Sheet 5 Solution

3.15

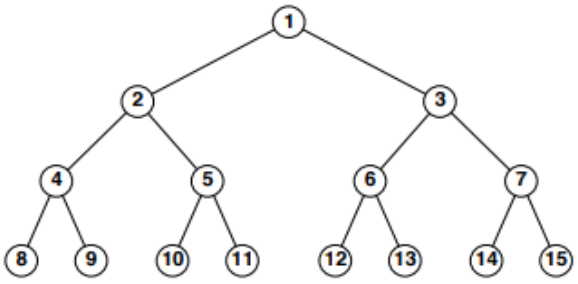

Figure S3.1 The state space for the problem defined in Ex. 3.15.

a. See Figure S3.1.

b. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 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 $\lfloor (n/2) \rfloor$. 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.

3.16

a. **Initial state**: one arbitrarily selected piece (say a straight piece).
Successor function: for any open peg, add any piece type from remaining types. (You can add to open holes as well, but that isn't necessary as all complete tracks can be made by adding to pegs.) For a curved piece, add *in either orientation*; for a fork, add *in either orientation* and (if there are two holes) connecting *at either hole*. It's a good idea to disallow any overlapping configuration, as this terminates hopeless configurations early. (Note: there is no need to consider open holes, because in any solution these will be filled by pieces added to open pegs.)
Goal test: all pieces used in a single connected track, no open pegs or holes, no overlapping tracks.
Step cost: one per piece (actually, doesn't really matter).

b. All solutions are at the same depth, so depth-first search would be appropriate. (One could also use depth-limited search with limit $n - 1$, but strictly speaking it's not necessary to do the work of checking the limit because states at depth $n - 1$ have no successors.) The space is very large, so uniform-cost and breadth-first would fail, and iterative deepening simply does unnecessary extra work. There are many repeated states, so it might be good to use a closed list.

c. A solution has no open pegs or holes, so every peg is in a hole, so there must be equal numbers of pegs and holes. Removing a fork violates this property. There are two other "proofs" that are acceptable: 1) a similar argument to the effect that there must be an even number of "ends"; 2) each fork creates two tracks, and only a fork can rejoin those tracks into one, so if a fork is missing it won't work. The argument using pegs and holes is actually more general, because it also applies to the case of a three-way fork that has one hole and three pegs or one peg and three holes. The "ends" argument fails here, as does the fork/rejoin argument (which is a bit handwavy anyway).

d. The maximum possible number of open pegs is 3 (starts at 1, adding a two-peg fork increases it by one). Pretending each piece is unique, any piece can be added to a peg, giving at most $12 + (2 \cdot 16) + (2 \cdot 2) + (2 \cdot 2 \cdot 2) = 56$ choices per peg. The total depth is 32 (there are 32 pieces), so an upper bound is $168^{32}/(12! \cdot 16! \cdot 2! \cdot 2!)$ where the factorials deal with permutations of identical pieces. One could do a more refined analysis to handle the fact that the branching factor shrinks as we go down the tree, but it is not pretty.

3.17 **a.** The algorithm expands nodes in order of increasing path cost; therefore the first goal it encounters will be the goal with the cheapest cost.

b. It will be the same as iterative deepening, d iterations, in which $O(b^d)$ nodes are generated.

c. d/ϵ

d. Implementation not shown.