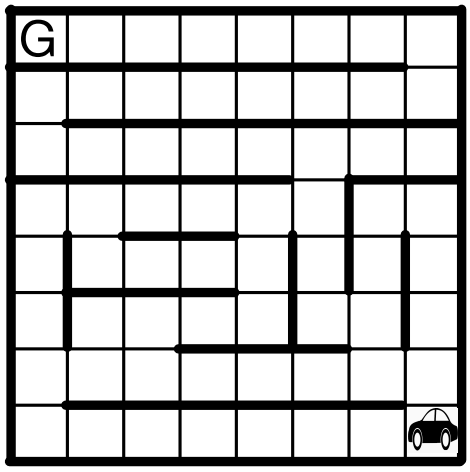
Artificial Intelligence Homework 1

If you have not already done so, read textbook sections 3.1-3.6.

1. Imagine a car-like agent wishes to navigate a maze such as this one:



The agent is directional and at all times faces some direction $d \in \{N, S, E, W\}$. With a single action, the agent can either move forward at an adjustable velocity v, or turn. It cannot move forward and turn at the same time.

The turning actions are LEFT and RIGHT, which change the agent's direction by a 90 degrees rotation. Turning is only permitted when the velocity is zero (and leaves it at zero).

The moving actions are MOVE, MOVE-FASTER and MOVE-SLOWER. MOVE moves the car in its current direction forward by the number of squares equal to its velocity. MOVE-FASTER increments the velocity by 1 and then moves forward the number of squares equal to its new velocity. MOVE-SLOWER decrements the velocity by 1 and then moves forward the number of squares equal to its new velocity.

Any action that would result in crashing into a wall is illegal (meaning it is not available as an action from the state in question). Any action that would reduce v below 0 or above some maximum speed V_{max} is also illegal.

The agent's goal is to find a driving strategy that parks it (stationary, with $v = 0$, direction facing irrelevant) on the goal square G in as few actions (time steps) as possible. Every action (turning 90 degrees, or driving forward at the current velocity) takes 1 unit of time.

Note that this is problem formulation is different from the regular navigation problem!

All of these questions pertain to a generic maze of size m by n, not specifically the maze above, which is only shown to illustrate how the problem is set up.

(a) Formulate what a state looks like for this problem. Define it using specific data types.

(b) If the grid is m by n, what is the (maximum) size of the state space? Justify your answer. You should assume that all possible states are reachable from the start state.

(c) What is the maximum branching factor of this problem? You may assume that illegal actions from a state are never considered. Briefly justify your answer.

(d) The Manhattan distance between two points is defined to be the sum of the total vertical and horizontal differences between the points. In other words, if you imagine a city laid out using a grid (like Manhattan), the Manhattan distance is the total distance you would have to walk to get from one intersection to another (because no streets run diagonally).

For our car driving problem, Is the Manhattan distance from the agent's location to the goal location an admissible heuristic? Why or why not?

(e) Describe and justify a non-trivial admissible heuristic for this problem that is not the Manhattan distance to the goal.

(f) If we used an inadmissible heuristic to solve this problem, could it change the completeness of the search? Why or why not?

(g) If we used an inadmissible heuristic to solve this problem, could it change the optimality of the search? Why or why not?

2. Assume you are given a graph like the one we used in class for A*, with vertices representing locations and the edge weights representing travel time between locations. (That graph is supplied for you at the end of this homework to refresh your memory.) The *travelling salesperson problem* asks, "What is the fastest route that begins at a specific starting location, visits all the other locations in the graph in any order, and returns back to the starting location?" Suppose we want to solve this problem using A*.

   (a) Formulate what a state could be represented for this problem. Define this representation using specific data types. What does the initial state look like, and what does a/the final state(s) look like? (If it helps to use specific examples from the in-class graph, you may.) Hint: You will need to represent/store more than just the current location of the salesperson.

