School of Electrical Engineering and Computer Science, Washington State University
Fall 2016

CptS 440/540 Artificial Intelligence

Homework 2

Solutions

1. Recall the Wumpus World search task. Consider a simplified search-based approach for moving our Wumpus World agent using the state representation [*x, y, d*], where *x, y* is the location of the agent, and *d* is the direction {U=up, D=down, L=left, R=right} that the agent is facing. The agent has three actions: F=GoForward, L=TurnLeft, and R=TurnRight. The initial state is [1, 1, R].

   a. Show the search tree generated by A* search, where the goal test is [2, 2, U], and actions are always tried in the order F, L, R. For each node, show the value of the evaluation function $f(n)=g(n)+h(n)$. The heuristic function

      $h(n)$ = (city-block-distance + discrete-angular-distance)

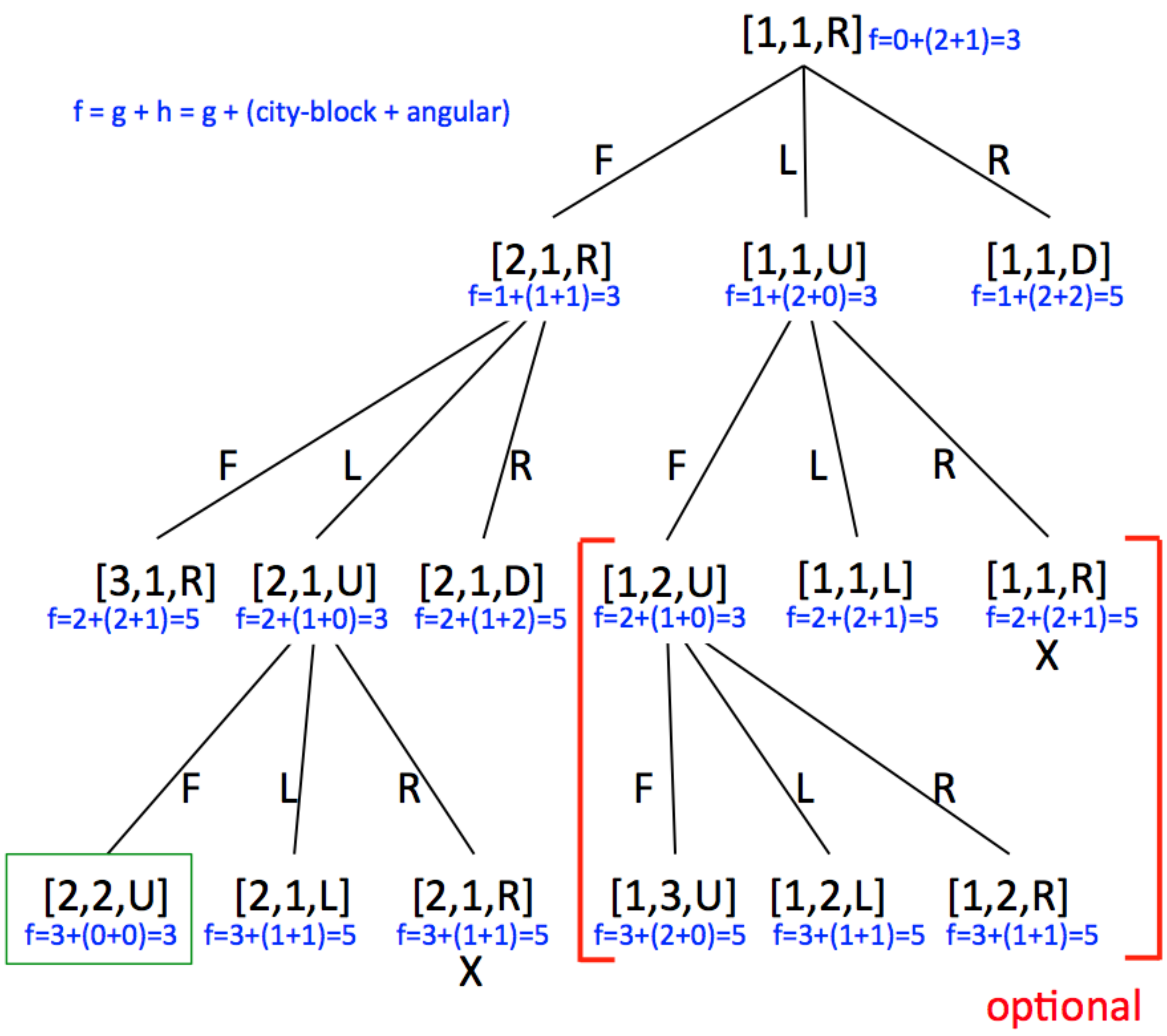
      For example, the initial state has a city-block-distance of 2 from the goal state, and a discrete-angular-distance of 1 (requires 1 TurnLeft to be in the up orientation); thus, the heuristic value of the initial state is 2 + 1 = 3. Note that discrete-angular-distance is always one of {0, 1, 2}.

