Started on	Monday, 3 August 2020, 1:32 PM
State	Finished
Completed on	Sunday, 23 August 2020, 5:06 PM
Time taken	20 days 3 hours
Marks	3.00/3.00
Grade	**100.00** out of 100.00

Information

Introduction

In this assignment you apply the search techniques you have learnt in the course to a routing problem. There are a number of mobile agents (think of self-driving taxis) scattered across a flat rectangular grid environment. There are also a number of call points (think of customers) waiting to be served. The objective is to, if possible, navigate an agent to a call point that are time-wise closest to each other.

Technical notes

The following points apply to all the questions in this super quiz.

1. In each instance of this problem we want to navigate one agent to one call point (not all agent and not all call points).
2. In all the questions you can assume that the search module (the file search.py) is available on the quiz server. This means that you can safely import the search module (in addition to all the standard Python modules) in your program. Do **not** repeat the code of search module in your answers.
3. Your solution must contain all the import statements that it requires, even for the search module.
4. You can have your entire program for all the three questions in one file and every time you want to submit your answer you can simply paste the content of the entire file in the answer box. Just make sure that you don't have any function calls in the global section of your file that may interfere with auto-grading. You can put all of your own test cases and calls to the print function in a main function and then have a statement like the following at the global level:

```
if __name__ == "__main__":
    main()
```

5. Answer the questions in order. The answer to some questions require your answer to earlier questions (to use or to build on).
6. Your answer to a question may pass all the test cases for the question (and receive full mark) but it may not have all the functionalities that are needed for later questions or it may a bug that is not detected by our test cases!

Question **1**

Correct

Mark 1.00 out of 1.00

Writing the RoutingGraph class

In the first step of the assignment you have to write a subclass of `Graph` for a routing problem in an environment. The map of the environment is given in the form of a multi-line string of characters. The following shows an example map.

```
map_str = """\
+--------+
|  G    G|
|  XXX   |
|  S X   |
|    X 2 |
+--------+
"""
```

Map description

- The map is always rectangular. We refer to positions in the map by a pair of numbers *(row, col)* which correspond to the row and column number of the position. Row numbers start from 0 for the first (topmost) line, 1 for the second line, and so on. The column numbers start from 0 for the first (leftmost) position (character) in the line, 1 for the second position, and so on.
- The environment is always surrounded by walls which are represented with characters `'+'` or `'-'` or `'|'`. For example the position *(0,0)* is always a '+' (i.e. wall) and so are all other three corners of the map. The first and last rows and the first and last columns are always `'-'` and `'|'` respectively (except for the corners).
- The obstacles are marked by `'X'`.
- There may be zero or more agents on the map. The location of agents are marked by `'S'` or digits `0..9`.
  - Agents indicated by `S` have solar panels and never require fueling. We can think of their fuel capacity to be infinity.
  - Agents indicated by digits have fuel tanks. The capacity of the tank is 9. The digit used to indicate the agent shows how much fuel is initially available in the tank.
- There may be zero or more call points (customers) on the map. The location of call points (potential destinations) are marked by `'G'`. To simplify textual representation, we assume that an agent is never initially at a call point.
- An agent can move in four directions, N, E, S, W, as long as it has fuel and there is no obstacle or wall in the way. This means that agents can also go to cells where other agents are present. The agent loses one unit fuel for each move. The order of actions is clockwise starting from N. For example, if from a position all four directional moves are possible, then the first arc in the sequence of arcs returned by `outgoing_arcs` is for going north, then east, and so on until the last arc which goes to west. All single directional moves take 5 units of time.
- If an agent is in a cell marked as `F` and its current fuel amount is less than 9 it can take the action of `"Fuel up"` which fills the tank to its maximum capacity of 9. In the sequence of arcs, the `"Fuel up"` action (if available) should appear after any other directional actions. The action costs 15 units of time (regardless of how much fuel is obtained).

Task

Write a class `RoutingGraph` which is initialised by the map string and represents the map in the form of a graph by implementing all the required methods in the `Graph` class including the method `estimated_cost_to_goal`. Represent the state of the agent by a tuple of the form `(row, column, fuel)`

Notes

1. It is recommended that you write your class as a subclass of `Graph`.
2. The test cases do not test the method `estimated_cost_to_goal` in this question.
3. Try to avoid using indices to refer to elements of a tuple. For example instead of using `position[0]` and `position[1]`, use readable names. For example use `row, col = position` instead.
4. You may find `math.inf` useful.
5. If you copy a multi-line string from the examples given in the questions into a function in your code, some leading spaces will be introduced. Use the method `strip` to get rid of these leading/trailing spaces. Also be mindful of the difference between the following two strings:

   ```
   str1 = """This string splits into one line.
   """
   def main():
       str2 = """This string splits into TWO lines.
       """
   ```

6. It is recommended (but not required) that your answer for `RoutingGraph` is shorter than 70 lines of code. If your code is much longer, you might be doing something wrong.
7. Avoid repetitive code. If you wish you can use the following list when implementing `outgoing_arcs`. You have to decipher it yourself.

   ```
   [('N' , -1, 0),
    ('E' ,  0, 1),
    ('S' ,  1, 0),
    ('W' ,  0, -1),]
   ```

For example:

Test	Result

Test	Result

from student_answer import RoutingGraph
import math
 	
map_str = """\
+-------+
| 9 XG|
|X XXX |
| S 0FG|
+-------+
"""
 	
graph = RoutingGraph(map_str)
 	
print("Starting nodes:",
sorted(graph.starting_nodes()))
print("Outgoing arcs (available actions) at
starting states:")
for s in sorted(graph.starting_nodes()):
    print(s)
    for arc in graph.outgoing_arcs(s):
        print (" " + str(arc))
 	
node = (1,1,5)
print("\nIs {} goal?".format(node),
graph.is_goal(node))
print("Outgoing arcs (available actions) at
{}:".format(node))
for arc in graph.outgoing_arcs(node):
    print (" " + str(arc))
 	
node = (1,7,2)
print("\nIs {} goal?".format(node),
graph.is_goal(node))
print("Outgoing arcs (available actions) at
{}:".format(node))
for arc in graph.outgoing_arcs(node):
    print (" " + str(arc))
 	
node = (3, 7, 0)
print("\nIs {} goal?".format(node),
graph.is_goal(node))
 	
node = (3, 7, math.inf)
print("\nIs {} goal?".format(node),
graph.is_goal(node))
 	
node = (3,6,5)
print("\nIs {} goal?".format(node),
graph.is_goal(node))
print("Outgoing arcs (available actions) at
{}:".format(node))
for arc in graph.outgoing_arcs(node):
    print (" " + str(arc))
 	
node = (3,6,9)
print("\nIs {} goal?".format(node),
graph.is_goal(node))
print("Outgoing arcs (available actions) at
{}:".format(node))
for arc in graph.outgoing_arcs(node):
    print (" " + str(arc))
	Starting nodes: [(1, 3, 9), (3, 2, inf), (3, 5,
	0)]
	Outgoing arcs (available actions) at starting
	states:
	(1, 3, 9)
	  Arc(tail=(1, 3, 9), head=(1, 4, 8), action='E',
	cost=5)
	  Arc(tail=(1, 3, 9), head=(1, 2, 8), action='W',
	cost=5)
	(3, 2, inf)
	  Arc(tail=(3, 2, inf), head=(2, 2, inf),
	action='N', cost=5)
	  Arc(tail=(3, 2, inf), head=(3, 3, inf),
	action='E', cost=5)
	  Arc(tail=(3, 2, inf), head=(3, 1, inf),
	action='W', cost=5)
	(3, 5, 0)
	
