Path Finding Algorithms(A\*,DFS,BFS,Dijkstra)

Anirudh Agrawal *Coding and Simulation*  
*Robotics and Circuits*Manipal, India  
agrawalanirudh18@gmail.com

*Abstract*—The Following paper discusses about different path finding algorithms like: A\*, DFS, BFS, Dijkstra. The application of the algorithms and the application and efficiency of these algorithms.

Keywords—A\*, DFS, BFS, Dijkstra

# A\* Search Algorithm

One of the best and most widely used methods for pathfinding and graph traversals is the A\* Search algorithm. A\* Search algorithms are said to have "brains" compared to other traversal methods. What it really means is that it is a clever algorithm, setting it apart from other traditional algorithms. It's also important to note that this approach is widely used in games and online maps to locate the shortest distance very efficiently.

## Explaination

Assume that we are given a beginning cell and a target cell in a square grid with numerous obstacles. If possible, we want to go as swiftly from the starting cell to the target cell. Here The A\* Search Algorithm saves our time.

The A\* Search Algorithm selects the node at each stage based on a value, "f," which is a parameter equal to the sum of two additional factors, "g" and "h." It chooses the node or cell with the lowest "f" at each step and processes that node or cell.

We define 'g' and 'h' below as simply as possible:

**g** = the movement cost to move along the created path from the starting point to a particular square on the grid.

h = calculated cost of moving from a specific grid square to the desired location. This is frequently referred to as the heuristic, which is just another word for a smart guess. Due to the possibility of obstructions, we truly can't tell the distance until we find the path.

## Heuristics

Things that we can do:

A) Determine the precise value of h. (which is certainly time consuming).

OR

B) Use some heuristics to estimate the value of h. (less time consuming).

1. Exact Heuristics –

Although we can obtain exact values of h, doing so usually takes a long time.

The ways to determine h's precise value are listed below:

1) Before using the A\* Search Algorithm, pre-calculate the distance between each pair of cells.

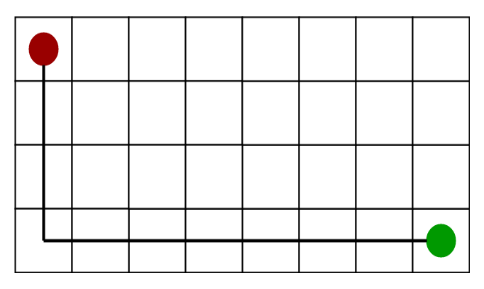
2) Using the distance formula/Euclidean Distance, we may directly determine the precise value of h in the absence of blocked cells or obstructions.

1. Approximation Heuristics –

There are generally three approximation heuristics to calculate h –

1) Manhattan Distance –

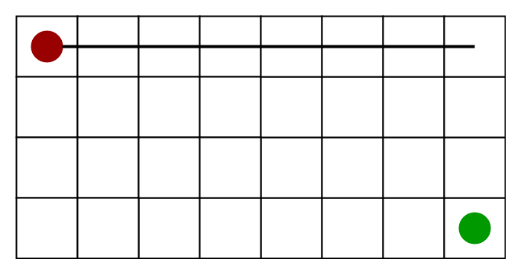
* It is the total of the absolute values of the differences between the x and y coordinates of the goal and the present cell, respectively.
* We should use this method when we are only permitted to move in four directions (right, left, top, bottom)



Source: <https://media.geeksforgeeks.org/wp-content/uploads/a_-search-algorithm-3.png>

2) Diagonal Distance-

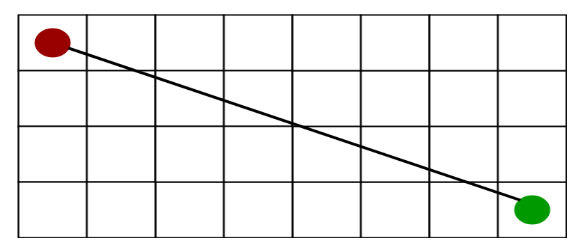
* It is the maximum absolute value of differences between the x and y coordinates of the objective and the current cell, respectively.
* When just eight ways are permitted for us to move (similar to a move of a King in Chess).



Source: <https://media.geeksforgeeks.org/wp-content/uploads/a_-search-algorithm-4.png>

3) Euclidean Distance-

* The distance between the present cell and the destination cell using the distance formula is Euclidean Distance.
* Is to be used when we are free to move whatever we like.



Source: <https://media.geeksforgeeks.org/wp-content/uploads/a_-search-algorithm-5.png>

## Limitations

Although being the best path finding algorithm around, A\* Search Algorithm doesn’t produce the shortest path always, as it relies heavily on heuristics / approximations to calculate – h.

## Applications

Although it was initially intended as a broad graph traversal technique, A\* is frequently utilized for the typical pathfinding problem in applications like video games. It has uses in a variety of issues, such as the stochastic grammars-based parsing issue in NLP. Informational searches combined with online learning are another example.

# DFS Search Algorithm

The depth-first search algorithm, often known as DFS, traverses or investigates data structures like trees and graphs. The algorithm begins at the root node (you may choose any random node to serve as the root node in a network) and analyses each branch as far as it can go before turning around. The Depth First Search (DFS) method explores a network in a deathward motion and employs a stack data structure remember to acquire the next vertex to start a search.

