Aidan Rohm

Artificial Intelligence - CISC 6525

Professor Lyons

5 February, 2026

Homework Assignment 1

1. Definitions
    1. Agent
        1. An Agent is a software system that may operate on a separate entity that exists in/on its environment or world. An Agent acts on this entity by using actuators to conduct actions and sensors to conduct sensing.
    2. Rational Agent
        1. Being a specific type of intelligent agent, a Rational Agent consistently selects the action that will maximize its expected performance measure. This is based on the agent's percept history, built in knowledge, and defined goals.
    3. Task Environment
        1. A Task Environment contains all information about an environment. The performance measure, general environment, and the agent's sensors and actuators, are implemented with the PEAS (Performance, Environment, Actuators, Sensors) to fully depict the context surrounding a problem.
    4. Partially Observable Environment

    1. A Partially Observable Environment is a type of environment that does not completely reveal its state to the agent’s sensors. This can be due to noise, limited range, or hidden information.

5. Nondeterministic Environment
    1. A Nondeterministic Environment is one in which the same action can be taken in the same state and lead to multiple different outcomes. Nondeterminism can stem from a multitude of causes such as randomness.
6. Dynamic Environment
    1. A Dynamic Environment is one that is constantly changing with spontaneous, unpredictable changes. Often, a Dynamic Environment can be created by things such as unpredictable pedestrian movements on the side of a busy street.
7. Discrete Environment
    1. A Discrete Environment is characterized by having a distinct number of states, actions, and perceptions. In a Discrete Environment, decisions are made in clearly defined steps rather than in smooth transitions or continuously.
8. Simple Reflex Agent
    1. A Simple Reflex Agent operates with direct sensor inputs, without using memory or considering any past states. Often characterized by a “condition-action rule” or an “if-then” rule, which makes them best suited for fully observable environments.
9. Goal-Based Agent

        1. A Goal-Based Agent is one that acts in order to achieve specified long term objectives. This contrasts from an agent that might respond immediately to stimuli or information.

2. The Jug Problem
    1. 6 Items Necessary to Define the Search Problem
        1. The state space; usually represented as a set.
        2. The initial or starting state; one specific state.
        3. The goal state; can be a single state, or sometimes a set of acceptable states.
        4. Actions; are usually represented as a set
        5. Transition model; may be able to find the result of a specific action
        6. Action cost function; demonstrates the price-to-pay for committing a certain action.
    2. Proposed Representation of the Problem State; Size of State Space
        1. s = (x1, x2, x3), where $x1 \in \{0, 1, \dots, 12\}$, $x2 \in \{0, 1, \dots, 8\}$, and $x3 \in \{0, 1, 2, 3\}$. There are 13 different values for x1, 9 different values for x2, and 4 different values for x3. This means that there are 13 * 9 * 4 = 468 different states in the state space.
    3. Start State/End State in this Representation
        1. It is natural to think that there will be no water in any of the jugs at the beginning, meaning that the start state would be (0, 0, 0). The end state just needs to satisfy the fact that one jug must have exactly one gallon of water. This is satisfied by G(x1, x2, x3) = (x1 = 1) V (x2 = 1) V (x3 = 1.

4. Number of Different Actions in the Problem
   1. Each jug can be filled in one action per jug = 3 actions
   2. Each jug can be emptied into a drain = 3 actions
   3. Each jug can be emptied to another jugs capacity = 3 jugs, 2 per action = 6 actions
   4. As a total, this means there are 12 actions
5. Action and the Corresponding Transition Model
   1. Result((x1, x2, x2), Fillx1) = (12, x2, x3)
   2. Result((x1, x2, x3), Fillx2) = (x1, 8, x3)
   3. Result((x1, x2, x3), Fillx3) = (x1, x2, 3)
   4. Result((x1, x2, x3), Emptyx1) = (0, x2, x3)
   5. Result((x1, x2, x3), Emptyx2) = (x1, 0, x3)
   6. Result((x1, x2, x3), Emptyx3) = (x1, x2, 0)
   7. For pouring from one jug to another, let's define t = min(xi, My - xy) where My is the maximum capacity of jug xy.
      1. Pouring x1 into x2 yields
         - a. t = min(x1, 8 - x2)
         - b. Result((x1, x2, x3), Pourx1x2) = (x1 - t, x2 + t, x3)
      2. Pouring x1 into x3 yields
         - a. t = min(x1, 3 - x3)
         - b. Result((x1, x2, x3), Pourx1x3) = (x1 - t, x2, x3 + t)
      3. Pouring x2 into x1 yields
         - a. t = min(x2, 12 - x1)

            b. Result((x1, x2, x3), Pourx2x1) = (x1 + t, x2 - t, x3)

