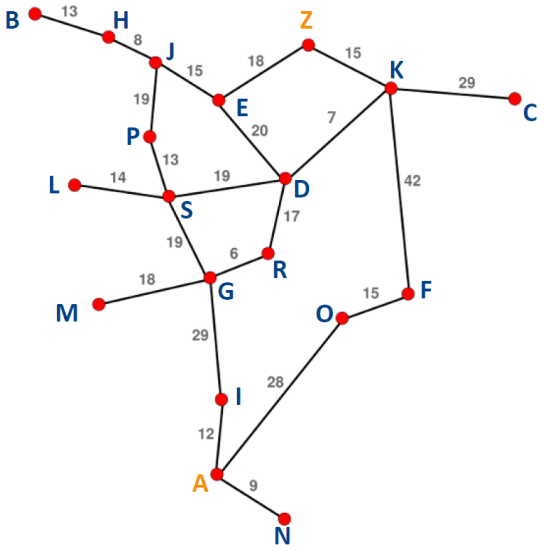
Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Lecture Chair XII for Natural Language Processing, Universität Würzburg

7. Exercise for "Algorithmen, KI & Data Science 1"

1 Two Friends Problem

Suppose two friends live in different cities on a map. On every turn, we must **simultaneously** move each friend to a neighboring city on the map. The amount of time needed to move from city i to neighbor j is equal to the road distance $d(i, j)$ between the cities, but on each turn, the friend that arrives first must wait until the other one arrives before the next turn can begin. We want the two friends to meet as quickly as possible (shortest amount of time).



1. Formulate this problem by giving (a) initial state, (b) goal test, (c) successor function, and (d) cost function.

(a) Start state s_o can be any pair of nodes $(i, j) \in VxV$, where V represent the set of cities.

(b) Goal state is reached if $i == j$

(c) Possible successors are all pairs (x, y), where x is an adjacent node of i and y is an adjacent node of j ==> $\{(x, y) | \forall x \in G.Adj[i] \; and \; \forall y \in G.Adj[j]\}$

(d) We are interested in the smallest amount of time until both friends meet. Hence, cost function to go from (i, j) to (x, y) is $max(d(i, x), d(j, y))$.

2. Let $D(i, j)$ be the straight-line distance between cities i and j. Which of the following heuristic functions are admissible/optimistic? (a) $D(i, j)$; (b) $2 \times D(i, j)$; $(c) D(i, j)/2$. Either give a counter-example or show that the heuristic is admissible/optimistic.

Best case: The two friends would always travel straight line distance with equal distance in each step.

(a) $D(i, j)$ is not optimistic: Assume we are in the best case scenario, and we are in the last transition before the gold state (x, x) is reached. The costs are $max(d(i, x), d(j, x)) = D(i, x)$ or equivalently $(= D(j, x))$. However, the heuristic results in $D(i, j) = d(i, x) + d(x, j) = 2 \times D(i, x)$ or equivalently $(= d(j, x) + d(x, i) = 2 \times D(j, x))$ ==> heuristic overestimates the true costs.

(b) $2 \times D(i, j)$ is not optimistic: previous argument is applicable analogously.

(c) $D(i, j)/2$ is optimistic: Same argument as for (a), but by dividing the heuristic by 2, it becomes optimistic.

3. Are there completely connected maps for which no solution exists?

Yes, if there are two nodes that are connected with each other. The two friends will switch the cities on every transition (because they have to move.) The same holds for any chain with even number of cities (as long as the friends start at the ends of the chain).

4. Are there maps in which all solutions require one friend to visit the same city twice?

Adding self-loops to the chains of the previous solution. At least one friend has to use the self-loop once, resulting in visiting a city twice in every solution.

2 Decantation Problem

You are given an 8-liter jar full of water and two empty jars of 5- and 3-liter capacity. You have to get exactly 4 liters of water in one of the jars. You can completely empty a jar into another jar with space or completely fill up a jar from another jar.

