EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATH. - NATURWISS. FAKULTÄT
FACHBEREICH INFORMATIK
KOGNITIVE SYSTEME · PROF. A. ZELL

Artificial Intelligence

Assignment 3

Assignment due by: 15.11.2017, Discussion: 17.11.2017

Question 1 Greedy best-first search (6 points)

For this question use the **tree search** version of greedy best-first search.

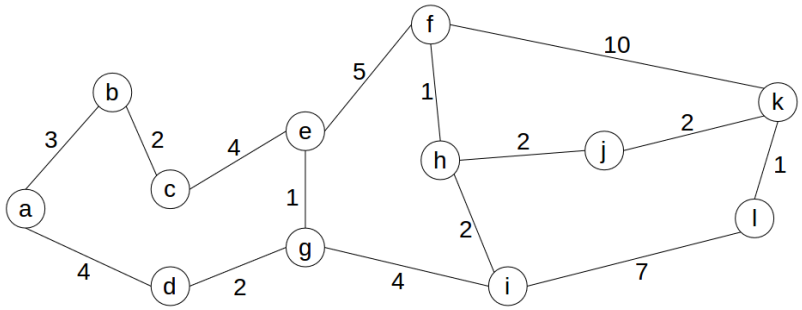


Table 1.

n	$h(n)$	n	$h(n)$
a	16	g	10
b	17	h	5
c	13	i	4
d	12	j	1
e	8	k	1
f	3	l	0

Table 2.

n	$h(n)$	n	$h(n)$
a	14	g	9
b	14	h	3
c	13	i	5
d	11	j	1
e	10	k	0
f	5	l	1

Table 3.

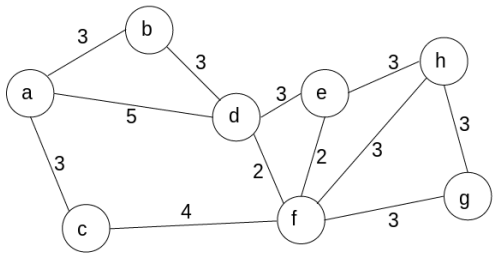
n	$h(n)$	n	$h(n)$
a	14	g	8
b	13	h	2
c	11	i	4
d	3	j	0
e	6	k	1
f	4	l	1

(a) Using the heuristic function shown in Table 1, is it possible to find a path between the node **a** and the target **l**? Is the path optimal? Justify your answer.

(b) Using the heuristic function shown in Table 2, is it possible to find a path between the node **c** and the target **k**? Is the path optimal? Justify your answer.

(c) Using the heuristic function shown in Table 3, is it possible to find the path from **a** to **j**? Is the path optimal? What can you conclude about the Greedy best-first search algorithm? Justify your answer.

Question 2 Pathfinding with A* (6 points)



n	$h(n)$
a	9
b	8
c	6
d	5
e	2
f	3
g	1
h	0

Find the shortest path between the node $\mathbf{a}$ and the target node $\mathbf{h}$ using A* by hand. Detail each intermediate step and indicate the f-values. Is the shortest path optimal? (2 points)

(a) Is there a unique shortest path? If not indicate all other path(s). (1 point)

(b) Explain in less than 4 lines the difference between a consistent heuristic and an admissible heuristic. (1 point)

(c) Modify the heuristic value of node $\mathbf{d}$ to make it inconsistent. Is the new heuristic still admissible? (1 point)

(d) Can you draw a small graph with at most 4 nodes whose heuristic is inconsistent but still admissible? (1 point)

Question 3 Programming in LISP (2+3+1+1+1 = 8 points)

In this exercise we will be implementing A* graph search in LISP. When running A* we need to keep track of two datastructures: the *frontier*, a priority queue of nodes that we need to visit and *explored*, the set of nodes that we have already visited and thus don't want to visit again. While keeping track of this 'state' may not seem like it would be very nice to implement in a functional language, it can be very nicely accomplished with a function that takes this state as arguments and calls itself recursively with the updated values of the state for the next step of the algorithm.

Download the file *graphsearch-astar.lisp*, which contains a graph of German cities and the distances between them, as well as the coordinates of each city and some functions to access that information. In addition it contains some auxiliary functions and to help you with the implementation and function stubs to show you what arguments the functions should take.

(a) Write a function `expand-node` that takes the current node, the heuristic function and the set of already explored nodes and returns the list of the new nodes to be added to the frontier.

(b) Implement a search step for A* in a function `search-step`: It should use a priority queue to represent the frontier and use `expand-nodes` to generate the nodes to be added to the frontier. The recursive call of the function should compute the next search step.

(c) Implement an admissible heuristic function for A* graph search based on the geographical data available for the graph. ($h(n) = 0$ is not an acceptable answer for this question).

(d) Using your functions from (a), (b) and (c), implement A* graph search. For illustrative purposes and to make debugging easier, the function should return three things: The lenght of the shortest path that was found, the shortest path itself, in order from start to goal and the list of all the nodes that were visited in the search.

(e) Dijkstra's algorithm can be expressed as A* with a heuristic $h(n) = 0$ and can also be used to just explore the graph and get a list of nodes in ascending order of distance from an origin point. Using this, write a function that generates a list of all cities reachable from a start city in ascending order of shortest path distance.