Started on	Thursday, 20 July 2023, 1:07 PM
State	Finished
Completed on	Thursday, 27 July 2023, 5:19 PM
Time taken	7 days 4 hours
Marks	12.00/12.00
Grade	**1.00** out of 1.00 (**100**%)

Information

Uninformed and cost-insensitive search

This quiz includes, among other things, two types of questions: tracing the frontier (in the generic graph search) for a given graph using DFS and BFS frontiers, and the implementation of these frontier classes in Python. You are expected to be able to do the tracing by hand, but you could also first write the frontier classes and make them generate the trace for you. Take the latter approach if you are fairly comfortable with doing the tracing by hand.

Information

Optional activity: using the AI-Space search applet

AI-Space Group (http://aispace.org) provides a number of pedagogical tools in the area of AI. These tools are in the form of Java applets. One of these tools is a search applet which is available in the download section of the site. The search applet may be helpful in better understudying the graph search algorithms. The web site has some useful tutorials.

In most desktop environments (including the lab computers) you should be able to run the applet by double-clicking on the file. You can also run the program from a terminal, by typing the following command at the shell prompt:

```
java -jar path/to/search.jar
```

You will need Java Runtime Environment (JRE) in order to run this program. JRE is already installed on all the lab machines. At home, however, you may need to download it from http://java.com if it is not already installed on your system.

Note: This activity is strictly optional. You can safely skip it. If you decide to do the activity, please bear in mind that in some cases the conventions or notations used in the applet are different from what we use in this course.

Question 1
Correct
Mark 1.00 out of 1.00

Which of the following statements about a solution (a path to a goal node) found by depth-first search (DFS) is true?

Select one or more:

- [ ] a. The algorithm always finds the shallowest solution---a path with the fewest number of arcs.
- [x] b. The length (number of arcs) of the solution found by DFS depends on the order of children (edges) in the graph. ✔
- [ ] c. The algorithm always finds a solution with the lowest cost.
- [ ] d. The algorithm always finds the deepest solution---a path with the largest number of arcs.

Correct
Marks for this submission: 1.00/1.00.

Information

Frontier trace format

This information box appears in any quiz that includes questions on tracing the frontier of a graph search problem.

Trace format

- Starting with an empty frontier we record all the calls to the frontier: to add or to get a path. We dedicate one line per call.
- When we ask the frontier to add a path, we start the line with a `+` followed by the path that is being added.
- When we ask for a path from the frontier, we start the line with a `-` followed by the path that is being removed.
- When using a priority queue, the path is followed by a comma and then the key (e.g. cost, heuristic, f-value, ...).
- The lines of the trace should match the following regular expression (case and space insensitive): `^[+-][a-z]+(,\d+)?!?$`
- The symbol `!` is used at the end of a line to indicate a pruning operation. It must be used when the path that is being added to, or is removed from the frontier is to a node that is already expanded. Note that if a path that is removed from the frontier is pruned, it is not returned to the search algorithm; the frontier has to continue removing paths until it removes one that can be returned.

Precheck

You can check the format of your answer by clicking the "Precheck" button which checks the format of each line against a regular expression. Precheck only looks at the syntax of your answer and the frontier that must be used. It does not use the information in the graph object (so for example, `+ a` would pass the format check for a BFS or DFS question even if the graph does not have a node called `a`).

If your answer fails the precheck, it will fail the check. If it passes the precheck, it may pass the main check, it may not. You will not lose or gain any marks by using "Precheck".

Notes

- All state-space graphs are directed.
- In explicit graphs, the order of arcs is determined by the order of elements in `edge_list`.
- Frontier traces are not case sensitive - you don't have to type capital letters.
- There can be any number of white space characters between characters in a trace line and in between lines. When evaluating your answer, white spaces will be removed even if they are between characters.
- You can have comments in a trace. Comments start with a hash sign, #. They can take an entire line or just appear at the end of a line. Any character string appearing after # in a line will be ignored by the grader. You can use comments to keep track of things such as the *expanded* set (when pruning).
- In questions where the search strategy is cost-sensitive (e.g. LCFS) you must include the cost information on all lines.
- In questions where a priority queue is needed, the queue must be stable.

Question 2

Correct

Mark 1.00 out of 1.00

Given the following graph, trace the frontier of a depth-first search (DFS).

```
ExplicitGraph(
    nodes={'S', 'G'},
    edge_list=[('S','G')],
    starting_nodes = ['S'],
    goal_nodes = {'G'},
    )
```

Note: This question does not attract any penalty for wrong submissions. [Please note the penalty regime in other questions.]

