Tutorial 2

Name: Zhaokun Su

Breakout group: 4

Zid: z5235878

(i) Try running the various search algorithms to make sure you understand their properties (as above), their implementations and the relationships between the algorithms. You can edit searchTest.py to call the search methods on the Romania map problem.

Using the Romania map, we could call this 5 below search methods on this problem:

```
******* romania_map

A*:
6 paths have been expanded and 10 paths remain in the frontier
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
there are 0 elements remaining on the queue with f-value= 418

A* with MPP:
6 paths have been expanded and 10 paths remain in the frontier
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
there are 0 elements remaining on the queue with f-value= 418

Branch and bound (with too-good initial bound of 418.01 )
Number of paths expanded: 6 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
Rerunning B&B
Number of paths expanded: 5 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Branch and bound (with not-very-good initial bound of 846 )
Number of paths expanded: 21 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
Rerunning B&B
Number of paths expanded: 5 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Depth-first search: (Use ^C if it goes on forever)
6 paths have been expanded and 8 paths remain in the frontier
Path found: Arad --> Zerind --> Oradea --> Sibiu --> Fagaras --> Bucharest   cost= 607

Process finished with exit code 0
```

(ii) The supplied code adds successors to the frontier in the order they are defined in the associated definition of the Romania map in searchProblem.py. By experimenting with different orderings of the successors (such as alphabetical ordering, as above), determine how sensitive the algorithms are to this ordering.

Before changing the orderings, the result is shown in the (i) part.

We can change ordering in specific Arc list by adding the following code in the searchGeneric.py file, and we sort every node's neighbors by their **Alphabetical order**.

```
        Returns None if no path exists.
        """
        while not self.empty_frontier():
            path = self.frontier.pop()
            self.display(2, "Expanding:", path, "(cost:", path.cost, ")")
            self.num_expanded += 1
            if self.problem.is_goal(path.end()):  # solution found
                self.display(1, self.num_expanded, "paths have been expanded and",
                             len(self.frontier), "paths remain in the frontier")
                self.solution = path  # store the solution found
                return path
            else:
                neighs = self.problem.neighbors(path.end())
                neighs = sorted(neighs, key=lambda x: (x.from_node, x.to_node))
                self.display(3, "Neighbors are", neighs)
                for arc in reversed(list(neighs)):
                    self.add_to_frontier(Path(path, arc))
                self.display(3, "Frontier:", self.frontier)
        self.display(1, "No (more) solutions. Total of",
                     self.num_expanded, "paths expanded.")


import heapq  # part of the Python standard library
from searchProblem import Path


class FrontierPQ(object):
    """A frontier consists of a priority queue (heap), frontierpq, of
        (value, index, path) triples, where
    * value is the value we want to minimize (e.g., path cost + h).
```

Rerunning this program to test our method:

We get a different result as following:

```
A*:
6 paths have been expanded and 10 paths remain in the frontier
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
there are 0 elements remaining on the queue with f-value= 418

A* with MPP:
6 paths have been expanded and 10 paths remain in the frontier
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
there are 0 elements remaining on the queue with f-value= 418

Branch and bound (with too-good initial bound of 418.01 )
Number of paths expanded: 6 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
Rerunning B&B
Number of paths expanded: 5 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Branch and bound (with not-very-good initial bound of 846 )
Number of paths expanded: 21 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
Rerunning B&B
Number of paths expanded: 5 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Depth-first search: (Use ^C if it goes on forever)
```

The program searching method for DFS seems like going to infinite loop and never end, which means if we change our neighbor nodes order to alphabetical order, DFS searching will not find a final solution for this problem.

Eventually, we terminate our program by ^C:

```
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Depth-first search: (Use ^C if it goes on forever)
Traceback (most recent call last):
  File "/Users/steve/Desktop/2020T2/comp9414/tutorials/aipython/searchTest.py", line 57, in <m
    run(searchProblem.romania_map, "romania_map")
  File "/Users/steve/Desktop/2020T2/comp9414/tutorials/aipython/searchTest.py", line 50, in ru
    print("Path found:", tsearcher.search(), "  cost=", tsearcher.solution.cost)
  File "/Users/steve/Desktop/2020T2/comp9414/tutorials/aipython/searchGeneric.py", line 59, in
    self.display(3, "Frontier:", self.frontier)
  File "/Users/steve/Desktop/2020T2/comp9414/tutorials/aipython/display.py", line 17, in displ
    def display(self,level,*args,**nargs):
KeyboardInterrupt
```

In conclusion, for this tutorial code, the **Arc order matters and ordering is sensitive to our searching appoarch, especially for DFS.**

(iii) (Harder) Although breadth-first search is a special case of A∗ search (explain how!), write a class BreadthFirstSearcher extending Searcher in searchGeneric.py that implements breadth-first search directly, i.e. maintains a set of states from nodes previously expanded and only adds a neighbour of an expanded node to the frontier if its state is not in the explored set or in a node already on the frontier, and terminates when a goal state is generated (not expanded). You will need to code the constructor and the search method.

Write an extra class BFSearcher (Inherited class Searcher):

```
class BFSearcher(Searcher):
    def __init__(self, problem):
        super().__init__(problem)
        self.frontier_city = []
        self.expanded_city = []
        self.add_to_frontier_city(Path(problem.start_node()).end())

    def add_to_frontier_city(self, city):
        self.frontier_city.append(city)

    def add_to_expanded_city(self, city):
        self.expanded_city.append(city)
```

And we define BFS method like below: city means nodes, which is evey city passing through:

```
@visualize
def search(self):
    path = Path(None)
    while not self.empty_frontier():
        path = self.frontier.pop(0)
        self.add_to_expanded_city(path.end())

        self.frontier_city.remove(path.end())
        self.display(2, "Expanding:", path, "(cost:", path.cost, ")")
        self.num_expanded += 1

        neighbors = self.problem.neighbors(path.end())
        self.display(3, "Neighbors are", neighbors)

        for arc in neighbors:
            if (arc.to_node not in self.expanded_city) or (arc.to_node in self.frontier_city):
                self.add_to_frontier(Path(path, arc))
                self.frontier_city.append(Path(path, arc).end())
                self.display(3, "Frontier:", self.frontier)
                if self.problem.is_goal(arc.to_node):  # solution found
                    self.display(1, self.num_expanded, "paths have been expanded and",
                                 len(self.frontier), "paths remain in the frontier")
                    self.solution = Path(path, arc)
                    return Path(path, arc)
```

Finally, it turns out that BFS costs is 450.

And we find an anther path which passes through only 4 cities.

```
A*:
6 paths have been expanded and 10 paths remain in the frontier
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
there are 0 elements remaining on the queue with f-value= 418

A* with MPP:
6 paths have been expanded and 10 paths remain in the frontier
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
there are 0 elements remaining on the queue with f-value= 418

Branch and bound (with too-good initial bound of 418.01 )
Number of paths expanded: 6 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
Rerunning B&B
Number of paths expanded: 5 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Branch and bound (with not-very-good initial bound of 846 )
Number of paths expanded: 21 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest   cost= 418
Rerunning B&B
Number of paths expanded: 5 (optimal solution found)
Path found: Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest

Depth-first search: (use ^C if it goes on forever)
6 paths have been expanded and 8 paths remain in the frontier
Path found: Arad --> Zerind --> Oradea --> Sibiu --> Fagaras --> Bucharest   cost= 607

Breadth-first search:
6 paths have been expanded and 4 paths remain in the frontier
Path found: Arad --> Sibiu --> Fagaras --> Bucharest   cost= 450
```