

# 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; tmap 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 [[-]]
138
139 class Map:
140     """A map of places in a 2D world: a graph with vertices and links between them.
141     In 'Map(links, locations)', 'links' can be either [(v1, v2)...] pairs,
142     or a {(v1, v2): distance...} dict. Optional 'locations' can be {(x, y)}
143     If 'directed=False' then for every (v1, v2) link, we add a (v2, v1) link."""
144
145     def __init__(self, links, locations=None, directed=False):
146         if not hasattr(links, 'items'): # Distances are 1 by default
147             # --AIMA ## returns t or f
148
149             links = {link: 1 for link in links} # If your links doesn't have
150             # 'items' ?attributes, set each
151             # 'items' to 1
152
153         if not directed:
154             # 'directed=False' by default
155             # above;
156
157         for (v1, v2) in list(links):
158             # list() creates list object.
159             # This adds mirror-like entries
160             links[v2, v1] = links[v1, v2]
161
162         self.distances = links
163             # distances attribute is the
164             # updated, backtrack-able links
165             # dict. This outputs a look-up
166             # table of the updated links
167             # dictionary, key=state pair,
168             # value = distance.
169
170         self.neighbors = multimap(links)
171             # get all neighbors of all
172             # states, i.e. actions
173             # available at each state
174
175         self.locations = locations or defaultdict(lambda: (0, 0))
176         self.locations = locations or defaultdict()
177             # "from collections import
178             # defaultdict" above
179
180         # In [7]: tmap.locations Out[7]: {'A': (76, 497), 'S': (187,
181         # 463), 'T': (83, 414), 'Z': (92, 539)}
182
183         # In [8]: tmap = Map(tlinks)
184
185         # In [9]: tmap.locations Out[9]: defaultdict(<function
186         # __main__.Map.__init__.<locals>.<lambda>()>, {})
187
188         # Maybe to avoid errors / exceptions?
189
190         # with lambda
191         # In [24]: tmap.locations
192         # Out[24]: defaultdict(<function __main__.Map.__init__.<locals>.<lambda>()>, {})
193
194         # without lambda
195         # In [32]: tmap.locations
196         # Out[32]: defaultdict(None, {})
197
198         def multimap(pairs) -> dict: # "-> dict" is function annotation. "Given
199             # (key, val) pairs, make a dict of {key:
200             # [val,...]}." --AIMA.
201             result = defaultdict(list) # "from collections import defaultdict" above;
202             # list() creates list object. defaultdict is a
203             # "dict subclass that calls a factory function
204             # to supply missing values" -- python.org
205             for key, val in pairs: # pairs is now our reversed tlinks.
206                 # pairs will be links, as in "self.neighbors =
207                 # multimap(links)" above
208                 result[key].append(val) # add key value pair to result
209             return result
210
211         #####
212         # run Re90 and Re91; t stands for test
213
214         tlinks = {'A': (76, 497), 'S': (187, 463), 'T': (83, 414), 'Z': (92, 539),
215                 'Yo': (60, 30), 'Momma': (87, 17), 'Dadda': (100, 40)}
216
217         tlocations = {'A': (76, 497), 'S': (187, 463), 'T': (83, 414), 'Z': (92, 539),
218                     'Yo': (60, 30), 'Momma': (87, 17), 'Dadda': (100, 40)}
219
220         # tmap = Map(tlinks, tlocations)
221
222         # In [1]:
223
224         # In [1]:
225
226         # In [1]:
227
228         # In [1]:
229
230         # In [1]:
231
232         # In [1]:
233
234         # In [1]:
235
236         # In [1]:
237
238         # In [1]:
239
240         # In [1]:
241
242         # In [1]:
243
244         # In [1]:
245
246         # In [1]:
247
248         # In [1]:
249
250         # In [1]:
251
252         # In [1]:
253
254         # In [1]:
255
256         # In [1]:
257
258         # In [1]:
259
260         # In [1]:
261
262         # In [1]:
263
264         # In [1]:
265
266         # In [1]:
267
268         # In [1]:
269
270         # In [1]:
271
272         # In [1]:
273
274         # In [1]:
275
276         # In [1]:
277
278         # In [1]:
279
280         # In [1]:
281
282         # In [1]:
283
284         # In [1]:
285
286         # In [1]:
287
288         # In [1]:
289
290         # In [1]:
291
292         # In [1]:
293
294         # In [1]:
295
296         # In [1]:
297
298         # In [1]:
299
300         # In [1]:
301
302         # In [1]:
303
304         # In [1]:
305
306         # In [1]:
307
308         # In [1]:
309
310         # In [1]:
311
312         # In [1]:
313
314         # In [1]:
315
316         # In [1]:
317
318         # In [1]:
319
320         # In [1]:
321
322         # In [1]:
323
324         # In [1]:
325
326         # In [1]:
327
328         # In [1]:
329
330         # In [1]:
331
332         # In [1]:
333
334         # In [1]:
335
336         # In [1]:
337
338         # In [1]:
339
340         # In [1]:
341
342         # In [1]:
343
344         # In [1]:
345
346         # In [1]:
347
348         # In [1]:
349
350         # In [1]:
351
352         # In [1]:
353
354         # In [1]:
355
356         # In [1]:
357
358         # In [1]:
359
360         # In [1]:
361
362         # In [1]:
363
364         # In [1]:
365
366         # In [1]:
367
368         # In [1]:
369
370         # In [1]:
371
372         # In [1]:
373
374         # In [1]:
375
376         # In [1]:
377
378         # In [1]:
379
380         # In [1]:
381
382         # In [1]:
383
384         # In [1]:
385
386         # In [1]:
387
388         # In [1]:
389
390         # In [1]:
391
392         # In [1]:
393
394         # In [1]:
395
396         # In [1]:
397
398         # In [1]:
399
400         # In [1]:
401
402         # In [1]:
403
404         # In [1]:
405
406         # In [1]:
407
408         # In [1]:
409
410         # In [1]:
411
412         # In [1]:
413
414         # In [1]:
415
416         # In [1]:
417
418         # In [1]:
419
420         # In [1]:
421
422         # In [1]:
423
424         # In [1]:
425
426         # In [1]:
427
428         # In [1]:
429
430         # In [1]:
431
432         # In [1]:
433
434         # In [1]:
435
436         # In [1]:
437
438         # In [1]:
439
440         # In [1]:
441
442         # In [1]:
443
444         # In [1]:
445
446         # In [1]:
447
448         # In [1]:
449
450         # In [1]:
451
452         # In [1]:
453
454         # In [1]:
455
456         # In [1]:
457
458         # In [1]:
459
460         # In [1]:
461
462         # In [1]:
463
464         # In [1]:
465
466         # In [1]:
467
468         # In [1]:
469
470         # In [1]:
471
472         # In [1]:
473
474         # In [1]:
475
476         # In [1]:
477
478         # In [1]:
479
480         # In [1]:
481
482         # In [1]:
483
484         # In [1]:
485
486         # In [1]:
487
488         # In [1]:
489
490         # In [1]:
491
492         # In [1]:
493
494         # In [1]:
495
496         # In [1]:
497
498         # In [1]:
499
500         # In [1]:
501
502         # In [1]:
503
504         # In [1]:
505
506         # In [1]:
507
508         # In [1]:
509
510         # In [1]:
511
512         # In [1]:
513
514         # In [1]:
515
516         # In [1]:
517
518         # In [1]:
519
520         # In [1]:
521
522         # In [1]:
523
524         # In [1]:
525
526         # In [1]:
527
528         # In [1]:
529
530         # In [1]:
531
532         # In [1]:
533
534         # In [1]:
535
536         # In [1]:
537
538         # In [1]:
539
540         # In [1]:
541
542         # In [1]:
543
544         # In [1]:
545
546         # In [1]:
547
548         # In [1]:
549
550         # In [1]:
551
552         # In [1]:
553
554         # In [1]:
555
556         # In [1]:
557
558         # In [1]:
559
560         # In [1]:
561
562         # In [1]:
563
564         # In [1]:
565
566         # In [1]:
567
568         # In [1]:
569
570         # In [1]:
571
572         # In [1]:
573
574         # In [1]:
575
576         # In [1]:
577
578         # In [1]:
579
580         # In [1]:
581
582         # In [1]:
583
584         # In [1]:
585
586         # In [1]:
587
588         # In [1]:
589
590         # In [1]:
591
592         # In [1]:
593
594         # In [1]:
595
596         # In [1]:
597
598         # In [1]:
599
600         # In [1]:
601
602         # In [1]:
603
604         # In [1]:
605
606         # In [1]:
607
608         # In [1]:
609
610         # In [1]:
611
612         # In [1]:
613
614         # In [1]:
615
616         # In [1]:
617
618         # In [1]:
619
620         # In [1]:
621
622         # In [1]:
623
624         # In [1]:
625
626         # In [1]:
627
628         # In [1]:
629
630         # In [1]:
631
632         # In [1]:
633
634         # In [1]:
635
636         # In [1]:
637
638         # In [1]:
639
640         # In [1]:
641
642         # In [1]:
643
644         # In [1]:
645
646         # In [1]:
647
648         # In [1]:
649
650         # In [1]:
651
652         # In [1]:
653
654         # In [1]:
655
656         # In [1]:
657
658         # In [1]:
659
660         # In [1]:
661
662         # In [1]:
663
664         # In [1]:
665
666         # In [1]:
667
668         # In [1]:
669
670         # In [1]:
671
672         # In [1]:
673
674         # In [1]:
675
676         # In [1]:
677
678         # In [1]:
679
680         # In [1]:
681
682         # In [1]:
683
684         # In [1]:
685
686         # In [1]:
687
688         # In [1]:
689
690         # In [1]:
691
692         # In [1]:
693
694         # In [1]:
695
696         # In [1]:
697
698         # In [1]:
699
700         # In [1]:
701
702         # In [1]:
703
704         # In [1]:
705
706         # In [1]:
707
708         # In [1]:
709
710         # In [1]:
711
712         # In [1]:
713
714         # In [1]:
715
716         # In [1]:
717
718         # In [1]:
719
720         # In [1]:
721
722         # In [1]:
723
724         # In [1]:
725
726         # In [1]:
727
728         # In [1]:
729
730         # In [1]:
731
732         # In [1]:
733
734         # In [1]:
735
736         # In [1]:
737
738         # In [1]:
739
740         # In [1]:
741
742         # In [1]:
743
744         # In [1]:
745
746         # In [1]:
747
748         # In [1]:
749
750         # In [1]:
751
752         # In [1]:
753
754         # In [1]:
755
756         # In [1]:
757
758         # In [1]:
759
760         # In [1]:
761
762         # In [1]:
763
764         # In [1]:
765
766         # In [1]:
767
768         # In [1]:
769
770         # In [1]:
771
772         # In [1]:
773
774         # In [1]:
775
776         # In [1]:
777
778         # In [1]:
779
780         # In [1]:
781
782         # In [1]:
783
784         # In [1]:
785
786         # In [1]:
787
788         # In [1]:
789
790         # In [1]:
791
792         # In [1]:
793
794         # In [1]:
795
796         # In [1]:
797
798         # In [1]:
799
800         # In [1]:
801
802         # In [1]:
803
804         # In [1]:
805
806         # In [1]:
807
808         # In [1]:
809
810         # In [1]:
811
812         # In [1]:
813
814         # In [1]:
815
816         # In [1]:
817
818         # In [1]:
819
820         # In [1]:
821
822         # In [1]:
823
824         # In [1]:
825
826         # In [1]:
827
828         # In [1]:
829
830         # In [1]:
831
832         # In [1]:
833
834         # In [1]:
835
836         # In [1]:
837
838         # In [1]:
839
840         # In [1]:
841
842         # In [1]:
843
844         # In [1]:
845
846         # In [1]:
847
848         # In [1]:
849
850         # In [1]:
851
852         # In [1]:
853
854         # In [1]:
855
856         # In [1]:
857
858         # In [1]:
859