Answer: (penalty regime: 0 %)

```
+ S
- S
+ SG
- SG
```

+S (OK)
-S (OK)
+SG (OK)
-SG (OK)

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 3

Correct

Mark 1.00 out of 1.00

Given the following graph, trace the frontier of a depth-first search (DFS).

```
ExplicitGraph(
    nodes={'S', 'A', 'B', 'T'},
    edge_list=[('A', 'B'), ('S', 'T'), ('S', 'A'), ('B', 'T')],
    starting_nodes = ['S'],
    goal_nodes = {'T'}
    )
```

Answer: (penalty regime: 20, 40, ... %)

```
+ S
- S
+ ST
+ SA
- SA
+ SAB
- SAB
+ SABT
- SABT
```

+S (OK)
-S (OK)
+ST (OK)
+SA (OK)
-SA (OK)
+SAB (OK)
-SAB (OK)
+SABT (OK)
-SABT (OK)

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 4

Correct

Mark 1.00 out of 1.00

Given the following graph, trace the frontier of a depth-first search (DFS).

```
ExplicitGraph(
    nodes={'S', 'A', 'B', 'T'},
    edge_list=[('A', 'B'), ('S', 'A'), ('S', 'T'), ('B', 'T')],
    starting_nodes=['S'],
    goal_nodes={'T'})
```

Answer: (penalty regime: 20, 40, ... %)

```
+ S
- S
+ SA
+ ST
- ST
```

+S (OK)
-S (OK)
+SA (OK)
+ST (OK)
-ST (OK)

Passed all tests!

Correct

Marks for this submission: 1.00/1.00.

Question 5

Correct

Mark 1.00 out of 1.00

Which of the following statements about depth-first search is correct? In all the statements assume that the *generic search* algorithm is being used with a DFS frontier on a finite directed graph and that the frontier does not do pruning (i.e. it does not keep track of expanded nodes).

Note the following terminology used in the statements.

- Phrases such as "edge ordering" and "order of children" refer to the order in which the outgoing arcs of a node appear in the list returned by the method outgoing_arcs.
- An algorithm halting means that it stops (terminates) after some time. In other words, it does not get into an infinite loop.

Select one:

- If for a given graph, DFS halts with every possible edge ordering, then we can conclude that the order of edges does not have any effect on whether or not the algorithm finds a solution on that graph. ✔ If every edge ordering halts then either a solution is found every time or the graph is completely explored (which means there are no cycles).
- The order of children (edges) does not have any effect on what solution is found by depth-first search.
- Depth-first search goes into an infinite loop on any finite directed graph that contains a cycle.
- The order of children (edges) does not have any effect on whether or not depth-first search halts.

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Given the following graph, trace the frontier of a depth-first search (DFS).

```
ExplicitGraph(
    nodes={'S', 'A', 'B', 'T'},
    edge_list=[('A', 'B'), ('S', 'A'), ('S', 'T'), ('B', 'T')],
    starting_nodes=['T', 'S'],
    goal_nodes={'T'})
```

Answer: (penalty regime: 20, 40, ... %)

```
+ T
+ S
- S
+ SA
+ ST
- ST
```

+T (OK)
+S (OK)
-S (OK)
+SA (OK)
+ST (OK)
-ST (OK)

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 7

Correct

Mark 1.00 out of 1.00

Given the following graph, trace the frontier of a breadth-first search (BFS).

```
ExplicitGraph(
    nodes={'S', 'A', 'B', 'G'},
    edge_list=[('A', 'B'), ('S', 'G'), ('S', 'A'), ('B', 'G')],
    starting_nodes = ['S'],
    goal_nodes = {'G'}
    )
```

Answer: (penalty regime: 20, 40, ... %)

```
+ S
- S
+ SG
+ SA
- SG
```

+S (OK)
-S (OK)
+SG (OK)
+SA (OK)
-SG (OK)

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 8

Correct

Mark 1.00 out of 1.00

Given the following graph, trace the frontier of a breadth-first search (BFS).

```
ExplicitGraph(
    nodes={'S', 'A', 'B', 'G'},
    edge_list=[('A', 'B'), ('S', 'A'), ('S', 'G'), ('B', 'G')],
    starting_nodes=['S'],
    goal_nodes={'G'},)
```

Answer: (penalty regime: 20, 40, ... %)

```
+ S
- S
+ SA
+ SG
- SA
+ SAB
- SG
```

+S (OK)
-S (OK)
+SA (OK)
+SG (OK)
-SA (OK)
+SAB (OK)
-SG (OK)

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Information

Getting familiar with search.py

search.py is a small Python library that contains generic classes and functions related to graph search. The library is reasonably documented and clear. You need to download the file, read it and clearly understand it before you can proceed to answer the following programming questions. The file must be placed in the same directory as your programs so that it can be imported by your programs.

