In recent years, Artificial Intelligence (AI) has had a tremendous impact on the technologies of robotics, pathfinding, and problem-solving. Moreover, Humans are dependent on these technologies to find solutions and perform tasks. However, what makes these actions possible? The answer is simple: Search Algorithms. Searching is an integral part of AI, and AI machines use it, along with various search algorithms techniques to systematically explore alternatives and find the most suitable path to reach the goal.

That's not all, these search algorithms are divided into two major parts Uninformed and Informed Search Algorithms, which make this search more streamlined and accurate. It is one of these types-the Uninformed Search Algorithm -that we are going to discuss in detail today, along with various uninformed search strategies.

WHAT IS UNINFORMED SEARCH?

Based on the available information about the problem, the Search Algorithm is first categorized into Uninformed Search Algorithm, which generates the search tree without any domain-specific knowledge or additional information about the states beyond the information provided in the problem definitions. This class-purpose search algorithm, which is also known as Blind Search or Brute-Force Search, operates in a brute-force way and examines each root node of the search tree until it achieves the goal state.

It works effectively with small possible states and can be applied to a variety of search problems, as it does not take into consideration the target problem and is only registers it once it achieves the final goal. Moreover, the uninformed search includes six (6) search strategies, which are obtained on the basis of the selection path implemented in the frontier.

But, before we move on to answer “what is uninformed strategy?”, we need to understand the requirements of the Uninformed Search Algorithm and the various components covered by its search strategy.

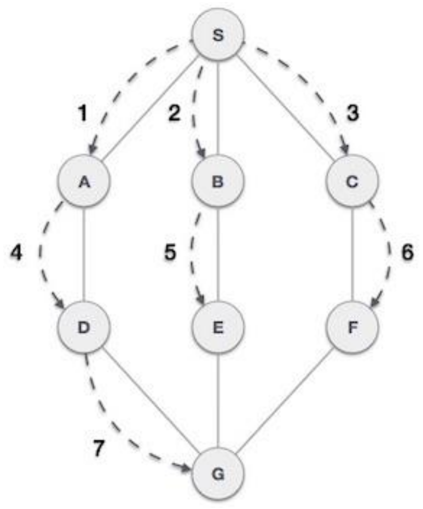
WHAT ARE THE VARIOUS UNINFORMED SEARCH TECHNIQUES?
Uniformed Search Techniques are various goal-achieving strategies that don’t take into account the location of the goal, as they are not focused on the path of reaching the goal until they find it and report success. These strategies are of sic (6) types:

1.) BREADTH-FIRST SEARCH:
The most common search strategy, breadth-first search, starts from the root node, explores the neighboring nodes first and moves towards the next level. Implemented using the First-in First-out (FIFO) queue data structure, this is an example of a general-graph search algorithm that provides the shortest path to the solution.

Moreover, it is useful in the following conditions:

When the problem is small enough and space complexity is not a problem.
you want a solution containing the fewest arcs.

EXAMPLE:
Consider the following graph, where the BFS algorithm traverses from A to B to E to F first then to C and G lastly to D to reach the goal state.



In this graph, the algorithm uses a queue to remember to get the next vertex to start a search as well as when a dead end occurs in any iteration. Without this queue, it won't be able to identify and mark visited vertices and will process them again, which can then become a non-terminating process.

In short, breadth-first search or BFS expands the shallowest node, with the lowest depth, first.

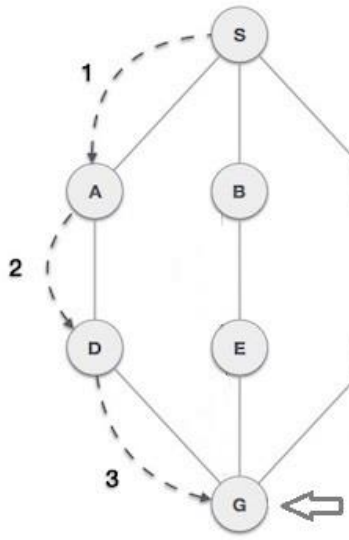
3.) DEPTH-LIMITED SEARCH:

The third search algorithm, the depth-limited search (DLS) is similar to the depth-first search, with a minor difference that it has a predetermined limit. DLS is used to treat the depth limit node, as it does not have any successor nodes. DLS is best suited in cases where there is prior knowledge of the problem, which at times is difficult to achieve.

In short, this memory-efficient search algorithm is used to solve the drawbacks of the infinite path in the Depth-first search.

EXAMPLE:

We can consider the example considered for Depth-First Search here too, as in DLS the only difference is that it has a set limit. Here, the algorithm will traverse in a depth-ward motion and uses a stack to remember the next vertex to start a search. But it will end the search when it reaches the limit, which in this case is G. The path used to traverse the graph by DLS is from S to A to D to G, which is the set limit.



4.) ITERATIVE DEEPENING DEPTH-FIRST SEARCH:

A combination of DFS and BFS algorithms, Iterative Deepening Depth-First Search identifies the best depth limit, by gradually increasing the limit until the goal is achieved. Unlike the other two search algorithms, IDDFS is complete and is used to perform iterations of depth-limited searches while increasing the expected runtime. Moreover, if a solution exists, IDDFS will find a solution path with the fewest arcs.