860         # In [1]:
861
862         # In [1]:
863
864         # In [1]:
865
866         # In [1]:
867
868         # In [1]:
869
870         # In [1]:
871
872         # In [1]:
873
874         # In [1]:
875
876         # In [1]:
877
878         # In [1]:
879
880         # In [1]:
881
882         # In [1]:
883
884         # In [1]:
885
886         # In [1]:
887
888         # In [1]:
889
890         # In [1]:
891
892         # In [1]:
893
894         # In [1]:
895
896         # In [1]:
897
898         # In [1]:
899
900         # In [1]:
901
902         # In [1]:
903
904         # In [1]:
905
906         # In [1]:
907
908         # In [1]:
909
910         # In [1]:
911
912         # In [1]:
913
914         # In [1]:
915
916         # In [1]:
917
918         # In [1]:
919
920         # In [1]:
921
922         # In [1]:
923
924         # In [1]:
925
926         # In [1]:
927
928         # In [1]:
929
930         # In [1]:
931
932         # In [1]:
933
934         # In [1]:
935
936         # In [1]:
937
938         # In [1]:
939
940         # In [1]:
941
942         # In [1]:
943
944         # In [1]:
945
946         # In [1]:
947
948         # In [1]:
949
950         # In [1]:
951
952         # In [1]:
953
954         # In [1]:
955
956         # In [1]:
957
958         # In [1]:
959
960         # In [1]:
961
962         # In [1]:
963
964         # In [1]:
965
966         # In [1]:
967
968         # In [1]:
969
970         # In [1]:
971
972         # In [1]:
973
974         # In [1]:
975
976         # In [1]:
977
978         # In [1]:
979
980         # In [1]:
981
982         # In [1]:
983
984         # In [1]:
985
986         # In [1]:
987
988         # In [1]:
989
990         # In [1]:
991
992         # In [1]:
993
994         # In [1]:
995
996         # In [1]:
997
998         # In [1]:
999
1000        # In [1]:

```

```

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 you 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, **kwargs):
22         self.__dict__.update(initial=initial, goal=goal, **kwargs)
23
24     def actions(self, state): raise NotImplementedError # **NOT DONE**
25     def result(self, state, action): raise NotImplementedError # **NOT DONE**
26
27     def is_goal(self, state): return state == self.goal # in romania, state is:
28     def action_cost(self, s, a, s1): return 1 # and action is '{0}'.
29     def h(self, node): return 0 # **NOT DONE**
30
31     def __str__(self):
32         return '{}({!r}, {!r})'.format(
33             type(self).__name__, self.initial, self.goal)
34
35
36 [[-]]
37
38 class RouteProblem(Problem): # e.g. atobproblem = RouteProblem('A', 'B',
39                             # map=simpleMap) """A problem to find a route
40                             # between locations on a 'Map'. Create a problem
41                             # with RouteProblem(start, goal, map=Map(...)). #
42                             # here we'd pass the map romania States are the
43                             # vertices in the Map graph; actions are
44                             # destination states."""
45
46     def actions(self, state): # e.g. atobproblem.actions('A')
47         # """The places neighboring state"""
48         return self.map.neighbors[state] # actions-STEP-1, e.g. atobproblem.map.neighb
49
50     def result(self, state, action):
51         # """Go to the 'action' place, if the map says that is possible."""
52         return action if action in self.map.neighbors[state] else state
53
54     def action_cost(self, s, action, s1):
55         # """The distance (cost) to go from s to s1."""
56         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]: 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*<sup>1</sup> 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.<sup>2</sup>  
The action\_cost(self, s, action, s1) method is part of the RouteProblem class.

<sup>1</sup>Russell & Norvig (2020) p. 65.

<sup>2</sup>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>