Started on	Wednesday, 16 August 2023, 1:58 PM
State	Finished
Completed on	Wednesday, 23 August 2023, 6:08 PM
Time taken	7 days 4 hours
Marks	2.85/3.00
Grade	**0.95** out of 1.00 (**95**%)

Information

Introduction

In this assignment you apply the search techniques you have learnt in the course to a routing problem. The problem involves 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 is time-wise the closest.

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, only the closest pair).
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 (e.g. do not print anything). 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()
   ```

   This way, the `main` function will not be run when your code is imported on the server and therefore it will not interfere with the output of the test cases.
5. Answer the questions in order. The answer to some questions require your answer to earlier questions (to use or to build on).
6. For each question, the test cases test some aspects of the provided answer. It is computationally infeasible to exhaustively test all possible scenarios. Your answer to a question may pass all of our test cases (and receive full mark) but lack some functionality that is needed later or even have an undetected bug. As with all programming tasks, it is your responsibility to read the specs carefully, and write, test, and debug your program.

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` to `9`.
  - Agents indicated by `S` are solar and do not need fueling. We assume they have unlimited energy capacity (and/or it's always sunny).
  - 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 of 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. All single directional moves take 5 units of time. This is what we regard as the cost of the action (since the objective of the problem is to minimise the service time).
- The symbol `F` indicates a fuel station. 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).
- The symbol `P` indicates a portal. There can be zero or more portals on a map. If an agent is in a cell marked as `P`, in addition to the usual movements, it has the option of *teleporting* to any other location on the map marked as `P`. In the sequence of arcs, the teleport action (if available) should appear after any other directional action. The action does not consume any fuel but costs 10 units of time (regardless of the destination). The action must be labeled as `"Teleport to (row, col)"` where *row* and *col* are the row and column indices of the destination portal. If multiple outgoing arcs of this type are available, they should appear in the ascending order of the row number and the column number of the target portals (i.e. arcs leading to portals on lower row numbers should come first and for portals on the same row, arcs leading to portals on lower column numbers should come first).

Task

Write a class `RoutingGraph` which is initialised by the map string and represents the map in the form of a graph by implementing the required methods in the `Graph` class. 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. You have the option of not implementing it for 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),]
```

8. In a more realistic scenario, teleportation corresponds to using a different mode of transportation using a different network (e.g. using a train or a ferry).

For example:

Test

```
from student_answer import RoutingGraph
import math

map_str = """\
+-------+
|  9  XG|
|X XXX P|
| S  0FG|
|XX P XX|
+-------+
"""

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))

node = (2, 7, 4)  # at a location with a portal
print("\nOutgoing arcs (available actions) at
{}:".format(node))
for arc in graph.outgoing_arcs(node):
    print ("  " + str(arc))
```

Result

```
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)

Outgoing arcs (available actions) at (2, 7, 4):
  Arc(tail=(2, 7, 4), head=(1, 7, 3), action='N', cost=5)
  Arc(tail=(2, 7, 4), head=(3, 7, 3), action='S', cost=5)
  Arc(tail=(2, 7, 4), head=(2, 6, 3), action='W', cost=5)
  Arc(tail=(2, 7, 4), head=(4, 4, 4), action='Teleport to (4, 4)',
cost=10)
```

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, ... %)

```
from search import *
from math import inf

class RoutingGraph(Graph):
    def __init__(self, map_str):
        self.map_list = map_str.splitlines()
        self.rmost_entity = len(self.map_list[0]) - 2
        self.dmost_entity = len(self.map_list) - 2
        self.max_tank = 9
        self.directions = [('N' , -1, 0),
                           ('E' ,  0, 1),
                           ('S' ,  1, 0),
                           ('W' ,  0, -1),]
        self.goal_nodes = []
        self.ps = []
        for i in range(1, self.dmost_entity+1):
            for j in range(1, self.rmost_entity+1):
                if self.map_list[i][j] == 'G':
                    self.goal_nodes += [(i, j)]
                elif self.map_list[i][j] == 'P':
                    self.ps += [(i, j)]

    def starting_nodes(self):
        start = []
        for i in range(1, self.dmost_entity+1):
            for j in range(1, self.rmost_entity+1):
                if self.map_list[i][j] == 'S':
                    start += [(i, j, inf)]
                elif self.map_list[i][j] in '0123456789':
                    start += [(i, j, int(self.map_list[i][j]))]
        return start

    def is_goal(self, node):
        for row, col in self.goal_nodes:
```

```
35            if row == node[0] and col == node[1]:
36                return True
37        return False
38
39    def estimated_cost_to_goal(node):
40        return 0
41
42    def outgoing_arcs(self, tail):
43        arcs = []
44        row, col, energy = tail
45        for action, row_delta, col_delta in self.directions:
46            if 1 <= (row+row_delta) <= self.dmost_entity and 1 <= (col+col_delt
47                if self.map_list[row+row_delta][col+col_delta] != 'X' and energ
48                    head = (row+row_delta, col+col_delta, energy-1)
49                    arcs.append(Arc(tail, head, action, 5))
50        if self.map_list[row][col] == 'P':
51            for row_p, col_p in self.ps:
52
```

