# Search in Python

Chapter 3

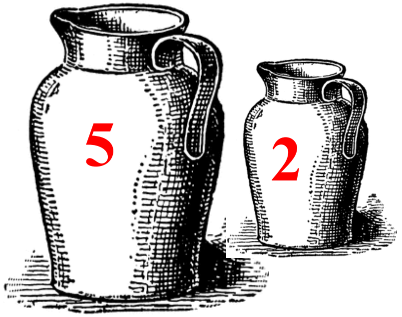
# Today's topics

- AIMA Python code
- What it does
- How to use it
- Worked example: water jug program

# Install AIMA Python ?

- Aimacode is a great github repo of python code linked to the AIMA book
- It's not available for pip installing ☹
  - Pete Norvig's recommendation
- One workaround is to:
  - Clone the repo on your computer and follow the instructions in its readme file
  - Add the directory to your PYTHONPATH environment variable

# Two Water Jugs Problem

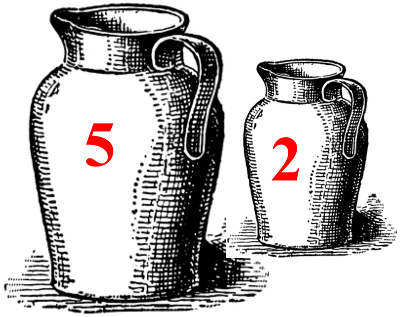



- Given two water jugs, J1 and J2, with capacities C1 and C2 and initial amounts W1 and W2, find actions to end up with amounts W1' and W2' in the jugs
- Example problem:
  - We have a 5 gallon and a 2 gallon jug
  - Initially both are full
  - We want to end up with exactly one gallon in J2 and don't care how much is in J1

# search.py

- Defines a *Problem* class for a search problem
- Has functions to perform various kinds of search given an instance of a Problem, e.g., breadth first, depth first, hill climbing, A*, …
- *InstrumentedProblem* subclasses *Problem* and is used with *compare_searchers* for evaluation
- To use for WJP: (1) decide how to represent the WJP, (2) define *WJP* as a subclass of *Problem* and (3) provide methods to (a) create a WJP instance, (b) compute successors and (c) test for a goal

# Example: Water Jug Problem




Given full 5-gal. jug and empty 2-gal. jug, fill 2-gal jug with one gallon

- State = (x,y), where x is water in jug 1; y is water in jug 2
- Initial State = (5,0)
- Goal State = (-1,1), where -1 means any amount

Action table

| Name | Cond. | Transition | Effect |
|---|---|---|---|
| dump1 | x>0 | (x,y)→(0,y) | Empty Jug 1 |
| dump2 | y>0 | (x,y)→(x,0) | Empty Jug 2 |
| pour_1_2 | x>0 & y<C2 | (x,y)→(x-D,y+D) *D = min(x,C2-y)* | Pour from Jug 1 to Jug 2 |
| pour_2_1 | y>0 & X<C1 | (x,y)→(x+D,y-D) *D = min(y,C1-x)* | Pour from Jug 2 to Jug 1 |

# Two Water Jugs Problem


Given J1 and J2 with capacities C1 and C2 and initial amounts W1 and W2, find actions to end up with W1' and W2' in jugs

**State Representation**

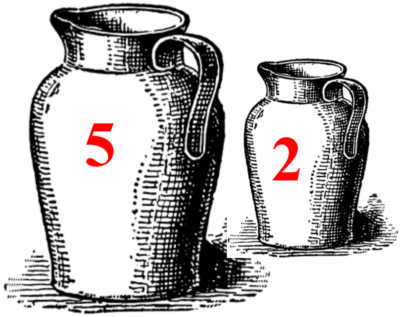
State = (x,y), where x & y are water in J1 & J2

- Initial state = (5,0)
- Goal state = (*,1), where * is any amount

Operator table

| Actions | Cond. | Transition | Effect |
|---|---|---|---|
| Empty J1 | – | $(x,y) \rightarrow (0,y)$ | Empty J1 |
| Empty J2 | – | $(x,y) \rightarrow (x,0)$ | Empty J2 |
| 2to1 | $x \leq 3$ | $(x,2) \rightarrow (x+2,0)$ | Pour J2 into J1 |
| 1to2 | $x \geq 2$ | $(x,0) \rightarrow (x-2,2)$ | Pour J1 into J2 |
| 1to2part | $y < 2$ | $(1,y) \rightarrow (0,y+1)$ | Pour J1 into J2 until full |

