Started on	Monday, 31 July 2023, 6:31 PM
State	Finished
Completed on	Friday, 4 August 2023, 10:47 AM
Time taken	3 days 16 hours
Marks	22.00/22.00
Grade	**1.00** out of 1.00 (**100**%)

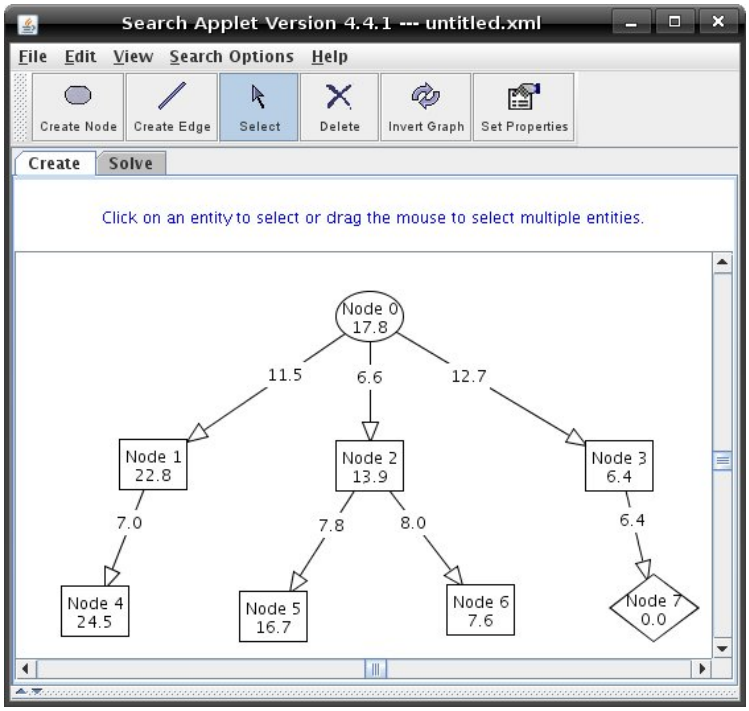
Information

Optional activity: using the search applet to explore cost-sensitive and heuristic algorithms

The search applet is not directly used in any of the questions in this quiz (or future quizzes) but you may find it useful in understanding cost-based and heuristic algorithms. Remember to study the relevant sections of the textbook and the lecture notes first. The jar file of the applet is available at http://www.aispace.org/downloads.shtml and the documentation and tutorials are available at http://www.aispace.org/search/index.shtml.

In order to create your own graphs follow the same procedure as that in the previous quiz. Node heuristics are specified when you create new nodes. Arc costs are specified when you create new arcs. If you want to change the node heuristics or arc costs in a graph, go to "Create" mode and depress "Set Properties" then select the node or the arc you want to change. Two viewing options that you may find helpful are "Show Node Heuristics" and "Show Edge Costs" which are available from the "View" menu.

If you would like to set the nodes heuristics to be the distance from the node to the goal node (on the canvas), go to the 'Search Options' menu and click on 'Set Node Heuristics Automatically'. If you would like to set the edge costs to be the distance (on the canvas) from the two nodes the edge is on, go to the 'Search Options' menu and click on 'Set Edge Costs Automatically'. After creating a few nodes and edges, and after enabling the "Show Node Heuristics" or "Show Edge Costs" check boxes under the view menu, your graph might look something like this:



Here are some sample problems (graphs) related to cost-sensitive and heuristic search that can be loaded by selecting "Load Sample Problem" from the "File" menu:

1. **Bicycle Courier Problem (cyclic or acyclic)**: Try solving this problem with lowest-cost-first search (LCFS) and A* search with various pruning options. Pruning options can be set through the "Search Options" menu.
2. **Misleading Heuristic Demo**: Try solving this problem using best-first search (with no loop detection or pruning) and observe the behaviour of the algorithm. Observe how loop detection or multiple-path pruning can help best-first search solve the problem. What search algorithms can solve this problem without pruning?
3. **Delivery Robot (cyclic)**: First try finding a solution using depth-first search (DFS). Can DFS find a solution? Does cycle-checking (or multiple-path pruning) help solve the problem? How about breadth-first search (BFS)? Does BFS need pruning in order to find an optimal solution? Does pruning (cycle-checking or multiple path pruning) reduce the number of nodes need to be expanded by BFS?
4. **Multiple-Path Pruning Demo**: First try solving this problem without pruning. Does it make any difference if you used lowest-cost-first search (LCFS) or breadth-first search (BFS)? How many nodes are expanded until a solution is found? Can cycle (loop) checking reduce the number of nodes that need to be expanded before a solution is found? How about multiple-path pruning?

