# Re90: *The* Map *Class* (BEST-FIRST-SEARCH Part 9, AIMA4e pp. 73–74)

retraice.com

*How instantiations of* Map *work.*

The attributes of Map are locations, neighbors and distances; multimap produces the neighbors dictionary; tlinks has the actions (in pairs of states) and cost values in miles of our state space tmap; tlocations has the states of tmap; Problem has our initial and goal states as attributes, and the is_goal method; RouteProblem has our actions, result and action_cost methods.

Air date: Thursday, 22nd Dec. 2022, 10:00 PM Eastern/US.

## A Map has locations, neighbors and distances

```
137 [[…]]
146 class Map:
147     """A map of places in a 2D world: a graph with vertexes and links between them.
148     In `Map(links, locations)`, `links` can be either [(v1, v2)...] pairs,
149     or a {(v1, v2): distance...} dict. Optional `locations` can be {v1: (x, y)}
150     If `directed=False` then for every (v1, v2) link, we add a (v2, v1) link."""
151
152     def __init__(self, links, locations=None, directed=False):
153         if not hasattr(links, 'items'):          # Distances are 1 by default
154                                                  # --AIMA ## returns t or f
155
156             links = {link: 1 for link in links}  # If your links doesn't have
157                                                  # 'items' ?attributes, set each
158                                                  # 'items' to 1
159         if not directed:                         # 'directed=False' by default
160                                                  # above;
161
162             for (v1, v2) in list(links):         # list() creates list object.
163                 links[v2, v1] = links[v1, v2]    # This adds mirror-like entries
164                                                  # in our dict, for backtracking
165
166         self.distances = links                   # distances attribute is the
167                                                  # updated, backtrack-able links
168                                                  # dict. This outputs a look-up
169                                                  # table of the updated links
170                                                  # dictionary, key=state pair,
171                                                  # value = distance.
172
173         self.neighbors = multimap(links)         # get all neighbors of all
174                                                  # states, i.e. actions
175                                                  # available at each state
176
177         self.locations = locations or defaultdict(lambda: (0, 0))
178 #        self.locations = locations or defaultdict()
179                                                  # "from collections import
180                                                  # defaultdict" above
181
182                     # In [7]: tmap.locations Out[7]: {'A': (76, 497), 'S': (187,
183                     # 463), 'T': (83, 414), 'Z': (92, 539)}
184
185                     # In [8]: tmap = Map(tlinks)
186
187                     # In [9]: tmap.locations Out[9]: defaultdict(<function
-UU-:%%--F1  BEST-FIRST-SEARCH-Re90.py   49% L137   (Python Elpy hs ft-p) ----------
```

```
182                     # In [7]: tmap.locations Out[7]: {'A': (76, 497), 'S': (187,
183                     # 463), 'T': (83, 414), 'Z': (92, 539)}
184
185                     # In [8]: tmap = Map(tlinks)
186
187                     # In [9]: tmap.locations Out[9]: defaultdict(<function
188                     # __main__.Map.__init__.<locals>.<lambda>()>, {})
189
190                     # Maybe to avoid errors / exceptions?
191
192
193 # with lambda
194 # In [24]: tmap.locations
195 # Out[24]: defaultdict(<function __main__.Map.__init__.<locals>.<lambda>()>, {})
196
197 # without lambda
198 # In [32]: tmap.locations
199 # Out[32]: defaultdict(None, {})
200
201 def multimap(pairs) -> dict:    # "-> dict" is function annotation.  "Given
202                                 # (key, val) pairs, make a dict of {key:
203                                 # [val,...]}." --AIMA.
204     result = defaultdict(list)  # "from collections import defaultdict" above;
205                                 # list() creates list object.  defaultdict is a
206                                 # "dict subclass that calls a factory function
207                                 # to supply missing values" -- python.org
208     for key, val in pairs:      # pairs is now our reversed tlinks.
209                                 # pairs will be links, as in "self.neighbors =
210                                 # multimap(links)" above
211         result[key].append(val) # add key value pair to result
212     return result
213
214 ################################################################################
215 # run Re90 and Re91; t stands for test
216
217 tlinks = {('A', 'Z'): 75, ('A', 'S'): 140, ('A', 'T'): 118,
218           ('Yo', 'Momma'): 90, ('Momma', 'Dadda'): 91, ('T', 'Yo'): 38,}
219
220 tlocations = {'A': ( 76, 497), 'S': (187, 463), 'T': ( 83, 414), 'Z': (92, 539),
221               'Yo': ( 60, 30), 'Momma': ( 87, 17), 'Dadda': (100, 40),}
222
223 # tmap = Map(tlinks, tlocations)
224 [[…]]
-UU-:%%--F1  BEST-FIRST-SEARCH-Re90.py   Bot L277   (Python Elpy hs ft-p) ----------
```

```
15 class Problem(object):
16     """The abstract class for a formal problem. A new domain subclasses this,
17     overriding `actions` and `results`, and perhaps other methods.
18     The default heuristic is 0 and the default action cost is 1 for all states.
19     When yiou create an instance of a subclass, specify `initial`, and `goal` states
20     (or give an `is_goal` method) and perhaps other keyword args for the subclass."""
21     def __init__(self, initial=None, goal=None, **kwds):
22         self.__dict__.update(initial=initial, goal=goal, **kwds)
23
24     def actions(self, state):        raise NotImplementedError  # ***NOT DONE***
25     def result(self, state, action): raise NotImplementedError  # ***NOT DONE***
26                                                                 # in romania, state is
27                                                                 # and action is "('O',
28     def is_goal(self, state):        return state == self.goal
29     def action_cost(self, s, a, s1): return 1                   # ***NOT DONE***
30     def h(self, node):               return 0
31
32     def __str__(self):
33         return '{}({!r}, {!r})'.format(
34             type(self).__name__, self.initial, self.goal)
35
36 [[…]]
116 class RouteProblem(Problem): # e.g. atobproblem = RouteProblem('A', 'B',
117                             # map=simpleMap) """A problem to find a route
118                             # between locations on a `Map`.  Create a problem
119                             # with RouteProblem(start, goal, map=Map(...)).  #
120                             # here we'd pass the map romania States are the
121                             # vertexes in the Map graph; actions are
122                             # destination states."""
123
124     def actions(self, state):   # e.g. atobproblem.actions('A')
125         # """The places neighboring `state`."""
126         return self.map.neighbors[state] # actions-STEP-1, e.g. atobproblem.map.neighb
127
128     def result(self, state, action):
129         # """Go to the `action` place, if the map says that is possible."""
130         return action if action in self.map.neighbors[state] else state
131
132     def action_cost(self, s, action, s1):
133         # """The distance (cost) to go from s to s1."""
134         return self.map.distances[s, s1]
```