	Is (1, 1, 5) goal? False
	Outgoing arcs (available actions) at (1, 1, 5):
	  Arc(tail=(1, 1, 5), head=(1, 2, 4), action='E',
	cost=5)
	
	Is (1, 7, 2) goal? True
	Outgoing arcs (available actions) at (1, 7, 2):
	  Arc(tail=(1, 7, 2), head=(2, 7, 1), action='S',
	cost=5)
	
	Is (3, 7, 0) goal? True
	
	Is (3, 7, inf) goal? True
	
	Is (3, 6, 5) goal? False
	Outgoing arcs (available actions) at (3, 6, 5):
	  Arc(tail=(3, 6, 5), head=(2, 6, 4), action='N',
	cost=5)
	  Arc(tail=(3, 6, 5), head=(3, 7, 4), action='E',
	cost=5)
	  Arc(tail=(3, 6, 5), head=(3, 5, 4), action='W',
	cost=5)
	  Arc(tail=(3, 6, 5), head=(3, 6, 9), action='Fuel
	up', cost=15)
	
	Is (3, 6, 9) goal? False
	Outgoing arcs (available actions) at (3, 6, 9):
	  Arc(tail=(3, 6, 9), head=(2, 6, 8), action='N',
	cost=5)
	  Arc(tail=(3, 6, 9), head=(3, 7, 8), action='E',
	cost=5)
	  Arc(tail=(3, 6, 9), head=(3, 5, 8), action='W',
	cost=5)

Test	Result

`from student_answer import RoutingGraph`
 	
`map_str = """\`
`+--+`
`|GS|`
`+--+`
`"""`
 	
`graph = RoutingGraph(map_str)`
 	
`print("Starting nodes:",`
`sorted(graph.starting_nodes()))`
`print("Outgoing arcs (available actions) at the start:")`
`for start in graph.starting_nodes():`
`  for arc in graph.outgoing_arcs(start):`
`  print ("  " + str(arc))`
 	
`node = (1,1,1)`
`print("\nIs {} goal?".format(node),`
`graph.is_goal(node))`
`print("Outgoing arcs (available actions) at {}:".format(node))`
`for arc in graph.outgoing_arcs(node):`
`  print ("  " + str(arc))`
	`Starting nodes: [(1, 2, inf)]`
	`Outgoing arcs (available actions) at the start:`
	`  Arc(tail=(1, 2, inf), head=(1, 1, inf), action='W', cost=5)`
	
	`Is (1, 1, 1) goal? True`
	`Outgoing arcs (available actions) at (1, 1, 1):`
	`  Arc(tail=(1, 1, 1), head=(1, 2, 0), action='E', cost=5)`

`from student_answer import RoutingGraph`
 	
`map_str = """\`
`+------+`
`|S  S|`
`|  GXXX|`
`|S  |`
`+------+`
`"""`
 	
`graph = RoutingGraph(map_str)`
`print("Starting nodes:",`
`sorted(graph.starting_nodes()))`
	`Starting nodes: [(1, 1, inf), (1, 6, inf), (3, 1, inf)]`

Answer: (penalty regime: 0, 15, ... %)

```
import math
from search import *

class RoutingGraph(Graph):
    """This is a concrete subclass of Graph where vertices and edges
     are explicitly enumerated. Objects of this type are useful for
     testing graph algorithms."""

    def __init__(self, map_str):
        """Initialises an Routing graph."""

        self.goal_nodes = []
        self.start_nodes = []
        self.fuel_points = []
        self.obstacle = []
        self.estimated_cost_to_goal = 0

        map_str.strip()
        map_array = map_str.split('\n')
        #print(map_array)
        for row in range(len(map_array)):
            for col in range(len(map_array[row])):
                #blank space so no need save anything
                if (map_array[row][col] == ' '):
                    pass
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '+'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == 'X'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '-'):
                    self.obstacle.append((row, col))
```

```
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '|'):
                    self.obstacle.append((row, col))
                #its a goal node so append to goal node set
                elif (map_array[row][col] == 'G'):
                    self.goal_nodes.append((row, col))
                #its a solar powered agent so append it with infinite cost
                elif (map_array[row][col] == 'S'):
                    self.start_nodes.append((row, col, math.inf))
                #its a fuel point does not need cost just +15 if choose to top up
                elif (map_array[row][col] == 'F'):
                    self.fuel_points.append((row, col))
                #only other case is that its from 0-9
                else:
                    self.start_nodes.append((row, col, int(map_array[row][col])))
        #print("obstacles are \n")
        #print(self.obstacle)
        #print("STARTING NODES ARE are \n")
        #print(self.start_nodes)


    #def estimated_cost_to_goal(self,


    def starting_nodes(self):
        """Returns a sequence of starting nodes."""
        return self.start_nodes

    def is_goal(self, node):
        """Returns true if the given node is a goal node."""
        #print("IS GOAL?")
        node_x, node_y, _ = node
        return (node_x, node_y) in self.goal_nodes

    def estimated_cost_to_goal(self):
        return 0

    def outgoing_arcs(self, node):
        """Returns a sequence of Arc objects that go out from the given
        node. The action string is automatically generated.

        """
        arcs = []
        movement_grid = [('N' , -1, 0),
                         ('E' ,  0, 1),
                         ('S' ,  1, 0),
                         ('W' ,  0, -1),]
        #for every edge in list of edges
        for directional_movement in movement_grid:
            #get all possible movements
            #eleminate those arcs which move into obstacles
            #append allowed movement arcs
            #print("Start node is:")
            #print(node)
            for i in range(len(movement_grid)):
                direction, horizontal, vertical =  movement_grid[i]
                new_node_location = (node[0] + horizontal, node[1]+ vertical, node[2] - 1)
                #if fuel is below 1 cant do action, cancel this iteration
                if (new_node_location[2] < 0):
                    continue
                #extract x and y coordinates
                new_node_x, new_node_y, _ = new_node_location
                #if i am crashing into obstacle, cancel this iteration
                if (new_node_x, new_node_y) in self.obstacle:
                    continue
                else:
                    #print("direction, horizontal, vertical is\n")
                    #print(direction, horizontal, vertical)
                    arcs.append(Arc(node, new_node_location, str(direction), 5))

            if ((node[0], node[1]) in self.fuel_points):
                #i can only stay and top up fuel if i am not on a full tank
                if (node[2] < 9):
                    arcs.append(Arc(node, (node[0], node[1], 9), "Fuel up", 15))
                    #print(arcs)
            return arcs
```

