1. Reflection
    a. Agent architectures and control
        If I had to be really honest, I really liked this module of the course. I always enjoy learning more about the inner workings of how artificial intelligence works. Here, I learned about artificial intelligence keeping track of all stimuli they receive in the percept trace as well as a history of the commands that they want to execute called a command trace. I also learned about what's called a belief state function. The proof was very intuitive! I really liked the steps that were taken to make the formula for a basic belief state function work. For instance, even if you'd get the same utility from a particular percept, the belief state function states that the one received later will be worth less.
    b. Searching and graphs
        A lot of the stuff covered in this module was stuff that I already knew (mostly because of 341 and 441). However, learning about what a State Space Search was fascinating. I've always wondered how, what goes into an artificial intelligence's mind when it tries to solve complex problems like pouring water into a jug puzzle. Now, I learned that artificial intelligence keeps track of a state of a problem, and traces them through some kind of graph. The artificial intelligence will keep traversing through the graph until it finds itself in a state that matches the goal state. I can see many useful applications of this, particularly path finding in a maze. I can see it now! The artificial intelligence starts at a particular coordinate, and keeps traversing to a new spot on the maze. It keeps doing this until its state (i.e. its coordinates) matches that of the goal state (the coordinates of the end of the maze).

c. Uninformed search strategies
   Much like the previous module, a lot of the stuff covered here was stuff that I already learned in previous classes. However, learning about the limitations of each search strategy was kinda new to me. For instance, I never really thought too much about it, but I guess it does make sense as to why depth first search doesn't a solution (if there is one) but BFS does. Since DFS uses a stack, it's possible that DFS could take the path that leads to an infinite amount of nodes (thus never leaving the branch). However, since BFS uses a queue, it can guarantee a solution since all nodes will be visited at some point.
d. Informed search strategies
   Out of all the modules so far, this was by far the most interesting! I really learned a lot of new things here, particularly how Best First and A* Search works (my favorite being A*). I like the idea of implementing a heuristic to each node as a 2nd metric. This heuristic basically just tells you how far a given state is from the goal state (a heuristic of 0 implies the goal state). Knowing how A* Search works, I can finally see how GPS like Google Maps work! Not only is the program trying to find the shortest path between where you are and your destination, but is also takes into consideration which path leads you physically closer to the destination (the heuristic part of the algorithm).

2. State-space search

a. $V \in \{a, b, c, d, e, f, s, g1, g2, g3\}$
   $E \in \{(a, g1, 9), (a, b, 3), (b, a, 2), (b, c, 1), (c, s, 6), (c, f, 7), (c, g2, 5), (d, c, 2),$
   $(d, s, 1), (d, e, 2), (e, g3, 7), (f, d, 2), (f, g3, 8), (s, a, 5), (s, b, 9), (s, d, 6)\}$
b. a:2 b:2 c:3 d:3 e:1 f:2 s:3 g1:0 g2:0 g3:0
c. $Branching\ Factor = (2 + 2 + 3 + 3 + 1 + 2 + 3 + 0 + 0 + 0) / 10 = 1.6$
d. s→g1: s→a→g1 [Cost = 14]
   s→g2: s→d→c→g2 [Cost = 13]
   s→g3: s→d→e→g3 [Cost = 15]

e. The heuristic function **is admissible**. In order for a heuristic function to be admissible, all heuristics mustn't overestimate the actual cost. To prove it, here are all the lowest costs for each node to reach the "closest" (lowest cost) goal state (either g1, g2, or g3).

Node	Heuristic	Lowest Cost	Statement
a	7	9	7 < 9
b	3	6	3 < 6
c	4	5	4 < 5
d	6	7	6 < 7
e	5	7	5 < 7
f	6	8	6 < 8
s	5	14	5 < 14
g1	0	0	0 <= 0
g2	0	0	0 <= 0
g3	0	0	0 <= 0

Since all heuristic values don't overestimate, we can say that the heuristic function is admissible.

f. Path Finding

i.