# Our WJ problem class




```
class WJ(Problem):
    def __init__(self, capacities=(5,2), initial=(5,0), goal=(0,1)):
        self.capacities = capacities
        self.initial = initial
        self.goal = goal

    def goal_test(self, state):  # returns True iff state is a goal state
        g = self.goal
        return (state[0] == g[0] or g[0] == -1 ) and
               (state[1] == g[1] or g[1] == -1)

  def __repr__(self):     # returns string representing the object
        return f"WJ({self.capacities},{self.initial},{self.goal}"
```

Note: f-string

# Our WJ problem class

```
def actions(self, state):
    """returns iterable with all state's legal actions"""
    (J1, J2) = state
    (C1, C2) = self.capacities
    if J1>0: yield ('dump', 1)
    if J2>0: yield ('dump', 2)
    if J2<C2 and J1>0: yield ('pour', 1, 2)
    if J1<C1 and J2>0: yield ('pour', 2, 1)
```

yield? See here

# Acytions returning a list

```
def actions(self, state):
    (J1, J2) = state
    (C1, C2) = self.capacities
    legal = []
    if J1>0: legal.append(('dump', 1))
    if J2>0: legal.append(('dump', 2))
    if J2<C2 and J1>0: legal.append(('pour', 1, 2))
    if J1<C1 and J2>0: yield (('pour', 2, 1))
    return legal
```

# WJ problem class

```
def result(self, state, action):
    """ Given state and action, returns successor
        after doing action"""
    if len(action) == 2:
        act, arg1 = action
    else:
        act, arg1, arg2 = action
    (J1, J2), (C1, C2) = state, self.capacities
    if act == 'dump':
        return (0, J2) if arg1 == 1 else (J1, 0)
    elif act == 'pour':
        if arg1 == 1:
            delta = min(J1, C2-J2)
            return (J1-delta, J2+delta)
        else:
            delta = min(J2, C1-J1)
            return (J1+delta, J2-delta)
```

# Our WJ problem class

```
def h(self, node):
    # heuristic function that estimates distance
    # to a goal node
    return 0 if self.goal_test(node.state) else 1
```

# Solving a WJP

```
code> python
>>> from wj import *                          # Import wj.py and search.py
>>> from aima3.search import *
>>> p1 = WJ((5,2), (5,2), ('*', 1))           # Create a problem instance
>>> p1
WJ((5, 2),(5, 2),('*', 1))
>>> answer = breadth_first_search(p1)         # Used the breadth 1st search function
>>> answer                                    # Will be None if the search failed or a
<Node (0, 1)>                                 #    a goal node in the search graph if successful
>>> answer.path_cost                          # The cost to get to every node in the search graph
6                                             #  is maintained by the search procedure
>>> path = answer.path()                      # A node's path is the best way to get to it from
>>> path                                      #   the start node, i.e., a solution
[<Node (5, 2)>, <Node (5, 0)>, <Node (3, 2)>, <Node (3, 0)>, <Node (1, 2)>, <Node (1, 0)>, <Node (0, 1)>]
```

# Comparing Search Algorithms Results

**Uninformed searches:** breadth_first_tree_search, breadth_first_search, depth_first_graph_ search, iterative_deepening_search, depth_limited_ search

- All but depth_limited_search are **sound** (i.e., solutions found are correct)
- Not all are **complete** (i.e., can find all solutions)
- Not all are **optimal** (find best possible solution)
- Not all are **efficient**
- AIMA code has a comparison function

# Comparing Search Algorithms Results

```
HW2> python
Python 2.7.6 |Anaconda 1.8.0 (x86_64)| ...
>>> from wj import *
>>> searchers=[breadth_first_search, depth_first_graph_search,
iterative_deepening_search]
>>> compare_searchers([WJ((5,2), (5,0), (0,1))], ['SEARCH ALGORITHM',
'successors/goal tests/states generated/solution'], searchers)
SEARCH ALGORITHM            successors/goal tests/states generated/solution
breadth_first_search        <   8/   9/  16/(0, >
depth_first_graph_search    <   5/   6/  12/(0, >
iterative_deepening_search  <  35/  61/  57/(0, >
>>>
```

# The Output

```
hhw2> python wjtest.py -s 5 0 -g 0 1
Solving WJ((5, 2),(5, 0),(0, 1)
  breadth_first_tree_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
  breadth_first_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
  depth_first_graph_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
  iterative_deepening_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
  astar_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
SUMMARY: successors/goal tests/states generated/solution
breadth_first_tree_search    <  25/  26/  37/(0, >
breadth_first_search         <   8/   9/  16/(0, >
depth_first_graph_search     <   5/   6/  12/(0, >
iterative_deepening_search   <  35/  61/  57/(0, >
astar_search                 <   8/  10/  16/(0, >
```