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph`
	`import math`
	
	`map_str = """\`
	`+-------+`
	`|  9  XG|`
	`|X XXX  |`
	`| S  0FG|`
	`+-------+`
	`"""`
	
	`graph = RoutingGraph(map_str)`
	
	`print("Starting nodes:", sorted(graph.starting_nodes()))`
	`print("Outgoing arcs (available actions) at starting states:")`
	`for s in sorted(graph.starting_nodes()):`
	`    print(s)`
	`    for arc in graph.outgoing_arcs(s):`
	`        print ("  " + str(arc))`
	
	`node = (1,1,5)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	`print("Outgoing arcs (available actions) at {}:".format(node))`
	`for arc in graph.outgoing_arcs(node):`
	`    print ("  " + str(arc))`
	
	`node = (1,7,2)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	`print("Outgoing arcs (available actions) at {}:".format(node))`
	`for arc in graph.outgoing_arcs(node):`
	`    print ("  " + str(arc))`
	
	`node = (3, 7, 0)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	
	`node = (3, 7, math.inf)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	
	`node = (3,6,5)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	`print("Outgoing arcs (available actions) at {}:".format(node))`
	`for arc in graph.outgoing_arcs(node):`
	`    print ("  " + str(arc))`
	
	`node = (3,6,9)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	`print("Outgoing arcs (available actions) at {}:".format(node))`
	`for arc in graph.outgoing_arcs(node):`
	`    print ("  " + str(arc))`
		`Starting nodes: [(1, 3, 9), (3, 2, inf), (3, 5, 0)]`
		`Outgoing arcs (available actions) at starting states:`
		`(1, 3, 9)`
		`  Arc(tail=(1, 3, 9), head=(1, 4, 8), action='E', cost=5)`
		`  Arc(tail=(1, 3, 9), head=(1, 2, 8), action='W', cost=5)`
		`(3, 2, inf)`
		`  Arc(tail=(3, 2, inf), head=(2, 2, inf), action='N', cost=5)`
		`  Arc(tail=(3, 2, inf), head=(3, 3, inf), action='E', cost=5)`
		`  Arc(tail=(3, 2, inf), head=(3, 1, inf), action='W', cost=5)`
		`(3, 5, 0)`
		
		`Is (1, 1, 5) goal? False`
		`Outgoing arcs (available actions) at (1, 1, 5):`
		`  Arc(tail=(1, 1, 5), head=(1, 2, 4), action='E', cost=5)`
		
		`Is (1, 7, 2) goal? True`
		`Outgoing arcs (available actions) at (1, 7, 2):`
		`  Arc(tail=(1, 7, 2), head=(2, 7, 1), action='S', cost=5)`
		
		`Is (3, 7, 0) goal? True`
		
		`Is (3, 7, inf) goal? True`
		
		`Is (3, 6, 5) goal? False`
		`Outgoing arcs (available actions) at (3, 6, 5):`
		`  Arc(tail=(3, 6, 5), head=(2, 6, 4), action='N', cost=5)`
		`  Arc(tail=(3, 6, 5), head=(3, 7, 4), action='E', cost=5)`
		`  Arc(tail=(3, 6, 5), head=(3, 5, 4), action='W', cost=5)`
		`  Arc(tail=(3, 6, 5), head=(3, 6, 9), action='Fuel up', cost=15)`
		
		`Is (3, 6, 9) goal? False`
		`Outgoing arcs (available actions) at (3, 6, 9):`
		`  Arc(tail=(3, 6, 9), head=(2, 6, 8), action='N', cost=5)`
		`  Arc(tail=(3, 6, 9), head=(3, 7, 8), action='E', cost=5)`
		`  Arc(tail=(3, 6, 9), head=(3, 5, 8), action='W', cost=5)`
			`Starting nodes: [(1, 3, 9), (3, 2, inf), (3, 5, 0)]`
			`Outgoing arcs (available actions) at starting states:`
			`(1, 3, 9)`
			`  Arc(tail=(1, 3, 9), head=(1, 4, 8), action='E', cost=5)`
			`  Arc(tail=(1, 3, 9), head=(1, 2, 8), action='W', cost=5)`
			`(3, 2, inf)`
			`  Arc(tail=(3, 2, inf), head=(2, 2, inf), action='N', cost=5)`
			`  Arc(tail=(3, 2, inf), head=(3, 3, inf), action='E', cost=5)`
			`  Arc(tail=(3, 2, inf), head=(3, 1, inf), action='W', cost=5)`
			`(3, 5, 0)`
			
			`Is (1, 1, 5) goal? False`
			`Outgoing arcs (available actions) at (1, 1, 5):`
			`  Arc(tail=(1, 1, 5), head=(1, 2, 4), action='E', cost=5)`
			
			`Is (1, 7, 2) goal? True`
			`Outgoing arcs (available actions) at (1, 7, 2):`
			`  Arc(tail=(1, 7, 2), head=(2, 7, 1), action='S', cost=5)`
			
			`Is (3, 7, 0) goal? True`
			
			`Is (3, 7, inf) goal? True`
			
			`Is (3, 6, 5) goal? False`
			`Outgoing arcs (available actions) at (3, 6, 5):`
			`  Arc(tail=(3, 6, 5), head=(2, 6, 4), action='N', cost=5)`
			`  Arc(tail=(3, 6, 5), head=(3, 7, 4), action='E', cost=5)`
			`  Arc(tail=(3, 6, 5), head=(3, 5, 4), action='W', cost=5)`
			`  Arc(tail=(3, 6, 5), head=(3, 6, 9), action='Fuel up', cost=15)`
			
			`Is (3, 6, 9) goal? False`
			`Outgoing arcs (available actions) at (3, 6, 9):`
			`  Arc(tail=(3, 6, 9), head=(2, 6, 8), action='N', cost=5)`
			`  Arc(tail=(3, 6, 9), head=(3, 7, 8), action='E', cost=5)`
			`  Arc(tail=(3, 6, 9), head=(3, 5, 8), action='W', cost=5)`
				✔	

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph`
	
	`map_str = """\`
	`+--+`
	`|GS|`
	`+--+`
	`"""`
	
	`graph = RoutingGraph(map_str)`
	
	`print("Starting nodes:", sorted(graph.starting_nodes()))`
	`print("Outgoing arcs (available actions) at the start:")`
	`for start in graph.starting_nodes():`
	`  for arc in graph.outgoing_arcs(start):`
	`  print ("  " + str(arc))`
	