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 1

Correct

Mark 2.00 out of 2.00

Given the following graph, trace the frontier of a lowest-cost-first search (LCFS).

```
ExplicitGraph(
    nodes={'A', 'B', 'C', 'D', 'G'},
    edge_list=[('A', 'B', 3), ('A', 'C', 2), ('B', 'D', 1),
               ('C', 'G', 6), ('D', 'G', 3),],
    starting_nodes=['A'],
    goal_nodes = {'G'},
    )
```

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

```
+ A, 0
- A, 0
+ AB, 3
+ AC, 2
- AC, 2
+ ACG, 8
- AB, 3
+ ABD, 4
- ABD, 4
+ ABDG, 7
- ABDG, 7
```

+A,0 (OK)
-A,0 (OK)
+AB,3 (OK)
+AC,2 (OK)
-AC,2 (OK)
+ACG,8 (OK)
-AB,3 (OK)
+ABD,4 (OK)
-ABD,4 (OK)
+ABDG,7 (OK)
-ABDG,7 (OK)

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question 2

Correct

Mark 2.00 out of 2.00

Given the following graph, trace the frontier of a lowest-cost-first search (LCFS) with pruning.

This question does not apply penalty to wrong answers.

```
ExplicitGraph(
    nodes={'A', 'B', 'C', 'D', 'G'},
    edge_list=[('A', 'B', 2), ('A', 'C', 3), ('B', 'D', 5),
               ('C', 'D', 5), ('D', 'G', 3),],
    starting_nodes=['A'],
    goal_nodes = {'G'},
    )
```

Answer: (penalty regime: 0 %)

```
1  + A, 0
2  - A, 0    #C(A)
3  + AB, 2
4  + AC, 3
5  - AB, 2    #C(AB)
6  + ABD, 7
7  - AC, 3    #C(ABC)
8  + ACD, 8
9  - ABD, 7    #C(ABCD)
10 + ABDG, 10
11 - ACD, 8!
12 - ABDG, 10
```

+A,0 (OK)
-A,0 (OK)
+AB,2 (OK)
+AC,3 (OK)
-AB,2 (OK)
+ABD,7 (OK)
-AC,3 (OK)
+ACD,8 (OK)
-ABD,7 (OK)
+ABDG,10 (OK)
-ACD,8! (OK)
-ABDG,10 (OK)

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question 3

Correct

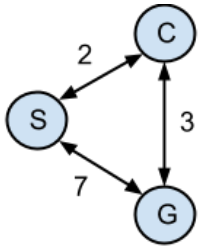
Mark 2.00 out of 2.00

Given the following graph, trace the frontier of a lowest-cost-first search (LCFS) without pruning.

Assume an alphabetic ordering over nodes such that if for example, a path leading to "S" is being expanded and "A" and "B" are neighbours of "S", then the edge (S,A) comes before (S, B); in other words, (S,A) is added to the frontier before (S,B).

Note that all the edges in this graph are bidirectional (i.e. they represent two directed edges with the same cost).

The starting state is 'S' and the goal state is 'G'.



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

```
+ S, 0
- S, 0
+ SC, 2
+ SG, 7
- SC, 2
+ SCG, 5
+ SCS, 4
- SCS, 4
+ SCSC, 6
+ SCSG, 11
- SCG, 5
```

+S,0 (OK)
-S,0 (OK)
+SC,2 (OK)
+SG,7 (OK)
-SC,2 (OK)
+SCG,5 (OK)
+SCS,4 (OK)
-SCS,4 (OK)
+SCSC,6 (OK)
+SCSG,11 (OK)
-SCG,5 (OK)

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question 4

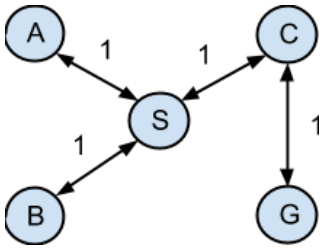
Correct

Mark 2.00 out of 2.00

Given the following graph, trace the frontier of a lowest-cost-first search (LCFS) with multiple-path pruning. The starting state is 'S' and the goal state is 'G'.

