CSC242: Homework 1-1
AIMA Chapter 3.0-3.3.1

1. Identify the components of a well-defined state-space search problem (both a "domain" of problems, and a specific instance from the domain). What is a solution to an instance of such a problem?

   ANSWER:

   A problem domain (or class of problems) is defined by:

   - A set of **states** S
   - A set of **actions** A
   - A **transition model**, consisting of
     - An **applicability function**, ACTIONS(s), which identifies the actions (subset of A) that can be performed in a state s.
     - A **result function** RESULT(s, a) which identifies the state (from S) resulting from performing action a in state s.
   - A **path cost** function on sequences of actions (often implemented as the sum of the individual **step costs**).

   An instance of a problem from a domain is given by also specifying:

   - The **initial state** s_0
   - The set of **goal states** G

   A solution to a state-space search problem is a sequence of applicable actions that leads from the initial state to the goal state:

   $$a_1, \ldots, a_n \quad \text{s.t.} \quad \text{RESULT}(\cdots \text{RESULT}(\text{RESULT}(s_0, a_1), a_2), \cdots, a_n) \in G$$

   See AIMA, starting bottom of page 66.

2. Formulate the 15-puzzle as a state-space search problem. Give a formal model as required for state-space search.

   ANSWER:

   An informal formulation of the 8-puzzle in words is given in AIMA on page 71.

   However a formal model requires more symbols and far fewer words. The following is one possible formal model of an $n \times n$ puzzle (so $n = 3$ for the 8-puzzle and $n = 4$ for the 15-puzzle):

- State: an $n \times n$ array containing the numbers from 0 to $n^2 - 1$, with 0 representing the blank. That is:

$$s = \begin{bmatrix} t_{0,0} & t_{1,0} & \cdots & t_{n-1,0} \\ t_{0,1} & t_{1,1} & \cdots & t_{n-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ t_{0,n-1} & t_{1,n-1} & \cdots & t_{n-1,n-1} \end{bmatrix}$$

  where each $t_{i,j}$ is a unique element of $\{0, \ldots, n^2 - 1\}$.
- Actions are moving the blank: $\{N, S, E, W\}$.
- Applicability: Consider action a and state $s = [t_{i,j}]$.
  Let r, c be numbers such that $t_{r,c} = 0$ (that is, r and c are the row and column of the blank).
  Then:

$$\begin{aligned} N &\in \textsc{Actions}(s) \quad \text{if } r > 0 \\ S &\in \textsc{Actions}(s) \quad \text{if } r < n - 1 \\ E &\in \textsc{Actions}(s) \quad \text{if } c < n - 1 \\ W &\in \textsc{Actions}(s) \quad \text{if } c > 0 \end{aligned}$$

- Result: Consider action a and state $s = [t_{i,j}]$. Let r, c be as above (row and column of the blank). Then

$$\textsc{Result}(s, a) = \begin{cases} \left.\begin{array}{l} t_{r,c} \leftarrow t_{r-1,c} \\ t_{r-1,c} \leftarrow 0 \end{array}\right\} \text{if } a = N \\ \left.\begin{array}{l} t_{r,c} \leftarrow t_{r+1,c} \\ t_{r+1,c} \leftarrow 0 \end{array}\right\} \text{if } a = S \\ \left.\begin{array}{l} t_{r,c} \leftarrow t_{r,c+1} \\ t_{r,c+1} \leftarrow 0 \end{array}\right\} \text{if } a = E \\ \left.\begin{array}{l} t_{r,c} \leftarrow t_{r,c-1} \\ t_{r,c-1} \leftarrow 0 \end{array}\right\} \text{if } a = W \end{cases}$$

- Cost: constant per action means least expensive solution is fewest actions (moves)
- Initial state: Any permutation of $\{0, \ldots, n^2 - 1\}$ into $[t_{i,j}]$.

- Goal state:

$$\begin{bmatrix} 1 & 2 & \cdots & n \\ n+1 & n+2 & \cdots & 2n \\ \vdots & \vdots & \ddots & \vdots \\ n^2-n+1 & n^2-n+2 & \cdots & 0 \end{bmatrix}$$

The advantages of the formal description are that (1) you can prove things using it; and (2) it's very close to being code for an implementation.

3. The CIA makes a lot of maps. To make it easy for the analysts, the maps are colored so that no two adjacent regions have the same color. Unfortunately, due to budget cuts, they only have four colors of ink in their printers. Help the CIA by formulating this as a state-space search problem.