1. Formulate this problem giving (a) initial state, (b) goal test, and (c) successor function. Represent the states by a 3-tuple where each element refers to one of the jars (e.g., $(8, 0, 0)$)

   (a) Initial state: The water is in the 8-liter jar ==> (8,0,0)

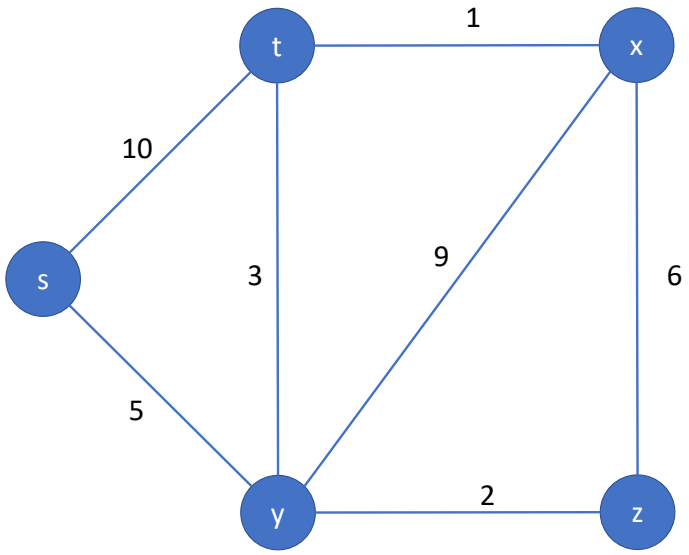
   (b) Gold test: One jar contains exactly 4 liters (one of the elements in the tuple is equal to 4).

   (c) Fill up a jar completely from another jar OR empty a jar completely into another jar. The sum over all jars has to be 8.

2. Implement the functions $succ(n)$ and $goal(s)$ in the provided $.ipynb$. The function $succ(s)$ takes a state s as input (e.g., $(8, 0, 0)$). The function returns a list of possible next states of the same form. One of the possible next states would be $(3, 5, 0)$. The function $goal$ takes a state s as input and returns true if the goal test is successful and false otherwise.
3. Implement the Breadth-First-Search algorithm ($bfs(search\_problem)$ in the provided $.ipynb$) to search the state space graph for a goal state. Keep track of the expanded states and expand each state only once. Return the goal state, number of pourings required and the solution path, if a goal state is found and false otherwise.
4. Implement the Iterative-Deepening-Search algorithm ($ids(search\_problem)$ and $limited - dfs(search\_problem)$ in the provided $.ipynb$) to search the state space graph for a goal state. In $limited\_dfs$, keep track of the expanded states and expand each state only once. Both methods return the goal state, the number of pourings required, and the solution path if a goal state is found and false otherwise.

3 Shortest Path Problem

1. Apply the A^* algorithm (see lecture 15, slides 21-24) to the graph below to find the shortest path and the total cost from s to x. You are given the following heuristics: $h(s) = 9, h(t) = 1, h(y) = 4, h(z) = 13, h(x) = 0$



Init:
$open = [(s, 0 + 9)]$
$visited = \{s : 0\}$

1. Iteration:
$expand(s, 0 + 9)$
$open = [(y, 5 + 4), (t, 10 + 1)]$
$visited = \{s : 0, t : 10, y : 5\}$

2. Iteration:
$expand(y, 0 + 9)$

$open = [(t, 8+1), (x, 14+0), (z, 7+13)]$
$visited = \{s : 0, t : 8, y : 5, x : 14, z : 7\}$

3. Iteration:
$expand(t, 8+1)$
$open = [(x, 9+0), (z, 7+13)]$
$visited = \{s : 0, t : 8, y : 5, x : 9, z : 7\}$

4. Iteration:
$goal(x) == True$

Total cost: 9
shortest-path: $s \rightarrow y \rightarrow t \rightarrow x$

2. Apply the $GreedyBestFirstSearch$ algorithm to the graph from the previous exercise to find the shortest path and the total cost from s to x. You are given the following heuristics: $h(s) = 9, h(t) = 1, h(y) = 4, h(z) = 13, h(x) = 0$

Init:
$open = [(s, 9)]$

1. Iteration:
$expand(s, 9)$
$open = [(t, 1), (y, 4)]$

2. Iteration:
$expand(t, 1)$
$open = [(x, 0), (y, 4), (s, 9)]$

3. Iteration:
$goal(x) == True$

Total cost: 11
shortest-path: $s \rightarrow t \rightarrow x$

3. Implement the A^* algorithm (function $a\_star(search_problem)$ in the provided $.ipynb$) for shortest path problems (similar to the previous exercise). Return the goal state and the total costs if a goal state is found and false otherwise.

Note: The implementation for the classes $PriorityQueue$ (similar to exercise 5) and $GraphSearchProblem$ are given.