*A. General Concept*

The beginning node for the DFS algorithm is u. The algorithm chooses one of the unvisited nodes from the neighbors of u and starts a recursive call from that node in each step. The process stops for node u after all its neighbors have been checked, and it then loops back to check the neighbors of the node that called node u in the first place.

DFS doesn't visit nodes level by level like the BFS algorithm does. Instead, it continues to go as deeply as it can. The algorithm attempts to move deeper from nodes that are adjacent to the previous node it visited once it reaches its finish.

We'll assume, for the sake of simplicity, that the graph is represented by an adjacency list data structure. All of the mentioned techniques are easily adaptable to work with other data structures.

*B. Limitations*

* Memory Limitations As it stores every node from the current level in preparation for the following level.
* A solution takes time if it is far away.

*C. Applications*

Using depth-first search, one may solve one-solution puzzles like mazes and sudokus as well as scheduling issues, cycle identification in graphs, and topological sorting.

DFS is also a subroutine in graph theory matching techniques like the Hopcroft-Karp method.

Route mapping, scheduling, and discovering spanning trees all require depth-first searches.

# BFS Search Algorithm

The most often used algorithm is called BFS, or breadth-first search.

With BFS, you start at a source node and work your way across the graph, layer by layer, examining the nodes that are closely related to the source node. You must then go to the subsequent-level neighbor nodes in BFS traversal.

The BFS states that you must move along the graph in a breadthwise direction:

* Start by traversing the current layer's nodes horizontally.
* Proceed to the following layer.

## General Concept

A queue data structure is used by Breadth-First Search to hold the node and mark it as "visited" until it has marked all the nearby vertices that are closely related to it. The First In First Out (FIFO) principle governs how the queue functions, thus

the neighbors of each node are inspected beginning with the node that was placed first.

## Features

The BFS Algorithm should be used to explore graph data structures for several reasons. Some of the crucial characteristics that make the BFS algorithm necessary include the following:

* A solution takes time if it is far away.
* A graph can be traversed by the BFS algorithm with the least number of iterations.
* The BFS algorithm's iterations are smooth, hence there is no chance for this technique to get caught in an endless loop.
* The outcome of the BFS algorithm is highly accurate when compared to other algorithms.
* The BFS method aids in node evaluation in a network and establishes the shortest route between nodes.
* The architecture of the BFS algorithm is uncomplicated and dependable.

## Applications

* BFS is utilized in peer-to-peer networks like BitTorrent to locate all neighbor nodes.
* Crawlers from search engines are utilized by BFS to generate indexes. To obtain new pages, it finds all of the links in the source page.
* To locate nearby locations, BFS uses a GPS navigation system.
* The BFS algorithm is used in networking when we wish to broadcast some packets.
* The BFS or DFS path finding algorithm is used.
* The Ford-Fulkerson algorithm uses BFS to determine the network's maximum flow.

## Limitations

* Memory Limitations As it stores every node from the current level in preparation for the following level.

# Dijkstra search algorithm

We can determine the shortest route between any two graph vertices using Dijkstra's approach.

Because not all the vertices of the graph may be included in the shortest distance between two vertices, it is different from the minimal spanning tree.

You may determine the shortest path between two nodes in a graph using Dijkstra's Algorithm. You may create a shortest-path tree by determining the shortest path from a node (referred to as the "source node") to every other node in the graph.

## Basics of Dijkstra

* Dijkstra's Algorithm begins at the node you select (the source node) and examines the graph to determine the shortest path from that node to every other node in the graph.
* The algorithm keeps track of the shortest paths that are currently known to exist between each node and the source node, and it updates these values when a shorter path is discovered.
* The other node is added to the path and designated as "visited" once the algorithm has determined the shortest path between the source node and that other node.
* Up until every node in the graph has been added to the path, the process is repeated. In this manner, a path that connects the source node to every other node and takes the shortest route to each node is created.

## Requirements

The only graphs that Dijkstra's Algorithm can use are those with positive weights. This is since to identify the shortest path, the weights of the edges must be added during the procedure.

The algorithm won't function properly if the network contains a negative weight. The current route to a node is designated as the quickest route to that node once it has been registered as "visited." If the overall weight can be decreased after this step, negative weights can change this.

## Limitations

* The algorithm's primary drawback is that it does a blind search, which wastes a lot of time and resources.
* Its inability to manage sharp edges is another drawback. This results in acyclic networks and frequently makes it impossible to find the ideal shortest path.
* The graph should be weighted and non- negative.

## Applications

* To find the shortest path
* In social networking applications
* In a telephone network
* To find the locations in the map

# Resources

## A\* resourses:

* content: <https://www.geeksforgeeks.org/a-search-algorithm/>
* image1:<https://www.youtube.com/watch?v=JtiK0DOeI4A>
* image2: <https://media.geeksforgeeks.org/wp-content/uploads/a_-search-algorithm-3.png>
* image3: <https://media.geeksforgeeks.org/wp-content/uploads/a_-search-algorithm-4.png>
* image4: <https://media.geeksforgeeks.org/wp-content/uploads/a_-search-algorithm-5.png>

## DFS Resources:

* content: <https://www.youtube.com/watch?v=sTRK9mQgYuc>

## BFS Resourses:

* content: <https://www.youtube.com/watch?v=D14YK-0MtcQ>

## Dijkstra Resourses:

* content:<https://www.programiz.com/dsa/dijkstra-algorithm>