	`node = (1,1,1)`
	`print("\nIs {} goal?".format(node), graph.is_goal(node))`
	`print("Outgoing arcs (available actions) at {}:".format(node))`
	`for arc in graph.outgoing_arcs(node):`
	`  print ("  " + str(arc))`
		`Starting nodes: [(1, 2, inf)]`
		`Outgoing arcs (available actions) at the start:`
		`  Arc(tail=(1, 2, inf), head=(1, 1, inf), action='W', cost=5)`
		
		`Is (1, 1, 1) goal? True`
		`Outgoing arcs (available actions) at (1, 1, 1):`
		`  Arc(tail=(1, 1, 1), head=(1, 2, 0), action='E', cost=5)`
			`Starting nodes: [(1, 2, inf)]`
			`Outgoing arcs (available actions) at the start:`
			`  Arc(tail=(1, 2, inf), head=(1, 1, inf), action='W', cost=5)`
			
			`Is (1, 1, 1) goal? True`
			`Outgoing arcs (available actions) at (1, 1, 1):`
			`  Arc(tail=(1, 1, 1), head=(1, 2, 0), action='E', cost=5)`
				✔

✔	`from student_answer import RoutingGraph`
	
	`map_str = """\`
	`+----+`
	`| X  |`
	`|XSX |`
	`| X G|`
	`+----+`
	`"""`
	
	`graph = RoutingGraph(map_str)`
	
	`print("Starting nodes:", sorted(graph.starting_nodes()))`
	`print("Available actions at the start:")`
	`for s in graph.starting_nodes():`
	`  for arc in graph.outgoing_arcs(s):`
	`  print ("  " + arc.action)`
		`Starting nodes: [(2, 2, inf)]`
		`Available actions at the start:`
			`Starting nodes: [(2, 2, inf)]`
			`Available actions at the start:`
				✔

✔	`from student_answer import RoutingGraph`
	
	`map_str = """\`
	`+------+`
	`|S  S|`
	`|  GXXX|`
	`|S  |`
	`+------+`
	`"""`
	
	`graph = RoutingGraph(map_str)`
	`print("Starting nodes:", sorted(graph.starting_nodes()))`
		`Starting nodes: [(1, 1, inf), (1, 6, inf), (3, 1, inf)]`	`Starting nodes: [(1, 1, inf), (1, 6, inf), (3, 1, inf)]`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Writing the AStarFrontier class

Write a class `AStarFrontier` for performing A* search on graphs. An instance of `AStarFrontier` together with an instance of `RoutingGraph` that you wrote in the previous question will be passed to the generic search procedure in order to find the lowest cost solution (if one exists) from one of the agents to the goal node.

In this question, the heuristic value returned by a graph object for any node (state) must be **zero**.

Notes:

1. Your solution must contain the definitions of both `AStarFrontier` and `RoutingGraph` classes.
2. Unlike other frontier objects, the `AStarFrontier` objects are initialised with an instance of a graph. This is because `AStarFrontier` needs to access the `estimated_cost_to_goal` method of the graph object.
3. Remember that in this course priority queues must be stable. See priority queue implementation notes in heapq documentation for a suggestion on how this can be achieved.
4. The algorithm must halt on all valid maps even when there is no solution.
5. It is recommended that before you submit your solution, you test it against the given test cases and some new examples (maps) designed by yourself with different positioning of agents and building blocks. You should think whether or not the frontier requires pruning and implement it accordingly.
6. Note the distinction between time and fuel consumption. We are interested in a solution with the shortest time.
7. The requirement of having a zero heuristic implies that the generic graph search algorithm will behave as an LCFS (lowest-cost-first search). In the next question, we will more thoroughly test your A* frontier and graph class.
8. When there is no solution, the generic graph search automatically returns `None` instead of a path which causes the `print_actions` procedure to print "There is no solution." This happens when the frontier becomes empty and no solution has been reached.
9. It is recommended (but not required) that your answer for `AStarFrontier` is shorter than 40 lines of code. If your code is much longer, you might be doing something wrong.

For example:

Test	Result

`from student_answer import RoutingGraph, AStarFrontier`
`from search import *`
 	
`map_str = """\`
`+-------+`
`|   G   |`
`|       |`
`|   S   |`
`+-------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_actions(solution)`
	`Actions:`
	`  N,`
	`  N.`
	`Total cost: 10`

`from student_answer import RoutingGraph, AStarFrontier`
`from search import *`
 	
`map_str = """\`
`+-------+`
`|  GG   |`
`|S    G |`
`|  S    |`
`+-------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_actions(solution)`
	`Actions:`
	`  N,`
	`  N.`
	`Total cost: 10`

Test	Result

from student_answer import RoutingGraph, AStarFrontier
from search import *
 	
map_str = """\
+-------+
| XG|
|X XXX |
| S |
+-------+
"""
 	
map_graph = RoutingGraph(map_str)
frontier = AStarFrontier(map_graph)
solution = next(generic_search(map_graph, frontier), None)
print_actions(solution)
	Actions:
	  E,
	  E,
	  E,
	  E,
	  N,
	  E,
	  N.
	Total cost: 35

from student_answer import RoutingGraph, AStarFrontier
from search import *
 	
map_str = """\
+-------+
| F X |
|X XXXXG|
| 3 |
+-------+
"""
 	
map_graph = RoutingGraph(map_str)
frontier = AStarFrontier(map_graph)
solution = next(generic_search(map_graph, frontier), None)
print_actions(solution)
	Actions:
	  N,
	  N,
	  E,
	  Fuel up,
	  W,
	  S,
	  S,
	  E,
	  E,
	  E,
	  E,
	  E,
	  N.
	Total cost: 75

from student_answer import RoutingGraph, AStarFrontier from search import * map_str = """\ +--+ |GS| +--+ """ map_graph = RoutingGraph(map_str) frontier = AStarFrontier(map_graph) solution = next(generic_search(map_graph, frontier), None) print_actions(solution)	Actions: W. Total cost: 5

from student_answer import RoutingGraph, AStarFrontier
from search import *
 	
map_str = """\
+---+
|GF2|
+---+
"""
map_graph = RoutingGraph(map_str)
frontier = AStarFrontier(map_graph)
solution = next(generic_search(map_graph, frontier), None)
print_actions(solution)
	Actions:
	  W,
	  W.
	Total cost: 10

from student_answer import RoutingGraph, AStarFrontier
from search import *
 	
map_str = """\
+----+
| S |
| SX |
|GX G|
+----+
"""
 	
map_graph = RoutingGraph(map_str)
frontier = AStarFrontier(map_graph)
solution = next(generic_search(map_graph, frontier), None)
print_actions(solution)
	Actions:
	  W,
	  S.
	Total cost: 10

Test	Result

`from student_answer import RoutingGraph, AStarFrontier`
`from search import *`
 	
`map_str = """\`
`+---------+`
`|         |`
`|    G    |`
`|         |`
`+---------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_actions(solution)`
	There is no solution!

