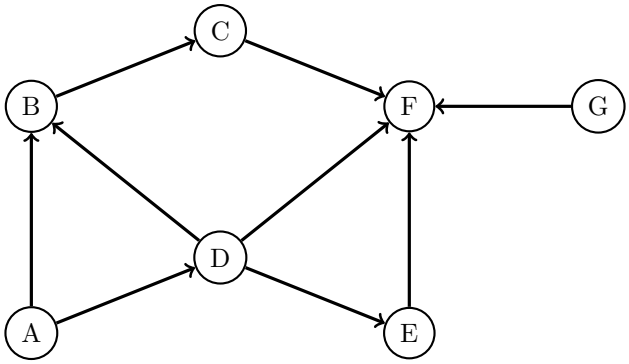
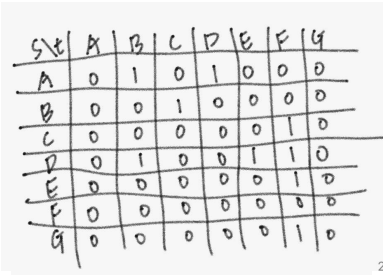
CS 61B Spring 2018

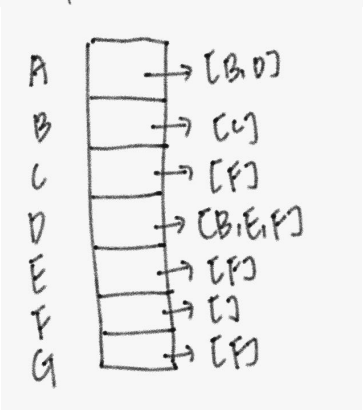
Graphs

Discussion 11: April 3, 2018

Graphs





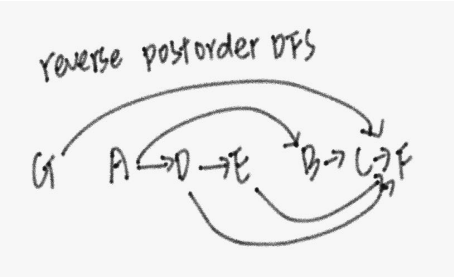


1.1 Write the graph above as an adjacency matrix, then as an adjacency list. What would be different if the graph were undirected instead?

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

DFS preorder: A-B-C-F-D-E
DFS postorder: F-C-B-E-D-A
BFS order: A-B-D-C-E-F

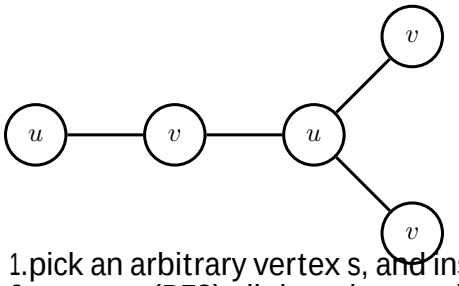
1.3 Give a valid topological sort of the graph. (*Hint*: Consider the reverse postorder of the whole graph.)

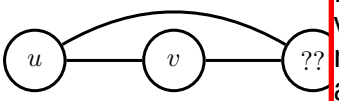


Graph Algorithm Design

we don't need this at all!

2.1 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 below, the graph on the left is bipartite, whereas on the graph on the right is not. Provide an algorithm which determines whether or not a graph is bipartite. What is the runtime of your algorithm?





the algorithm can be simplified with mark (keeping an extra array of size V) . we traverse E times and mark a vertex to be u or v. When we want to mark a vertex to be u but it has already been marked v, or vice versa, return false.->O(V+E)

1.pick an arbitrary vertex s, and insert it into U
2.traverse (BFS) all the other vertices E times
if their distance to s is even, insert them to U; otherwise V
3. if U and V contain some same vertices, return false; otherwise true
O(E^2)

the solution is traversal s(O(V)), find all vertices that have an edge pointing to s (O(E)), and run BFS on all those vertices (O (V+E)) ->O(EV)

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

keep track of a length = INT_MAX
for every vertice v:
do DFS & incrementing depth:
if the path cannot have a second v->continue;otherwise if length > depth ->length = depth
return length I think my solution is okay, although the space complexity is not O(E)

2.3 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
```

we should only mark a vertex as true when visiting it

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