CSPB 3202 - Truong - Artificial Intelligence

Started on	Thursday, 23 May 2024, 8:11 PM
State	Finished
Completed on	Thursday, 23 May 2024, 8:13 PM
Time taken	2 mins 5 secs
Marks	100.00/100.00
Grade	**80.00** out of 80.00 (**100**%)

Question **1**

Correct

Mark 10.00 out of 10.00

Implement a function breadth_first(start, goal, state_graph, return_cost) to search the state space (and step costs) defined by state_graph using breadth-first search:

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
from collections import OrderedDict
map_distances = dict(
    chi=OrderedDict([("det",283), ("cle",345), ("ind",182)]),
    cle=OrderedDict([("chi",345), ("det",169), ("col",144), ("pit",134), ("buf",189)]),
    ind=OrderedDict([("chi",182), ("col",176)]),
    col=OrderedDict([("ind",176), ("cle",144), ("pit",185)]),
    det=OrderedDict([("chi",283), ("cle",169), ("buf",256)]),
    buf=OrderedDict([("det",256), ("cle",189), ("pit",215), ("syr",150)]),
    pit=OrderedDict([("col",185), ("cle",134), ("buf",215), ("phi",305), ("bal",247)]),
    syr=OrderedDict([("buf",150), ("phi",253), ("new",254), ("bos",312)]),
    bal=OrderedDict([("phi",101), ("pit",247)]),
    phi=OrderedDict([("pit",305), ("bal",101), ("syr",253), ("new",97)]),
    new=OrderedDict([("syr",254), ("phi",97), ("bos",215), ("pro",181)]),
    pro=OrderedDict([("bos",50), ("new",181)]),
    bos=OrderedDict([("pro",50), ("new",215), ("syr",312), ("por",107)]),
    por=OrderedDict([("bos",107)]))

map_times = dict(
    chi=dict(det=280, cle=345, ind=200),
    cle=dict(chi=345, det=170, col=155, pit=145, buf=185),
```

	Test	Expected	Got	

✔	`path, cost = breadth_first('chi','pit',map_distances,True)`
	`print(path)`
	`print(cost)`
		`['chi', 'cle', 'pit']`
		`479`
			`['chi', 'cle', 'pit']`
			`479`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **2**

Correct

Mark 10.00 out of 10.00

Implement a function depth_first(start, goal, state_graph, return_cost) to search the state space (and step costs) defined by state_graph using depth-first search:

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
from collections import OrderedDict

map_distances = dict(
    chi=OrderedDict([("det",283), ("cle",345), ("ind",182)]),
    cle=OrderedDict([("chi",345), ("det",169), ("col",144), ("pit",134), ("buf",189)]),
    ind=OrderedDict([("chi",182), ("col",176)]),
    col=OrderedDict([("ind",176), ("cle",144), ("pit",185)]),
    det=OrderedDict([("chi",283), ("cle",169), ("buf",256)]),
    buf=OrderedDict([("det",256), ("cle",189), ("pit",215), ("syr",150)]),
    pit=OrderedDict([("col",185), ("cle",134), ("buf",215), ("phi",305), ("bal",247)]),
    syr=OrderedDict([("buf",150), ("phi",253), ("new",254), ("bos",312)]),
    bal=OrderedDict([("phi",101), ("pit",247)]),
    phi=OrderedDict([("pit",305), ("bal",101), ("syr",253), ("new",97)]),
    new=OrderedDict([("syr",254), ("phi",97), ("bos",215), ("pro",181)]),
    pro=OrderedDict([("bos",50), ("new",181)]),
    bos=OrderedDict([("pro",50), ("new",215), ("syr",312), ("por",107)]),
    por=OrderedDict([("bos",107)]))


map_times = dict(
```

	Test	Expected	Got	

✔	`path, cost = depth_first('chi','pit',map_distances,True)`
	`print(path)`
	`print(cost)`
		`['chi', 'ind', 'col', 'pit']`
		`543`
			`['chi', 'ind', 'col', 'pit']`
			`543`
				✔

✔	`path, cost = depth_first('ind','por',map_distances,True)`
	`print(path)`
	`print(cost)`
		`['ind', 'col', 'pit', 'phi', 'new', 'bos', 'por']`
		`1085`
			`['ind', 'col', 'pit', 'phi', 'new', 'bos', 'por']`
			`1085`
				✔

✔	`path, cost = depth_first('bal','det',map_distances,True)`
	`print(path)`
	`print(cost)`
		`['bal', 'pit', 'buf', 'det']`
		`718`
			`['bal', 'pit', 'buf', 'det']`
			`718`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **3**

