Search algorithms in action (*)

For each of the following graph search strategies, work out the order in which states are expanded, as well as the path returned by graph search. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. The start and goal state are S and G, respectively. Remember that in graph search, a state is expanded only once.

a) Depth-first search.
   States Expanded: Start, A, C, D, B, Goal
   Path Returned: Start-A-C-D-Goal

b) Breadth-first search.
   States Expanded: Start, A, B, D, C, Goal
   Path Returned: Start-D-Goal

c) Uniform cost search. (*)
   States Expanded: Start, A, B, D, C, Goal
   Path Returned: Start-A-C-Goal

d) Greedy search with the heuristic $h$ shown on the graph.
   States Expanded: Start, D, Goal
   Path Returned: Start-D-Goal

e) $A^*$ search with the same heuristic. (*)
   States Expanded: Start, A, D, B, C, Goal
   Path Returned: Start-A-C-Goal
2 15-puzzle (⋆)

The puzzle involves sliding tiles until they are ordered correctly. To solve these puzzles efficiently with A* search, good heuristics are important.

(1) Create a heuristic for the 15-puzzle based on the number of misplaced tiles. 
*Count the number of tiles out of place, not including the blank tile.*

(2) Create a heuristic using Manhattan distance. 
*Sum the Manhattan (city block) distances between each tile’s current position and its intended position.*

(3) Explain why your heuristics are admissible. 
*These heuristics are both relaxations of the original problem so the heuristic estimate is always less than or equal to the actual cost of reaching the goal. Note that if you include the blank tile in the estimate, this heuristic is no longer admissible (think about the case where only 1 tile is out of place).*
3 Pancake Heuristics (⋆)

Here, we consider the pancake problem. A server is given a stack of \( n \) pancakes. Each pancake is a different size. The server can flip the top \( k \) pancakes, reversing their order. The cost of flipping \( k \) pancakes is \( k \). The server’s goal is to order the pancakes from smallest (top) to largest (bottom), with minimal cost. More formally, the search states are all permutations \( \sigma \) of \( (1, 2, 3, \ldots, n) \), and the goal is \( (1, 2, 3, \ldots, n) \). The successor function gives the outcome of flips, for example:

<table>
<thead>
<tr>
<th>action</th>
<th>cost</th>
<th>successor state</th>
</tr>
</thead>
<tbody>
<tr>
<td>flip 2</td>
<td>2</td>
<td>(4,3,1,2,5)</td>
</tr>
<tr>
<td>flip 3</td>
<td>3</td>
<td>(1,4,3,2,5)</td>
</tr>
<tr>
<td>flip 4</td>
<td>4</td>
<td>(2,1,4,3,5)</td>
</tr>
<tr>
<td>flip 5</td>
<td>5</td>
<td>(5,2,1,4,3)</td>
</tr>
</tbody>
</table>

Here are three heuristics for the pancake problem:

1. \( H_1 \), The largest pancake that is out of place: largest \( i \) such that \( i \neq \sigma_i \)
2. \( H_2 \), The number of pancakes out of position: count of all \( i \) such that \( i \neq \sigma_i \)
3. \( H_3 \), One less than the size of the pancake at the top of the stack: \( \sigma_1 - 1 \)

a) Circle all of the following heuristics that are admissible:

i.) \( H_1 \)  
ii.) \( H_2 \)  
iii.) \( H_1 + H_2 \)  
iv.) \( H_2 + H_3 \)  
v.) \( \text{max}(H_1, H_2, H_3) \)

- \( H_1 \) is admissible because putting pancake \( i \) into place has cost at least \( i \).
- \( H_2 \) is admissible because putting \( k \) pancakes into position requires at least a flip of size \( k \).
- \( H_3 \) is admissible. If \( \sigma_1 = 1 \), then the heuristic is 0. Otherwise, A flip of size \( \sigma_1 \) is required to move the top pancake into position.
- \( H_1 + H_2 \) is not admissible: \( s=(3 \ 2 \ 1) \) is 3 away from the goal, but \( H_1(s) + H_2(s) = 5 \)
- \( H_2 + H_3 \) is not admissible: \( s=(3 \ 2 \ 1) \) is 3 away from the goal, but \( H_2(s) + H_3(s) = 4 \)
- \( \text{max}(H_1, H_2, H_3) \) is the max of admissible heuristics, so it too is admissible.

b) A heuristic \( H_A \) dominates a heuristic \( H_B \) if \( H_A(n) \geq H_B(n) \) for every state. Circle all of the following statements that are true:

i. \( H_1 \) dominates \( H_2 \). True: If \( k \) pancakes are out of place, at least one of them is size \( k \) or greater.
ii. \( H_1 \) dominates \( H_3 \). True: If \( \sigma_1 > 1 \), then at least the \( \sigma_1 \) pancake is out of place.
iii. \( H_2 \) dominates \( H_1 \). False: A counterexample is \( (5 \ 2 \ 3 \ 4 \ 1) \).
c) Circle all of the following heuristics that are consistent:

i.) \( H_1 \)  

ii.) \( H_2 \)  

iii.) \( H_3 \)

Consider a state \( s \) and its successor \( s' \)

- \( H_1 \) is consistent: \( H_1(s) - H_1(s') > 0 \) only if the pancake \( H_1(s) \) was moved into correct position, which requires cost at least \( H_1(s) \).
- \( H_2 \) is consistent: \( H_2(s) - H_2(s') \) is bounded by the number of pancakes that were flipped, which is the cost of the flip.
- \( H_3 \) is not consistent: \( s = (5 1 2 3 4), s' = (1 5 2 3 4), H_3(s) - H_3(s') = 4 \), but cost from \( s \) to \( s' \) is 2.

4 Conformant Search

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 \([W,N,E,E,N,E,E,E]\), 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) 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) 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.