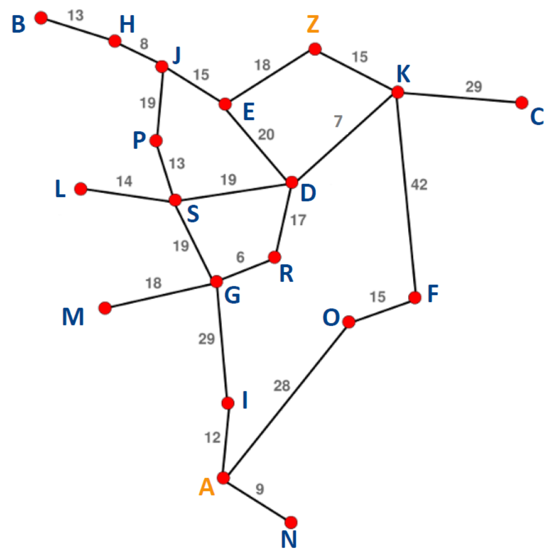
Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Lecture Chair XII for Natural Language Processing, Universität Würzburg

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

1 Two Friends Problem

Suppose two friends live in different cities on a map (similar to the map shown below). 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.

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.
3. Are there completely connected maps for which no solution exists?
4. Are there maps in which all solutions require one friend to visit the same city twice?

2 Decantation Problem

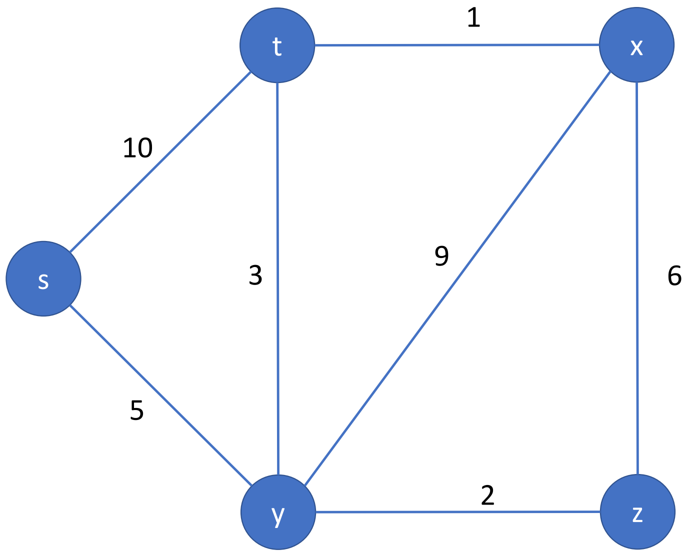
You are given an 8-liter jar full of water and two empty jars of 5- and 3-liter capacity. You have to get exactly 4 liters of water in one of the jars. You can completely empty a jar into another jar with enough space or completely fill up a jar from another jar.

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., $(8, 0, 0)$)
2. Implement the functions $succ(s)$ 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:

$h(s) = 9, h(t) = 1, h(y) = 4, h(z) = 13, h(x) = 0$



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: $h(s) = 9, h(t) = 1, h(y) = 4, h(z) = 13, h(x) = 0$
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.