We use Python generators quite often in this course. If you are not familiar with them, read the documentation or find a tutorial online. For the impatient, the answers to this question on the Stackoverflow might be helpful.

Information

How your answer is tested on the server

Every time you submit a program to the server, the content of the answer box is written to a new file named "`student_answer.py`". The test code then imports `student_answer` and other necessary modules and tests the functionality of the classes and functions by generating some output. A test passes if the generated output matches the expected output. This model entails the following points which you should keep in mind when working on similar programming questions:

1. You do **not** need to (and you shouldn't) copy the content of the search module into your file; only import it. The search module is available on the server.
2. Put your examples, test cases, or any other code that generates output in a separate function (for example `main`) so that when your answer is imported as a module by the test code, it does not print or do unwanted things and therefore does not affect the overall output of the test.
3. The idea of using a function like `main` is only for your convenience; it is not imported or used in the test cases. Therefore it does not matter whether or not you include it in the answer. If you include it, however, make sure the main function is not called by your module code and is not executed when the module is imported. For example use `if __name__ == "__main__": main()` when appropriate.
4. You must not have any global code that generates output.
5. If your code imports a module (for example imports the search module) then the code you paste in the answer box should also contain that import statement.
6. These points imply that if you follow a good coding practice, then you can simply paste the entire content of your program file in the answer box.

Question 9

Correct

Mark 1.00 out of 1.00

DFS Frontier

As it is mentioned in the documentation of the search module, you need to write a concrete subclass of Graph and a concrete subclass of Frontier in order to have a working graph search code. A concrete subclass of `Graph` called `ExplicitGraph` is already included in the search module. In this question, you have to write a concrete subclass of `Frontier` called `DFSFrontier` such that when an instance of it along with an instance of Graph are passed to the `generic_search` function, the function behaves as a depth-first search algorithm.

Note: you need to download the the search module from the link provided at the beginning of this page and put the file in the current directory (the directory in which you save and run your programs).

▶ **Spoiler/Hint**: a skeleton code is given here (click to expand)

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from search import *
from student_answer import DFSFrontier

graph = ExplicitGraph(nodes=set('SAG'),
                      edge_list=[('S','A'), ('S', 'G'), ('A', 'G')],
                      starting_nodes=['S'],
                      goal_nodes={'G'})

solutions = generic_search(graph, DFSFrontier())
solution = next(solutions, None)
print_actions(solution)</pre></td><td><pre>Actions:
  S->G.
Total cost: 1</pre></td></tr>
<tr><td><pre>from search import *
from student_answer import DFSFrontier

graph = ExplicitGraph(nodes=set('SAG'),
                      edge_list=[('S', 'G'), ('S','A'), ('A', 'G')],
                      starting_nodes=['S'],
                      goal_nodes={'G'})

solutions = generic_search(graph, DFSFrontier())
solution = next(solutions, None)
print_actions(solution)</pre></td><td><pre>Actions:
  S->A,
  A->G.
Total cost: 2</pre></td></tr>
<tr><td><pre>from search import *
from student_answer import DFSFrontier

available_flights = ExplicitGraph(
    nodes=['Christchurch', 'Auckland',
           'Wellington', 'Gold Coast'],
    edge_list=[('Christchurch', 'Gold Coast'),
               ('Christchurch','Auckland'),
               ('Christchurch','Wellington'),
               ('Wellington', 'Gold Coast'),
               ('Wellington', 'Auckland'),
               ('Auckland', 'Gold Coast')],
    starting_nodes=['Christchurch'],
    goal_nodes={'Gold Coast'})

my_itinerary = next(generic_search(available_flights, DFSFrontier()), None)
print_actions(my_itinerary)</pre></td><td><pre>Actions:
  Christchurch->Wellington,
  Wellington->Auckland,
  Auckland->Gold Coast.
Total cost: 3</pre></td></tr>
</table>

