1 Slac-Man and the Hazards of Compiled Knowledge (4 points)

One way in which agents avoid re-solving problems from scratch is to *compile knowledge* into their environments. For example, rather than making each of us wander the freeways to see where they go, society has been kind enough to put up signs (e.g. *San Francisco 20 miles*) to tell us the results of such explorations. Tired of solving PositionSearchProblem, generations of Pac-Man agents have wised up to this strategy and have annotated some cells of a one-dot maze with the actual shortest path distance to the dot. Slac-Man, a particularly lazy agent, is trying to compute an optimal route in such an environment. For speed, Slac-Man uses A∗ graph search with the heuristic defined as follows: for the special annotated cells, $h(x) = h^*(x)$, while for all others, $h(x) = 0$. ($h^*(x)$ is the actual lowest cost path to a goal from $x$.)

(a) (1 point) Is $h$ admissible? Briefly justify why or show why not with a simple counterexample.
Yes, $h$ is admissible. Either $h(x) = 0 \leq$ true distance, or $h(x) = true distance$

(b) (1 point) Is $h$ consistent? Briefly justify why or show why not with a simple counterexample.
No, $h$ is not necessarily consistent. Counterexample: suppose we moved from a node $x$ which was annotated with a true cost $c > 1$ to one of its neighbors $y$ which happened not to be annotated. Then $h(x) = c > step-cost(x,y) + h(y) = 1 + 0$.

(c) (1 point) Will Slac-Man’s search be complete? If not, give a specific change to make it complete.
Pacman’s search is complete; all graph searches are complete in finite state spaces. A route will be found if one exists.

(d) (1 point) Will Slac-Man’s search be optimal? If not, give a specific change to make it optimal.
Pacman’s search is not optimal. To make it optimal, several options exist, including (1) every could be annotated with the true cost, not just some nodes, (2) $h = 0$ could be used, (3) tree search could be used, (4) improved nodes can be re-expanded, and other methods. Of course not all of these options are equally practical or equally efficient.
2 Conformant Search (4 points)

Consider an agent in a maze-like grid, as shown to the right. Initially, the agent might be in any location \( x \) (including the exit \( e \)). The agent can move in any direction (\( N, S, E, W \)). Moving into a wall is a legal action, but does not change the agent’s actual position. Formally, let \( post(\ell, d) \) be the possibly unchanged location resulting from choosing direction \( d \) from location \( \ell \). The agent is trying to reach a designated exit location \( e \) where it can be rescued. However, while the agent knows the layout of the maze, it has no sensors and cannot tell where it is.

The agent must devise a plan which, on completion, guarantees that the agent will be in the exit location, regardless of the starting location. For example, here, the agent might execute \([W,N,N,E,E,N,N,E,E,E]\), after which it will be at \( e \) no matter where it started.

Your answers below should be in terms of the maze dimensions \( n \times m \) and not specific to the example maze shown here.

(a) (3 points) Formally state this problem as a single agent state-space search problem. You should formulate your problem so that your state space is finite (e.g. do not use an encoding where each partial plan is a state).

States:
A set \( L \) of possible locations for the agent.

Size of State Space:
The number of states is \( \leq 2^{m \times n} \), the number of subsets \( L \) of the \( m \times n \) locations.

Start state:
Set containing every map location that is not a wall.

Successor function:
\( \text{Successor}(L, d) = \{ post(\ell, d) \text{ for } \ell' \in L \} \), The set of locations that result from performing the action at every location in the previous set. Each state has four successor states, resulting from the four actions \( N, S, E, \) and \( W \)

Maximum branching factor:
4, the number of actions

Goal test:
\( L = \{ e \} \), i.e. the state only contains the exit location

(b) (1 point) Give a non-trivial admissible heuristic for this problem. Briefly justify its admissibility.

\[ h(L) = \max_{\ell \in L} \text{manhattan}(\ell, e), \]

where \( e \) is the exit location and \( L \) is a state containing possible locations \( \ell \). The heuristic is admissible because in any shorter plan an agent in the furthest possible location from the exit cannot have reached it.
### 3 Campus Layout (4 points)

You are asked to determine the layout of a new, small college. The campus will have three structures: an administration building (A), a bus stop (B), a classroom (C), and a dormitory (D). Each building must be placed somewhere on the grid below. The following constraints must be satisfied:

(i) The bus stop (B) must be adjacent to the road.

(ii) The administration building (A) and the classroom (C) must both be adjacent to the bus stop (B).

(iii) The classroom (C) must be adjacent to the dormitory (D).

(iv) The administration building (A) must not be adjacent to the dormitory (D).

(v) The administration building (A) must not be on a hill.

(vi) The dormitory (D) must be on a hill or near the road.

(vii) All buildings must be in different grid squares.

Here, “adjacent” means that the buildings must share a grid edge, not just a corner.

(a) (1 point) Let the variables A, B, C, and D each range over the set of locations on the grid. Express the description above as unary and binary constraints over these variables. Implicit statements in precise but evocative notation such as different(X,Y) are acceptable.

