CMSC 671 (Principles of AI)

Homework 1: AI, Agents, Search I

Turnin: Blackboard.

Submit:
- Parts I-IV together as a **single PDF file** named _yourlastname__hw1.pdf.
- Part V as a **single python file** named _yourlastname__hw1.py.

Notes:
- These are individual assignments, not group work.
- Please clearly delineate individual sections of the homework.

PART I. BEING INTELLIGENT (40 PTS)

Reading: Read Chapter 27.1 and 27.2 in our textbook. You will likely need to look for additional information about artificial intelligence tasks, successes, and failures over time. Make sure to list these additional sources in a bibliography.

Assignment: Answer all of the following in a single short **essay** (roughly 500-750 words, not counting your bibliography). Because this is an essay, all parts of your answer must form a consistent, coherent story. Do not give bullet points, and do not give separate essays.

- Do you think an artificial agent can be 'intelligent'? Why or why not?
  - Discuss your answer **in terms of the traditional arguments** against thinking machines—do you agree with some of those arguments? Why or why not?
- Consider the things AI can currently do, and some outstanding problems/tasks AI cannot currently handle.
  - What is the most useful thing AI can currently do? What are you most excited about that AI can't currently do but should be able to in future?
  - What tasks do you think will be **hardest** for machines to accomplish, and why? Do you think an AI will eventually be able to accomplish it? Why or why not?

PART II. AGENTS (15 PTS)

Fill out the following PEAS information for agents doing these tasks. This is a design question—how would you design this agent? What would you use from the environment? What would you consider a 'good' performance? (Create your own table of responses.)

System	Performance measure	Environment	Actuators	Sensors
Example: *Robot Soccer Player*	*Winning games, scoring goals for team*	*Field, ball, teammates, other team, own body*	*Kickers (legs), movement (legs or wheels)*	*Camera, touch sensors, wheel encoders*
(a) Stock market agent				
(b) Luggage robot navigating an airport				
(c) Minesweeper-playing agent				
(d) Adaptive thermostat				

PART III. SEARCH ALGORITHMS (20 PTS)

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 arcs (in *blue*). Table 1 gives the value of a heuristic function for each node.

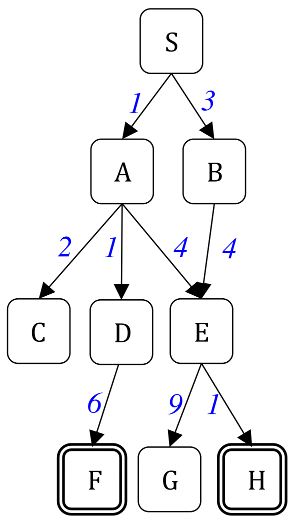


Figure 1: A simple search tree

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.

a) Depth-first

b) Breadth-first search

$h(S)$	=	2
$h(A)$	=	3
$h(B)$	=	5
$h(C)$	=	∞
$h(D)$	=	4
$h(E)$	=	9
$h(F)$	=	0
$h(H)$	=	0
$h(G)$	=	∞

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

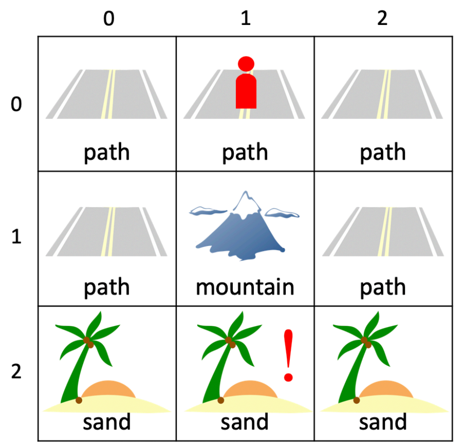
c) Uniform-cost search

d) A* search

Part IV. Navigating (Search Spaces and States) (35 pts)

The General Idea

Consider navigating an *n* x *m* space (3 x 3 example shown to the right). There are three kinds of terrain[1], each of which takes some amount of effort to traverse: entering a "path" cell costs 10 calories, entering a "sand" cell costs 50 calories, and entering a "mountain" cell costs 200 calories. In this example, your agent is at **(0,1)**, as shown, and is trying to reach square (2,1) (marked '!') via the path that costs the fewest possible calories. The agent cannot move diagonally.



