**Code Algorithm and Implementation**

Dijkstra algorithm explores a graph by prioritizing the path to be explored. Unlike BFS that explores all possible paths in all directions, Dijkstra prioritizes paths we lower costs. Lower costs encourage moving in that direction, while higher costs are paths to be avoided.

In the Dijkstra algorithm, we want to keep track of the total cost for moving from the start location. So each node has an associated cost (in my implementation, cost\_so\_far) for moving from the start. A node can be visited multiple times, with different costs. Therefore, if we find a lower cost to reach a node we have reached already, we replace the old path with the new path that has a lower cost and we update the cost\_so\_far for that node. We also, obviously, add a node to cost\_so\_far and to came\_from if the node had not been reached before.

In Dijkstra, we also make use of a *priority queue* instead of just a queue. A priority queue allows to re-order the queue based on priority. Therefore, nodes in the queue with lower costs will have a higher priority in the queue. Using this data structure, hence, changes the way the frontier expands.

A\* algorithm, like Dijkstra, uses actual distance explored from the start (cost\_so\_far), however, it also uses a heuristic for the estimated distance to the goal. The heuristic used in this implementation is a Manhattan distance between a node and the goal. In terms of implementation, A\* is very similar to Dijkstra. It also uses a priority queue, for instance. However, the priority of a node in the priority queue is determined not only by the new\_cost (which is cost\_so\_far + graph cost), but it also sums the heuristic. A\* can find an optimal path so long the heuristic does overestimate distances.

**Test Example, Test Result and Explanation**

Chart

Description automatically generated

Dijkstra took 64 steps to reach the goal from the start.

Chart

Description automatically generated

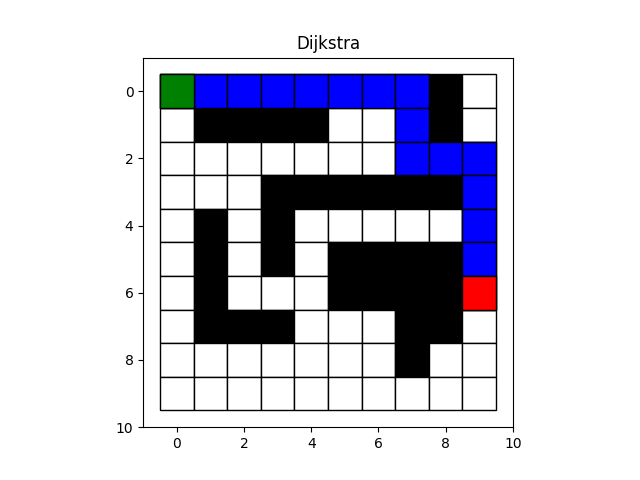
A\* took 44 steps to reach the goal from the start.

Although the resulting path from start to goal was found to be the same using Dijkstra and A\*, A\* took less steps to find the goal. This shows that, in this case, as expected, A\* can find an optimal path so long the heuristic is admissible (does not overestimate distances) while also taking less steps.

I ran another experiment, below were the results:

A picture containing text, crossword puzzle, clipart

Description automatically generated



In this other experiment, with the results above, the paths were found to be the same using both Dijkstra and A\*. However, this time, it took 62 steps to find a path using Dijkstra and only took 37 steps to find a path using A\*. Again, we see A\* perform better than Dijkstra while still finding the optimal path.

Interestingly, in both experiments, Dijkstra seemed to behave like BFS. I plotted the intermediate steps and saw the frontier expansion and it looked very similar to BFS. I believe this is because the cost from adjacent nodes is always 1, therefore, there is no real weight between going to one node versus another one. If, however, the edges were weighted, then we would see a different behavior.

**Resources**

https://www.redblobgames.com/pathfinding/a-star/introduction.html