1. adjacent(B, road)
2. adjacent(A,B)
3. adjacent(C,B)
4. adjacent(C,D)
5. NOT adjacent(A,D)
6. NOT on(A, hill)
7. on(D, hill) OR adjacent(D, road)
8. different(X,Y) if X ≠ Y for X, Y ∈ {A, B, C, D}
(b) (1 point) Cross out eliminated values to show the domains of all variables after unary constraints and arc consistency have been applied (but no variables have been assigned).

\[
\begin{array}{c}
\text{A} & [ & 3 & 5 & 6 & ] \\
\text{B} & [ & 3 & 6 & ] \\
\text{C} & [ & 2 & 3 & 5 & 6 & ] \\
\text{D} & [ & 2 & 3 & 4 & ] \\
\end{array}
\]

(c) (1 point) Cross out eliminated values to show the domains of the variables after \( B = 3 \) has been assigned and arc consistency has been rerun.

\[
\begin{array}{c}
\text{A} & [ & ] \\
\text{B} & [ & 3 & ] \\
\text{C} & [ & ] \\
\text{D} & [ & ] \\
\end{array}
\]

(d) (1 point) Give a solution for this CSP or state that none exist.

\[
\begin{align*}
B &= 6 \\
A &= \{3, 5\} \\
C &= \{3, 5\} \\
D &= \{2, 3, 4\} \\
\text{Assigning A:} \\
A &= 3 \\
C &= \{5\} \\
D &= \{4\}
\end{align*}
\]
4 Search as a CSP (6 points)

Consider the following generic search problem formulation with finitely many states:

- **States**: there are \( d + 2 \) states: \( \{s_s, s_g\} \cup \{s_1, \ldots, s_d\} \)
- **Initial state**: \( s_s \)
- **Successor function**: \( \text{Succ}(s) \) generates at most \( b \) successors
- **Goal test**: \( s_g \) is the only goal state
- **Step cost**: each step has a cost of 1

(a) (1 point) Suppose an optimal solution exists with cost \( n \). What is the tightest upper bound on \( n \) in terms of the quantities defined above?
The optimal solution is acyclic, and will not visit any state twice; thus, \( n \leq d + 1 \).

(b) (1 point) Suppose we must solve this search problem using BFS, but with limited memory. Specifically, assume we can only store \( k \) states during search. Give an upper bound on \( n \) for which the search will fit in the available memory (do not worry about off-by-one errors here, but give the tightest bound possible).
Note that in the worst case, the distinction between tree and graph search is not important here.
\[ b^n = k, \text{ so } n \leq \log_b(k) \]

(c) (1 point) Would any other search procedure allow problems with substantially deeper solutions to be solved? Either argue why not, or give a method along with an improved bound on \( n \).
DFS tree search will find deeper solutions (if very slowly), and iterative deepening will even find shallowest and therefore optimal solutions, all with the lower memory requirements of DFS. These methods would require \( O(nb) \) memory, giving a bound of \( k/b \). It is also acceptable to claim that no graph search method gives a better bound, because all must store a potentially exponential closed list.
(d) (1 point) If we knew the exact value of \( n \), we could formulate a CSP whose complete assignment specifies an optimal solution path \((X_0, X_1, \ldots, X_n)\) for this search problem. Give binary and/or unary constraints which guarantee that a satisfying assignment is a valid solution to the original search problem.

Variables: \( X_0, X_1, \ldots, X_n \)
Domains: \( \text{Dom}(X_i) = \{s_s, s_g\} \cup \{s_1, \ldots, s_d\} \quad \forall i \in \{0, 1, \ldots, n\} \)
Constraints: \( X_0 = s_s, X_i \in \text{Succ}(X_{i-1}) \quad \forall i \in \{1, \ldots, n\}, X_n = s_g \)

(e) (1 point) Assume the branching factor \( b \) is much less than \( d \). How can the successor function be used to efficiently enforce the consistency of an arc \( X_i \rightarrow X_{i-1} \)? (Reminder: Enforcing the consistency of this arc prunes values from the domain of \( X_i \), not \( X_{i-1} \).)

The consistent values in the domain of \( X_i \) are \( \bigcup \text{Succ}(x_{i-1}) \) for values \( x_{i-1} \) in the domain of \( X_{i-1} \). This allows the consistency of an arc to be checked / enforced in \( O(db) \) time, rather than \( O(d^2) \).

(f) (1 point) After reducing the domains of any variables with unary constraints, suppose we then make all arcs \( X_i \rightarrow X_{i-1} \) consistent, processed in order from \( i = 1 \) to \( n \). Next, we try to assign variables in reverse order, from \( X_n \) to \( X_0 \), using backtracking DFS. Why is this a particularly good variable ordering?

Because the tree-structured CSP is directionally arc-consistent, there is always a value for \( X_{i-1} \) that is consistent with the value chosen for \( X_i \); thus, the assignment search will not backtrack.