02180 Intro to AI
Exercises for week 2, 12/2-19
SOLUTIONS

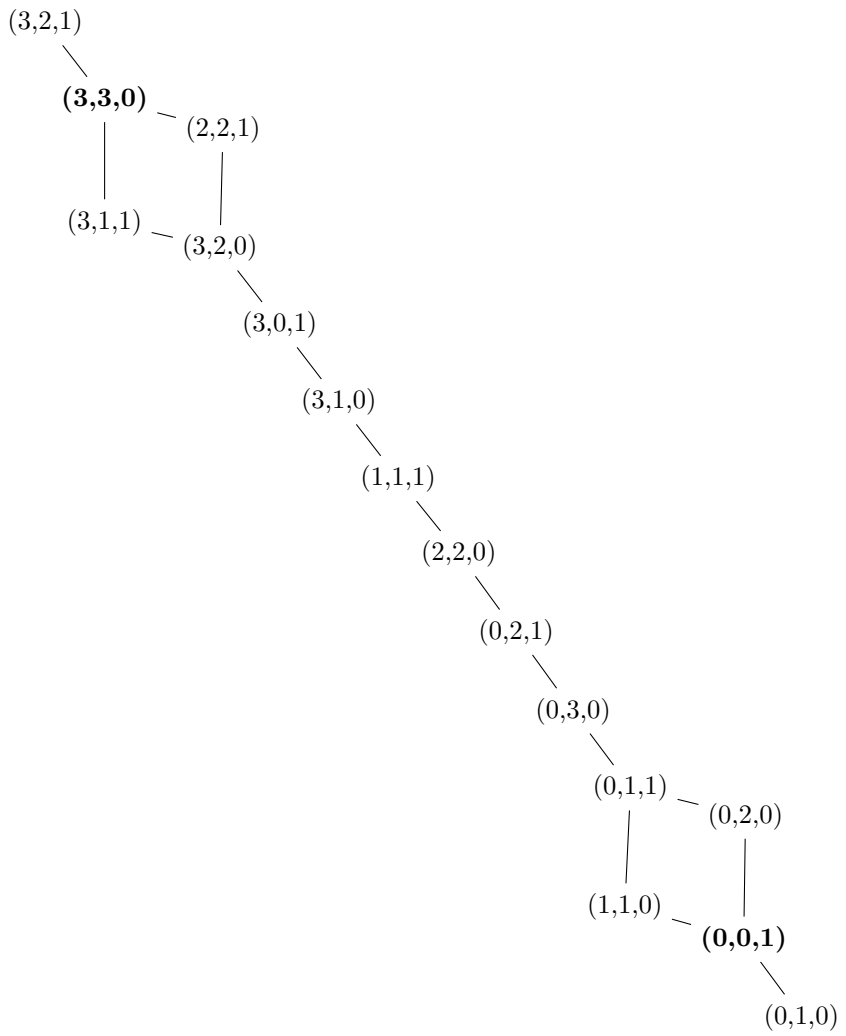
Exercise 1

The following exercise is adapted from Exercise 9 in Chapter 3 of the textbook (3rd ed, Global Edition). The *missionaries and cannibals problem* is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.[1] This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

a. How can the states of the problem be represented? What is then the initial state? Draw a diagram of the complete state space. What is the size of the state space (measured in number of states it contains)?

SOLUTION. In each state we only have to keep track of how many missionaries and cannibals are at each bank. Since the number of missionaries and cannibals at the right bank can be deduced from the number of missionaries and cannibals at the left, we only need to keep track of how many of each are at the left bank. The only additional thing we have to keep track of is whether the boat is at the left or right bank. Hence each state can be represented as a triple (m, c, b), where $m \in [0, 3]$ is the number of missionaries at the left bank, $n \in [0, 3]$ is the number of cannibals at the left bank, and $b = 0$ if the boat is at the left bank, otherwise $b = 1$. Hence an upper bound on the size of the reachable part of the state space is $4 \times 4 \times 2 = 32$. The initial state can be assumed to be $(3, 3, 0)$ (everything initially at the left bank). The goal state is then $(0, 0, 1)$. The available actions are for the boat to take 1–2 people from the current bank to the opposite bank. However, we also have to make sure that the number of cannibals at any bank never outnumber the missionaries. This leads to the following state space (edge labels left out):

[1] Whenever the boat goes from one side to the other, everybody has to go out before the boat can go back again. The boat can not sail without anybody in it.