Assume an alphabetic ordering over nodes such that if, for example, a path leading to "S" is being expanded and "A" and "B" are neighbours of "S", then the edge (S,A) comes before (S, B); in other words, (S,A) is added to the frontier before (S,B).

Note that all the edges in this graph are bidirectional (i.e. each one represents two directed edges).



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

```
+ S, 0
- S, 0 #C(S)
+ SA, 1
+ SB, 1
+ SC, 1
- SA, 1 #C(AS)
+ SAS, 2!
- SB, 1 #C(ABS)
+ SBS, 2!
- SC, 1 #C(ABCS)
+ SCG, 2
+ SCS, 2!
- SCG, 2
```

+S,0 (OK)
-S,0 (OK)
+SA,1 (OK)
+SB,1 (OK)
+SC,1 (OK)
-SA,1 (OK)
+SAS,2! (OK)
-SB,1 (OK)
+SBS,2! (OK)
-SC,1 (OK)
+SCG,2 (OK)
+SCS,2! (OK)
-SCG,2 (OK)

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question 5

Correct

Mark 2.00 out of 2.00

Given the following graph, trace the frontier of a A-star search without pruning. The h-value of nodes are given in `estimates`.

```
ExplicitGraph(
    nodes={'A', 'B', 'C', 'D', 'G'},
    estimates={'A':5, 'B':5, 'C':9, 'D':1, 'G':0},
    edge_list=[('A','B',2), ('A','C',3), ('B','D',5), ('C','D',10), ('D','G',3)],
    starting_nodes=['A'],
    goal_nodes={'G'},
    )
```

Note: we suggest that you solve this type of problems without drawing the graph, however, if drawing it makes it more understandable to you, draw this graph by putting the nodes along the same line in this order: CABDG. This way the length of the arcs would be more proportional to their costs and the heuristic values may make more sense.

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

```
+ A, 5
- A, 5
+ AB, 7
+ AC, 12
- AB, 7
+ ABD, 8
- ABD, 8
+ ABDG, 10
- ABDG, 10
```

+A,5 (OK)
-A,5 (OK)
+AB,7 (OK)
+AC,12 (OK)
-AB,7 (OK)
+ABD,8 (OK)
-ABD,8 (OK)
+ABDG,10 (OK)
-ABDG,10 (OK)

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question 6

Correct

Mark 2.00 out of 2.00

Consider the explicit graph in the previous question and select the correct statement(s).

Select one or more:

- [ ] A-star with pruning would have produced a non-optimal solution (path to goal).
- [ ] A-star expanded as many paths on this graph as LCFS would have.
- [x] LCFS would have produced an optimal solution for this graph. ✔
- [x] The heuristic function is admissible. ✔
- [x] The heuristic function is consistent. ✔

Your answer is correct.

Correct

Marks for this submission: 2.00/2.00.

Question 7

Correct

Mark 2.00 out of 2.00

Given the following graph, trace the frontier of a A-star search with pruning. The h-value of nodes are given in `estimates`. When there are multiple start nodes, all of them must be added to the frontier in the order of `starting_nodes` list.

```
ExplicitGraph(
    nodes={'A', 'B', 'C', 'D', 'G'},
    estimates={'A':6, 'B':3, 'C':2, 'D':1, 'G':0},
    edge_list=[('C','D',3), ('B','D',4), ('A','D',5), ('D', 'G', 2)],
    starting_nodes=['B', 'C', 'A'],
    goal_nodes={'G'},
    )
```

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

```
+ B, 3
+ C, 2
+ A, 6
- C, 2 #C
+ CD, 4
- B, 3 #BC
+ BD, 5
- CD, 4 #BCD
+ CDG, 5
- BD, 5!
- CDG, 5
```

