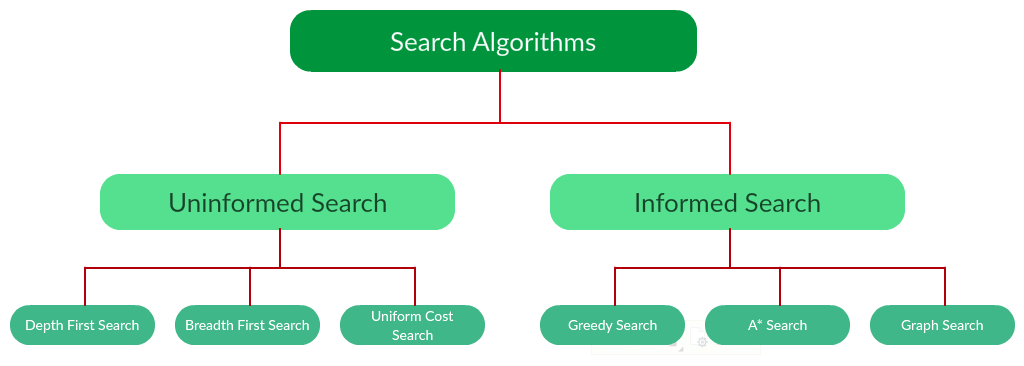
**Search Algorithms in AI**

***Artificial Intelligence***is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

* A search problem consists of:   
  + **A State Space.**Set of all possible states where you can be.
  + **A Start State.**The state from where the search begins.
  + **A Goal State.**A function that looks at the current state returns whether or not it is the goal state.
* The **Solution**to a search problem is a sequence of actions, called the **plan**that transforms the start state to the goal state.
* This plan is achieved through search algorithms.

**Types of search algorithms:**

There are far too many powerful search algorithms out there to fit in a single article. Instead, this article will discuss *six*of the fundamental search algorithms, divided into *two*categories, as shown below.



Note that there is much more to search algorithms than the chart I have provided above. However, this article will mostly stick to the above chart, exploring the algorithms given there.

**Uninformed Search Algorithms:**

The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**. These algorithms can only generate the successors and differentiate between the goal state and non goal state.   
  
The following uninformed search algorithms are discussed in this section.

1. Depth First Search
2. Breadth First Search
3. Uniform Cost Search

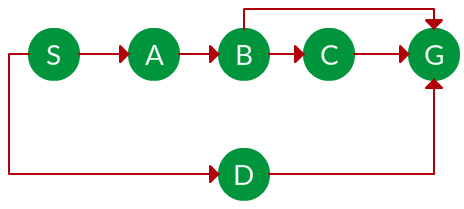
Each of these algorithms will have:

* A problem **graph,**containing the start node S and the goal node G.
* A **strategy,**describing the manner in which the graph will be traversed to get to G.
* A **fringe,**which is a data structure used to store all the possible states (nodes) that you can go from the current states.
* A **tree,**that results while traversing to the goal node.
* A solution **plan,**which the sequence of nodes from S to G.

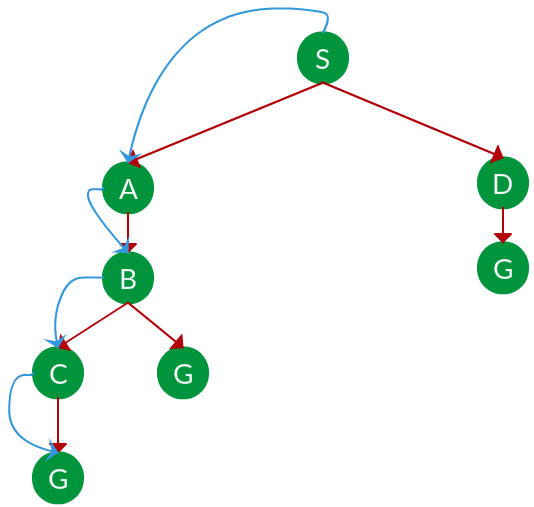
[Depth First Search](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/)**:**

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. It uses last in- first-out strategy and hence it is implemented using a stack.  
  
**Example:**

***Question.*** *Which solution would DFS find to move from node S to node G if run on the graph below?*



**Solution.** The equivalent search tree for the above graph is as follows. As DFS traverses the tree “deepest node first”, it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.

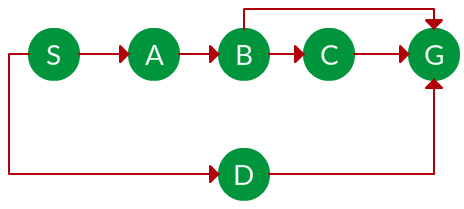


**Path:**   S -> A -> B -> C -> G -> G -> D -> G

***Time complexity:*** *Equivalent to the number of nodes traversed in DFS.****Space complexity:*** *Equivalent to how large can the fringe get.****Completeness:*** *DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.****Optimality:*** *DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.*

[Breadth First Search](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/)**:**

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph) and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It is implemented using a queue.  
  
**Example:**  
**Question.**Which solution would BFS find to move from node S to node G if run on the graph below?



**Solution.** The equivalent search tree for the above graph is as follows. As BFS traverses the tree “shallowest node first”, it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.

*= the depth of the shallowest solution.   
= number of nodes in level .****Time complexity:*** *Equivalent to the number of nodes traversed in BFS until the shallowest solution.****Space complexity:*** *Equivalent to how large can the fringe get.****Completeness:*** *BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.****Optimality:*** *BFS is optimal as long as the costs of all edges are equal.*