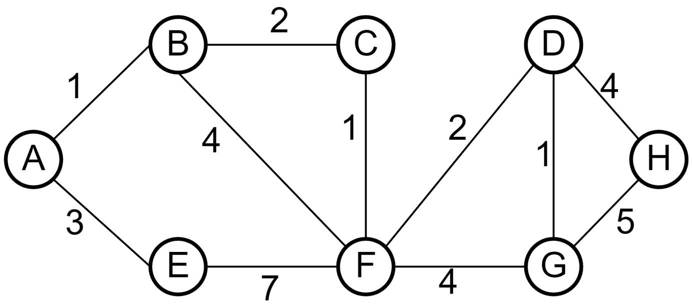
DFS, BFS, Dijkstra's, A*

Here is a video walkthrough of the solutions.

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

Explanation: Preorder visits the current node, then recursively calls on each of its children. The chain of calls looks like this:

```
dfs(A)
    dfs(B)
        dfs(C)
            dfs(F)
                dfs(D)
                    dfs(G)
                        dfs(H)
                dfs(E)
```

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

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

Explanation: Postorder recurses on all children then visits the current node. See above for the order of calls.

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

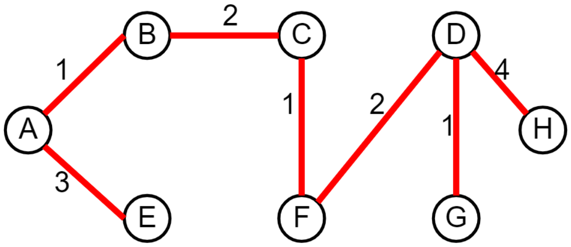
A, B, E, C, F, D, G, H

Explanation: BFS visits nodes in increasing distance (by number edges) from the source (ties broken alphabetically as given in the instructions). The groups in increasing distance from A are: 0: (A), 1: (B, E), 2: (C, F), 3: (D, G), 4: (H). Alternatively, draw out the queue to confirm this ordering for yourself.

(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

Explanation: Dijkstra's visits nodes in increasing distance from the source (weighted). This order is: A: 0, B: 1, C: 3, E: 3, F: 4, D: 6, G: 7, H: 10. You can confirm this ordering is the same as manually running Dijkstra's.



(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

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

Explanation: Note that this heuristic is not admissible so it may not return the shortest path ($h(u, v) > dist(A, G) = 7$). The A* execution trace is as follows:

(a) Pop off A.

```
pq = [B: (3, 8), E: (3, 13)] // (dist, dist + h)
prev = [B: A, E: A]
```

(b) Pop off B.

```
pq = [C: (3, 7), F: (5, 8), E: (3, 13)]
prev = [B: A, C: B, E: A, F: B]
```

(c) Pop off C.

```
pq = [F: (4, 7), E: (3, 13)]
prev = [B: A, C: B, E: A, F: C]
```

(d) Pop off F.

```
pq = [D: (6, 7), E: (3, 13)]
prev = [B: A, C: B, D: F, E: A, F: C]
```

(e) Pop off D.

```
pq = [G: (7, 7), H: (10, 15), E: (3, 13)]
prev = [B: A, C: B, D: F, E: A, F: C, G: D, H: D]
```

(f) Pop off G. Race the `prev` pointers to get the path.