CSPB 3202 - Truong - Artificial Intelligence

Started on	Wednesday, 29 May 2024, 2:13 PM
State	Finished
Completed on	Tuesday, 4 June 2024, 5:49 PM
Time taken	6 days 3 hours
Grade	**70.00** out of 70.00 (**100**%)

Question **1**

Correct

Mark 10.00 out of 10.00

Modify your code for uniform-cost search from Homework 1 so that it provides optionally as output the number of nodes expanded in completing the search.

Include a new optional logical (True/False) argument return_nexp, so your function calls to the new uniform cost search will look like: uniform_cost(start, goal, state_graph, return_cost,return_nexp).

• If return_nexp is True, then the last output in the output tuple should be the number of nodes expanded.

• If return_nexp is False, then the code should behave exactly as it did in Homework 1.

Then, verify that your revised codes are working by checking Neal's optimal route from New York to Chicago. Include the number of nodes expanded and the path cost (using map_distances).

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np
import heapq
import unittest

def path(previous, s):
    '''
    `previous` is a dictionary chaining together the predecessor state that led to each state
    `s` will be None for the initial state
    otherwise, start from the last state `s` and recursively trace `previous` back to the initial
    constructing a list of states visited as we go
    '''
    if s is None:
        return []
    else:
        return path(previous, previous[s])+[s]

def pathcost(path, step_costs):
    '''
    add up the step costs along a path, which is assumed to be a list output from the `path` funct
    '''
    cost = 0
```

	Test	Expected	Got	

✔	`path, cost, nexp = uniform_cost('chi','new',map_distances,return_cost=True,return_nexp=True)`
	`print("Path:",path)`
	`print("Cost:",cost)`
	`print("Nexp:",nexp)`
		`Path: ['chi', 'cle', 'pit', 'phi', 'new']`
		`Cost: 881`
		`Nexp: 11`
			`Path: ['chi', 'cle', 'pit', 'phi', 'new']`
			`Cost: 881`
			`Nexp: 11`
				✔

✔	`path = uniform_cost('chi','new',map_distances,return_cost=False,return_nexp=False)` `print("Path:",path)`	`Path: ['chi', 'cle', 'pit', 'phi', 'new']`	`Path: ['chi', 'cle', 'pit', 'phi', 'new']`	✔

✔	`path, cost = uniform_cost('chi','new',map_distances,return_cost=True,return_nexp=False)`
	`print("Path:",path)`
	`print("Cost:",cost)`
		`Path: ['chi', 'cle', 'pit', 'phi', 'new']`
		`Cost: 881`
			`Path: ['chi', 'cle', 'pit', 'phi', 'new']`
			`Cost: 881`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **2**

Correct

Mark 10.00 out of 10.00

Define a function to take as an argument the state that Neal is in (city on our graphs), and return as output the value of the straight-line distance heuristic, between Neal's state and Providence.

Note that your function should be quite short, and amounts to looking up the proper value from the sld_providence dictionary defined in the helper functions. Call this function heuristic_sld_providence.

Answer: (penalty regime: 0 %)

Reset answer

```
sld_providence = dict(
    chi=833,
    cle=531,
    ind=782,
    col=618,
    det=596,
    buf=385,
    pit=458,
    syr=253,
    bal=325,
    phi=236,
    new=157,
    pro=0,
    bos=38,
    por=136)