Breadth First Search

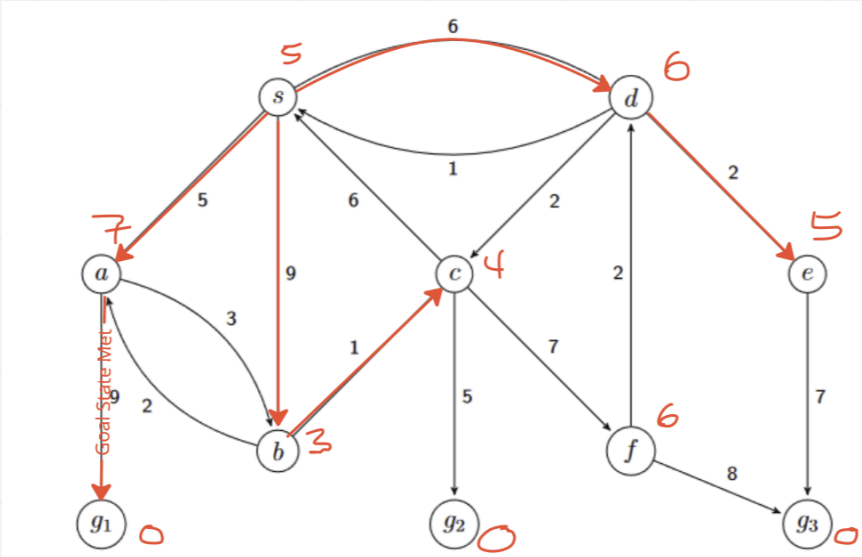

Figure 1

Queue: [~~s~~, ~~a~~, ~~b~~, ~~d~~, ~~b~~, ~~g1~~, c, c, e]

Visited: [s, a, b, d, g1]

ii.

Depth First Search

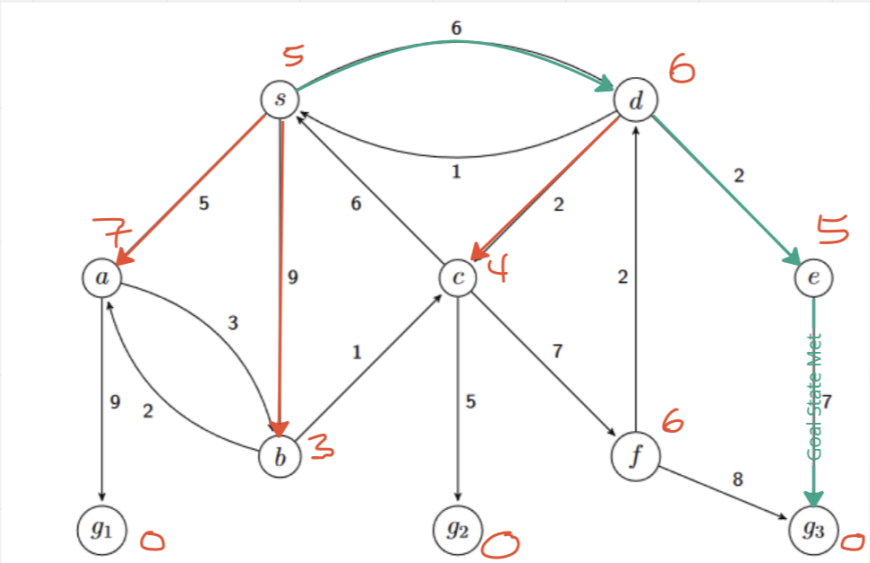

Figure 1

Stack: [~~s~~, a, b, ~~d~~, c, ~~e~~, ~~g3~~]

Visited: [s, d, e, g3]

iii.

Uniform Cost Search (Lowest First Cost Search)

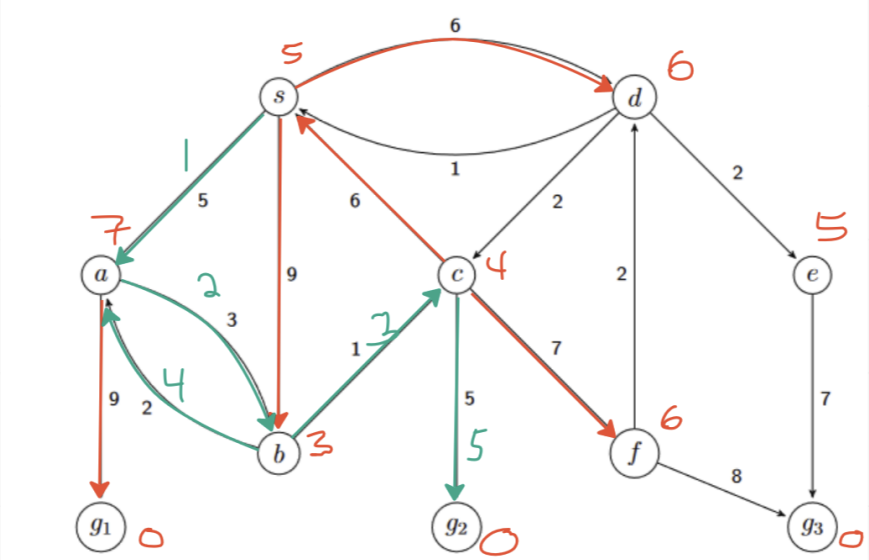

Figure 1

Priority Queue: [s, a, b, d, g1, c, g2, f]

Visited: [s, a, b, c, g2]

iv.