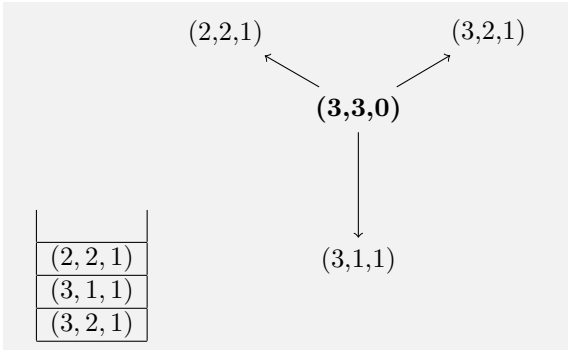


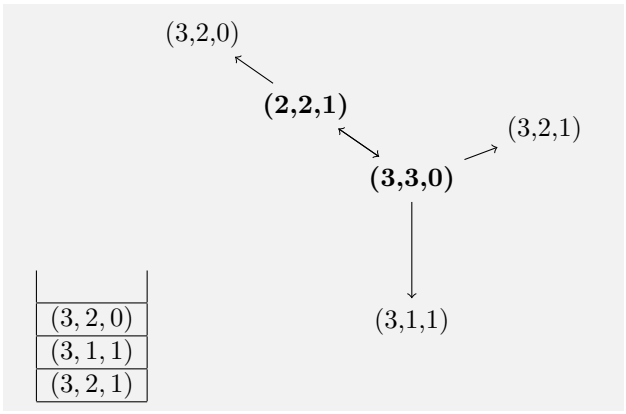
b. Solve the problem using the GRAPH-SEARCH algorithm. You can use the pseudocode from the book or the slides. Make sure to keep track of of your *frontier* and your *explored set*. Which *search strategy* are you using (in which order do you choose frontier nodes for expansion)? Does you search strategy guarantee that you find an optimal solution (a solution with fewest possible actions)?

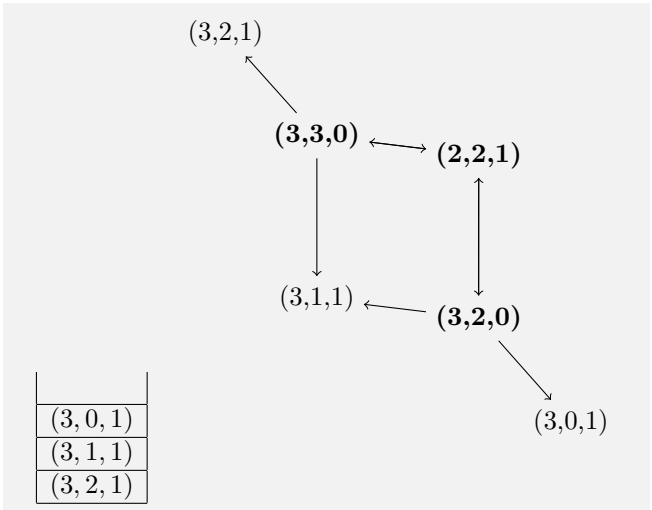
SOLUTION. GRAPH-SEARCH automatically checks for repeated states, which is a good idea in this case, since a DFS tree search might get stuck in the loop $(3,3,0) \rightarrow (2,2,1) \rightarrow (3,2,0) \rightarrow (3,1,1) \rightarrow (3,3,0)$. BFS tree search does not risk entering an infinite loop, but the search tree will grow exponentially with the depth, because the initial state will keep being re-expanded at every alternate level of the tree, and it has an outdegree of two (two outgoing edges).

