Path Planning Algorithms (DFS, BFS, A\* and Dijkstra)

Aman Agarwal  
*Reg No. 220962104*

Introduction:

Path Planning Algorithms study the given path or the trajectory of a robot and generate either a time or a space efficient path for the robot, as desired by the user, in the joint space or the operating space of the robot. The Algorithm basically helps the robot to find the shortest and most obstacle free path. Path Planning Algorithms usually requires a map of the environment along with start and end points as an input. These maps can be represented in different ways such as grid maps, state spaces and topological roadmaps.

# **DFS (Depth First Search) Algorithm:**

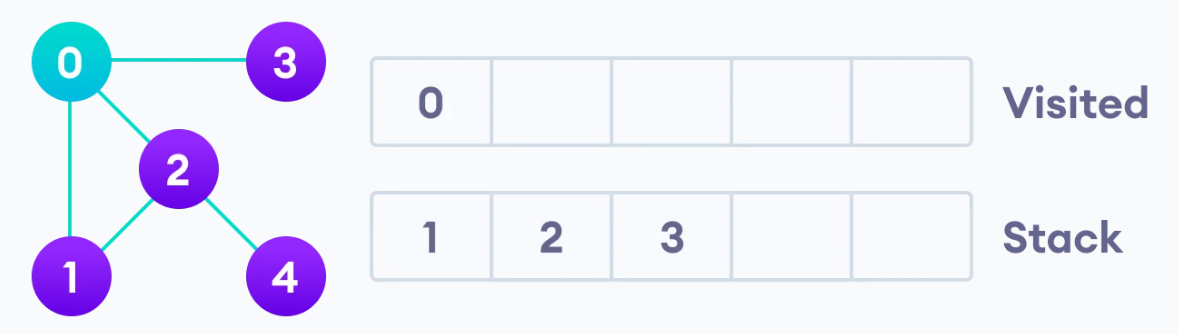
Depth First search is a recursive algorithm for traversing or searching graphs or trees data structures. The algorithm starts at the root node (usually the topmost node in the tree) and goes as far as it can down a branch until it reaches an already visited node or a dead end, then backtracks till it finds an unvisited node and explores it. Traversing through a graph means visiting all the nodes of a graph.

## **EXPLANATION:**

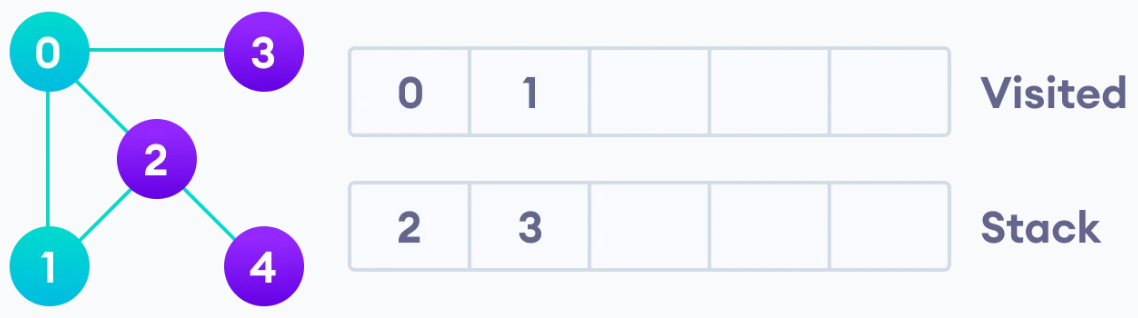
In Depth First search we divide the nodes of a graph into one of the two categories that is visited or not visited. We must keep on exploring the nodes till we encounter a dead end or find a cycle and mark it as visited or not visited. DFS uses a stack data structure to traverse.