ANSWER:

Informal description:

- States: Sets of assignments of colors to regions.
- Actions: Assigning a color to a region.
- Applicability: Can only assign a color not already used by an adjacent region.
- Result: Update state with new assignment.
- Initial state: No regions colored.
- Goal test: All regions colored (and no two adjacent regions have the same color).
- Cost function: Number of assignments (doesn't really matter since all colorings will have the same number of assignments).

Formal definition:

- Assume that the map is represented as a undirected graph $\langle V, E \rangle$ where there is one vertex for each region and an edge between two vertices if the corresponding regions are adjacent. Let $Colors$ be the set of possible colors.
- A state is an assignment of colors to regions (no more than one color per region):

$$\{\langle v_i, c_i \rangle \mid v_i \in V, c_i \in Colors, v_i \text{ unique}\}$$

- An action is to assign a color to a region: $\langle v_i, c_i \rangle$.
- Applicability: Action $a = \langle v_i, c_i \rangle$ is applicable in state $s = \{\langle v_j, c_j \rangle\}$ if there is no $\langle v_j, c_i \rangle \in s$ such that $\langle v_i, v_j \rangle \in E$ (that is, no v_j adjacent to v_i is already assigned c_i in s).

- Result: Let action $a = \langle v_i, c_i \rangle$ and state $s = \{\langle v_j, c_j \rangle\}$. Then

$$\text{RESULT}(s, a) = s \cup \{\langle v_i, c_i \rangle\}$$

- Cost: doesn't matter since all solutions will have the same number of assignments
- Initial state: $s = \{\ \} = \varnothing$
- Goal state: $s = \{\langle v_j, c_j \rangle\}$ such that $|s| = |V|$ (that is, all regions are assigned a color). Note that this is easy to specify if the adjacency constraints are enforced by the applicability function.

Again, the code almost writes itself, given the formal definition.

4. A 3-foot-tall monkey is in a room with where some bananas are suspended from the 8-foot ceiling. The monkey would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates. Help the monkey by formulating this as a state-space search problem.

   ANSWER:

   Informal description:

   - States: 3D location of monkey, bananas, and boxes; whether the monkey has the bananas or not
   - Actions: monkey walks from one location to another, monkey pushes crate from one location to another, monkey stacks one crate on the other, monkey climbs up a crate (could have climbs down also), monkey grabs bananas from ceiling.
   - Applicability: Easier to specify formally, but you could say it in words.
   - Result: Ditto.
   - Cost function: Number of actions (*i.e.*, prefer shorter solutions).
   - Initial state: As described in the text.
   - Goal test: Monkey has bananas.

   Formal definition:

   - States: $\langle L_m, L_b, L_{c_1}, L_{c_2}, B \rangle$ where:

$$\begin{aligned} L_m &= \text{Location of monkey} = \langle x_m, y_m, z_m \rangle \\ L_b &= \text{Location of bananas} = \langle x_b, y_b, z_b \rangle \\ L_{c_1} &= \text{Location of crate 1} = \langle x_{c_1}, y_{c_1}, z_{c_1} \rangle \\ L_{c_2} &= \text{Location of crate 2} = \langle x_{c_2}, y_{c_2}, z_{c_1} \rangle \\ B &\in \{\textit{true}, \textit{false}\} = \text{Monkey has bananas?} \end{aligned}$$

- Actions:

$walk(x, y)$:	Monkey walks to x, y
$push(c, x, y)$:	Monkey pushes crate c to x, y
$stack(c_i, c_j)$:	Monkey stacks crate c_i on top of crate c_j
$climb(c)$:	Monkey climbs crate c
$grab$:	Monkey grabs bananas

- Applicability: In state $\langle L_m, L_b, L_{c_1}, L_{c_2}, B\rangle$:

$walk(x, y)$:	if $z_m = 0$
$push(c, x, y)$:	if $L_m = L_c$ and $z_m = z_c = 0$
$stack(c_i, c_j)$:	if $L_{c_i} = L_{c_j} = L_m$
$climb(c)$:	if $L_m = L_c$
$grab$:	if $x_m = x_b$ and $y_m = y_b$ and $z_m \geq z_b - 3$

  I may be ignoring a few details, such as whether you need to be at the same location as a crate to climb it, or whether you need to be adjacent (enough) to it. But this close enough and those *could* be expressed using the formalization.

