Algorithms & Data Structures
Exercise Sheets

Weeks 9-10: Graph Search and Pathfinding

Submit your solutions via the peergrade page on Brightspace, Week 9-10. Please make sure to submit your solutions **before the deadline** (typically a Friday, but check the deadline in peergrade). If an exercise requires you to write C++ code, then create a new C++ file for each exercise (i.e. do not combine exercises into a single file); if two or more exercises are about the same ADT, it is OK to use the same file. You MUST hand-in a pdf with the solutions. If code is needed, copy-paste the code into the pdf. Submit the code as a ZIP file, and note in the pdf what file (cpp, h or tpp) that solves the problem. You can hand-in in pairs of two. If an exercise requires other type of material (e.g. video), write the name of the video-file in the pdf and include the video in the Zip file.

Note that you also needs to review both other students' and your own solution. If you handed in as a group, you will have to review your submission individually (I kindly ask you to do it individually, that gives you the best learning outcome). The deadline for reviews is typically Tuesday after the hand-in at 8.00 am (BUT check the actual deadline in peergrade).

Exercises

(1) 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.

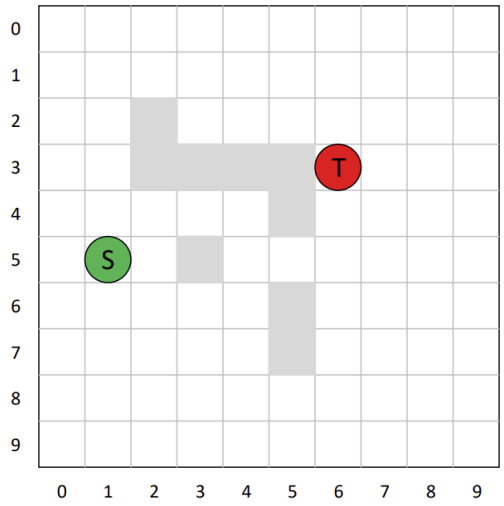
(2) The version of Dijkstra's algorithm that we studied cannot handle negative-weight edges correctly and provides incorrect answers in such cases. The *Bellman-Ford algorithm* modifies Dijkstra's algorithm to also handle those instances correctly, but still cannot handle negative-weight cycles. The 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) 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) 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) 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 must include the heuristic function h as parameter. 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.