Answer: (penalty regime: 0, 15, ... %)

```
import math
from search import *
import heapq
import numpy as np

class AStarFrontier(Frontier):
    """A* Frontier class but all heuristics node value is zero
    """

    def __init__(self, graph_map):
        self.graph_map = graph_map
        self.pruned = []
        self.heap = []
        self.entry = 0


    def add(self, path):
        """Adds a new path to the frontier. A path is a sequence (tuple) of
        Arc objects.
        """
        #if it is a starting node, it should have no cost
        if path[-1].tail == None:
            totalCost = 0 + 0

        else:
            #totalCost = 0
            #print("path is")
            #print(path)
            #for arc in path:
                #totalCost += arc.cost
            totalCost = sum(arc.cost for arc in path)

        if path[-1].head not in self.pruned:
            #print(path[-1].head)
            #print("path[-1].head here")
            self.entry += 1
            heapq.heappush(self.heap,(totalCost, ) + (self.entry, ) + path)
            #print(self.heap)
            #print("heap here")
        #print("path to be appended is\n")
        #print(path)
        #print("\n")
        #self.heap.append(path)


    def __iter__(self):
        """We don't need a separate iterator object. Just return self. You
        don't need to change this method."""
        return self


    def __next__(self):

        """Selects, removes, and returns a path on the frontier if there is
        any.Recall that a path is a sequence (tuple) of Arc
        objects. Override this method to achieve a desired search
        strategy. If there nothing to return this should raise a
        StopIteration exception.

        """
        while len(self.heap) > 0:
            path = heapq.heappop(self.heap)
```

```
            path = heapq.heappop(self.heap)
            if path[-1].head not in self.pruned:
                self.pruned.append(path[-1].head)
                return path[2:]
        raise StopIteration



def print_actions(path):
    """Given a path (a sequence of Arc objects), prints the actions that
    need to be taken and the total cost of those actions. The path is
    usually a solution (a path from the starting node to a goal
    node."""

    if path:
        print("Actions:")
        print(",\n".join("  {}".format(arc.action) for arc in path[1:]) + ".")
        print("Total cost:", sum(arc.cost for arc in path))
    else:
        print("There is no solution!")









class RoutingGraph(Graph):
    """This is a concrete subclass of Graph where vertices and edges
     are explicitly enumerated. Objects of this type are useful for
     testing graph algorithms."""

    def __init__(self, map_str):
        """Initialises an Routing graph."""

        self.goal_nodes = []
        self.start_nodes = []
        self.fuel_points = []
        self.obstacle = []
        self.estimated_cost_to_goal = 0

        map_str.strip()
        map_array = map_str.split('\n')
        #print(map_array)
        for row in range(len(map_array)):
            for col in range(len(map_array[row])):
                #blank space so no need save anything
                if (map_array[row][col] == ' '):
                    pass
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '+'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == 'X'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '-'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '|'):
                    self.obstacle.append((row, col))
                #its a goal node so append to goal node set
                elif (map_array[row][col] == 'G'):
                    self.goal_nodes.append((row, col))
                #its a solar powered agent so append it with infinite cost
                elif (map_array[row][col] == 'S'):
                    self.start_nodes.append((row, col, math.inf))
                #its a fuel point does not need cost just +15 if choose to top up
                elif (map_array[row][col] == 'F'):
                    self.fuel_points.append((row, col))
                #only other case is that its from 0-9
                else:
                    self.start_nodes.append((row, col, int(map_array[row][col])))
        #print("obstacles are \n")
        #print(self.obstacle)
        #print("STARTING NODES ARE are \n")
        #print(self.start_nodes)
```

```
149
150
151     #def estimated_cost_to_goal(self,
152
153
154     def starting_nodes(self):
155         """Returns a sequence of starting nodes."""
156         return self.start_nodes
157
158     def is_goal(self, node):
159         """Returns true if the given node is a goal node."""
160         #print("IS GOAL?")
161         node_x, node_y, _ = node
162         return (node_x, node_y) in self.goal_nodes
163
164     def estimated_cost_to_goal(self):
165         return 0
166
167     def outgoing_arcs(self, node):
168         """Returns a sequence of Arc objects that go out from the given
169         node. The action string is automatically generated.
170
171         """
172         arcs = []
173         movement_grid = [('N' , -1, 0),
174                          ('E' ,  0, 1),
175                          ('S' ,  1, 0),
176                          ('W' ,  0, -1),]
177         #for every edge in list of edges
178         for directional_movement in movement_grid:
179             #get all possible movements
180             #eleminate those arcs which move into obstacles
181             #append allowed movement arcs
182             #print("Start node is:")
183             #print(node)
184             for i in range(len(movement_grid)):
185                 direction, horizontal, vertical =  movement_grid[i]
186                 new_node_location = (node[0] + horizontal, node[1]+ vertical, node[2] - 1)
187                 #if fuel is below 1 cant do action, cancel this iteration
188                 if (new_node_location[2] < 0):
189                     continue
190                 #extract x and y coordinates
191                 new_node_x, new_node_y, _ = new_node_location
192                 #if i am crashing into obstacle, cancel this iteration
193                 if (new_node_x, new_node_y) in self.obstacle:
194                     continue
195                 else:
196                     #print("direction, horizontal, vertical is\n")
197                     #print(direction, horizontal, vertical)
198                     arcs.append(Arc(node, new_node_location, str(direction), 5))
199
200             if ((node[0], node[1]) in self.fuel_points):
201                 #i can only stay and top up fuel if i am not on a full tank
202                 if (node[2] < 9):
203                     arcs.append(Arc(node, (node[0], node[1], 9), "Fuel up", 15))
204                     #print(arcs)
205             return arcs
```

	Test	Expected	Got	

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+-------+`
	`|   G   |`
	`|       |`
	`|   S   |`
	`+-------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		N,
		N.
		Total cost: 10
			Actions:
			N,
			N.
			Total cost: 10
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+-------+`
	`|  GG   |`
	`|S    G |`
	`|  S    |`
	`+-------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		N,
		N.
		Total cost: 10
			Actions:
			N,
			N.
			Total cost: 10
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+-------+`
	`|     XG|`
	`|X XXX  |`
	`| S     |`
	`+-------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		E,
		E,
		E,
		E,
		N,
		E,
		N.
		Total cost: 35
			Actions:
			E,
			E,
			E,
			E,
			N,
			E,
			N.
			Total cost: 35
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+-------+`
	`|  F  X |`
	`|X XXXXG|`
	`| 3     |`
	`+-------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		N,
		N,
		E,
		Fuel up,
		W,
		S,
		S,
		E,
		E,
		E,
		E,
		E,
		N.
		Total cost: 75
			Actions:
			N,
			N,
			E,
			Fuel up,
			W,
			S,
			S,
			E,
			E,
			E,
			E,
			E,
			N.
			Total cost: 75
				✔

