3.15

![Figure S3.1](Image)

The state space for the problem defined in Ex. 3.15.

a. See Figure S3.1.
b. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
   Depth-limited: 1 2 4 5 9 5 10 11
   Iterative deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11
c. Bidirectional search is very useful, because the only successor of n in the reverse direction is [(n/2)]. This helps focus the search. The branching factor is 2 in the forward direction; 1 in the reverse direction.
d. Yes; start at the goal, and apply the single reverse successor action until you reach 1.
e. The solution can be read off the binary numeral for the goal number. Write the goal number in binary. Since we can only reach positive integers, this binary expansion beings with a 1. From most- to least- significant bit, skipping the initial 1, go Left to the node 2n if this bit is 0 and go Right to node 2n + 1 if it is 1. For example, suppose the goal is 11, which is 1011 in binary. The solution is therefore Left, Right, Right.

3.27

a. $n^{2n}$. There are $n$ vehicles in $n^2$ locations, so roughly (ignoring the one-per-square constraint) $(n^2)^n = n^{2n}$ states.
b. $5^n$.
c. Manhattan distance, i.e., $|n - i + 1 - x_i| + |n - y_i|$. This is exact for a lone vehicle.
d. Only (iii): $\min \{h_1, \ldots, h_n\}$. The explanation is nontrivial as it requires two observations. First, let the work $W$ in a given solution be the total distance moved by all vehicles over their joint trajectories; that is, for each vehicle, add the lengths of all the steps taken. We have $W = \sum_i h_i \geq n \cdot \min \{h_1, \ldots, h_n\}$. Second, the total work we can get done per step is $\leq n$. (Note that for every car that jumps 2, another car has to stay put (move 0), so the total work per step is bounded by $n$.) Hence, completing all the work requires at least $n \cdot \min \{h_1, \ldots, h_n\}/n = \min \{h_1, \ldots, h_n\}$ steps.
**Figure S5.2** A simple game tree showing that setting a trap for MIN by playing $a_i$ is a win if MIN fails for it, but may also be disastrous. The minimax move is of course $a_2$, with value $-5$.

**Figure S5.3** The game tree for the four-square game in Exercise 5.8. Terminal states are in single boxes, loop states in double boxes. Each state is annotated with its minimax value in a circle.

5.8

a. (5) The game tree, complete with annotations of all minimax values, is shown in Figure S5.3.

b. (5) The "?" values are handled by assuming that an agent with a choice between winning the game and entering a "?" state will always choose the win. That is, min(-1,?) is -1 and max(+1,?) is +1. If all successors are "?", the backed-up value is "?".

c. (5) Standard minimax is depth-first and would go into an infinite loop. It can be fixed
by comparing the current state against the stack; and if the state is repeated, then return a “?” value. Propagation of “?” values is handled as above. Although it works in this case, it does not always work because it is not clear how to compare “?” with a drawn position; nor is it clear how to handle the comparison when there are wins of different degrees (as in backgammon). Finally, in games with chance nodes, it is unclear how to compute the average of a number and a “?”. Note that it is not correct to treat repeated states automatically as drawn positions; in this example, both (1,4) and (2,4) repeat in the tree but they are won positions.

What is really happening is that each state has a well-defined but initially unknown value. These unknown values are related by the minimax equation at the bottom of 164. If the game tree is acyclic, then the minimax algorithm solves these equations by propagating from the leaves. If the game tree has cycles, then a dynamic programming method must be used, as explained in Chapter 17. (Exercise 17.7 studies this problem in particular.) These algorithms can determine whether each node has a well-determined value (as in this example) or is really an infinite loop in that both players prefer to stay in the loop (or have no choice). In such a case, the rules of the game will need to define the value (otherwise the game will never end). In chess, for example, a state that occurs 3 times (and hence is assumed to be desirable for both players) is a draw.

d. This question is a little tricky. One approach is a proof by induction on the size of the game. Clearly, the base case $n = 3$ is a loss for A and the base case $n = 4$ is a win for A. For any $n > 4$, the initial moves are the same: A and B both move one step towards each other. Now, we can see that they are engaged in a subgame of size $n - 2$ on the squares $[2, \ldots, n - 1]$, except that there is an extra choice of moves on squares 2 and $n - 1$. Ignoring this for a moment, it is clear that if the “$n - 2$” is won for A, then A gets to the square $n - 1$ before B gets to square 2 (by the definition of winning) and therefore gets to $n$ before B gets to 1, hence the “$n$” game is won for A. By the same line of reasoning, if “$n - 2$” is won for B then “$n$” is won for B. Now, the presence of the extra moves complicates the issue, but not too much. First, the player who is slated to win the subgame $[2, \ldots, n - 1]$ never moves back to his home square. If the player slated to lose the subgame does so, then it is easy to show that he is bound to lose the game itself—the other player simply moves forward and a subgame of size $n - 2k$ is played one step closer to the loser’s home square.

6.7 The “Zebra Puzzle” can be represented as a CSP by introducing a variable for each color, pet, drink, country, and cigarette brand (a total of 25 variables). The value of each variable is a number from 1 to 5 indicating the house number. This is a good representation because it easy to represent all the constraints given in the problem definition this way. (We have done so in the Python implementation of the code, and at some point we may reimplement this in the other languages.) Besides ease of expressing a problem, the other reason to choose a representation is the efficiency of finding a solution. Here we have mixed results—on some runs, min-conflicts local search finds a solution for this problem in seconds, while on other runs it fails to find a solution after minutes.

Another representation is to have five variables for each house, one with the domain of colors, one with pets, and so on.