1. For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.

Every agent function is implementable by some program/machine combination.

An agent function maps perception histories to possible machine actions. Unfortunately, it is possible for both histories and actions to be infinite, making an injective mapping between them impossible to achieve by any program or machine.

Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.

True, if the task is to be a random number or choice generator. For example, the machine that decides what the slot machine result is must choose a random result each time. This satisfies the condition for rationality, being that the agent chooses the "right" choice, in this case a random choice, from the actions that it can perform.

It is possible for a given agent to be perfectly rational in two distinct task environments.

True, since the requirement for being rational is to maximize success. For example, a machine learning model might be able to both play chess and backgammon given two distinct training sets and task environments, simply because it has the possibility of learning what a successful move looks like based on its observations of the boards and set of possible actions.

Every agent is rational in an unobservable environment.

False, since a rational action requires a rationale. Without an observable environment, it is impossible for any rationale to occur, and each action would have equal probability of making the agent successful. It is important to note here that the lack of a clear choice does not make all choices therefore rational. For example, take a robot (with the objective of staying alive) standing at the edge of a cliff. The robot has the choice to either drive off (forward) or retreat (backwards). Since the robot cannot observe its environment, it cannot make the rational choice of driving backwards, which is the action that would cause it to be the most successful. It cannot strive to do the right thing without knowledge of what that might be.

2. For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed in Chapter 2 of the textbook.

playing a tennis match

performance measure: points scored, balls served/returned within court
environment: tennis court
actuators: Two jointed arms for holding racket, two jointed legs for moving around on court
sensors: Video camera for finding position of ball, opponent, and position of robot on court. Gyroscope for finding orientation of racket, force sensor for determining how hard ball was hit

practicing tennis against a wall

performance measure: Balls returned to wall, trajectory of returning ball from wall
environment: Half a tennis court with wall in place of net
actuators: Two jointed arms for holding racket, two jointed legs for moving around on court

sensors: Video camera for finding position of ball and position of robot on court. Gyroscope for finding orientation of racket, force sensor for determining how hard ball was hit

performing a high jump

performance measure: Height of jump, making it over the bar, landing safely
environment: High jump field, high jump pole
actuators: Humanoid body
sensors: Accelerometer/Gyroscope/positioning system to know when to jump and the orientation of the body mid flight. Stereoscopic camera for seeing the bar and measuring height off of the ground.

3. Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall.

a) Given a maze size of M x N, the state space is defined by the location of the object, the orientation of the robot (NSEW), the successor location update, and the goal is to reach the end. In this maze, the search state is 4(M*N - A), where A is the amount of space covered by the walls of the maze.

b) In this problem, we only need to turn at intersections, and as a result we no longer need to consider each position within the length of a corridor independently, and can consider each corridor and corner as units. In this case, we only need to know the amount of intersections, I, and the amount of passages, P. In each passage, the only action is to continue and the successor is updating which passage/corridor the robot might be in, with the goal remaining the same. Each search state contains 4 possible branches for each direction, with a total size of 4(I+P).

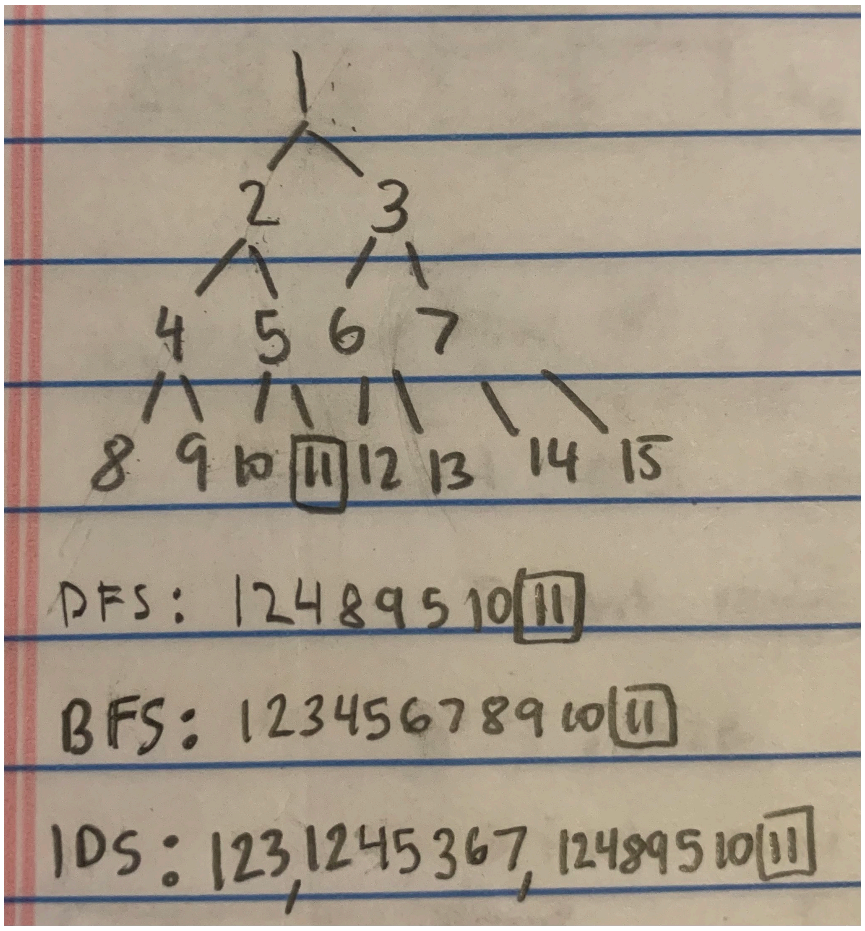
c) With the understanding that we only need to turn when hitting a corner, we can now cut down our search space to eliminate positions where the robot will be directly facing a wall. For passages, the only possible action is to continue moving through the tunnel. Since we know that in a 2D maze the solution can always be found by following the right hand wall, we can direct the robot to turn only when a right hand turn is available or when it hits a wall and therefore must turn left (or 180° in the case of a dead end). Therefore, we no longer need to track the orientation of the robot, and the search space becomes (I+P).