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

```
from search import *
class DFSFrontier(Frontier):
    """when an instance of it along with an instance of Graph are passed to
    the generic_search function, the function behaves as a
    depth-first search algorithm.
    """
    def __init__(self):
        """The constructor takes no argument. It initialises the
        container to an empty stack."""
        self.container = []
```

```python

    def add(self, path):
        """Adds a new path to the frontier. A path is a sequence (tuple) of
        Arc objects. You should override this method.

        """
        self.container.append(path)

    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.

        """
        if len(self.container) == 0:
            raise StopIteration
        else:
            return self.container.pop()

    def __iter__(self):
            """The object returns itself because it is implementing a __next__
            method and does not need any additional state for iteration."""
            return self
```

	Test	Expected	Got	

	from search import *
	from student_answer import DFSFrontier
	
	graph = ExplicitGraph(nodes=set('SAG'),
	                      edge_list=[('S','A'), ('S', 'G'), ('A', 'G')],
	                      starting_nodes=['S'],
	                      goal_nodes={'G'})
	
	solutions = generic_search(graph, DFSFrontier())
	solution = next(solutions, None)
	print_actions(solution)
		Actions:
		  S->G.
		Total cost: 1
			Actions:
			  S->G.
			Total cost: 1	

	from search import *
	from student_answer import DFSFrontier
	
	graph = ExplicitGraph(nodes=set('SAG'),
	                      edge_list=[('S', 'G'), ('S','A'), ('A', 'G')],
	                      starting_nodes=['S'],
	                      goal_nodes={'G'})
	
	solutions = generic_search(graph, DFSFrontier())
	solution = next(solutions, None)
	print_actions(solution)
		Actions:
		  S->A,
		  A->G.
		Total cost: 2
			Actions:
			  S->A,
			  A->G.
			Total cost: 2	

	Test	Expected	Got	

✔	`from search import *`
	`from student_answer import DFSFrontier`
	
	`available_flights = ExplicitGraph(`
	`    nodes=['Christchurch', 'Auckland',`
	`           'Wellington', 'Gold Coast'],`
	`    edge_list=[('Christchurch', 'Gold Coast'),`
	`               ('Christchurch','Auckland'),`
	`               ('Christchurch','Wellington'),`
	`               ('Wellington', 'Gold Coast'),`
	`               ('Wellington', 'Auckland'),`
	`               ('Auckland', 'Gold Coast')],`
	`    starting_nodes=['Christchurch'],`
	`    goal_nodes={'Gold Coast'})`
	
	`my_itinerary = next(generic_search(available_flights,`
	`DFSFrontier()), None)`
	`print_actions(my_itinerary)`
		`Actions:`
		`  Christchurch-`
		`>Wellington,`
		`  Wellington->Auckland,`
		`  Auckland->Gold Coast.`
		`Total cost: 3`
			`Actions:`
			`  Christchurch-`
			`>Wellington,`
			`  Wellington->Auckland,`
			`  Auckland->Gold Coast.`
			`Total cost: 3`
				✔

✔	`from search import *`
	`from student_answer import DFSFrontier`
	
	`graph = ExplicitGraph(nodes=set('SAG'),`
	`                      edge_list=[('S', 'G'), ('S','A'),`
	`                                 ('A', 'S'), ('A', 'G')],`
	`                      starting_nodes=['S'],`
	`                      goal_nodes={'G'})`
	
	`solutions = generic_search(graph, DFSFrontier())`
	`solution = next(solutions, None)`
	`print_actions(solution)`
		`Actions:`
		`  S->A,`
		`  A->G.`
		`Total cost: 2`
			`Actions:`
			`  S->A,`
			`  A->G.`
			`Total cost: 2`
				✔

✔	`from search import *`
	`from student_answer import DFSFrontier`
	
	`graph = ExplicitGraph(nodes=['Knowledge',`
	`                             'Commerce',`
	`                             'Wisdom',`
	`                             'Wealth',`
	`                             'Happiness'],`
	`                      edge_list=[('Knowledge', 'Wisdom'),`
	`                             ('Commerce', 'Wealth'),`
	`                             ('Happiness', 'Happiness')],`
	`                      starting_nodes=['Commerce'],`
	`                      goal_nodes={'Happiness'})`
	
	`solutions = generic_search(graph, DFSFrontier())`
	`solution = next(solutions, 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 10

Correct

Mark 1.00 out of 1.00

BFS Frontier

Write a class `BFSFrontier` such that when an instance of it along with a graph object is passed to `generic_search`, breadth-first search (BFS) is performed.

