Search in Python

Chapter 3

Today’s topics

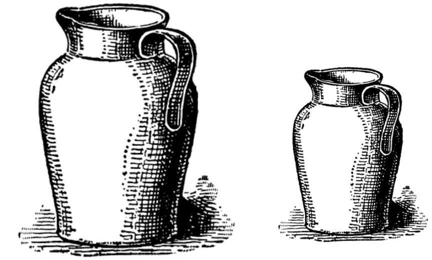
- Norvig’s Python code
- What it does
- How to use it
- A worked example: water jug program
- What about Java?

Overview

To use the AIMA python code for solving the two water jug problem (WJP) using search we'll need four files

- **wj.py**: need to write this to define the problem, states, goal, successor function, etc.
- **search.py**: Norvig's generic search framework, imported by wj.py
- **util.py** and **agents.py**: more generic Norvig code imported by search.py

Two Water Jugs Problem



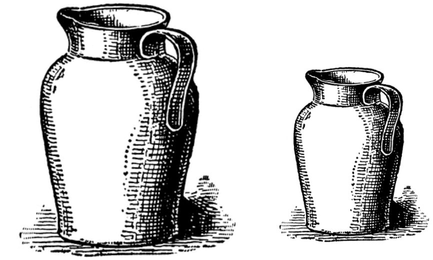
- Given two water jugs, J1 and J2, with capacities C1 and C2 and initial amounts W1 and W2, find actions to end up with W1' and W2' in the jugs
- Example problem:
  - We have a 5 gallon and a 2 gallon jug
  - Initially both are full
  - We want to end up with exactly one gallon in J2 and don't care how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

search.py

- Defines a Problem class for a search problem
- Provides functions to perform various kinds of search given an instance of a Problem

  e.g.: breadth first, depth first, hill climbing, A*, …
- Has a Problem subclass, InstrumentedProblem, and function, compare_searchers, for evaluation experiments
- 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.

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	$x \le 3$	$(x,2) \rightarrow (x+2,0)$	Pour J2 into J1
1to2	$x \ge 2$	$(x,0) \rightarrow (x-2,2)$	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 if state is a goal state
    g = self.goal
    return (state[0] == g[0] or g[0] == '*' ) and \
           (state[1] == g[1] or g[1] == '*')

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

Our WJ problem class

```
def successor(self, (J1, J2)):       # returns list of successors to state
    successors = []
    (C1, C2) = self.capacities
    if J1 > 0: successors.append((‘Dump J1', (0, J2)))
    if J2 > 0: successors.append((‘Dump J2', (J1, 0)))
    if J2 < C2 and J1 > 0:
        delta = min(J1, C2 – J2)
        successors.append((‘Pour J1 into J2', (J1 - delta, J2 + delta)))
    if J1 < C1 and J2 > 0:
        delta = min(J2, C1 – J1)
        successors.append(('pour J2 into J1', (J1 + delta, J2 - delta)))
    return successors
```

Solving a WJP

```
code> python
>>> from wj import *; from search import *          # Import wj.py and search.py
>>> p1 = WJ((5,2), (5,2), ('*', 1))                  # Create a problem instance
>>> p1
WJ((5, 2),(5, 2),('*', 1))
>>> answer = breadth_first_graph_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 (0, 1)>, <Node (1, 0)>, <Node (1, 2)>, <Node (3, 0)>, <Node (3, 2)>, <Node (5, 0)>, <Node (5, 2)>]
>>> path.reverse()
>>> path
[<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_graph_search, depth_first_graph_search, iterative_deepening_search, depth_limited_search
- All but depth_limited_search are sound (solutions found are correct)
- Not all are complete (always find a solution if one exists)
- Not all are optimal (find best possible solution)
- Not all are efficient
- AIMA code has a comparison function

Comparing Search Algorithms Results

```
def main():
   searchers = [breadth_first_tree_search, breadth_first_graph_search, depth_first_graph_search, …]
   problems = [WJ((5,2), (5,0), (0,1)),  WJ((5,2), (5,0), (2,0))]
   for p in problems:
      for s in searchers:
         print 'Solution to', p, 'found by', s.__name__
         path = s(p).path()     # call search function with problem
         path.reverse()
         print path, '\n'         # print solution path
   print 'SUMMARY: successors/goal tests/states generated/solution'
   # Now call the comparison function to show data about the performance of the dearches
    compare_searchers(problems=problems,
      header=['SEARCHER', 'GOAL:(0,1)', 'GOAL:(2,0)'],
      searchers=[breadth_first_tree_search,  breadth_first_graph_search, depth_first_graph_search,…])

# if called from the command line, call main()
if __name__ == "__main__":  main()
```

The Output

```
code> python wj.py
Solution to WJ((5, 2),  (5, 0),  (0, 1)) found by breadth_first_tree_search
[<Node (5, 0)>, <Node (3, 2)>, <Node (3, 0)>, <Node (1, 2)>, … , <Node (0, 1)>]
…
Solution to WJ((5, 2),  (5, 0),  (2, 0)) found by depth_limited_search
[<Node (5, 0)>, <Node (3, 2)>, <Node (0, 2)>, <Node (2, 0)>]

SUMMARY: successors/goal tests/states generated/solution
SEARCHER                          GOAL:(0,1)              GOAL:(2,0)
breadth_first_tree_search    <  25/  26/  37/(0, >   <   7/   8/  11/(2, >
breadth_first_graph_search   <   8/  17/  16/(0, >   <   5/   8/   9/(2, >
depth_first_graph_search     <   5/   8/  12/(0, >   <   8/  13/  16/(2, >
iterative_deepening_search   <  35/  61/  57/(0, >   <   8/  16/  14/(2, >
depth_limited_search         < 194/ 199/ 200/(0, >   <   5/   6/   7/(2, >
code>
```