d) One simplification is that the robot can only face in four possible directions. Finding exact angles in practice can be troublesome.

Another simplification is that the robot will automatically stop when approaching a wall. In practice, this requires telemetry, sensors and brakes that we abstract away.

The fact we know the initial location of the robot is also a simplification. In the real world, we often are placed in an entirely unknown or new environment with no prior information or knowledge.

4. Consider a state space where the start state is number 1 and each state k has two successors: numbers 2k and 2k+1.



DFS: 1 2 4 8 9 5 10 $\boxed{11}$

BFS: 1 2 3 4 5 6 7 8 9 10 $\boxed{11}$

IDS: 1 2 3, 1 2 4 5 3 6 7, 1 2 4 8 9 5 10 $\boxed{11}$

3) Bidirectional search could work on this problem, since both the goal and origin are well defined and we can easily find the parent node (for node n, simply go to floor(n/2)). The branching factor from start to goal is 2 and from goal to start it is only 1. This might present a problem for bidirectional search, since the forward search (if using unbounded DFS) could take a very long time to meet the backwards search and therefore not serve useful. In fact, any search algorithm that does not immediately find the optimal route will be slower than the backwards algorithm, since there no decisions need to be made.

4) Since the tree is very structured, it would be much more efficient to search backwards, since the only node to be visited from node n is floor(n/2). Reformulating this problem to be a simple recursive calculation removes all search elements outside of finding the node in memory.

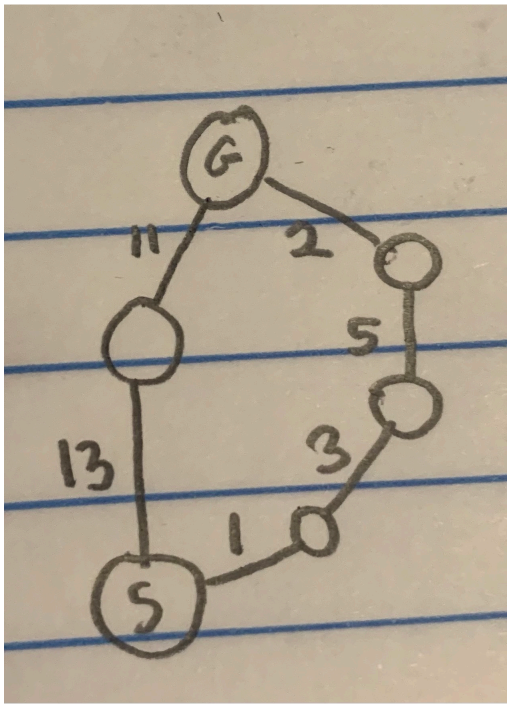
5)
```
1: Create empty array of length ceiling(log_2(G)), where G is the goal state.
    2: While G > 0,
            insert G into array
            G = floor(G/2)
    3: Reverse and return array
```
This array will contain the path from start node (1).

5) Show that breadth-first search given constant costs for all actions is guaranteed to find the shortest solution. Show that this is not true for the case of varying costs.

We must first note two important parts of Constant Cost Breadth First Search. First, the cost to get to a particular node is equivalent to the shortest path to such node. Second, each node with the same cost is visited in the same pass of BFS.

Assume that BFS found path A, but shortest path B exists with len(A) > len(B). The first nodes on each path are visited together, as are the second and third nodes on each path. We can continue this trend to the len(A)th node on each path, which will be the goal on path A and a node on path B. Since these nodes are visited at the same sweep of BFS, BFS would find path A. This is a contradiction to the initial assumption, and therefore BFS will always find the shortest path in a graph with constant costs.

Below is an example of BFS not working in the case of varying costs.



Here, BFS will find the depth 2 path to G, which has a much higher cost than the depth 4 path.