Example:

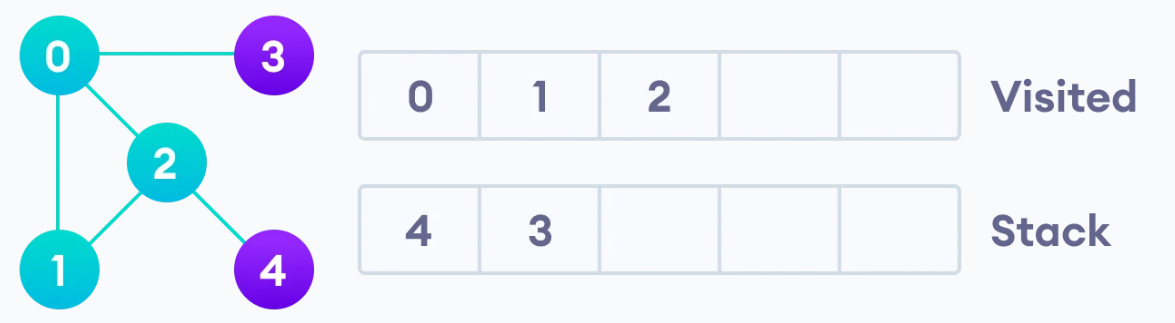
The basic idea is to pick a starting node and push all the adjacent nodes into the stack. For example, if we start with the node 0, then we need to put the adjacent nodes of 0 (1,2,3) in the stack and as we have already visited 0, we will put it in the visited list.



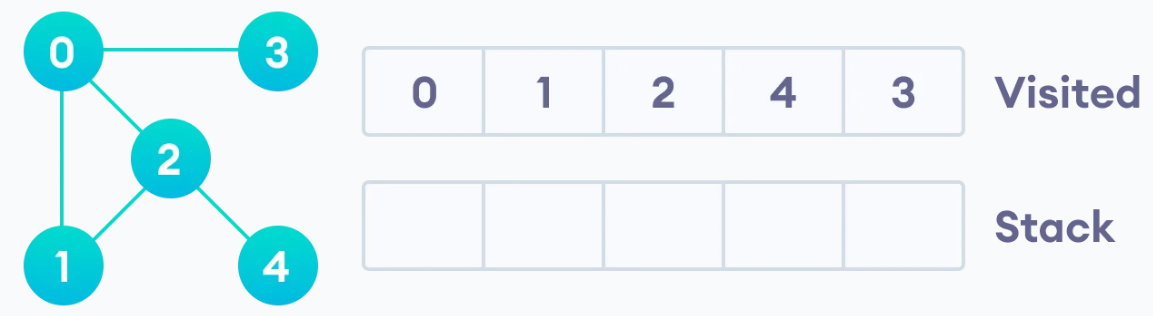
Next, we have 3 options for what node we want to visit next, let’s take node one as the second node then we will put it in the visited list and search for nodes adjacent to node 1 (which are not visited), here we don’t have any nodes adjacent to 1, so we backtrack to 0 and search another node.



Now we take node 2 as our third node and put it in the visited list and search for nodes adjacent to 2, here we found that node 4 is adjacent to node 2, so we add it to our stack. We now visit node 4 and add it to the visited list and then backtrack to 2 and again to 0, as it is a dead end.



Now we take node 3 as our fifth node and put it in the visited list, and search for the adjacent unvisited nodes, here there are not so we backtrack to 0 and now we have explored the whole tree and have the complete depth first traversal of the graph.



## **APPLICATIONS:**

DFS is often used as subroutines in other complex Algorithms. These Problems use DFS:

### 1) We can use the DFS Algorithm to find the cycles

in a Graph. When there are cycles in a graph the DFS back tracks, so we easily keep a count of the number of backtracks in the Algorithm find the number of cycles in the Graph.

2) Path Finding: DFS is used to find the path

Between 2 specific nodes or vertices. We can explore the entire graph and keep recording the distance from the source node to the vertex. That way we can find the path between the desired nodes.

3) Solving maze problems: DFS can be optimized to solve problems like finding a way out of the maze which only has one solution. We can do this by selecting a random path in a maze by considering the starting point as a source node and then exploring it until we hit a dead end, then backtracking to the point where there is another route and then again exploring it. This process will be repeated until we find the exit.

4) Find strongly connected graphs: A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex. We can check if each node is connected to all other nodes by finding the adjacent nodes.

## **LIMITATIONS:**

### The biggest limitation of DFS is that it is not guarenteed to give you a solution as it could get stuck in an infinite branch and could never generate a solution.

