20CS6033/4033 AI – I
Fall 2024
Instructor: Anca Ralescu

Homework Assignment #1
Assigned on September 8, 2024
Due on September 18, 2024
11:59PM on Canvas
50 points

In this assignment you are asked to implement *uninformed and informed search algorithms* for the Romanian road map data given in your textbook.

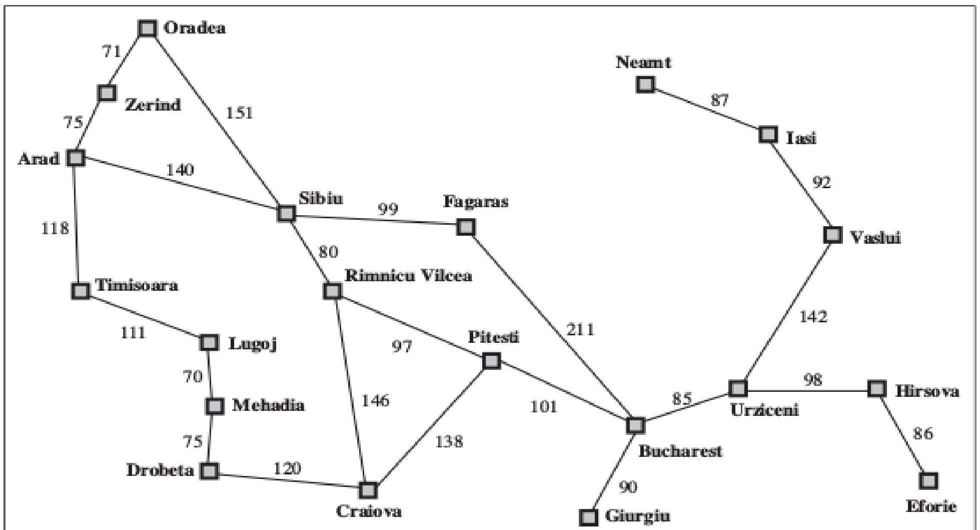

Figure 3.2 A simplified road map of part of Romania.

For the informed search, when the goal city is Bucharest, the straight-line distance heuristic is given in the textbook.

When the goal city is different from Bucharest, one must estimate the straight-line distance (SLD) by using the triangle inequality from elementary geometry (the side of a triangle is less than or equal to the sum of the other two sides). **Consider at least two ways of doing this, and hence two different heuristics.**

You will implement:
(1) Breadth first
(2) Depth first
(3) Best first (greedy algorithm)
(4) A* algorithm

and compare their performances from two points of view:

(1) Correctness (i.e., finds the path from a start city to the goal city), or returns empty if no path exists.
(2) Efficiency (consider the number of cities visited before the path is found or the algorithm returns that there is no path (in this latter case, you may want to put a bound on how many times the same cities are revisited); you may also time the algorithm but since the problem is small (small number of cities), you may have to run each algorithm in a loop of say 100 times, and time the loop).

You may use python or Matlab.

Note that in each type of search the same high-level strategy is used as shown in the pseudocode from the textbook: *maintain the nodes to be expanded in the fringe* – always implemented as a queue or as a priority queue. The difference is on the criterion on how the queue is constructed:

(1) In *depth first*, the **first** child of the current node is put in the *front* of the queue.
(2) In *breadth first*, **all** the children of the current node are put in the at the *back* of the queue.
(3) In best first, the queue is maintained in *nondecreasing order of the SLD,* $h(n)$*, of the* children of the current city to the goal city.
(4) In A* the queue is maintained in *nondecreasing order of the evaluation function* $f(n) = g(n) + h(n)$ of the children of the current city to the goal city. Note that there is some backtracking taking place here, because the algorithm may need to discard a current node and backtrack to a previous step.

Each time a node is retrieved from the front of the queue, the algorithm tests if it is the goal node, and when the test returns true, the algorithm ends.

To begin with, a path is empty. Eventually, the path must start with the start node, and each node is added to it according to a search specific criterion. The path as a list of cities.

When there is no path, the algorithm returns fail or an empty list.

Usually, a network as shown in the road map is represented by a weighted adjacency matrix (the weights show the distances between two cities). Here, we have 21 cities, hence the adjacency matrix would be a 21 x 21. One way is to arrange your cities in alphabetical order and then number them 1, 2, …, 21.

For example, assume that our set of cities is {Sibiu, Fagaras, Rimnicu Vilcea, Pitesti, Bucharest}.

Numbered based on the alphabetical order they are:

Bucharest	Fagaras	Pitesti	Rimnicu Vilcea	Sibiu
1	2	3	4	5

A table representation of the intercity distances is then

	1	2	3	4	5
1	0	211	101		
2	211	0			99
3	101		0	97	
4			97	0	80
5		99		80	0

Note the empty cells – they correspond to when there is no direct road between the two corresponding cities.

As a matrix, this data would be represented as

$$A = \begin{matrix} 0 & 211 & 101 & inf & inf \\ 211 & 0 & inf & inf & 99 \\ 101 & inf & 0 & 97 & inf \\ inf & inf & 97 & 0 & 80 \\ inf & 99 & inf & 80 & 0 \end{matrix}$$

With $A(i,j) = A(j,i)$; *inf* denotes "infinity" to be represented by a very large value (*maxint* for example). Also note the waste in storage (the matrix is symmetric).

A better way of storing the road map information is by using weighted *adjacency lists*. For our small example, this would be as follows:

1	(2,211)	(3, 101)
2	(5, 99)	(1, 211)
3	(4, 97)	(1,101)
4	(5, 80)	(3,97)
5	(2,99)	(4,80)

Decide the representation you use.

REPORT AND ANALYSIS

Write a report detailing:

1. **Implementation Details:**
   - How the algorithms were implemented.
   - Description of the heuristic functions used.
2. **Experimental Results:**
   - Tables and graphs showing the performance of each algorithm and heuristic.
   - Comparisons of time and space complexity.

SUBMISSION REQUIREMENTS

1. **Code:** Submit the code files for all implementations.
2. **Report:** A PDF report with your findings, analysis, and comparisons. At the top of the report include the following:
   a) **the names and credit of each team member, as follows:** Name: x% of the work, or the statement "all team members contributed in equal measure"
   b) **Honor statement:** "In completing this assignment, all team members have followed the honor pledge specified by the instructor for this course".
   c) **Bibliography:** List all the bibliographic sources that were used. If no such sources just write "None".
3. **Execution Instructions:** Clear instructions on how to run your code.

GRADING CRITERIA

1. **Correctness (50%):** Accurate implementation of the algorithms and correct application of heuristics.
2. **Performance Analysis (25%):** Clear and insightful comparison of algorithm performance.
3. **Code Quality (15%):** Clean, well-documented, and efficient code.
4. **Report Quality (10%):** Well-structured, clear, and comprehensive report with in-depth analysis.