Correct

Mark 10.00 out of 10.00

First, let's create our own Frontier_PQ class to represent the frontier (priority queue) for uniformcost search. Note that the heapq package is imported in the helpers at the bottom of this assignment; you may find that package useful. You could also use the Queue package. Your implementation of the uniform-cost search frontier should adhere to these specifications

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
import heapq

map_distances = dict(
    chi=dict(det=283, cle=345, ind=182),
    cle=dict(chi=345, det=169, col=144, pit=134, buf=189),
    ind=dict(chi=182, col=176),
    col=dict(ind=176, cle=144, pit=185),
    det=dict(chi=283, cle=169, buf=256),
    buf=dict(det=256, cle=189, pit=215, syr=150),
    pit=dict(col=185, cle=134, buf=215, phi=305, bal=247),
    syr=dict(buf=150, phi=253, new=254, bos=312),
    bal=dict(phi=101, pit=247),
    phi=dict(pit=305, bal=101, syr=253, new=97),
    new=dict(syr=254, phi=97, bos=215, pro=181),
    pro=dict(bos=50, new=181),
    bos=dict(pro=50, new=215, syr=312, por=107),
    por=dict(bos=107))


map_times = dict(
```

	Test	Expected	Got	

✔	`path, cost = uniform_cost('chi','pit',map_distances,True)`
	`print(path)`
	`print(cost)`
		`['chi', 'cle', 'pit']`
		`479`
			`['chi', 'cle', 'pit']`
			`479`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **4**

Correct

Mark 10.00 out of 10.00

Use each of your search functions to find routes for Neal to travel from New York to Chicago, with path costs defined by the distance between cities.

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
from collections import OrderedDict
import heapq
map_distances = dict(
    chi=OrderedDict([("det",283), ("cle",345), ("ind",182)]),
    cle=OrderedDict([("chi",345), ("det",169), ("col",144), ("pit",134), ("buf",189)]),
    ind=OrderedDict([("chi",182), ("col",176)]),
    col=OrderedDict([("ind",176), ("cle",144), ("pit",185)]),
    det=OrderedDict([("chi",283), ("cle",169), ("buf",256)]),
    buf=OrderedDict([("det",256), ("cle",189), ("pit",215), ("syr",150)]),
    pit=OrderedDict([("col",185), ("cle",134), ("buf",215), ("phi",305), ("bal",247)]),
    syr=OrderedDict([("buf",150), ("phi",253), ("new",254), ("bos",312)]),
    bal=OrderedDict([("phi",101), ("pit",247)]),
    phi=OrderedDict([("pit",305), ("bal",101), ("syr",253), ("new",97)]),
    new=OrderedDict([("syr",254), ("phi",97), ("bos",215), ("pro",181)]),
    pro=OrderedDict([("bos",50), ("new",181)]),
    bos=OrderedDict([("pro",50), ("new",215), ("syr",312), ("por",107)]),
    por=OrderedDict([("bos",107)]))

map_times = dict(
    chi=dict(det=280, cle=345, ind=200),
```

	Test	Expected	Got	

✔	`start = 'new'`
	`goal = 'chi'`
	`graph = map_distances`
	`print(uniform_cost(start, goal, graph, True))`
	`print(depth_first(start, goal, graph, True))`
	`print(breadth_first(start, goal, graph, True))`
		`(['new', 'phi', 'pit', 'cle', 'chi'], 881)`
		`(['new', 'phi', 'pit', 'buf', 'det', 'chi'], 1156)`
		`(['new', 'syr', 'buf', 'det', 'chi'], 943)`
			`(['new', 'phi', 'pit', 'cle', 'chi'], 881)`
			`(['new', 'phi', 'pit', 'buf', 'det', 'chi'], 1156)`
			`(['new', 'syr', 'buf', 'det', 'chi'], 943)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **5**

Correct

Mark 10.00 out of 10.00

Which algorithm yields the shortest path?

Select one:

a. Breadth First Search

b. Uniform Cost Search ✔

c. Depth First Search

Your answer is correct.

Question **6**

Correct

Mark 10.00 out of 10.00

Use your choice of search function to show the list of cities that Neal will traverse to get to Chicago on time, should such a path exist.