   (b) Suppose I define a heuristic for this problem to be the sum of all the straight-line distances from the salesperson to all the locations the salesperson hasn't visited yet. Explain why this heuristic is not admissible.

   (c) Define an admissible heuristic for this problem.

3. Suppose you are solving a problem using A* and you manage to invent a *perfect* heuristic function for your problem, which you call h*. In other words, instead of being an estimate of how far away node n is from a goal state, h*(n) tells you the *exact* cost of the shortest path from n to a goal state. For instance, in the navigation problem from class, where we were trying to travel from location I to location A, h*(n) wouldn't be the straight-line distance from node n to location A, it would be the length of the true shortest path from n to A, using the edge weights in the graph itself. In other words, h*(I) = 10, because the true shortest path is I-H-D-B-A.

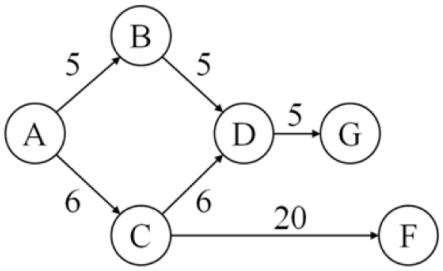
   Recall that as a heuristic function gets better and better (i.e., a better approximation of the true shortest path length), the A* algorithm runs faster and faster --- this happens because using heuristic that are better estimates of the true shortest path causes us to visit/expand fewer frontier nodes that do not fall along the true shortest path from the initial state to a goal state. Therefore, logically, if we had access to this perfect heuristic function, h*, then the A* algorithm would run as fast as possible.

   Note that one could certainly generate this perfect heuristic h* for any problem, simply by running

Dijkstra's algorithm from the initial state to all possible goal states and storing the shortest path length. But that's clearly very inefficient and would remove the point of using A* anyway.

Assume we have access to this theoretical perfect heuristic function h*. Normally, during the node expansion step of A*, while visiting a node n, we generate and add all of n's successor states to the frontier (that aren't already on the explored list or already exist in the frontier with lower f-costs). It turns out that during this step, *if we have access to h*,* we can figure out the one single successor state of n that lies on the shortest path from n to a goal and only add that state to the frontier, skipping all the others. Clearly, this would save us a lot of time---we wouldn't need to bother putting states on the frontier that will never be part of the shortest path. **Describe, for a node n, how to use the h* function to directly find the successor state of n that is on the shortest path from n to a goal state. (Mathematically, this is a pretty short description, so be brief.)**

4. Consider the following graph, where we are trying to find the shortest path from vertex A to either F or G (both F and G are goal states).



Here are four possible heuristic functions h1(n) through h4(n):

	A	B	C	D	F	G
h1	0	0	0	0	0	0
h2	11	7	7	3	0	0
h3	13	9	7	1	0	0
h4	15	10	11	5	0	0

(a) Which, if any, of the heuristics are admissible?

(b) Which, if any, of the heuristics are consistent?