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+--+`
	`|GS|`
	`+--+`
	`"""`
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		W.
		Total cost: 5
			Actions:
			W.
			Total cost: 5
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+---+`
	`|GF2|`
	`+---+`
	`"""`
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		W,
		W.
		Total cost: 10
			Actions:
			W,
			W.
			Total cost: 10
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+----+`
	`| S  |`
	`| SX |`
	`|GX G|`
	`+----+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		Actions:
		W,
		S.
		Total cost: 10
			Actions:
			W,
			S.
			Total cost: 10
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+---------+`
	`|         |`
	`|    G    |`
	`|         |`
	`+---------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		There is no solution!	There is no solution!	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

The print_map procedure

Write a procedure `print_map` that takes three parameters: an instance of `RoutingGraph`, a frontier object, and a solution (which is a sequence of `Arc` objects that make up a path from a starting position to the goal position) and then prints a map such that:

- the position of the walls, obstacles, agents, and the goal points are all unchanged and they are marked by the same set of characters as in the original map string; and
- those free spaces (space characters) that have been expanded during the search are marked with a `'.'` (a period character); and
- those free spaces (spaces characters) that are part of the solution (best path to the goal) are marked with `'*'` (an asterisk character).

Further assumptions and requirements

1. For this question, the graph class must have a proper heuristic function named `estimated_cost_to_goal`. You have to design the most dominant (highest value) function that can be computed very efficiently. See the signature of the method in the `Graph` class in `search.py`.
2. In this question, we are only concerned with agents of type `S`—agents that have infinite amount of fuel and do not require to fuel up. The test cases do not include any fuel-based agents or fuel stations. Do not consider anything fuel-related when devising the heuristic function.
3. Only the first solution returned by the generic search procedure (if there is one) is used to test your procedure.
4. In addition to `print_map`, your solution must include the code for `RoutingGraph` and `AStarFrontier`.

Notes

1. A node is said to have been *expanded* if a path leading to that node is removed from the frontier and then if the node has neighbours, corresponding extended paths are added to the frontier.
2. This question puts your code for the graph and A* frontier into real test. Previous questions did not test the heuristic function and as long as the frontier class could provide the functionality of the LCFS, it would pass the test cases. In this question, however, your code needs to produce the correct A* behaviour. Therefore, even if your code has passed previous tests, you may still need to modify it in order to meet the required spec in this question.
3. Note that you only need to consider movement actions when designing the heuristic function and that all movement actions have the same cost.
4. If your algorithm expands more nodes than the expected output, you might be using a heuristic that is not good enough; you need to find a better heuristic. Refer to the specification of the heuristic function stated above.
5. If for any reason you decide to answer this question before Question 2, please remember that in Question 2 the heuristic function is required to always return zero.
6. The generic graph search algorithm returns `None` when no solution if found.
7. It is recommended (but not required) that your answer for `print_map` is shorter than 20 lines of code. If your code is much longer, you might be doing something wrong.

For example:

Test	Result

```
from student_answer import RoutingGraph, AStarFrontier, print_map
from search import *

map_str = """\
+----------------+
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|        S       |
|                |
|                |
|     G          |
|                |
|                |
|                |
+----------------+
"""

map_graph = RoutingGraph(map_str)
frontier = AStarFrontier(map_graph)
solution = next(generic_search(map_graph, frontier), None)
print_map(map_graph, frontier, solution)
```

```
+----------------+
|                |
|                |
|                |
|                |
|                |
|     ...S       |
|     ...*       |
|     ...*       |
|     G***       |
|                |
|                |
|                |
+----------------+
```

Test	Result

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+----------------+`
`|  |`
`|  |`
`|  |`
`|  |`
`|  |`
`|  |`
`|  S  |`
`|  |`
`|  |`
`|  G  |`
`|  |`
`|  |`
`|  |`
`+----------------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`# changing the heuristic so the search behaves like LCFS`
`map_graph.estimated_cost_to_goal = lambda node: 0`
 	
`frontier = AStarFrontier(map_graph)`
 	
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+----------------+`
	`|  .  |`
	`|  ...  |`
	`|  .....  |`
	`|  .......  |`
	`|  .........  |`
	`|  ...........  |`
	`|  .....S.....  |`
	`|  ....*.....  |`
	`|  ...*....  |`
	`|  G***...  |`
	`|  .....  |`
	`|  ...  |`
	`|  .  |`
	`+----------------+`

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+-------------+`
`| G  G |`
`|  S  |`
`| G  G |`
`+-------------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+-------------+`
	`| G....*****G |`
	`| .....S..... |`
	`| G.........G |`
	`+-------------+`

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+-------+`
`|  XG|`
`|X XXX  |`
`|  S  |`
`+-------+`
`"""`
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+-------+`
	`|  XG|`
	`|X XXX**|`
	`|  S***.|`
	`+-------+`

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+--+`
`|GS|`
`+--+`
`"""`
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+--+`
	`|GS|`
	`+--+`

Test	Result

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+----+`
`|    |`
`| SX |`
`| X G|`
`+----+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+----+`
	`| ***|`
	`|.SX*|`
	`|.X G|`
	`+----+`

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+---------------+`
`|    G          |`
`|XXXXXXXXXXX   |`
`|           X   |`
`|  XXXXXX   X   |`
`|  X S  X   X   |`
`|  X        X   |`
`|  XXXXXXXXX   |`
`|               |`
`+---------------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+---------------+`
	`|    G*******  |`
	`|XXXXXXXXXXX*  |`
	`|.********..X*  |`
	`|.*XXXXXX*..X*  |`
	`|.*X.S**X*..X*  |`
	`|.*X...***..X*  |`
	`|.*XXXXXXXXX*  |`
	`|.**********  |`
	`+---------------+`

`from student_answer import RoutingGraph, AStarFrontier, print_map`
`from search import *`
 	
`map_str = """\`
`+---------+`
`|         |`
`|    G    |`
`|         |`
`+---------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+---------+`
	`|         |`
	`|    G    |`
	`|         |`
	`+---------+`

