CSci 4511W: Intro To AI
Homework 1
[Colab Notebook Link]
Question on agents:

1. **Why does the textbook state "As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave"?**
   The textbook emphasizes the importance of focusing on what you want the agent to do, not focusing on how to do it. This is because specifying behavior can be too restrictive and might prevent the agent from finding more creative and efficient solutions. By focusing on the performance measure, the agent will be allowed to explore different strategies and adapt to the environment. In other words, you define success, and the agent will figure out how to achieve it.
2. **Why a rational an agent does not need to be omniscient? Explain briefly.**
   A rational agent does not need to be omniscient because it acts based on perceptions and the information it has available. It doesn't need to know everything about the environment to make good decisions. Being rational means making the best possible decision given the current knowledge even in uncertainty and updating its knowledge as new information becomes available. An omniscient agent would always make the best decision by knowing all possible future states, but that is not always the case in real-world situations and is unrealistic.
3. **Is the performance measure encoded in the agent function or in the agent program?**
   The performance measure is not encoded in the agent function or program. It is defined externally by the environment as it evaluates how well an agent performs in its environment. The agent program is responsible for processing inputs, making decisions, and taking actions. The agent function maps sequences to actions, but doesn't define how success is measured. Additionally, the performance measure is defined by the designer and determines how good or successful the agent's actions are for achieving its goal.

State space representation: You are given a number of blocks, all of the same size. The blocks can be on a table or on top of other blocks. You are allowed to pick up one block at a time if nothing is on top of it. A block can be moved either to the table or on top of another block.

1. **Describe a state-space representation for the problem, specifying the state representation, the initial state, goal condition, and actions.**
   - State representation: A state can be represented as a set of stacks, where each stack is a list of blocks in the order they are stacked. The table is the most bottom level in this particular stack.
     - **[[A, Table], [B, C]]** -> Means A is alone on the table, and B is on the table with C on top of it.

- Initial State: is the initial arrangement of blocks before any moves are made.
    - **[[A, B], [C, Table]]** -> A is on top of B, and C is on a separate stack.
- Goal State: a set of stacks of blocks representing the final desired arrangement of blocks.
    - If the goal is to have C on A, and A on B, and B on the table, the goal state would be: **[[C, A], [A, B], [B, Table]]**
- Actions: the possible actions an agent can take.
    - **Pick-up(X)**: if block X is clear with nothing on top, remove it from its current stack (or if it's on the table).
    - **Put-down(X)**: place block X alone on the table.
    - **Stack(X, Y)**: if block Y is clear, place X on top of Y.
    - **Unstack(X, Y)**: if X is on top of Y, remove X and place it separately.

2. **Is the state space finite? Is it a tree or a graph?**
   The state space is finite because the number of blocks is finite, and so there are only a limited number of ways to arrange them. It is a graph because a specific state can be achieved through multiple different series of actions. A tree structure would mean each state has only one parent/predecessor which is not the case in the block state space.

This question is on properties of search algorithms. Consider the following uninformed search algorithms from Chapter 3.3: depth-first, breadth-first, uniform cost, depth-limited, iterative deepening, and bidirectional search.

1. **Suppose you have a large and shallow state space, with many actions for each state. Which search algorithm(s) would you use if you are concerned about time complexity, and which if you are concerned about space complexity? Explain your reasoning.**
   For time complexity, breadth-first search is a good choice because it will explore the state space level by level and will find the solution relatively quickly in a shallow space. Uniform-cost search would also be a good fit.
   If concerned with space complexity, Depth-First Search or Iterative Deeping Search would be better applicable than Breadth-First or Uniform-Cost Search. Depth-First Search only stores the current path which makes it space-efficient. Iterative Deepening Search combines DFS's space efficiency as well as completeness.
2. **Suppose you have a deep and narrow state space with a finite number of states. Which search algorithm(s) would you use if you are concerned about time complexity, and which if you are concerned about space complexity? Explain your reasoning.**
   For time complexity, Iterative Deepening Search would be a good choice because it will explore a single path to the bottom quickly like Depth-First Search, but won't get stuck exploring a longer path if a shorter one exists which can occur using just Depth-First Search.

For space complexity, Depth-First Search is a good choice because it only needs to store the nodes along the current path which makes it very memory efficient.

Suppose you search a graph using uniform cost search (which is also called Dijkstra's algorithm, see Fig. 3.9 in the textbook), but do not keep track of the nodes that have been explored.

1. **If the graph is acyclic, will your algorithm find the cost optimal solution or not? why or why not? Will it take more time than the standard uniform cost algorithm to find the solution?**
   In an acyclic graph, the uniform cost search algorithm will find the cost-optimal solution even without keeping track of the visited/explored nodes. The algorithm expands the nodes in order of increasing path cost, so since there are no cycles, a cheaper path will never be discovered later. However, it will take more time than a standard uniform cost algorithm to find the solution because it may revisit nodes multiple times.
2. **If instead the state-space has cycles, will your answers be different? Why or why not?**
   In a cyclic graph, the uniform cost search algorithm may not find the cost-optimal solution without keeping track of the explored nodes. The algorithm would likely get stuck in a cycle, exploring the same nodes again and never finding the shortest path. If the algorithm doesn't get stuck, it might end up discovering a longer path before finding the optimal path. This will take much longer than the standard algorithm uniform cost algorithm since the standard algorithm avoids this issue by keeping track of explored nodes and never re-expanding them.