It is strongly recommended that you use `collections.deque`. Also write `BFSFrontier` as a subclass of `Frontier` class (as opposed to writing it from scratch). This way you can make sure that your implementation has all the methods required by the abstract base class.

Note: if you are interested, as an object-oriented design exercise, you can define an abstract subclass of Frontier that implements the common functionalities in both BFS and DFS and is the parent of both BFSFrontier and DFSFrontier.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from search import *
from student_answer import BFSFrontier

graph = ExplicitGraph(nodes=set('SAG'),
                      edge_list = [('S','A'), ('S', 'G'), ('A', 'G')],
                      starting_nodes = ['S'],
                      goal_nodes = {'G'})

solutions = generic_search(graph, BFSFrontier())
solution = next(solutions, None)
print_actions(solution)</pre></td><td><pre>Actions:
  S->G.
Total cost: 1</pre></td></tr>
<tr><td><pre>from search import *
from student_answer import BFSFrontier

flights = ExplicitGraph(nodes=['Christchurch', 'Auckland',
                               'Wellington', 'Gold Coast'],
                        edge_list = [('Christchurch', 'Gold Coast'),
                                 ('Christchurch','Auckland'),
                                 ('Christchurch','Wellington'),
                                 ('Wellington', 'Gold Coast'),
                                 ('Wellington', 'Auckland'),
                                 ('Auckland', 'Gold Coast')],
                        starting_nodes = ['Christchurch'],
                        goal_nodes = {'Gold Coast'})

my_itinerary = next(generic_search(flights, BFSFrontier()), None)
print_actions(my_itinerary)</pre></td><td><pre>Actions:
  Christchurch->Gold Coast.
Total cost: 1</pre></td></tr>
</table>

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

```
from search import *
class BFSFrontier(Frontier):
    """when an instance of it along with an instance of Graph are passed to
    the generic_search function, the function behaves as a
    breadth-first search algorithm.
    """
    def __init__(self):
        """The constructor takes no argument. It initialises the
        container to an empty stack."""
        self.container = []

    def add(self, path):
        """Adds a new path to the frontier. A path is a sequence (tuple) of
        Arc objects. You should override this method.

        """
        self.container.append(path)

    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
```

```
        StopIteration exception.

        """
        if len(self.container) == 0:
            raise StopIteration
        else:
            return self.container.pop(0)

    def __iter__(self):
        """The object returns itself because it is implementing a __next__
        method and does not need any additional state for iteration."""
        return self
```

	Test	Expected	Got	

✔	from search import *
	from student_answer import BFSFrontier
	
	graph = ExplicitGraph(nodes=set('SAG'),
	                      edge_list = [('S','A'), ('S', 'G'), ('A', 'G')],
	                      starting_nodes = ['S'],
	                      goal_nodes = {'G'})
	
	solutions = generic_search(graph, BFSFrontier())
	solution = next(solutions, None)
	print_actions(solution)
		Actions:
		  S->G.
		Total cost: 1
			Actions:
			  S->G.
			Total cost: 1
				✔

✔	from search import *
	from student_answer import BFSFrontier
	
	graph = ExplicitGraph(nodes=set('SAG'),
	                      edge_list = [('S','G'), ('S', 'A'), ('A', 'G')],
	                      starting_nodes = ['S'],
	                      goal_nodes = {'G'})
	
	solutions = generic_search(graph, BFSFrontier())
	solution = next(solutions, None)
	print_actions(solution)
		Actions:
		  S->G.
		Total cost: 1
			Actions:
			  S->G.
			Total cost: 1
				✔

✔	from search import *
	from student_answer import BFSFrontier
	
	flights = ExplicitGraph(nodes=['Christchurch', 'Auckland',
	                               'Wellington', 'Gold Coast'],
	                        edge_list = [('Christchurch', 'Gold Coast'),
	                                     ('Christchurch','Auckland'),
	                                     ('Christchurch','Wellington'),
	                                     ('Wellington', 'Gold Coast'),
	                                     ('Wellington', 'Auckland'),
	                                     ('Auckland', 'Gold Coast')],
	                        starting_nodes = ['Christchurch'],
	                        goal_nodes = {'Gold Coast'})
	
	my_itinerary = next(generic_search(flights, BFSFrontier()), None)
	print_actions(my_itinerary)
		Actions:
		  Christchurch->Gold Coast.
		Total cost: 1
			Actions:
			  Christchurch->Gold Coast.
			Total cost: 1
				✔