### Another limitation is that if the depth of the node to find is smaller then the time complexity is more wrt to BFS.

# **BFS (Breadth First Search) Algorithm:**

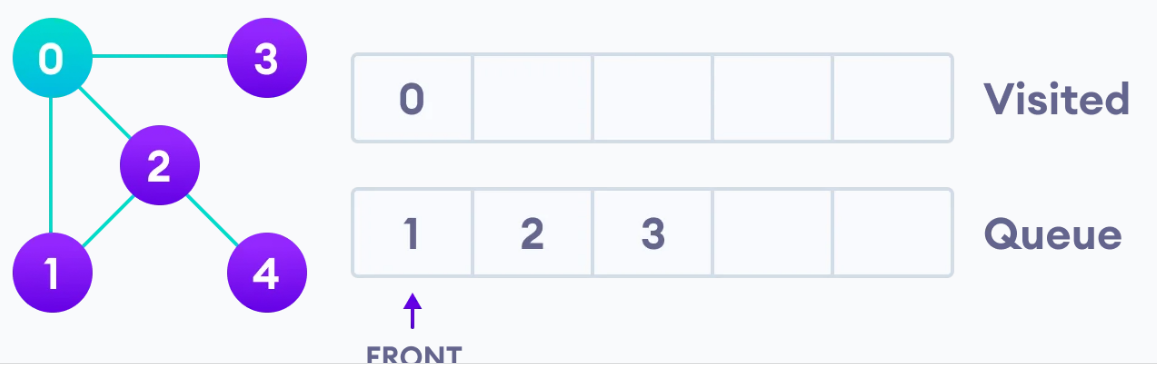
Breadth First search is a graph traversal algorithm that starts traversing the graph from the root node, then moves to the next deep node and searches through all the nodes at that same depth first, then moves to another adjacent node at the next depth and this process continues till the whole graph is explored.

## **EXPLANATION:**

In BFS we divide the nodes of a graph into one of the two categories that is visited or not visited. We must keep on exploring the nodes on the same depth till we have explored all the nodes on the same depth. BFS uses queue data structure to traversal.

Example:

The basic idea is to pick a starting node, here we picked 0, and then start putting all its adjacent nodes, on the same depth, in a queue. Here we have put 1,2,3 in the queue.



We store 0 in the visited list and then visit the next element which is in the front of the queue and search for adjacent nodes of the nodes we are visiting now. Here now we visit node 1 (no adjacent nodes so we move to next item in the queue). Now we visit 2 and since it has an adjacent node (4) at the next depth we store that in the queue. We keep all the nodes which are visited in the visited list.



Similarly, we visit 3 and search for its adjacent node and store 3 in the already visited list. Then we move to the elements which were adjacent to the recently stored elements in the next depth of the graph. We repeat the process until the whole graph is explored. Here we visit 4 and search for its adjacent nodes, since it doesn’t have any our graph traversing ends here.



## **APPLICATIONS:**

BFS is one of the core structures of many algorithms. Some of its applications are:

### 1) We can use the BFS algorithm in GPS navigation as it explores the neighboring edges first before moving to the next level of edges which makes it very perfect to use in GPS navigation.

2) In Ford-Fulkerson algorithm to find maximum flow in a network: The idea is that we must start with source and find the path to the sink. Whenever we find the path from source to sink, we add it to our flow and in the end, we will get our maximum flow.

3) Crawlers in the search engine: web crawlers use the concept of BFS, they start with the main source page and follow all the links of the page and search them, then they follow all the links at the next level and this process continues till the desired result is achieved.

4) Peer to Peer networks: BFS is widely used in P2P file sharing applications. The basic idea is that the source generates a message or a file and then it is propagated to all its neighbours (peers).

## **LIMITATIONS:**

The biggest limitation of the BFS algorithm is that it has a memory constraint as it stores all the nodes before moving to the next depth.

It cannot be effectively used when the search space is long because it consumes a lot of time going deep in the nodes as compared to DFS.

