringe
                mark neighbor
```

Give an example of a graph where this algorithm may not traverse in DFS order.