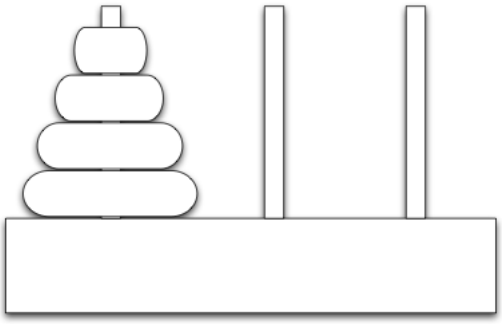
Section 1: Search

1 Towers of Hanoi



The Towers of Hanoi is a famous problem for studying recursion in computer science and recurrence equations in discrete mathematics. We start with N discs of varying sizes on a peg (stacked in order according to size), and two empty pegs. We are allowed to move a disc from one peg to another, but we are never allowed to move a larger disc on top of a smaller disc. The goal is to move all the discs to the rightmost peg (see figure).

In this problem, we will formulate the Towers of Hanoi as a search problem.

(a) Propose a state representation for the problem

One possible state representation would be to store three lists, corresponding to which discs are on which peg. If we assume that the N discs are numbered in order of increasing size $1, ..., n$, then we can represent each peg as an ordered list of integers corresponding to which discs are on that peg.

(b) What is the size of the state space? If there are k pegs and n disks, then the size of the state space is k^n. For this setup, the size is 3^N.

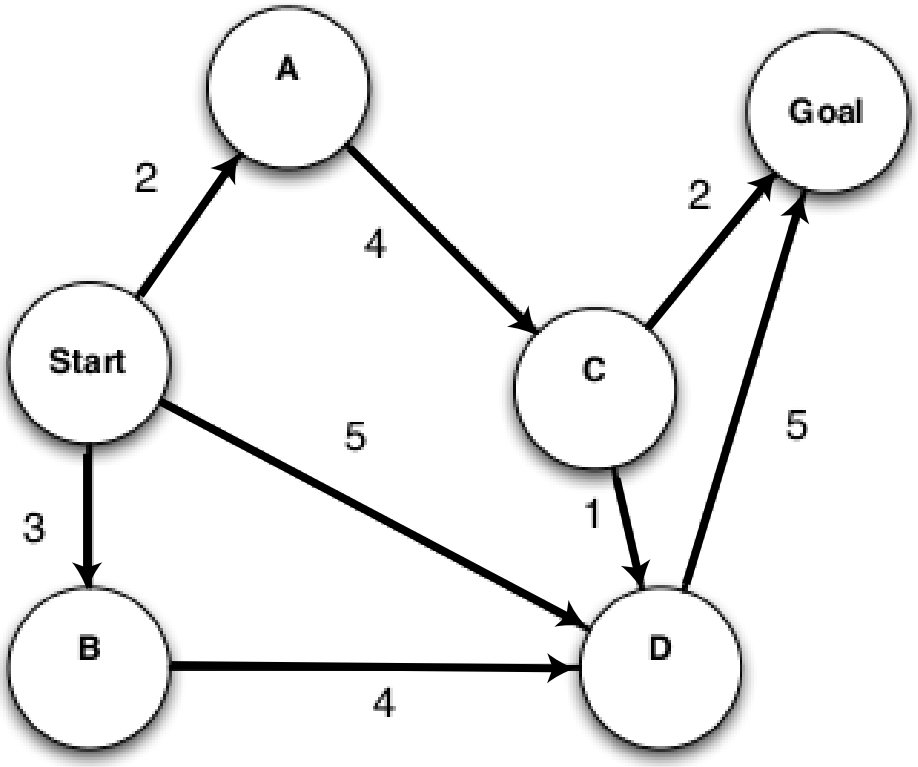
(c) What is the start state? $([1, ..., n], [], [])$

(d) From a given state, what actions are legal?

We can pop the first integer from any list (i.e., peg) and push it onto the front of another list (peg), so long as it is smaller than the integer currently at the front of the list being pushed to (i.e., peg being moved to).

(e) What is the goal test? Is the state the same as $([], [], [1, ..., n])$?

2 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. Remember that in graph search, a state is expanded only once.

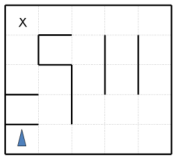
a) Depth-first search.
States Expanded: Start, A, C, D, Goal
Path Returned: Start-A-C-D-Goal

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

3 Search and Heuristics

Imagine a car-like agent wishes to exit a maze like the one shown below:



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. The turning actions are *left* and *right*, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero). The moving actions are *fast* and *slow*. *Fast* increments the velocity by 1 and *slow* decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its NEW adjusted velocity (see example below). A consequence of this formulation is that it is impossible for the agent to move in the same nonzero velocity for two consecutive timesteps. Any action that would result in a collision with a wall crashes the agent and is illegal. Any action that would reduce v below 0 or above a maximum speed $V_{\max}$ is also illegal. The agent's goal is to find a plan which parks it (stationary) on the exit square using as few actions (time steps) as possible.

As an example: if at timestep t the agent's current velocity is 2, by taking the *fast* action, the agent first increases the velocity to 3 and move 3 squares forward, such that at timestep $t+1$ the agent's current velocity will be 3 and will be 3 squares away from where it was at timestep t. If instead the agent takes the *slow* action, it first decreases its velocity to 1 and then moves 1 square forward, such that at timestep $t+1$ the agent's current velocity will be 1 and will be 1 squares away from where it was at timestep t. If, with an instantaneous velocity of 1 at timestep $t+1$, it takes the slow action again, the agent's current velocity will become 0, and it will not move at timestep $t+1$. Then at timestep $t+2$, it will be free to turn if it wishes. Note that the agent could not have turned at timestep $t+1$ when it had a current velocity of 1, because it has to be stationary to turn.

