Lecture 2

Searching – Problem Formulation

2.1 Solving Problems by Searching

To solve problem, we first try to **formulate** the problem or find the model of the problem. Then, we find the solutions from the model. Basically, each obtained solution is a solution in terms of the model. We have more confidence in the solution when our model accurately represents the problem.

$$\begin{tikzcd}
\text{problem} \arrow[r] & \boxed{\textit{formulate}} \arrow[r] & \text{model} \arrow[r] & \boxed{\textit{solve}} \arrow[r] & \text{solution}
\end{tikzcd}$$

Example 2.1 A rectangle has an area of 56 square centimeters. Its height is 5 centimeters longer than its width. What is the dimension of this rectangle?

Let x be the width. Then, the height is $x + 5$. We set up a quadratic equation

$$x(x+5) = 56$$
$$x^2 + 5x - 56 = 0$$

Solving the equation provides the value of x.

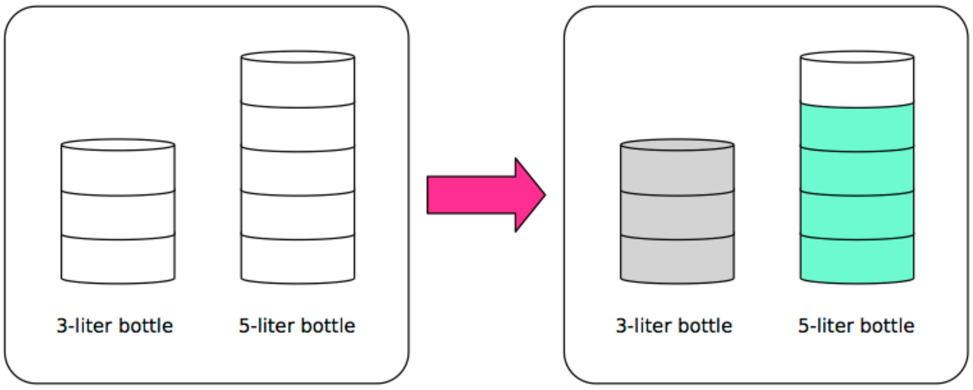
2.2 Problem Solving Agents

Given a model of a problem, a problem solving agent finds **a sequence of** (3.1)
actions that lead from an initial state to the goal. The sequence is also called a **solution** which can be **executed** later by the agent.

Searching is a basic technique to find a solution of problem. It is performed by trying all possible actions iteratively in order to get to the goal.

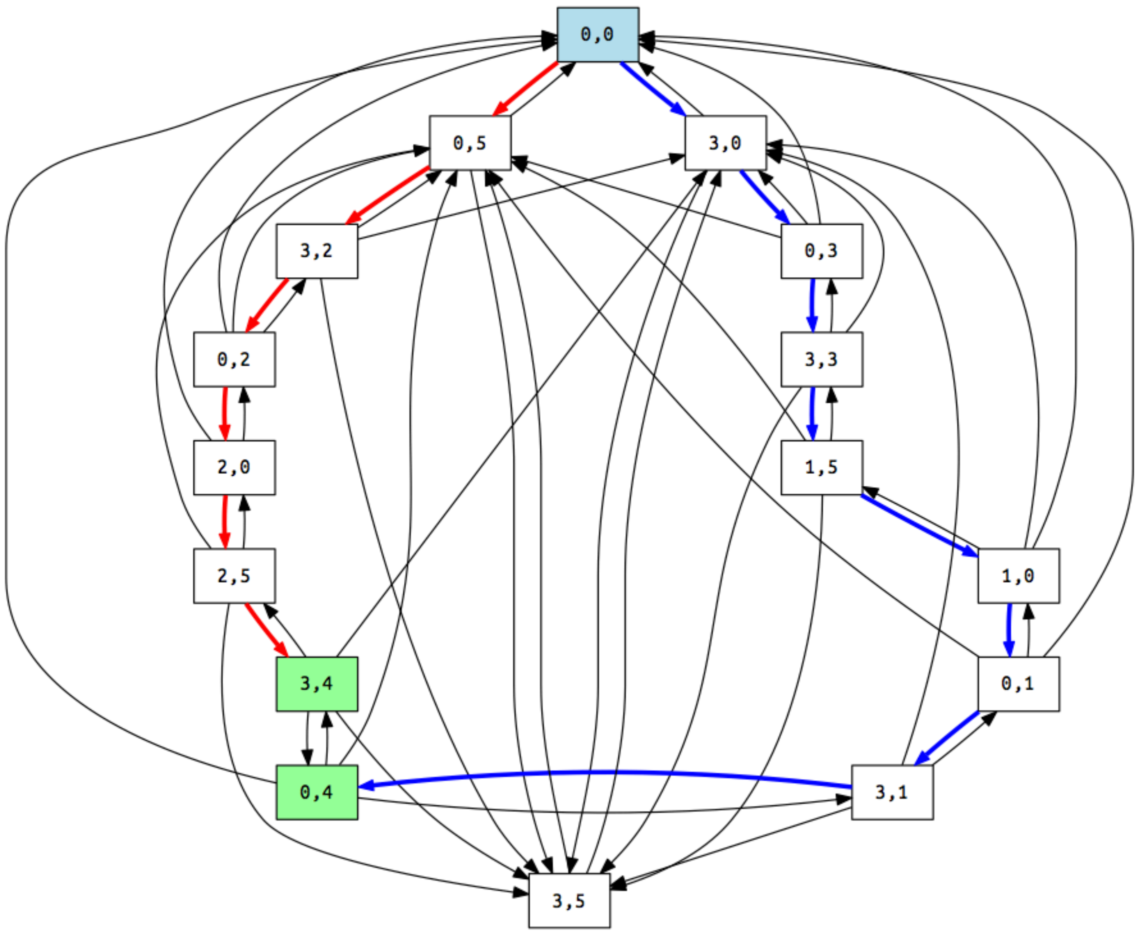
$$\begin{tikzcd}
\textit{model} \arrow[r] & \boxed{\text{Problem-Solving Agent}} \arrow[r] & \textit{sequence of actions}
\end{tikzcd}$$