""" heuristic_sld_providence - Returns the straight-line distance heuristic between the given stat
    Input:
        state - The current city/state as a string.
    Output:
        The straight-line distance from the given state to Providence as an integer.
```

	Test	Expected	Got	
✔	`sld = heuristic_sld_providence('chi')` `print(sld)`	833	833	✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **3**

Correct

Mark 10.00 out of 10.00

We are finally ready to help Neal use his knowledge of straight-line distances from various cities to Providence to inform his family's route to move from Chicago to Providence!

Modify your uniform-cost search codes from 1.1 even further so that they now perform A* search, using as the heuristic function the straight-line distance to Providence.

Provide heuristic as an additional argument, which should just be the function to call within the A* code. So your call to the A routine should look like: astar_search(start, goal, state_graph,

heuristic, return_cost, return_nexp). (This kind of modular programming will make it much easier to swap in alternative heuristic functions later, and also helps to facilitate debugging if something goes wrong.)

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np
import heapq
import unittest

def path(previous, s):
    '''
    `previous` is a dictionary chaining together the predecessor state that led to each state
    `s` will be None for the initial state
    otherwise, start from the last state `s` and recursively trace `previous` back to the initial
    constructing a list of states visited as we go
    '''
    if s is None:
        return []
    else:
        return path(previous, previous[s])+[s]

def pathcost(path, step_costs):
    '''
    add up the step costs along a path, which is assumed to be a list output from the `path` funct
    '''
    cost = 0
```

	Test	Expected	Got	
✔	`path = astar_search('pit','pro', map_distances, heuristic_sld_providence, return_cost=False, return_nexp=False)` `print("Path:",path)`	`Path: ['pit', 'phi', 'new', 'pro']`	`Path: ['pit', 'phi', 'new', 'pro']`	✔

✔	`path, cost = astar_search('pit','pro', map_distances, heuristic_sld_providence, return_cost=True, return_nexp=False)`
	`print("Path:",path)`
	`print("Cost:",cost)`
		`Path: ['pit', 'phi', 'new', 'pro']`
		`Cost: 583`
			`Path: ['pit', 'phi', 'new', 'pro']`
			`Cost: 583`
				✔

✔	`path, cost, nexp = astar_search('pit','pro', map_distances, heuristic_sld_providence, return_cost=True, return_nexp=True)`
	`print("Path:",path)`
	`print("Cost:",cost)`
	`print("NextP:",nexp)`
		`Path: ['pit', 'phi', 'new', 'pro']`
		`Cost: 583`
		`NextP: 5`
			`Path: ['pit', 'phi', 'new', 'pro']`
			`Cost: 583`
			`NextP: 5`
				✔

Passed all tests! ✔

Pass the astar_search unit tests

Correct

Marks for this submission: 10.00/10.00.

Question **4**

Correct

Mark 10.00 out of 10.00

Print the the following using your code:

1. the optimal path

2. the optimal path cost (miles traveled)

3. the number of states expanded during the A* search

Additionally, print how many states must be expanded to find the optimal path from Buffalo to Providence using the regular old uniform-cost search algorithm from 1.1

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np
import heapq
import unittest

def path(previous, s):
    '''
    `previous` is a dictionary chaining together the predecessor state that led to each state
    `s` will be None for the initial state
    otherwise, start from the last state `s` and recursively trace `previous` back to the initial
    constructing a list of states visited as we go
    '''
    if s is None:
        return []
    else:
        return path(previous, previous[s])+[s]

def pathcost(path, step_costs):
    '''
    add up the step costs along a path, which is assumed to be a list output from the `path` funct
    '''
    cost = 0
```

	Test	Expected	Got	

✔	`start = 'buf'`
	`goal = 'pro'`
	`print("A-star:",astar_search(start, goal, map_distances, heuristic_sld_providence, return_cost=True, return_nexp=True))`
	`print("Uniform-Cost:",uniform_cost(start, goal, map_distances, return_cost=True, return_nexp=True))`
		`A-star: (['buf', 'syr', 'bos', 'pro'], 512, 4)`
		`Uniform-Cost: (['buf', 'syr', 'bos', 'pro'], 512, 12)`
			`A-star: (['buf', 'syr', 'bos', 'pro'], 512, 4)`
			`Uniform-Cost: (['buf', 'syr', 'bos', 'pro'], 512, 12)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **5**

Correct

Mark 10.00 out of 10.00

Comment on the difference in states that must be explored by each algorithm. Sanity check: No matter what your start and goal states are, how should the output from astar_search and uniform_cost search compare?

Answer: (penalty regime: 0 %)

```
'''
The output from the astar_search and uniform_cost search should have the same paths as one another.
The difference between the two should lie in how many nodes are expanded for each.