+B,3 (OK)
+C,2 (OK)
+A,6 (OK)
-C,2 (OK)
+CD,4 (OK)
-B,3 (OK)
+BD,5 (OK)
-CD,4 (OK)
+CDG,5 (OK)
-BD,5! (OK)
-CDG,5 (OK)

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Information

The search module is used for graph search. It is available here: search.py. You can use the module for your local development and testing. This module is also available on the server (where your program gets tested). Therefore you do not need to upload the module when submitting your answer (however do include the import statement).

Question 8

Correct

Mark 4.00 out of 4.00

In this question, you have to write a graph class for an agent that can be at certain locations on a 2D plane. The graph class is called `LocationGraph`. An object of this class is initialised with a dictionary called `location` and a non-negative number called `radius`. The keys of the dictionary are of type string and are the nodes of the graph. The value of each key is a pair of numbers that designates a location for that node.

From a node, there will be an outgoing arc to every other node in the graph that is within the given `radius`. The *action* field of the arc must of the form `"A->B"` where `A` and `B` are placeholders for the tail node and the head node of the arc respectively. The cost of the arc is the straight-line distance between the tail node and the head node. The outgoing arcs of a node must be in the alphabetical order of the head nodes.

Notes

- The search module is available here: search.py.
- On a 2D plane, two points are within a radius *r* of each other if (and only if) the euclidean distance between the two points is less than or equal to *r*.
- While from a scalability perspective, it makes sense to use an efficient data structure for querying points on a plane or to cache the edges explicitly, for this exercise, simply iterate through the location dictionary every time the outgoing arcs of a node need to be computed.
- Remember to include all the import statements in the solution (or just paste your entire file if it does not produce unwanted output when it is imported by another module).

For example:

Test	Result

`from student_answer import LocationGraph`
 	
`graph = LocationGraph(`
`  location={'A': (0, 0),`
`  'B': (3, 0),`
`  'C': (3, 4),`
`  'D': (7, 0),},`
`  radius = 5,`
`  starting_nodes=['A'],`
`  goal_nodes={'C'}`
`)`
 	
`for arc in graph.outgoing_arcs('A'):`
`  print(arc)`
 	
`print()`
 	
`for arc in graph.outgoing_arcs('B'):`
`  print(arc)`
 	
`print()`
 	
`for arc in graph.outgoing_arcs('C'):`
`  print(arc)`
	`Arc(tail='A', head='B', action='A->B', cost=3.0)`
	`Arc(tail='A', head='C', action='A->C', cost=5.0)`
	
	`Arc(tail='B', head='A', action='B->A', cost=3.0)`
	`Arc(tail='B', head='C', action='B->C', cost=4.0)`
	`Arc(tail='B', head='D', action='B->D', cost=4.0)`
	
	`Arc(tail='C', head='A', action='C->A', cost=5.0)`
	`Arc(tail='C', head='B', action='C->B', cost=4.0)`

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

Reset answer

```
from search import Arc, Graph
from math import sqrt

def distance(tup1, tup2):
    return ((tup1[0]-tup2[0])**2 + (tup1[1]-tup2[1])**2)**0.5

class LocationGraph(Graph):
    def __init__(self, location, radius, starting_nodes, goal_nodes):
        self.location = location
        self.radius = radius
        self._starting_nodes = starting_nodes
        self.goal_nodes = goal_nodes

    def starting_nodes(self):
        return self._starting_nodes

```

```
    def is_goal(self, node):
        return node in self.goal_nodes

    def outgoing_arcs(self, tail):
        arcs = []
        for key, loc in sorted(self.location.items()):
            if key != tail and distance(self.location[tail], loc) <= self.radiu
                arcs.append(Arc(tail, key, f"{tail}->{key}", distance(self.loca
        return arcs
```

	Test	Expected	Got	

✔	`from student_answer import LocationGraph`
	
	`graph = LocationGraph(`
	`  location={'A': (0, 0),`
	`  'B': (3, 0),`
	`  'C': (3, 4),`
	`  'D': (7, 0),},`
	`  radius = 5,`
	`  starting_nodes=['A'],`
	`  goal_nodes={'C'}`
	`)`
	