Example 2.2 We has a 3-liter bottle, a 5-liter bottle, as well as a water faucet. How can we have 4 liters of water in the 5-liter bottle?



To solve this problem, we define a state representing the amount of water in both bottles. Each state composes of two values i.e. the amount of water in the 3-liter bottle, and the amount of water in the 5-liter bottle. For example, we can write `[3,0]` when the 3-liter bottle is full, and the 5-liter is empty.

We can draw a graph showing the changes of amount of water after applying actions.



2.2.1 Problem Formulation

Here are what we need to determine as the *model* of a search problem.

State represents a situation of problem. A state contains all necessary information to identify a situation of problem. A state is transformed to another state by applying an action.

Initial state is the state that the system starts in.

Goal test is the condition to determine whether a given state is a goal state. In real-world problems, more than one state can be considered a goal.

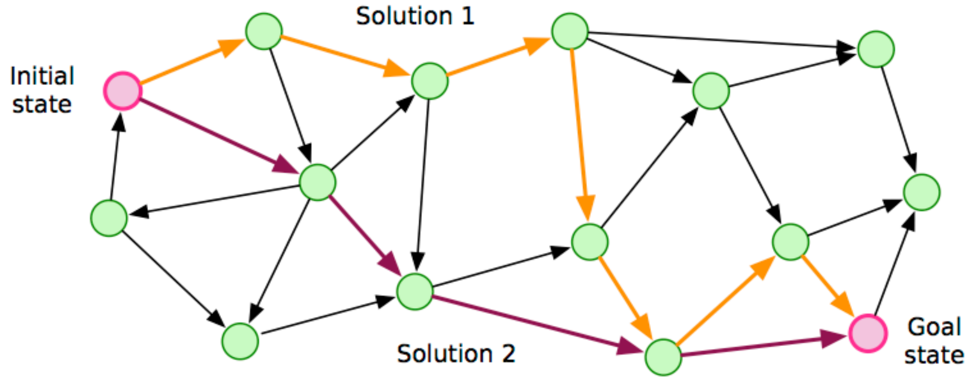
Actions are a set of transitions between states. Different states may have different sets of possible actions.

A **successor function** takes a state and returns a set of its possible actions.

If we connect all possible states with the actions, it becomes a graph called '**state space**'. However, it is usually difficult to *explicitly* define the state space in the real-world problems.

Path cost represents the cost of solutions since actions may cost differently. The path cost is sometimes computed from the sum of the **step costs** along the path.

$$PathCost = \sum_i StepCost_i$$



Example 2.3 Formulate the water-measuring problem. Here, we have a 3-liter and 5-liter bottles. We want to measure 4 liters of water.

State a tuple `[`x`,`y`]` where x, y show the amount of water in 3- and 5-liter bottles respectively.