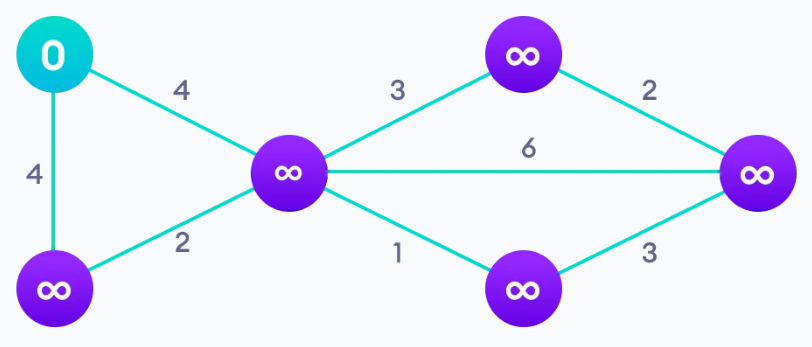
# **DIJKSTRA’s Algorithm:**

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. It was conceived by computer scientist Edger W. Dijkstra in 1956 and published three years later. The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a common variant uses the source node as a starting point and finds the shortest path from that source to all the different nodes in the graph.

## **EXPLANATION:**

In Dijkstra, to find the shortest path between 2 nodes or from source to a node, the algorithm initially marks all the distance between nodes as infinity, as to ensure that there is infinite distance, as those nodes have not been visited yet. We must maintain a distance array in which we have to keep track of the node visited and the shortest length of the path. We also must maintain a node index distance key/value pairs to keep track of the distance between the nodes. This will help to decide which node to visit next by sorting the most promising key value pair and removing the longest path from the pair.

Initially we set the values of all the distances in the distance array to infinite. If after completing the process, some node is still having infinite value in the array, then that node must be unreachable. We start with node 0 assigning the key/value pair (0,0) as the shortest path to reach 0 is 0.

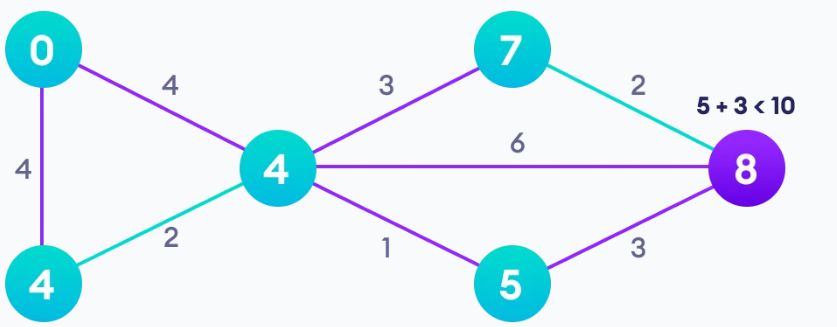


Then to visit the next node 2 we have an option to go from 0 to 1 to 2 or directly 0 to 2. We have to consider the shortest distance here between 0 to 2. From 0-1-2 the distance is 4+2 > 4, which is direct distance from 0-2. So, we must store the value of 4 in the array under the node 2 as the shortest distance.

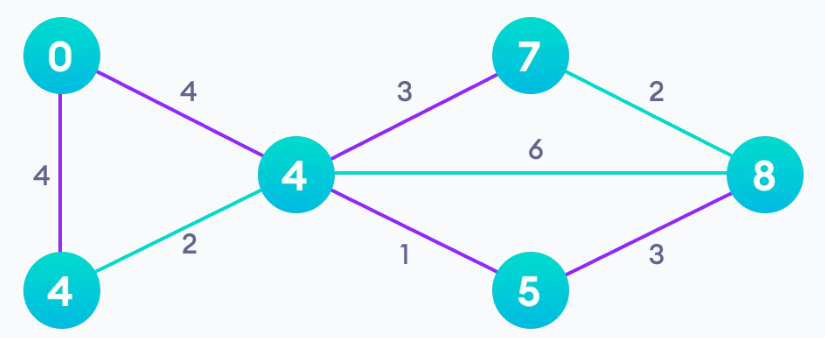
A picture containing text, watch, clock

Description automatically generated

To reach node 7 and node 5 we can directly get the shortest distance from 4 to 7 which will automatically give us the shortest distance between 0-7. Now we update this distance in our array under 7.



Similarly, we can get the shortest paths of all the nodes in the graph and update it in the array.



Now using the information in the array, we can find the shortest path between any 2 nodes in the graph.

## **APPLICATIONS:**

### 1) Digital Mapping services in google maps: We can use Dijktras Algorithm to find the shortest distance between 2 cities or from your location to your destination. There are many routes and paths connecting them but using the algo we can find the most effective path.