        4. Pouring x2 into x3 yields

            a. t = min(x2, 3 - x3)

            b. Result((x1, x2, x3), Pourx2x3) = (x1, x2 - t, x3 + t)

        5. Pouring x3 into x1 yields

            a. t = min(x3, 12 - x1)

            b. Result((x1, x2, x3), Pourx3x1) = (x1 + t, x2, x3 - t)

        6. Pouring x3 into x2 yields

            a. t = min(x3, 8 - x2)

            b. Result((x1, x2, x3), Pourx3x2) = (x1, x2 + t, x3 - t)

    6. Python Program is submitted in the Assignment Attachments

    7. Please find an analysis of my program vs the ChatGPT program towards the end of this document.

3. Questions related to uninformed search algorithms

    1. Constraints needed to use BFS

        1. The floor plan would have to be bound by a finite width and height in order for BFS to terminate. This step alone is essential. For convenience, it would also make sense to map the entire floor plan into an equivalent grid of integers. With any specific point in the grid being represented by some integer, we can then implement a uniform step cost for the search agent. This is realistic to a human's movements as we know stepping in any direction causes the same amount of cost. In order to allow the agent to keep optimal routes, it will be further important to make diagonal

movements also cost the same. If diagonal movements did not cost the same as horizontal and vertical movements, then the agent would never find the fastest/most optimal path.

   2. Please see following pages for the different trees drawn out
   3. A BFS tree displayed on the floor plan itself would essentially look like an expanding ring of locations being visited continuously. Since BFS searches the shallowest unexpanded/unsearched node first, it will expand from the starting state in rings until eventually reaching the goal. As for DPS, the node will expand straight in one direction first, before back tracking and continuing to the next direction. In a sense, DPS would display the search on the floor plan as if rays are shooting from the start point, differing from the rings of expansion from BFS.

4. Questions related to the vacuum cleaner and two rooms
   1. Refer to the following pages for the diagram.
   2. Refer to the following pages for the documentation of the number of states and levels.
   3. Figure 3.2 differs from the full tree diagram because it utilizes specific states more than once as opposed to drawing them out. The representation from 3.2 is pretty similar to a diagram of a Deterministic Finite Automata (DFA) diagram that has a start state, and resulting states depending on the input, leading to a goal state. When expanding a tree (particularly without constraints/rules) the same state may be created repeatedly without regard to whether or not it has been visited already. When building a tree without constraints, it is imperative to simply continue expanding nodes based on what actions are available. The tree

diagram is also a more accurate representation of what a program or agent would actually be “thinking” while the DFA representation is a bit more accurate for the reality of the actions from an outside perspective.

Analysis: My Program vs. Chat GPT

At first notice, the algorithm that Chat GPT creates and the one that I implement are logically the same "version" of Uniform Cost Search. Both of the programs execute a Uniform Cost Search that finds the same optimal goal. This can be observed as they both visit the same number of states (16 total), and take the same actions to reach the overall goal (3 actions).

Baked into both of the programs is a basic demonstration of the code's performance. The code that I wrote is certainly more "complete" in the sense that it is user friendly. Predefining the distinct actions for the user to see is a good way to document code, though it did lead to a performance hit. Based on the performance metrics, Chat GPT operates at nearly twice the speed simply because of its structure to its code. While my code is iterating over different actions and doing string comparisons for each one, Chat GPT's is doing simple integer arithmetic and array indexing, easily contributing to faster execution times.