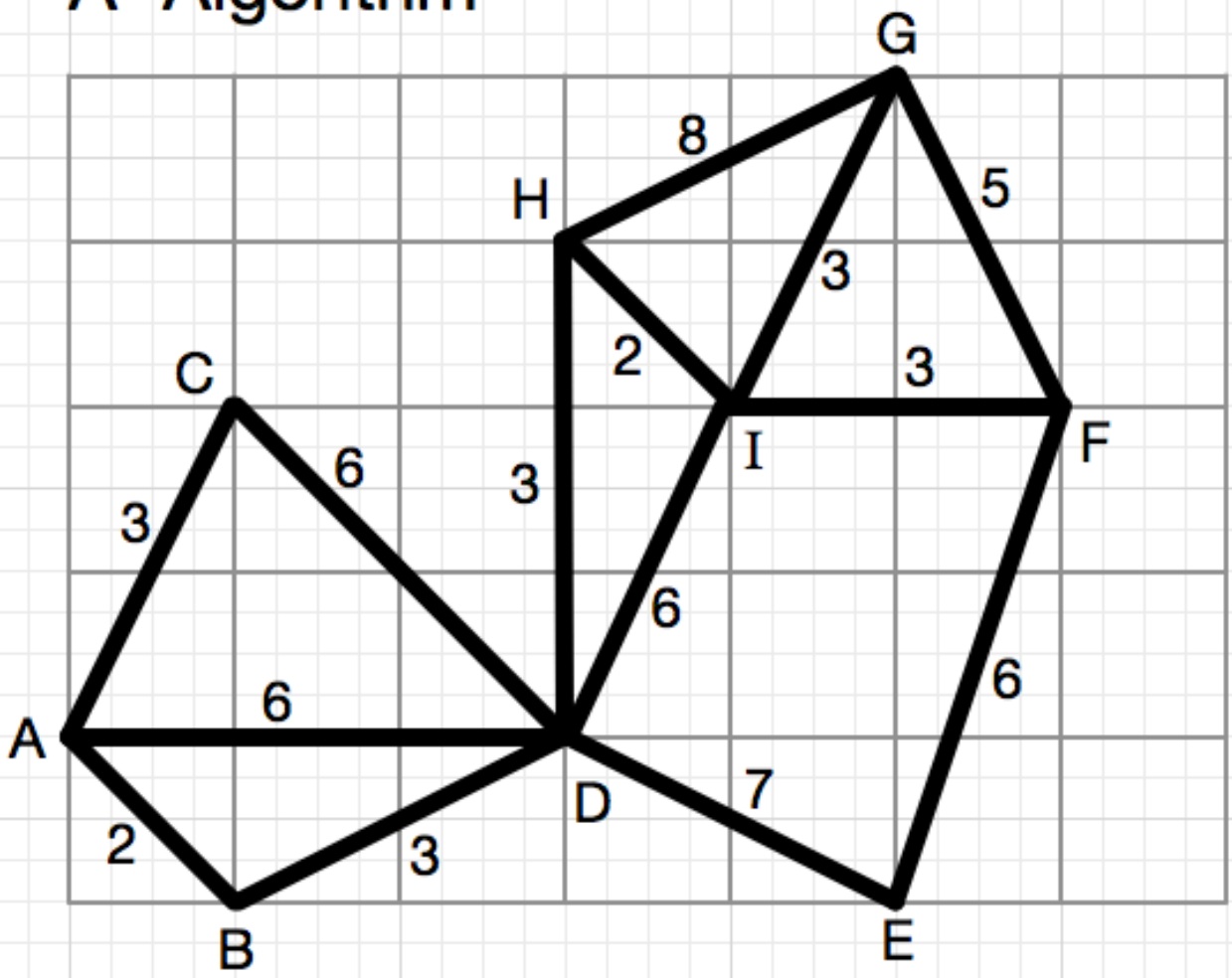
(c) Run A* (by hand) using h3, showing the search tree generated, and the frontier and explored lists. At the end, please also provide a list showing in what order nodes were visited (this can be done by numbering your frontier nodes as you pop them off the priority queue, or just make a separate list somewhere else on the page).

What is the best path returned by A*?

(d) Recall that the term best-first search refers to any search algorithm that uses a heuristic function to prioritize the frontier by which nodes would be best to examine first. A* is an example of an algorithm that falls into this category. Consider a different algorithm that also falls into this category, called greedy best-first search. This algorithm sorts the frontier by f(n) = h(n), rather than f(n) = g(n) + h(n). Run greedy best first search by hand using h3, following the same procedure as part (c).

What is the best path returned by greedy best-first search?

A* Algorithm



h(n) estimates for distance from n to A

n	h(n)
A	0
B	1.4
C	2.2
D	3
E	5.1
F	6.3
G	6.4
H	4.2
I	4.5