Initial state a state `[0,0]`

Goal test check the amount of the 5-liter bottle is 4, or any state `[`x`,4]` where $0 \leq x \leq 3$.

Actions

- Empty one of the bottles.
- Fill up one of the bottles.
- Pour water from one bottle to the other bottle.

Successor function given a state `[`x`,`y`]`, the successors generate a set of states as follow:

- Empty
  - if $x > 0$, generate `[0,`y`]`.
  - if $y > 0$, generate `[`x`,0]`.
- Fill up
  - if $x < 3$, generate `[3,`y`]`.
  - if $y < 5$, generate `[`x`,5]`.
- Pour water from one bottle to another
  - if $x < 3$ and $y \geq a$, generate `[3,`$y - a$`]` where $a = 3 - x$.
  - if $x < 3$ and $y < a$, generate `[`$x + y$`,0]` where $a = 3 - x$.
  - if $y < 5$ and $x \geq b$, generate `[`$x - b$`,5]` where $b = 5 - y$.
  - if $y < 5$ and $x < b$, generate `[0,`$y + x$`]` where $b = 5 - y$.

Step cost amount of water changed.

Exercise 2.1 Write a list of successors of a state `[3,1]`.

2.2.2 Implementing the Problem Formulation

From the formulation, we can implement the state representation, the goal test, and the successor function. This is the first step to solve a problem by searching.

Example 2.4 Implementation of the water measuring problem.

```
class water:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return '[%d, %d]' % (self.x, self.y)

    def __repr__(self):
        return str(self)

    def isgoal(self):
        if self.y == 4:
            return True
        else:
            return False

    def successors(self):
        if self.x > 0:
            yield (water(0, self.y), "Empty {3}", self.x)
        if self.y > 0:
            yield (water(self.x, 0), "Empty {5}", self.y)

        if self.x < 3:
            yield (water(3, self.y), "Fill up {3}", 3 - self.x)
        if self.y < 5:
            yield (water(self.x, 5), "Fill up {5}", 5 - self.y)

        a = 3 - self.x
        b = 5 - self.y

        if self.x > 0 and self.x < 3:
            if self.y >= a:
                yield (water(3, self.y-a), "Pour {5} -> {3}", a)
            else:
                yield (water(self.x+self.y, 0), "Pour {5} -> {3}", self.y)
        if self.x > 0 and self.y < 5:
            if self.x >= b:
                yield (water(self.x-b, 5), "Pour {3} -> {5}", b)
            else:
                yield (water(0, self.x+self.y), "Pour {3} -> {5}", self.x)
```

We can then simply generate a list of successors.

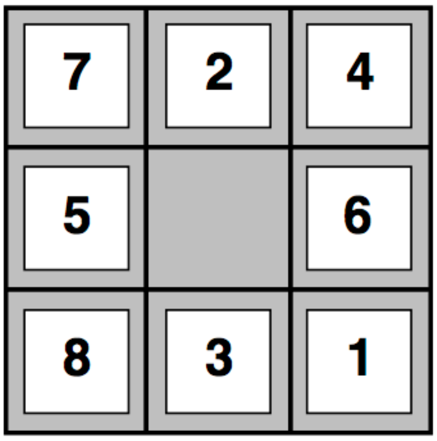
```
from water import water

s = water(0, 0)
print("isgoal? ", s.isgoal())
for l in s.successors():
    print(l)
print('-----')
s = water(3, 0)
for l in s.successors():
    print(l)
```

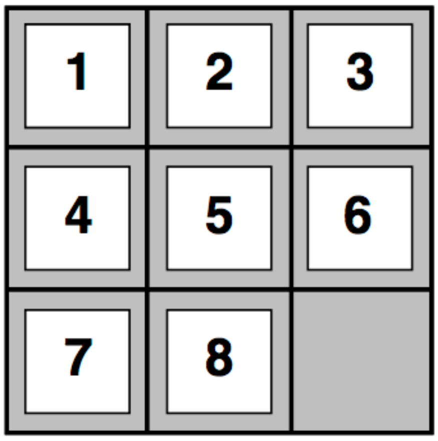
Output

```
([3, 0], 'Fill up {3}', 3)
([0, 5], 'Fill up {5}', 5)
-----
([0, 0], 'Empty {3}', 3)
([3, 5], 'Fill up {5}', 5)
([0, 3], 'Pour {3} -> {5}', 3)
```

Exercise 2.2 8-puzzle is a sliding puzzle with the objective to order the tiles in order by making sliding moves. If we formulate this problem as a search problem, explain how to represent a *state* and the possible *actions*.

