SGN-44006: Artificial Intelligence **(Due: 24/01/20)**

Exercise set 2

Instructor: Tapio Elomaa
Course Assistant: Aashish Sah

Problem 1: Pen & Paper

(6 points)

Consider again the same problem presented in last week's exercise.

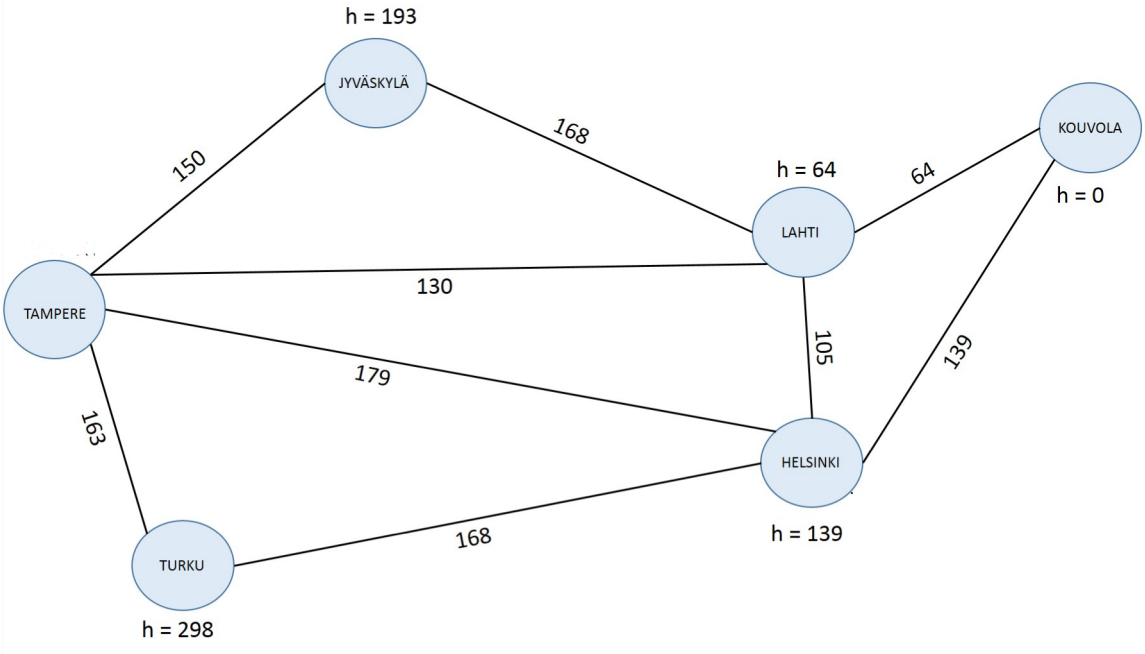

Figure 1: Route graph for Tampere-Kouvola

For each of the following graph search strategies, find out the order in which the states are expanded and also path returned by graph search. The start and goal states are Tampere and Kouvola respectively. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first.

(a) Uniform cost search.

(b) Greedy search with the heuristic h shown on the graph.

(c) A* search with the same heuristic.

Problem 2: Programming (6 points)

We will continue the same project as last week but this time we will implement uniform cost search and A* search algorithm. You will use the same python files as last week and edit **search.py** under uniform cost search and A* search method. You can use the information from the last week's problem set.

(a) Uniform cost search

While BFS will find a fewest-actions path to the goal, we might want to find paths that are "best" in other senses. Consider **mediumDottedMaze** and **mediumScaryMaze**.

By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response.

Implement the uniform-cost graph search algorithm in the **uniformCostSearch** function in **search.py**. We encourage you to look through util.py for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

>> python3 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
>> python3 pacman.py -l mediumDottedMaze -p StayEastSearchAgent
>> python3 pacman.py -l mediumScaryMaze -p StayWestSearchAgent

Note: You should get very low and very high path costs for the **StayEastSearchAgent** and **StayWestSearchAgent** respectively, due to their exponential cost functions (see searchAgents.py for details).

(b) A* search

Implement A* graph search in the empty function **aStarSearch** in **search.py**. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The **nullHeuristic** heuristic function in **search.py** is a trivial example.

You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py).

>> python3 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

You should see that A* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly). What happens on *openMaze* for the various search strategies?

Bonus: Min-max and Alpha-beta pruning (10 points)

You can find all the details regarding the problem here.

Note: The due date for the submission of bonus task is set to 31.01.2020.