Assignment: Answer the following questions about puzzles of this kind.

2) Represent this as a search problem. (10 pts)

- **a)** Describe the **state space**. (That is, explain what information is needed to describe any state an agent may be in while solving such a puzzle.)
- **b)** Provide a list of **actions (operators)**, including **constraints**.
- **c)** Draw the undirected **search graph** representing the complete space of the example puzzle. Mark the start and goal states.
- **d)** What is the **goal test** for the example puzzle?

3) What is the (worst-case) branching factor *b* for an *n* x *n* puzzle? (5 pts)

4) How many **unique, legal, reachable** states are there in this search space? (5 pts)

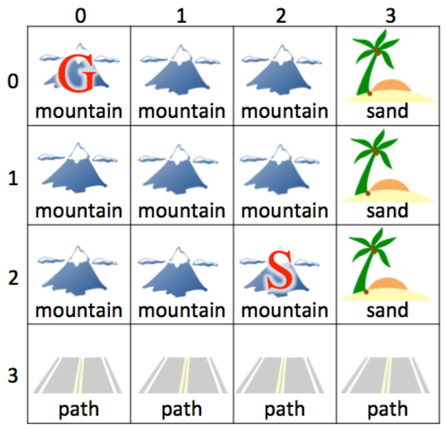
5) If you were using heuristic search: (15 pts)

- **a)** If the grid is *n* by *n*, what is the (maximum) size of the state space? Justify your answer.
- **b)** Describe an admissible heuristic *h*(*n*) for this problem.
- **c)** Explain how you know it is admissible.
- **d)** If we used an inadmissible heuristic to solve this problem, could it change the completeness of the search? Why or why not?
- **e)** If we used an inadmissible heuristic to solve this problem, could it change the optimality of the search? Why or why not?

[1] Image credits: pixabay.com/en/road-crossing-crosswalk-street-304283, pixabay.com/en/sand-beach-island-palm-sun-tree-304525, pixabay.com/en/mountain-peak-snow-summit-304054

Part V. Path-Finding (40 pts)

The General Idea



Write a program **in Python** to read in and solve puzzles of the type described above. The puzzle may be of any size and will have a random selection of path, sand, and mountain cells. The goal of the puzzle is to move your agent from a starting cell to a goal cell. S and G may be any cell. The agent's task is to find the lowest-cost path from S to G.

Some puzzles of this form will have multiple optimal solutions.

Details

You may assume:

- The size of the square is nonzero.
- The agent may only move one square at a time (not diagonally).
- The array is always 0-indexed.
- **Tuples are always in (x,y) order. (horizontal, vertical/row, column)**

You may *not* assume:

- The agent is unable to backtrack. (The search may and probably will, but the agent can too!)
- Start or goal states will always be along an edge. (They won't!)
- The start state will be different from the goal state. (Usually it will, but not always!)

Assignment: Write a function called `solve` which takes a matrix (the problem) and two tuples (start and goal), and tries to find a path from the start state to the goal state that is optimal (lowest possible cost). Our two examples would be passed in as follows:

```
>>> solve((1,0), (2,1), [[p,p,p], [p,m,p], [s,s,s]])
>>> solve((2,2), (0,0), [[m,m,m,s], [m,m,m,s], [m,m,m,s], [p,p,p,p]])
```

Your solver should take these values, and **return** (not print) a string containing a list of moves. Moves should be represented by the capital letters N,S,E,W for north, south, east, and west moves. The optimal solution to the 3 x 3 board would be returned as "NEESS". **Please don't print** anything inside the function; everything must be in the return value.

Implement your solution to this problem as: A* search. Use the heuristic you gave in Part I. For now, you **may not** use any of the optimizations we covered in class (e.g., keeping track of previously expanded nodes)—this is pure A*.