1. If the grid is M by N, what is the size of the state space? Justify your answer. You should assume that all configurations are reachable from the start state.

   The size of the state space is $4MN(V_{max}+1)$. The state representation is (direction facing, x, y, speed). Note that the speed can take any value in $\{0, ..., V_{max}\}$.

2. Is the Manhattan distance from the agent's location to the exit's location admissible? Why or why not?

   No, Manhattan distance is not an admissible heuristic. The agent can move at an average speed of greater than 1 (by first speeding up to V_{max} and then slowing down to 0 as it reaches the goal), and so can reach the goal in less time steps than there are squares between it and the goal. A specific example: A timestep 0, the agent's starts stationary at square 0 and the target is 9 squares away at square 9. At timestep 0, the agent takes the *fast* action and ends up at square 1 with a velocity of 1. At timestep 1, the agent takes the *fast* action and ends up at square 3 with a velocity of 2. At timestep 2, the agent takes the *fast* action and ends up at square 6 with a velocity of 3. At timestep 3, the agent takes the *slow* action and ends up at square 8 with a velocity of 2. At timestep 4, the agent takes the *slow* action and ends up at square 9 with a velocity of 1. At timestep 5, the agent takes the *slow* action and stays at square 9 with a velocity of 0. Therefore, the agent can move 9 squares by taking 6 actions.

3. State and justify a non-trivial admissible heuristic for this problem which is not the Manhattan distance to the exit.

   There are many answers to this question. Here are a few, in order of weakest to strongest:

   (a) The number of turns required for the agent to face the goal.

   (b) Consider a relaxation of the problem where there are no walls, the agent can turn, change speed arbitrarily, and maintain constant velocity. In this relaxed problem, the agent would move with V_{max}, and then suddenly stop at the goal, thus taking $d_{manhattan}/V_{max}$ time.

   (c) We can improve the above relaxation by accounting for the acceleration and deceleration dynamics. In this case the agent will have to accelerate from 0 from the start state, maintain a constant velocity of V_max, and slow down to 0 when it is about to reach the goal. Note that this heuristic will always return a greater value than the previous one, but is still not an overestimate of the true cost to reach the goal. We can say that this heuristic *dominates* the previous one.

   In particular, let us assume that $d_{manhattan}$ is greater than and equal to the distance it takes to accelerate to and decelerate from V_{max} (In the case that $d_{manhattan}$ is smaller than this distance, we can still use $d_{manhattan}/V_{max}$ as a heuristic). We can break up the $d_{manhattan}$ into three parts: d_{accel}, $d_{V_{max}}$, and d_{decel}. The agent travels a distance of d_{accel} when it accelerates from 0 to V_{max} velocity. The agent travels a distance of d_{decel} when it decelerates from V_{max} to 0 velocity. In between acceleration and deceleration, the agent travels a distance of $d_{V_{max}} = d_{manhattan} - d_{accel} - d_{decel}$. $d_{accel} = 1 + 2 + 3 + V_{max} = \frac{(V_{max})(V_{max}+1)}{2}$ and $d_{decel} = (V_{max} - 1) + (V_{max} - 2) + ... + 1 + 0 = \frac{(V_{max})(V_{max}-1)}{2}$. So $d_{V_{max}} = d_{manhattan} - \frac{(V_{max})(V_{max}+1)}{2} - \frac{(V_{max})(V_{max}-1)}{2} = d_{manhattan} - V_{max}^2$. It takes V_{max} steps to travel the initial d_{accel}, $\frac{d_{manhattan}-V_{max}^2}{V_{max}}$ steps to travel the middle $d_{V_{max}}$ and V_{max} steps to travel the last d_{decel}. Therefore, our heuristic is

$$\begin{cases} \frac{d_{manhattan}}{V_{max}}, & \text{if } d_{manhattan} \leq V_{max}^2 = d_{accel} + d_{decel} \\ \frac{d_{manhattan}}{V_{max}} + V_{max}, & \text{if } d_{manhattan} > V_{max}^2 = d_{accel} + d_{decel} \end{cases}$$

4. If we used an inadmissible heuristic in A* graph search, would the search be complete? Would it be optimal?

   If the heuristic function is bounded, then A* graph search would visit all the nodes eventually, and would find a path to the goal state if there exists one. An inadmissible heuristic does not guarantee optimality as it can make the good optimal goal look as though it is very far off, and take you to a suboptimal goal.

5. If we used an *admissible* heuristic in A* graph search, is it guaranteed to return an optimal solution? What if the heuristic was consistent? What if we were using A* tree search instead of A* graph search?

   Although admissible heuristics guarantee optimality for A* *tree* search, they do not necessarily guarantee optimality for A* *graph* search; they are only guaranteed to return an optimal solution if they are consistent as well.

6. Give a general advantage that an inadmissible heuristic might have over an admissible one.

   The time to solve an A* search problem is a function of two factors: the number of nodes expanded, and the time spent per node. An inadmissible heuristic may be faster to compute, leading to a solution that is obtained faster due to less time spent per node. It can also be a closer estimate to the actual cost function (even though at times it will overestimate!), thus expanding less nodes. We lose the guarantee of optimality by using an inadmissible heuristic. But sometimes we may be okay with finding a suboptimal solution to a search problem.