   b. Repeat part (a), but with a different heuristic function
      $h(n)$ = (city-block-distance)^2 + (discrete-angular-distance).

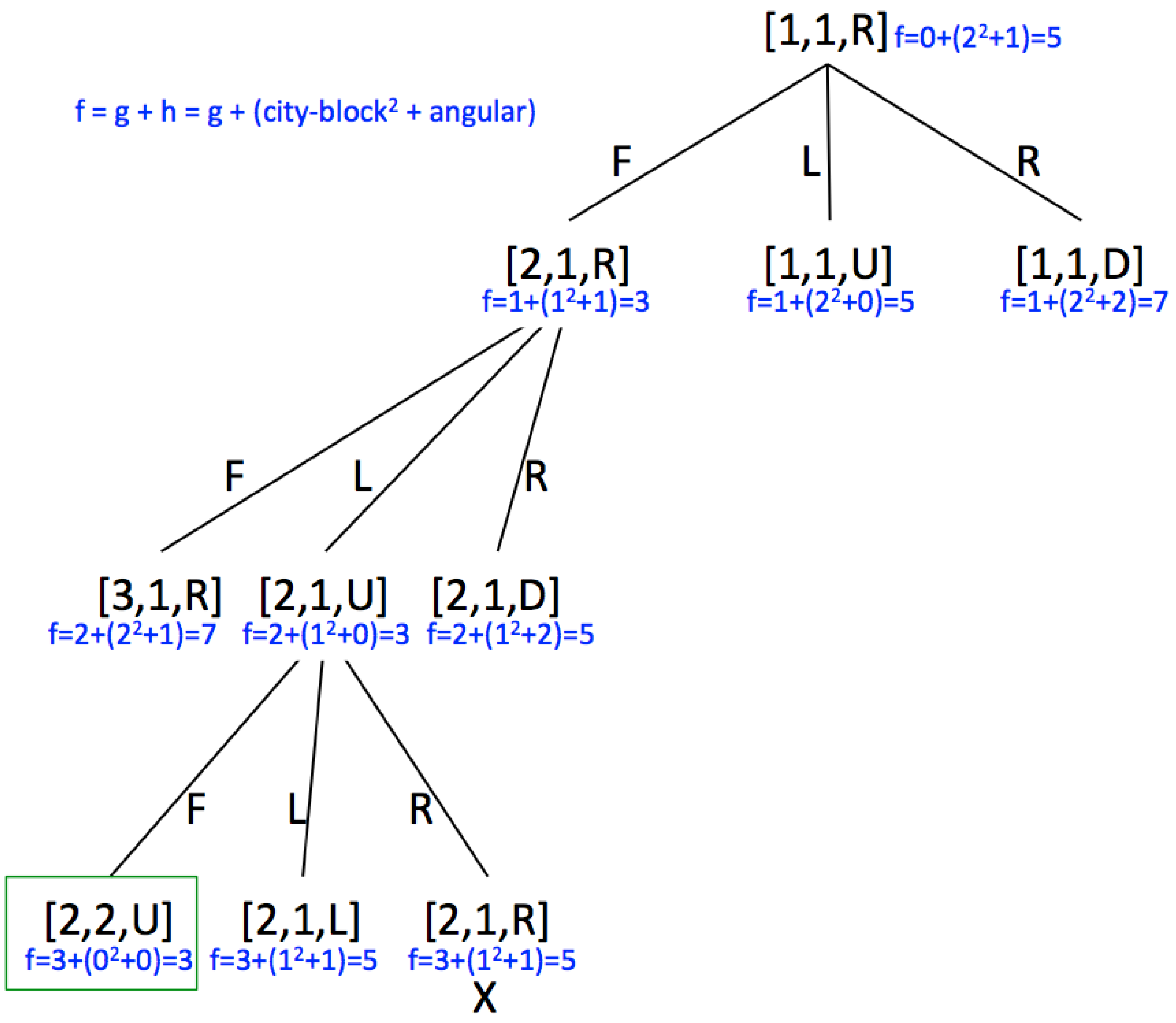
   c. Is the heuristic function from part (b) admissible? Justify your answer.

