Re85: *The Details* (BEST-FIRST-SEARCH Part 4, AIMA4e pp. 73–74)

retraice.com

Looking ahead at the code we'll need.

An attempt to build a toy `problem` reveals unsatisfied dependencies; the need for a `problem` implementation with state space, actions sets, transition model and action cost function; AIMA's `RouteProblem` class and `best_first_search` function implementations as guides; walking through the suites of each; the need for `PriorityQueue` and `f` to order our search tree's frontier of nodes.

Air date: Saturday, 17th Dec. 2022, 10:00 PM Eastern/US.

`expand(problem, node)` dependencies

```
################################################################################################################################
failure = Node('failure', path_cost=math.inf) # Indicates an algorithm couldn't find a solution.
cutoff  = Node('cutoff',  path_cost=math.inf) # Indicates iterative deepening search was cut off.
################################################################################################################################
def expand(problem, node):
    "Expand a node, generating the children nodes."
    s = node.state                                  # set variable s equal to the state attribute of the given node
    for action in problem.actions(s):               # for each action
        s1 = problem.result(s, action)              # s1 is the state that results from applying given action to state s
        cost = node.path_cost + problem.action_cost(s, action, s1)  # set variable cost to cost of going to s1
        yield Node(s1, node, action, cost) # generate child node given arguments s1, parent node, action applied, cost
################################################################################################################################
# run (Re85):

stspace = [1, 27, 88, 77, 11, 4, 32]  # state space... needs to have actions
getToEleven = Problem(1, 11)          # problem... ?? needs to subclass Problem and implement actions, result??

################################################################################################################################
```

We need to implement a `problem`, with a state space, initial and goal states, actions sets, transition model and action cost function.

The `RouteProblem` example

```
################################################################################################################################
class RouteProblem(Problem):
    """A problem to find a route between locations on a `Map`.
    Create a problem with RouteProblem(start, goal, map=Map(...)}).
    States are the vertexes in the Map graph; actions are destination states."""

    def actions(self, state):
        """The places neighboring `state`."""
        return self.map.neighbors[state]

    def result(self, state, action):
        """Go to the `action` place, if the map says that is possible."""
        return action if action in self.map.neighbors[state] else state

    def action_cost(self, s, action, s1):
        """The distance (cost) to go from s to s1."""
        return self.map.distances[s, s1]

    def h(self, node):
        "Straight-line distance between state and the goal."
        locs = self.map.locations
        return straight_line_distance(locs[node.state], locs[self.goal])

################################################################################################################################
```

The AIMA implementation of `RouteProblem`, a subclass of `Problem`.

Looking ahead at BEST-FIRST-SEARCH implemented

```
###############################################################################################################################
def best_first_search(problem, f):           # ***PROBLEM NOT DONE***
    "Search nodes with minimum f(node) value first."
    node = Node(problem.initial)             # done
    frontier = PriorityQueue([node], key=f)  # ***NOT DONE***
    reached = {problem.initial: node}    # create dict. reached: {'<problem initial state>': '<Node(problem.initial)>',}
    while frontier:                          # Starting with first node in frontier, ??and continuing through queue??
        node = frontier.pop()                # set node equal to the top node in frontier, removing it from queue
        if problem.is_goal(node.state):      # check if the state of this node is the goal state of the problem
            return node                      # if it is, return the node as output and stop
        for child in expand(problem, node):  # if not, expand the node and for each child node of it....
            s = child.state                  # set s equal to the child node's state
            if s not in reached or child.path_cost < reached[s].path_cost: # if node is new or cheaper than known ones
                reached[s] = child           # ?? add pair to reached dict. {'<child.state>': '<child node>',} ??
                frontier.add(child)          # add <child node> to frontier queue.<see ***CORRECTION*** below>
    return failure                           # if goal state not found by while loop above, return failure and stop.

# ***CORRECTION***: During the livestream I said that after the while loop was done, we'd return to the newly updated
# frontier. False.  The frontier updates at the end of each loop through the while, then the while starts over, and if
# the while never finds a goal state, it terminates and return failure is executed.
###############################################################################################################################
```

We're also going to need a `PriorityQueue`, and `f`, an implemented evaluation function that will prioritize our nodes for next expansion.

Other sources consulted during this livestream:

- Russell & Norvig (2020);
- Retraice (2022/12/14);
- Retraice (2022/12/15);
- Retraice (2022/12/16);
- http://aima.cs.berkeley.edu/figures.pdf;
- https://github.com/aimacode/aima-python/blob/master/search4e.ipynb;
- https://github.com/retraice/ReAIMA4e/tree/main/Re85–BEST-FIRST-Part-4.

□

References

Retraice (2022/12/14). Re82: What is a problem? (BEST-FIRST-SEARCH Part 1, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re82 Retrieved 15th Dec. 2022.

Retraice (2022/12/15). Re83: A Problem Instantiated (BEST-FIRST-SEARCH Part 2, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re83 Retrieved 16th Dec. 2022.

Retraice (2022/12/16). Re84: A Node Instantiated (BEST-FIRST-SEARCH Part 3, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re84 Retrieved 17th Dec. 2022.

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson, 4th ed. ISBN: 978-0134610993. Searches: https://www.amazon.com/s?k=978-0134610993 https://www.google.com/search?q=isbn+978-0134610993 https://lccn.loc.gov/2019047498