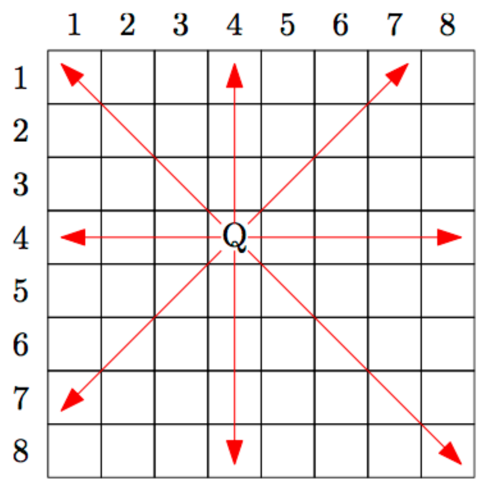


Start State



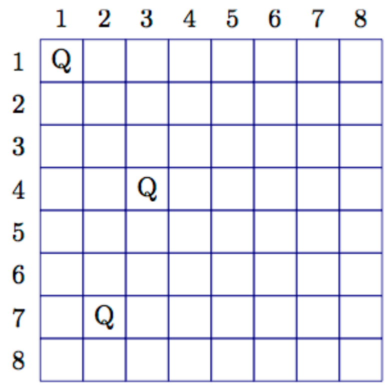
Goal State

Exercise 2.3 8-queens is a puzzle on placing eight queens on an 8×8 chess board. Explain how to represent a *state*, the *goal test*, and the *actions*.

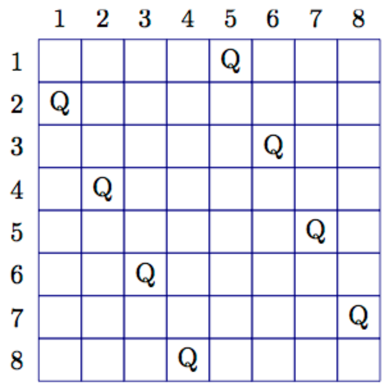


Example 2.5 There are many ways to formulate the 8-queens problem. Here is another formalation.

State an 8-tuple representing the row number that a queen is placed in each column; 0 means no queen in that column; no attacking between any pair of queens is allowed. For example,



[1, 7, 4, 0, 0, 0, 0, 0]



[2, 4, 6, 8, 1, 3, 5, 7]

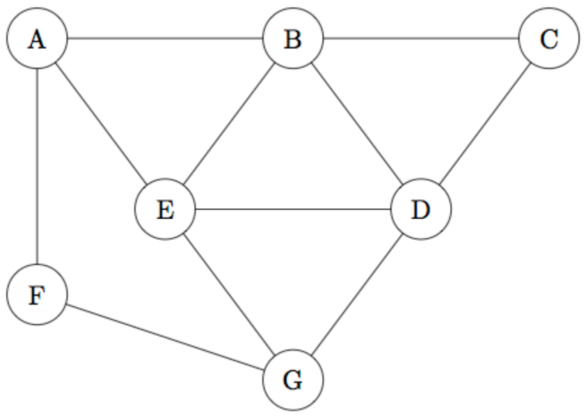
Initial state an empty chess board i.e. `[0,0,0,0,0,0,0,0]`

Goal test a board with 8 queens on the board

Actions add a queen to any row in the leftmost empty column. This new queen must not attacked any other queen.

Any formulations can be used to find a solution for the 8-queen problem. In the 1st formulation, there are $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57 \approx 1.8 \times 10^{14}$ possible states. In the 2nd formulation, there are only 2,057 states.

Exercise 2.4 From an undirected graph shown below, we want to color the nodes of the graph using four colors, i.e. red, green, blue, and yellow. Formulate this graph-coloring problem as a search problem.



2.3 Searching for Solutions

Problem-solving agents find a solution by conducting search. Here is the search algorithm.

1. Start from the initial state
2. Apply all possible actions to the state, generate the set of successors, and keep them in a data structure.
3. Choose one of the successors out of the data structure.
4. Iteratively follow step 2 and 3 until the goal test is satisfied.
5. Trace back to obtain the sequence of actions from the goal state to the initial state

References

Russell, S. and Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd edition). Pearson/Prentice Hall.
Michalewicz, F. and Fogel, D. B. (1998). How to Solve It: Modern Heuristics. Springer.