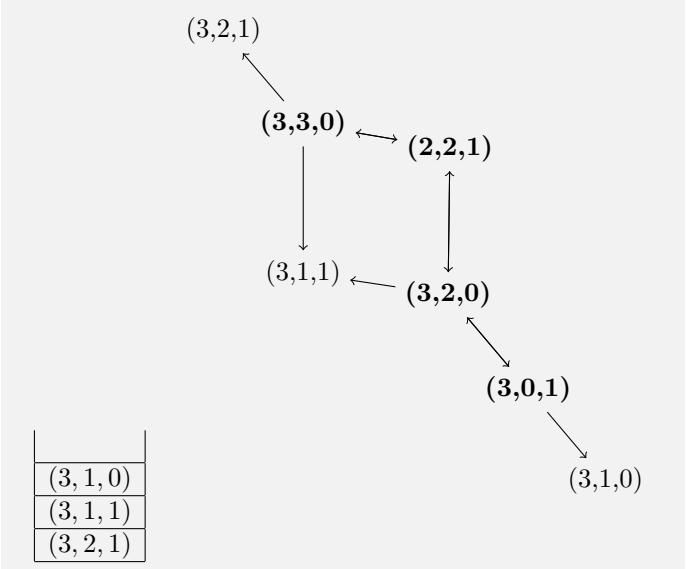
GRAPH-SEARCH can be used with different search strategies. We will here choose DFS. This means that the frontier is a stack. Below we show the first 4 steps of the algorithm,

where the boldface nodes are the explored ones, and the frontier is visualised as a stack to the left:









c. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

SOLUTION. The are several reasons. One of them is that it is a bit tricky to keep track of which moves are legal. Another reason is that the solution is relatively long, 11 steps. Long solutions are not always a problem to find for humans, but it is only easy for us when we have a good heuristics to guide our search. For instance, finding a walking route from A to B on a map is usually not difficult for us, even if there are more than 11 intersections where we have to make a choice. Usually, it is sufficient in any intersection to choose the next road segment to be the one that ends up in a new intersection with minimal straight line distance to B (technically speaking, we would be doing a greedy best-first search where the heuristics is the straight line distance to B). However, for the missionary and cannibals problem, there is no simple heuristics to guide the search. Indeed, the obvious heuristics which counts the number of people not yet at the right bank is not particularly efficient, as in every second move, we have to move further away from the solution according to this heuristics (we need to move people back from the right to the left bank). Finally, for humans it is not so easy to keep track of which states have already been visited, so we easily end up in loops (a greedy best-first search with the mentioned heuristics would end up in an infinite loop unless it keeps track of the already visited states).

d. Assume instead that there are n missionaries and n cannibals. What is then the size of the state space? You can report the size of the full state space, including the states that are not reachable without violating the constraint of cannibals not outnumbering missionaries. You can measure the size of the state space in number of distinct states. Does the size of the state space imply that the problem is difficult for a computer to solve for big values of n?

SOLUTION. With n cannibals and n missionaries, the state can now be represented as an element of $[0, n] \times [0, n] \times \{0, 1\}$. There are $2(n+1)^2$ such triples, which is hence the size of the (full) state space. As there is hence still only a polynomial number of states in n, it should not be too hard for a computer to solve it even for large values of n. Actually, it has

been proven that there is no solution to the problem with $n > 3$, but a computer searching for a solution might of course not realise that until having explored the full reachable state space.

e. Assume that in addition to the n missionaries, n cannibals there are now also n boats. Does this imply that the problem is difficult for a computer to solve for big values of n?

SOLUTION. Now states can be represented as elements $(m, c, b) \in [0, n] \times [0, n] \times [0, n]$, where m is the number of missionaries at the left bank, c is the number of cannibals at the left bank, and b is the number of boats at the left bank. The size of the state space becomes $(n + 1)^3$, but this is still polynomial in n and should not be too hard for a computer.

f. Assume that in addition to the n missionaries, n cannibals and n boats there are now also n banks instead of only 2. This means that any of the persons and boats can be at any of n banks. Does this imply that the problem is difficult for a computer to solve for big values of n?

