A\* Blog

This post discusses A\*, pronounced A Star, one of the most used pathfinding algorithms. It sees a wide range of use in video games, as well as other graph traversal uses.

History

Before we get into the backstory of A\*, we should discuss Dijkstra briefly. Dijkstra was created in 1956, and then published in 1959. It operates on a wide, breadth first search, that goes down every path until the shortest path is already ending in the goal node. This is a very useful strategy; however, it can be time consuming, and can be improved upon. The original use of A\* was in a lab at Stanford Research Institute in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael. They were attempting to help a newly constructed robot pathfind, and it was using an algorithm that ignored the g(n) of Dijkstra, and instead only focused on a heuristic function h(n) to guide it from its first position to its final position. The group then decided to combine the heuristic that the robot was already using with the g(n) Dijkstra used. Thus, they programmed the robot to use an equation of g(n) + h(n). A\* is so common in the modern day that some sources refer to Dijkstra as a special case of A\*, where the heuristic is always set to zero, even though Dijkstra was created nearly a decade earlier.

A\* Versus Dijkstra

This new formula incredibly increased the time it took the robot to find its path. Just using Dijkstra, it takes longer because it does not add the estimated distance left to the destination, causing it to exhaust all options until the current path is confirmed to be the quickest path. Adding the heuristic to that, it allows it to estimate the distance remaining, and allows it to come to a conclusion quicker. \*Insert gif showing the comparison between the two here\* As can be seen, Dijkstra visits nearly six times as many nodes, and still finds paths of comparable distance. One of the main downsides is that due to A\* using a heuristic, it is possible that Dijkstra finds a faster path, but the comparison in how quickly it finds the path with A\* is considered well worth it in many circumstances, finding a path of comparable time in a fraction of the nodes visited.

Examples

A\* uses the same code as Dijkstra for the base and goes through the exact same process as Dijkstra. The only difference is that the priority queue also includes the heuristic defined values for how much it has left to go. \*Insert the last image on references here\* Going through this with the two styles, starting at point 0 and trying to reach point 4, watch how the priority queue differs with adding the heuristic (The heuristic in this case is a rough measurement from node A to node B). The next item to be looked at in the priority queue is at the top, and tables ignoring the infinity values and the initial step of only the starting node in the queue.

Dijkstra

Viewing from node 0

|  |  |
| --- | --- |
| Node | Value g(n) |
| 0 | 0 |
| 1 | 3 |
| 4 | 7 |
| 3 | 8 |

Viewing from Node 1

|  |  |
| --- | --- |
| Node | Value g(n) |
| 1 | 3 |
| 2 | 4 |
| 4 | 7 |
| 3 | 7 |

Viewing from Node 2

|  |  |
| --- | --- |
| Node | Value g(n) |
| 2 | 4 |
| 3 | 6 |
| 4 | 7 |

Viewing from Node 3

|  |  |
| --- | --- |
| Node | Value g(n) |
| 3 | 6 |
| 4 | 7 |

Viewing from Node 4

|  |  |
| --- | --- |
| Node | Value g(n) |
| 4 | 7 – Goal Reached, Distance is 7 |

A\*

|  |  |
| --- | --- |
| Node | Value g(n) + h(n) |
| 0 | 0 |
| 4 | 7 = 7 |
| 1 | 3 + 5 = 8 |
| 3 | 8 + 4 = 12 |

Viewing from Node 4

|  |  |
| --- | --- |
| Node | Value |
| 4 | 7 = 7, Value determined, Distance is 7 |
| 1 | 3 + 5 = 8 |
| 2 | 4 + 6.5 = 10.5 |
| 3 | 7 + 4 = 11 |

As can be seen, A\* accomplishes the goal in far fewer nodes visited, literally going directly from the first node to the destination node. The pattern it goes through is the exact same, but the heuristic allows it to prioritize nodes seemingly closer to the destination than others.