This searching strategy is extremely beneficial, as it combines the benefits of both Breadth-First Search and Depth-First Search. The only drawback associated with this method is that it performs the search and work of previous stages.

5.) BIDIRECTIONAL SEARCH:

Extremely different from other search algorithms, the Bidirectional search executes two concurrent searches to find the final goal node, which is known as forward-search (Start State) and backward-search (Goal State). Bidirectional Search uses two small subgraphs, instead of one graph, wherein one starts the search from an initial vertex and other from goal vertex. It is only when these two vertices intersect that the search reaches its end. This algorithm has less time complexity and does not have high memory requirements.

It can be used in the following instances:

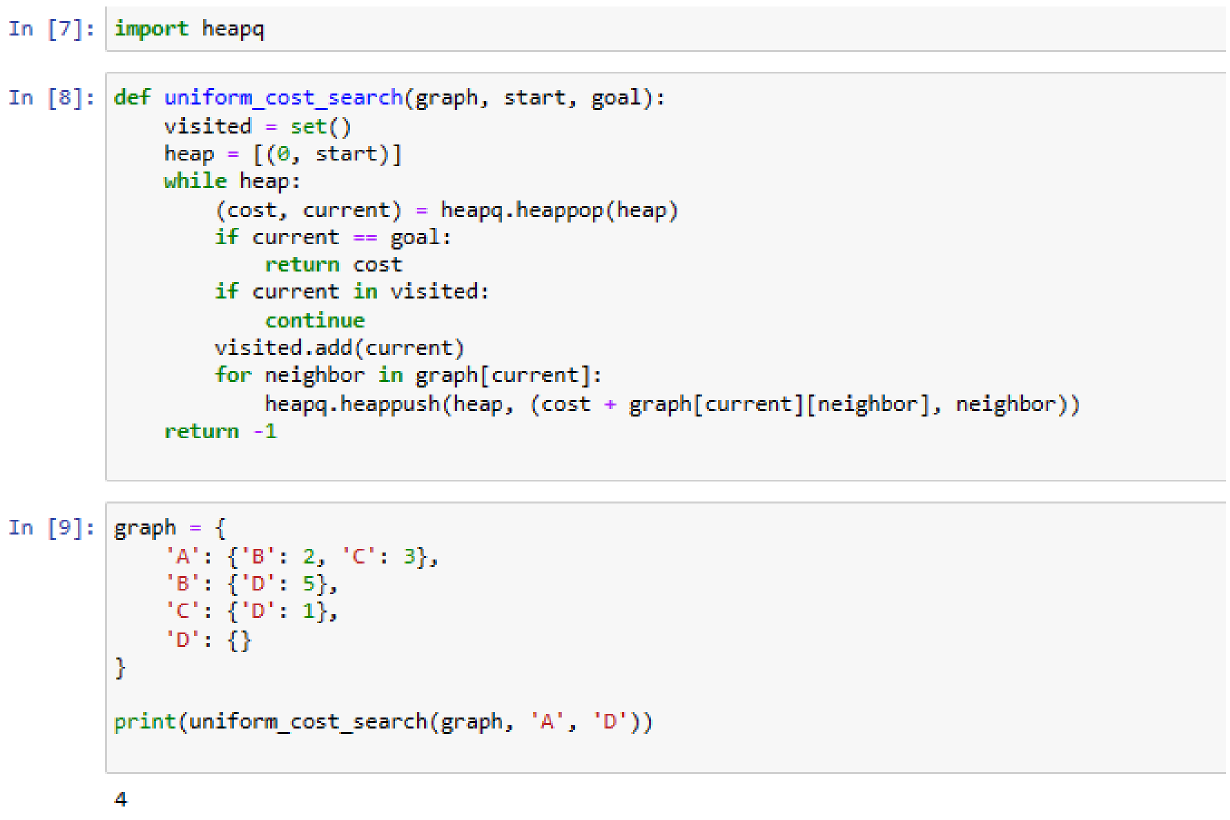
When both initial & goal states are unique and completely defined.

The branching factor is similar in both directions.

6.)UNIFORM COST SEARCH

Implemented through the priority queue, Uniform Cost Search is the best algorithm for a search problem, as it does not involve the use of heuristics. It gives maximum priority to the lowest cumulative cost and is equivalent to the Breadth-First Search algorithm in cases where the path cost of all edges is the same.

As this algorithm chooses the path with the least cost at every state, from start to goal, it can be used to solve any graph/tree that demands optimal cost.l



```
In [7]: import heapq
```

```
In [8]: def uniform_cost_search(graph, start, goal):
            visited = set()
            heap = [(0, start)]
            while heap:
                (cost, current) = heapq.heappop(heap)
                if current == goal:
                    return cost
                if current in visited:
                    continue
                visited.add(current)
                for neighbor in graph[current]:
                    heapq.heappush(heap, (cost + graph[current][neighbor], neighbor))
            return -1
```

```
In [9]: graph = {
            'A': {'B': 2, 'C': 3},
            'B': {'D': 5},
            'C': {'D': 1},
            'D': {}
        }

        print(uniform_cost_search(graph, 'A', 'D'))

        4
```

NAME : KANISHKAN PERUMAL

REG :122012173011

COURSE :BSC(AI)

SUBJECT:FOUNDATION OF AI

TOPIC :UNDER UNUNIFORM SEARCH