Answer: (penalty regime: 0, 15, ... %)

```
1
2  import math
3  from search import *
4  import heapq
5  import numpy as np
6  #i tried my best but i had help with the printing map portion...
7  class AStarFrontier(Frontier):
8      """A* Frontier class but all heuristics node value is zero
9      """
10
11     def __init__(self, graph_map):
12         self.graph_map = graph_map
13         self.pruned = []
14         self.heap = []
15         self.entry = 0
16
17
18     def add(self, path):
19         """Adds a new path to the frontier. A path is a sequence (tuple) of
20         Arc objects.
21         """
22         #if it is a starting node, it should have no cost
23         if path[-1].tail == None:
24             #print("head is")
```

```
            #print(path[-1].head)
            #print(self.graph_map.estimated_cost_to_goal(path[-1].head) )
            totalCost = 0 + self.graph_map.estimated_cost_to_goal(path[-1].head)
        else:
            #totalCost = 0
            #print("path is")
            #print(path)
            #for arc in path:
                #totalCost += arc.cost
            totalCost = sum(arc.cost for arc in path) + self.graph_map.estimated_cost_to_goal(path[

        if path[-1].head not in self.pruned:
            #print(path[-1].head)
            #print("path[-1].head here")
            self.entry += 1
            heapq.heappush(self.heap,(totalCost, ) + (self.entry, ) + path)
            #print(self.heap)
            #print("heap here")
        #print("path to be appended is\n")
        #print(path)
        #print("\n")
        #self.heap.append(path)


    def __iter__(self):
        """We don't need a separate iterator object. Just return self. You
        don't need to change this method."""
        return self


    def __next__(self):

        """Selects, removes, and returns a path on the frontier if there is
        any.Recall that a path is a sequence (tuple) of Arc
        objects. Override this method to achieve a desired search
        strategy. If there nothing to return this should raise a
        StopIteration exception.

        """
        while len(self.heap) > 0:
            path = heapq.heappop(self.heap)
            if path[-1].head not in self.pruned:
                self.pruned.append(path[-1].head)
                return path[2:]
        raise StopIteration



def print_actions(path):
    """Given a path (a sequence of Arc objects), prints the actions that
    need to be taken and the total cost of those actions. The path is
    usually a solution (a path from the starting node to a goal
    node."""

    if path:
        print("Actions:")
        print(",\n".join("  {}".format(arc.action) for arc in path[1:]) + ".")
        print("Total cost:", sum(arc.cost for arc in path))
    else:
        print("There is no solution!")









class RoutingGraph(Graph):
    """This is a concrete subclass of Graph where vertices and edges
     are explicitly enumerated. Objects of this type are useful for
     testing graph algorithms."""

    def __init__(self, map_str):
        """Initialises an Routing graph."""

        self.goal_nodes = []
```

```
        self.start_nodes = []
        self.fuel_points = []
        self.obstacle = []
        self.map_array = []
        #self.estimated_cost_to_goal = 0

        map_str.strip()
        self.mapy = self.create_map(map_str)
        map_array = map_str.split('\n')
        self.map_array = map_array

        #print(map_array)
        for row in range(len(map_array)):
            for col in range(len(map_array[row])):
                #blank space so no need save anything
                if (map_array[row][col] == ' '):
                    pass
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '+'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == 'X'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '-'):
                    self.obstacle.append((row, col))
                #its an obstacle so cannot traverse this direction
                elif (map_array[row][col] == '|'):
                    self.obstacle.append((row, col))
                #its a goal node so append to goal node set
                elif (map_array[row][col] == 'G'):
                    self.goal_nodes.append((row, col))
                #its a solar powered agent so append it with infinite cost
                elif (map_array[row][col] == 'S'):
                    self.start_nodes.append((row, col, math.inf))
                #its a fuel point does not need cost just +15 if choose to top up
                elif (map_array[row][col] == 'F'):
                    self.fuel_points.append((row, col))
                #only other case is that its from 0-9
                else:
                    self.start_nodes.append((row, col, int(map_array[row][col])))
        #print("obstacles are \n")
        #print(self.obstacle)
        #print("STARTING NODES ARE are \n")
        #print(self.start_nodes)


    #def estimated_cost_to_goal(self,
    def create_map(self, map_str):
        """ """
        map_list = []
        for i in map_str.split('\n'):
            i = i.strip()
            map_list.append(i)

        return map_list


    def starting_nodes(self):
        """Returns a sequence of starting nodes."""
        return self.start_nodes

    def is_goal(self, node):
        """Returns true if the given node is a goal node."""
        #print("IS GOAL?")
        node_x, node_y, _ = node
        return (node_x, node_y) in self.goal_nodes

    def estimated_cost_to_goal(self):
        return 0

    def outgoing_arcs(self, node):
        """Returns a sequence of Arc objects that go out from the given
        node. The action string is automatically generated.

        """
        arcs = []
        movement_grid = [('N' , -1, 0),
                         ('E' ,  0, 1),
                         ('S' ,  1, 0),
                         ('W' ,  0, -1),]
        #for every edge in list of edges
        for directional_movement in movement_grid:
            #get all possible movements
            #eleminate those arcs which move into obstacles
```

```
            #append allowed movement arcs
            #print("Start node is:")
            #print(node)
            for i in range(len(movement_grid)):
                direction, horizontal, vertical =  movement_grid[i]
                new_node_location = (node[0] + horizontal, node[1]+ vertical, node[2] - 1)
                #if fuel is below 1 cant do action, cancel this iteration
                if (new_node_location[2] < 0):
                    continue
                #extract x and y coordinates
                new_node_x, new_node_y, _ = new_node_location
                #if i am crashing into obstacle, cancel this iteration
                if (new_node_x, new_node_y) in self.obstacle:
                    continue
                else:
                    #print("direction, horizontal, vertical is\n")
                    #print(direction, horizontal, vertical)
                    arcs.append(Arc(node, new_node_location, str(direction), 5))

            if ((node[0], node[1]) in self.fuel_points):
                #i can only stay and top up fuel if i am not on a full tank
                if (node[2] < 9):
                    arcs.append(Arc(node, (node[0], node[1], 9), "Fuel up", 15))
                    #print(arcs)
            return arcs


    def estimated_cost_to_goal(self, node):
        """Return the estimated cost to a goal node from the given
        state. This function is usually implemented when there is a
        single goal state. The function is used as a heuristic in
        search. The implementation should make sure that the heuristic
        meets the required criteria for heuristics."""
        #print("node_coord is")
        node_coord = (node[0], node[1])
        #print(node_coord)

        if node_coord in self.goal_nodes:
            return 0
        if node_coord is None:
            return 0
        heuristic = lambda x1,x2,y1,y2 : abs(x1-x2) + abs(y1-y2) #manhattan distance
        pool = []
        nrow = node_coord[0]
        ncol = node_coord[1]
        #print(self.goal_nodes)
        for row,col in self.goal_nodes:
            pool.append(heuristic(ncol,col,nrow,row))
        #print("pool is")
        #print(pool)
        return min(pool)*5

def print_map(map_graph, frontier, solution):
    graph_rows = []
    agentCoord = []
    #converted graph into rows
    for row in map_graph.map_array:
        graph_rows.append(row.strip())
    #converted startnodes into x, y
    for coord in map_graph.start_nodes:
        agentCoord.append((coord[0], coord[1]))
    #for every node that got visited, should be .
    for node in frontier.pruned:
        node = (node[0], node[1]) #shorten from len 3 tuple to len 2
        #if the pruned node is not start node and goal node
        if node not in agentCoord and node not in map_graph.goal_nodes:
            row, col = node[0], node[1]
            #replacing the item in the list with the .
            graph_rows[row] = graph_rows[row][:col] + '.' + graph_rows[row][col+1:]
        if solution is not None:
            for arcs in solution:
                node = (arcs.head[0], arcs.head[1])
                if node not in agentCoord and node not in map_graph.goal_nodes:
                    row, col = node[0], node[1]
                    graph_rows[row] = graph_rows[row][:col] + '*' + graph_rows[row][col+1:]
    graphMap = '\n'.join(graph_rows)
    print(graphMap)
```