2) Social Networking Applications: In many social applications we have seen that the app suggests lists of friends that a user may know. This feature works on the concept of Dijkstra’s Algorithm by finding the shortest path between users through the connections among them.

3) Telephone Network: In a telephone network each line has a bandwidth, which is the highest frequency the line can support. Bandwidth also represents the amount of information which can be transmitted through that line. If we consider all the transmitting towers as nodes and the weight of the paths as bandwidths, we can use Dijkstra’s Algorithm to find the smallest bandwidth and with some modifications even the largest one.

4) Robotic path: Basically, all the algorithms can be used in robotics if we slightly modify them according to our needs. Drones or Robots, which are automated and are designed to deliver some goods from a source to a destination can use this algorithm to figure out the shortest or the fastest path to deliver in the minimum amount of time.

## **LIMITATIONS:**

One big constraint of Dijkstra’s algorithm is that the graph must only contain non-negative edge weights. This constraint is imposed to ensure that once a node has been visited its optimal distance cannot be improved.

Dijkstra’s method does a blind search through the graph by comparing all the paths, so it wastes a lot of time and memory.

# **A\* Algorithm:**

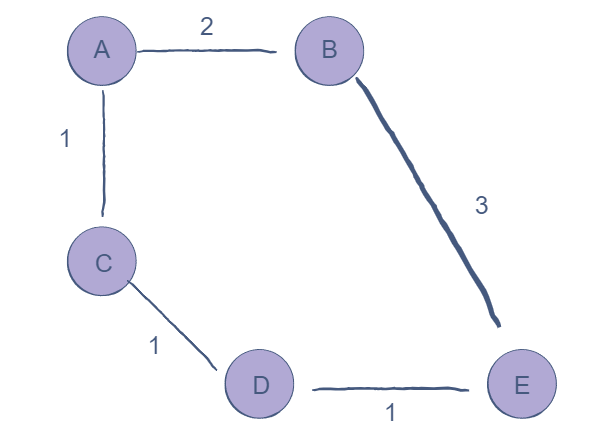
A\* Algorithm is a searching algorithm that searches for the shortest distance between the initial and final state. It is used in applications like graphs, to find the shortest distance between your location and destination.

## **EXPLANATION:**

In A\* we consider a graph with multiple nodes, and we want to traverse through the graph as quickly as possible. We must define a function F in this algo, which contains the value of the shortest path. We also define a function G and H (Heuristic function, which is basically a constant). Here F = G + H.

Example:

Suppose if our initial node is A and our destination is node E. At every node we must calculate the value of F which is the sum of G and H. The minimum distance from node A to node E must be the output.

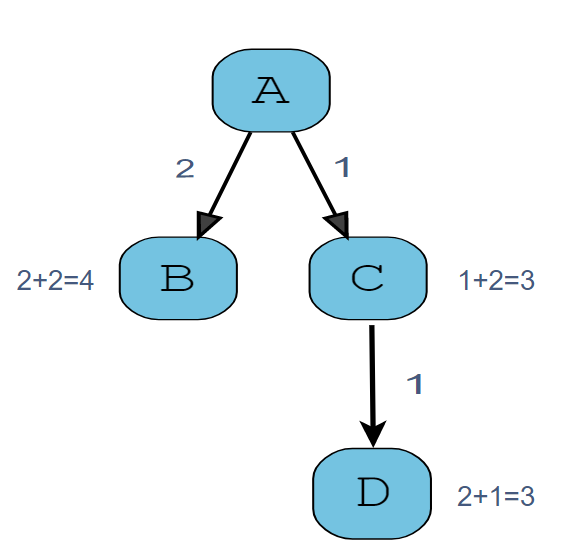


First from source node A, we have 2 options B and C. Let’s assume the value of H is 2. If we take path B, then our value of F for B is 4. If we take path C, then our value of F for C is 3. Now, we can conclude that C is shorter than B, so we consider that path.