SOLUTION. Take the case $n = 5$. We can represent the missionaries as a list of dots:

We can then place $n - 1$ vertical bars to represent how they are distributed into the n banks. For instance

means that there is 1 missionary on the first bank, 2 on the second, none on the third, 2 on the fourth, and none on the fifth. The number of distributions of missionaries is equal to the number of ways we can position the $n - 1$ vertical bars. Each bar can be positioned in $n + 1$ different ways. This gives in total $(n + 1)^{(n-1)}$ different possibilities. But this is assuming that the bars are distinguishable, which they are not, so we have to divide by $n!$. So the number of possible distributions of missionaries is

$$\frac{(n+1)^{(n-1)}}{n!}.$$

We have the same number of possible distribution of cannibals and boats. This expression grows exponentially in n which can be seen by the following calculations:

$$\frac{(n+1)^{(n-1)}}{n!} \geq \frac{1}{n}\Pi_{i=1}^{n-1}\frac{n+1}{i} \geq \frac{1}{n}\Pi_{i=1}^{n-1}\frac{n}{i} \geq \Pi_{i=2}^{n-1}\frac{n}{i} = \Pi_{i=2}^{\lfloor n/2 \rfloor}\frac{n}{i} + \Pi_{i=\lfloor n/2 \rfloor+1}^{n-1}\frac{n}{i}$$
$$\geq \Pi_{i=2}^{\lfloor n/2 \rfloor}\frac{n}{i} \geq \Pi_{i=2}^{\lfloor n/2 \rfloor}\frac{n}{(n/2)} \geq \Pi_{i=2}^{\lfloor n/2 \rfloor}2 \geq 2^{\lfloor n/2 \rfloor - 1} \geq 2^{(n/2)-2} = \frac{1}{4}(\sqrt{2})^n.$$

This means that in principle it might be very difficult for a computer to solve the problem for big values of n, *unless* it employs an efficient heuristics to avoid having to explore the entire state space. Having more banks available than in the version considered in question (e) ought to make the problem *easier* to solve, since it is easier to satisfy the constraint of never having the cannibals outnumber the missionaries at any bank. In fact, there is a trivial solution in this case: use half of the boats to bring all the cannibals to the goal bank, then use the rest to bring all the missionaries over there. But even though the problem should be easier in principle (and easier in practice for a human), it still might create more problems for a (naively implemented) computer program, since there are many more available actions to consider. For instance, a breadth-first search will certainly run out of time and memory for large values of n: it will explore an exponentially large state space (in n) before finding a solution.

Exercise 2

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. When thinking about the size of the state space etc. it can be an advantage to look at concrete mazes. Consider for instance the maze and video shown here:

http://ing.dk/video/video-robotmus-overvinder-labyrint-pa-3921-sekunder-124487

a. How can the states of this problem be represented? How large is then the state space, measured as a function of the number of locations in the maze?

SOLUTION. The states of the problem can be represented as triples (x, y, d) consisting of the x- and y-coordinates of the agent as well as its direction d. The size of the state space is the number of possible triples (x, y, d), which is 4 times the number of locations in the maze.

b. Define ACTIONS and RESULTS.

SOLUTION. The actions can be taken to be $\mathit{Turn}(d)$, where $d = W, E, N, S$ and $\mathit{Move}(l)$, where l is a natural number (the length/distance). All actions are applicable in any state, so

$$\textsc{Actions}(s) = \{\mathit{Turn}(d) \mid d \in W, E, N, S\} \cup \{\mathit{Move}(l) \mid l > 0\}$$

The transition model is:

$$\textsc{Result}((x, y, d), \mathit{Turn}(d')) = (x, y, d')$$
$$\textsc{Result}((x, y, d), \mathit{Move}(l)) = (x + x', y + y', d),$$

where if $d = E$ then x' is the maximal number $\leq l$ such that there is no wall in any of the locations $(x + 1, y), (x + 2, y), \ldots, (x + x', y)$. Similarly for $d = W, N, E$.

c. In navigating a maze, the only places we need to turn is at dead ends or intersections of two or more corridors (why?). Hence, from each point of the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using this observation, that is, reformulate ACTIONS and RESULTS. How large is the state space now?