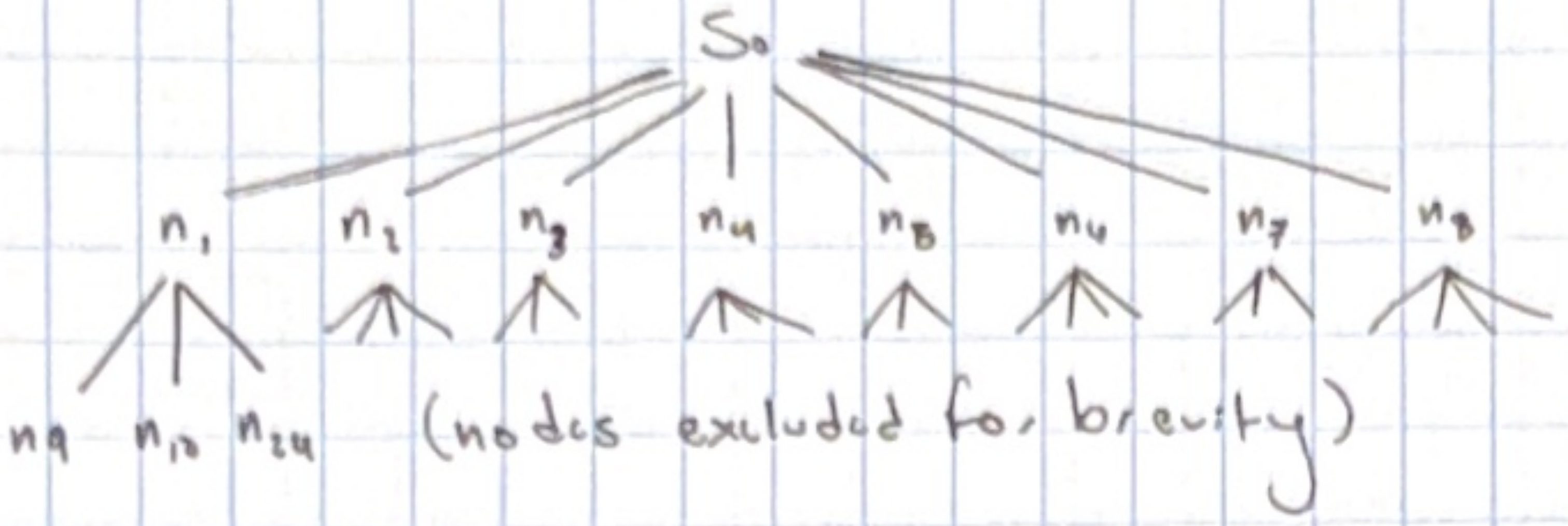
I did find that Chat GPT non-hardcoded creation of the different actions was favorable not just because of the speed, but also because of its ability to be reused. If you wanted to ask the same question with different boundaries, Chat GPT's setup allows you to do so by simply changing jug capacities and liquid goal amounts. I chose to hard code the actions as strings because that is what best matched my environment description from the previous parts of the question. Furthermore, I already knew them to be true. In the future, I will try to implement concepts with a more general approach.

When designing my code, I wanted to be sure that I was making a sound tool with concepts that could be utilized for future use cases. Admittedly, some of this was probably overdone. After seeing the comparatively brief implementation from Chat GPT with a more streamlined approach, it is clear to me that I literally made some costly implementation. Interestingly, Chat GPT did also avoid visiting the same state twice but did not make an effort to update the path/cost to a node if a shorter/less expensive cost is found. This concept was generally discussed in class, though refraining from revisiting previously visited states is imperative for Uniform Cost Search, so Chat GPT correctly implementing this was not surprising. What was surprising, was that it did not make an effort to update the cost to visit a node if it happened to find a less expensive route.