## tmap's locations, neighbors and distances

```
In [63]: tmap = Map(tlinks, tlocations)

In [64]: tmap.locations
Out[64]:
{'A': (76, 497),
 'S': (187, 463),
 'T': (83, 414),
 'Z': (92, 539),
 'Yo': (60, 30),
 'Momma': (87, 17),
 'Dadda': (100, 40)}

In [65]: tmap.neighbors
Out[65]:
defaultdict(list,
            {'A': ['Z', 'S', 'T'],
             'Yo': ['Momma', 'T'],
             'Momma': ['Dadda', 'Yo'],
             'T': ['Yo', 'A'],
             'Z': ['A'],
             'S': ['A'],
             'Dadda': ['Momma']})

In [66]: tmap.distances
Out[66]:
{('A', 'Z'): 75,
 ('A', 'S'): 140,
 ('A', 'T'): 118,
 ('Yo', 'Momma'): 90,
 ('Momma', 'Dadda'): 91,
 ('T', 'Yo'): 38,
 ('Z', 'A'): 75,
 ('S', 'A'): 140,
 ('T', 'A'): 118,
 ('Momma', 'Yo'): 90,
 ('Dadda', 'Momma'): 91,
 ('Yo', 'T'): 38}

In [67]:
```

```
In [67]: tproblem = RouteProblem('A', 'Dadda', map=tmap)

In [68]: tproblem.initial
Out[68]: 'A'

In [69]: tproblem.goal
Out[69]: 'Dadda'

In [70]: tproblem.actions('A')
Out[70]: ['Z', 'S', 'T']

In [72]: tproblem.action_cost('A', 'Z', 'Z')
Out[72]: 75

In [73]: tproblem.result('A', 'Z')
Out[73]: 'Z'

In [74]: tproblem.result('Yo', 'Momma')
Out[74]: 'Momma'
```

