3.7
a) Initial state: No regions colored.
   Goal test: All regions colored, and no two adjacent regions have the same color.
   Successor function: Assign a color to a region.
   Cost function: Number of assignments.
b) Initial state: As described in the text.
   Goal test: Monkey has bananas.
   Successor function: Hop on / off crate; Push crate for one spot to another (possibly on a grid); walk from one spot to another; stack / unstuck crates; reach for bananas.
   Cost function: Number of actions.
c) Initial state: Considering all input records.
   Goal test: Considering a single record which give an “illegal input” message.
   Successor function: Run again on the first half of records and on second half.
   Continue only with one half that gives the error message.
   Cost function: Number of runs.
d) Initial state: Jugs are empty – [0,0,0].
   Goal test: Any of – [1,y,z], [x,1,z] or [x,y,1].
   Successor function: Given values [x,y,z], generate [12,y,z], [x,8,z], [x,y,3] (by filling), [0,y,z], [x,0,z], [x,y,0] (by emptying), or for any two jugs with values u and v, pour u into v. This will result in the jug with v having min(u+v; capacity of jug) and the jug with u with losing the amount gained by the first jug.
   Cost function: Number of actions.

3.12
If there are two paths from the start node to a given node, discarding the more expensive one cannot eliminate any optimal solution. Uniform-cost search (UCS) and breadth-first search (BFS) with constant step costs both expand paths in order of g-cost. Therefore, if the current node has been expanded previously, the current path to it must be more expensive than the previously found path and it is correct to discard it (using graph-search).

For increasing-depth search (IDS), it is easy to find an example where the algorithm returns a suboptimal solution: have two paths to the goal, one with one step costing 3 and one with two steps that cost 1 each.

3.17
a) Any path, no matter how bad it appears, might lead to an arbitrarily large reward (negative-cost). Therefore, one would have to exhaust all possible paths to be sure of finding the best one.
b) Suppose the greatest possible reward is \( c \). Then if we also know the maximum depth of the state space (when the state space is a tree), then any path with \( d \) levels remaining can be improved by at most \( cd \), so any paths worse than \( cd \) less than the
best path can be pruned. For state spaces with loops, this guarantee doesn’t help, because it is possible to go around a loop any number of times, picking up \( c \) reward each time.

c) The agent should plan to go around this loop forever (unless it can find another loop with even better reward).

d) The value of the scenic loop is lessened each time one revisits it; a novel scenic sight is a great reward, but seeing it again and again is tedious and not rewarding. To accommodate this, we would have to expand the state space to include a memory – a state is now represented by a current location and a collection of past-visited locations. The reward for visiting a new location is now a (diminishing) function of the number of times it has been seen before.

e) Real domains with looping behavior include eating junk food and avoiding it or smoking and quitting.

4.1

The sequence of queues is as follows:

- \( L[0+244=244] \)
- \( M[70+241=311], T[111+329=440] \)
- \( L[140+244=384], D[145+242=387], T[111+329=440] \)
- \( D[145+242=387], T[111+329=440], M[210+241=451], T[251+329=580] \)
- \( C[265+160=425], T[111+329=440], M[210+241=451], M[220+241=461], T[251+329=580] \)
- \( T[111+329=440], M[210+241=451], M[220+241=461], P[403+100=503], T[251+329=580], R[411+193=604], D[385+242=627] \)
- \( M[210+241=451], M[220+241=461], L[222+244=466], P[403+100=503], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627] \)
- \( M[220+241=461], L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627] \)
- \( L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], L[290+244=534], D[285+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627] \)

4.2

\( w=0 \) gives \( f(n)=2g(n) \). This behaves exactly like UCS – the factor of two makes no difference in the ordering of the nodes. \( w=1 \) gives A* search. \( w=2 \) gives \( f(n)=2h(n) \), i.e., greedy best-first search. We also have

\[
 f(n) = (2-w) g(n) + \frac{w}{2-w} h(n) 
\]

which behaves exactly like A* search with heuristic \( \frac{w}{2-w} h(n) \). For \( w \leq 1 \), this is always less than \( h(n) \) and hence admissible, provided \( h(n) \) is itself admissible.