	`for arc in graph.outgoing_arcs('A'):`
	`  print(arc)`
	
	`print()`
	
	`for arc in graph.outgoing_arcs('B'):`
	`  print(arc)`
	
	`print()`
	
	`for arc in graph.outgoing_arcs('C'):`
	`  print(arc)`
		`Arc(tail='A', head='B', action='A->B', cost=3.0)`
		`Arc(tail='A', head='C', action='A->C', cost=5.0)`
		
		`Arc(tail='B', head='A', action='B->A', cost=3.0)`
		`Arc(tail='B', head='C', action='B->C', cost=4.0)`
		`Arc(tail='B', head='D', action='B->D', cost=4.0)`
		
		`Arc(tail='C', head='A', action='C->A', cost=5.0)`
		`Arc(tail='C', head='B', action='C->B', cost=4.0)`
			`Arc(tail='A', head='B', action='A->B', cost=3.0)`
			`Arc(tail='A', head='C', action='A->C', cost=5.0)`
			
			`Arc(tail='B', head='A', action='B->A', cost=3.0)`
			`Arc(tail='B', head='C', action='B->C', cost=4.0)`
			`Arc(tail='B', head='D', action='B->D', cost=4.0)`
			
			`Arc(tail='C', head='A', action='C->A', cost=5.0)`
			`Arc(tail='C', head='B', action='C->B', cost=4.0)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 4.00/4.00.

Question 9

Correct

Mark 4.00 out of 4.00

Write a class `LCFSFrontier` such that when an instance of this class along with a graph object is passed to `generic_search`, lowest-cost-first search (LCFS) is performed. Your answer must also include the code for `LocationGraph` class since it is used in some test cases.

Notes

- The search module is available here: search.py.
- We require priority queues to be stable. We recommend you use `heapq` from the standard Python library. Read the documentation and pay attention to the "implementation notes" to see how you can make it stable.
- It is recommended that you write `LCFSFrontier` as a subclass of `Frontier` class (instead of writing it from scratch). Although the `Frontier` class does not provide any functionality to reuse, subclassing makes it easier to check whether the new class implements all the methods required by the abstract base class.

For example:

Test	Result

`from search import Arc`
`from student_answer import LCFSFrontier`
 	
`frontier = LCFSFrontier()`
`frontier.add((Arc(None, None, None, 17),))`
`frontier.add((Arc(None, None, None, 11), Arc(None, None, None, 4)))`
`frontier.add((Arc(None, None, None, 7), Arc(None, None, None, 8)))`
 	
`for path in frontier:`
`  print(path)`
	`(Arc(tail=None, head=None, action=None, cost=11), Arc(tail=None, head=None, action=None, cost=4))`
	`(Arc(tail=None, head=None, action=None, cost=7), Arc(tail=None, head=None, action=None, cost=8))`
	`(Arc(tail=None, head=None, action=None, cost=17),)`

`from search import *`
`from student_answer import LocationGraph, LCFSFrontier`
 	
`graph = LocationGraph(`
`  location={'A': (25, 7),`
`  'B': (1, 7),`
`  'C': (13, 2),`
`  'D': (37, 2)},`
`  radius=15,`
`  starting_nodes=['B'],`
`  goal_nodes={'D'}`
`)`
 	
`solution = next(generic_search(graph, LCFSFrontier()))`
`print_actions(solution)`
	`Actions:`
	`  B->C,`
	`  C->A,`
	`  A->D.`
	`Total cost: 39.0`

`from search import *`
`from student_answer import LCFSFrontier`
 	
`graph = ExplicitGraph(nodes=set('ABCD'),`
`  edge_list=[('A', 'D', 7), ('A', 'B', 2),`
`  ('B', 'C', 3), ('C', 'D', 1)],`
`  starting_nodes=['A'],`
`  goal_nodes={'D'})`
 	
`solution = next(generic_search(graph, LCFSFrontier()))`
`print_actions(solution)`
	`Actions:`
	`  A->B,`
	`  B->C,`
	`  C->D.`
	`Total cost: 6`