	Test	Expected	Got	

✔	`from search import *`
	`from student_answer import BFSFrontier`
	
	`graph = ExplicitGraph(nodes=set('SABG'),`
	`edge_list = [('S','A'), ('S', 'B'),`
	`('B','S'), ('A', 'G')],`
	`starting_nodes = ['S'],`
	`goal_nodes = {'G'})`
	
	`solutions = generic_search(graph, BFSFrontier())`
	`solution = next(solutions, None)`
	`print_actions(solution)`
		`Actions:`
		`S->A,`
		`A->G.`
		`Total cost: 2`
			`Actions:`
			`S->A,`
			`A->G.`
			`Total cost: 2`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **11**

Correct

Mark 1.00 out of 1.00

Consider a state space graph with the following properties:

- Each state (node) is represented as an integer
- There is only a single starting state.
- For each state (number) *n*, two actions are available (in order): `"1down"` that goes to state *n-1* and `"2up"` which goes to state *n+2*.
- A state is considered a goal node if the number is divisible by 10.

Write a class `FunkyNumericGraph` as a subclass of `Graph` that implements the above description. The constructor should take one numeric argument which represent the single starting state. The cost of outgoing arcs must be set to 1 although the cost is not used during the search.

It can be seen that this graph has infinitely many states and edges and therefore it can never fit in the memory as an explicit graph. The given description, however, is enough to generate the graph on the fly as needed. Also the graph has infinitely many solutions. The object `solutions` in some test cases is an iterator. We use `next` to start/resume the algorithm and get the next solution.

Important: you must also provide the code for `BFSFrontier` from the previous question.

▶ **Spoiler/Hint**: A Sample Incomplete Code (click here to expand)

For example:

Test	Result

`from student_answer import FunkyNumericGraph`
`graph = FunkyNumericGraph(4)`
`for node in graph.starting_nodes():`
`    print(node)`
	`4`

`from student_answer import FunkyNumericGraph`
`graph = FunkyNumericGraph(4)`
`for arc in graph.outgoing_arcs(7):`
`    print(arc)`
	`Arc(tail=7, head=6, action='1down', cost=1)`
	`Arc(tail=7, head=9, action='2up', cost=1)`

`from search import *`
`from student_answer import FunkyNumericGraph, BFSFrontier`
 	
`graph = FunkyNumericGraph(3)`
`solutions = generic_search(graph, BFSFrontier())`
`print_actions(next(solutions))`
`print()`
`print_actions(next(solutions))`
	`Actions:`
	`  1down,`
	`  1down,`
	`  1down.`
	`Total cost: 3`
	
	`Actions:`
	`  1down,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up.`
	`Total cost: 5`

`from search import *`
`from student_answer import FunkyNumericGraph, BFSFrontier`
`from itertools import dropwhile`
 	
`graph = FunkyNumericGraph(3)`
`solutions = generic_search(graph, BFSFrontier())`
`print_actions(next(dropwhile(lambda path: path[-1].head <= 10, solutions)))`
	`Actions:`
	`  1down,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up,`
	`  2up.`
	`Total cost: 10`

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

```
from search import *

class FunkyNumericGraph(Graph):
    """A graph where nodes are numbers. A number n leads to n-1 and
    n+2. Nodes that are divisible by 10 are goal nodes."""

    def __init__(self, starting_number):
        self.starting_number = starting_number

    def outgoing_arcs(self, tail_node):
```

```python
        """Takes a node (which is an integer in this problem) and returns
        outgoing arcs (always two arcs in this problem)"""
        return [Arc(tail_node, tail_node-1, action="1down", cost=1),
                Arc(tail_node, tail_node+2, action="2up", cost=1)]

    def starting_nodes(self):
        """Returns a sequence (list) of starting nodes. In this problem
        the seqence always has one element."""
        return [self.starting_number]

    def is_goal(self, node):
        """Determine whether a given node (integer) is a goal."""
        return node % 10 == 0

class BFSFrontier(Frontier):
    """when an instance of it along with an instance of Graph are passed to
    the generic_search function, the function behaves as a
    breadth-first search algorithm.
    """
    def __init__(self):
        """The constructor takes no argument. It initialises the
        container to an empty stack."""
        self.container = []

    def add(self, path):
        """Adds a new path to the frontier. A path is a sequence (tuple) of
        Arc objects. You should override this method.

        """
        self.container.append(path)

    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.

        """
        if len(self.container) == 0:
            raise StopIteration
```