SOLUTION. The idea is to remove the *Turn* action, and only have actions $\mathit{Move}(d)$ where $d = W, E, N, S$. The transition model is:

$$\textsc{Result}((x, y, d), \mathit{Move}(d')) = (x + x', y + y', d'),$$

where if $d' = E$ then x' is the maximal number such that $(x+1, y), (x+2, y), \ldots, (x+x'-1, y)$ are not goal locations, not walls and not intersections, and $(x+x', y)$ is not a wall. Note that the parameter d in the argument of RESULT is not used in defining the value of RESULT. This means that we can omit d from the state description and simply describe states as pairs (x, y). The transition model then becomes:

$$\textsc{Result}((x, y), \mathit{Move}(d')) = (x + x', y + y'),$$

where x' and y' are defined in terms of d' as above. The size of the state space is clearly now only the number of locations. We are abstracting away from the direction the robot is facing. In fact, the state space is only the number of intersections and dead ends plus possible the initial state and the goal (that might neither be intersections nor dead ends).

d. In our initial description of the problem we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

SOLUTION. (1) The robot can only be in a discrete location, not between locations (compare with the robot mouse video). (2) The robot can only face one of 4 discrete directions, N, S, E and W. (3) The robot is assumed to be able to move precisely a given number of cells. In real life, the robot would probably be a bit imprecise and would need sensors to tell whether a wall has been hit (even if it is assumed to have a full internal representation of the maze).

e. Can you say something about the search strategy that the mouse in the video appears to be using? You are not expected to be able to give a very precise answer to this question.

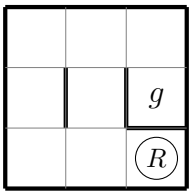
SOLUTION. It appears to be a kind of depth-first (online) search. It could be a greedy-best first search with some heuristics guiding it to the center of the maze, but this is not quite clear. Not also that the mouse does not initially know the maze, so it is really a search problem under *partial observability*, which is different from what we have considered so far.

f. Is finding your way through a maze like this a difficult problem in AI?

SOLUTION. No, in general it is a very simple problem, since it is a rather basic search problem. It can still be a challenge in very big and complex mazes, but computers would generally be much better than humans at solving mazes. They can search a bigger space much more quickly, and there is no deep intuition that a human can use to solve mazes more efficiently than using a simple search algorithm as the computer does (unlike the situation in e.g. chess or Go or natural language understanding).

Exercise 3

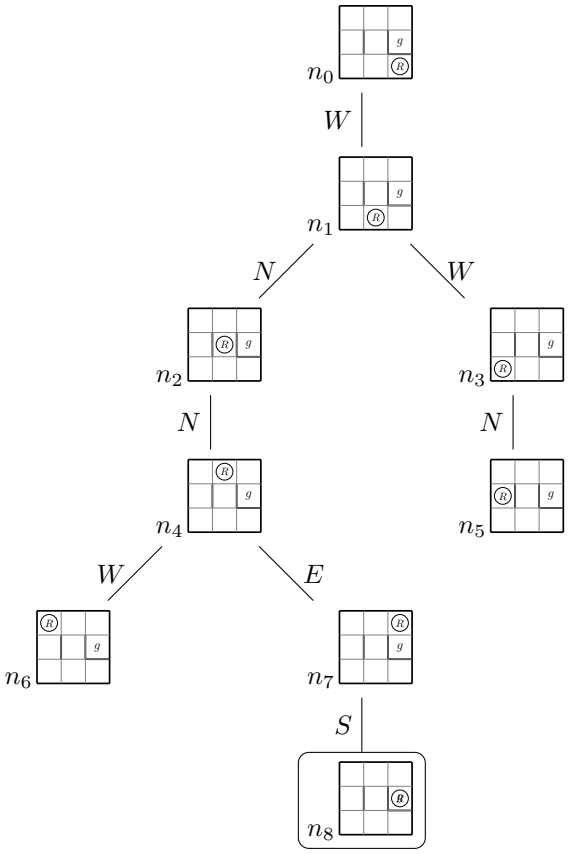
In the previous exercise, we considered a robot navigating a maze. Consider the following slight variation of the problem. The robot, R, is characterised by its position only, it does not have a direction. In each state, the robot can move one cell north (N), west (W), south (S) or east (E) (unless, of course, blocked by a wall in that direction). The goal of the robot is to reach the cell marked with a g. Consider the following instance of the problem:



You should assume that the robot—and the algorithms considered below—always explore the possible moves in the following order: N, W, S, E.

a. Illustrate how BFS graph search would solve this instance of the problem. You should: 1) draw the graph generated by BFS; 2) put numbers on the nodes of the graph according to the order in which they are generated, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the FIFO queue used for the frontier.