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
import heapq

map_distances = dict(
    chi=dict(det=283, cle=345, ind=182),
    cle=dict(chi=345, det=169, col=144, pit=134, buf=189),
    ind=dict(chi=182, col=176),
    col=dict(ind=176, cle=144, pit=185),
    det=dict(chi=283, cle=169, buf=256),
    buf=dict(det=256, cle=189, pit=215, syr=150),
    pit=dict(col=185, cle=134, buf=215, phi=305, bal=247),
    syr=dict(buf=150, phi=253, new=254, bos=312),
    bal=dict(phi=101, pit=247),
    phi=dict(pit=305, bal=101, syr=253, new=97),
    new=dict(syr=254, phi=97, bos=215, pro=181),
    pro=dict(bos=50, new=181),
    bos=dict(pro=50, new=215, syr=312, por=107),
    por=dict(bos=107))


map_times = dict(
```

	Test	Expected	Got	

✔	`start = 'new'`
	`goal = 'chi'`
	`graph = map_times`
	`print(uniform_cost(start, goal,`
	`graph, True))`
		`(['new', 'syr', 'buf',`
		`'cle', 'chi'], 935)`
			`(['new', 'syr', 'buf',`
			`'cle', 'chi'], 935)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **7**

Correct

Mark 10.00 out of 10.00

Use your choice of search function to show the list of cities that Neal would traverse to get to Chicago as quickly as possible.

Answer: (penalty regime: 0 %)

Reset answer

```
from collections import deque
import heapq

map_distances = dict(
    chi=dict(det=283, cle=345, ind=182),
    cle=dict(chi=345, det=169, col=144, pit=134, buf=189),
    ind=dict(chi=182, col=176),
    col=dict(ind=176, cle=144, pit=185),
    det=dict(chi=283, cle=169, buf=256),
    buf=dict(det=256, cle=189, pit=215, syr=150),
    pit=dict(col=185, cle=134, buf=215, phi=305, bal=247),
    syr=dict(buf=150, phi=253, new=254, bos=312),
    bal=dict(phi=101, pit=247),
    phi=dict(pit=305, bal=101, syr=253, new=97),
    new=dict(syr=254, phi=97, bos=215, pro=181),
    pro=dict(bos=50, new=181),
    bos=dict(pro=50, new=215, syr=312, por=107),
    por=dict(bos=107))


map_times = dict(
```

	Test	Expected	Got	

✔	`start = 'new'`
	`goal = 'chi'`
	`graph = map_times`
	`print(uniform_cost(start, goal,`
	`graph, True))`
		`(['new', 'syr', 'buf',`
		`'cle', 'chi'], 935)`
			`(['new', 'syr', 'buf',`
			`'cle', 'chi'], 935)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **8**

Correct

Mark 10.00 out of 10.00

Pass the maze-to-graph unit test.

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np

""" maze_to_graph - Converts a maze represented as a numpy array into a graph
    Input:
        maze - 2D numpy array where 0 represents walkable cells and 1 represents walls
    Algorithm:
        * Initialize an empty dictionary to represent the graph
        * Define the directions for North, South, East, and West movements
        * Iterate over each cell in the maze
            * Initialize an empty dictionary for each cell in the graph
            * For each direction, calculate the neighboring cell's coordinates
            * Check if the neighboring cell is within the maze bounds and is walkable (
                * If it is, add the neighbor to the current cell's dictionary in the gr
    Output:
        Returns a dictionary representing the graph where keys are coordinates of cells
        of neighboring cells with directions
"""
def maze_to_graph(maze):
    ''' takes in a maze as a numpy array, converts to a graph '''
    graph = {}
    rows, cols = maze.shape
```

	Test	Expected	Got	

✔	`testmaze = np.ones((4,4))`
	`testmaze[1,1] = 0`
	`testmaze[2,1] = 0`
	`testmaze[2,2] = 0`
	`testgraph = maze_to_graph(testmaze)`
	`print(testgraph[(1,1)])`
	`print(testgraph[(1,2)])`
	`print(testgraph[(2,2)])`
		`{(1, 2): 'N'}`
		`{(1, 1): 'S', (2, 2): 'E'}`
		`{(1, 2): 'W'}`
			`{(1, 2): 'N'}`
			`{(1, 1): 'S', (2, 2): 'E'}`
			`{(1, 2): 'W'}`
				✔

✔	`testmaze = np.array([[1, 1, 1, 1, 1],[1, 0, 0, 0, 1],[1, 0, 0, 0, 1],[1, 1, 1, 1, 1]])`
	`testgraph = maze_to_graph(testmaze)`
	`print(testgraph[(1,1)])`
	`print(testgraph[(1,2)])`
	`print(testgraph[(2,1)])`
	`print(testgraph[(2,2)])`
	`print(testgraph[(3,1)])`
	`print(testgraph[(3,2)])`
		`{(1, 2): 'N', (2, 1): 'E'}`
		`{(1, 1): 'S', (2, 2): 'E'}`
		`{(3, 1): 'E', (1, 1): 'W', (2, 2): 'N'}`
		`{(1, 2): 'W', (3, 2): 'E', (2, 1): 'S'}`
		`{(3, 2): 'N', (2, 1): 'W'}`
		`{(3, 1): 'S', (2, 2): 'W'}`
			`{(1, 2): 'N', (2, 1): 'E'}`
			`{(1, 1): 'S', (2, 2): 'E'}`
			`{(3, 1): 'E', (1, 1): 'W', (2, 2): 'N'}`
			`{(1, 2): 'W', (3, 2): 'E', (2, 1): 'S'}`
			`{(3, 2): 'N', (2, 1): 'W'}`
			`{(3, 1): 'S', (2, 2): 'W'}`
				✔

