CMSC 671 – HW 2, Fall 2019

Homework 2: Search & CSP
Turnin: Blackboard.
Please submit Parts I, II, and IV as a **single PDF file** named *yourlastname*_hw2.pdf.
Please submit Part II as a **.py** file, named *yourlastname*_hw2.py.

All files must start with your last name and have your full name in the file, at/near the top.

PART I. UNINFORMED AND INFORMED SEARCH

Description: For the tree in Figure 1, S is the start state, and any node with a double line is a goal state. Actual arc costs are given on the arc (in *blue*). Table 1 gives the value of a heuristic function for each node.

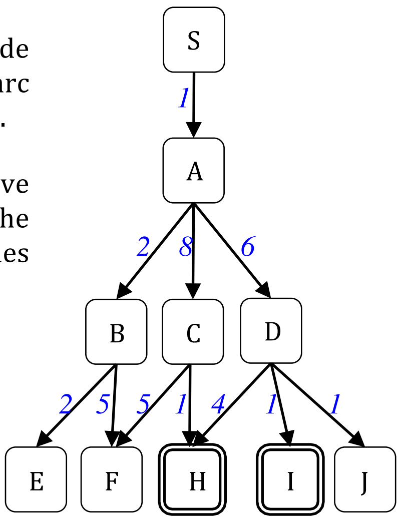

Figure 1: A simple search tree

1) For each of the following algorithms, *at each timestep*, please give the current node plus all nodes on the frontier (in order), using the same notation as we used in class. Then, give the list of all nodes visited by that search, in order. *12 pts*

a) Depth-first search

b) Breadth-first search

h(S) = 2
h(A) = 3
h(B) = 5
h(C) = 10
h(D) = 4
h(E) = ∞
h(F) = ∞
h(H) = 0
h(I) = 0
h(J) = ∞

Table 1: Values of some heuristic function applied to the nodes of that tree.

c) Uniform-cost search

d) A* search

PART II. JUMPING PUZZLES

Description: We will consider number-jumping problems (example in Fig. 3).[1] In this class of problems, the goal is to get from a start space (in the example, top left, in red) to a goal space (bottom right, orange). Each cell contains a number; from each cell, the agent can move exactly that number of spaces horizontally or vertically.

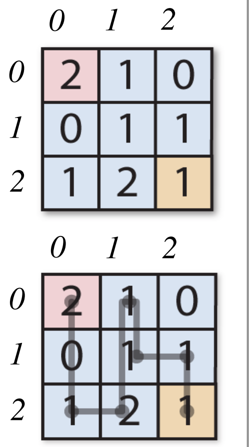

Figure 3: A 3x3 number-jumping problem and its 6-jump solution

The problem is to find a path to the goal. The *optimal* solution has the smallest number of jumps (not the smallest distance traveled). There may be multiple solutions.

1) What is this problem's *state space*? (What information is needed to describe any given state an agent may be in while solving one?) *8 pts*

2) Describe a good admissible heuristic function $h(n)$ for this problem. *5 pts*

3) What is the (worst-case) branching factor b for an n x n puzzle? What is its depth d? *4 pts*

4) In Figure 3, the agent takes six actions and moves through seven states (one per stopping point), including the start and end states. What value does your heuristic function return when applied to *each* of these seven states? *7 pts*

[1] Examples from: http://www.mazelog.com/show?7

PART III. A* IMPLEMENTATION

Description: We will write three search-based solvers for the number-jumping problem described above. They will all take in an *n* x *n* matrix of arbitrary size, and return a list of cells visited along the path to the solution. For each one, experiment with how long it takes to solve a puzzle and how large a puzzle it can solve in reasonable time. You may (and are encouraged to) use the algorithms in the book as a starting point. Careful coding of your search and queueing functions will reduce the workload substantially. Document any other sources you use, and please don't import (or copy) any *search* code from elsewhere. *40 pts*

All solutions will take two tuples (start and goal) and a matrix containing the problem as arguments, and find a path from the start state to the goal state. **Return** (not print) a list of tuples containing the cells visited on the path found.

The example in Figure 3 would be as follows:

```
>>> jumpsolve_astar ((0,0),  (2,2),  [[2,1,0],  [0,1,1],  [1,2,1]]) =>
                                [(0,0), (2,0), (2,1), (0,1), (1,1), (1,2), (2,2)]
```

1) Write a function called jumpsolve_bfs that finds any solution using **breadth-first search**. Enqueue generated nodes in the following order: left (west), up (north), right (east), down (south).

2) Write a function called jumpsolve_ids that finds any solution using **iterative-depth search**. Enqueue generated nodes in the following order: left (west), up (north), right (east), down (south).

3) Write a function called jumpsolve_astar that finds an *optimal* solution using **A* search**. Use the heuristic you gave in Part II. (Things to think about: if you experiment with different heuristics, how does this change? Can you do better?)

You may assume:

- The puzzle dimensions are nonzero.
- The array is always 0-indexed.
- **Tuples are always in (x,y) order (row, column).**

Do not assume:

- The *agent* is unable to backtrack. (It can!) Remember this is different from *search* backtracking.
- Start or goal states will always be along an edge or in a corner. (They won't!)
  The start state will be different from the goal state.

PART IV. CONSTRAINT SATISFACTION

Description: You are trying to find professors to teach the following classes: CS101 at 10am, CS201 at 11am, CS301 at 12pm, CS401 at 1pm, and CS501 at 2pm. This is constrained by the following:

- Dr. Jones can teach 101, 301 or 401.
- Dr. Smith can teach 101 or 201.
- Dr. Taylor can teach 101, 201, or 301.
- Anybody can teach 501.
- Professors can't teach back-to-back classes (e.g., at 10 and 11, or 1 and 2.)
- Dr. Smith can only teach afternoon classes.
- Dr. White can teach 101 or 401, but not both.
- No-one can teach more than two classes.

1) Give a CSP formulation for this problem (NOT the solution). *10 pts*

a) What are the *random variables*? (There are multiple options; choose carefully.)

b) What is the *domain*?

c) Express the constraints using the domain and variables (give statements of values each variable can take or cannot take).

2) Give a full legal instantiation for this problem, or explain why none exists. *4 pts*