Solution:

a. Below is the search tree with the evaluation function f(n) shown at each generated node, and the goal in green. Nodes marked with an X are generated and evaluated, but they are not added to the Frontier list, because they are duplicates (i.e., already on the Explored list). The nodes within the red area are optional, depending on in which order the f=3 nodes are removed from the Frontier.



b. Below is the search tree with the evaluation function f(n) shown at each generated node, and the goal in green. Nodes marked with an X are generated and evaluated, but they are not added to the Frontier list, because they are duplicates (i.e., already on the Explored list). Note that in this case the optional nodes from part (a) would definitely not be generated, because their f value is greater than 3 using the new heuristic.



c. No. The heuristic can over-estimate the cost to the goal. E.g., if the goal was [3, 1, R], then the heuristic would evaluate to 4, but the optimal cost is 2 (F, F).

2. Suppose we want to use the Hill-Climbing search algorithm to solve the 8-puzzle problem using the "Number of Tiles Correct" heuristic (which equals 8 – "Number of Tiles Out").
   a. Draw the search tree generated for the initial state and goal state in **Figure 1a**. Show the value of the heuristic next to each node. Indicate which state the algorithm will return.
   b. Consider a different initial state (shown below in **Figure 1b**). Repeat part (a) for this new problem. Again, show the value of the heuristic next to each node and indicate which state the algorithm will return.

1	2	3
8	5	6
4	7	

Initial State

1	2	3
4	5	6
7	8	

Goal State

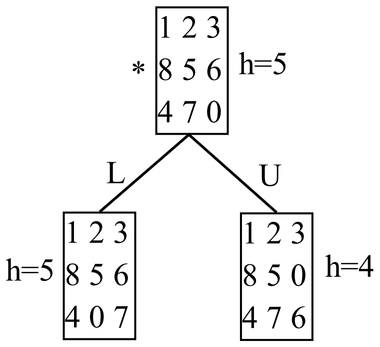
Figure (1a)

1	2	3
4	8	5
7		6

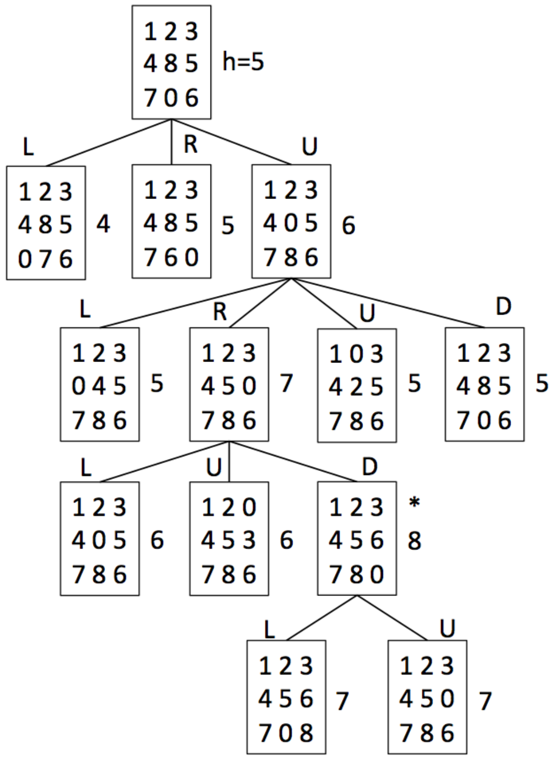
Figure (1b)

Solution:

a. See below. Since neither child's heuristic value is greater than the parent, then the Hill-Climbing algorithm terminates and returns the parent (node with asterisk).

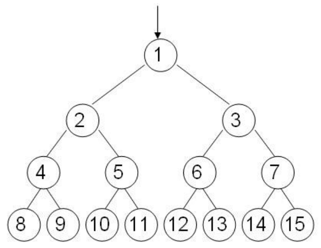


b. See below. For this problem the heuristic works and finds the solution. Hill-climbing does keep search beyond the solution, but does terminate there, since both children have a small heuristic value.