	Test	Expected	Got	

✔	`testmaze = np.array([[1, 1, 1, 1, 1, 1],[1, 0, 1, 1, 1, 1],[1, 0, 0, 0, 0, 1],[1, 1, 1, 1, 0, 1],[1, 1, 1, 1, 1, 1]])`
	`testgraph = maze_to_graph(testmaze)`
	`print(testgraph[(1,1)])`
	`print(testgraph[(1,2)])`
	`print(testgraph[(2,2)])`
	`print(testgraph[(3,2)])`
	`print(testgraph[(4,2)])`
	`print(testgraph[(4,3)])`
		`{(1, 2): 'N'}`
		`{(1, 1): 'S', (2, 2): 'E'}`
		`{(1, 2): 'W', (3, 2): 'E'}`
		`{(4, 2): 'E', (2, 2): 'W'}`
		`{(3, 2): 'W', (4, 3): 'N'}`
		`{(4, 2): 'S'}`
			`{(1, 2): 'N'}`
			`{(1, 1): 'S', (2, 2): 'E'}`
			`{(1, 2): 'W', (3, 2): 'E'}`
			`{(4, 2): 'E', (2, 2): 'W'}`
			`{(3, 2): 'W', (4, 3): 'N'}`
			`{(4, 2): 'S'}`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **9**

Correct

Mark 10.00 out of 10.00

Use your depth-first search function to solve the maze and provide the solution path

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np
from collections import OrderedDict
from collections import deque
maze = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                 [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1],
                 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
                 [1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1],
                 [1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
                 [1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1],
                 [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1],
                 [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
                 [1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1],
                 [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1],
                 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

map_distances = dict(
    chi=OrderedDict([("det",283), ("cle",345), ("ind",182)]),
    cle=OrderedDict([("chi",345), ("det",169), ("col",144), ("pit",134), ("buf",189)]),
    ind=OrderedDict([("chi",182), ("col",176)]),
    col=OrderedDict([("ind",176), ("cle",144), ("pit",185)]),
```

	Test	Expected	Got	

✔	`g_maze = maze_to_graph(maze)`
	
	`maze_sol_dfs = depth_first((1,1), (10,10), g_maze)`
	`print('Depth-first search yields: {}, ({} steps)'.format(maze_sol_dfs, len(maze_sol_dfs)))`
		`Depth-first search yields: [(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (8, 2), (8, 3), (9, 3), (10, 3), (10, 4), (10, 5), (9, 5), (8, 5), (7, 5), (6, 5), (6, 6), (6, 7), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (10, 9), (10, 10)], (27 steps)`	`Depth-first search yields: [(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (8, 2), (8, 3), (9, 3), (10, 3), (10, 4), (10, 5), (9, 5), (8, 5), (7, 5), (6, 5), (6, 6), (6, 7), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (10, 9), (10, 10)], (27 steps)`	✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

Question **10**

Correct

Mark 10.00 out of 10.00

Use your breadth-first search function to solve the maze and provide the solution path and its length

Answer: (penalty regime: 0 %)

Reset answer

```
import numpy as np
maze = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                 [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1],
                 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                 [1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1],
                 [1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1],
                 [1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1],
                 [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1],
                 [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1],
                 [1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1],
                 [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1],
                 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

##################
from collections import deque

map_distances = dict(
    chi=dict(det=283, cle=345, ind=182),
    cle=dict(chi=345, det=169, col=144, pit=134, buf=189),
    ind=dict(chi=182, col=176),
```

	Test	Expected	Got	

✔	`g_maze = maze_to_graph(maze)`
	
	`maze_sol_bfs = breadth_first((1,1), (10,10), g_maze)`
	`print('Breadth-first search yields: {} ({} steps)'.format(maze_sol_bfs, len(maze_sol_bfs)))`
		`Breadth-first search yields: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (3, 6), (3, 7), (4, 7), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (10, 9), (10, 10)] (19 steps)`	`Breadth-first search yields: [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (3, 6), (3, 7), (4, 7), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (10, 9), (10, 10)] (19 steps)`	✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.