## Formalizing a search *problem*[1] with implementation

- **state space,** a set of possible states of the environment and the actions that transition from one to another:
  `tmap`, an instantiation of `Map` with arguments `tlinks` (actions, with costs in miles), and `tlocations` (the set of possible states, with coordinates).

- **initial state,** the state in which the agent starts:
  Given as the first argument (`'A'`) in `tproblem = RouteProblem('A', 'Dadda', map=tmap)`.

- **goal state(s),** a set of one or more; account for one, some, infinite (by means of a property) by specifying IS-GOAL method for problem:
  Given as the second argument (`'Dadda'`) in `tproblem = RouteProblem('A', 'Dadda', map=tmap)`.
  The `is_goal(self, state)` method is part of the `Problem` parent class.

- **actions,** what the agent can do; ACTIONS($state$) returns a finite set of actions that can be executed in $state$:
  `tlinks`, which also has costs in miles.
  The `actions(self, state)` method is part of the `RouteProblem` class.

- **transition model,** describes what actions do; RESULT($state, action$) returns the state $s'$ that results from doing $action$ in $state$:
  The `result(self, state, action)` method is part of the `RouteProblem` class.

- **action cost function,** ACTION-COST($s, a, s'$) gives the numeric cost of applying action $a$ in state $s$ to reach new state $s'$. Cf. the **evaluation function**, which we'll use to prioritize our nodes for next expansion, and the **objective function**, which was our cost measure to be minimized in the airport problem.[2]
  The `action_cost(self, s, action, s1)` method is part of the `RouteProblem` class.

[1] Russell & Norvig (2020) p. 65.
[2] Retraice (2022/12/11).

Other sources consulted during this livestream:

- Russell & Norvig (2020);
- Retraice (2022/12/14);
- Retraice (2022/12/15);
- Retraice (2022/12/16);
- Retraice (2022/12/17);
- Retraice (2022/12/18);
- Retraice (2022/12/19);
- Retraice (2022/12/20);
- Retraice (2022/12/21);
- http://aima.cs.berkeley.edu/figures.pdf;
- https://github.com/aimacode/aima-python/blob/master/search4e.ipynb;
- https://github.com/retraice/ReAIMA4e/.

□

# References

Retraice (2022/12/11). Re78: Recap of Gradients and Partial Derivatives (AIMA4e pp. 119–122). *retraice.com*. https://www.retraice.com/segments/re78 Retrieved 12th Dec. 2022.

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.

Retraice (2022/12/17). Re85: The Details (BEST-FIRST-SEARCH Part 4, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re85 Retrieved 18th Dec. 2022.

Retraice (2022/12/18). Re86: Code Reading (BEST-FIRST-SEARCH Part 5, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re86 Retrieved 19th Dec. 2022.

Retraice (2022/12/19). Re87: The multimap Function, Part A (BEST-FIRST-SEARCH Part 6, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re87 Retrieved 20th Dec. 2022.

Retraice (2022/12/20). Re88: The multimap Function, Part B (BEST-FIRST-SEARCH Part 7, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re88 Retrieved 21th Dec. 2022.

Retraice (2022/12/21). Re89: The multimap Function, Part C (BEST-FIRST-SEARCH Part 8, AIMA4e pp. 73–74). *retraice.com*. https://www.retraice.com/segments/re89 Retrieved 22st 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

 