- Result: In state $\langle L_m, L_b, L_{c_1}, L_{c_2}, B\rangle$:

$walk(x, y)$:	$L_m \leftarrow \langle x, y, 0\rangle$
$push(c, x, y)$:	$L_c, L_m \leftarrow \langle x, y, 0\rangle$
$stack(c_i, c_j)$:	$z_{c_i} \leftarrow z_{c_j} + 3$
$climb(c)$:	$z_m \leftarrow z_c + 3$
$grab$:	$B \leftarrow true$

- Cost: Nothing in the description describes the costs of the actions, so don't worry about it, assume constant, or do something exciting involving the distance travelled or the work to stack or climb.
- Initial state: $\langle L_m, L_b, L_{c_1}, L_{c_2}, B\rangle$ such that

$$\begin{aligned} L_m &= \langle x_m, y_m, 0\rangle \\ L_b &= \langle x_b, y_b, 8\rangle \\ L_{c_1} &= \langle x_{c_1}, y_{c_1}, 0\rangle \\ L_{c_2} &= \langle x_{c_2}, y_{c_2}, 0\rangle \\ B &= \textit{false} \end{aligned}$$

- Goal state: Any $s = \langle L_m, L_b, L_{c_1}, L_{c_2}, B \rangle$ such that $B = true$.

FYI: It took me a few iterations to get the state representation right, but then the rest of the formalization is straightforward and could easily be turned into code.

5. Beer is being served at a party in Wegmans Hall. The keg is delivered with a 12-cup measure, an 8-cup measure, and a 3-cup measure. The bartender can fill a measure completely from the keg, empty a measure completely down the drain, or pour the contents of one measure into another measure (as much as will fit without spilling). Brynn orders a single cup of beer. Help the bartender by formulating this as a state-space search problem.

   ANSWER:

   - States: $\langle x_1, x_2, x_3 \rangle$, where x_1 is the amount of beer in the 12-cup measure, x_2 is the amount in the 8-cup measure, and x_3 is the amount in the 3-cup measure.
   - Actions: *fill*(i), *empty*(i), *pour*(i, j) for $1 \leq i, j \leq 3$.
   - Applicability: All actions are always applicable, although it might be foolish to pour from or empty an empty measure, so you could put in those conditions if you wanted.
   - Result of performing an action in in state $s = \langle x_1, x_2, x_3 \rangle$:

$$
\begin{aligned}
\mathit{fill}(1): &\quad \langle 12, x_2, x_3 \rangle \\
\mathit{fill}(2): &\quad \langle x_1, 8, x_3 \rangle \\
\mathit{fill}(3): &\quad \langle x_2, x_2, 3 \rangle \\
\mathit{empty}(1): &\quad \langle 0, x_2, x_3 \rangle \\
\mathit{empty}(2): &\quad \langle x_1, 0, x_3 \rangle \\
\mathit{empty}(3): &\quad \langle x_2, x_2, 0 \rangle
\end{aligned}
$$

   For action $pour(i, j)$: If measure i has capacity c_i and measure j has capacity c_j, then the amount transferred is the entire amount in the source if there is room, or as much as fits in the destination. In other words: $\delta x = \max(x_i, c_j - x_j)$. So in the successor state, x_i is reduced by δx and x_j is incremented by δx. You might write this as:

$$
\begin{aligned}
x_i &\leftarrow x_i - \delta x \\
x_j &\leftarrow x_j + \delta x
\end{aligned}
$$

   Or in a program:

```
x[i] -= dx;
x[j] += dx;
```

- Initial state: All measures empty $\langle 0, 0, 0 \rangle$ (or whatever).
- Goal state: Any $s = \langle x_1, x_2, x_3 \rangle$ such that $x_i = 1$ for at least one i.
- Cost function: Number of actions.

This is an example of a nice, crisp, mathematical formalization of the successor function. It also illustrates nicely how one might implement this problem (preferably without hard-wiring the number of measures).

6. Compare the `treeSearch` and `graphSearch` functions shown in AIMA Figure3.7. What are the pros and cons of `graphSearch`?

   ANSWER:

   The `graphSearch` algorithm uses an "explored list" or "closed set" to remember the states it has already seen. This allows it to avoid repeated states (also known as "loopy" or "redundant" states). Following redundant paths can cause a tractable problem to become intractable. "Algorithms that forget their history are doomed to repeat it."

   The downside of `graphSearch` is the space to store the explored list and the time to search and update it. These are not insignificant, as we shall see.

   See AIMA pp. 75–78.