In the astar search, there should be less nodes that are expanded (and this is true in the previous
and for uniform cost search there should be more (and there is in the previous example).
'''
123
```

	Test
✔	123

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **6**

Correct

Mark 10.00 out of 10.00

How many states are expanded by each of A*search and uniform cost search to find the optimal path from Philadelphia to Providence?

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np
import heapq
import unittest

def path(previous, s):
    '''
    `previous` is a dictionary chaining together the predecessor state that led to each state
    `s` will be None for the initial state
    otherwise, start from the last state `s` and recursively trace `previous` back to the initial
    constructing a list of states visited as we go
    '''
    if s is None:
        return []
    else:
        return path(previous, previous[s])+[s]

def pathcost(path, step_costs):
    '''
    add up the step costs along a path, which is assumed to be a list output from the `path` funct
    '''
    cost = 0
```

	Test	Expected	Got	

✔	`start = 'phi'`
	`goal = 'pro'`
	`print("A-star:",astar_search(start, goal, map_distances, heuristic_sld_providence, return_cost=True, return_nexp=True))`
	`print("Uniform-Cost:",uniform_cost(start, goal, map_distances, return_cost=True, return_nexp=True))`
		`A-star: (['phi', 'new', 'pro'], 278, 3)`
		`Uniform-Cost: (['phi', 'new', 'pro'], 278, 5)`
			`A-star: (['phi', 'new', 'pro'], 278, 3)`
			`Uniform-Cost: (['phi', 'new', 'pro'], 278, 5)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **7**

Correct

Mark 10.00 out of 10.00

Moodle Quiz Problem 7. Pass the unit tests for the CSP class

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import OrderedDict

canada = OrderedDict(
    [("AB"  , ["BC","NT","SK"]),
    ("BC" , ["AB","NT","YT"]),
    ("LB" , ["NF", "NS", "PE","QC"]),
    ("MB" , ["ON","NV","SK"]),
    ("NB" , ["NS","QC"]),
    ("NF" , ["LB","QC"]),
    ("NS" , ["LB","NB","PE"]),
    ("NT" , ["AB","BC","NV","SK","YT"]),
    ("NV" , ["MB","NT"]),
    ("ON" , ["MB","QC"]),
    ("PE" , ["LB","NS","QC"]),
    ("QC" , ["LB","NB","NF","ON","PE"]),
    ("SK" , ["AB","MB","NT"]),
    ("YT" , ["BC","NT"])])

states = ["AB", "BC", "LB", "MB", "NB", "NF", "NS", "NT", "NV", "ON", "PE", "QC", "SK", "YT"]
colors = ["blue", "green", "red"]

class CSP:
```

	Test
✔	`OrderedDict([('AB', ['BC', 'NT', 'SK']), ('BC', ['AB', 'NT', 'YT']), ('LB', ['NF', 'NS', 'PE', 'QC']), ('MB', ['ON', 'NV', 'SK']), ('NB', ['NS', 'QC']), ('NF', ['LB', 'QC']), ('NS', ['LB', 'NB', 'PE']), ('NT', ['AB', 'BC', 'NV', 'SK', 'YT']), ('NV', ['MB', 'NT']), ('ON', ['MB', 'QC']), ('PE', ['LB', 'NS', 'QC']), ('QC', ['LB', 'NB', 'NF', 'ON', 'PE']), ('SK', ['AB', 'MB', 'NT']), ('YT', ['BC', 'NT'])])` `['AB', 'BC', 'LB', 'MB', 'NB', 'NF', 'NS', 'NT', 'NV', 'ON', 'PE', 'QC', 'SK', 'YT']` `['blue', 'green', 'red']`

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.