3. *Solution*:

   a. See search tree below.



   b. Let the goal state G = 11. List the order of nodes that are expanded for breadth-first search, depth-limited search to depth 3, iterative deepening DFS.
      - BFS: < 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 >
      - DLS to depth 3: < 1, 2, 4, 8, 9, 5, 10, 11 >
      - IDDFS: << 1 >,< 1, 2, 3 >,< 1, 2, 4, 5, 3, 6, 7 >,< 1, 2, 4, 8, 9, 5, 10, 11 >>

c. Suppose we call the action of going from state *i* to state *2i* LEFT while the action of going from state *i* to state *2i + 1* RIGHT. Devise an algorithm that outputs the solution to this problem without any search at all.

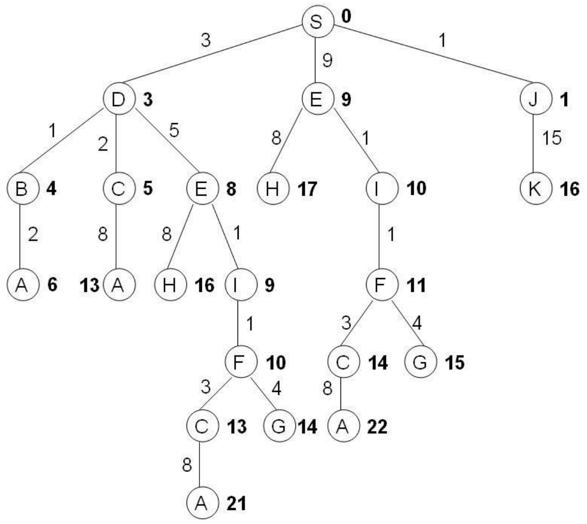
Solution: See the *LeftRight* algorithm that takes the goal G (a positive integer) as its initial input.

```
LeftRight (G)
    {
        If (G = 1)
        Then return
        Else {
            LeftRight (⌊G/2⌋)
          if (G mod 2 = 0) // if G is even
            Then print LEFT
          Else print RIGHT // if G is odd
            }
        }
```

NOTE: English explanation is also fine; or any combo of English and (pseudo-)code, as long as the core idea is correct.

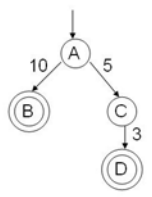
4) Solution:

a) The search tree is given below. The sequence of nodes and their associated path costs are: S(0), J(1), D(3), B(4), C(5), A(6), E(8), E(9), I(9), F(10), I(10), F(11), A(13), C(13), C(14), G(14).



b) In uniform-cost search, why is it important to apply the goal test to a node when it is selected for expansion rather than when it is generated? Please give an example.

Solution: It is important to apply the goal test to a node when it is selected for expansion rather than when it is generated to avoid missing the goal node with the lowest path cost g(n). For example, given the following state space where node A is the initial state and nodes B and D are the goal states, when we expand node A to generate nodes B and C, UCS will return a solution with path cost of 10 instead of 8 when the goal test is performed during node generation rather than during node expansion.



5) Explain in 1-2 concise, to-the-point sentences each of the following statements:
Solutions:

a. Breadth-first search is a special case of uniform-cost search.
Solution: If we assume that all the step costs are equal, then the path cost *g(n)* of a node *n* is directly proportional to its depth *depth(n)*. Uniform-cost search with $g(n) \propto depth(n)$ is breadth-first search, because it will expand the shallower nodes first then deeper nodes.

b. Breadth-first search is a special case of best-first search.
Solution: Best-first search with $f(n) = depth(n)$ is breadth-first search.

c. Depth-first search is a special case of best-first search.
Solution: Best-first search with $f(n) = -depth(n)$ or $f(n) = 1 / (depth(n) + 1)$ is depth-first search. (Basically, any function *f(n)* that prefers lower depth over higher depth, and is monotonically decreasing with depth)

d. Uniform-cost search is a special case of A* search.
Solution: A* search with $h(n) = 0$ is uniform-cost search.

e. Greedy best-first search is a special case of A* search.
Solution: A* search with g(n) = 0 is greedy best-first search.

f. Hill climbing is a special case of beam search.
Solution: Beam search with $k = 1$ (that is, the number of current states is 1) is equivalent to hill climbing.

g. Depth-first search is a special case of depth-limited search.
Solution: Depth-limited search with depth limit $l = \infty$ is depth-first search.