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+----------------+`
	`|  |`
	`|  |`
	`|  |`
	`|  |`
	`|  |`
	`|  |`
	`|  S  |`
	`|  |`
	`|  |`
	`|  G  |`
	`|  |`
	`|  |`
	`|  |`
	`+----------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+----------------+`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  ...S  |`
		`|  |`
		`|  ...*  |`
		`|  |`
		`|  ...*  |`
		`|  |`
		`|  G***  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`|  |`
		`+----------------+`
			`+----------------+`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  ...S  |`
			`|  |`
			`|  ...*  |`
			`|  |`
			`|  ...*  |`
			`|  |`
			`|  G***  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`|  |`
			`+----------------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+----------------+`
	`|  |`
	`|  |`
	`|  |`
	`|  |`
	`|  |`
	`|  |`
	`|  S  |`
	`|  |`
	`|  |`
	`|  G  |`
	`|  |`
	`|  |`
	`|  |`
	`+----------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`# changing the heuristic so the search behaves like LCFS`
	`map_graph.estimated_cost_to_goal = lambda node: 0`
	
	`frontier = AStarFrontier(map_graph)`
	
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+----------------+`
		`|  .  |`
		`|  |`
		`|  ...  |`
		`|  |`
		`|  .....  |`
		`|  |`
		`|  .......  |`
		`|  |`
		`|  .........  |`
		`|  |`
		`|  ...........  |`
		`|  |`
		`|  |`
		`.....S.....  |`
		`|  ....*.....  |`
		`|  |`
		`|  ...*....  |`
		`|  |`
		`|  G***...  |`
		`|  |`
		`|  .....  |`
		`|  |`
		`|  ...  |`
		`|  |`
		`|  .  |`
		`|  |`
		`+----------------+`
			`+----------------+`
			`|  .  |`
			`|  |`
			`|  ...  |`
			`|  |`
			`|  .....  |`
			`|  |`
			`|  .......  |`
			`|  |`
			`|  .........  |`
			`|  |`
			`|  ...........  |`
			`|  |`
			`|  |`
			`.....S.....  |`
			`|  ....*.....  |`
			`|  |`
			`|  ...*....  |`
			`|  |`
			`|  G***...  |`
			`|  |`
			`|  .....  |`
			`|  |`
			`|  ...  |`
			`|  |`
			`|  .  |`
			`|  |`
			`+----------------+`
				✔

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+-------------+`
	`| G         G |`
	`|      S      |`
	`| G         G |`
	`+-------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+-------------+`
		`| G....*****G |`
		`| .....S..... |`
		`| G.........G |`
		`+-------------+`
			`+-------------+`
			`| G....*****G |`
			`| .....S..... |`
			`| G.........G |`
			`+-------------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+-------+`
	`|     XG|`
	`|X XXX  |`
	`|  S    |`
	`+-------+`
	`"""`
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+-------+`
		`|     XG|`
		`|X XXX**|`
		`|  S***.|`
		`+-------+`
			`+-------+`
			`|     XG|`
			`|X XXX**|`
			`|  S***.|`
			`+-------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+--+`
	`|GS|`
	`+--+`
	`"""`
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+--+`
		`|GS|`
		`+--+`
			`+--+`
			`|GS|`
			`+--+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+----+`
	`|    |`
	`| SX |`
	`| X G|`
	`+----+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+----+`
		`| ***|`
		`|.SX*|`
		`|.X G|`
		`+----+`
			`+----+`
			`| ***|`
			`|.SX*|`
			`|.X G|`
			`+----+`
				✔

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+---------------+`
	`|    G          |`
	`|XXXXXXXXXXX   |`
	`|           X   |`
	`|  XXXXXX   X   |`
	`|  X S  X   X   |`
	`|  X        X   |`
	`|  XXXXXXXXX   |`
	`|               |`
	`+---------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+---------------+`
		`|    G********  |`
		`|XXXXXXXXXXX*  |`
		`|.********..X*  |`
		`|.*XXXXXX*..X*  |`
		`|.*X.S**X*..X*  |`
		`|.*X...***..X*  |`
		`|.*XXXXXXXXX*  |`
		`|.***********  |`
		`+---------------+`
			`+---------------+`
			`|    G********  |`
			`|XXXXXXXXXXX*  |`
			`|.********..X*  |`
			`|.*XXXXXX*..X*  |`
			`|.*X.S**X*..X*  |`
			`|.*X...***..X*  |`
			`|.*XXXXXXXXX*  |`
			`|.***********  |`
			`+---------------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+------------+`
	`|         X  |`
	`| S       X G|`
	`|         X  |`
	`|         X  |`
	`|         X  |`
	`+------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+------------+`
		`|.........X  |`
		`|.S.......X G|`
		`|.........X  |`
		`|.........X  |`
		`|.........X  |`
		`+------------+`
			`+------------+`
			`|.........X  |`
			`|.S.......X G|`
			`|.........X  |`
			`|.........X  |`
			`|.........X  |`
			`+------------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+---------+`
	`|         |`
	`|    G    |`
	`|         |`
	`+---------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+---------+`
		`|         |`
		`|    G    |`
		`|         |`
		`+---------+`
			`+---------+`
			`|         |`
			`|    G    |`
			`|         |`
			`+---------+`
				✔

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+-------------+`
	`|         G   |`
	`| S           |`
	`|         S   |`
	`+-------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+-------------+`
		`|         G   |`
		`| S       *   |`
		`|         S   |`
		`+-------------+`
			`+-------------+`
			`|         G   |`
			`| S       *   |`
			`|         S   |`
			`+-------------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+------+`
	`|      |`
	`|S X   |`
	`|XXXXX |`
	`|G X   |`
	`|      |`
	`+------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+------+`
		`|****..|`
		`|S.X***|`
		`|XXXXX*|`
		`|G*X***|`
		`|.***..|`
		`+------+`
			`+------+`
			`|****..|`
			`|S.X***|`
			`|XXXXX*|`
			`|G*X***|`
			`|.***..|`
			`+------+`
				✔

✔	`from student_answer import RoutingGraph, AStarFrontier, print_map`
	`from search import *`
	
	`map_str = """\`
	`+-------------+`
	`|S            |`
	`|             |`
	`|   G      S  |`
	`|             |`
	`| G           |`
	`+-------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+-------------+`
		`|S***         |`
		`|...*         |`
		`|...G      S  |`
		`|..            |`
		`|.G            |`
		`+-------------+`
			`+-------------+`
			`|S***         |`
			`|...*         |`
			`|...G      S  |`
			`|..            |`
			`|.G            |`
			`+-------------+`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

◄ 11. Games Jump to... A2. Predicting number sequences ►