	Test	Expected	Got	

✔	`from student_answer import FunkyNumericGraph`
	`graph = FunkyNumericGraph(4)`
	`for node in graph.starting_nodes():`
	`    print(node)`
		`4`	`4`	✔

✔	`from student_answer import FunkyNumericGraph`
	`graph = FunkyNumericGraph(4)`
	`for arc in graph.outgoing_arcs(7):`
	`    print(arc)`
		`Arc(tail=7, head=6, action='1down', cost=1)`
		`Arc(tail=7, head=9, action='2up', cost=1)`
			`Arc(tail=7, head=6, action='1down', cost=1)`
			`Arc(tail=7, head=9, action='2up', cost=1)`
				✔

✔	`from search import *`
	`from student_answer import FunkyNumericGraph, BFSFrontier`
	
	`graph = FunkyNumericGraph(3)`
	`solutions = generic_search(graph, BFSFrontier())`
	`print_actions(next(solutions))`
	`print()`
	`print_actions(next(solutions))`
		`Actions:`
		`  1down,`
		`  1down,`
		`  1down.`
		`Total cost: 3`
		
		`Actions:`
		`  1down,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up.`
		`Total cost: 5`
			`Actions:`
			`  1down,`
			`  1down,`
			`  1down.`
			`Total cost: 3`
			
			`Actions:`
			`  1down,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up.`
			`Total cost: 5`
				✔

	Test	Expected	Got	

✔	`from search import *`
	`from student_answer import FunkyNumericGraph, BFSFrontier`
	`from itertools import dropwhile`
	
	`graph = FunkyNumericGraph(3)`
	`solutions = generic_search(graph, BFSFrontier())`
	`print_actions(next(dropwhile(lambda path: path[-1].head <= 10, solutions)))`
		`Actions:`
		`  1down,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up,`
		`  2up.`
		`Total cost: 10`
			`Actions:`
			`  1down,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up,`
			`  2up.`
			`Total cost: 10`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 12

Correct

Mark 1.00 out of 1.00

Sliding puzzles were discussed as an example in the lecture notes. Examples of this puzzle are the 8-puzzle and the 15-puzzle. The puzzle has an *n*-by-*n* board. There are n^2-*1* tiles on the board. The tiles are numbered starting from 1. The tiles can slide into an adjacent blank space. The objective of the game is to arrange the tiles in a way that the top-left corner is blank and then the numbers are ordered from left to right in each row and then from top to bottom.

Write a class `SlidingPuzzleGraph` as a subclass of `Graph` that implements (n squared minus one)-puzzles. States must be a list of *n* rows (lists) each having *n* elements. The first row (with index zero) is on the top. Complete the template provided in the answer box. You must also provide the code for `BFSFrontier`.

Note: since we assume that your search strategy (frontier class) is a simple BFS (and without more advanced features such as pruning), the test cases only contain relatively small solvable puzzles. You will later see more advanced search strategies capable of solving more challenging instances of this problem.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>from student_answer import SlidingPuzzleGraph, BFSFrontier
from search import generic_search, print_actions

graph = SlidingPuzzleGraph([[1, 2, 5],
                            [3, 4, 8],
                            [6, 7, ' ']])

solutions = generic_search(graph, BFSFrontier())
print_actions(next(solutions))</pre></td><td><pre>Actions:
  Move 8 down,
  Move 5 down,
  Move 2 right,
  Move 1 right.
Total cost: 4</pre></td></tr>
<tr><td><pre>from student_answer import SlidingPuzzleGraph, BFSFrontier
from search import generic_search, print_actions

graph = SlidingPuzzleGraph([[3,' '],
                            [1, 2]])

solutions = generic_search(graph, BFSFrontier())
print_actions(next(solutions))</pre></td><td><pre>Actions:
  Move 3 right,
  Move 1 up,
  Move 2 left,
  Move 3 down,
  Move 1 right.
Total cost: 5</pre></td></tr>
<tr><td><pre>from student_answer import SlidingPuzzleGraph, BFSFrontier
from search import generic_search, print_actions

graph = SlidingPuzzleGraph([[1, ' ', 2],
                            [6, 4, 3],
                            [7, 8, 5]])

solutions = generic_search(graph, BFSFrontier())
print_actions(next(solutions))</pre></td><td><pre>Actions:
  Move 4 up,
  Move 3 left,
  Move 5 up,
  Move 8 right,
  Move 7 right,
  Move 6 down,
  Move 3 left,
  Move 4 down,
  Move 1 right.
Total cost: 9</pre></td></tr>
</table>