3b.) assume the environment is set up in the following way

⋱	⋮	⋮	⋮	⋮	⋮	⋰
…	n_{15}	n_{16}	n_{17}	n_{18}	n_{19}	…
…	n_{14}	n_4	n_5	n_6	n_{20}	…
…	n_{13}	n_3	$\boxed{S_0}$	n_7	n_{21}	…
…	n_{12}	n_2	n_1	n_8	n_{22}	…
…	n_{11}	n_{10}	n_9	n_{24}	n_{23}	…
⋰	⋮	⋮	⋮	⋮	⋮	⋱

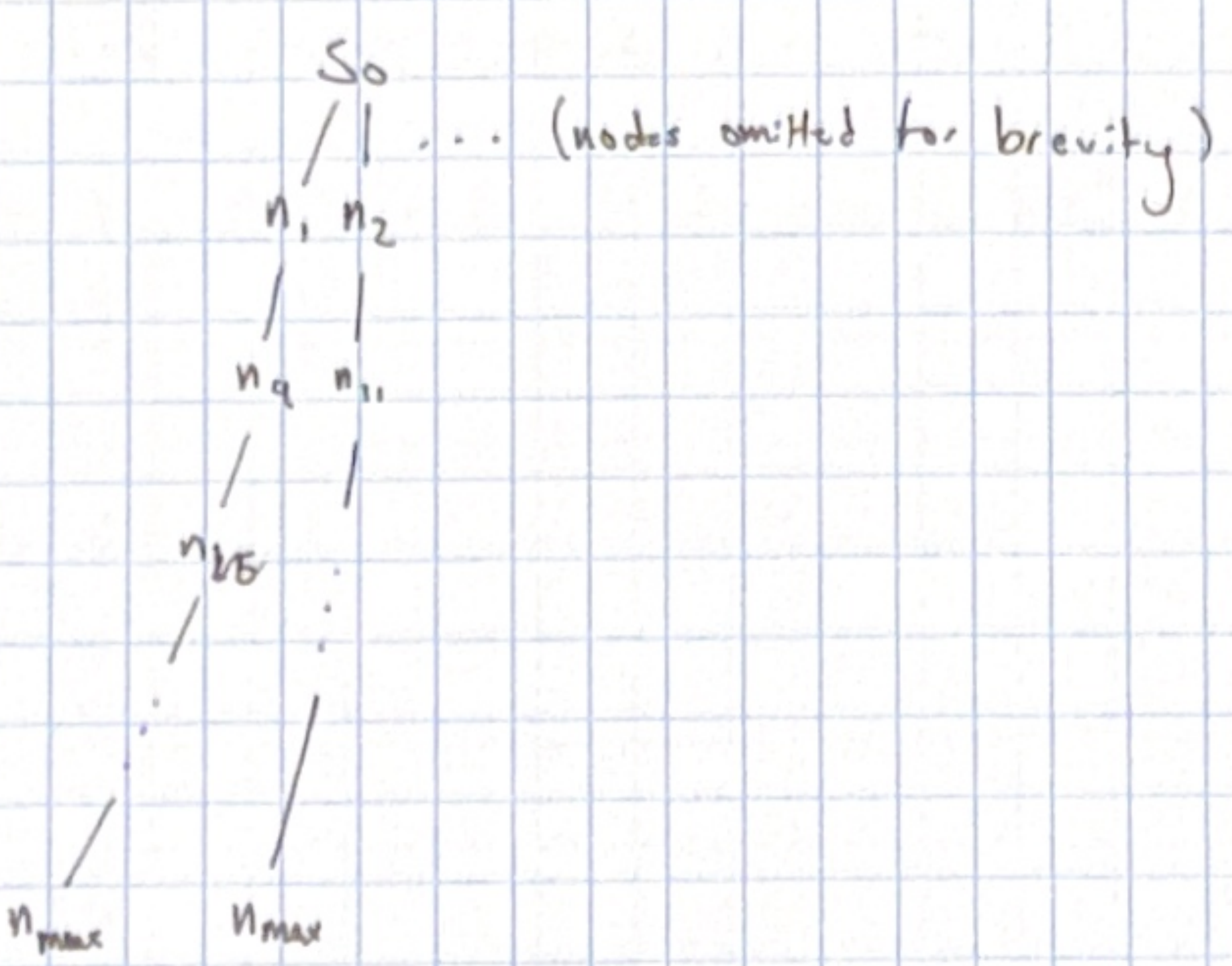
for BFS, the tree would expand in this manner