Greedy Best-First Search using h1

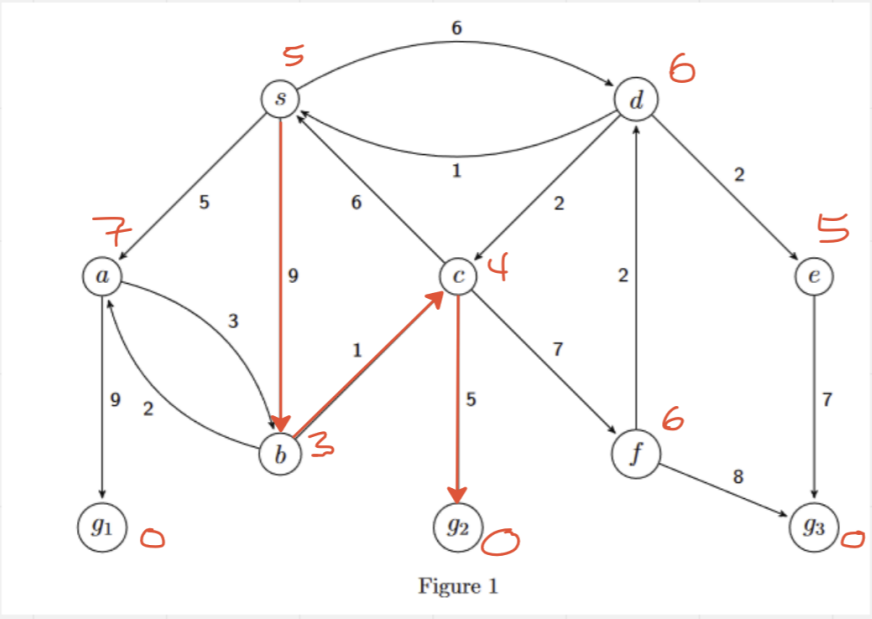

Figure 1

Visited: [s, b ,c, g2]

v.

A/A* Search Using h1

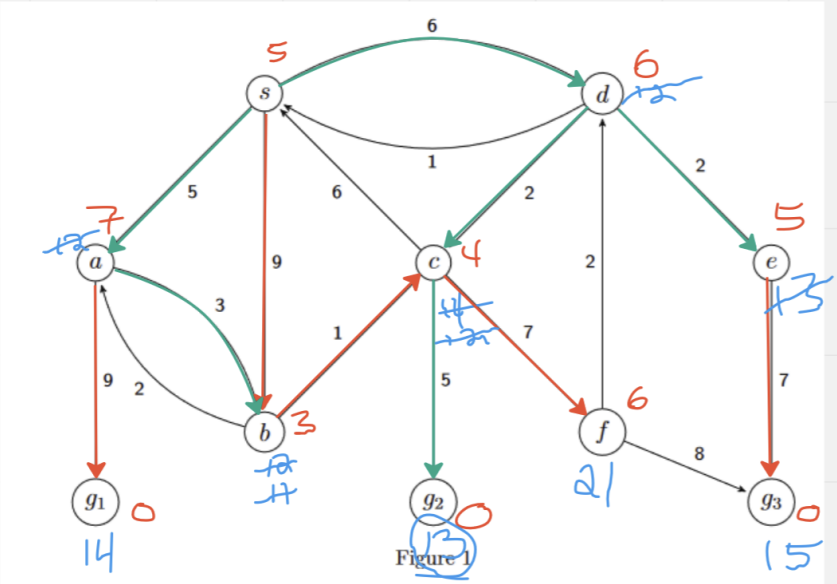

Figure 1

Priority Queue: [s, a, b, d, g1, c, e, g2, f, g3]

Visited: [s, a, b, d, c, e, g2]

g. Breadth-first search: 14
Depth-first search: 15
Uniform cost search: 14
Greedy best-first search using h1: 15
A/A* search using h1: 13