# Algorithms & Data Structures
# Exercise Sheets

## *Weeks 9-10: Graph Search and Pathfinding*

### Exercises

(1) (Week 9) Simplify the implemented version of Dijkstra's algorithm included in the lecture slides and code handout (method `dijkstra` in class `graph_class.cpp`) to make explicit use of a *priority queue* data structure for its frontier of nodes to be explored. You may use either `std::priority_queue` or your own implementation of the priority queue interface that you did for Exercise 2 in the Week 7-8 exercise solutions.

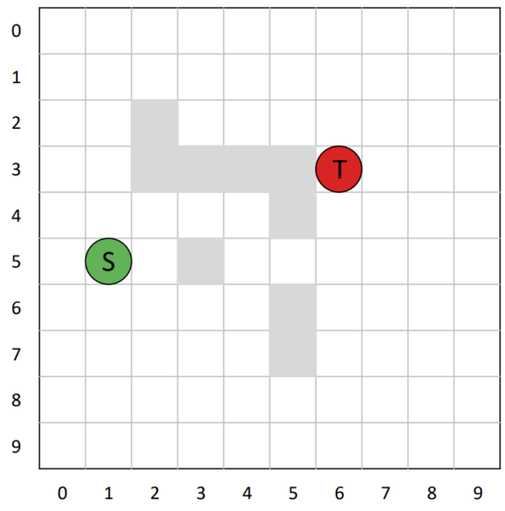
(2) (Week 9) The version of Dijkstra's algorithm that we studied cannot handle negative-weight edges correctly and provides incorrect answers in such cases. The *Bellman-Ford algorithm* modifies Dijkstra's algorithm to also handle those instances correctly, but still cannot handle negative-weight cycles. The algorithm works by relaxing all edges in all iterations instead of just the neighborhood of the vertex with currently the minimum estimated distance. Please solve the tasks below:
a) Give an example of a graph in which Dijkstra's algorithm fails to compute the shortest distance correctly.
b) Implement the Bellman-Ford algorithm. You can find a short video describing it at `https://youtu.be/obWXjtg0L64`. Test with the sample instance from a).
c) Modify the implementation from b) to *detect* negative weight cycles
d) Argue the complexity of the resulting algorithm. (You can find the complexity at the end of the video, but you need to argue)

(3) (Week 9) Computing a single-source shortest path in Direct Acyclic Graphs (DAGs) can be performed more efficiently than in general weighted graphs. The idea is to relax the vertices in the order given by topological sorting.
a) Give an example of a DAG.
b) Modify our implementation of Dijkstra's algorithm to relax the edges from the neighborhood of vertices given in topological sorting order.
c) Use the example from a) to argue the correctness of b).
d) Argue the complexity of the resulting algorithm.

(4) (Week 10) In the grid on the figure below, you can move left, right, up and down, but you cannot move diagonally. The cost of moving from one square to another is 1. You cannot move through the wall (grey shaded area). Make a video explaining how to find a shortest path from S to T using the A* algorithm. You must explain which heuristic you are going to use for this exercise and why. And you must illustrate all the steps of the algorithm. This includes noting for each node $n$ inserted into the frontier:

- The order in which $n$ is inserted in the frontier queue

- The distance/cost so far ($g(n)$)
- The value of the heuristic ($h(n)$)
- The previous node from $n$



(5) (Week 10) Implement the A* algorithm by modifying the version of Dijkstra's algorithm that you did in Exercise 1 (explicitly using a priority queue). Your implementation should work on graphs represented as grids with 4-way movements (up, down, left, right) and you will use the Manhattan 4-way heuristic. Hint: Your priority queue must be able to prioritize between nodes based on $f(n) = g(n) + h(n)$. Test your solution on the graph from Exercise 4.