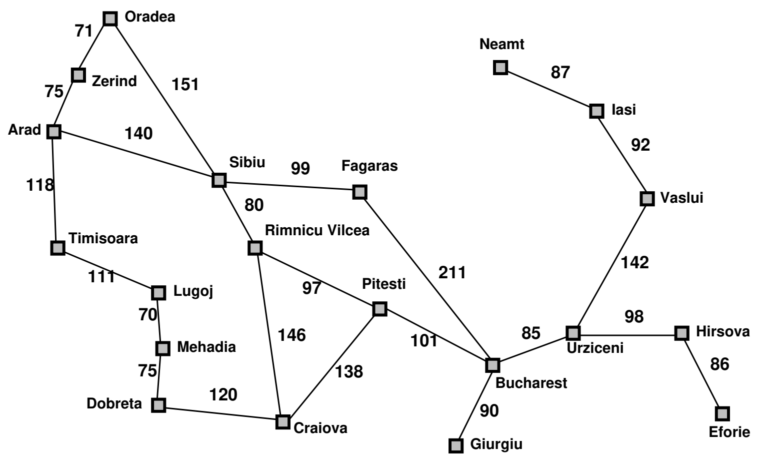
COMP9414: Artificial Intelligence
Tutorial 2: Search

1. This exercise concerns the route-finding problem using the Romania map from Russell & Norvig (*Artificial Intelligence: A Modern Approach*) as an example.



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Define the route-finding problem (from Arad to Bucharest) as a state space search problem (give short descriptions of the state space, etc., in English). What order are nodes in the search graph expanded (give the associated states) for each of the following algorithms when searching for a (shortest) path between Arad and Bucharest? Assume the successors of a state are returned in alphabetical order. Make sure you understand the key properties of the different algorithms listed below.

To clarify, for breadth-first search, stop the search when a node with the goal state is generated and use a check to ensure that nodes with the same state as a previously expanded node are not added to the frontier. For the other search algorithms, stop the search when a node with the goal state is expanded; for uniform-cost search include a check that a node with the same state as a previously expanded node is not added to the frontier (as in breadth-first search) and a test so that only one node for a given state is stored on the frontier (that with the shortest path to that state), and for depth-first search and its variants use cycle checking along a path to avoid repeated states that can lead to infinite branches.

(i) Depth-first search (efficient use of space but may not terminate)

(ii) Breadth-first search (space inefficient, guaranteed to find a solution)

(iii) Uniform-cost search (similar to breadth-first, but order nodes by cost)

(iv) Iterative deepening depth-first search (space efficient, but repeated work)

(v) Greedy best-first search (efficient, not guaranteed optimal solution)

(vi) A* search with straight-line distance heuristic (inefficient, guaranteed optimal solution)

Which algorithm is suitable in practice for solving route-finding problems such as this?

2. This version of the map (from the first edition of the book) differs from that in the second edition of the book in that the heuristic value for Fagaras is 178 rather than 176, and that for Pitesti is 98 rather than 100 (also Drobeta is mistakenly spelt as Dobreta). What difference does this make?

3. **Programming.** Consider the Python code for generic search algorithms `searchGeneric.py`.

   (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.

   (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.

   (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.