SOLUTION. Following the pseudocode from R&N figure 3.11, we get the search tree:



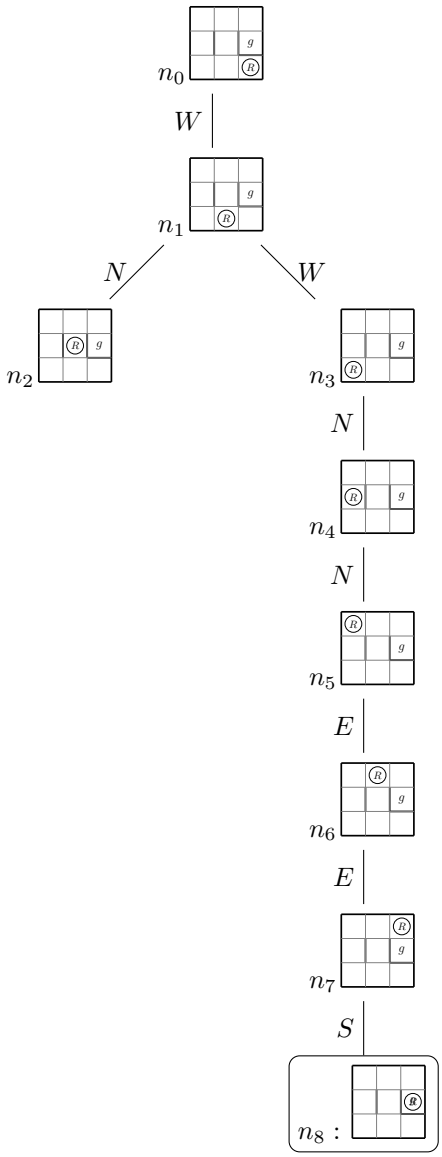
The frontier content is shown as it looks in the beginning of each loop iteration:

Iteration	**Frontier**
1	$[n_0]$
2	$[n_1]$
3	$[n_2, n_3]$
4	$[n_3, n_4]$
5	$[n_4, n_5]$
6	$[n_5, n_6, n_7]$
7	$[n_6, n_7]$
8	$[n_7]$

Because the BFS algorithm checks for goal states before adding states to the frontier, we will never see n_8 in the frontier and the algorithm terminates in the 8th iteration. The solution found is $WNNES$ of length 5. It is optimal (since we're using BFS).

b. Illustrate how DFS graph search would solve this instance of the problem. You should: 1) draw the graph generated by DFS; 2) put numbers on the nodes of the graph according to the order in which they are visited, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the LIFO queue (stack) used for the frontier.

SOLUTION. Following the pseudocode in R&N figure 3.7 for Graph-Search with a stack for the frontier, we get the search tree:



