*1. Give a complete problem formulation for each of the following. Choose a formulation that is precise enough to be implemented.*

*a. Using only four colors, you have to color a planar map so that no two adjacent regions have the same color.*

a. Initial state: No regions colored.

Actions (3rd ed.)/Successors (2nd ed.): Assign a color to an uncolored region.

Transition model (3rd ed.): The previously uncolored region has the assigned color.

Goal test: All regions colored, and no two adjacent regions have the same color.

Cost function: Number of assignments.

 *b. A 3-foot tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.*

b. Initial state: As described in the text.

Actions/Transition model/Successors: Hop on crate; Hop off crate; Push crate from one spot to another; Stack one crate on another; Walk from one spot to another; Grab bananas (if standing on crate).

Goal test: Monkey has bananas.

Cost function: Number of actions.

*c. You have a program that outputs the message “illegal input record” when fed a certain file of input records. You know that processing of each record is independent of the other records. You want to discover what record is illegal.*

c. Initial state: considering all input records.

Goal test: considering a single record, and it gives “illegal input record” message.

Actions/Transition model/Successors: run again on the first half of the records; run again on the second half of the records.

Cost function: Number of runs.

Note: This is a contingency problem; you need to see whether a run gives an error

message or not to decide what to do next.

*d. You have three jugs measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.*

d. Initial state: jugs have values [0, 0, 0].

Actions/Transition model/Successors: 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 current values x and y, pour y into x; this changes the jug with x to the minimum of x + y and the capacity of the jug, and decrements the jug with y by the amount gained by the first jug.

Cost function: Number of actions.

*2. (Adapted from R&N 3.15, p. 116, by changing “visited” to “expanded”) Consider a state space where the start state is the number 1 and each state k has two successors: numbers 2k and 2k+1.*

*a. Draw the portion of the state space for states 1 to 15.*

 1

 2 3

 4 5 6 7

8 9 10 11 12 13 14 15

*b. Suppose the goal state is 11. List the order in which nodes will be expanded for breadth-first search, depth-limited search with limit 3, and iterative deepening search.*

b. Breadth-first: 1 2 3 4 5 (11)

 (BFS does goal-test when children are generated, so 11 is found when 5 is expanded.

 See R&N Fig.3.11, p. 82.)

Depth-limited: 1 2 4 5 (11)

 (DLS iterates over children one at a time, doing goal-test as the first step of the recursive

 call, so 11 is found on the expansion of 5 when the goal-test is done on 11.

 Note that goal-test is done on nodes 8, 9, 10, and 11, but they are not expanded because

 they are at the depth limit of 3 [recall root = depth 0]. R&N Fig. 3.17-18.)

Iterative deepening: 1; 1 2 3; 1 2 4 5 (11)

 (Similar comments to DLS.)

*c. How well would bidirectional search work on this problem? What is the branching factor in each direction of the bidirectional search?*

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. Does the answer to (c) suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a goal state with almost no search?*

d. Yes; start at the goal, and apply the single reverse successor action until you reach 1.

*e. Call the action of going from state k to 2k Left, and the action of going to 2k+1 Right. Can you find an algorithm that outputs the solution to this problem without any search at all?*

e. f(n) =

{IF (n=1) THEN () ELSEIF (even(n)) THEN f(floor(n/2)).Left ELSE f(floor(n/2)).Right}

*3. Prove each of the following statements, or give a counter-example:*

*a. Breadth-first search is a special case of uniform-cost search.*

a. When all step costs are equal, g(n) ∝ depth(n), so uniform-cost search reproduces breadth-first search.

*b. Depth-first search is a special case of best-first tree search.*

b. Depth-first search is best-first search with f(n) = −depth(n); breadth-first search is best-first search with f(n) = depth(n); uniform-cost search is best-first search with f(n) = g(n);

greedy-best-first search is best-first search with f(n) = h(n); A\* search is best-first search with f(n) = g(n) + h(n).

*c. Uniform-cost search is a special case of A\* search.*

c. Uniform-cost search is A\* search with h(n) = 0.

*4. Give the name that results from each of the following special cases:*

*a. Local beam search with k=1.*

a. Local beam search with k = 1 is hill-climbing search.

*b. Local beam search with one initial state and no limit on the number of states retained.*

b. Local beam search with k = ∞: strictly speaking, this doesn’t make sense. The idea is that if every successor is retained (because k is unbounded), then the search resembles breadth-first search in that it adds one complete layer of nodes before adding the next layer. Starting from one state, the algorithm would be essentially identical to breadth-first search except that each layer is

generated all at once.

*c. Simulated annealing with T=0 at all times (and omitting the termination test).*

c. Simulated annealing with T = 0 at all times: ignoring the fact that the termination step

would be triggered immediately, the search would be identical to first-choice hill climbing because every downward successor would be rejected with probability 1.

*d. Simulated annealing with T=infinity at all times.*

d. Simulated annealing with T = infinity at all times: ignoring the fact that the termination step would never be triggered, the search would be identical to a random walk because every successor would be accepted with probability 1. Note that, in this case, a random walk is approximately equivalent to depth-first search.

*e. Genetic algorithm with population size N=1.*

e. Genetic algorithm with population size N = 1: if the population size is 1, then the

two selected parents will be the same individual; crossover yields an exact copy of the

individual; then there is a small chance of mutation. Thus, the algorithm executes a

random walk in the space of individuals.