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

Reset answer

```
from search import *
import copy

BLANK = ' '

class SlidingPuzzleGraph(Graph):
    """Objects of this type represent (n squared minus one)-puzzles.
    """

    def __init__(self, starting_state):
        self.starting_state = starting_state

    def outgoing_arcs(self, state):
        """Given a puzzle state (node) returns a list of arcs. Each arc
        represents a possible action (move) and the resulting state."""

        n = len(state) # the size of the puzzle
```

```python
        # Find i and j such that state[i][j] == BLANK
        for a in range(n):
            for b in range(n):
                if state[a][b] == ' ':
                    i, j = a, b

        arcs = []
        if i > 0:
            action = "Move {} down".format(state[i-1][j]) # or blank goes up
            new_state = copy.deepcopy(state)
            new_state[i][j], new_state[i-1][j] = new_state[i-1][j], BLANK
            arcs.append(Arc(state, new_state, action, 1))
        if i < n - 1:
            action = "Move {} up".format(state[i+1][j]) # or blank goes down
            new_state = copy.deepcopy(state)
            new_state[i][j], new_state[i+1][j] = new_state[i+1][j], BLANK
            arcs.append(Arc(state, new_state, action, 1))
        if j > 0:
            action = "Move {} right".format(state[i][j-1]) # or blank goes left
            new_state = copy.deepcopy(state)
            new_state[i][j], new_state[i][j-1] = new_state[i][j-1], BLANK
            arcs.append(Arc(state, new_state, action, 1))
        if j < n - 1:
            action = "Move {} left".format(state[i][j+1]) # or blank goes left
            new_state = copy.deepcopy(state)
            new_state[i][j], new_state[i][j+1] = new_state[i][j+1], BLANK
            arcs.append(Arc(state, new_state, action, 1))
        return arcs

    def starting_nodes(self):
        return [self.starting_state]

    def is_goal(self, state):
```

	Test	Expected	Got	

✔	`from student_answer import SlidingPuzzleGraph, BFSFrontier`
	`from search import generic_search, print_actions`
	
	`graph = SlidingPuzzleGraph([[1, 2, 5],`
	`                            [3, 4, 8],`
	`                            [6, 7, ' ']])`
	
	`solutions = generic_search(graph, BFSFrontier())`
	`print_actions(next(solutions))`
		Actions:
		Move 8 down,
		Move 5 down,
		Move 2 right,
		Move 1 right.
		Total cost: 4
			Actions:
			Move 8 down,
			Move 5 down,
			Move 2 right,
			Move 1 right.
			Total cost: 4
				✔

✔	`from student_answer import SlidingPuzzleGraph, BFSFrontier`
	`from search import generic_search, print_actions`
	
	`graph = SlidingPuzzleGraph([[3,' '],`
	`                            [1, 2]])`
	
	`solutions = generic_search(graph, BFSFrontier())`
	`print_actions(next(solutions))`
		Actions:
		Move 3 right,
		Move 1 up,
		Move 2 left,
		Move 3 down,
		Move 1 right.
		Total cost: 5
			Actions:
			Move 3 right,
			Move 1 up,
			Move 2 left,
			Move 3 down,
			Move 1 right.
			Total cost: 5
				✔

✔	`from student_answer import SlidingPuzzleGraph, BFSFrontier`
	`from search import generic_search, print_actions`
	
	`graph = SlidingPuzzleGraph([[1, ' ', 2],`
	`                            [6,   4,  3],`
	`                            [7,   8,  5]])`
	
	`solutions = generic_search(graph, BFSFrontier())`
	`print_actions(next(solutions))`
		Actions:
		Move 4 up,
		Move 3 left,
		Move 5 up,
		Move 8 right,
		Move 7 right,
		Move 6 down,
		Move 3 left,
		Move 4 down,
		Move 1 right.
		Total cost: 9
			Actions:
			Move 4 up,
			Move 3 left,
			Move 5 up,
			Move 8 right,
			Move 7 right,
			Move 6 down,
			Move 3 left,
			Move 4 down,
			Move 1 right.
			Total cost: 9
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.