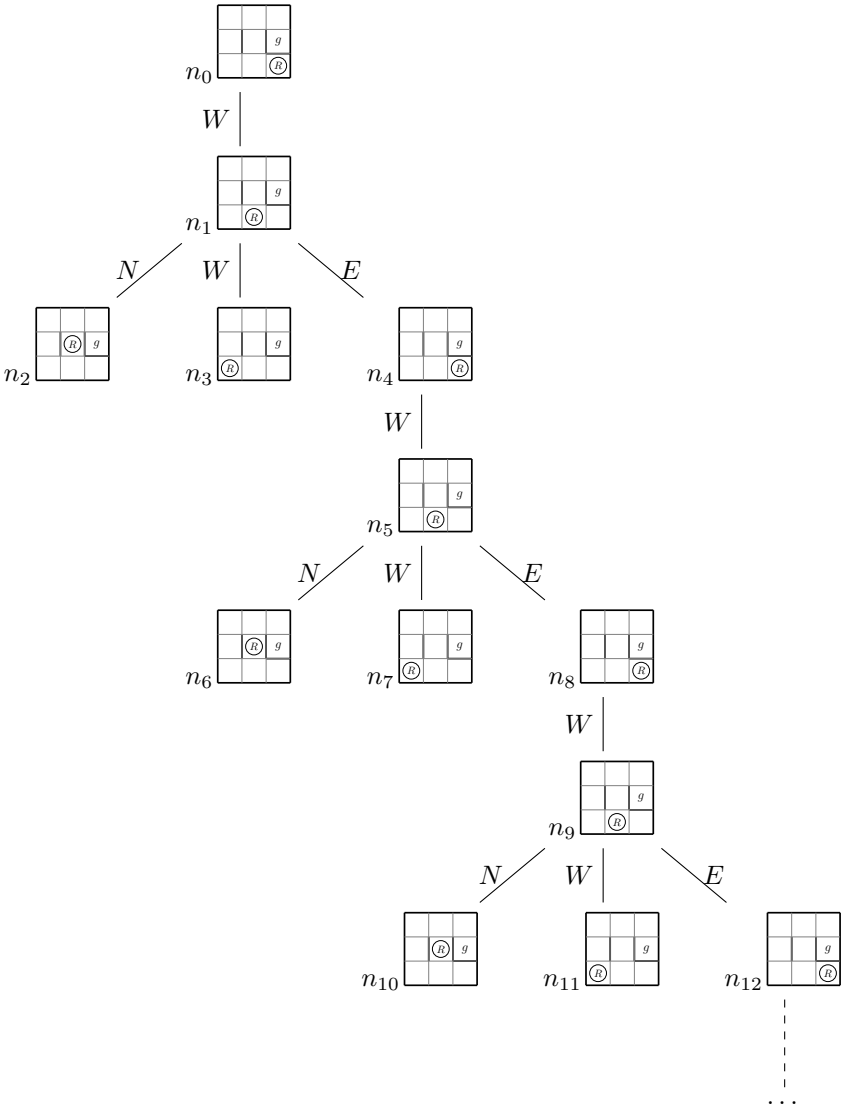
The frontier content is shown as it looks in the beginning of each loop iteration:

Iteration	Frontier
1	$[n_0]$
2	$[n_1]$
3	$[n_2, n_3]$
4	$[n_2, n_4]$
5	$[n_2, n_5]$
6	$[n_2, n_6]$
7	$[n_2, n_7]$
8	$[n_2, n_8]$

This time we see the goal state n_8 in the frontier, and terminate when we pop it in the 8th iteration. The solution found is $WWNNEES$ of length 7. It is not optimal.

c. Illustrate how DFS tree search would solve this instance of the problem. You should: 1) draw the tree generated by DFS; 2) put numbers on the nodes of the tree according to the order in which they are visited, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the LIFO queue (stack) used for the frontier.

SOLUTION. Following the pseudocode in R&N figure 3.7 for Tree-Search with a stack for the frontier, we get the search tree:



The frontier content is shown as it looks in the beginning of each loop iteration:

Iteration	**Frontier**
1	$[n_0]$
2	$[n_1]$
3	$[n_2, n_3, n_4]$
4	$[n_2, n_3, n_5]$
5	$[n_2, n_3, n_6, n_7, n_8]$
6	$[n_2, n_3, n_6, n_7, n_9]$
7	$[n_2, n_3, n_6, n_7, n_{10}, n_{11}, n_{12}]$
...	...

The search is stuck in a loop of exploring the same sequence of non-goal states over and over, and will never terminate.

d. *OPTIONAL.* Do the same as in (c) for iterative deepening DFS. This time you don't have to show each single node explored and each single configuration of the LIFO queue, but you should try to at least informally go through each step of the algorithm.

e. Could a different order of exploring the moves change the solution found by DFS? And the solution found by BFS?

SOLUTION. There is only one optimal solution, so any order of exploring possible moves will result in BFS finding the same (optimal) solution. Different orders of possible moves can lead DFS to find different solutions, e.g. the order W, S, E, N will result in DFS finding the optimal solution with both Graph-Search and Tree-Search. In general, there may not be any single fixed order leading DFS to find optimal solutions (or any solutions at all in the case of the Tree-Search variant).

f. Provide a table with an overview of the performance of the algorithms considered above in terms of: 1) number of nodes generated; 2) cost of the solution found (number of moves to get to the goal).

SOLUTION. The performance of each algorithm is as follows:

Algorithm	**Nodes generated**	**Solution length**
BFS	9	5
Graph-Search DFS	9	7
Tree-Search DFS	∞	N/A