10/24/2019

40430623

Edinburgh Napier University

Artificial Intelligence

Coursework

# Dijkstras Algorithm

Dijkstra’s is a algorthm used when trying to solve problems with the shortest path. In this particular case the robot is trying to find the cavern, using the shortest path. The Algorithim keeps track of every nodes distance from the start node. It will check the nodes neighbours and set a prospective to equal its neighbour node, plus the cost to get there. If the distance is less it will set it as the new distance and parent to the current node. Then it searches again.

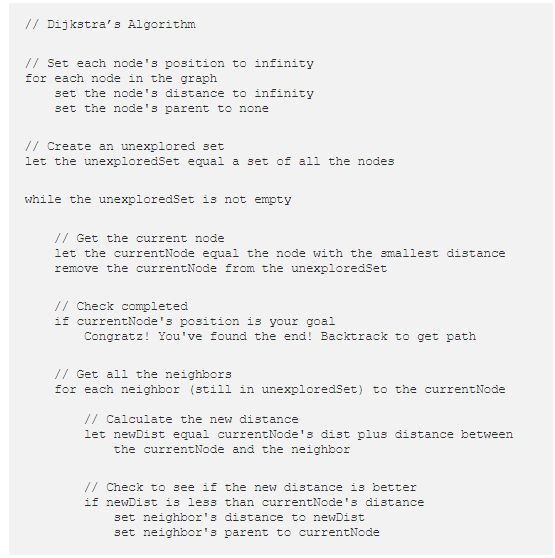
**Implementation:**

First we need to set all the distances to infinity (as we don’t know how far the path is), and add them to an unexplored set, each distance should then be set to 0. The node will then look for a node with the lowest distance and let this be its current node. It will then remove it from the unexplored set. Each time a node comes across a problem it will determine:

Can the node go there? Otherwise it will have to calculate a new distance from the current node and set the potential new distance and set the node’s parent to the current node.

The node will be programed to end when all unexplored sets have been found or failed to find the end node.

Here is some pseudocode:



(Swift, 2017)

Consider **Fig 1.0,** the objective is to get from ‘C- E’. However the algorithm must check nodes and parents. Clearly we can see that the fastest method to get to ‘E’ is ‘2’. However the algorithm must consider going ‘C’ -> ‘D’ -> ‘E’ this being a total of 5 would not be the fastest route. The other route would also be considered is from ‘C’ -> ‘A’ ->‘B’ -> ‘E’, which a total of 10 distance/weight is.

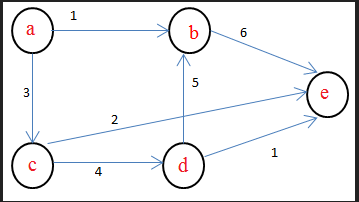


Fig 1.0

**Evaluation of the algorithm**

Dijskras Algorithm is really useful to find the shortest path. In extension A star search is even better as it doesn’t check slower routes by using a heuristic. Dikstras Algorithm finds a route by looking at every route then sorts it from fastest to slowest. This computation time for finding the route can depend on how big the cavern is. Dijsktras Algorthim is used in every day life such as maps and in our telephone systems.

However problems we could encounter is if we have a single or multi directional route, this route could lead to problems as the route itself may not lead anywhere and could cause the algorithm to break. It is important that when creating a map we consider ignoring routes that would not be viable. As lots of time ‘calculating’ the distance/weights it may lead to bugs or crashes in our program.

# A \* Search Algorithm

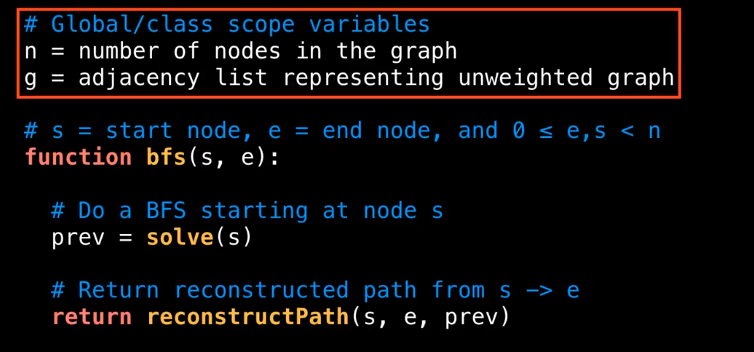
A search would be a good algorithm to use because it will solve the problem in the least amount of time. The algorithm priorities routes which have the fastest path. A star is an extension of Dijskra’s algorithm, but has a lot less computation time due to its heuristic. Consider the **fig 1.0.** Instead of checking all other routes it would ignore any route slower than ‘C’-> ‘E’, which ultimately is the fastest route.

The robot would not have to check all the routes in the cavern as the A search algorithm will only check those routes which are in the ***right*** direction, it would then weigh it up according to its distance and show the fastest route.

**What if the problem changed?**

Breadth First search explores paths which are uncertain (unweighted) and finds the shortest path. The Algorithm explores the data according to layers. It will look at neighbour nodes and adds them to a queue. This is how the algorithm checks which node to visit.

Pseudocode:



First the algorithm reconstructs the path from s - > e. If the node is empty it will then de-queue the node. If the path needs reconstructed it will then return a ‘null ‘value and move on to its neighbouring node.

In conclusion A star search is the best algorithm for finding a path-length/distance however in the terms of completeness, breath-first- search is a better algorithm as it can deal with uncertain weights or uncertain paths.

# Bibliography

algorithm, A. s. (2019). *Wikipedia*. Retrieved from A\* search algorithm: https://en.wikipedia.org/wiki/A\*\_search\_algorithm

Redblobgames. (2016). *Introduction to the A\* Algorithm*. Retrieved from https://www.redblobgames.com/pathfinding/a-star/introduction.html

Swift, N. (2017, March 1). *medium*. Retrieved from https://medium.com/@nicholas.w.swift/easy-dijkstras-pathfinding-324a51eeb0f

Wikipedia. (2019). *Dijkstra's algorithm*. Retrieved from https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm#History