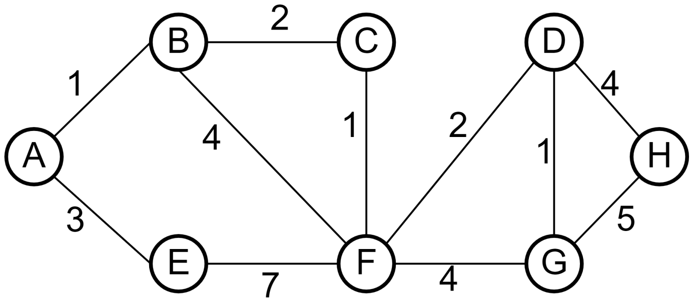
CS 61B Fall 2020 — DFS, BFS, Shortest Paths

Exam Prep Discussion 10: October 26, 2020

1 DFS, BFS, Dijkstra's, A*

For the following questions, use the graph below and assume that we break ties by visiting lexicographically earlier nodes first.



(a) Give the depth first search preorder traversal starting from vertex A.

A, B, C, F, D, G, H, E

(b) Give the depth first search postorder traversal starting from vertex A.

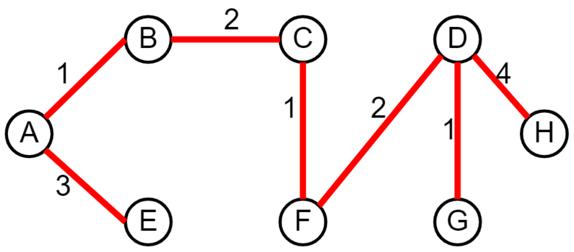
H, G, D, E, F, C, B, A

(c) Give the breadth first search traversal starting from vertex A.

A, B, E, C, F, D, G, H

(d) Give the order in which Dijkstra's Algorithm would visit each vertex, starting from vertex A. Sketch the resulting shortest paths tree.

A, B, C, E, F, D, G, H



(e) Give the path A* search would return, starting from A and with G as a goal. Let $h(u, v)$ be the valued returned by the heuristic for nodes u and v.

u	v	$h(u,v)$
A	G	9
B	G	7
C	G	4
D	G	1
E	G	10
F	G	3
H	G	5

$A \to B, B \to C, C \to F, F \to D, D \to G$

2 Graph Conceptuals

Answer the following questions as either **True** or **False** and provide a brief explanation:

1. If a graph with n vertices has $n-1$ edges, it **must** be a tree.

   False. The graph **must** be connected.

2. The adjacency matrix representation is **typically** better than the adjacency list representation when the graph is very connected.

   True. The adjacency matrix representation is usually worse than the adjacency list representation with regards to space, scanning a vertex's neighbors, and full graph scans. However, when the graph is very connected, the adjacency matrix representation has roughly same asymptotic runtime in these operations, while "winning" in operations like hasEdge.

3. Every edge is looked at exactly twice in **every** iteration of DFS on a connected, undirected graph.

   True. The two vertices the edge is connecting will look at that edge when it's their turn.

4. In BFS, let $d(v)$ be the minimum number of edges between a vertex v and the start vertex. For any two vertices u, v in the fringe, $|d(u) - d(v)|$ is **always less than** 2.

   True. Suppose this wasn't the case. Then, we could have a vertex 2 edges away and a vertex 4 edges away in the fringe at the same time. But, the only way to have a vertex 4 edges away is if a vertex 3 edges away was removed from the fringe. We see this could never occur because the vertex 2 edges away would be removed before the vertex 3 edges away!

5. Given a fully connected, directed graph (a directed edge exists between every pair of vertices), a topological sort can never exist.

   False. Consider the graph constructed as follows: for all vertices i, j such that $i < j$, draw a directed edge from i to j. A valid topological ordering of this graph is simply enumerating the vertices: $1, 2, 3, \ldots .N$.

3 Cycle Detection

Given an undirected graph, provide an algorithm that returns true if a cycle exists in the graph, and false otherwise. Also, provide a Θ bound for the worst case runtime of your algorithm. You may use either an adjacency list or an adjacency matrix to represent your graph. (We are looking for an answer in plain English, not code).

We do a depth first search traversal through the graph. While we recurse, if we visit a node that we visited already, then we've found a cycle. Assuming integer labels, we can use something like a `visited` boolean array to keep track of the elements that we've seen, and while looking through a node's neighbors, if `visited` gives true, then that indicates a cycle. Note that this algorithm differs slightly if the graph is directed. If the graph is directed, then there is a cycle if a node has already been visited *and it is still in the recursive call stack (i.e. still in the fringe and not popped off yet).*

In the worst case, we have to explore V edges to find a cycle (number of edges doesn't matter). So, this runs in $\Theta(V)$.