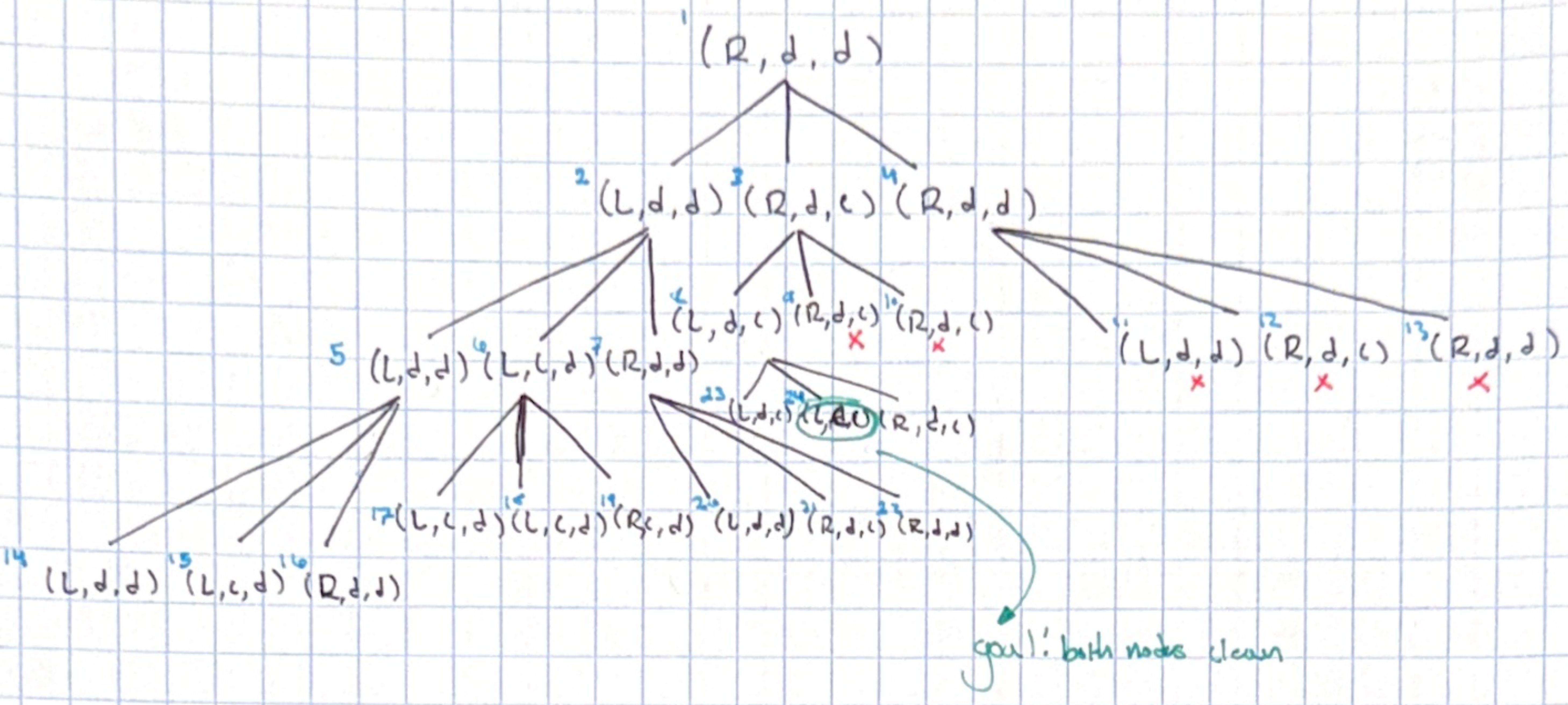
| all level 1 nodes/states first

| all level 2 nodes/states next

for DFS, the tree would expand in this manner



#4a.) Tree of the state space considering a starting state of (R, d, d)



#4b.) 5 nodes from the second level are left unexplored
the goal node exists in the 3rd level going on the path:
(R, d, d) ⟶ (R, d, c) ⟶ (L, d, c) ⟶ (L, c, c)
1 2 3rd level

counting the start node, there are a total of ~~24~~ 24 needed/necessary nodes/states to depict