	Test	Expected	Got	

✔	`from student_answer import RoutingGraph`
	`import math`
	
	`map_str = """\`
	`+-------+`
	`|  9  XG|`
	`|X XXX P|`
	`| S  0FG|`
	`|XX P XX|`
	`+-------+`
	`"""`
	
	`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))`
	
	`node = (2, 7, 4)  # at a location with a portal`
	`print("\nOutgoing 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)`
		
		`Outgoing arcs (available actions) at (2, 7, 4):`
		`  Arc(tail=(2, 7, 4), head=(1, 7, 3), action='N', cost=5)`
		`  Arc(tail=(2, 7, 4), head=(3, 7, 3), action='S', cost=5)`
		`  Arc(tail=(2, 7, 4), head=(2, 6, 3), action='W', cost=5)`
		`  Arc(tail=(2, 7, 4), head=(4, 4, 4), action='Teleport to (4, 4)', cost=10)`
			`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)`
			
			`Outgoing arcs (available actions) at (2, 7, 4):`
			`  Arc(tail=(2, 7, 4), head=(1, 7, 3), action='N', cost=5)`
			`  Arc(tail=(2, 7, 4), head=(3, 7, 3), action='S', cost=5)`
			`  Arc(tail=(2, 7, 4), head=(2, 6, 3), action='W', cost=5)`
			`  Arc(tail=(2, 7, 4), head=(4, 4, 4), action='Teleport to (4, 4)', cost=10)`
				✔

	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 0.85 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 (i.e. shortest time) solution (if one exists) from one of the agents to the goal node.

In this question, modify the method `estimated_cost_to_goal` in the `RoutingGraph` class so that it always returns `0` (zero) for any given node. In the next question, we will ask you to implement a proper heuristic.

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` in 40 lines of code or shorter. 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

Test	Result

`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`

`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, ... %)

```
from search import *
from math import inf

class RoutingGraph(Graph):
    def __init__(self, map_str):
        self.map_list = map_str.splitlines()
        self.rmost_entity = len(self.map_list[0]) - 2
        self.dmost_entity = len(self.map_list) - 2
        self.max_tank = 9
        self.directions = [('N' , -1, 0),
                           ('E' ,  0, 1),
                           ('S' ,  1, 0),
                           ('W' ,  0, -1),]
        self.goal_nodes = []
        self.ps = []
        for i in range(1, self.dmost_entity+1):
            for j in range(1, self.rmost_entity+1):
                if self.map_list[i][j] == 'G':
                    self.goal_nodes += [(i, j)]
                elif self.map_list[i][j] == 'P':
                    self.ps += [(i, j)]

    def starting_nodes(self):
        start = []
        for i in range(1, self.dmost_entity+1):
            for j in range(1, self.rmost_entity+1):
                if self.map_list[i][j] == 'S':
                    start += [(i, j, inf)]
                elif self.map_list[i][j] in '0123456789':
                    start += [(i, j, int(self.map_list[i][j]))]
        return start

    def is_goal(self, node):
        for row, col in self.goal_nodes:
            if row == node[0] and col == node[1]:
                return True
        return False

    def estimated_cost_to_goal(self, node):
        return 0

```

```
    def outgoing_arcs(self, tail):
        arcs = []
        row, col, energy = tail
        for action, row_delta, col_delta in self.directions:
            if 1 <= (row+row_delta) <= self.dmost_entity and 1 <= (col+col_delt
                if self.map_list[row+row_delta][col+col_delta] != 'X' and energ
                    head = (row+row_delta, col+col_delta, energy-1)
                    arcs.append(Arc(tail, head, action, 5))
        if self.map_list[row][col] == 'P':
            for row_p, col_p in self.ps:
```

	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`
				✔

	Test	Expected	Got	

✔	`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
				✔

✔	`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
				✔

	Test	Expected	Got	

✔	`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!`	✔

✔	`from student_answer import RoutingGraph, AStarFrontier`
	`from search import *`
	
	`map_str = """\`
	`+------------+`
	`|  P  |`
	`| 7  |`
	`|XXXX  |`
	`|P F X  G  |`
	`+------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_actions(solution)`
		`Actions:`
		`  N,`
		`  E,`
		`  E,`
		`  E,`
		`  Teleport to (4, 1),`
		`  E,`
		`  E,`
		`  Fuel up,`
		`  W,`
		`  W,`
		`  Teleport to (1, 5),`
		`  E,`
		`  E,`
		`  E,`
		`  S,`
		`  S,`
		`  S.`
		`Total cost: 105`
			`Actions:`
			`  N,`
			`  E,`
			`  E,`
			`  E,`
			`  Teleport to (4, 1),`
			`  E,`
			`  E,`
			`  Fuel up,`
			`  W,`
			`  W,`
			`  Teleport to (1, 5),`
			`  E,`
			`  E,`
			`  E,`
			`  S,`
			`  S,`
			`  S.`
			`Total cost: 105`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.85/1.00**.