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

```
1  from search import *
2  from math import sqrt
3
4  def distance(tup1, tup2):
5      return sqrt((tup1[0]-tup2[0])**2 + (tup1[1]-tup2[1])**2)
6
7  class LocationGraph(Graph):
8      def __init__(self, location, radius, starting_nodes, goal_nodes):
```

```
    def __init__(self, location, radius, starting_nodes, goal_nodes):
        self.location = location
        self.radius = radius
        self._starting_nodes = starting_nodes
        self.goal_nodes = goal_nodes

    def starting_nodes(self):
        return self._starting_nodes

    def is_goal(self, node):
        return node in self.goal_nodes

    def outgoing_arcs(self, tail):
        arcs = []
        for key, loc in sorted(self.location.items()):
            if key != tail and distance(self.location[tail], loc) <= self.radiu
                arcs.append(Arc(tail, key, f"{tail}->{key}", distance(self.loca
        return arcs

def f(path):
    total = 0
    for arc in path:
        total += arc.cost
    return total

class LCFSFrontier(Frontier):
    def __init__(self):
        """The constructor takes no argument. It initialises the
        container to an empty stack."""
        self.container = []

    def add(self, path):
        self.container.append(path)
        self.container.sort(key=f)
        #for i in range(1, len(self.container)):
            #if f(self.container[-i]) < f(self.container[-(i+1)]):
                #self.container[-i], self.container[-(i+1)] = self.container[-(



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

	Test	Expected	Got	

✔	`from search import Arc`
	`from student_answer import LCFSFrontier`
	
	`frontier = LCFSFrontier()`
	`frontier.add((Arc(None, None, None, 17),))`
	`frontier.add((Arc(None, None, None, 11), Arc(None, None, None, 4)))`
	`frontier.add((Arc(None, None, None, 7), Arc(None, None, None, 8)))`
	
	`for path in frontier:`
	`  print(path)`
		`(Arc(tail=None, head=None, action=None, cost=11), Arc(tail=None, head=None, action=None, cost=4))`
		`(Arc(tail=None, head=None, action=None, cost=7), Arc(tail=None, head=None, action=None, cost=8))`
		`(Arc(tail=None, head=None, action=None, cost=17),)`
			`(Arc(tail=None, head=None, action=None, cost=11), Arc(tail=None, head=None, action=None, cost=4))`
			`(Arc(tail=None, head=None, action=None, cost=7), Arc(tail=None, head=None, action=None, cost=8))`
			`(Arc(tail=None, head=None, action=None, cost=17),)`
				✔

	Test	Expected	Got	

✔	`from search import *`
	`from student_answer import LocationGraph, LCFSFrontier`
	
	`graph = LocationGraph(`
	`  location={'A': (25, 7),`
	`  'B': (1, 7),`
	`  'C': (13, 2),`
	`  'D': (37, 2)},`
	`  radius=15,`
	`  starting_nodes=['B'],`
	`  goal_nodes={'D'}`
	`)`
	
	`solution = next(generic_search(graph, LCFSFrontier()))`
	`print_actions(solution)`
		`Actions:`
		`  B->C,`
		`  C->A,`
		`  A->D.`
		`Total cost: 39.0`
			`Actions:`
			`  B->C,`
			`  C->A,`
			`  A->D.`
			`Total cost: 39.0`
				✔

✔	`from search import *`
	`from student_answer import LCFSFrontier`
	
	`graph = ExplicitGraph(nodes=set('ABCD'),`
	`  edge_list=[('A', 'D', 7), ('A', 'B', 2),`
	`  ('B', 'C', 3), ('C', 'D', 1)],`
	`  starting_nodes=['A'],`
	`  goal_nodes={'D'})`
	
	`solution = next(generic_search(graph, LCFSFrontier()))`
	`print_actions(solution)`
		`Actions:`
		`  A->B,`
		`  B->C,`
		`  C->D.`
		`Total cost: 6`
			`Actions:`
			`  A->B,`
			`  B->C,`
			`  C->D.`
			`Total cost: 6`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 4.00/4.00.