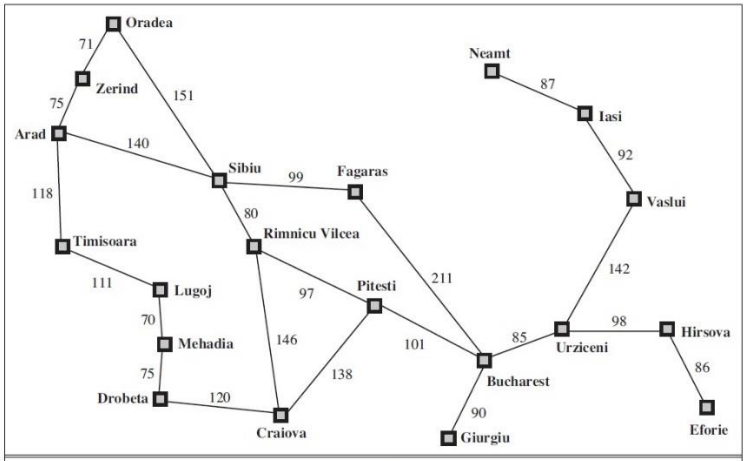
ETGG 4803 **Lab #2 (30 points)**
Graph Traversal and Path Search **Due 9 February 2020**

Path planning involves finding an optimal path between two locations in a known, partially known, or unknown environment. Search performance might be described by completeness, optimality, time complexity, or space complexity.

Consider the following simplified map from *Artificial Intelligence: A Modern Approach* (Third Edition) by Russell and Norvig.



A simplified road map of part of Romania.

Problem 0. Suppose we are in Arad and wish to find the shortest path to Bucharest. What do we do?

Dijkstra's algorithm [Dijkstra (1956/1959)]

Dijkstra's algorithm is a single-source shortest-path algorithm.

STEP 1 Mark all nodes as *unvisited*.

STEP 2 Set the distance for the start node to 0 and the temporary distance for all other nodes to ∞. Set the start node as *current*.

STEP 3 For the current node, consider each unvisited neighbour *v* and calculate the temporary distance through *current*. Compare the temporary distance to the current distance for *v* and assign the smaller of the two as the new temporary distance for *v*.

STEP 4 When all the neighbours of *current* have been updated, mark *current* as *visited*.

STEP 5 If the destination node has been marked *visited* or if the smallest temporary distance amongst unvisited nodes is ∞ then STOP.

STEP 6 Otherwise, select the unvisited node with smallest temporary distance, set it as *current*, and go to STEP 3.

Problem 1. Implement Dijkstra's algorithm in a language of your choice that is available on *repl.it*.

Problem 2. Apply Dijkstra's algorithm to **Problem 0**. Include a visual component which shows the progress of the algorithm.

Problem 3. Modify the output of this version of Dijkstra's algorithm to find the desired path.

A* search [Hart, Nilsson, and Raphael (1968)]

A* is a generalization of Dijkstra's algorithm. It is similar to Dijkstra's algorithm in that accumulated distances are used but adds an important component to the decision-making process.

To decide which path to extend, A* chooses the path that minimises

$$f(n) = g(n) + h(n)$$

where n is the next node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a *heuristic function* that estimates the cost of the cheapest path from n to the goal. A heuristic function provides additional knowledge of the problem to the search algorithm. A well-chosen heuristic function for the task at hand is vital to success.

A heuristic function h is *admissible* if h never overestimates the actual cost to get to the goal. If $w(x, y)$ represents the weight of edge xy, then a heuristic function h is *consistent* if $h(a) \leq w(a, b) + h(b)$ for every edge ab.

Can you give a nontrivial heuristic function that is always admissible for **Problem 0**?

If the heuristic function is admissible, then A* is guaranteed to return a least-cost path from the start node to the goal. If the heuristic function is also consistent, then A* will return an optimal path without processing any node more than once.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

A* search is identical to Dijkstra's algorithm except that A* minimises the function f in STEP 3.

Problem 4. Implement the A* algorithm in a language of your choice that is available on *repl.it*.

Problem 5. Apply A* to **Problem 0**. Again, include a visual component which shows the progress of the algorithm.

Problem 6. Modify the output of this version of A* to find the desired path.