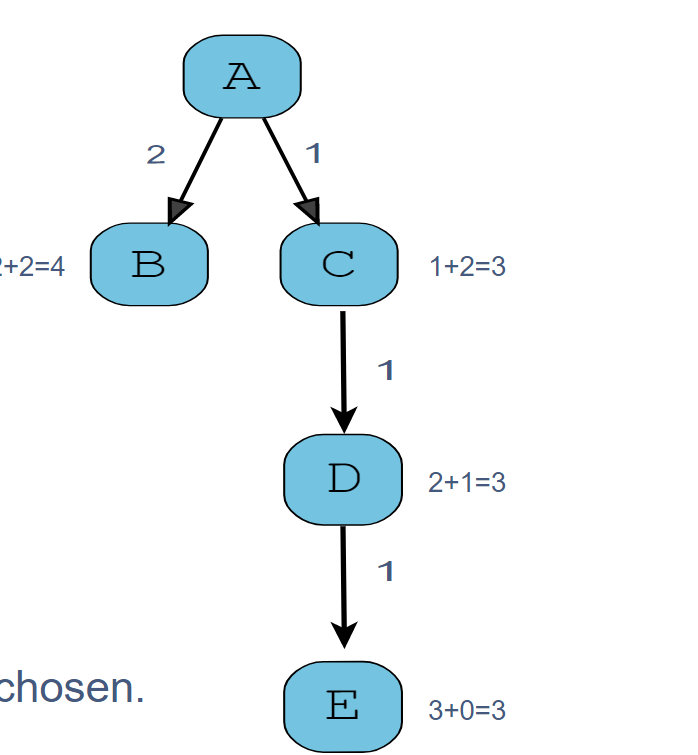
A picture containing text, clock

Description automatically generated

Then we check the next node from C, which is D and there is only one node, so node D is chosen. The value of F for D is 3.



Now after node D, we must check the next node, which is E. We calculate the value of F for E, and our graph traversing ends as we have reached the end. Now, we can calculate the shortest path between the source node and any node using the values of F stored.



## **APPLICATIONS:**

1) It is mostly used in video games and online maps to predict the most efficient and shortest path. It can detect the shortest path efficiently unlike Dijkstra’s algorithm which consumes much more memory.

2) It is used in many Artificial Intelligence applications like the Search Engine, as it follows the shortest path, it can help the Engine to generate the results in a quick and an efficient manner.

3) Bellman Ford Algorithm: This is a graph searching algorithm, which searches the smallest path and uses the concept of A\* algo. Unlike Dijkstra, A\* can be used to calculate the path of the graphs with negative weights.

## **LIMITATIONS:**

The main constraint is that the accuracy of A\* searching is dependent on the Heuristic function.

**Resources:**

<https://en.wikipedia.org/wiki>

<https://www.programiz.com/dsa>

<https://www.geeksforgeeks.org/fundamentals-of-algorithms/>

<https://www.javatpoint.com/bellman-ford-algorithm>

<https://www.techwithtim.net/>

<https://www.freecodecamp.org/news/tag/algorithms/>

<https://www.youtube.com/channel/UC9-y-6csu5WGm29I7JiwpnA>

1. This Is the Heading for a Table
2. This is a table footnote.

You can cite your references in text by including the corresponding number, in square brackets [1]. If you need to cite a specific part of the source, you can include a page number [2, p. 13] or range [3, pp. 41–56].

##### Acknowledgments

“Acknowledgment(s)” is spelled without an “e” after the “g” in American English.

As you can see, the formatting ensures that the text ends in two equal-sized columns rather than only displaying one column on the last page.

This template was adapted from those provided by the IEEE on their own website.

##### References

1. D. V. Lindberg and H. K. H. Lee, “Optimization under constraints by applying an asymmetric entropy measure,” *J. Comput. Graph. Statist.*, vol. 24, no. 2, pp. 379–393, Jun. 2015, doi: 10.1080/10618600.2014.901225.
2. B. Rieder, *Engines of Order: A Mechanology of Algorithmic Techniques*. Amsterdam, Netherlands: Amsterdam Univ. Press, 2020.
3. I. Boglaev, “A numerical method for solving nonlinear integro-differential equations of Fredholm type,” *J. Comput. Math.*, vol. 34, no. 3, pp. 262–284, May 2016, doi: 10.4208/jcm.1512-m2015-0241.

**Make sure to remove all placeholder and explanatory text from the template when you add your own text. This text should not be here in the final version!**