Question 3

Correct

Mark 1.00 out of 1.00

The print_map procedure

Write a procedure `print_map(map_graph, frontier, solution)` that takes three parameters: an instance of `RoutingGraph`, an instance of `AStarFrontier` which has just been used to run a graph search on the given graph, and the result of the search, 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;
- 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. The frontier object that is passed to `print_map` is your own implementation of `AStarFrontier`. You should use this object to see which nodes have been expanded.
2. The solution parameter is either a sequence of `Arc` objects that make up a path from a starting node to a goal node, or is `None`.
3. 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`.
4. 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.
5. The test cases do not include any fuel-based agents, fuel stations, or portals. Do not consider anything fuel-related or portal-related when devising the heuristic function.
6. Only the first solution returned by the generic search procedure (if there is one) is used to test your procedure.
7. 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 and returned by the frontier. If the returned node has neighbours, the 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 (as defined in the first question).
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***  |`
	`|  |`
	`|  |`
	`|  |`
	`+----------------+`

`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 |`
	`+-------------+`

Test	Result

`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|`
	`+--+`

`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          |`
`|XXXXXXXXXXXX   |`
`|           X   |`
`|  XXXXXX   X   |`
`|  X S  X   X   |`
`|  X        X   |`
`|  XXXXXXXXXX   |`
`|               |`
`+---------------+`
`"""`
 	
`map_graph = RoutingGraph(map_str)`
`frontier = AStarFrontier(map_graph)`
`solution = next(generic_search(map_graph, frontier), None)`
`print_map(map_graph, frontier, solution)`
	`+---------------+`
	`|    G*******  |`
	`|XXXXXXXXXXXX*  |`
	`|.*******..X*  |`
	`|.*XXXXXX*..X*  |`
	`|.*X.S**X*..X*  |`
	`|.*X...***..X*  |`
	`|.*XXXXXXXXX*  |`
	`|.***********  |`
	`+---------------+`

Test	Result

`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, ... %)

```
from search import *
from math import inf

class RoutingGraph(Graph):
    def __init__(self, map_str):
        self.map_list = map_str.splitlines()
        self.rmost_entity = len(self.map_list[0]) - 2
        self.dmost_entity = len(self.map_list) - 2
        self.max_tank = 9
        self.directions = [('N' , -1, 0),
                           ('E' ,  0, 1),
                           ('S' ,  1, 0),
                           ('W' ,  0, -1),]
        self.goal_nodes = []
        self.ps = []
        for i in range(1, self.dmost_entity+1):
            for j in range(1, self.rmost_entity+1):
                if self.map_list[i][j] == 'G':
                    self.goal_nodes += [(i, j)]
                elif self.map_list[i][j] == 'P':
                    self.ps += [(i, j)]

    def starting_nodes(self):
        start = []
        for i in range(1, self.dmost_entity+1):
            for j in range(1, self.rmost_entity+1):
                if self.map_list[i][j] == 'S':
                    start += [(i, j, inf)]
                elif self.map_list[i][j] in '0123456789':
                    start += [(i, j, int(self.map_list[i][j]))]
        return start

    def is_goal(self, node):
        for row, col in self.goal_nodes:
            if row == node[0] and col == node[1]:
                return True
        return False

    def estimated_cost_to_goal(self, node):
        min_cost = inf
        row, col, energy = node
        for i, j in self.goal_nodes:
            cost = (abs(i-row) + abs(j-col))*5
            min_cost = min(min_cost, cost)
        return min_cost

    def outgoing_arcs(self, tail):
        arcs = []
        row, col, energy = tail
        for action, row_delta, col_delta in self.directions:
            if 1 <= (row+row_delta) <= self.dmost_entity and 1 <= (col+col_delt
```

	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          |`
	`|XXXXXXXXXXXX   |`
	`|           X   |`
	`|  XXXXXX   X   |`
	`|  X S  X   X   |`
	`|  X        X   |`
	`|  XXXXXXXXXX   |`
	`|               |`
	`+---------------+`
	`"""`
	
	`map_graph = RoutingGraph(map_str)`
	`frontier = AStarFrontier(map_graph)`
	`solution = next(generic_search(map_graph, frontier), None)`
	`print_map(map_graph, frontier, solution)`
		`+---------------+`
		`|    G********  |`
		`|XXXXXXXXXXXX*  |`
		`|.********..X*  |`
		`|.*XXXXXX*..X*  |`
		`|.*X.S**X*..X*  |`
		`|.*X...***..X*  |`
		`|.*XXXXXXXXXX*  |`
		`|.***********  |`
		`+---------------+`
			`+---------------+`
			`|    G********  |`
			`|XXXXXXXXXXXX*  |`
			`|.********..X*  |`
			`|.*XXXXXX*..X*  |`
			`|.*X.S**X*..X*  |`
			`|.*X...***..X*  |`
			`|.*XXXXXXXXXX*  |`
			`|.***********  |`
			`+---------------+`
				✔

✔	`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    |`
			`|         |`
			`+---------+`
				✔

✔	`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   |`
			`+-------------+`
				✔

	Test	Expected	Got	

✔	`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***|`
			`|.***..|`
			`+------+`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.